# El gran libro de Android

Jesús Tomás Gironés

2ª Edición

Alfaomega

marcombo
ediciones técnicas

# El gran libro de Android

Jesús Tomás Gironés

# El gran libro de Android

Jesús Tomás Gironés

Alfaomega

marcombo
ediciones técnicas

Datos catalográficos

Tomás, Jesús
El gran libro de Android
Segunda Edición

Alfaomega Grupo Editor, S.A. de C.V., México

ISBN: 978-607-707-506-6

Formato: 17 x 23 cm Páginas: 404

**El gran libro de Android**
Jesús Tomás Gironés
ISBN: 978-84-267-1832-7, edición en español publicada por MARCOMBO, S.A., Barcelona, España


Segunda edición: Alfaomega Grupo Editor, México, julio 2012


Pitágoras 1139, Col. Del Valle, 03100, México D.F.

Miembro de la Cámara Nacional de la Industria Editorial Mexicana
Registro No. 2317

Pág. Web: **http://www.alfaomega.com.mx**
E-mail: **atencionalcliente@alfaomega.com.mx**

**ISBN: 978-607-707-506-6**



**Impreso en México. Printed in Mexico.**

**Empresas del grupo**:

**México:** Alfaomega Grupo Editor, S.A. de C.V. – Pitágoras 1139, Col. Del Valle, México, D.F. – C.P. 03100.
Tel.: (52-55) 5575-5022 – Fax: (52-55) 5575-2420 / 2490. Sin costo: 01-800-020-4396
E-mail: atencionalcliente@alfaomega.com.mx

**Colombia:** Alfaomega Colombiana S.A. – Carrera 15 No. 64 A 29, Bogotá, Colombia,
Tel.: (57-1) 2100122 – Fax: (57-1) 6068648 – E-mail: cliente@alfaomega.com.mx

**Chile:** Alfaomega Grupo Editor, S.A. – Dr. La Sierra 1437, Providencia, Santiago, Chile
Tel.: (56-2) 235-4248 – Fax: (56-2) 235-5786 – E-mail: agechile@alfaomega.cl

**Argentina:** Alfaomega Grupo Editor Argentino, S.A. – Paraguay 1307 P.B. Of. 11, C.P. 1057, Buenos Aires, Argentina, – Tel./Fax: (54-11) 4811-0887 y 4811 7183 – E-mail: ventas@alfaomegaeditor.com.ar

*Para Bea, con amor y gratitud*

# Índice general

# Lista de acrónimos

| | |
|---|---|
| AIDL | *Android Interface Definition Language* |
| API | *Application Programming Interface* |
| AVD | *Android Virtual Device* |
| DOM | *Modelo de Objetos del Documento* |
| DTD | *Document Type Definition* |
| FTP | *File Transfer Protocol* |
| GPS | *Global Positioning System* |
| HTTP | *Hypertext Transfer Protocol* |
| IDE | *Integrated Development Environment* |
| JDK | *Java Development Kit* |
| JRE | *Java Runtime Environment* |
| JSON | *JavaScript Object Notation* |
| JVM | *Java Virtual Machine* |
| MIME | *Multipurpose Internet Mail Extensions* |
| NDK | *Native Development Kit* |
| OpenGL | *Open Graphic Library* |
| PCM | *Pulse-Code Modulation* |
| RAM | *Random Access Memory* |
| REST | *REpresentational State Transfer* |
| RPC | *Remote Procedure Calls* |
| SAX | *Simple API for XML* |
| SDK | *Software Developers Kit* |
| SMS | *Short Message Service* |
| SOA | *Service-Oriented Architecture* |
| SOAP | *Simple Object Access Protocol* |
| SQL | *Structured Query Language* |
| TCP | *Transmission Control Protocol* |
| UI | *User Interface* |
| URL | *Universal Resource Locator* |

| | |
|---|---|
| URI | *Uniform Resource Identifier* |
| USB | *Universal Serial Bus* |
| W3C | *World Wide Web Consortium* |
| WSDL | *Web Services Description Language* |
| WWW | *World Wide Web* |
| XML | *Extensible Markup Language* |

# ¿Cómo leer este libro?

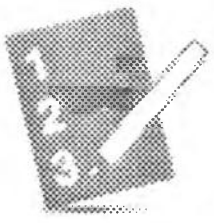
Este libro quiere ser una guía para aquellos lectores que pretendan introducirse en la programación de Android. Se ha estructurado en 11 capítulos que abordan aspectos especificos del desarrollo de aplicaciones. Resulta conveniente realizar una lectura secuencial de estos capítulos, dado que muchos de los conceptos que se abordan serán compredidos mejor si hemos leído los capítulos anteriores. Además, a lo largo del libro se desarrolla una aplicación de ejemplo, el mítico juego Asteroides. Para que muchos de los ejercicios funcionen correctamente resulta imprescindible realizar los anteriores.

El libro que tienes entre las manos no ha sido concebido solo para ser leído. Es más bien una guía estructurada que te irá proponiendo una serie de ejercicios, actividades, videos explicativos, test de auto evaluación,... Todo este material y muchos más recursos adicionales están disponibles en la Web www.androidcurso.com. En ella se publicarán las novedades, erratas e información complementaria relativas a este libro. Por lo tanto, resulta imprescindible para sacarle partido a este libro un ordenador con el SDK de Android instalado para hacer los ejercicios y acceso a Internet para el material en línea.

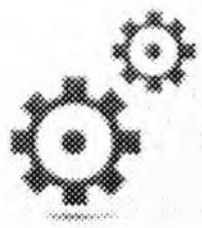
A lo largo del libro se utilizan los siguientes iconos para indicar los tipos de actividades:

**Objetivos:** Antes de empezar cada capítulo lee con detenimiento la introducción y los objetivos.

Polli**[Media]**: Más de 50 videos grabados por el autor del libro donde se exponen de forma didáctica los aspectos clave del sistema Android. Se utiliza una moderna herramienta desarrollada en la Universidad Politécnica de Valencia que te permitirá ver simultáneamente las presentaciones y al profesor mientras se desarrollan los conceptos de cada capítulo.



**Ejercicio paso a paso:** La mejor forma de aprender es haciendo. No tendrás más que ir siguiendo los pasos uno tras otro para descubrir como se resuelve el ejercicio propuesto. Para que no se ta haga pesado teclear todo el código, te proponemos que lo copies y peges desde la página Web del curso.



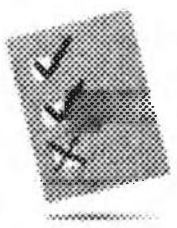
**Práctica:** Este será el momento de que tomes la iniciativa y trates de resolver el problema que se propone. Recuerda que para aprender hay que practicar.

**Solución:** Te será de ayuda si tienes problemas al resolver una práctica, o simplemente quieres comparar tu solución con otra diferente.

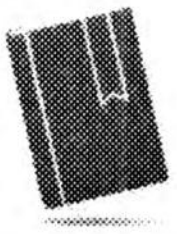
**Nota sobre Java:** Si no dominas el lenguaje de programación Java no tendrás problema en seguir el libro. Cada vez que aparezca algún concepto complejo sobre Java, trataremos de aclararlo.

**Recursos adicionales:** Te proporcionamos la información clave que te ayudará en el desarrollo de tus aplicaciones.

**Enlaces de interés:** Internet te será de gran ayuda para completar la información necesaria para programar en Android. Te proponemos las páginas más interesantes de cada apartado.



**Preguntas de repaso y reflexión:** ¿Has comprendido correctamente los aspectos claves? Sal de dudas haciendo los test de autoevaluación.



**Referencias rápidas:** Utiliza los anexos para localizar rápidamente esa palabra clave o clase que no recuerdas.

De forma adicional, en la Web www.androidcurso.com encontrarás:

- **Tutoriales sobre Java:** ¿Sabes lo que es la herencia, el polimorfismo o la sobrecarga en Java? Si no dominas el lenguaje de programación Java te recomendamos realizar alguno de los tutoriales propuestos.
- **Código abierto de proyectos Android:** Muchos alumnos que han realizado un curso basado en este libro han tenido la generosidad de compartir sus proyectos con todos nosotros. Te recomendamos que consultes la lista de proyectos disponibles de código abierto, puedes aprender mucho estudiando su código. Cuando termines de leer este libro, también tu podrás hacer un proyecto como los que se muestran.
- **Material adicional sobre Android:** Encontrarás además nuevos tutoriales, Polimedias, referencias,... no incluidos en el Libro.
- **Foros:** Puedes comunicarte con otras personas y consultar cualquier duda que se te plante en un apartado o ejercicios del libro.
- **Cursos online:** Si te interesa ampliar tu formación puedes matricularte en cursos sobre Android impartidos por la Universidad Politécnica de Valencia.

# CAPÍTULO 1.
# Visión general y entorno de desarrollo

La telefonía móvil está cambiando la sociedad actual de una forma tan significativa como lo ha hecho Internet. Esta revolución no ha hecho más que empezar, los nuevos terminales ofrecen unas capacidades similares a las de un ordenador personal, lo que permite que puedan ser utilizados para leer nuestro correo o navegar por Internet. Pero a diferencia de un ordenador, un teléfono móvil siempre está en el bolsillo del usuario. Esto permite un nuevo abanico de aplicaciones mucho más cercanas al usuario. De hecho, muchos autores coinciden en que el nuevo ordenador personal del siglo veintiuno será un terminal móvil.

El lanzamiento de Android como nueva plataforma para el desarrollo de aplicaciones móviles ha causado una gran expectación y está teniendo una importante aceptación, tanto por los usuarios como por la industria. En la actualidad se está convirtiendo en la alternativa estándar frente a otras plataformas como iPhone, Windows Phone o BlackBerry.

A lo largo de este capítulo veremos las características de Android que lo hacen diferente de sus competidores. Se explicará también cómo instalar y trabajar con el entorno de desarrollo (Eclipse + Android SDK).

**Objetivos:**

- Conocer las características de Android, destacando los aspectos que lo hacen diferente de sus competidores.
- Estudiar la arquitectura interna de Android.
- Aprender a instalar y trabajar con el entorno de desarrollo (Eclipse + Android SDK).
- Enumerar las principales versiones de Android y aprender a elegir la más idónea para desarrollar nuestras aplicaciones.

- Crear una primera aplicación y estudiar la estructura de un proyecto en Android.
- Conocer donde podemos conseguir documentación sobre Android.
- Aprender a utilizar las herramientas disponibles para detectar errores en el código.

## 1.1. ¿Qué hace que Android sea especial?

Como hemos comentado, existen muchas plataformas para móviles (iPhone, Symbian, Windows Phone, BlackBerry, Palm, Java Mobile Edition, Linux Mobile (LiMo), etc.); sin embargo Android presenta una serie de características que lo hacen diferente. Es el primero que combina, en una misma solución, las siguientes cualidades:

- **Plataforma realmente abierta.** Es una plataforma de desarrollo libre basada en Linux y de código abierto. Una de sus grandes ventajas es que se puede usar y "customizar" el sistema sin pagar *royalties*.
- **Portabilidad asegurada.** Las aplicaciones finales son desarrolladas en Java, lo que nos asegura que podrán ser ejecutadas en una gran variedad de dispositivos, tanto presentes como futuros. Esto se consigue gracias al concepto de máquina virtual.
- **Arquitectura basada en componentes inspirados en Internet.** Por ejemplo, el diseño de la interfaz de usuario se hace en XML, lo que permite que una misma aplicación se ejecute en un móvil de pantalla reducida o en un *netbook*.
- **Filosofía de dispositivo siempre conectado a Internet.**
- **Gran cantidad de servicios incorporados.** Por ejemplo, localización basada tanto en GPS como en redes, bases de datos con SQL, reconocimiento y síntesis de voz, navegador, multimedia, etc.
- **Aceptable nivel de seguridad.** Los programas se encuentran aislados unos de otros gracias al concepto de ejecución dentro de una caja que hereda de Linux. Además, cada aplicación dispone de una serie de permisos que limitan su rango de actuación (servicios de localización, acceso a Internet, etc.).
- **Optimizado para baja potencia y poca memoria.** Por ejemplo, Android utiliza la Máquina Virtual Dalvik. Se trata de una implementación de Google de la máquina virtual de Java optimizada para dispositivos móviles.
- **Alta calidad de gráficos y sonido.** Gráficos vectoriales suavizados, animaciones inspiradas en Flash, gráficos en 3 dimensiones basados en OpenGL. Incorpora los *codecs* estándar más comunes de audio y vídeo, incluyendo H.264 (AVC), MP3, AAC, etc.

Como hemos visto, Android combina características muy interesantes. No obstante, la pregunta del millón es, ¿se convertirá Android en el estándar de sistema operativo (S.O.) para móviles? Para contestar a esta pregunta habrá que esperar un tiempo para ver la evolución del iPhone de Apple y cuál es la respuesta de Windows con el lanzamiento de su nuevo S.O. para móviles.

En conclusión, Android nos ofrece una forma sencilla y novedosa de implementar potentes aplicaciones para móviles. A lo largo de este texto trataremos de mostrar de la forma más sencilla para conseguirlo.

## 1.2. Los orígenes

Google adquiere Android Inc. en el año 2005. Se trataba de una pequeña compañía que acababa de ser creada, orientada a la producción de aplicaciones para terminales móviles. Ese mismo año empiezan a trabajar en la creación de una máquina virtual Java optimizada para móviles (Dalvik VM).

En el año 2007 se crea el consorcio Handset Alliance[1] con el objetivo de desarrollar estándares abiertos para móviles. Está formado por Google, Intel, Texas Instruments, Motorola, T-Mobile, Samsung, Ericsson, Toshiba, Vodafone, NTT DoCoMo, Sprint Nextel y otros. Una pieza clave de los objetivos de esta alianza es promover el diseño y difusión de la plataforma Android. Sus miembros se han comprometido a publicar una parte importante de su propiedad intelectual como código abierto bajo la licencia Apache v2.0.

En noviembre del 2007 se lanza una primera versión del Android SDK. Al año siguiente aparece el primer móvil con Android (T-Mobile G1). En octubre, Google libera el código fuente de Android, principalmente bajo licencia de código abierto Apache (licencia GPL v2 para el núcleo). Ese mismo mes, se abre Android Market para la descarga de aplicaciones. En abril del 2009, Google lanza la versión 1.5 del SDK que incorpora nuevas característica como el teclado en pantalla. A finales del 2009 se lanza la versión 2.0 y durante el 2010 las versiones 2.1, 2.2 y 2.3.

Durante el año 2010, Android se consolida como uno de los sistemas operativos para móviles más utilizados, con resultados cercanos al iPhone, e incluso superando al sistema de Apple en EE.UU.

En el 2011 se lanzan la versión 3.x específica para tabletas y 4.x tanto para móviles como para tabletas. Durante este año, Android se consolida como la plataforma para móviles más importante, alcanzando una cuota de mercado superior al 50%. En 2012, Google cambia su estrategia en su tienda de descargas online, remplazando Android Market por Google Play Store, donde en un solo portal unifica la descarga de aplicaciones como de contenidos.

---

[1] http://www.openhandsetalliance.com

Polii[Media]: *Introducción a la plataforma para móviles Android.*

**Preguntas de repaso y reflexión:** *Características y orígenes de Android.*

## 1.3. Comparativa con otras plataformas

En este apartado vamos a describir las características de las principales plataformas móviles disponibles en la actualidad. Dado la gran cantidad de datos que se indican, hemos utilizado una tabla para representar la información. De esta forma resulta más sencillo comparar las distintas plataformas.

Polii[Media]: *Comparativa de las principales plataformas para móviles.*


| | Apple iOS 5.1 | Android 4.0 | Windows Phone 7 | BlackBerry OS 7 | Symbian 9.5 |
|---|---|---|---|---|---|
| **Compañía** | Apple | Open Handset Alliance | Windows | RIM | Symbian Foundation |
| **Núcleo del SO** | Mac OS X | Linux | Windows CE | Mobile OS | Mobile OS |
| **Familia CPU soportada** | ARM | ARM, MIPS, Power, x86 | ARM | ARM | ARM |
| **Lenguaje de programación** | Objective-C, C++ | Java, C++ | C#, muchos | Java | C++ |

| | Apple iOS 5.1 | Android 4.0 | Windows Phone 7 | BlackBerry OS 7 | Symbian 9.5 |
|---|---|---|---|---|---|
| **Licencia de *software*** | Propietaria | *Software* libre y abierto | Propietaria | Propietaria | *Software* libre |
| **Año de lanzamiento** | 2007 | 2008 | 2010 | 2003 | 1997 |
| **Motor del navegador web** | WebKit | WebKit | Pocket Internet Explorer | WebKit | WebKit |
| **Soporte Flash** | No | Sí | No | Si | Sí |
| **HTML5** | Sí | Sí | Parcial | Sí | No |
| **Tienda de aplicaciones** | App Store | Google Play | Windows Marketplace | BlackBerry App World | Ovi Store |
| **Número de aplicaciones** | 400.000 | 300.000 | 50.000 | 30.000 | 50.000 |
| **Coste publicar** | $99 al año | $25 una vez | $99 al año | Sin coste | $1 una vez |
| **Plataforma de desarrollo** | Mac | Windows, Mac, Linux | Windows | Windows, Mac | Windows, Mac, Linux |
| **Interfaz personalizable** | No | Sí | Sí | Sí | Sí |
| **Actualizaciones automáticas del S.O.** | Sí | Depende del fabricante | Depende del fabricante | Sí | Sí |
| **Soporte memoria externa** | No | Sí | No | Sí | Sí |
| **Fabricante único** | Sí | No | No | Sí | No |
| **Variedad de dispositivos** | Modelo único | Muy alta | Baja | Baja | Muy alta |
| **Tipo de pantalla** | Capacitativa | Capacitiva /resistiva | Capacitativa | Capacitativa /resistiva | Capacitiva /resistiva |
| **Aplicaciones nativas** | Sí | Sí | No | No | Sí |

***Tabla 1:*** *Comparativa de las principales plataformas móviles.*

Otro aspecto fundamental a la hora de comparar las plataformas móviles es su cuota de mercado. En la siguiente gráfica podemos ver un estudio realizado por la empresa Gratner Group, donde se muestra la evolución del mercado de los sistemas operativos para móviles según el número de terminales vendidos. Podemos destacar: el importante descenso de ventas de la plataforma Symbian de Nokia; el declive continuo de BlackBerry; como la plataforma de Windows parece que no despega; como Apple tiene afianzada una cuota de mercado y ha

experimentado un importante repunte a finales del 2011. Finalmente, destacamos el espectacular ascenso de la plataforma Android, que le ha permitido alcanzar en dos años una cuota de mercado superior al 50%.

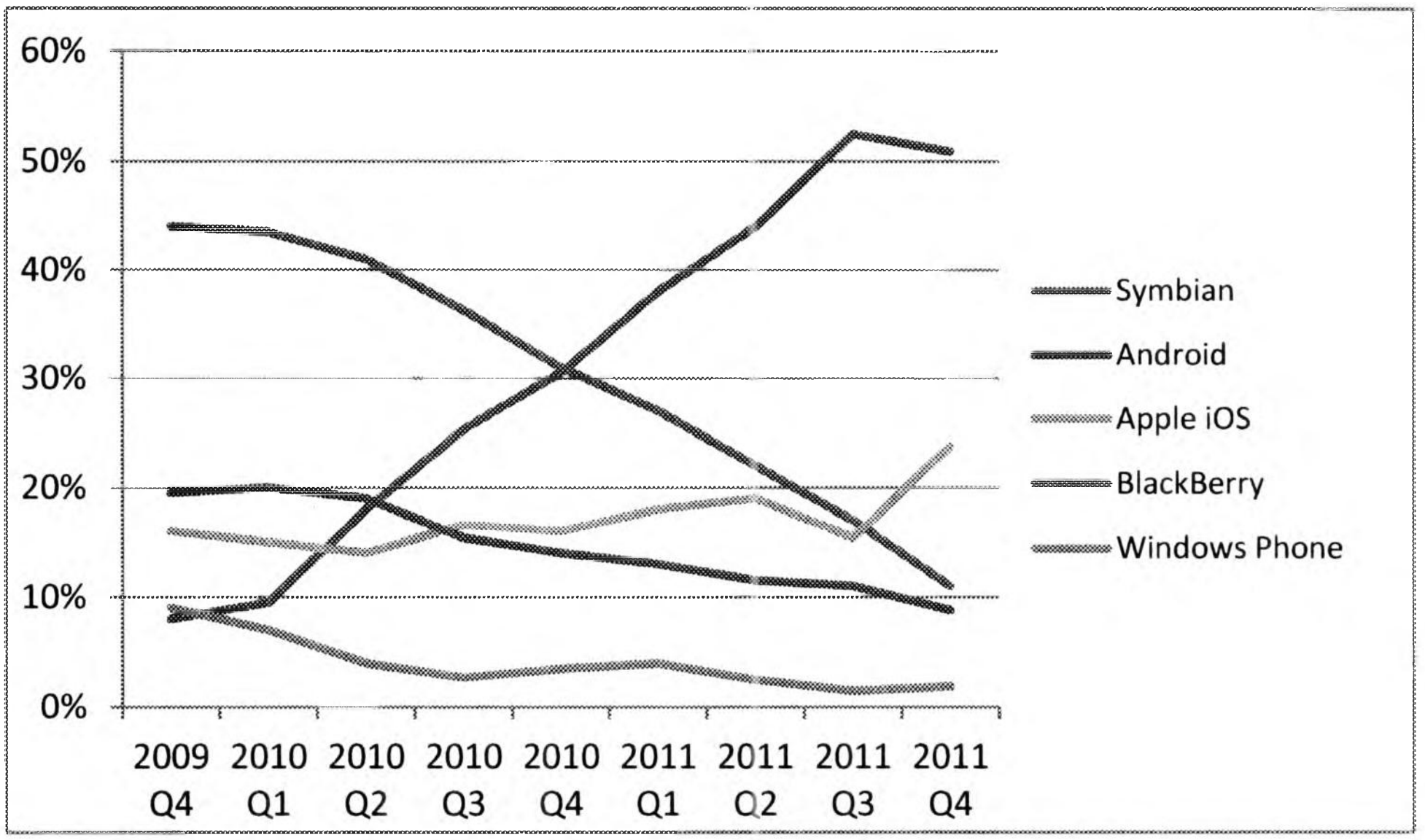
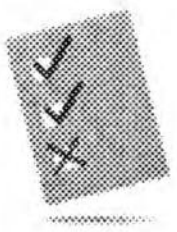



***Figura 1:*** *Cuota de mercado de los sistemas operativos para móviles hasta el 2011 en el mundo (fuente: Gratner Group).*

Se han realizado otros tipos de estudios que miden la actividad de los usuarios en Internet. En estos casos se comprueba como los usuarios de Android e iPhone son los más activos, mientras que los usuarios con otras plataformas, como Symbian, utilizan sus terminales de forma más convencional.



**Preguntas de repaso y reflexión:** *Plataformas para móviles.*

## 1.4. Arquitectura de Android

El siguiente gráfico muestra la arquitectura de Android. Como se puede ver, está formada por cuatro capas. Una de las características más importantes es que todas las capas están basadas en *software* libre.

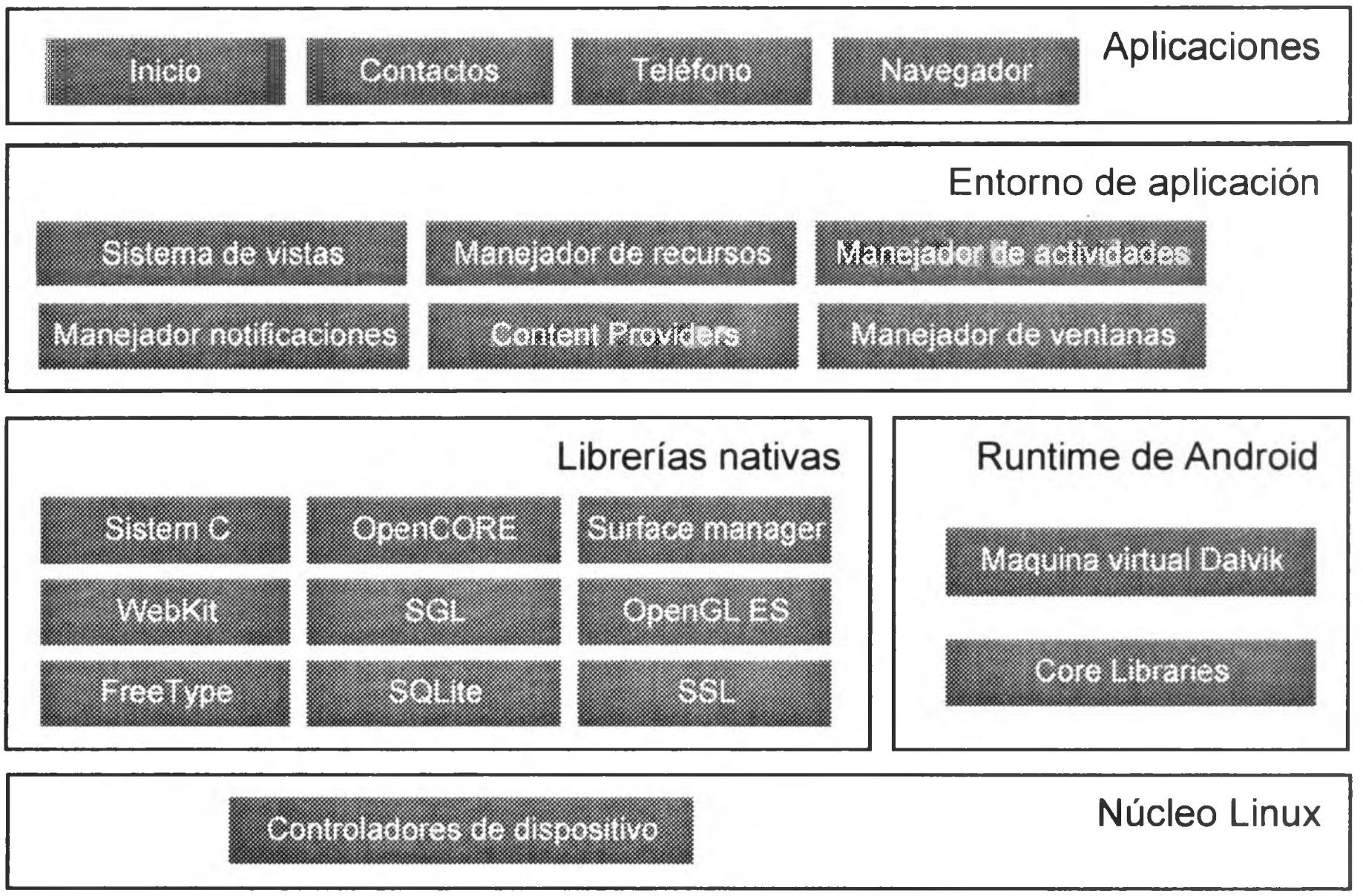



***Figura 2:*** *Arquitectura de Android.*

## 1.4.1. El núcleo Linux

El núcleo de Android está formado por el sistema operativo Linux, versión 2.6. Esta capa proporciona servicios como la seguridad, el manejo de la memoria, el multiproceso, la pila de protocolos y el soporte de *drivers* para dispositivos.

Esta capa del modelo actúa como capa de abstracción entre el hardware y el resto de la pila. Por lo tanto, es la única que es dependiente del *hardware*.

## 1.4.2. *Runtime* de Android

Está basado en el concepto de máquina virtual utilizado en Java. Dado las limitaciones de los dispositivos donde ha de ejecutarse Android (poca memoria y procesador limitado) no fue posible utilizar una máquina virtual Java estándar. Google tomó la decisión de crear una nueva, la máquina virtual Dalvik, que respondiera mejor a estas limitaciones.

Algunas características de la máquina virtual Dalvik que facilitan esta optimización de recursos son: que ejecuta ficheros Dalvik ejecutables (.dex) (formato optimizado para ahorrar memoria). Además, está basada en registros. Cada aplicación corre en su propio proceso Linux con su propia instancia de la máquina virtual Dalvik. Delega al kernel de Linux algunas funciones como *threading* y el manejo de la memoria a bajo nivel.

También se incluye en el *Runtine de Android* el "core libraries" con la mayoría de las librerías disponibles en el lenguaje Java.

### 1.4.3. Librerías nativas

Incluye un conjunto de librerías en C/C++ usadas en varios componentes de Android. Están compiladas en el código nativo del procesador. Muchas de las librerías utilizan proyectos de código abierto. Algunas de estas librerías son:

- **System C library**: una derivación de la librería BSD de C estándar (libc), adaptada para dispositivos embebidos basados en Linux.
- **Media Framework**: librería basada en PacketVideo's OpenCORE; soporta *codecs* de reproducción y grabación de multitud de formatos de audio, vídeo e imágenes MPEG4, H.264, MP3, AAC, AMR, JPG y PNG.
- **Surface Manager**: maneja el acceso al subsistema de representación gráfica en 2D y 3D.
- **WebKit**: soporta un moderno navegador web utilizado en el navegador Android y en la vista webview. Se trata de la misma librería que utiliza Google Chrome y Safari de Apple.
- **SGL**: motor de gráficos 2D.
- **Librerías 3D**: implementación basada en OpenGL ES 1.0 API. Las librerías utilizan el acelerador hardware 3D si está disponible, o el software altamente optimizado de proyección 3D.
- **FreeType**: fuentes en bitmap y renderizado vectorial.
- **SQLite**: potente y ligero motor de bases de datos relacionales disponible para todas las aplicaciones.
- **SSL**: proporciona servicios de encriptación *Secure Socket Layer*.

### 1.4.4. Entorno de aplicación

Proporciona una plataforma de desarrollo libre para aplicaciones con gran riqueza e innovaciones (sensores, localización, servicios, barra de notificaciones, etc.).

Esta capa ha sido diseñada para simplificar la reutilización de componentes. Las aplicaciones pueden publicar sus capacidades y otras pueden hacer uso de ellas (sujetas a las restricciones de seguridad). Este mismo mecanismo permite a los usuarios remplazar componentes.

Una de las mayores fortalezas del entorno de aplicación de Android es que se aprovecha el lenguaje de programación Java. El SDK de Android no acaba de ofrecer todo lo disponible para su estándar del entorno de ejecución Java (JRE), pero es compatible con una fracción muy significativa de la misma.

Los servicios más importantes que incluye son:

- **Views**: extenso conjunto de vistas (parte visual de los componentes).
- **Resource Manager**: proporciona acceso a recursos que no son en el código.

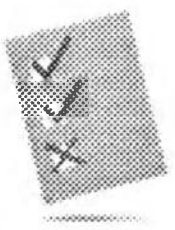
- **Activity Manager**: maneja el ciclo de vida de las aplicaciones y proporciona un sistema de navegación entre ellas.
- **Notification Manager**: permite a las aplicaciones mostrar alertas personalizadas en la barra de estado.
- **Content Providers**: mecanismo sencillo para acceder a datos de otras aplicaciones (como los contactos).

### 1.4.5. Aplicaciones

Este nivel está formado por el conjunto de aplicaciones instaladas en una máquina Android. Todas las aplicaciones han de ser ejecutadas en la máquina virtual Dalvik para garantizar la seguridad del sistema.

Normalmente, las aplicaciones Android están escritas en Java. Para desarrollar aplicaciones en Java podemos utilizar el Android SDK. Existe otra opción consistente en desarrollar las aplicaciones utilizando C/C++. Para esta opción podemos utilizar el Android NDK (*Native Development Kit*).

Polii**[Media]**: *La arquitectura de Android.*



**Preguntas de repaso y reflexión:** *La arquitectura de Android.*

**Enlaces de interés:** *Android, iOS, tiempos de respuestas y por qué nada es gratis en sistemas informáticos.* (Ricardo Galli). Interesante artículo que explica varios aspectos relacionados sobre el funcionamiento interno del sistema operativo Android:

http://gallir.wordpress.com/2011/12/07/android-ios-tiempos-de-respuestas-y-por-que-nada-es-gratis-en-sistemas-informaticos/

## 1.5. Instalación del entorno de desarrollo

Para el desarrollo de las aplicaciones vamos a poder utilizar un potente y moderno entorno de desarrollo. Al igual que Android, todas las herramientas están basadas en *software* libre. Aunque existen varias alternativas para desarrollar aplicaciones en Android. En este texto se supondrá que estamos trabajando con el *software* enumerado a continuación:

- Java Runtime Environment 5.0 o superior.
- Eclipse (Eclipse IDE for Java Developers).

- Android SDK (Google).
- Eclipse Plug-in (Android Development Toolkit- ADT).

Describiremos, a continuación, el proceso a seguir para instalar el *software* anterior. Te proponemos dos alternativas para instalar el entorno de desarrollo. La primera es instalar cada componente por separado, la segunda es instalar *MOTODEV Studio* que incorpora todos estos componentes y alguna herramienta adicional.

## 1.5.1. Instalación de los componentes por separado

Si ya tienes instalado Eclipse en tu ordenador puedes completar la instalación añadiendo *Android SDK* y *Eclipse Plug-in*. De esta forma mantendrás tu configuración actual y simplemente añadirás nuevas funcionalidades. Si no es tu caso te recomendamos pasar al siguiente apartado e instalar *MOTODEV Studio*.

### *1.5.1.1. Instalación de la máquina virtual Java*

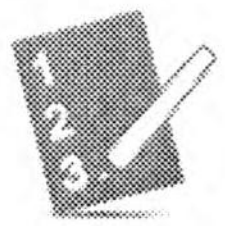
Este *software* va a permitir ejecutar código Java en tu equipo. A la máquina virtual Java también se la conoce como entorno de ejecución Java, *Java Runtime Environment* (JRE) o *Java Virtual Machine* (JVM).

Muy posiblemente ya tengas instalada la máquina virtual Java en tu equipo. Si es así puedes pasar directamente al punto siguiente. En caso de dudas, puedes pasar también al punto siguiente. Al concluirlo te indicará si la versión de la máquina virtual Java es incorrecta. En caso necesario, regresa a este punto para instalar la versión adecuada.

Para instalar la máquina virtual Java accede a http://java.com/es /download/ y descarga e instala el fichero correspondiente a tu sistema operativo.

### *1.5.1.2. Instalación de Eclipse*

Eclipse resulta el entorno de desarrollo más recomendable para Android, es libre y además es soportado por Google (ha sido utilizado por los desarrolladores de Google para crear Android). La versión mínima es la 3.3.1.



**Ejercicio paso a paso:** *Instalación de Eclipse.*

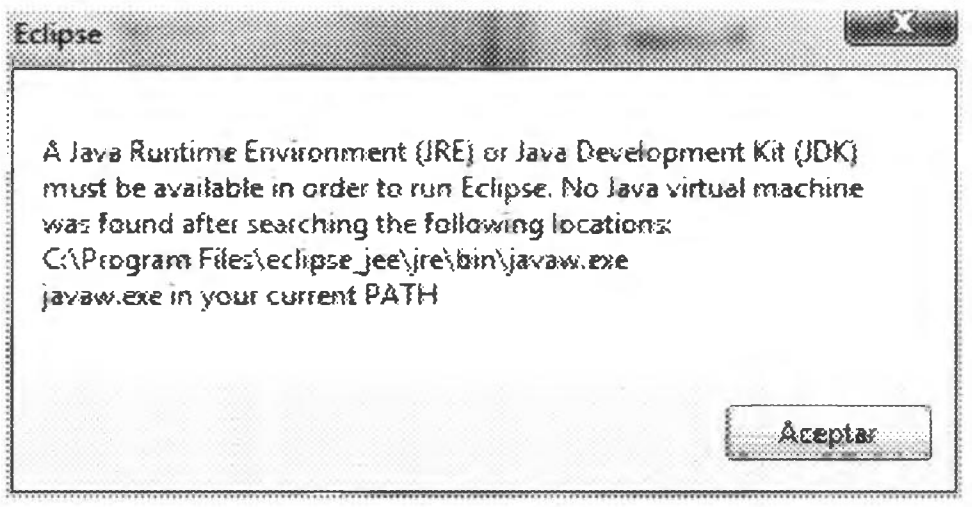
Para instalar Eclipse hay que seguir los siguientes pasos:

1. Accede a la página http://www.eclipse.org/downloads/ y descarga la última versión de "Eclipse IDE para desarrolladores Java". Verás que se encuentra disponible para los sistemas operativos más utilizados, como Windows, Linux y Mac OS.

***NOTA:*** *en este texto hemos utilizado la versión Helios para Windows 32 bits (fichero: "eclipse-java-indigo-win32.zip").*

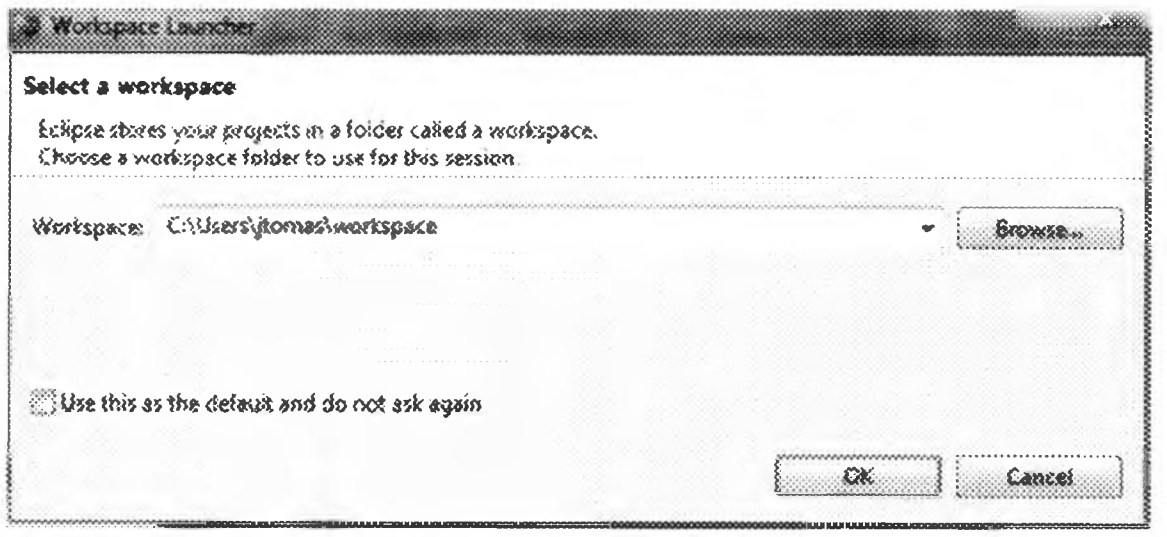
2. Este *software* no requiere una instalación específica, simplemente descomprimir los ficheros en la carpeta que prefieras. Si así lo deseas puedes crear un acceso directo del fichero *eclipse.exe* en el escritorio o en el menú inicio.

***NOTA:*** *Si al ejecutar Eclipse te aparece el siguiente mensaje:*




*Nos indica que no tenemos instalada la máquina virtual Java (o la versión no es la adecuada). Para solucionarlo regresa al punto anterior.*

3. Al arrancar Eclipse comenzará preguntándonos que carpeta queremos utilizar como *workspace*. En esta carpeta se almacenarán los proyectos que crees en Eclipse. Es importante que conozcas su ubicación para poder hacer copias de seguridad de los proyectos.




4. Aparecerá una ventana de bienvenida. Ciérrala y dele un vistazo al entorno de desarrollo.

### *1.5.1.3. Instalar Android SDK de Google*

El siguiente paso va a consistir en instalar Android SDK de Google.

**Ejercicio paso a paso:** *Instalación de Android SDK.*

1. Accede a la siguiente página http://developer.android.com/sdk y descarga el fichero correspondiente a tu sistema operativo.

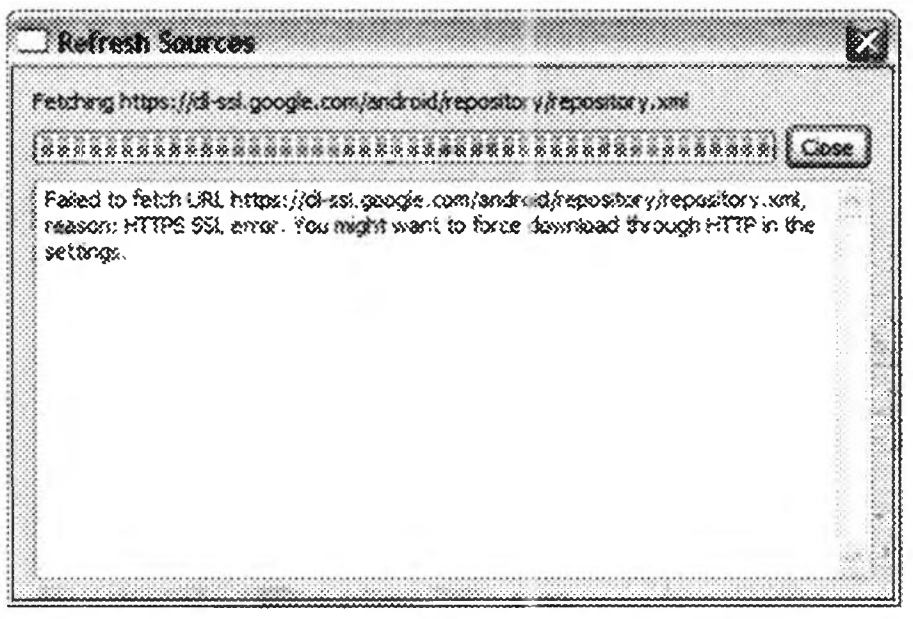
***NOTA:*** *En este texto hemos utilizado el fichero "android-sdk_r12-windows.zip".*

2. Este *software* no requiere una instalación específica, simplemente descomprimir los ficheros en la carpeta que prefieras.

***NOTA:*** *En algunos sistemas tendremos problemas cuando la ruta donde se descomprime los ficheros contiene un espacio en blanco.*

3. Ejecuta el programa `SDK Manager`.

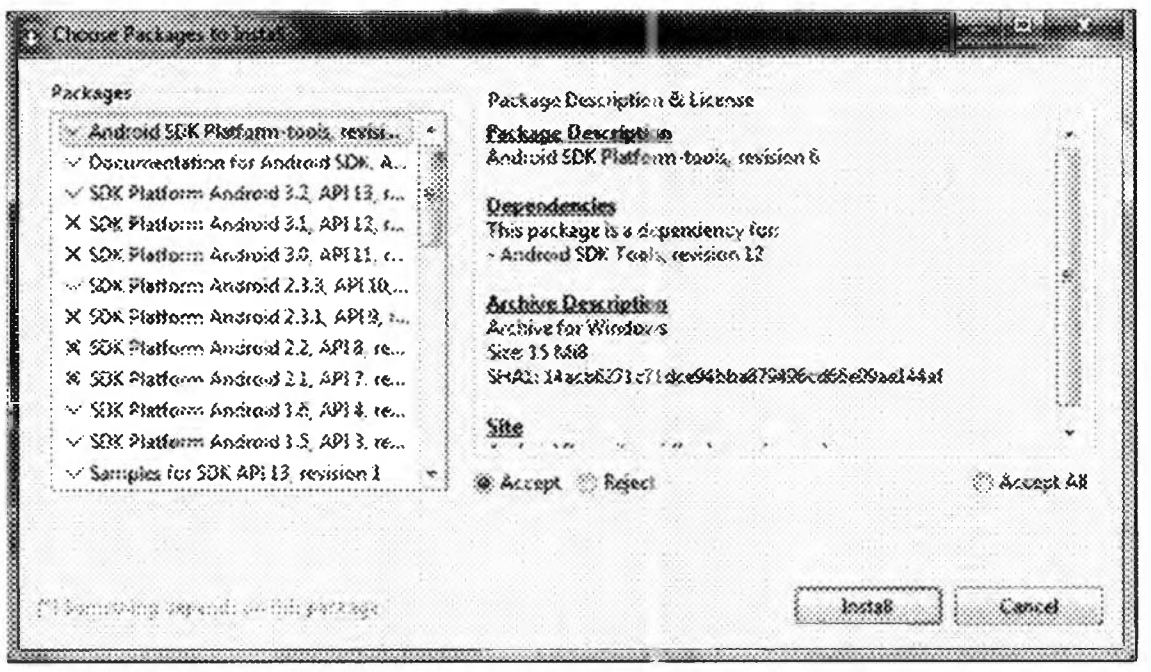
***NOTA:*** *Es posible que te aparezca el siguiente error:*




*En tal caso, hay que forzar al programa a utilizar el protocolo http en lugar del protocolo https. Para ello sigue los siguientes pasos:*

- *Cierra la ventana Refresh Sources.*
- *Pulsa Cancel en la ventana Choose Packages to Install.*
- *Selecciona Settings en la parte izquierda de la ventana Android SDK and AVD Manager.*
- *Marca el checkbox Force https://... sources to be fetched using http://...*
- *Pulsar Save and Apply.*
- *Cierra y vuelve a ejecutar el programa* `SDK Manager`.

4. Seleccionar los paquetes a instalar.

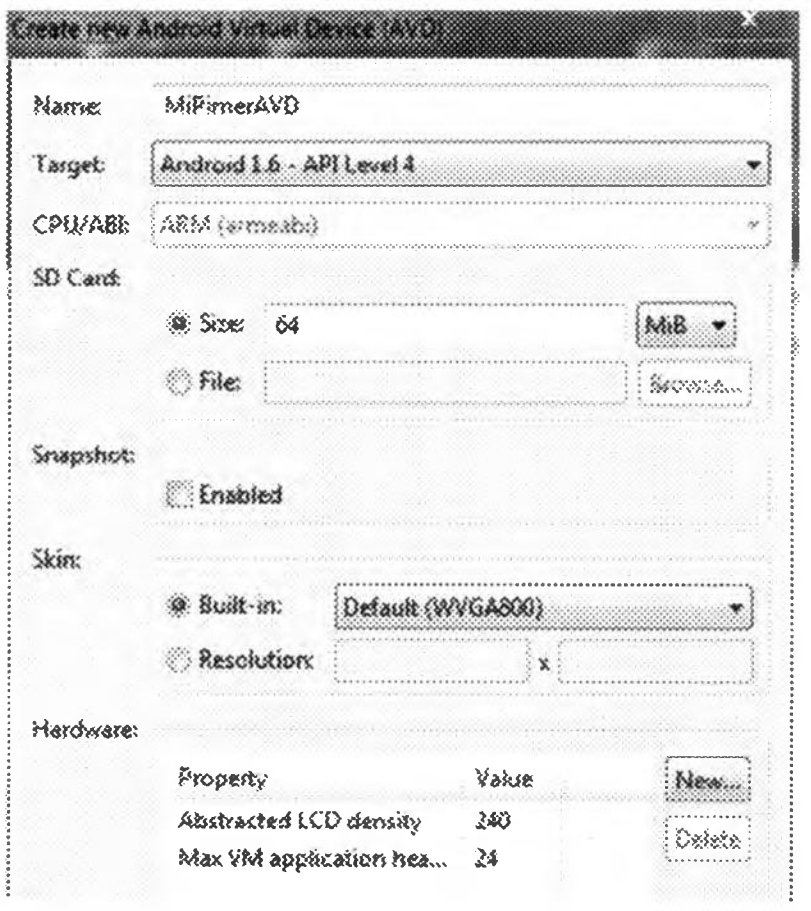



Aparecerá una ventana donde podremos seleccionar los paquetes a instalar. Si lo deseas puedes instalar todos los paquetes (*Accept All*); en

este caso el proceso de instalación puede tardar más de una hora. Si no dispones de tanto tiempo puedes seleccionar solo algunos paquetes, por ejemplo los que se muestran en la captura anterior. Más adelante podrás instalar más paquetes si necesitas otras plataformas de desarrollo u otras máquinas virtuales.

5. Crea un nuevo dispositivo virtual Android (AVD): un dispositivo virtual Android te permite emular en tu ordenador cualquier tipo de dispositivos móvil. De esta forma podrás probar tus aplicaciones en gran variedad de teléfonos y tabletas.

6. Selecciona *Virtual Devices* en la parte izquierda de la ventana *Android SDK and AVD Manager*. Pulsa a continuación el botón *New*. Aparecerá la siguiente ventana:



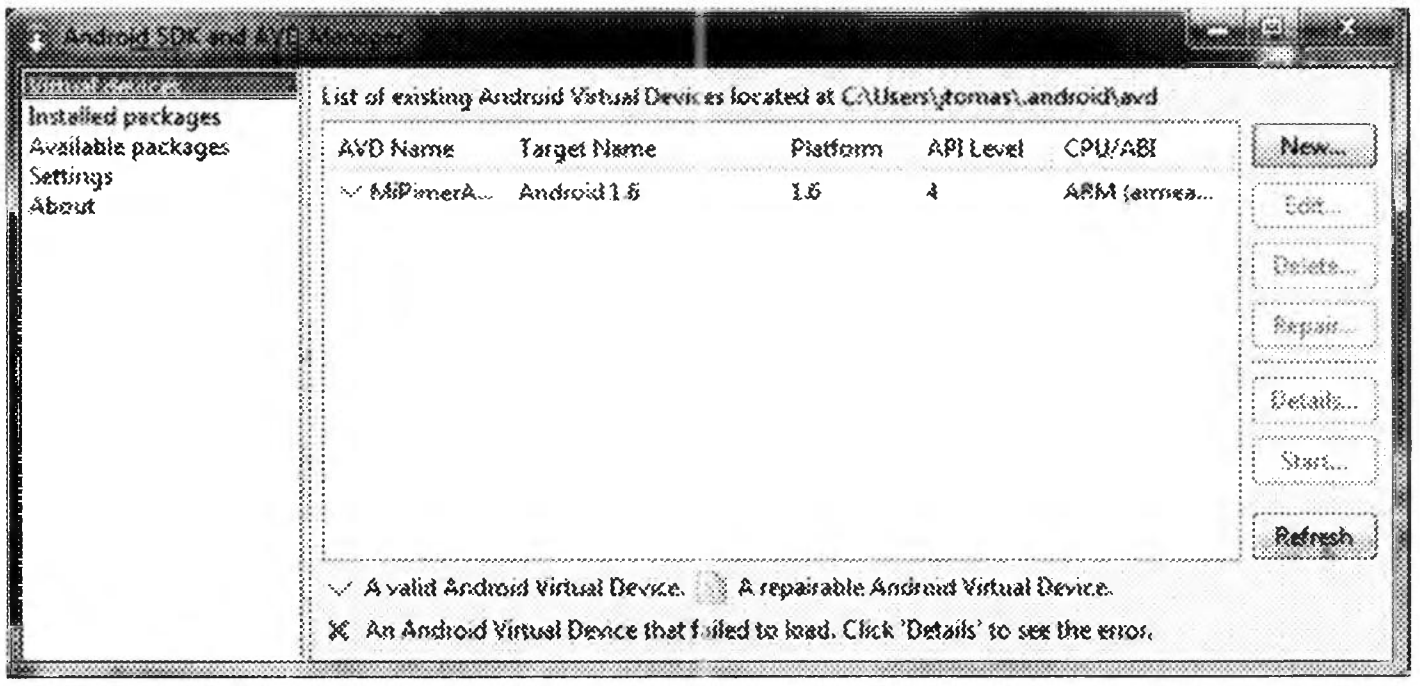

Donde tendremos que introducir los siguientes datos:

- ***Name:*** Nombre que quieras dar al nuevo dispositivo virtual.
- ***Target:*** Versión SDK que soportará el dispositivo.
- ***SD Card:*** Podemos introducir el tamaño de la tarjeta SD usada en el emulador y de forma opcional indicar el fichero donde almacenará los datos. No debes excederte en la cantidad de memoria, ésta se reserva en el disco duro.
- ***Snapshot:*** Si lo seleccionas podrás congelar la ejecución del dispositivo en un determinado instante. Más tarde podrás retomar la ejecución en ese mismo instante, sin tener que esperar a que se inicie el dispositivo.
- ***Skin:*** Podremos seleccionar la resolución de la pantalla, indicando uno de los valores predefinidos o una resolución personalizada.
- ***Hardware:*** Permite definir las características hardware del dispositivo. Podemos indicar cosas como si dispone de GPS,

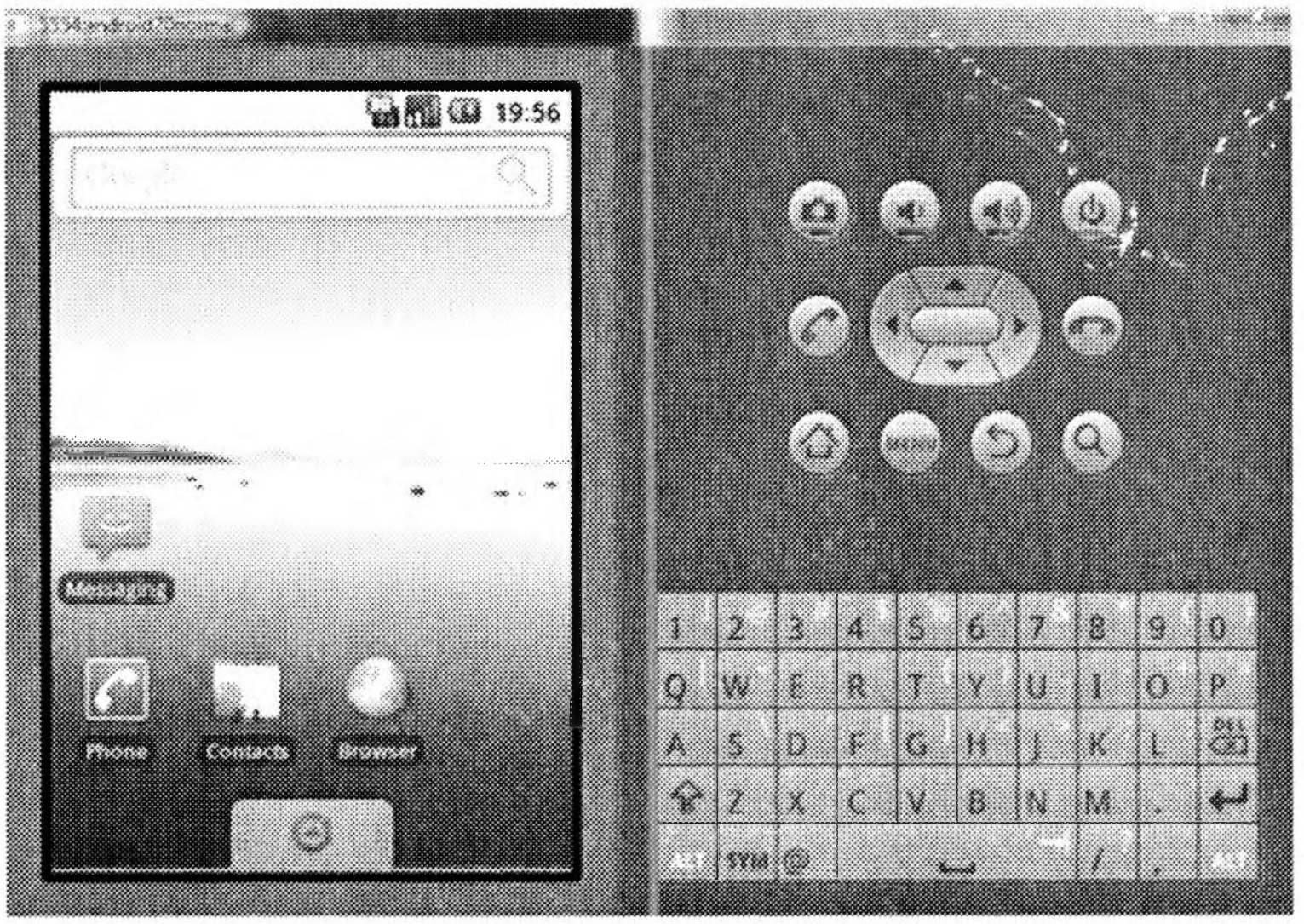
teclado físico, etc. No es imprescindible que definas estas características. Es decir, si quieres probar un programa que haga uso del GPS no resulta imprescindible que en el AVD hayas indicado que dispone de GPS.

Una vez introducida la configuración pulsa el botón *Create AVD*.

7. Arranque el dispositivo.



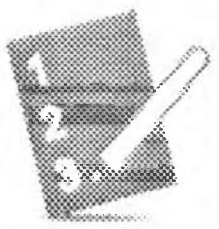

Una vez creado el emulador (AVD), podremos ponerlo en ejecución seleccionándolo de la lista y pulsando el botón *Start...* Aparecerá la ventana *Launch Options*. Pulsa el botón *Launch*. Puede tardar unos minutos en arrancar.




***NOTA:*** *El emulador mostrado no está previsto para duplicar un dispositivo en particular, sino para tratar algunas de las características principales. Algunas características de hardware no están disponibles en el emulador, por ejemplo, el multi-touch o los sensores.*

## 1.5.2. Instalación del plug-in Android para Eclipse (ADT)

El último paso consiste en instalar el plug-in Android para Eclipse, también conocido como ADT. Este software desarrollado por Google, instala una serie de complementos en Eclipse, de forma que el entorno de desarrollo se adapte al desarrollo de aplicaciones para Android. Se crearán nuevos botones, tipos de aplicación, vistas, etc., para integrar Eclipse con el Android SDK que acabamos de instalar.



**Ejercicio paso a paso**: *Instalación del plug-in Android para Eclipse (ADT).*

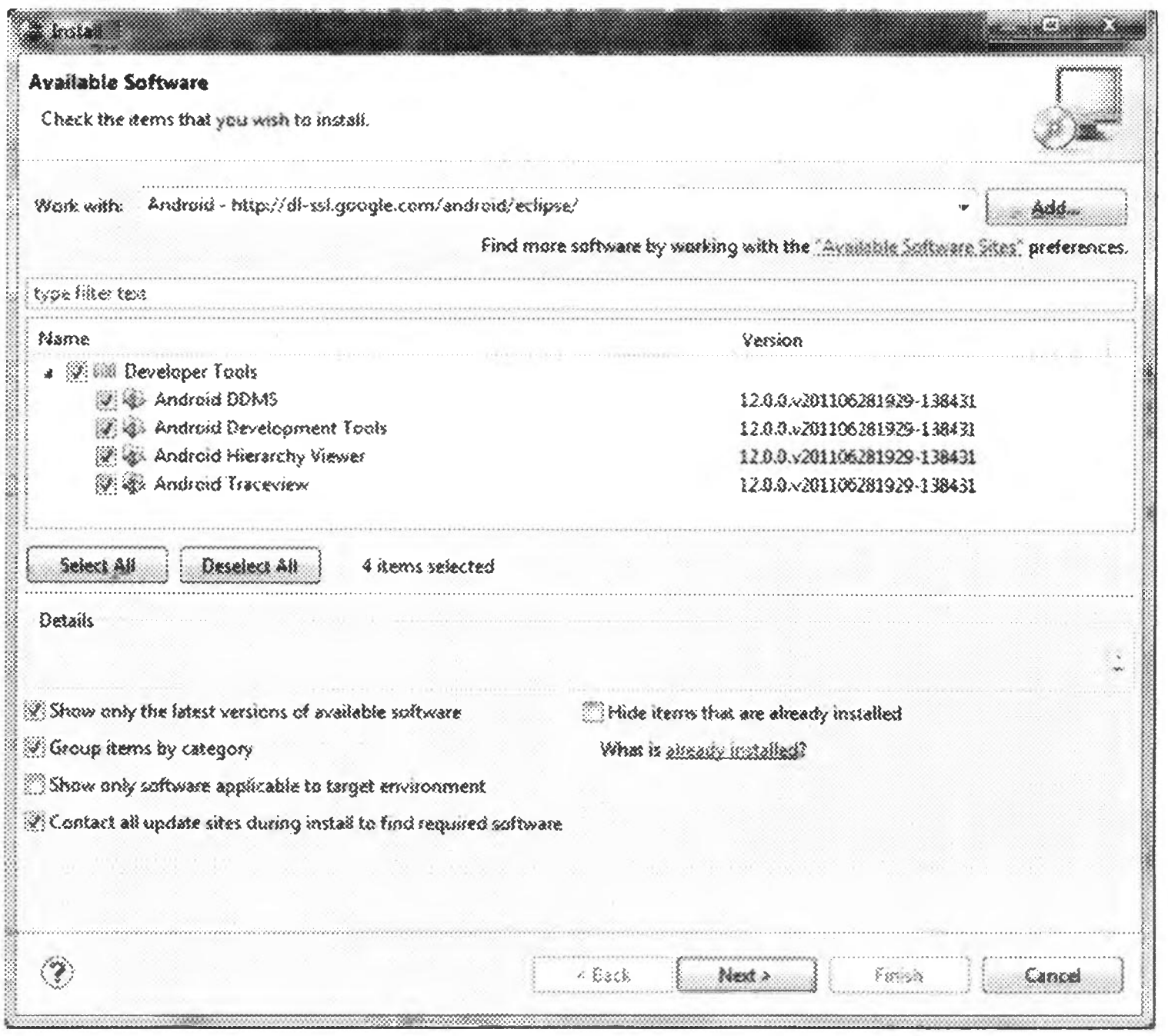
Para instalar el plug-in Android sigue los siguientes pasos:

1. Arranca Eclipse y selecciona *Help>Install New Software*.
2. En el cuadro de diálogo *Available Software* que aparece, haz clic en *Add...* En el cuadro de diálogo *Add Site* que sale introduce un nombre para el sitio remoto (por ejemplo, *Plug-in Android*) en el campo *Name*. En el campo *Location*, introduce la siguiente URL:

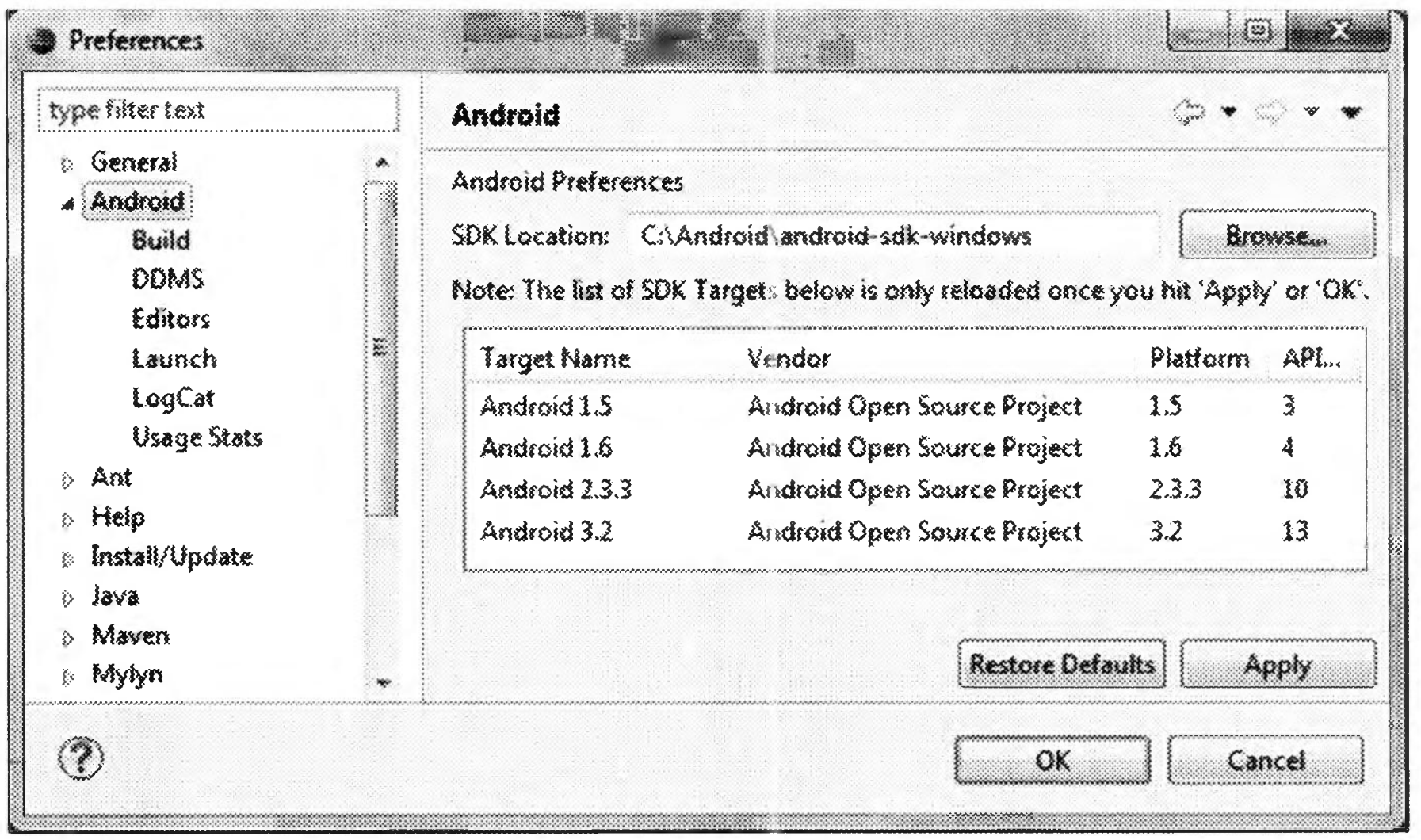
```
http://dl-ssl.google.com/android/eclipse/
```

***NOTA:*** *Si tienes algún problema en adquirir el plug-in, puedes intentar utilizar https en el URL en vez de http. Finalmente pulsa OK.*

Ahora en el cuadro de diálogo *Available Software* debe aparecer *Developer Tools* en la lista.



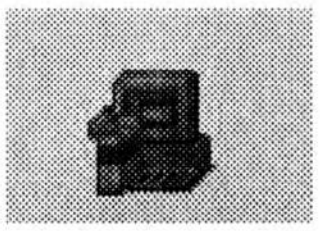
3. Selecciona los paquetes a instalar y pulsa *Next*. Ahora aparecen listadas las características de Android DDMS y Android Development Tools.
4. Pulsa *Next* para leer y aceptar la licencia e instalar cualquier dependencia y pulsa *Finish*.
5. Reinicia Eclipse.
6. Configura Eclipse para que sepa donde se ha instalado el Android SDK. Para ello entra en las preferencias en *Windows>Preferences...* y selecciona Android del panel de la izquierda. Ahora pulsa *Browse...* para selecciona el *SDK Location* y elige la ruta donde hayas descomprimido Android SDK. Aplica los cambios y pulsa *OK*.




## 1.5.3. Instalación con MOTODEV Studio

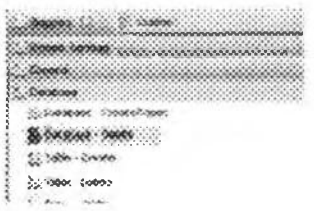
*MOTODEV Studio for Android* es un entorno de desarrollo creado por Motorola para el desarrollo de aplicaciones Android. Se basa en Eclipse y añade una serie de características adicionales. Es la opción de instalación más sencilla.

**Recursos adicionales:** *Características adicionales de MOTODEV Studio no incluidas en Eclipse.*



**Instalación más sencilla:** Un solo instalador y conseguirás disponer de Eclipse IDE, herramientas Java y plugins de Android perfectamente integrados. Si ya tienes Eclipse instalado puedes instalar también *MOTODEV Studio* como plugin.


**Validación de la aplicación:** Detectar y reparar condiciones inadecuadas, tales como permisos que faltan y configuraciones que están en conflicto con las especificaciones del dispositivo.



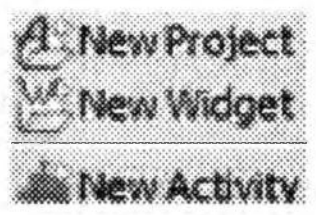
**Fragmentos de código:** Agrega código para resolver tareas frecuentes. Ejemplo: enviar SMS, mostrar un Toast, hacer vibrar el teléfono, etc.


**Gestor de bases de datos:** Permite ver y editar las bases de datos SQLite para tus aplicaciones Android.


**Emulador integrado en el entorno de desarrollo:** Prueba tus aplicaciones sin cambiar del entorno de desarrollo al emulador del terminal.

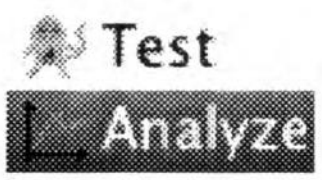

**Video Tutoriales:** *MOTODEV Studio* dispone de muchos tutoriales que se puede ver desde el interior del entorno de desarrollo.


**Traducido al castellano:** Si tu ingles no es muy bueno puedes instalar la versión en castellano

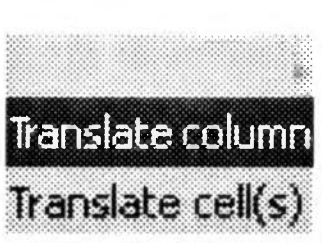



**Asistentes para la creación de aplicaciones:** Crea nuevos proyectos y clases esenciales de Android mediante sencillos asistentes.


**Asistente de generación de código:** Deja que *Studio MOTODEV* genere parte del código repetitivo de Java. Sólo tiene que describir la lógica de negocio.

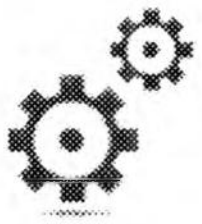



**Herramientas de diagnóstico:** Utiliza las avanzadas herramientas de diagnóstico para detectar problemas de memoria y probar las aplicaciones antes de su distribución.




**Traducción automática de aplicaciones:** Si quieres que tu aplicación esté disponible en múltiples idiomas, esta herramienta te facilitará el trabajo.


**Escribe aplicaciones nativas:** Si necesitas algo más de velocidad puedes escribir aplicaciones nativas con Native Development Kit (NDK).



**Práctica:** *Instalación de MOTODEV Studio for Android.*

1. Abre la página *Web MOTODEV Studio for Android* (http://developer.motorola.com/tools/motodevstudio/).
2. Descarga el software adecuado según la versión de tu sistema operativo.
3. Sigue las instrucciones de instalación.

**Recursos adicionales:** *Teclas de acceso rápido imprescindibles.*

Ctrl+O: Añade *imports* de las clases no resueltas.

Ctrl+F: Formatea automáticamente el código.

Ctrl+Espacio: Auto completar.

**Enlaces de interés:**

- *Página oficial de Eclipse:* Podrás encontrar todas las versiones e información sobre los proyectos Eclipse.

  http://www.eclipse.org/

- *Mi primera hora con Eclipse:* Interesante si quieres sacarle el máximo provecho a esta herramienta, aunque mucho de lo que se explica no resulta imprescindible para el curso.

  http://ubuntulife.files.wordpress.com/2008/03/intro_eclipse_espanol.pdf

- *MOTODEV Studio for Android:* Puedes verificar si hay nuevas versiones en la página oficial y visionar video-tutoriales.

  http://developer.motorola.com/docstools/motodevstudio/

## 1.6. Las versiones de Android y niveles de API

Antes de empezar a proyecto en Android hay que elegir la versión del sistema para la que deseamos realizar la aplicación. Es muy importante observar que hay clases y métodos que están disponibles a partir de una versión. Si las vamos a usar, hemos de conocer la versión mínima necesaria.

Cuando se ha lanzado una nueva plataforma siempre ha sido compatible con las versiones anteriores. Es decir, solo se añaden nuevas funcionalidades y, en el caso

de modificar alguna funcionalidad, no se elimina, se etiquetan como obsoletas pero se pueden continuar utilizando.

A continuación se describen las plataformas lanzadas hasta la fecha con una breve descripción de las novedades introducidas. Las plataformas se identifican de tres formas alternativas: versión, nivel de API y nombre. El nivel de API corresponde a números enteros comenzando desde 1. Para los nombres se han elegido postres en orden alfabético Cupcake (v1.5), Donut (v1.6), Éclair (v2.1), Froyo (v2.2), Gingerbread (v2.3), Honeycomb (v3.0), Ice Cream (v2.4), etc. Las dos primeras versiones, que hubieran correspondido a las letras A y B, no recibieron nombre.

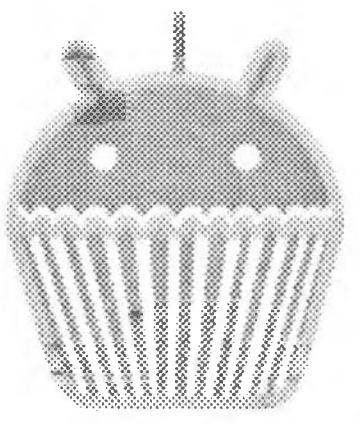
### 1.6.1. Android 1.0 Nivel de API 1 (septiembre 2008)

Primera versión de Android. Nunca se utilizó comercialmente, por lo que no tiene mucho sentido desarrollar para esta plataforma.

### 1.6.2. Android 1.1 Nivel de API 2 (febrero 2009)

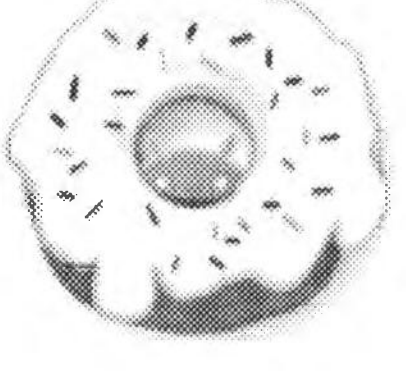
No se añadieron apenas funcionalidades, simplemente se fijaron algunos errores de la versión anterior. Es la opción a escoger si queremos desarrollar una aplicación compatible con todos los dispositivos Android. No obstante apenas existen usuarios con esta versión.

### 1.6.3. Android 1.5 Nivel de API 3 (abril 2009, Cupcake)



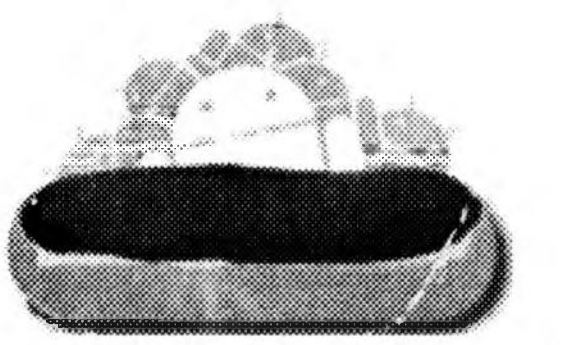
Es la primera versión con algún usuario (un 1,3% en julio del 2011). Como novedades, incorpora la posibilidad de teclado en pantalla con predicción de texto, los terminales ya no tienen que tener un teclado físico, así como la capacidad de grabación avanzada de audio y vídeo. También aparecen los *widgets* de escritorio y *live folders*. Incorpora soporte para *bluetooth* estéreo, por lo que permite conectarse automáticamente a auriculares *bluetooth*. Las transiciones entre ventanas se realizan mediante animaciones.

### 1.6.4. Android 1.6 Nivel de API 4 (diciembre 2009, Donut)



Permite capacidades de búsqueda avanzada en todo el dispositivo. También incorpora *gestures* y *multi-touch*. Permite la síntesis de texto a voz. También facilita que una aplicación pueda trabajar con diferentes densidades de pantalla. Soporte para resolución de pantallas WVGA. Aparece un nuevo atributo XML, `onClick`, que puede especificarse en una vista. Se mejora Play Store, antes Android Market, permitiendo una búsqueda más sencilla de aplicaciones. Soporte para CDMA/EVDO, 802.1x y VPNs. Mejoras en la aplicación de la cámara.

## 1.6.5. Android 2.0 Nivel de API 5 (octubre 2009, Éclair)



Esta versión de API apenas cuenta con usuarios, dado que la mayoría de fabricantes pasaron directamente de la versión 1.6 a la 2.1. Como novedades cabría destacar que incorpora un API para manejar *bluetooth* 2.1. Nueva funcionalidad que permite sincronizar adaptadores para conectarlo a cualquier dispositivo. Ofrece un servicio centralizado de manejo de cuentas. Mejora la gestión de contactos y ofrece más ajustes en la cámara. Se optimiza la velocidad de *hardware*. Se aumenta el número de tamaños de ventana y resoluciones soportadas. Nueva interfaz del navegador y soporte para HTML5. Mejoras en el calendario y soporte para Microsoft Exchange. La clase `MotionEvent` ahora soporta eventos en pantallas multitáctil.

## 1.6.6. Android 2.1 Nivel de API 7 (enero 2010, Éclair)

Se considera una actualización menor, por lo que le siguieron llamando Éclair. Destacamos el reconocimiento de voz que permite introducir un campo de texto sin necesidad de utilizar el teclado. También permite desarrollar fondos de pantalla animados. Se puede obtener información sobre la señal de la red actual que posea el dispositivo. En el paquete WebKit se incluyen nuevos métodos para manipular bases de datos almacenadas en Web. También se permite obtener permisos de geolocalización y modificarlos en WebView. Incorpora mecanismos para administrar la configuración de la caché de aplicaciones, almacenamiento web y modificar la resolución de la pantalla. También se puede manejar vídeo, historial de navegación, vistas personalizadas, etc.

## 1.6.7. Android 2.2 Nivel de API 8 (mayo 2010, Froyo)

Como característica más destacada se puede indicar la mejora de velocidad de ejecución de las aplicaciones (ejecución del código de la CPU de 2 a 5 veces más rápido que en la versión 2.1 de acuerdo a varios *benchmarks*). Esto se consigue con la introducción de un nuevo compilador JIT de la máquina Dalvik.

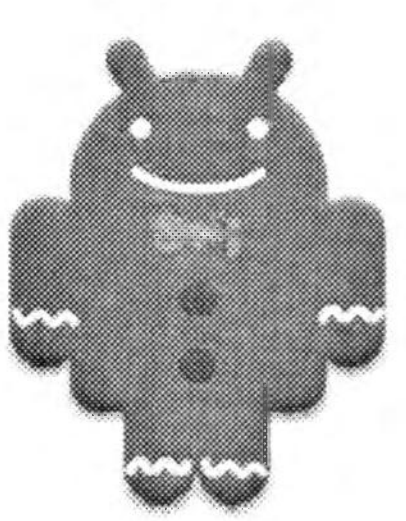
Se añaden varias mejoras relacionadas con el navegador Web, como el soporte de Adobe Flash 10.1 y la incorporación del motor Javascript V8 utilizado en Chrome, o la incorporación del campo de "subir fichero" en un formulario.

El desarrollo de aplicaciones permite las siguientes novedades: se puede preguntar al usuario si desea instalar una aplicación en un medio de almacenamiento externo (como una tarjeta SD), como alternativa a la instalación en la memoria interna del dispositivo. Las aplicaciones se actualizan de forma automática cuando aparece una nueva versión. Proporciona un servicio para la copia de seguridad de datos que se puede realizar desde la propia aplicación para garantizar al usuario el mantenimiento de sus datos. Por último, facilita que las aplicaciones interaccionen con el reconocimiento de voz y que terceras partes proporcionen nuevos motores de reconocimiento.

Se mejora la conectividad: ahora podemos utilizar nuestro teléfono para dar acceso a Internet a otros dispositivos (*tethering*), tanto por USB como por Wi-Fi. También se añade el soporte a Wi-Fi IEEE 802.11n.

Se añaden varias mejoras en diferentes componentes: en el API gráfica OpenGL ES se pasa a soportar la versión 2.0. También puede realizar fotos o vídeos en cualquier orientación (incluso vertical) y configurar otros ajustes de la cámara. Para finalizar, permite definir modos de interfaz de usuario ("automóvil" y "noche") para que las aplicaciones se configuren según el modo seleccionado por el usuario.

### 1.6.8. Android 2.3 Nivel de API 9 (diciembre 2010, Gingerbread)



Debido al éxito de Android en las nuevas tabletas, ahora soporta mayores tamaños de pantalla y resoluciones (WXGA y superiores).

Incorpora un nuevo interfaz de usuario con un diseño actualizado. Dentro de las mejoras de la interfaz de usuario destacamos la mejora de la funcionalidad de "cortar, copiar y pegar" y un teclado en pantalla con capacidad multitáctil.

Incluye soporte nativo para varias cámaras, pensado en la segunda cámara usada en videoconferencia. La incorporación de esta segunda cámara ha propiciado la inclusión de reconocimiento facial para identificar al usuario del terminal.

La máquina virtual Dalvik para Android introduce un nuevo recolector de basura que minimiza las pausas de la aplicación, ayudando a garantizar una mejor animación y el aumento de la capacidad de respuesta en juegos y aplicaciones similares. Se trata de corregir así una de las lacras de este sistema operativo móvil, que en versiones previas no ha sido capaz de cerrar bien las aplicaciones en desuso. Se dispone de mayor apoyo para el desarrollo de código nativo (NDK).También se mejora la gestión de energía y control de aplicaciones. Y se cambia el sistema de ficheros, que pasa de YAFFS a EXT4.

Entre otras novedades destacamos el soporte nativo para telefonía sobre Internet VoIP/SIP. El soporte para reproducción de vídeo WebM/VP8 y codificación de audio AAC. El soporte para la tecnología NFC. Las facilidades en el audio, gráficos y entradas para los desarrolladores de juegos. El soporte nativo para más sensores (como giroscopios y barómetros) y un gestor de descargas para las descargas largas.

### 1.6.9. Android 3.0 Nivel de API 11 (febrero 2011, Honeycomb)

Para mejorar la experiencia de Android en las nuevas tabletas se lanza la versión 3.0 optimizada para dispositivos con pantallas grandes. La nueva interfaz de usuario ha sido completamente rediseñada con paradigmas nuevos para la interacción, navegación y personalización. La nueva interfaz se pone a disposición de todas las aplicaciones, incluso las construidas para versiones anteriores de la plataforma.

Las principales novedades de este SDK son:

Con el objetivo de adaptar la interfaz de usuario a pantallas más grandes se incorporan las siguientes características: resolución por defecto WXGA (1280×800), escritorio 3D con *widgets* rediseñados, nuevos componentes y vistas, notificaciones mejoradas, arrastrar y soltar, nuevo cortar y pegar, barra de acciones para que las aplicaciones dispongan de un menú contextual siempre presente y otras características para aprovechar las pantallas más grandes.

Se mejora la reproducción de animaciones 2D/3D gracias al renderizador OpenGL acelerado por hardware. El nuevo motor de gráficos Rederscript saca un gran rendimiento de los gráficos en Android e incorpora su propia API.

Primera versión de la plataforma que soporta procesadores multinúcleo. La máquina virtual Dalvik ha sido optimizada para permitir multiprocesado, lo que permite una ejecución más rápida de las aplicaciones, incluso aquellas que son de hilo único.

Se incorporan varias mejoras multimedia, como listas de reproducción M3U a través de *HTTP Live Sreaming*, soporte a la protección de derechos musicales (DRM) y soporte para la transferencia de archivos multimedia a través de USB con los protocolos MTP y PTP.

En esta versión se añaden nuevas alternativas de conectividad, como las nuevas APIS de Bluetooth A2DP y HSP con streaming de audio. También se permite conectar teclados completos por USB o Bluetooth.

El uso de los dispositivos en un entorno empresarial es mejorado. Entre las novedades introducidas destacamos las nuevas políticas administrativas con encriptación del almacenamiento, caducidad de contraseña y mejoras para administrar los dispositivos de empresa de forma eficaz.

A pesar de la nueva interfaz gráfica optimizada para tabletas, Android 3.0 es compatible con las aplicaciones creadas para versiones anteriores. La tecla de menú, inexistente en las nuevas tabletas, es remplazada por un menú que aparece en la barra de acción.

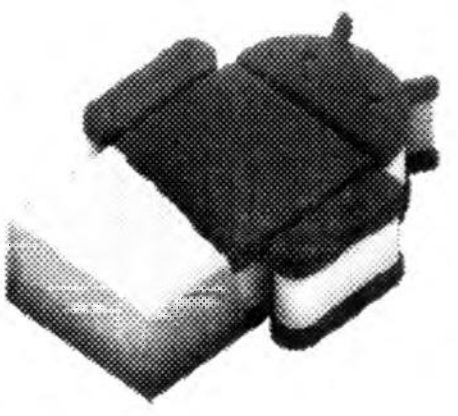
### 1.6.10. Android 3.1 Nivel de API 12 (mayo 2011)

Permite manejar dispositivos conectados por USB (tanto host como dispositivo). Protocolo de transferencia de fotos y vídeo (PTP/MTP) y de tiempo real (RTP).

### 1.6.11. Android 3.2 Nivel de API 13 (julio 2011)

Optimizaciones para distintos tipos de tableta. Zoom compatible para aplicaciones de tamaño fijo. Sincronización multimedia desde SD.

### 1.6.12. Android 4.0 Nivel de API 14 (octubre 2011, Ice Cream Sandwich)



La característica más importante es que se unifican las dos versiones anteriores (2.x para teléfonos y 3.x para tabletas) en una sola compatible con cualquier tipo de dispositivo. Entre las características más interesantes destacamos:

Se introduce un nuevo interfaz de usuario totalmente renovado. Por ejemplo, se remplazan los botones físicos por botones en pantalla (como ocurría en las versiones 3.x).

Nuevo API de reconocedor facial que permite, entre otras muchas aplicaciones, desbloquear el teléfono. También se mejora en el reconocimiento de voz. Por ejemplo, se puede empezar a hablar en cuanto pulsamos el botón.

Aparece un nuevo gestor de tráfico de datos por Internet, donde podremos ver el consumo de forma gráfica y donde podemos definir los límites a ese consumo para evitar cargos inesperados con la operadora. Incorpora herramientas para la edición de imágenes en tiempo real, con herramientas para distorsionar, manipular e interactuar con la imagen al momento de ser capturada. Se mejora el API para comunicaciones por NFC y la integración con redes sociales.

En diciembre del 2011 aparece una actualización de mantenimiento (versión 4.0.2) que no aumenta el nivel de API.

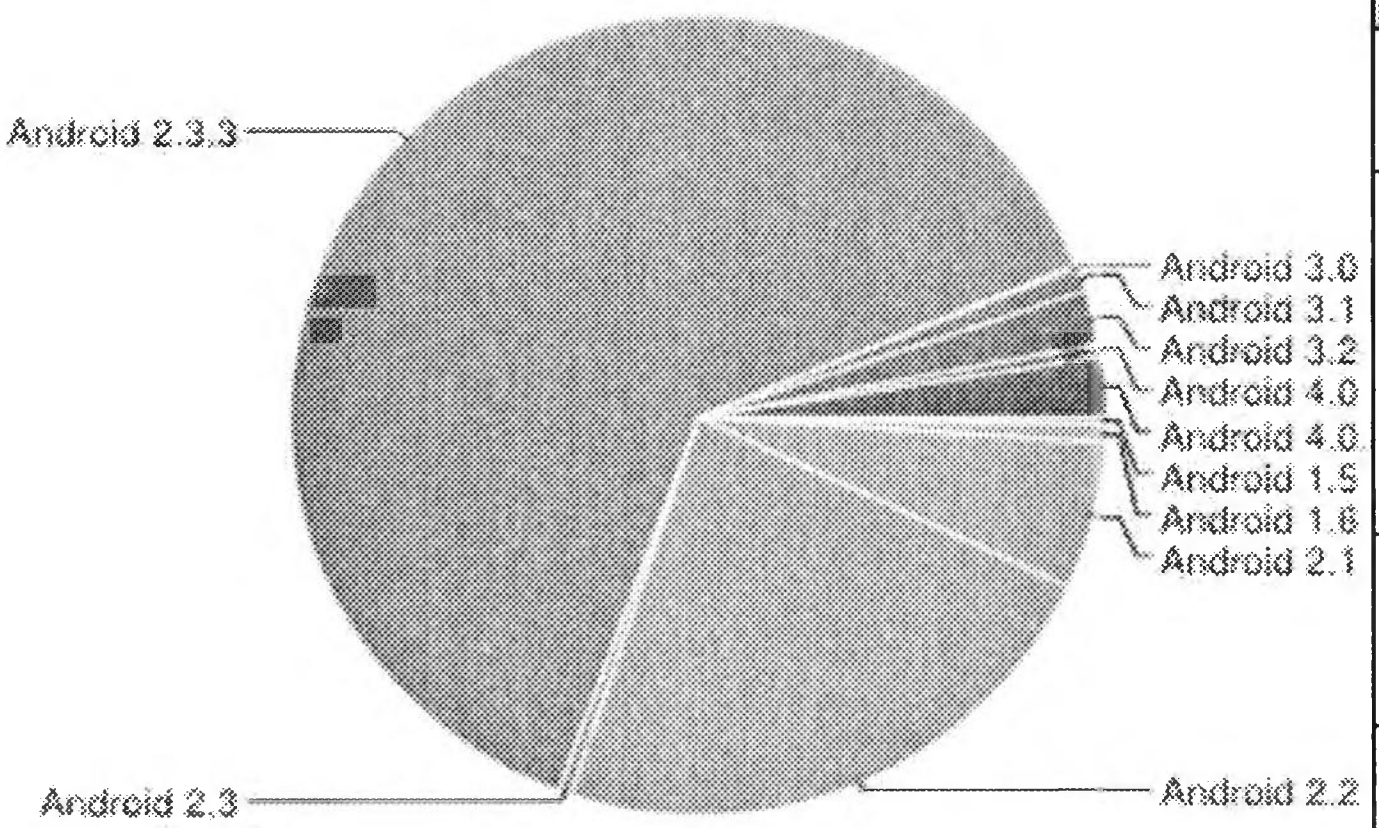
### 1.6.13. Android 4.0.3 Nivel de API 15 (diciembre 2011)

Se introducen ligeras mejoras en algunas APIs incluyendo el de redes sociales, calendario, revisor ortográfico, texto a voz y bases de datos entre otros. En marzo de 2012 aparece la actualización 4.0.4.

### 1.6.14. Elección de la plataforma de desarrollo

A la hora de seleccionar la plataforma de desarrollo hay que consultar si necesitamos alguna característica especial que solo esté disponible a partir de una versión. Todos los usuarios con versiones inferiores a la seleccionada no podrán instalar la aplicación. Por lo tanto, es recomendable seleccionar la menor versión posible que nuestra aplicación pueda soportar. Por ejemplo, si nuestra aplicación necesita utilizar varios cursores simultáneos en la pantalla táctil (*multi-touch*), tendremos que utilizar la versión 1.6, al ser la primera que lo soporta. Pero, la aplicación no podrá ser instalada en versiones anteriores. Para ayudarnos a tomar la decisión de que plataforma utilizar puede ser interesante consultar los porcentajes de utilización[2].

[2] http://developer.android.com/resources/dashboard/platform-versions.html




| Plataforma | Nivel API | Porcent. |
|---|---|---|
| Android 1.5 | 3 | 0.3% |
| Android 1.6 | 4 | 0.7% |
| Android 2.1 | 7 | 6.0% |
| Android 2.2 | 8 | 23.1% |
| Android 2.3<br>Android 2.3.2 | 9 | 0.9% |
| Android 2.3.3<br>Android 2.3.7 | 10 | 63.2% |
| Android 3.0 | 11 | 0.1% |
| Android 3.1 | 12 | 1.0% |
| Android 3.2 | 13 | 2.2% |
| Android 4.0 -<br>Android 4.0.2 | 14 | 0.5% |
| Android 4.0.3 | 15 | 2.4% |

***Figura 3:*** *Dispositivos Android según plataforma instalada, que han accedido a Google Play Store durante dos semanas terminado el 2 de abril de 2012.*

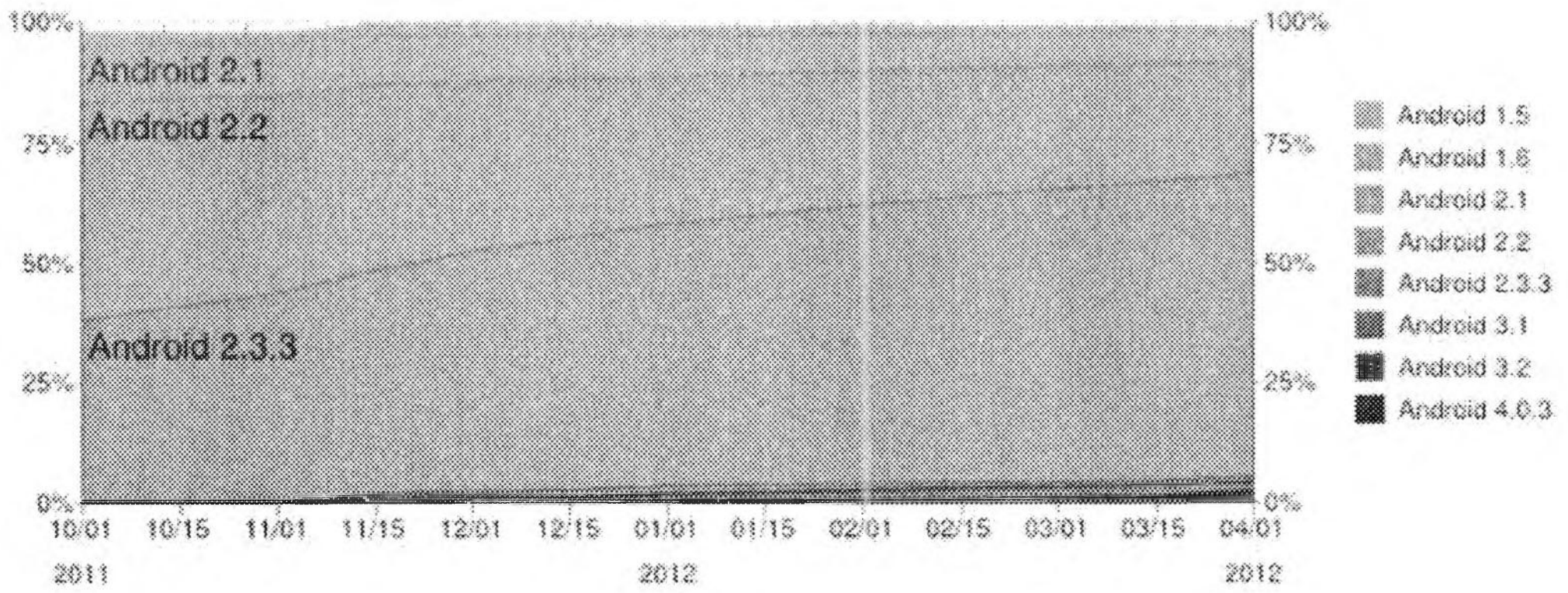



***Figura 4:*** *Evolución del porcentaje de dispositivos Android según la plataforma instalada, hasta el 2 de abril de 2012.*

Tras estudiar las gráficas podemos destacar el reducido número de usuarios que utilizan las versiones 1.x (1%). Por lo tanto, puede ser buena idea utilizar la versión 2.1 para desarrollar nuestro proyecto, dado que daríamos cobertura al 99% de los terminales. Las versiones 3.x, con un 3,3%, han tenido muy poca difusión y presentan una clara tendencia a disminuir. Las versiones 4.x, con un 2,9%, todavía son minoritarias pero es previsible que este porcentaje vaya incrementando.

**Enlaces de interés:**

- *Android developers: Plataform Versions:* Estadística de dispositivos Android según plataforma instalada, que han accedido a Android Market.

  http://developer.android.com/resources/dashboard/platform-versions.html

- *Android developers: SDK:* En el menú de la izquierda aparecen links a las principales versiones de la plataforma. Si pulsas sobre ellos encontrarás una descripción exhaustiva de cada plataforma.

  http://developer.android.com/sdk/index.html

**Preguntas de repaso y reflexión:** *Las versiones de Android.*

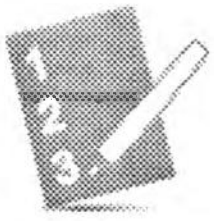
## 1.7. Creación de un primer programa

Utilizar un entorno de desarrollo nos facilita mucho la creación de programas. Esto es especialmente importante en Android, dado que tendremos que utilizar una gran variedad de ficheros. Gracias al plug-in Android que hemos instalado en el entorno de desarrollo Eclipse, la creación y gestión de proyectos se realizará de forma muy rápida, acelerando los ciclos de desarrollo.

Poli[Media]: *Un primer proyecto Android.*

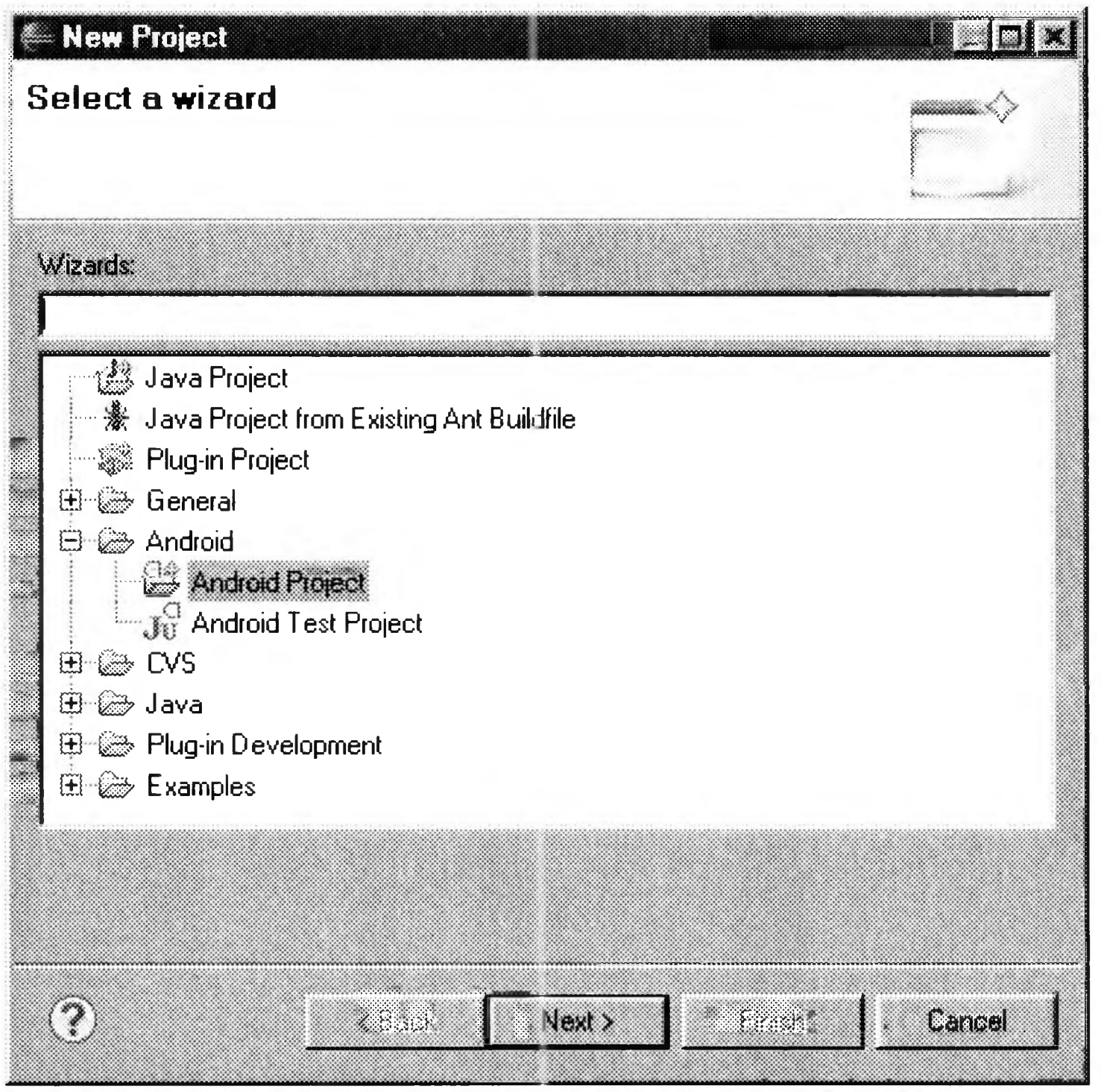
### 1.7.1. Creación del proyecto

Para crear un primer proyecto Android sigue los siguientes pasos:



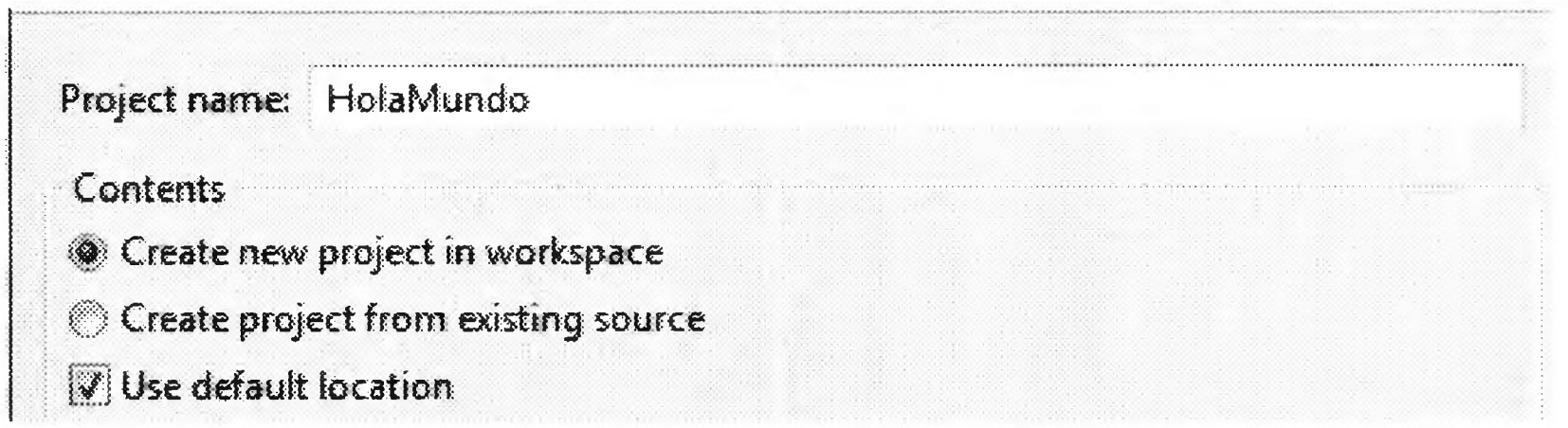
**Ejercicio paso a paso:** *Crear un primer proyecto.*

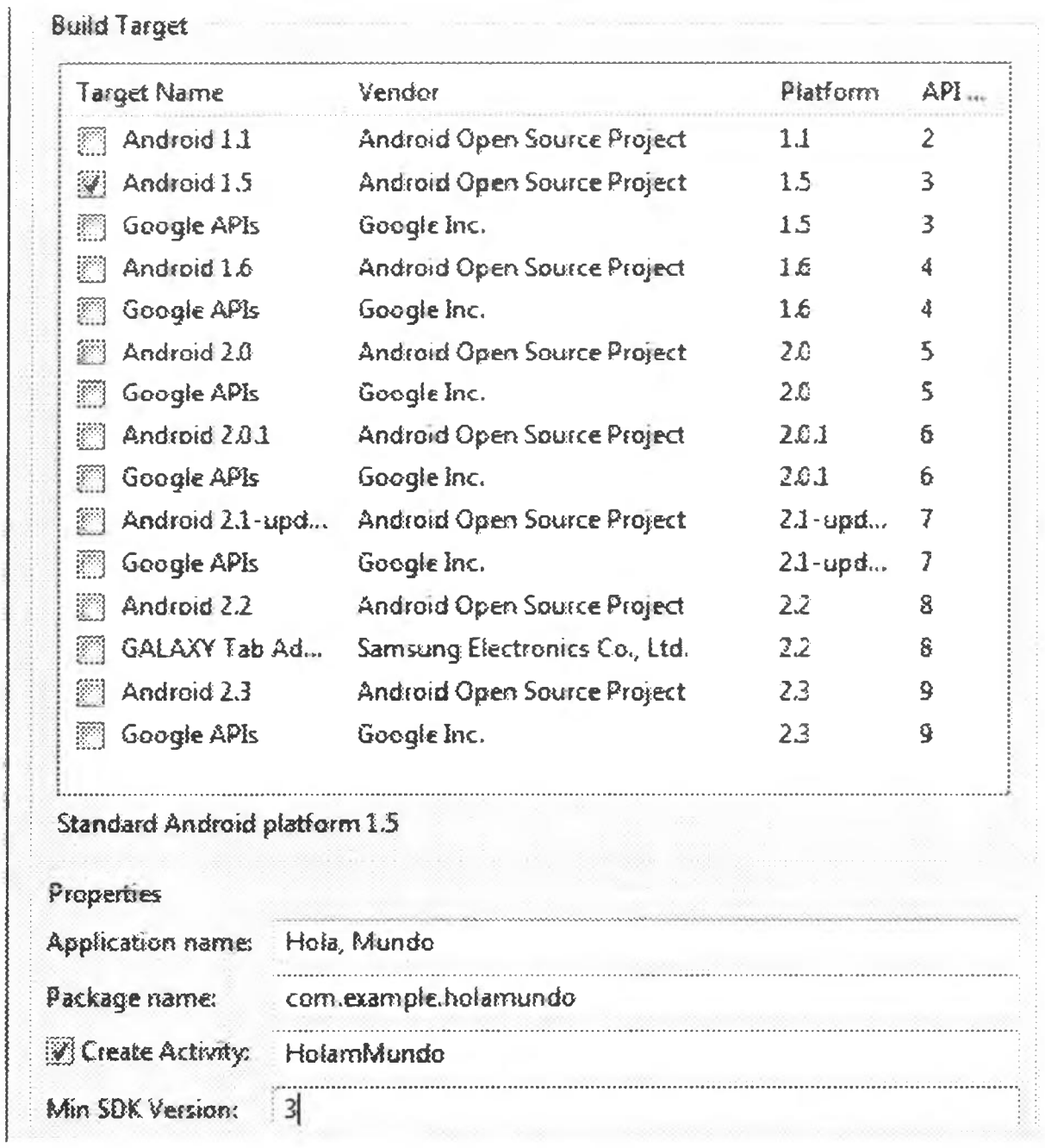
1. Selecciona *File* > *New* > *Project*. Si el plug-in de Android se ha instalado correctamente, el cuadro de diálogo que aparece debe tener un directorio llamado *Android* que debe contener *Android Project*.
2. Selecciona *Android Project* y pulsa *Next*.




3. Rellena los detalles del proyecto con los siguientes valores:

```
Project name: HolaMundo
BuildTarget: Android 1.5
Application name: Hola, Mundo
Package name: com.example.holamundo
Create Activity: HolaMundo
Min SDK Version: 3
```






A continuación vemos una pequeña descripción para cada campo:

***Project Name***: Es el nombre del proyecto. Se creará un directorio con el mismo nombre que contendrá los ficheros del proyecto.

***Build Target***: Es la versión de la plataforma con la que será compilada la aplicación. Normalmente ha de coincidir con el valor introducido en *Min SDK Version*.

***Application Name***: Es el nombre de la aplicación que aparecerá en el dispositivo Android. Tanto en la barra superior cuando esté en ejecución, como en el icono que se instalará en el menú de programas.

***Package Name***: Indicamos el paquete con el espacio de nombres utilizado por nuestra aplicación. Debemos utilizar las reglas de los espacios de nombre en el lenguaje de programación Java. Las clases que creemos estarán dentro de él. Esto también establece el nombre del paquete donde se almacenará la aplicación generada.

**Nota sobre Java:** *El nombre del paquete debe ser único en todos los paquetes instalados en el sistema; por ello, es muy importante utilizar para tus aplicaciones un dominio de estilo estándar en los paquetes. El espacio de nombres "com.example" está reservado para la documentación de ejemplos. Cuando desarrollas tu propia aplicación, debes utilizar un espacio de nombres que sea apropiado.*

***Create Activity***: Este es el nombre de la clase que será generada por el plug-in. Esta será una subclase de *Activity* de Android. Una *actividad* es simplemente una clase que se puede ejecutar y modificar. Se pueden crear las diferentes pantallas de usuario en diferentes actividades. Una actividad casi siempre se utiliza como base para una aplicación.

***Min SDK Version***: Este valor especifica el mínimo nivel del API que requiere tu aplicación. Si el nivel del API introducido coincide con el proporcionado por uno de los targets disponibles, entonces se seleccionará automáticamente *Build Target*. Si una aplicación requiere un nivel del API superior al nivel soportado por el dispositivo, entonces la aplicación no será instalada.

***Otros campos***: La casilla de verificación *Use default location* permite cambiar la ruta del disco donde se crean y almacenan los ficheros de los proyectos.

Podremos ejecutar sobre la plataforma Android 1.5, aunque el *Build Target* seleccionado sea la plataforma 1.1. Estos no tienen que coincidir ya que una aplicación construida con la librería de la plataforma 1.1 se ejecutará normalmente en plataformas superiores a esta, pero no al revés.

Deberías tener visible el explorador de paquetes a la izquierda. Abre el fichero HolaMundo.java (situado en *HolaMundo* > *src* > *com.example.HolaMundo*). Debe tener este aspecto:

```
package com.example.holamundo;

import android.app.Activity;
import android.os.Bundle;

public class HolaMundo extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

Comenzamos declarando el paquete con el espacio de nombres utilizado por nuestra aplicación. A continuación indicamos los "import" con las clases que vamos a utilizar. Esto nos evitará tener que poner delante de cada clase su espacio de nombres.

Observe que la clase `HolaMundo` extiende de `Activity`. Una *actividad* es una entidad de aplicación que se utiliza para representar una pantalla de nuestra aplicación. Más adelante crearemos más actividades, pero el usuario interactúa con una de ellas cada vez. El método `onCreate()` se llamará por el sistema Android cuando se inicie su ejecución –es donde se debe realizar la inicialización y

la configuración de la interfaz del usuario. Una actividad no precisa tener una interfaz de usuario, pero la suele tener.

Antes de este método se ha utilizado la anotación `@Override` (sobrescribir). Esto indica al compilador que el método ya existe en la clase padre y queremos remplazarlo. Es opcional, aunque conviene incluirlo para evitar errores.

Lo primero que hay que hacer al sobrescribir un método suele ser llamar al método de la clase de la que hemos heredado. Esto se hace con la palabra reservada `super`.

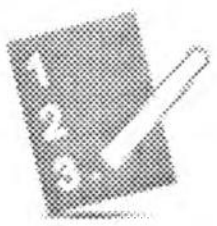
Terminamos indicando que la actividad que se va a crear se visualizará en una determinada vista. Esta vista está definida en los recursos.

## 1.8. Ejecución del programa

Una vez creada esta primera aplicación veamos dos alternativas para ejecutarla.

### 1.8.1. Ejecución en el emulador

Para ejecutar la aplicación sigue los siguientes pasos:

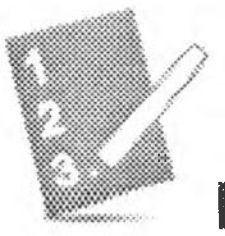


**Ejercicio paso a paso:** *Ejecución en el emulador.*

1. Selecciona *Run* > *Run* (Ctrl-F11).
2. Escoge *Android Application*. El ADT de Eclipse creará automáticamente una nueva configuración de ejecución para tu proyecto y lanzará el emulador (la inicialización del emulador puede ser algo lenta, por esta razón es mejor no cerrar el emulador). Una vez que el emulador esté cargado, debes ver algo parecido a la siguiente figura.

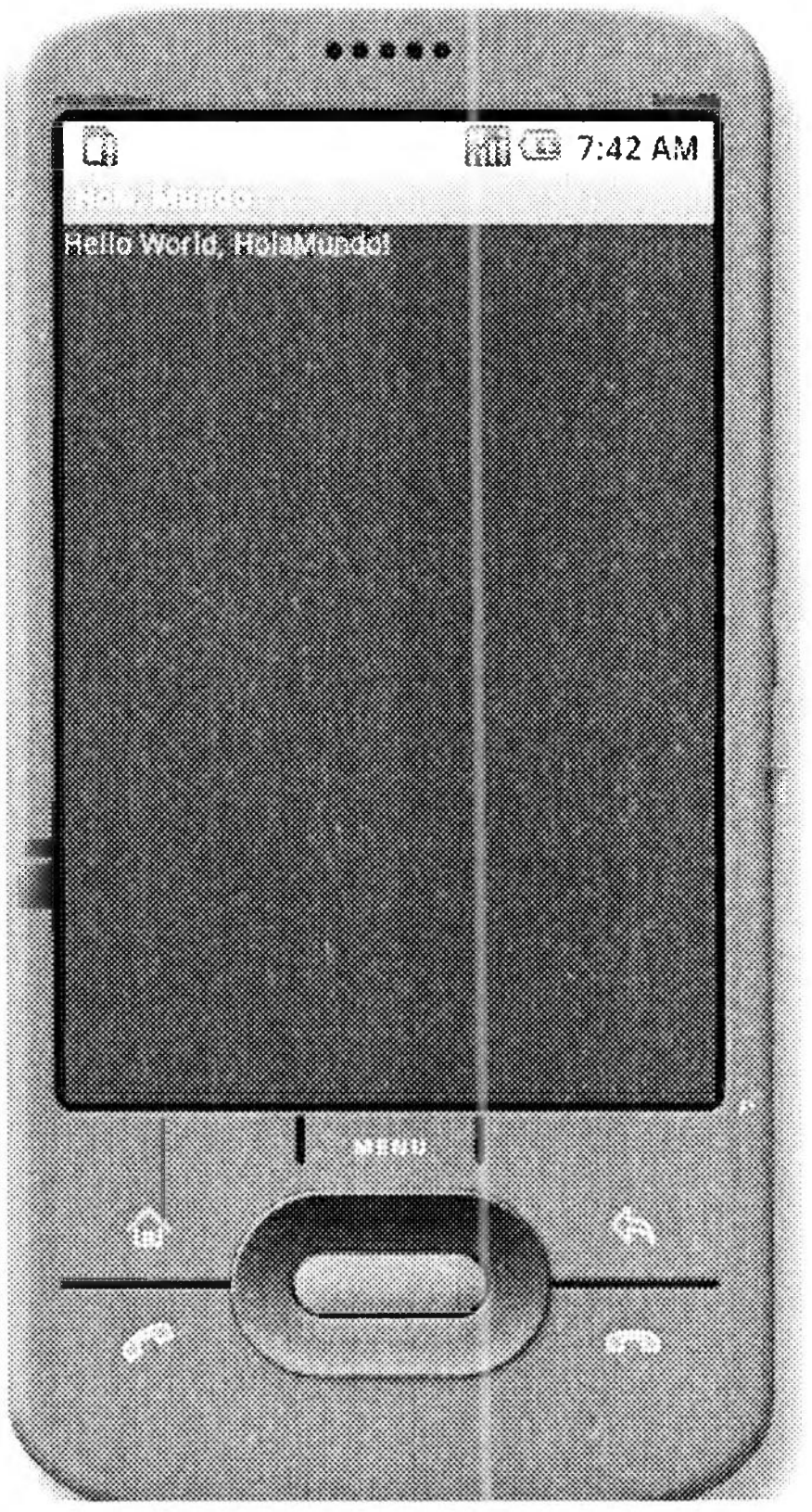
### 1.8.2. Ejecución en un terminal real

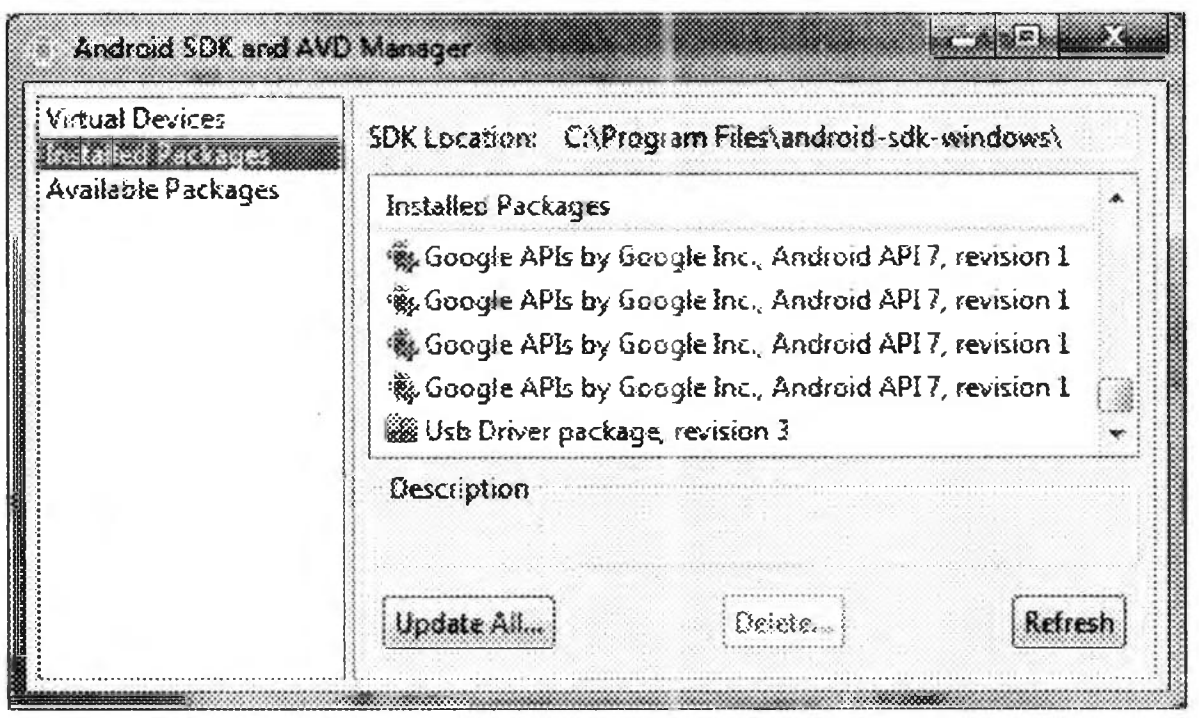
Ejecutar y depurar tus programas en un terminal real resulta posible desde el SDK. No tienes más que usar un cable USB para conectarlo al PC. Resulta imprescindible haber i**nstalado un *driver* especial en el PC. Puedes encontrar un *driver* genérico** en la carpeta *android-sdk-windows\usb_driver* del SDK Android. Aunque lo más probable es que tengas que utilizar el driver del fabricante.



**Ejercicio paso a paso:** *Ejecución en un terminal real.*

1. Abre *Android SDK and AVD Manager* y asegúrate que está instalado el paquete *USB Driver*. En caso contrario pulsa en *Available Packages* e instálalo.

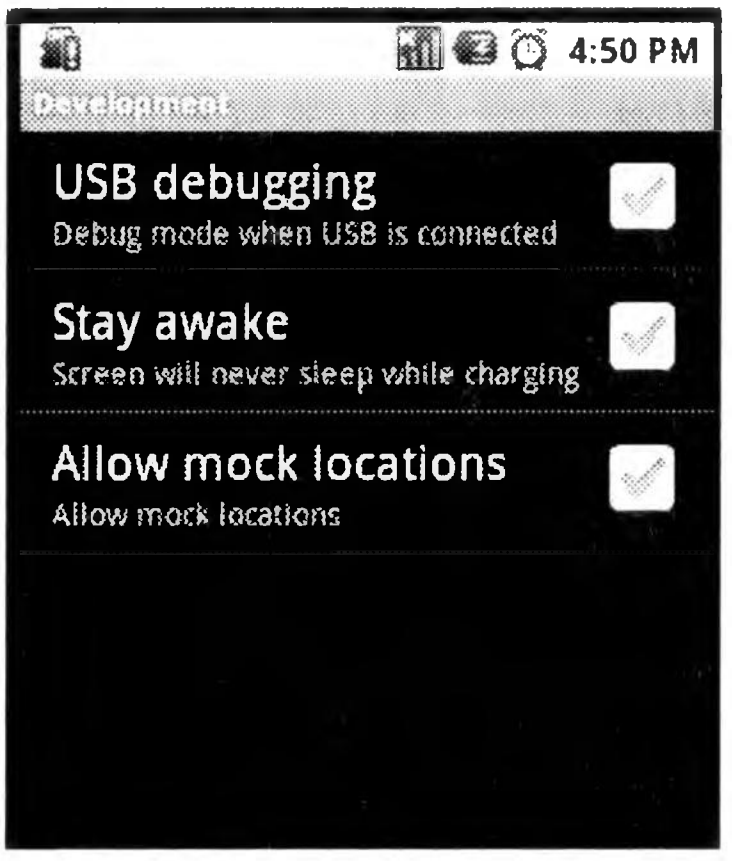



Posiblemente este driver genérico no sea adecuado para tu terminal y tengas que utilizar el del fabricante. Si no dispones de él, puedes buscarlo en:

http://developer.android.com/sdk/oem-usb.html

2. En el terminal accede al menú *Settings* > *Aplications* > *Development* y asegúrate que la opción *USB debugging* está activada.




3. Conecta el cable USB.
4. Se indicará que hay un nuevo *hardware* y te pedirá que le indiques el controlador.

***NOTA:*** *En Windows si indicas un controlador incorrecto no funcionará. Además la próxima vez que conectes el cable no te pedirá la instalación del controlador. Para desinstalar el controlador sigue los siguientes pasos:*

1. *Desinstala el controlador.*
2. *Accede al registro del sistema (Inicio > Ejecutar > RegEdit). Busca la siguiente clave y bórrala: "vid_0bb4&pid_0c02".*
3. *Vuelve al paso 3 del ejercicio.*

## 1.9. Elementos de un proyecto Android

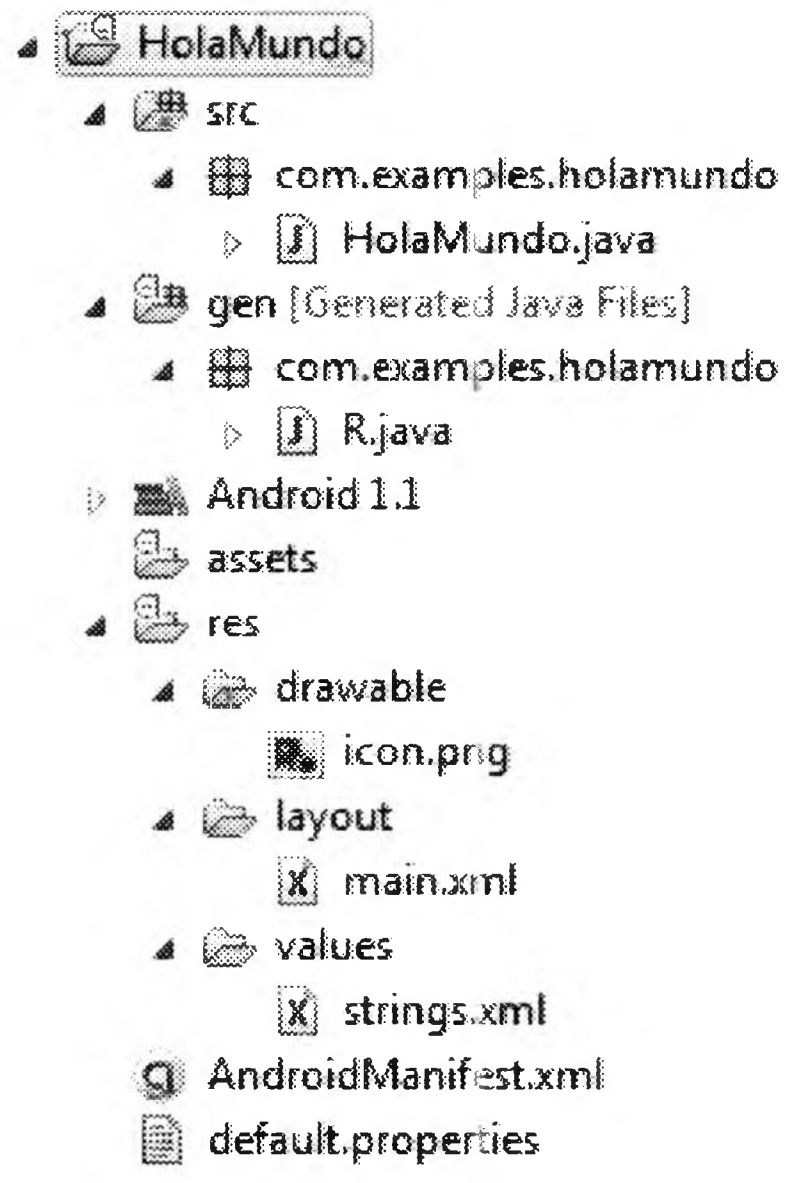
Poli[Media]: *Elementos de un proyecto Android.*

Un proyecto Android está formado básicamente por un descriptor de la aplicación (*AndroidManifest.xml*), el código fuente y una serie de ficheros con recursos. Cada elemento se almacena en una carpeta específica. Aprovecharemos el proyecto que acabamos de crear para estudiar la estructura de un proyecto Android:

***src***: Carpeta que contiene el código fuente de la aplicación. Como puedes observar los ficheros Java se almacenan en un espacio de nombres.

***gen***: Carpeta que contiene el código generado de forma automática por el SDK. Nunca hay que modificar de forma manual estos ficheros. Dentro encontraremos el siguiente fichero:

***R.java***: Define una clase que asocia los recursos de la aplicación con identificadores. De esta forma los recursos podrán ser accedidos desde Java.




***Android x.x***: Código JAR, el API de Android según la versión seleccionada.

***assets***: Carpeta que puede contener una serie arbitraria de ficheros o carpetas que podrán ser utilizados por la aplicación (ficheros de datos, ficheros JAR externos, fuentes, etc.). A diferencia de la carpeta *res,* nunca se modifica el contenido de los ficheros de esta carpeta.

***res***: Carpeta que contiene los recursos usados por la aplicación.

***drawable***: En esta carpeta se almacenan los ficheros de imágenes y descriptores de imágenes.

***layout***: Contiene ficheros XML con vistas de la aplicación. Las vistas nos permitirán configurar las diferentes pantallas que compondrán la interfaz de usuario de la aplicación. Serán tratadas en el siguiente capítulo.

***menu***: Ficheros XML con los menús de la aplicación.

***values***: También utilizaremos ficheros XML para indicar valores del tipo string, color o estilo. De esta manera podremos cambiar los valores sin necesidad de ir al código fuente. Por ejemplo, nos permitiría traducir una aplicación a otro idioma

***anim***: Contiene ficheros XML con descripciones de animaciones.

***xml***: Otros ficheros XML requeridos por la aplicación.

***raw***: Ficheros adicionales que no se encuentran en formato XML.

***doc***: Documentación asociada al proyecto.

***AndroidManifest.xml***: Este fichero describe la aplicación Android. En él se indican las *actividades*, *intenciones*, *servicios* y *proveedores de contenido* de la aplicación. También se declaran los permisos que requerirá la aplicación. Se indica la versión mínima de Android para poder ejecutarla.

***default.properties***: Fichero generado automáticamente por el SDK. Nunca hay que modificarlo. Se utiliza para comprobar la versión del API y otras características cuando se instala la aplicación en el terminal.

> ***NOTA:*** *Como se acaba de ver, el uso de XML es ampliamente utilizado en las aplicaciones Android. También es conocido el excesivo uso de espacio que supone utilizar este formato para almacenar información. Esto parece contradecir la filosofía de Android que intenta optimizar al máximo los recursos. Para solucionar el problema, los ficheros XML son compilados a un formato más eficiente antes de ser transferidos al terminal móvil.*

## 1.10. Componentes de una aplicación

Existen una serie de elementos clave que resultan imprescindibles para desarrollar aplicaciones en Android. En este apartado vamos a realizar una descripción inicial de algunos de los más importantes. A lo largo del libro se describirán con más detalle las clases Java que implementan cada uno de estos componentes.

### 1.10.1. Vista (*View*)

Las *vistas* son los elementos que componen la interfaz de usuario de una aplicación. Son por ejemplo, un botón, una entrada de texto,... Todas las vistas van a ser objetos descendientes de la clase *View*, y por tanto, pueden ser definidos utilizando código Java. Sin embargo, lo habitual va a ser definir las vistas utilizando un fichero XML y dejar que el sistema cree los objetos por nosotros a partir de este fichero. Esta forma de trabajar es muy similar a la definición de una página web utilizando código HTML.

### 1.10.2. Layout

Un *Layout* es un conjunto de vistas agrupadas de una determinada forma. Vamos a disponer de diferentes tipos de Layouts para organizar las vistas de forma lineal, en cuadrícula o indicando la posición absoluta de cada vista. Los Layouts también son objetos descendientes de la clase *View*. Igual que las vistas, los Layouts pueden ser definidos en código, aunque la forma habitual de definirlos es utilizando código XML.

### 1.10.3. Actividad (*Activity*)

Una aplicación en Android va a estar formada por un conjunto de elementos básicos de visualización, coloquialmente conocidos como pantallas de la aplicación. En Android cada uno de estos elementos, o pantallas, se conoce como *actividad*. Su función principal es la creación del interfaz de usuario. Una aplicación suele necesitar varias *actividades* para crear el interfaz de usuario. Las diferentes *actividades* creadas serán independientes entre sí, aunque todas trabajarán para un objetivo común. Toda actividad ha de pertenecer a una clase descendiente de *Activity*.

### 1.10.4. Servicio (*Service*)

Un *servicio* es un proceso que se ejecuta "detrás", sin la necesidad de una interacción con el usuario. Es algo parecido a un *demonio* en Unix o a un *servicio* en Windows. En Android disponemos de dos tipos de servicios: servicios locales, que pueden ser utilizados por aplicaciones del mismo terminal y servicios remotos, que pueden ser utilizados desde otros terminales. Los servicios son estudiados en el capítulo 8.

### 1.10.5. Intención (*Intent*)

Una *intención* representa la voluntad de realizar alguna acción; como realizar una llamada de teléfono, visualizar una página web. Se utiliza cada vez que queramos:

- Lanzar una *actividad*.
- Lanzar un *servicio*.
- Lanzar un *anuncio de tipo broadcast*.
- Comunicarnos con un *servicio*

Los componentes lanzados pueden ser internos o externos a nuestra aplicación. También utilizaremos las *intenciones* para el intercambio de información entre estos componentes.

En muchas ocasiones una *intención* no será inicializada por la aplicación, si no por el sistema, por ejemplo, cuando pedimos visualizar una página web. En otras ocasiones será necesario que la aplicación inicialice su propia *intención*. Para ello se creará un objeto de la clase *Intent*.

### 1.10.6. Receptor de anuncios *(Broadcast receiver)*

Un *receptor de anuncios* recibe y reacciona ante anuncios de tipo *broadcast*. Existen muchos originados por el sistema, como por ejemplo *Batería baja, Llamada entrante,...* Aunque, las aplicaciones también puede lanzar un *anuncio broadcast*. No tienen interfaz de usuario, aunque pueden iniciar una actividad para atender a un anuncio.

### 1.10.7. Proveedores de contenido (*Content Provider*)

La compartición de información entre teléfonos móviles resulta un tema vital. Android define un mecanismo estándar para que las aplicaciones puedan compartir datos sin necesidad de comprometer la seguridad del sistema de ficheros. Con este mecanismo podremos acceder a datos de otras aplicaciones, como la lista de contactos, o proporcionar datos a otras aplicaciones. Los *Content Provider* son estudiados en el capítulo 9.

## 1.11. Documentación y ApiDemos

Aunque en este libro vas a aprender mucho, resultaría imposible tocar todos los aspectos de Android a un elevado nivel de profundidad. Por lo tanto, resulta imprescindible que dispongas de fuentes de información para consultar los aspectos que vayas necesitando. En este apartado te proponemos dos alternativas: el acceso a documentación sobre Android y el estudio de ejemplos.

### 1.11.1. Donde encontrar documentación

Puedes encontrar una completa documentación del SDK de Android, incluyendo referencia de las clases, conceptos clave y otro tipo de recursos en (siempre que hayas instalado la documentación on-line):

```
…\android-sdk-windows\docs\index.html
```

Esta documentación también está disponible en línea a través de Internet:

http://developer.android.com/

Muchos de los recursos utilizados en este libro puedes encontrarlos en:

http://www.androidcurso.com/

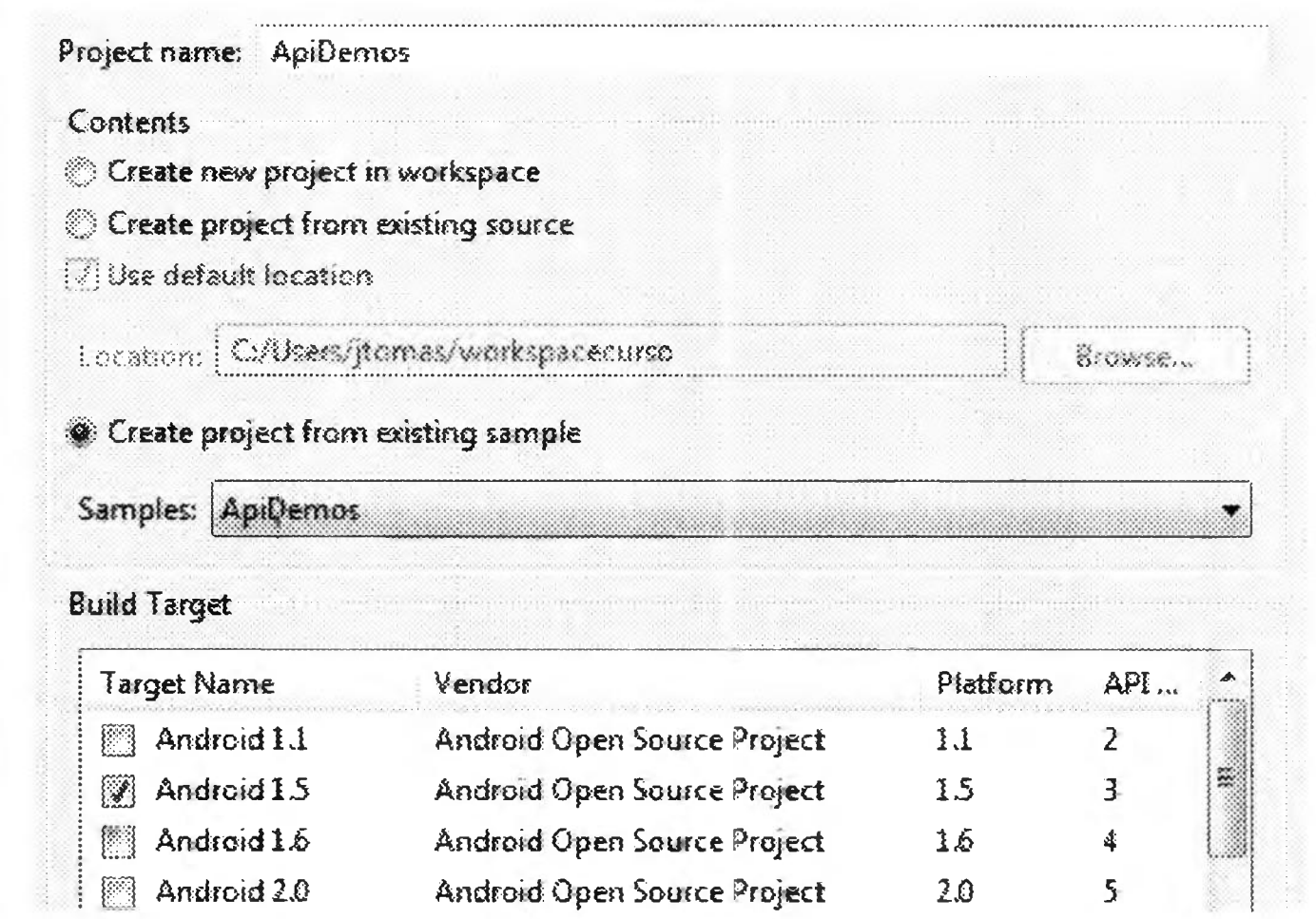
Puedes encontrar información referente a este libro en la Web de la editorial:

http://libroweb.alfaomega.com.mx

### 1.11.2. La aplicación ApiDemos

Otra opción muy interesante para aprender nuevos aspectos de programación consiste en estudiar ejemplos. Con este propósito se ha instalado en el SDK de Android una aplicación, ApiDemos, formada por cientos de ejemplos, donde no solo podrás ver las funcionalidades disponibles en el API de Android, sino que además podrás estudiar su código.

Para crear un proyecto con esta aplicación sigue los siguientes pasos: Selecciona *File* > *New* > *Project* e indica *Android Project*. Pulsa *Next* y aparecerá un cuadro de diálogo similar al siguiente:

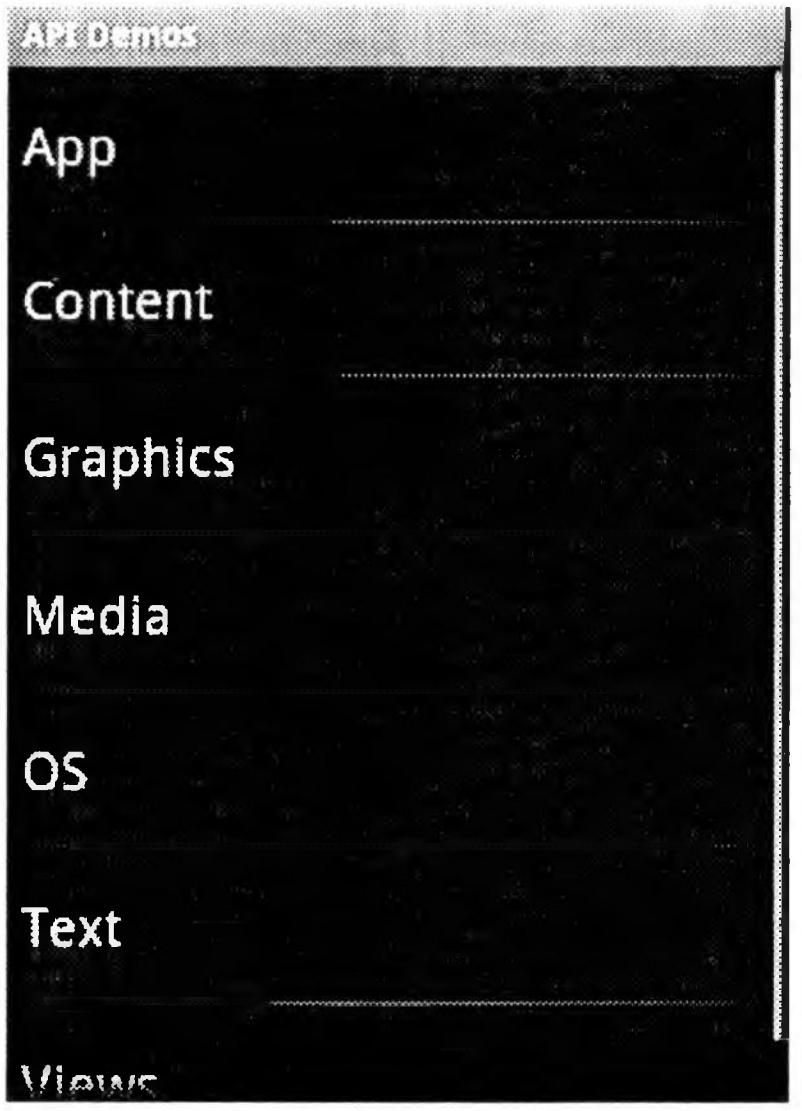


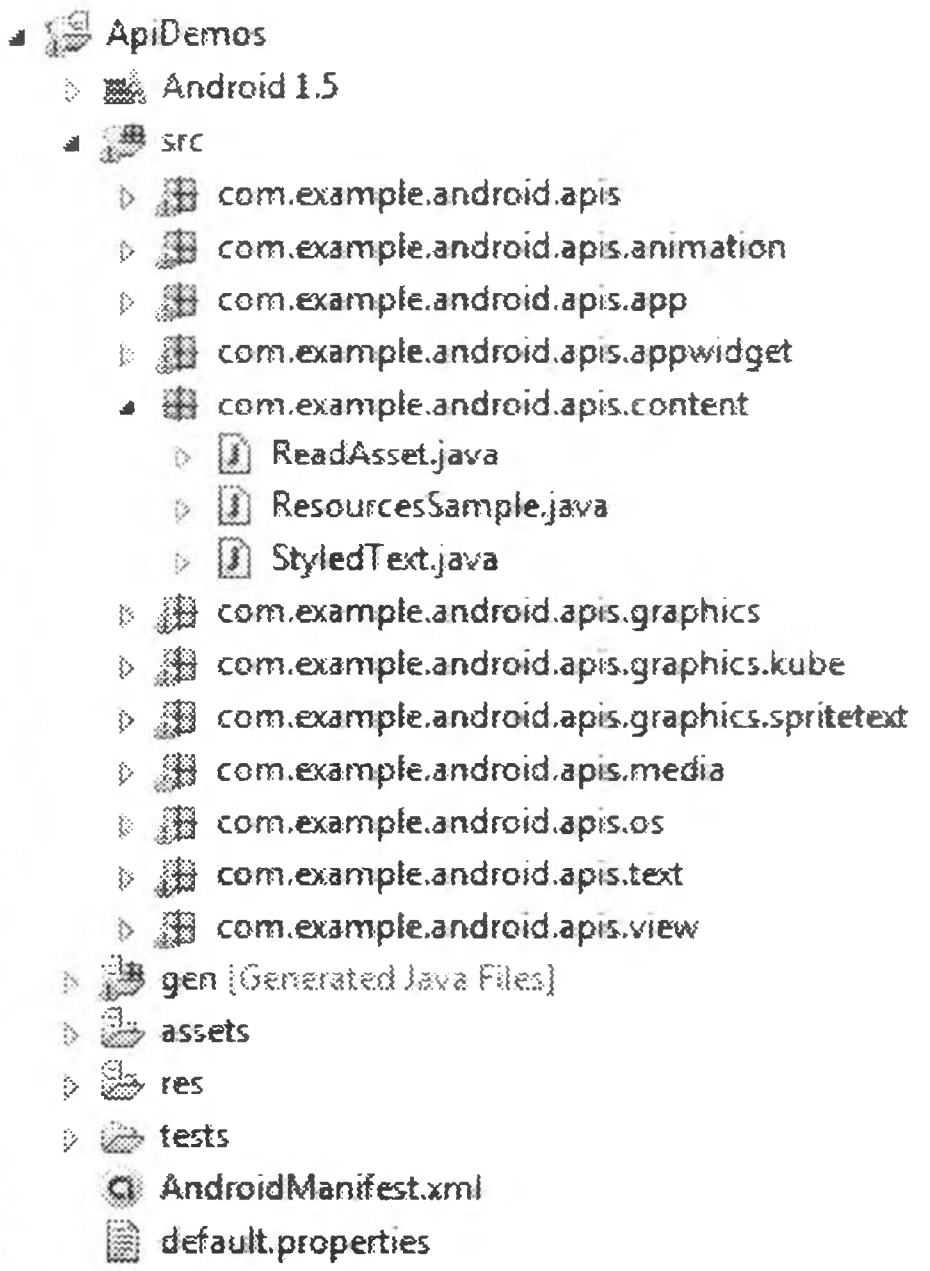

Has de marcar la opción *Create from existing sample* y seleccionar un nivel de API. El número de ejemplos aumenta a medida que aumentamos el nivel de API. A continuación selecciona uno de los ejemplos en *Samples:* en nuestro caso *ApiDemos*. Pulsa *Finish* para crear el proyecto.

Si no ha habido ningún error ya puedes ejecutar *ApiDemos*. Verás como los diferentes ejemplos se organizan por carpetas. En el nivel superior tenemos:

1. **App**: Ejemplos a nivel de aplicación trabajando con las clases *Activity*, *Alarm*, *Dialog* y *Service*.
2. **Content**: Describe cómo leer datos desde ficheros, recursos y archivos XML.
3. **Graphics**: Gran cantidad de ejemplos gráficos, tanto en 2D como en 3D.
4. **Media**: Reproducción de audio y vídeo con las clases *MediaPlayer* y *VideoView*.
5. **OS**: Ejemplos de cómo utilizar diferentes servicios del sistema operativo, como *VIBRATOR_SERVICE* y *SENSOR_SERVICE*.
6. **Text**: Diferentes ejemplos de manipulación de texto. Nos muestra, por ejemplo, como conseguir que parte de un texto se convierta en un *link,* que habrá automáticamente una URL o una llamada de teléfono.
7. **Views**: Android utiliza como elemento básico de representación la clase *View* (vista). Tenemos a nuestra disposición gran cantidad de descendientes de esta clase para representar una interfaz gráfica (botones, cajas de texto, entradas,...). Visualiza estos ejemplos para comprobar las diferentes posibilidades.

Una vez seleccionado un ejemplo, si quieres estudiar su código, búscalo en la carpeta *src*. Para organizar mejor el código se han creado diferentes espacios de nombre de paquetes.




*__NOTA:__ Si cuando ejecutas la aplicación no te deja y te indica el siguiente error:*

```
[2011-12-07 13:42:04 - ApiDemos] Installing ApiDemos.apk...
[2011-12-07 13:42:14 - ApiDemos] Re-installation failed due to different application signatures.
[2011-12-07 13:42:14 - ApiDemos] You must perform a full uninstall of the application. WARNING: This will rem
[2011-12-07 13:42:14 - ApiDemos] Please execute 'adb uninstall com.example.android.apis' in a shell.
[2011-12-07 13:42:14 - ApiDemos] Launch canceled!
```

*Es porque ya existe una aplicación con el mismo nombre de paquete (com.example.android.apis). Además, ha detectado que la aplicación ha sido firmada con un certificado digital diferente al que estás utilizando.*

*Se soluciona desinstalando antes la aplicación. Tienes dos opciones:*

- *Como lo hacemos con nuestro teléfono: Ajustes/Aplicaciones/Administar Aplicaciones/ApiDemos/Desinstalar.*
- *Desde la línea de comando:* `adb uninstall com.example.android.apis`.

*Este problema aparece cuando se ejecuta ApiDemos en un dispositivo donde ya está instalado. También si hemos ejecutado una aplicación en nuestro teléfono y cambiamos de entorno de desarrollo. El nuevo entorno utiliza un certificado digital diferente por lo que no permite remplazarlo.*

# 1.12. Depurar

## 1.12.1. Depurar con Eclipse

El *plug-in* de Android para Eclipse tiene una excelente integración con el depurador de Eclipse. Introduce un error en tu código modificando el código fuente de `HolaMundo` para que tenga este aspecto:

```
public class HolaMundo extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        Object o = null;
        o.toString();
        setContentView(R.layout.main);
    }
}
```

Este cambio introduce un *NullPointerException* en tu código. Si ahora ejecutas tu aplicación, te aparecerá esto:


Pulsa "*Force close*" para finalizar la aplicación y cerrar la ventana del emulador. Para averiguar más sobre el error, inserta un punto de ruptura (*breakpoint*) en el código fuente en la línea *Object o = null;* (el *breakpoint* se introduce haciendo doble clic en la barra de la izquierda). Entonces selecciona *Run / Debug History / HolaMundo* para entrar en modo *debug*. Tu aplicación se reiniciará en el emulador, pero esta vez quedará suspendida cuando alcance el punto de ruptura que se ha introducido. Entonces puedes recorrer el código en modo *Debug*, igual que se haría en cualquier otro entorno de programación.

## 1.12.2. Depurar con mensajes *Log*

La clase *Log* proporciona un mecanismo para depurar nuestros programas o para verificar el funcionamiento de los objetos que estamos utilizando. Disponemos de varios tipos de mensajes tal y como se muestra a continuación:

`Log.e()`: Errors.
`Log.w()`: Warnings.
`Log.i()`: Information.
`Log.d()`: Debugging.
`Log.v()`: Verbose.

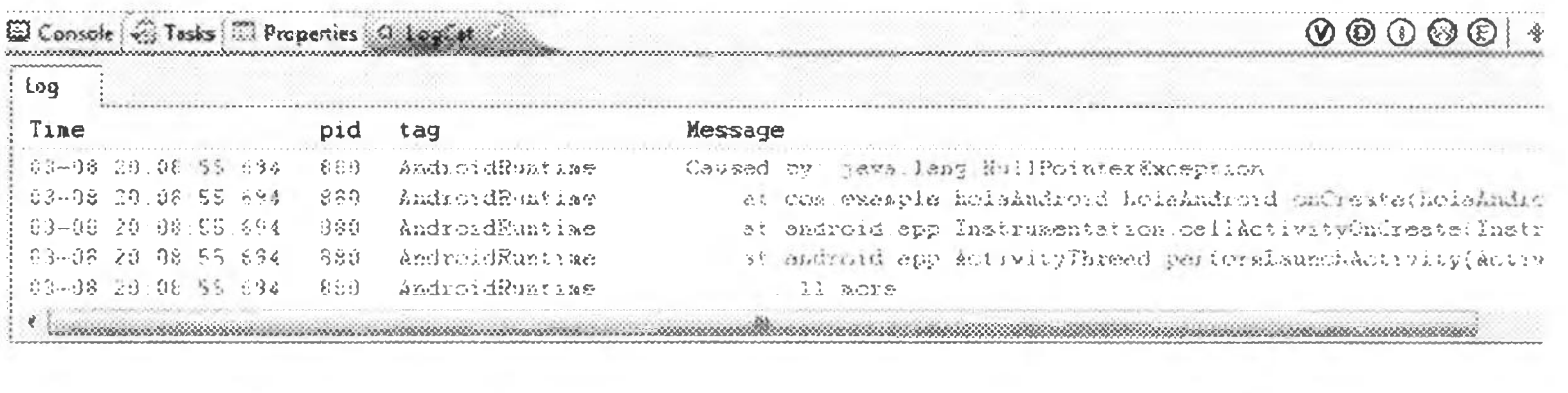
**Ejercicio paso a paso:** *Depurar con mensajes Log.*

1. Modifica la clase `HolaMundo` introduciendo la línea que aparece en negrita:

```
public class HolaMundo extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
       Log.d("HolaMundo","Entramos en onCreate");
       super.onCreate(savedInstanceState);
        Object o = null;
        o.toString();
        setContentView(R.layout.main);
    }
}
```

2. Ejecuta la aplicación. Aparecerá un error.

   Para ver el fichero de *Log* desde Eclipse accede al menú *Window / Show View / Others... / Android / LogCat*.




*LogCat* puede utilizarse para detectar errores o problemas en el código. Por ejemplo, si no tenemos muy claro qué parte del código causó el error del ejemplo anterior, podemos buscar el último error y ver cuál fue la llamada de nuestro código que lo generó. En el siguiente cuadro se muestra como el error lo ha ocasionado "holaAndroid.java:10".

# CAPÍTULO 2.

# Diseño de la interfaz de usuario: Vistas y Layouts

El diseño de la interfaz de usuario cobra cada día más importancia en el desarrollo de una aplicación. La calidad de la interfaz de usuario puede ser uno de los factores que conduzca el proyecto al éxito o al fracaso.

Si has realizado alguna aplicación utilizando otras plataformas, advertirás que el diseño de la interfaz de usuario en Android sigue una filosofía muy diferente. En Android la interfaz de usuario no se diseña en código, sino utilizando un lenguaje de marcado similar al HTML.

A lo largo de este capítulo mostraremos una serie de ejemplos que te permitirán entender el diseño de la interfaz de usuario en Android. Aunque no será la forma habitual de trabajar, comenzaremos creando el interfaz de usuario mediante código. De esta forma comprobaremos como cada uno de los elementos del interfaz de usuario (vistas) realmente son objetos Java. Continuaremos mostrando cómo se define el interfaz de usuario utilizando código XML. Pasaremos luego a ver las herramientas de diseño integradas en Eclipse. Se describirá el uso de *Layouts* que nos permitirá una correcta organización de las vistas y como el uso de recursos alternativos nos permitirá adaptar nuestra interfaz a diferentes circunstancias y tipos de dispositivos. En este capítulo también comenzaremos creando la aplicación de ejemplo desarrollada a lo largo del curso, Asteroides. Crearemos la actividad principal, donde simplemente mostraremos cuatro botones, uno de los cuales arrancará una segunda actividad para mostrar una caja Acerca de... A continuación, aprenderemos a crear estilos y temas y los aplicaremos a estas actividades. Para terminar el capítulo, propondremos varias prácticas para aprender a utilizar diferentes tipos de vistas y Layouts.

**Objetivos:**

- Describir cómo se realiza el diseño del interfaz de usuario en una aplicación Android.

- Aprender a trabajar con vistas y mostrar sus atributos más importantes.
- Enumerar los tipos de Layouts que nos permitirán organizar las vistas.
- Mostrar cómo se utilizan los recursos alternativos.
- Aprenderemos a crear estilos y temas para personalizar nuestras aplicaciones.
- Mostrar cómo interactuar con las vistas desde el código Java.
- Describir el uso de TabLayout.

## 2.1. Creación de una interfaz de usuario por código

Veamos un primer ejemplo de cómo crear una interfaz de usuario utilizando exclusivamente código. Esta no es la forma recomendable de trabajar con Android, sin embargo resulta interesante para discutir algunos conceptos.

**Ejercicio paso a paso**: *Creación del interfaz de usuario por código.*

1. Abre el proyecto creado en el capítulo anterior y visualiza *HolaMundo.java*.
2. Comenta la última sentencia del programa y añade las tres que se muestran a continuación en negrita:

```
package com.example.holamundo;

import android.app.Activity;
import android.os.Bundle;

public class HolaMundo extends Activity {
    /** Se llama cuando se crea la actividad por primera vez. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //setContentView(R.layout.main);
        TextView texto = new TextView(this);
        texto.setText("Hello, Android");
        setContentView(texto);
    }
}
```

**Nota sobre Java:** *Para poder utilizar el objeto* `TexView` *has de importar un nuevo paquete, para ello añade al principio* `import android.widget.TextView;`. *Otra alternativa es pulsar* ***Ctrl-Shift-O****, para que se añadan automáticamente los paquetes que faltan.*

La interfaz de usuario de Android está basada en una jerarquía de objetos descendientes de la clase `View` (vista). Una vista es un objeto que se puede dibujar y se utiliza como un elemento en el diseño de la interfaz de usuario (un botón, una imagen, una etiqueta de texto como el utilizado en el ejemplo,...). Cada uno de estos elementos se define como una subclase de la clase `View`; la subclase para representar un texto es `TextView`.

El ejemplo comienza creando un objeto de la clase `TextView`. El constructor de la clase acepta como parámetro una instancia de la clase `Context` (contexto). Un contexto es un manejador del sistema que proporciona servicios como la resolución de recursos, obtención de acceso a bases de datos o preferencias. La clase `Activity` es una subclase de `Context`, y como la clase `HelloAndroid` es una subclase de `Activity`, también es tipo `Context`. Por ello, puedes pasar `this` (el objeto actual de la clase `HolaMundo`) como contexto del `TextView`.

Después se define el contenido del texto que se visualizará en el `TextView` mediante `setText(CharSequence)`. Finalmente, mediante `setContentView()` se indica la vista utilizada por la actividad.

3. Ejecuta el proyecto para verificar que funciona.

## 2.2. Creación de una interfaz de usuario usando XML

En el ejemplo anterior hemos creado la interfaz de usuario directamente en el código. A veces puede ser muy complicado programar interfaces de usuario, ya que pequeños cambios en el diseño pueden corresponder a complicadas modificaciones en el código. Un principio importante en el diseño de *software* es que conviene separar todo lo posible el diseño, los datos y la lógica de la aplicación.

Android proporciona una alternativa para el diseño de interfaces de usuario: los ficheros de diseño basados en XML. Veamos uno de estos ficheros. Para ello accede al fichero *res/layout/main.xml* de nuestro proyecto. Se muestra a continuación. Este *layout* o fichero de diseño proporciona un resultado similar al del ejemplo de diseño por código anterior:

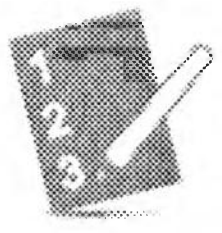
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/hello"/>
</LinearLayout>
```

Resulta sencillo interpretar su significado. Se introduce un elemento de tipo `LinearLayout`; como se estudiará más adelante su función es contener otros elementos de tipo `View`. Este `LinearLayout` tiene cuatro atributos. El primero, `xmlns:android`, es una declaración de un espacio de nombres de XML que utilizaremos en Android (este parámetro solo es necesario especificarlo en el primer elemento). El siguiente atributo indica que los elementos se van a distribuir de forma vertical. Los dos siguientes permiten definir el ancho y alto de la vista. En el ejemplo se ocupará todo el espacio disponible.

Dentro del `LinearLayout` solo tenemos un elemento de tipo `TextView`. Este dispone de tres atributos. Los dos primeros definen el ancho y alto. El último indica el texto a mostrar. No se ha indicado un texto en concreto si no una referencia, `"@string/hello"`. Esta referencia ha de estar definida en el fichero *res/values/strings.xml*. Si abres este fichero, tienes el siguiente contenido:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Hello World, HolaMundo!</string>
    <string name="app_name">Hola, Mundo</string>
</resources>
```

Esta es la práctica recomendada en Android para la inserción de textos en tu aplicación, dado que facilita su localización a la hora de realizar la traducción a otros idiomas. Es decir, utilizaremos los ficheros *layout* para introducir el diseño de los interfaces y el fichero *strings* para introducir los textos utilizados en los distintos idiomas.



**Ejercicio paso a paso:** *Creación del interfaz de usuario con XML.*

1. Para utilizar el diseño en XML regresa al fichero *HolaMundo.java* y deshaz los cambios que hicimos antes (elimina las tres últimas líneas y quita el comentario).
2. Ejecuta la aplicación y verifica el resultado. Ha de ser muy similar al anterior.
3. Modifica algún valor del fichero *res/values/strings.xml*.
4. Vuelve a ejecutar la aplicación y visualiza el resultado.

   Analicemos ahora la línea en la que acabas de quitar el comentario:

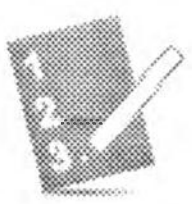
   ```
   setContentView(R.layout.main);
   ```

Aquí, `R.layout.main` corresponde a un objeto `View` que será creado en tiempo de ejecución a partir del recurso *main.xml*. Trabajar de esta forma, en comparación con el diseño basado en código, no quita velocidad y requiere menos espacio. El *plug-in* de Eclipse crea automáticamente esta referencia en la clase `R` del proyecto a partir de todos los elementos de la carpeta *res*. Abre el fichero llamado *gen/R.java*. Este fichero será similar al que se muestra a continuación:

```
package com.example.holamundo;

public final class R {
    public static final class attr {
    }
    public static final class drawable {
        public static final int icon=0x7f020000;
    }
    public static final class layout {
        public static final int main=0x7f030000;
    }
    public static final class string {
        public static final int app_name=0x7f040001;
        public static final int hello=0x7f040000;
    }
}
```

***NOTA:*** *Este fichero se genera automáticamente. Nunca debes editarlo de forma manual.*

Has de tener claro que las referencias de la clase `R` son meros números que informan al gestor de recursos, que datos ha de cargar. Por lo tanto no se trata de verdaderos objetos, estos serán creados en tiempo de ejecución solo cuando sea necesario usarlos.



**Ejercicio paso a paso:** *El fichero R.java.*

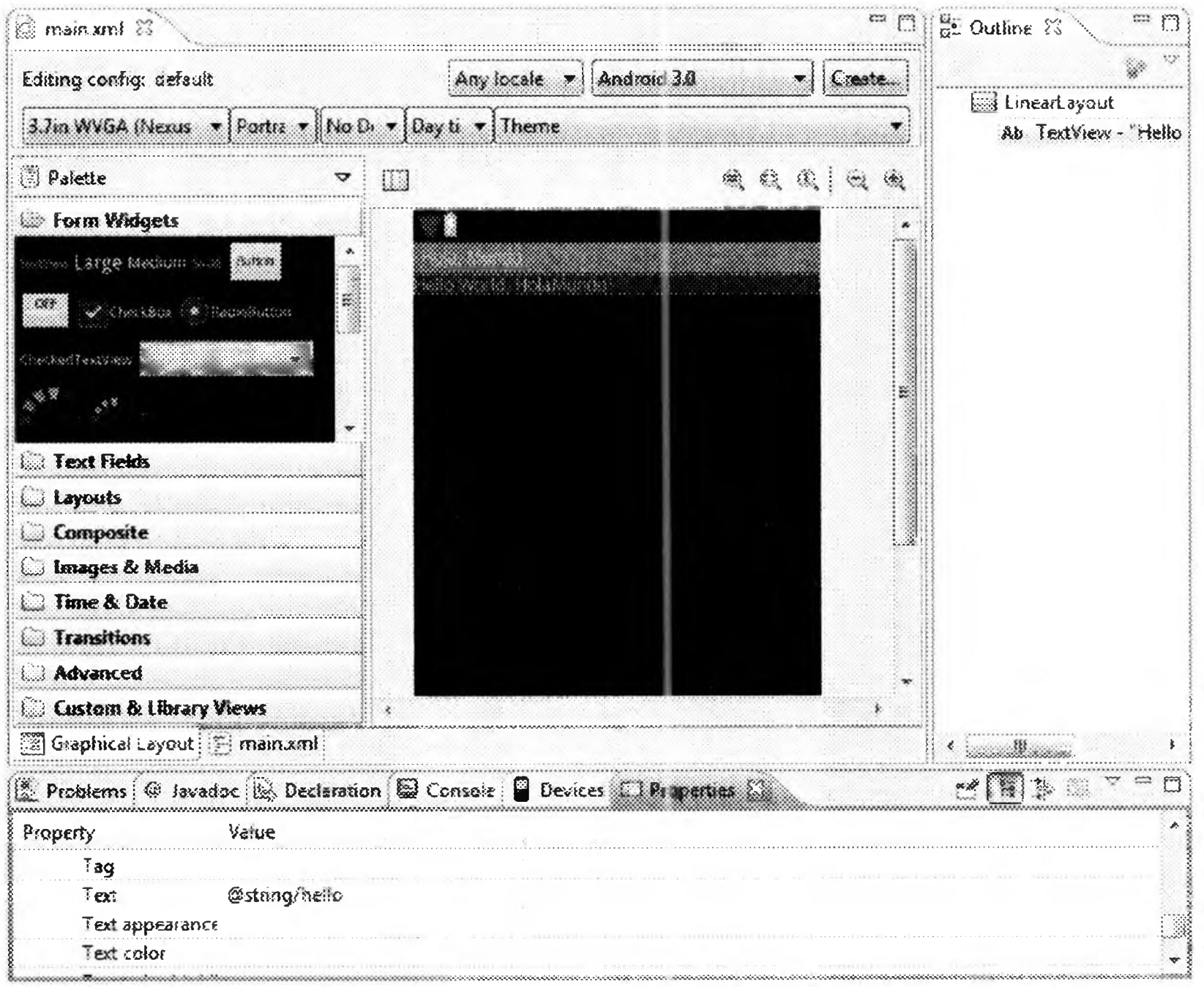
1. Abre el fichero llamado *gen/R.java* de tu aplicación.
2. Compáralo con el fichero mostrado previamente. ¿Qué diferencias encuentras?
3. Abre el fichero *HolaMundo.java* y reemplaza `R.layout.main` por el valor numérico al que corresponde en *R.java*.
4. Ejecuta de nuevo el proyecto. ¿Funciona? ¿Crees que sería adecuado dejar este valor numérico?
5. Este valor puede cambiar en un futuro. Por lo tanto, para evitar problemas futuros vuelve a remplazarlo por `R.layout.main`.

**Polli[Media]**: *Un primer proyecto Android.*

**Preguntas de repaso y reflexión:** *Diseño del interfaz de usuario y el fichero R.java.*

## 2.2.1. Edición visual de las vistas

Veamos ahora como editar los *layouts* o ficheros de diseño en XML. En el explorador de paquetes abre el fichero *res/layout/main.xml*. Verás que en la parte inferior de la ventana central aparecen dos lengüetas: *Layout* y *main.xml*. El *plug-in* de Android te permite dos tipos de diseño: editar directamente el código XML (lengüeta *main.xml*) o realizar este diseño de forma visual (lengüeta *Grafical Layout*). Veamos cómo se realizaría el diseño visual:

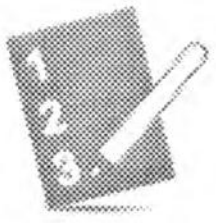



En el marco derecho se visualiza una lista con todos los elementos de la vista. En el anexo B se ha incluido un listado con las vistas disponibles. En este momento solo hay dos: un `LinearLayout` que contiene un `TextView`. En el marco central aparece una representación de cómo se verá el resultado. En la parte superior aparecen varios controles para representar esta vista en diferentes configuraciones. Cuando diseñamos una vista en Android, hay que tener en cuenta que desconocemos el dispositivo final donde será visualizada y la configuración específica elegida por el usuario. Por esta razón, resulta importante que verifiques que la vista se ve de forma adecuada en cualquier configuración. Podemos seleccionar la configuración regional (*locale*), la versión de Android utilizada, el tamaño en pulgadas y la resolución del dispositivo, orientación horizontal (*lanscape*) o vertical (*portrait*), configuración de utilización (escritorio o automóvil), configuración según la hora del día (diurna o nocturna) y cómo se verá nuestra vista tras aplicar un tema.

Para editar un elemento selecciónalo en el marco de la derecha y pincha sobre él en la ventana de previsualización. Al seleccionarlo puedes modificar alguna de

sus propiedades en el marco que aparece en la parte inferior. Si este marco no está visible, pulsa con el botón derecho sobre el elemento en el marco de la derecha y seleccionas *Properties*, aparecerá el marco en la parte inferior con la lista de propiedades. Para activar esta vista accede al menú W*indows* > *Show View* > *Others...* > *General* > *Properties.* Dele un vistazo a las propiedades disponibles para *TextView* y modifica alguna de ellas. En muchos casos te aparecerá un desplegable con las opciones disponibles.

El marco de la izquierda te permite insertar de forma rápida nuevos elementos a la vista. Puedes arrastrar cualquier elemento a la ventana de previsualización o al marco *Outline*.



**Ejercicio paso a paso:** *Creación visual de Vistas.*

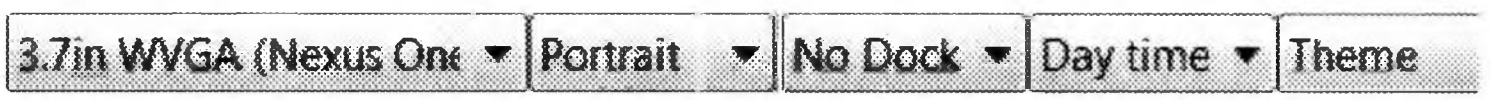
1. Crea un nuevo proyecto con nombre *PrimerasVistas*. Selecciona la versión 1.6 e introduce `org.example.primerasvistas` en *Package name*.
2. Abre el fichero *res/layout/main.xml*.
3. Desde la paleta de la izquierda arrastra los siguientes elementos: `TroggleButton`, `CheckBox`, `ProgressBar` y `RatingBar`.
4. Selecciona la primera vista que estaba ya creada (`TextView`) y pulsa el botón <Supr> para eliminarla.
5. Pulsa el primer botón (*Set Horizontal Orientation*) para conseguir que el `LinerLayout` donde están las diferentes vistas tenga una orientación horizontal. Comprobarás que no caben todos los elementos.
6. Pulsa el siguiente botón (*Set Vertical Orientation*) para volver a una orientación vertical.
7. Selecciona la vista `TroggleButton`. Pulsa el botón siguiente al que acabamos de utilizar (*Trodgle Fill Width*). Conseguirás que el ancho del botón se ajuste al ancho de su contenedor.
8. Pulsa el botón siguiente (*Trodgle Fill Height*). Conseguirás que el alto del botón se ajuste al alto de su contenedor. El problema es que el resto de elementos dejan de verse. Vuelve a pulsar este botón para regresar a la configuración anterior (también puedes pulsar *Ctrl-Z*).
9. Selecciona la vista *CheckBox*. Pulsa el botón siguiente al que acabamos de utilizar (*Change Margins...*). Conseguirás que el ancho del botón se ajuste al ancho de su contenedor. En la entrada *All* introduce "20px".
10. Pulsa el botón siguiente (*Change Gravity*) y selecciona *Center Horizontal*.
11. Observa como ha sar en la parte inferior del *Layout*.

Es equivalente a poner *Layout Weights* =1 para todas las vistas de este *Layout*. Esta propiedad será modificada en el siguiente punto.

12. Pulsa el botón siguiente (*Change Layout Weights*) e introduce el valor 2. Selecciona la vista `ToggleButton` y usando este mismo botón, introduce el valor 0.5. Selecciona la vista `ProgressBar` e introduce el valor 4. Como puedes observar estos pesos permiten ajustar la altura de las vistas.

13. Utiliza los siguientes botones  para ajustar el zoom.

14. Utiliza los botones de la barra superior para observar cómo se representará el Layout en diferentes circunstancias:



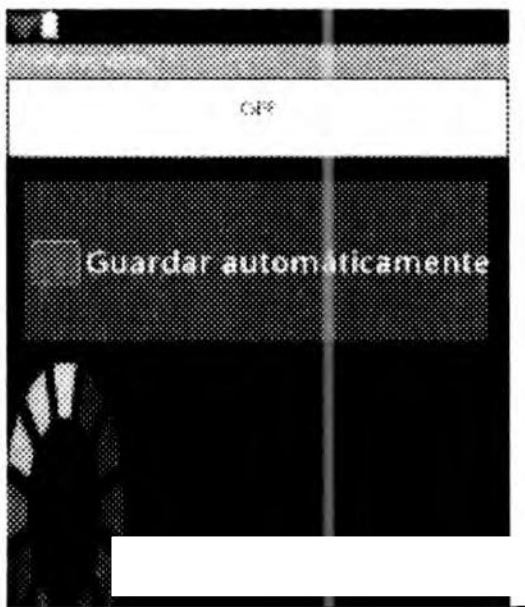

El primer desplegable permite escoger entre diferentes tamaños de pantalla / resolución. El siguiente visualiza en vista vertical (*Portrait*) u horizontal (*Landscape*). Los dos siguientes permiten definir diferentes modos de visualización (*Card Dock* significa que esta activado el modo automóvil y *Nigth time* el modo noche). Estos modos están disponibles a partir de la versión 2.2. El último desplegable permite ver cómo quedaría el *Layout* tras aplicar un tema.

15. Si no está visible la vista de Eclipse *Propiedades*, actívala utilizando el menú *Ventana/Mostrar Vista/Otras.../General/Propiedades*.

16. Selecciona la vista `CheckBox` y observa las diferentes propiedades que podemos definir. Algunas ya han sido definidas por medio de la barra de botones. En concreto, y siguiendo el mismo orden que en los botones, hemos modificado: *Layout width= wrap content, Layout height = wrap_content, Layout margin = 20px, Layout gravity = center_horizontal y Layout weights =2*.

17. Busca la propiedad *Text* y sustituye el valor "CheckBox" por "Guardar automáticamente" y *Text size* por "9pt".

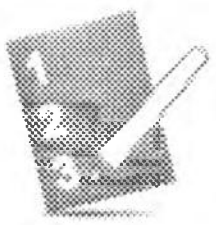
18. El resultado final obtenido se muestra a continuación:



19. Pulsa sobre la lengüeta de la parte inferior con nombre "*main.xml*". Pulsa las teclas *Shift-Ctrl-F* para que formatee adecuadamente el código XML. Este código se muestra a continuación.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:orientation="vertical">
   <ToggleButton android:text="ToggleButton"
      android:id="@+id/toggleButton1"
      android:layout_height="wrap_content"
      android:layout_width="fill_parent"
      android:layout_weight="0.5"/>
   <CheckBox android:layout_margin="20px"
      android:id="@+id/checkBox1"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_gravity="center_horizontal"
      android:layout_weight="2"
      android:text="Guardar automáticamente"
      android:textSize="9pt"/>
   <ProgressBar android:layout_width="wrap_content"
      style="?android:attr/progressBarStyleLarge"
      android:id="@+id/progressBar1"
      android:layout_height="wrap_content"
      android:layout_weight="4"/>
   <RatingBar android:id="@+id/ratingBar1"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_weight="0.5"/>
</LinearLayout>
```

20. Ejecuta el proyecto para ver el resultado.



**Ejercicio paso a paso:** *Vistas de entrada de texto.*

1. Añade en la parte superior del *Layout* anterior una vista de tipo entrada de texto `EditText`, de tipo normal (*Text Plain*). Debajo de esta una de tipo nombre y apellido (*Person Name*) seguida de una de tipo palabra secreta (*Password*). Continua así con otros tipos de entradas de texto.
2. Ejecuta la aplicación.
3. Observa, como al introducir el texto de una entrada se mostrará un tipo de teclado diferente.

**Recursos adicionales:** *Tipos de vista y sus atributos.*

Consulta el Anexo B para conocer una lista con todos los descendientes de la clase `View` y sus atributos.

**Preguntas de repaso y reflexión:** *Las vistas y sus atributos.*

## 2.3. Layouts

Si queremos combinar varios elementos de tipo vista tendremos que utilizar un objeto de tipo *Layout*. Un *Layout* es un contenedor de una o más vistas y controla su comportamiento y posición. Hay que destacar que un *Layout* puede contener a otro *Layout* y que es un descendiente de la clase *View*. La siguiente lista describe los *Layout* más utilizados en Android:

***LinearLayout***: Dispone los elementos en una fila o en una columna.

***TableLayout***: Distribuye los elementos de forma tabular.

***RelativeLayout***: Dispone los elementos en relación a otro o al padre.

***AbsoluteLayout***: Posiciona los elementos de forma absoluta.

***FrameLayout***: Permite el cambio dinámico de los elementos que contiene.

Dado que un ejemplo vale más que mil palabras, pasemos a mostrar cada uno de estos *layouts* en acción.

*LinearLayout* es el *layout* más utilizado en la práctica. Distribuye los elementos uno detrás de otro, bien de forma horizontal o vertical.

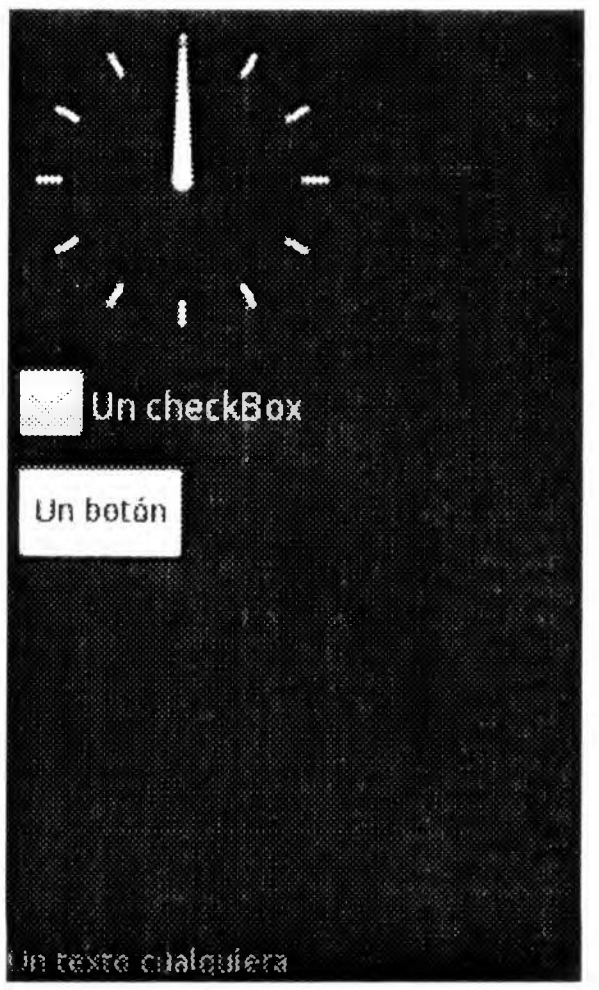
```
<LinearLayout  xmlns:android="http://...
    android:layout_height="fill_parent"
    android:layout_width="fill_parent"
    android: orientation ="vertical">
  <AnalogClock
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
  <CheckBox
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Un checkBox" />
  <Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Un botón" />
  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Un texto cualquiera" />
</LinearLayout>
```

*TableLayout* distribuye los elementos de forma tabular. Se utiliza la etiqueta *TableRow* cada vez que queremos insertar una nueva línea.

```
<TableLayout xmlns:android="http://...
    android:layout_height="fill_parent"
    android:layout_width="fill_parent">
  <TableRow>
   <AnalogClock
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"/>
   <CheckBox
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Un checkBox"/>
  </TableRow>
  <TableRow>
   <Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Un botón"/>
   <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Un texto cualquiera"/>
  </TableRow>
</TableLayout>
```


*RelativeLayout* permite comenzar a situar los elementos en cualquiera de los cuatro lados del contenedor e ir añadiendo nuevos elementos pegados a estos.

```
<RelativeLayout
    xmlns:android="http://schemas...
    android:layout_height="fill_parent"
    android:layout_width="fill_parent">
   <AnalogClock
    android:id="@+id/AnalogClock01"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentTop="true"/>
   <CheckBox
    android:id="@+id/CheckBox01"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/AnalogClock01"
    android:text="Un checkBox"/>
  <Button
    android:id="@+id/Button01"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Un botón"
    android:layout_below="@+id/CheckBox01"/>
  <TextView
    android:id="@+id/TextView01"
    android:layout_width="wrap_content"
```



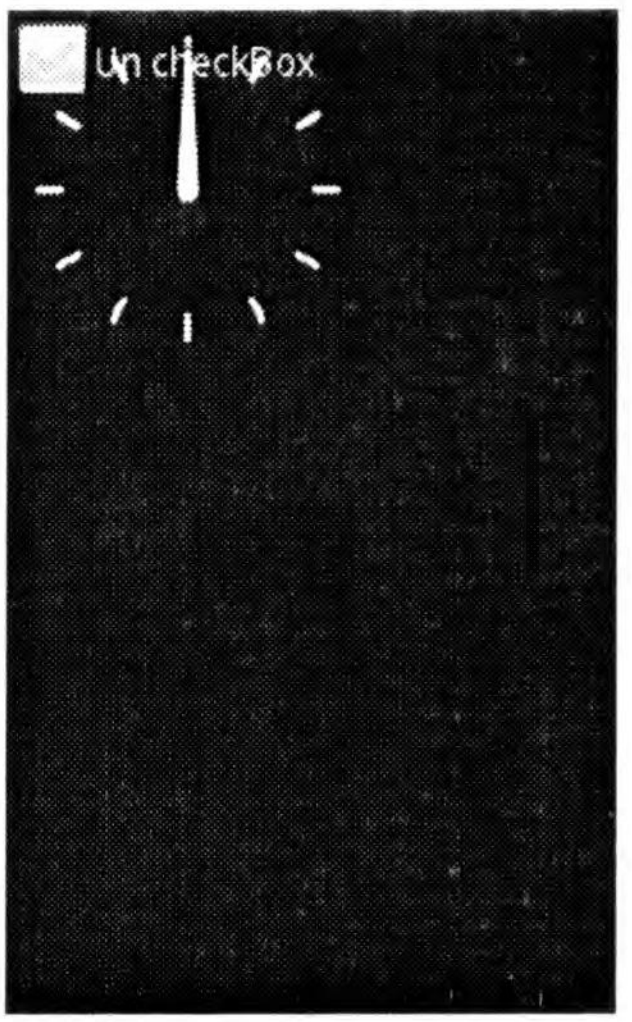
```
    android:layout_height="wrap_content"
    android:layout_alignParentBottom="true"
    android:text="Un texto cualquiera"/>
</RelativeLayout>
```

*AbsoluteLayout* permite indicar las coordenadas (x,y) donde queremos que se visualice cada elemento.

```
<AbsoluteLayout
xmlns:android="http://schemas.
    android:layout_height="fill_parent"
    android:layout_width="fill_parent">
<AnalogClock
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:layout_x="50px"
  android:layout_y="50px"/>
<CheckBox
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="Un checkBox"
  android:layout_x="150px"
  android:layout_y="50px"/>
<Button
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="Un botón"
  android:layout_x="50px"
  android:layout_y="250px"/>
  <TextView
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="Un texto cualquiera"
  android:layout_x="150px"
  android:layout_y="200px"/>
</AbsoluteLayout>
```


*FrameLayout* posiciona todos los elementos usando todo el contenedor, sin distribuirlos espacialmente. Este *Layout* suele utilizarse cuando queremos que varios elementos ocupen un mismo lugar pero solo uno será visible. Para modificar la visibilidad de un elemento utilizaremos la propiedad *visibility*.

```
<FrameLayout
xmlns:android="http://schemas...
    android:layout_height="fill_parent"
    android:layout_width="fill_parent">
   <AnalogClock
    android:layout_width="wrap_content"
    android:layout_height="wrap_conten"/>
   <CheckBox
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Un checkBox"/>
   <Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Un botón"
    android:visibility="invisible"/>
   <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Un texto cualquiera"
    android:visibility="invisible"/>
</FrameLayout>
```




**Poli[Media]**: *Los Layouts en Android.*

En algunas ocasiones, como en el *AbsoluteLayout*, tendremos que indicar las coordenadas donde ha de situarse un elemento. Dado que nuestra aplicación podrá ejecutarse en gran variedad de dispositivos con resoluciones muy diversas, Android nos permite indicar estas coordenadas de varias formas. En la siguiente tabla se muestran las diferentes posibilidades:

**px** (píxeles): Estas dimensiones representan los píxeles en la pantalla.

**mm** (milímetros): Distancia real medida sobre la pantalla.

**in** (pulgadas): Distancia real medida sobre la pantalla.

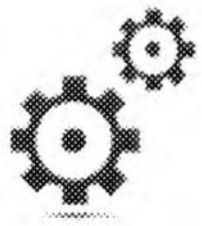
**pt** (puntos): Equivale a 1/72 pulgadas.

**dp** o **dip** (píxeles independientes de la densidad): Presupone un dispositivo de 160 píxeles por pulgada. Si luego el dispositivo tiene otra densidad, se realizará la correspondiente regla de tres. Cuando sea representado en dispositivos con una densidad grafica diferente, este hará un recalculado de forma que se conserve la misma medida midiendo sobre la pantalla dispositivo. Es decir *160dp* equivaldrá siempre a una pulgada en cualquier tipo de dispositivo.

**sp** (píxeles escalados): Similar a *dp* pero también se escala en función del tamaño de fuente que el usuario ha escogido en las preferencias. Indicado cuando se trabaja con fuentes.

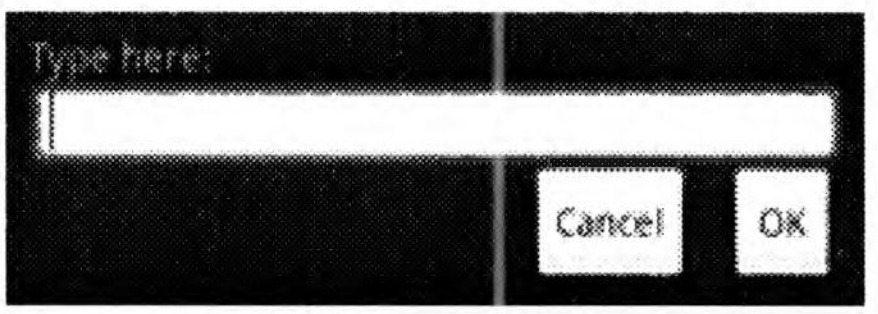
**Recursos adicionales:** *propiedades de RelativeLayout.*

La lista de propiedades específicas de RelativeLayout es muy grande. Te recomendamos que la consultes en el Anexo B.



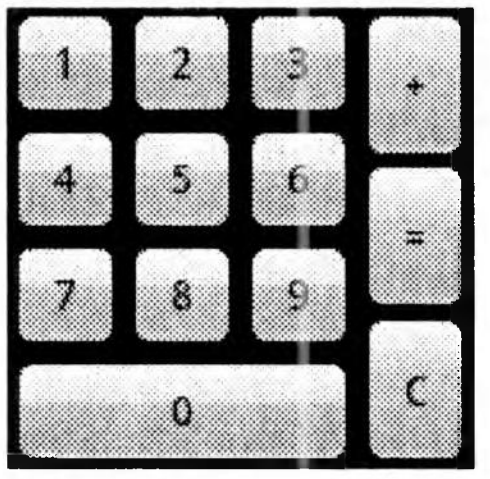
**Práctica:** *Uso de Layouts.*

1. Utiliza un `RelativeLayout` para realizar un diseño similar al siguiente:




2. Utiliza un `TableLayout` para realizar un diseño similar al siguiente:


3. Utiliza un `AbsoluteLayout` para realizar un diseño similar al siguiente. Ojo, `AbsoluteLayout` lo encontrarás en la paleta *Advanced* y no en la de *Layouts*. Trata de reutilizar el *Layout* creado en el punto anterior.




4. Visualiza el resultado obtenido en el punto anterior en diferentes tamaños de pantalla. ¿Ha sido una buena idea usar `AbsoluteLayout`? ¿Se te ocurre otra forma de realizar este diseño de forma que se visualice correctamente en cualquier tipo de pantalla?
5. Trata de hacer el ejercicio anterior utilizando *LinearLayout*.
6. Visualiza el resultado obtenido en diferentes tamaños de pantalla. ¿Has resuelto el problema?

**Preguntas de repaso y reflexión:** *Los Layouts.*

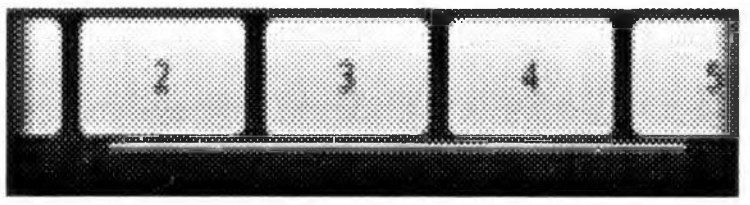
**Enlaces de interés:**

- *Hello, Views*: Colección de sencillos tutoriales para practicar con diferentes tipos de vistas:
  (http://developer.android.com/resources/tutorials/views/index.html)
- *Common Layout Objects*: Descripción de los tipo de Layouts más importantes.
  (http://developer.android.com/guide/topics/ui/layout-objects.html)

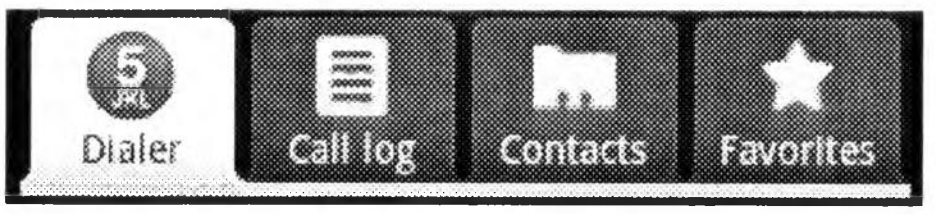
También podemos utilizar otras clases de *Layouts*, que son descritas a continuación:

***ScrollView***: Visualiza una columna de elementos; cuando estos no caben en pantalla se permite un deslizamiento vertical.

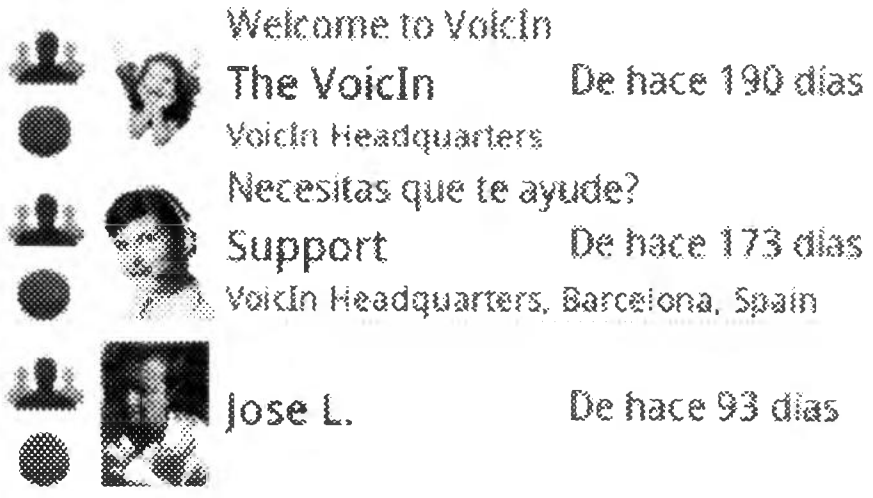
***HorizontalScrollView***: Visualiza una fila de elementos; cuando estos no caben en pantalla se permite un deslizamiento horizontal.



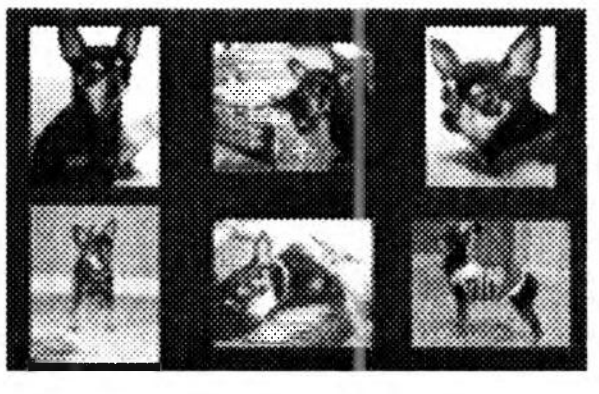

***TabHost***: Proporciona una lista de ventanas seleccionables por medio de etiquetas que pueden ser pulsadas por el usuario para seleccionar la ventana que desea visualizar. Se estudia al final del capítulo.




***ListView***: Visualiza una lista deslizable verticalmente de varios elementos. Su utilización es algo compleja. Se verá un ejemplo en el capítulo siguiente.



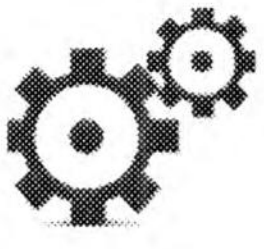

***GridView***: Visualiza una cuadrícula deslizable de varias filas y varias columnas.



***ViewFlipper***: Permite visualizar una lista de elementos de forma que se visualice uno cada vez. Puede ser utilizado para intercambiar los elementos cada cierto intervalo de tiempo.

## 2.4. Una aplicación de ejemplo: Asteroides

A lo largo de este libro vamos a ir creando una aplicación de ejemplo que toque los aspectos más significativos de Android. Comenzamos en este capítulo creando una serie de vistas que nos permitirán diseñar un sencillo interfaz de usuario.



**Práctica:** *Creación de la aplicación Asteroides.*

1. Crea un nuevo proyecto con los siguientes datos:

```
Project name: Asteroides
Build Target: Android 1.6
Application name: Asteroides
Package name: org.example.asteroides
Create Activity: Asteroides
Min SDK Version: 3
```

***NOTA:*** *En el desarrollo del ejemplo hemos indicado que para poder ejecutar la aplicación necesitamos como versión mínima el SDK 3, correspondiente a la versión 1.5. No obstante, vamos a utilizar la versión del SDK 1.6, dado que nos proporciona ciertos beneficios respecto a versiones anteriores (por ejemplo iconos de la aplicación en diferentes resoluciones). Esto es perfectamente posible, pero hay que tener precaución de no utilizar ninguna característica que no sea compatible con la versión 1.5.*

2. Abre el fichero *res/Layout/main.xml* y trata de crear una vista similar a la que ves a continuación. Ha de estar formada por un `LinearLayout` que contiene un `TexView` y cuatro `Button`. Trata de utilizar recursos para introducir los cinco textos que aparecen.


## Solución:

3. El fichero *main.xml* ha de ser similar al siguiente:

```
<LinearLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:gravity="center"
   android:padding="30dip">
<TextView
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:text="@string/tituloAplicacion"
   android:gravity="center"
   android:textSize="25sp "
   android:layout_marginBottom="20dip"/>
<Button android:id="@+id/Button01"
   android:layout_height="wrap_content"
   android:layout_width="fill_parent"
   android:text="@string/Arrancar"/>
<Button android:id="@+id/Button02"
   android:layout_height="wrap_content"
   android:layout_width="fill_parent"
   android:text="@string/Configurar"/>
<Button android:id="@+id/Button03"
   android:layout_height="wrap_content"
   android:layout_width="fill_parent"
   android:text="@string/Acercade"/>
```

```
    <Button android:id="@+id/Button04"
        android:layout_height="wrap_content"
        android:layout_width="fill_parent"
        android:text="@string/Salir"/>
</LinearLayout>
```

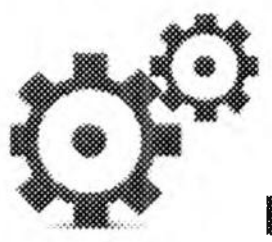
4. El fichero *res/strings.xml* ha de tener el siguiente contenido:

```
<resources>
    <string name="Arrancar">Jugar</string>
    <string name="Configurar">Configurar</string>
    <string name="Acercade">Acerca de </string>
    <string name="Salir">Salir</string>
    <string name="tituloAplicacion">Asteroides</string>
    <string name="hello">Hello World, Asteroides! </string>
    <string name="app_name">Asteroides</string>
</resources>
```

## 2.4.1. Recursos alternativos

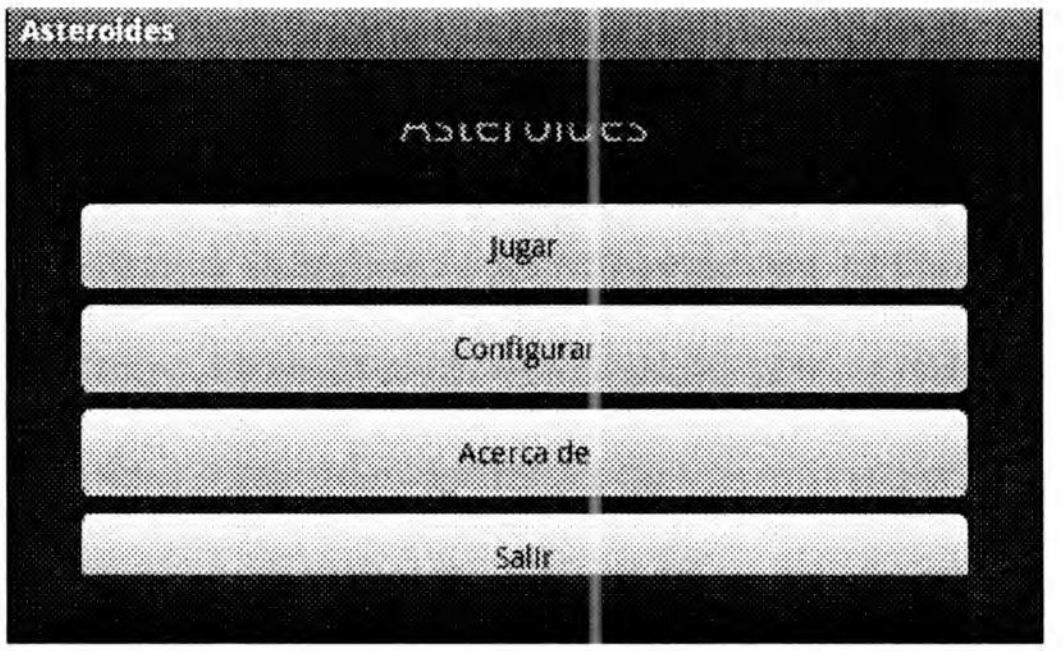
Una aplicación Android va a poder ser ejecutada en una gran variedad de dispositivos. El tamaño de pantalla, la resolución o el tipo de entrada puede variar mucho de un dispositivo a otro. Por otra parte, nuestra aplicación ha de estar preparada para diferentes modos de funcionamiento, como el modo automóvil o el modo noche, y para poder ser ejecutada en diferentes idiomas.

A la hora de crear el interfaz de usuario hemos de tener en cuenta todas estas circunstancias. Afortunadamente, la plataforma Android nos proporciona una herramienta de gran potencia para resolverlo, el uso de los recursos alternativos.



**Práctica:** *Recursos alternativos en Asteroides.*

1. Ejecuta la aplicación creada en el punto anterior en el emulador.
2. Los teléfonos móviles basados en Android permiten cambiar la configuración en apaisado y en vertical. Para conseguir este efecto con el emulador pulsa Ctrl+F11. Si observas el resultado de la vista que acabas de diseñar en vertical no queda todo lo bien que desearíamos.



Para resolver este problema Android te permite diseñar una vista diferente para la configuración horizontal y otra para vertical.

3. Crea la carpeta *res/layout-land*.
4. Copia en ella el fichero *main.xml*. *Para ello selecciona el fichero y pulsa Ctrl-C*. Selecciona la carpeta destino y pulsa *Ctrl-V*.
5. Crea una vista similar a la que ves a continuación: formada por un *LinearLayout* que contiene un *TexView* y un *TableLayout* con dos *Button* por columna.


6. Ejecuta de nuevo la aplicación y observa como la vista se ve correctamente en las dos orientaciones.

## Solución:

Has de obtener un código XML similar al siguiente:

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:gravity="center"
    android:padding="30dip">
  <TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/tituloAplicacion"
    android:gravity="center"
    android:textSize="25sp "
    android:layout_marginBottom="20dip"/>
<TableLayout
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:gravity="center"
    android:stretchColumns="*">
    <TableRow>
      <Button android:id="@+id/Button01"
```

```
        android:layout_height="wrap_content"
        android:layout_width="fill_parent"
        android:text="@string/Arrancar"/>
    <Button android:id="@+id/Button02"
        android:layout_height="wrap_content"
        android:layout_width="fill_parent"
        android:text="@string/Configurar"/>
</TableRow>
<TableRow>
    <Button android:id="@+id/Button03"
    android:layout_height="wrap_content"
    android:layout_width="fill_parent"
    android:text="@string/Acercade"/>
    <Button android:id="@+id/Button04"
    android:layout_height="wrap_content"
    android:layout_width="fill_parent"
    android:text="@string/Salir"/>
</TableRow>
</TableLayout>
</LinearLayout>
```

***NOTA:*** *Para conseguir en un* `TableLayout`*, que las columnas se ajusten a todo el ancho de la tabla poner* `stretchColumns="*"`*.* `stretchColumns="0"` *significa que asigne el ancho sobrante a la primera columna.* `stretchColumns="1"` *significa que asigne el ancho sobrante a la segunda columna.* `stretchColumns="*"` *significa que asigne el ancho sobrante entre todas las columnas.*

Android utiliza una lista de sufijos para expresar recursos alternativos. Estos sufijos pueden hacer referencia a la orientación del dispositivo, al lenguaje, la región, la densidad de píxeles, la resolución, el método de entrada,...

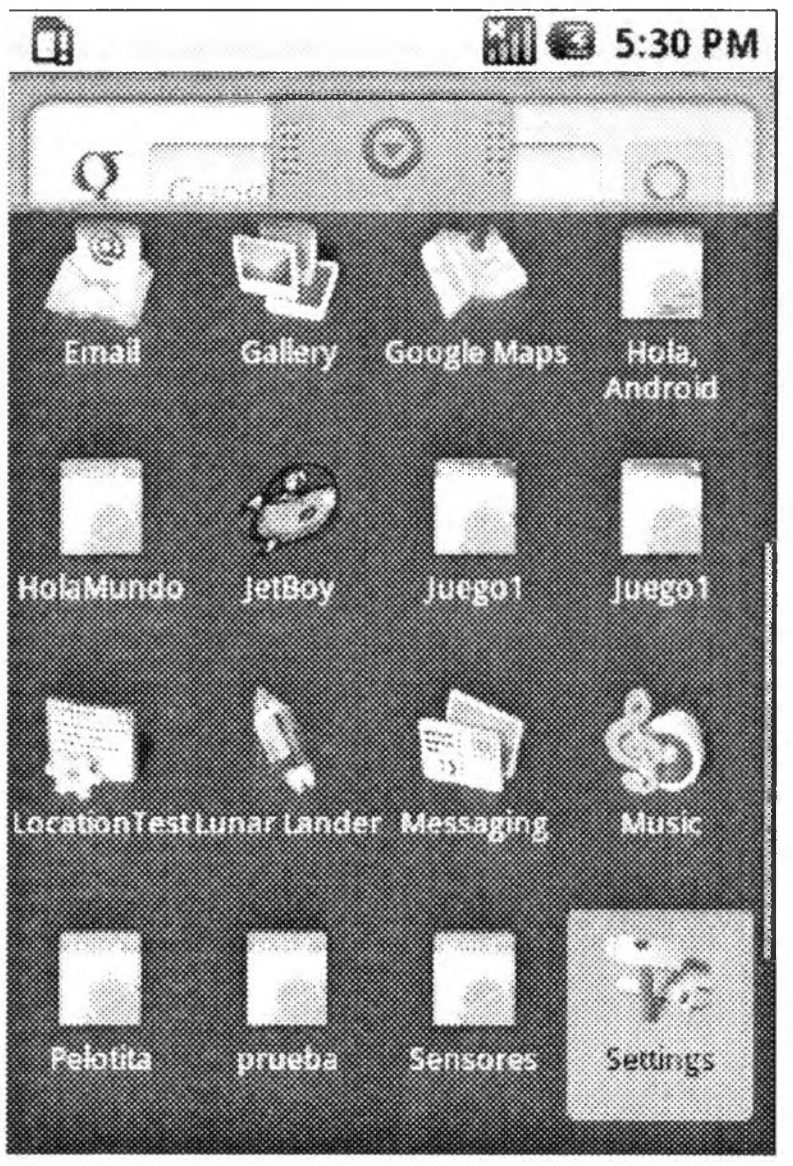
Por ejemplo, si queremos traducir nuestra aplicación al inglés, español y francés. Siendo el primer idioma el usado por defecto, crearíamos tres versiones del fichero *strings.xml* y lo guardaríamos en los siguientes tres directorios:

```
res/values/strings.xml
res/values-es/strings.xml
res/values-fr/strings.xml
```

**Ejercicio paso a paso:** *Traducción de Asteroides.*

1. Crea la carpeta *res/values-en*.
2. Copia en ella el fichero *strings.xml*.
3. Traduce en este fichero todas las cadenas al inglés.
4. Ejecuta la aplicación en el emulador y verifica que la aplicación está en inglés.

5. Vamos a cambiar la configuración de idioma del emulador. Para ello, accede al escritorio (pulsando el botón con forma de casa). En la parte inferior aparecerá un botón que permite desplegar un menú con todas las aplicaciones instaladas.




6. Abre la aplicación *Settings* y desciende hasta encontrar *Locale & text*. Dentro de esta opción selecciona *Select locale*. Selecciona ahora *Spanish*.

***NOTA:*** *Observa que en otros idiomas permite seleccionar tanto el idioma como la región. Por desgracia, para el español no.*

7. Ejecuta de nuevo la aplicación y observa cómo ha traducido el texto.

Otro ejemplo de utilización de recursos diferenciados lo podemos ver al crear una aplicación con Eclipse a partir de la versión 1.6 de Android. En este caso no creará una única carpeta *drawable* para almacenar el icono de la aplicación, sino tres según la densidad de píxeles del dispositivo:

```
res/drawable-hdpi/icon.png
res/drawable-mdpi/icon.png
res/drawable-ldpi/icon.png
```

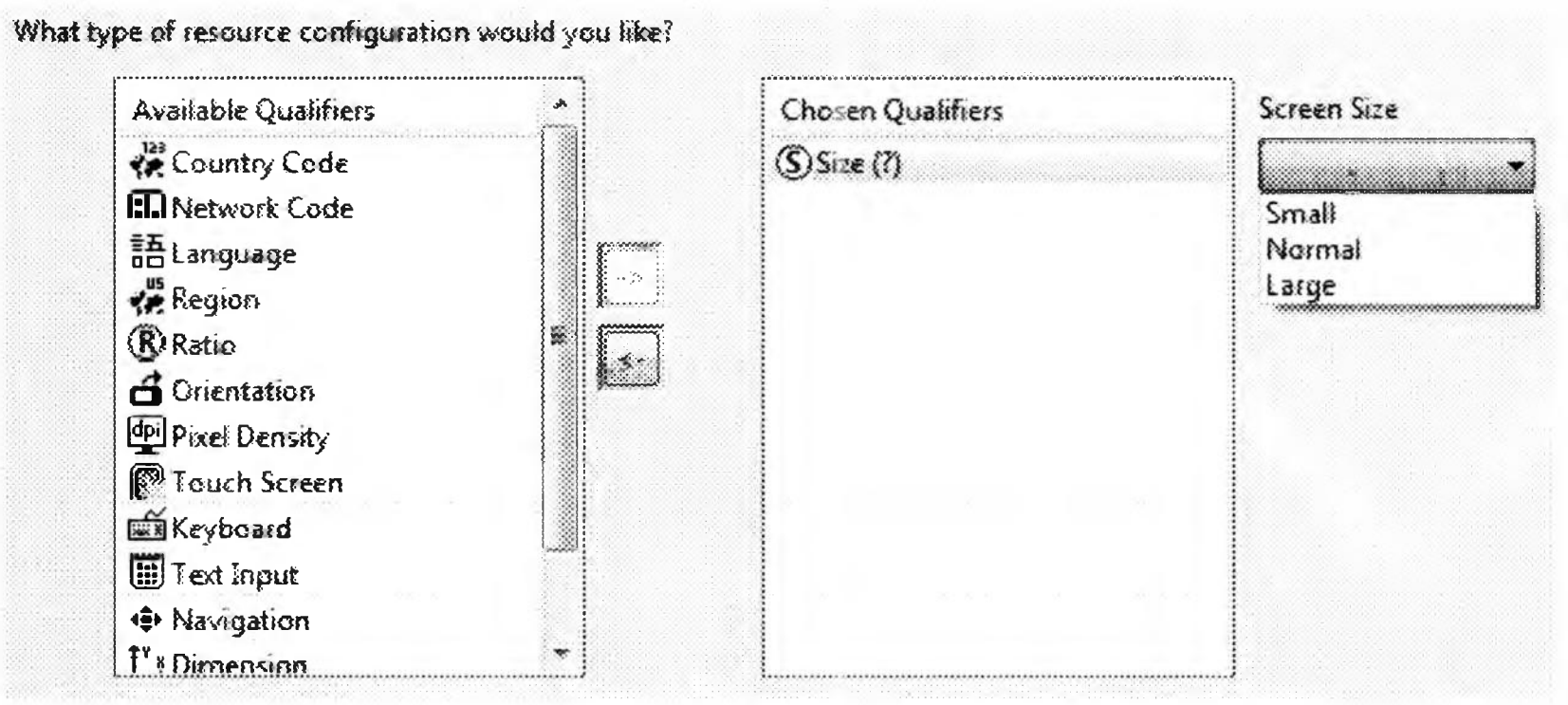
Resulta posible indicar varios sufijos concatenados; por ejemplo:

```
res/values-en-rUS/strings.xml
res/values-en-rUK/strings.xml
```

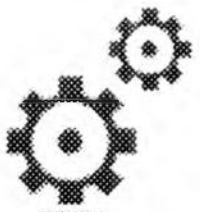
Pero cuidado, Android establece un orden a la hora de encadenar sufijos. Puedes encontrar una lista de estos sufijos en la documentación del SDK:

```
…/android-sdk-windows/docs/guide/topics/resources/resources-
i18n.html#AlternateResources
```

Para ver los sufijos disponibles también puedes pulsar con el botón derecho sobre una carpeta de recursos y seleccionar *New* > *Other...* > *Android* > *Android XML File*. Esta opción te permite crear un nuevo fichero XML y poner el sufijo deseado de forma y orden correcto.



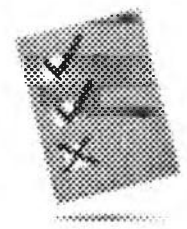

Poli[**Media**]: *Uso de recursos alternativos en Android.*



**Práctica:** *Creando un Layout para tabletas en Asteroides.*

Si ejecutas la aplicación Asteroides en una tableta observarás que el tamaño de los botones es demasiado grande.

1. Trata de hacer un *Layout* alternativo a *main.xml*, que sea utilizado en pantallas de tamaño `xlarge` (7-10,5 pulgadas).
2. Si lo deseas también puedes personalizar el fondo de la pantalla (atributo `background`), los tipos de letras, colores,...
3. Verifica que la aplicación se visualiza correctamente en todos los tipos de pantalla, tanto en horizontal como en vertical.



**Preguntas de repaso y reflexión:** *Recursos alternativos.*

## 2.5. Estilos y temas

Si tienes experiencia con el diseño de páginas Web habrás advertido grandes similitudes entre HTML y el diseño de *Layouts*. En los dos casos se utiliza un lenguaje de marcado y se trata de crear diseños independientes del tamaño de la

pantalla donde serán visualizados. En el diseño web, resultan clave las hojas de estilo en cascada (CSS) que permiten crear un patrón de diseño y aplicarlo a varias páginas. Cuando diseñes los *Layouts* de tu aplicación vas a poder utilizar unas herramientas similares conocidas como estilos y temas. Te permitirán crear patrones de estilo que podrán ser utilizados en cualquier parte de la aplicación. Estas herramientas te ahorrarán mucho trabajo y te permitirán conseguir un diseño homogéneo en toda tu aplicación.

## 2.5.1. Los estilos

Un estilo es una colección de propiedades que definen el formato que tendrá una vista. Podemos especificar cosas como tamaño, márgenes, color, fuentes, etc. Un estilo se define en ficheros XML, diferente al fichero XML *Layout* que lo utiliza.

Veamos un ejemplo. El siguiente código:

```
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:textColor="#00FF00"
    android:typeface="monospace"
    android:text="Un texto" />
```

Es equivalente a escribir:

```
<TextView
    style="@style/MiEstilo"
    android:text="Un texto" />
```

Habiendo creado en el fichero *res/values/styles.xml* con el siguiente código:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <style name="MiEstilo"
     parent="@android:style/TextAppearance.Medium">
       <item name="android:layout_width">fill_parent</item>
       <item name="android:layout_height">wrap_content</item>
       <item name="android:textColor">#00FF00</item>
       <item name="android:typeface">monospace</item>
   </style>
</resources>
```

Observa como un estilo puede heredar las propiedades de un padre (parámetro `parent`) y a partir de estas propiedades realizar modificaciones.

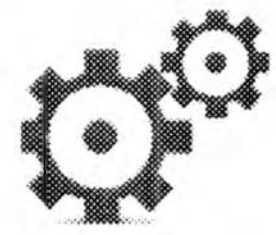
### *2.5.1.1. Heredar de un estilo propio*

Si vas a heredar de un estilo definido por ti no es necesario utilizar el atributo `parent`. Por el contrario, puedes utilizar el mismo nombre de un estilo ya creado y completar el nombre con un punto más un sufijo. Por ejemplo:

```
<style name="MiEstilo.grande">
     <item name="android:textSize">18pt</item>
</style>
```

Crearía un nuevo estilo que sería igual a `MiEstilo` más la nueva propiedad indicada. A su vez puedes definir otro estilo a partir de este:

```
<style name="MiEstilo.grande.negrita">
    <item name="android:textStyle">"bold"</item>
</style>
```



**Práctica:** *Creando un estilo en Asteroides.*

1. Abre el proyecto Asteroides.
2. Crea un nuevo estilo con nombre `TextoAsteroides`.
3. Aplícalo al título que aparece en el *Layout main.xml*.
4. Crea un nuevo estilo con nombre `TextoAsteroides.Botones`. Este ha de modificar alguno de los atributos anteriores y añadir otros, como `pading`.
5. Aplícalo a todos los botones del *Layout*.
6. Visualiza el resultado.

## 2.5.2. Los temas

Un tema es un estilo aplicado a toda una actividad o aplicación, en lugar de a una vista individual. Cada elemento del estilo solo se aplicará a aquellos elementos donde sea posible. Por ejemplo, `CodeFont` solo afectará al texto.

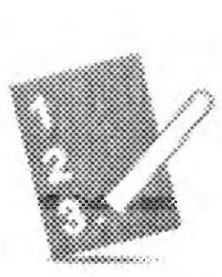
Para aplicar un tema a toda una aplicación edita el fichero *AndroidManifest.xml* y añade el parámetro `android:theme` en la etiqueta `application`:

```
<application android:theme="@style/MiTema">
```

También puedes aplicar un tema a una actividad en concreto:

```
<activity android:theme="@style/MiTema">
```

Además de crear tus propios temas vas a poder utilizar algunos disponibles en el sistema. Puedes encontrar una lista de todos los estilos y temas disponibles en Android en: http://developer.android.com/reference/android/R.style.html



**Ejercicio paso a paso:** *Aplicando un tema del sistema.*

1. Abre el proyecto Asteroides.
2. Aplica a la actividad principal el tema `android:style/Theme.Dialog` tal y como se acaba de mostrar.
3. Visualiza el resultado.

**Práctica:** *Creando un tema en Asteroides.*

1. Abre el proyecto Asteroides.
2. Crea un tema con nombre `TemaAsteroides` que herede de `android:style/Theme.NoTitleBar`. Este tema no muestra la barra con el nombre de la aplicación.
3. Aplica este tema a la aplicación.

**Polli[Media]**: *Estilos y temas en Android.*

## 2.6. Uso práctico de Vistas y Layouts

En este apartado vamos a aprender a usar varios tipos de vistas y *Layouts* desde un punto de vista práctico. También empezaremos a escribir código que será ejecutado cuando ocurran ciertos eventos:

**Ejercicio paso a paso:** *Un botón con gráficos personalizados.*

1. Crea un nuevo proyecto con nombre *MasVistas*. Selecciona la versión 1.6 y en *Package name* introduce `org.example.masvistas`.
2. Crea el fichero `boton.xml` en la carpeta `res/drawable/`. Para ello puedes utilizar el menú *Archivo/Nuevo/Android XML File* y pon en File: "botón" y selecciona en tipo Drawable. Reemplaza el código por el siguiente:

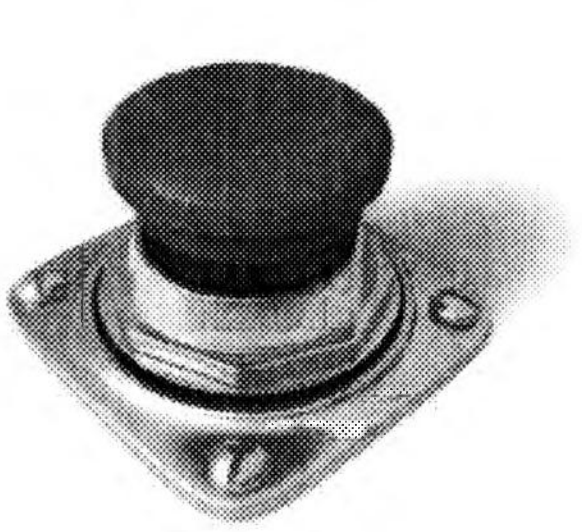
```
<?xml version="1.0" encoding="utf-8"?>
<selector
  xmlns:android="http://schemas.android.com/apk/res/android">
     <item android:drawable="@drawable/boton_pulsado"
           android:state_pressed="true" />
    <item android:drawable="@drawable/boton_con_foco"
           android:state_focused="true" />
    <item android:drawable="@drawable/boton_normal" />
</selector>
```

Este XML define un recurso único gráfico (*drawable*) que cambiará en función del estado del botón. El primer <item> define la imagen usada cuando se pulsa el botón, el segundo <item> define la imagen usada cuando el botón tiene el foco (cuando el botón está seleccionado con la

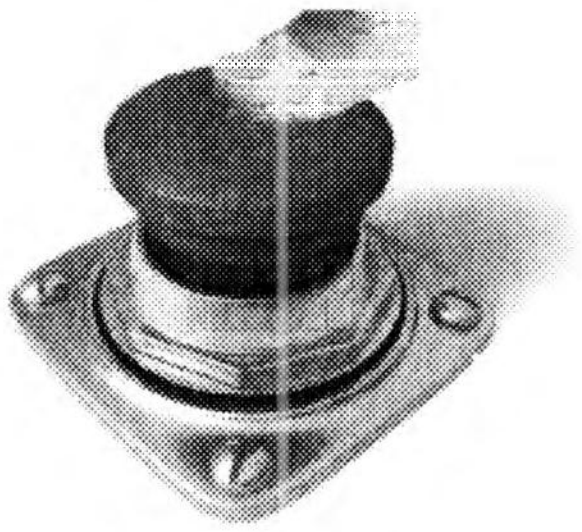
rueda de desplazamiento o las teclas de dirección), el tercero la imagen en estado normal. Los gráficos, y en concreto los *drawables,* serán estudiados en el capítulo 4.

***NOTA:*** *El orden de los elementos <item> es importante. Cuando se va a dibujar se recorren los ítems en orden hasta que se cumpla una condición. Debido a que "boton_normal" es el último, sólo se aplica cuando las condiciones* `state_pressed` *y* `state_focused` *no se cumplen.*

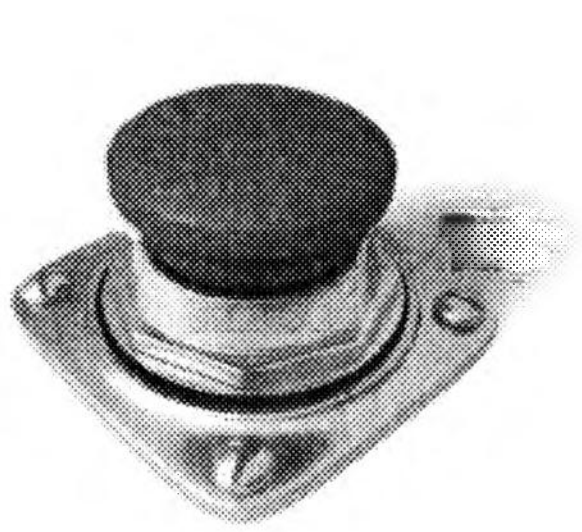
3. Descarga las tres imágenes que aparecen a continuación de www.androidcurso.com. El nombre que ha de tener cada fichero aparece debajo:



`boton_norma.jpg`



`boton_con_foco.jpg`



`boton_pulsado.jpg`

4. Arrastra estas imágenes a la carpeta `res/drawable/` del proyecto.
5. Abre el fichero `res/layout/main.xml`.
6. Elimina el `TextView` que encontrarás dentro del `LinearLayout`.
7. Selecciona el `LinerLayout` e introduce en el atributo `Background` el valor #FFFFFF.
8. Arrastra una vista de tipo `Button` dentro del `LinearLayout`.
9. Selecciona el atributo `Background` y pulsa en el botón selector de recurso (con puntos suspensivos). Selecciona *Drawable/boton*.
10. Modifica el atributo `Text` para que no tenga ningún valor.
11. Introduce en el atributo `onClick` el valor `sePulsa` (ojo; este atributo solo está disponible a partir de la versión 1.6).
12. Abre el fichero *MasVistasActivity.java* e introduce al final, antes de la última llave, el código:

```
public void sePulsa(View view){
    Toast.makeText(this, "Pulsado", Toast.LENGTH_SHORT).show();
}
```

***NOTA:*** *Pulsa Ctrl-Shift-O para que se añadan automáticamente los paquetes que faltan en la sección* `import`.

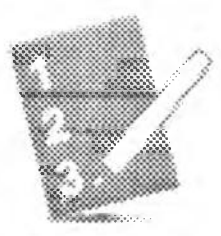
El método anterior será ejecutado cuando se pulse el botón. Este método se limita a lanzar un *Toast*, es decir, un aviso que permanece un cierto tiempo sobre la pantalla y luego desaparece. Los tres parámetros son: 1 para el contexto utilizado, coincide con la actividad; 2 para el texto a mostrar y 3 para el tiempo que permanecerá este texto. Los conceptos de *actividad* y *contexto* serán desarrollados en el siguiente capítulo.

13. Ejecuta el proyecto y verifica el resultado.

14. El código resultante para el fichero *main.xml* se muestra a continuación:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="#FFFFFF">
    <Button android:id="@+id/button1"
     android:layout_width="wrap_content"
     android:layout_height="wrap_content"
     android:background="@drawable/boton"
     android:onClick="sePulsa"/>
</LinearLayout>
```

## 2.6.1. Acceder y modificar las propiedades de las vistas por código



**Ejercicio paso a paso:** *Acceder y modificar las propiedades de las vistas por código*

1. Abre el *Layout main.xml* creado en el ejercicio anterior.
2. En la paleta de vistas, dentro de *Text Fields*, busca *Number (Decimal)* y arrástralo encima del botón rojo.
3. Modifica algunos atributos de esta vista: `Hint` = "Introduce un número", `id` = "@+id/entrada"
4. En la paleta de vistas, dentro de *Form Widgets*, busca *Button* y arrástralo encima del botón rojo.
5. Modifica algunos atributos de esta vista: Haz que su ancho ocupe toda la pantalla, que su texto sea "0".
6. En la paleta de vistas, dentro de *Form Widgets*, busca *Large Text* y arrástralo debajo del botón rojo.
7. Modifica algunos atributos de esta vista: `TextColor` = #0000FF, `Text` = "", `Hint` = "Resultado", `id`= "@+id/salida".

8. Abre el fichero *MasVistaActivity.java*. Vamos a añadir dos nuevas propiedades a la clase. Para ello copia el siguiente código al principio de la clase (antes del método `onCreate`):

```
private EditText entrada;
private TextView salida;
```

9. Copia al final del método `onCreate` las siguientes dos líneas:

```
entrada = (EditText) findViewById(R.id.entrada);
salida = (TextView) findViewById(R.id.salida);
```

10. Como se explicó al principio del capítulo, las diferentes vistas definidas en *main.xml*, son creadas como objetos Java cuando se ejecuta `setContentView(R.layout.main)`. Si queremos manipular algunos de estos objetos hemos de declararlos (paso 8) y asignarles el objeto correspondiente. Para ello, hay que introducir el atributo `id` en *xml* y utilizar el método `findViewById(R.id.valor_en_atributo_id)`. Este método devuelve un objeto de la clase `View`, no obstante los objetos declarados (`entrada` y `salida`) no son exactamente de esta clase por lo que Java no permite una asignación directa. En estos casos hemos de utilizar una conversión de tipo (*type cast*) para poder hacer la asignación. Para más información al respecto leer el apartado Polimorfismo del tutorial de Java Esencial.

11. Introduce en el atributo `onClick` del botón con id `boton0` el valor "sePulsa0".

12. Añade el siguiente método al final de la clase `MasVistasActivity`.

```
public void sePulsa0(View view){
  entrada.setText(entrada.getText()+"0");
}
```

Lo que hace es asignar como textos de `entrada` el resultado de concatenar al texto de `entrada` el carácter "0".

13. Añade al botón con texto "0" el atributo *tag* = "0".

14. Modifica el método `sePulsa0` de la siguiente forma:

```
public void sePulsa0(View view){
  entrada.setText(entrada.getText()+(String)view.getTag());
}
```

El resultado obtenido es equivalente al anterior. En algunos casos será interesante utilizar un mismo método como escuchador de eventos de varias vistas. Podrás averiguar la vista que causó el evento, dado que esta es pasada como parámetro del método. En el ejemplo sabemos que en el atributo `tag` guardamos el carácter a insertar. El atributo `tag` puede ser usado libremente por el programador para almacenar un objeto de la clase `Object`. En nuestro caso hemos almacenado un objeto `String`, por lo que necesitamos una conversión de tipo.

***NOTA**: Utiliza esta forma de trabajar en la práctica para no tener que crear un método onClick para cada botón de la calculadora.*

15. Modifica el código de `sePulsa` con el siguiente código:

```
public void sePulsa(View view){
  salida.setText(String.valueOf(
  Float.parseFloat(entrada.getText().toString())*2.0));
}
```

En este código el valor de `entrada` es convertido en Float, multiplicado por dos y convertido en String para ser asignado a `salida`.

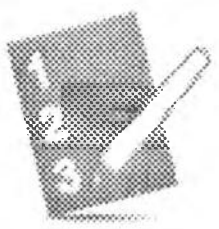
16. Ejecuta el proyecto y verifica el resultado.

## 2.7. Uso de TabLayout

Para crear una interfaz de usuario con pestañas, es necesario utilizar `TabHost` y `TabWidget`. El `TabHost` debe ser el nodo raíz para el diseño, que contendrá tanto el `TabWidget` para la visualización de las pestañas, como un `FrameLayout` para mostrar el contenido de la ficha.

Puedes implementar el contenido de dos maneras: usando las pestañas para intercambiar puntos de vista dentro de la misma actividad, o puedes utilizar las pestañas para cambiar entre actividades totalmente independientes.

En este apartado, vamos a crear una interfaz de usuario con pestañas que utiliza una única actividad. Para hacerlo con diferentes actividades para cada pestaña puedes seguir el tutorial: http://developer.android.com/resources/tutorials/views/hello-tabwidget.html



**Ejercicio paso a paso:** *Uso de TabLayout.*

1. Crea un nuevo proyecto con nombre *TabLayout*. Selecciona la versión 1.6 y en *Package name* introduce `org.example.tablayout`.
2. Remplaza el código de *main.xml* por el siguiente:

```
<?xml version="1.0" encoding="utf-8"?>
<TabHost
xmlns:android="http://schemas.android.com/apk/res/android"
  android:id="@android:id/tabhost"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent">
  <LinearLayout
     android:orientation="vertical"
     android:layout_width="fill_parent"
```

```
        android:layout_height="fill_parent">
        <TabWidget
            android:id="@android:id/tabs"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content" />
        <FrameLayout
            android:id="@android:id/tabcontent"
            android:layout_width="fill_parent"
            android:layout_height="fill_parent">
            <LinearLayout
            android:id="@+id/tab1Layout"
            android:orientation="vertical"
            android:layout_height="wrap_content"
            android:layout_width="fill_parent">
            <TextView
                android:layout_width="fill_parent"
                android:layout_height="fill_parent"
                android:text="esto es una pestaña" />
            <EditText
                android:layout_height="wrap_content"
                android:layout_width="fill_parent"/>
        </LinearLayout>
        <TextView
            android:id="@+id/tab2Layout"
                android:layout_width="fill_parent"
                android:layout_height="fill_parent"
                android:text="esto es otra pestaña" />
            <TextView android:id="@+id/tab3Layout"
                android:layout_width="fill_parent"
                android:layout_height="fill_parent"
                android:text="esto es la tercera pestaña" />
        </FrameLayout>
    </LinearLayout>
</TabHost>
```

Como puedes observar se ha creado un `TabHost` debe ser el nodo raíz para el diseño, que contiene dos elementos combinados por medio de un `LinearLayout`. El primero es un `TabWidget` para la visualización de las pestañas y el segundo es un `FrameLayout` para mostrar el contenido de la ficha. Dentro de este `FrameLayout` hay tres elementos, cada uno de los cuales contendrá las vistas a mostrar para cada una de las lengüetas. Tienes que tener especial cuidado con los atributos `id`. El `TabHost` debe llamarse siempre *"@android:id/tabhost"*, mientras que el `TabWidget` ha de llamarse *"@android:id/tabs"*. De esta forma, el sistema conocerá la finalidad de cada uno de estos elementos. Por otra parte, cada una de las vistas de contenido introducidas dentro del `FrameLayout` han de tener un `id` asignado por ti, como *"@+id/tab1Layout"*.

3. Para que un *TabLayout* funcione resulta imprescindible introducir código. Abre el fichero *TabLayoutActivity.java* y remplaza el código por el siguiente.

```
public class TabLayoutActivity extends TabActivity {
  /** Called when the activity is first created. */
  @Override
  public void onCreate(Bundle savedInstanceState) {
     super.onCreate(savedInstanceState);
     setContentView(R.layout.main);
     TabHost tabHost = getTabHost();
     tabHost.addTab(tabHost.newTabSpec("tab1").setIndicator(
        "Título 1", null).setContent(R.id.tab1Layout));
     tabHost.addTab(tabHost.newTabSpec("tab2").setIndicator(
        "Título 2", null).setContent(R.id.tab2Layout));
     tabHost.addTab(tabHost.newTabSpec("tab3").setIndicator(
        "Título 3", null).setContent(R.id.tab3Layout));
  }
}
```

Observa como la clase creada extiende de `TabActivity` en lugar de `Activity`. Además, se han añadido varias líneas al final del método `onCreate()`. La primera línea obtiene la actividad que muestra las pestañas mediante `getTabHost()`. Luego añadimos tantas pestañas como nos interese. Para cada una se crea indicando un *tag* (`newTabSpec()`), se le asocia un título y un icono (`setIndicator()`) y se indica donde está el contenido (`setContent()`).

***NOTA:*** *Los iconos disponibles en el sistema y cómo crear nuevos icono será estudiado en el siguiente capítulo.*

4. Ejecuta el proyecto y verifica el resultado.

## 2.7.1. Uso de la etiqueta <include> en Layouts

Un diseño basado en *TabLayout* puede requerir ficheros xml muy extensos. Para organizar correctamente el trabajo y reutilizar diseños previos puede ser de gran ayuda la etiqueta <include>

**Ejercicio paso a paso:** *Uso de la etiqueta <include> en Layouts.*

1. Empezaremos realizando una copia del fichero *MasVistas/res/layout/main.xml* a *TabLayout/res/layout/boton_rojo.xml*. Para ello selecciona el primer fichero y pégalo en la carpeta *MasVistas/res/layout*. Te pedirá un nuevo nombre escribe *boton_rojo.xml*. Arrastra el nuevo fichero hasta la carpeta *TabLayout/res/layout*.

2. Realiza el mismo proceso con uno de los *Layout* creado en el punto 7, donde se diseñaba el teclado de una calculadora. En nuestro proyecto ha de llamarse *calculadora.xml*.

3. Realiza el mismo proceso con el *Layout* creado en el punto 6. En nuestro proyecto ha de llamarse *primer_layout.xml*.

4. Añade el atributo `android:id="@+id/tab1Layout"` en el `LinearLayout` raíz de *botón_rojo.xml*. Añade el atributo `android:id="@+id/tab2Layout"` en *calculadora.xml* y `"@+id/tab3Layout"` en *primer_layout.xml*.

5. Abre el fichero *main.xml* y remplaza el `FrameLayout` por:

```
<FrameLayout
  android:id="@android:id/tabcontent"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent">
  <include layout="@layout/boton_rojo"/>
  <include layout="@layout/calculadora"/>
  <include layout="@layout/primer_layout"/>
</FrameLayout>
```

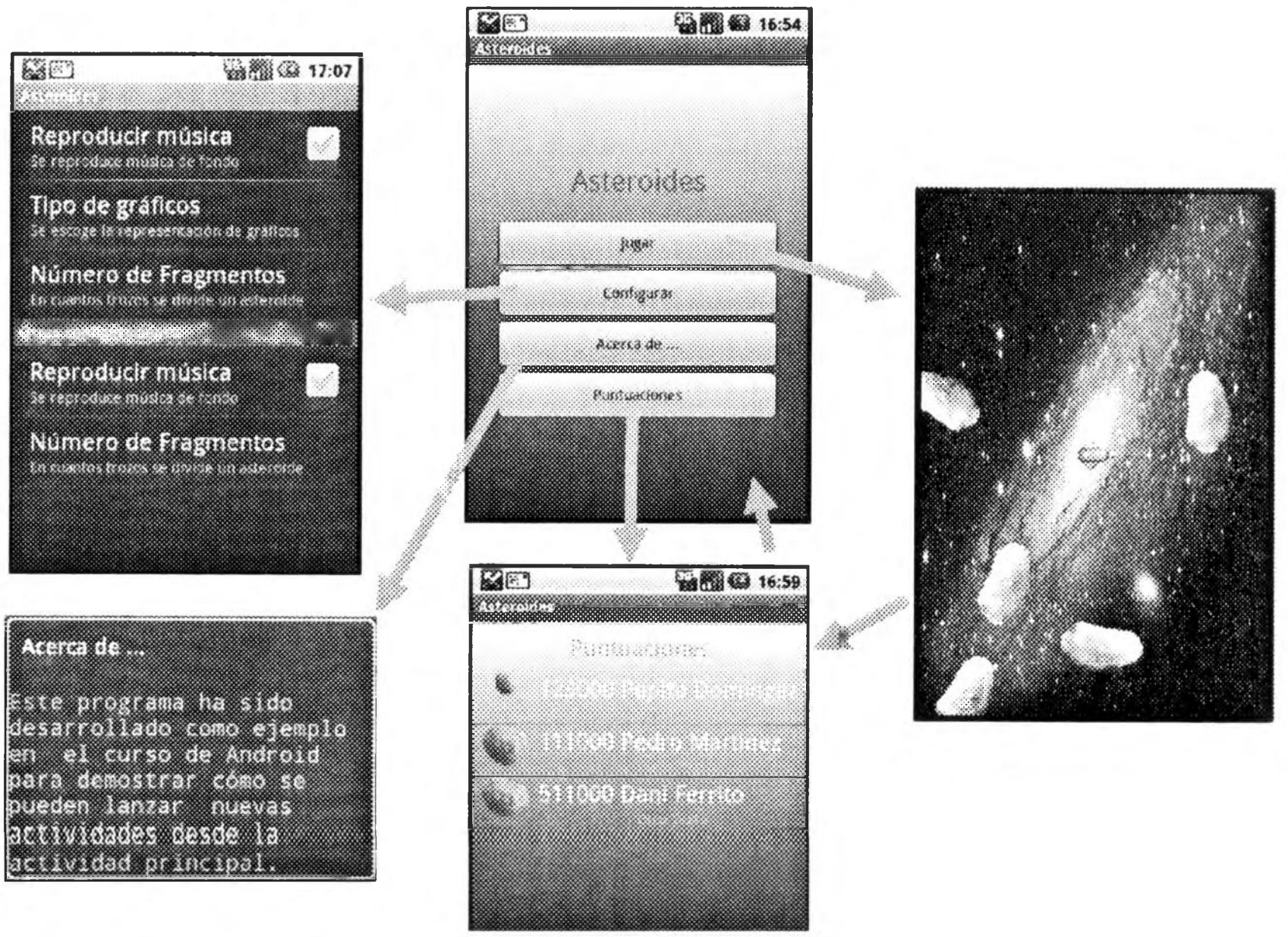
6. Ejecuta el proyecto y verifica el resultado. Ojo: si pulsas alguno de los botones es posible que no funcione. Recuerda que no hemos copiado el código.

Has podido comprobar cómo hemos conseguido un diseño muy complejo sin la necesidad de crear un xml demasiado grande. De esta forma, tenemos el código xml separado en cuatro ficheros diferentes. Además ha sido muy sencillo reutilizar diseños previos.

# CAPÍTULO 3.
# Actividades e Intenciones

En este capítulo seguiremos trabajando con el diseño del interfaz de usuario. En lugar de tratar aspectos de diseño visual, como hicimos en el capítulo anterior, vamos a tratar temas más relacionados con el código. En concreto, nos centraremos en las Actividades y los Intentos. Estudiaremos también dos herramientas de gran utilidad para cualquier aplicación: los menús contextuales y la definición de parámetros de configuración. Además, se tratará un tipo de vista muy práctica aunque algo compleja de manejar: `ListView`.

Nos vamos a centrar en el ejemplo de aplicación que estamos desarrollando, Asteroides, para añadirle diferentes actividades. A continuación se muestra el esquema de navegación entre las actividades que queremos crear:



Objetivos:

- Describir cómo el interfaz de usuario en una aplicación Android estará formado por un conjunto de actividades.
- Mostrar como desde una actividad podemos invocar a otras y cómo podemos comunicarnos con ellas.
- Permitir incorporar a nuestras aplicaciones ciertos elementos prácticos como son los menús o las preferencias.
- Describir cómo podemos utilizar y crear iconos en nuestras aplicaciones.
- Estudiar una vista muy útil en Android: *ListView*.
- Describir el uso de intenciones para invocar actividades estándar en Android.

# 3.1. Creación de nuevas actividades

El concepto de actividad en Android representa una unidad de interacción con el usuario. Corresponde a lo que coloquialmente llamamos una pantalla de la aplicación. Una aplicación suele estar formada por una serie de actividades, de forma que el usuario puede ir navegando entre actividades. En concreto, Android suele disponer de un botón (físico o en pantalla) que nos permite volver a la actividad anterior.

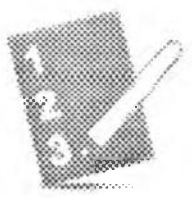
Toda actividad ha de tener una vista asociada, que será utilizada como interfaz de usuario. Esta vista suele ser de tipo *Layout* aunque también puede ser una vista simple, como se verá en el siguiente ejemplo.

Una aplicación estará formada por un conjunto de actividades independientes, es decir se trata de clases independientes que no comparten variables, aunque todas trabajan para un objetivo común. Otro aspecto importante es que toda actividad ha de ser una subclase de `Activity`.

Las aplicaciones creadas en los ejemplos hasta ahora disponían de una única actividad. Esta era creada automáticamente y se le asignaba la vista definida en *res/Layout/main.xml*. Esta actividad era arrancada al comenzar la aplicación. A medida que nuestra aplicación crezca va a ser imprescindible crear nuevas actividades. En esta apartado describiremos como hacerlo. Este proceso se puede resumir en cuatro pasos:

- Crear un nuevo *Layout* para la actividad.
- Crear una nueva clase descendiente de `Activity`. En esta clase tendrás que indicar que el *Layout* a visualizar es el desarrollado en el punto anterior.
- Para que nuestra aplicación sea visible será necesario activarla desde otra actividad.
- De forma obligatoria tendremos que registrar toda nueva actividad en *AndroidManifert.xml*.

Veamos un primer ejemplo de cómo crear una nueva actividad en la aplicación Asteroides.



**Ejercicio paso a paso:** *Implementación de una caja Acerca de.*

Vamos a crear una caja *Acerca de...* y visualizarla al pulsar el botón adecuado.

1. En primer lugar crea el fichero *res/layout/acercade.xml*. Para ello pulsa con el botón derecho sobre una carpeta *res/layout*, selecciona *New* > *Other...* *Selecciona Android* > *Android XML File* e indica *acercade.xml* en *File.*

2. Selecciona la lengüeta de edición en XML y copia el siguiente contenido:

```
<?xml version="1.0" encoding="utf-8"?>
<TextView
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:id="@+id/TextView01"
      android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="Este programa ha sido desarrollado como
    ejemplo en el curso de Android para demostrar cómo se
    pueden lanzar nuevas actividades desde la actividad
    principal.">
</TextView>
```

3. Creamos ahora una nueva actividad, que será la responsable de visualizar esta vista. Para ello se crea el fichero *AcercaDe.java*. Botón derecho sobre *src/org.example.asteroides* y seleccionamos *New* > *Class*. Remplaza el código por:

```
package org.example.asteroides;

import android.app.Activity;
import android.os.Bundle;

public class AcercaDe extends Activity {
    /** Called when the activity is first created. */
    @Override public void onCreate(Bundle
savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.acercade);
    }
}
```

4. Pasemos ahora a crear un método en la clase *Asteroides.java* que será ejecutado cuando sea pulsado el botón *Acerca de*.

```
public void lanzarAcercaDe(View view){
  Intent i = new Intent(this, AcercaDe.class);
  startActivity(i);
}
```

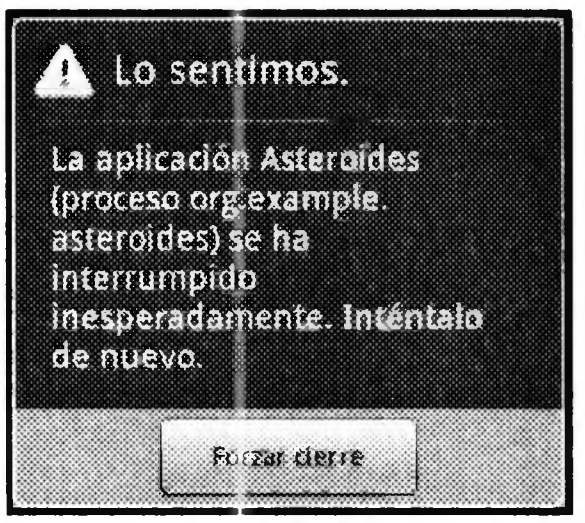
**Nota sobre Java:** *Pulsa **Ctrl-Shift-O**, para que automáticamente se añadan los paquetes que faltan.*

5. Para asociar este método al botón edita el *Layout main.xml*. Selecciona el botón *Aderca de...* y en la vista de Eclipse *Properties* busca el atributo *onClick* e introduce el valor `lanzarAcercaDe`.

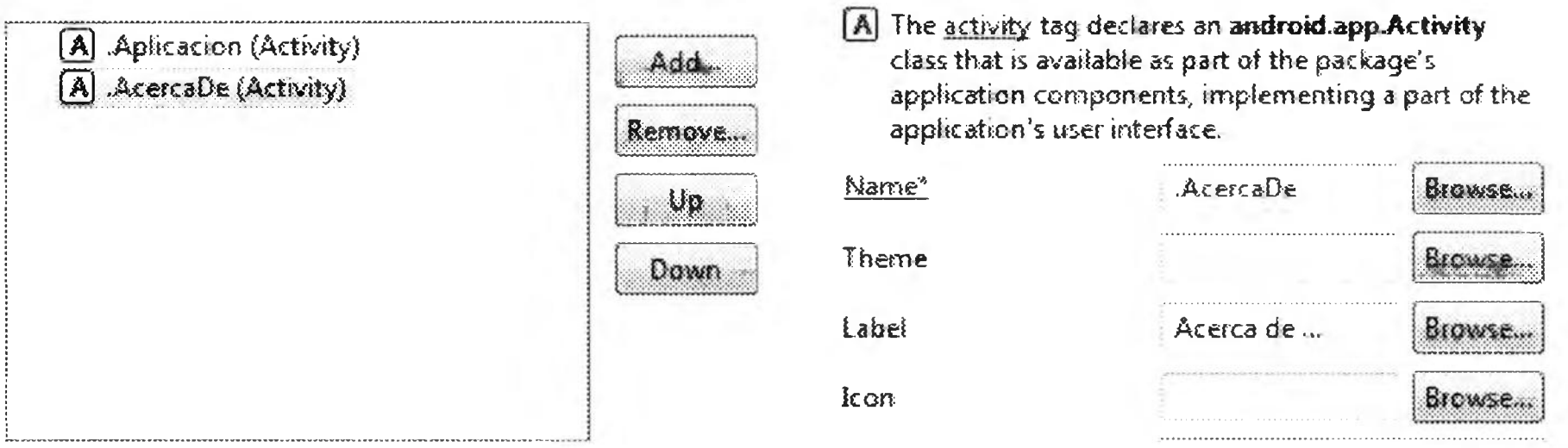
6. Pasa a la edición xml pulsando en la lengüeta *main.xml* y observa como en la etiqueta `<Button>` correspondiente, se ha añadido el atributo:

```
android:onClick="lanzarAcercaDe"
```

7. Ejecuta ahora la aplicación y pulsa en el botón *Acerca de*. Observarás que el resultado no es satisfactorio ¿Qué ha ocurrido?



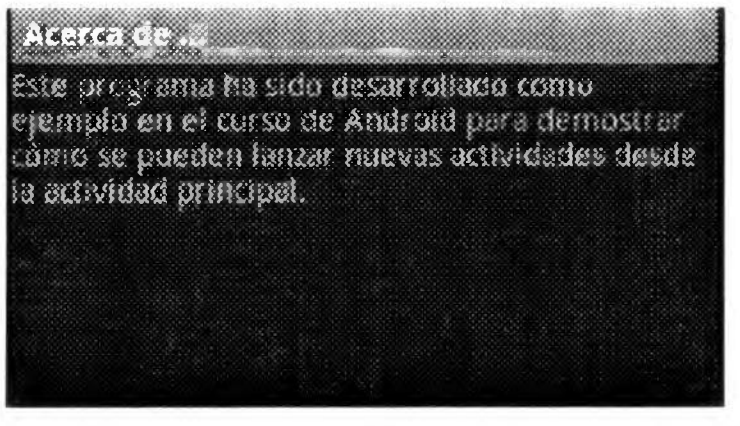

8. El problema es que toda actividad que ha de ser lanzada por una aplicación ha de ser registrada en el fichero *AndroidManifest.xml*. Para registrar la actividad, abre *AndroidManifest.xml* y selecciona la lengüeta *Application*. En *Aplication Nodes* pulsa el botón *Add...* y selecciona *Activity*. Rellena los campos de la derecha tal y como se muestra a continuación:




9. Observa como en el fichero xml se añade la siguiente línea:

```
<activity   android:name=".AcercaDe"
            android:label="Acerca de ..."/>
```

10. Ejecuta de nuevo el programa. El resultado ha de ser similar al mostrado a continuación:



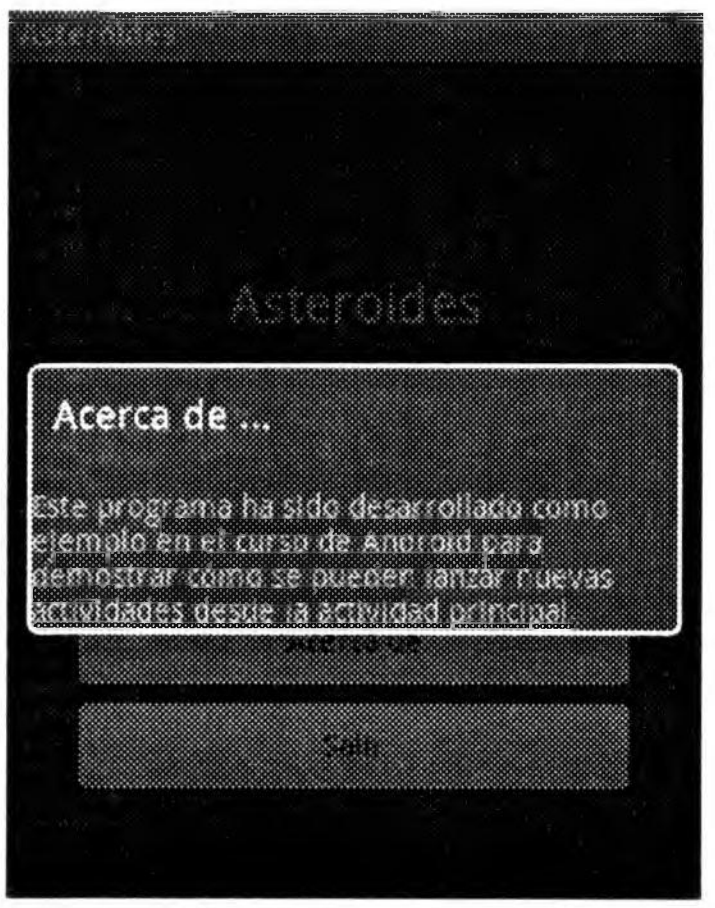

La vista mostrada en el ejemplo anterior no parece muy atractiva. Tratemos de mejorarla aplicando un tema. Como vimos en el capítulo anterior un tema es una colección de estilos que define el aspecto de una activad o aplicación. Puedes utilizar alguno de los temas disponibles en Android o crear el tuyo propio.

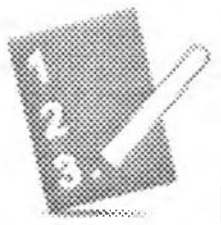
11. En este caso utilizaremos uno de los de Android. Para ello abre *AndroidManifest.xml* y selecciona la lengüeta *Application*. En *Aplication Nodes* pulsa sobre *AcercaDe* y a la derecha introduce el valor: `@android:style/Theme.Dialog` en el campo *Theme*.

12. Si lo prefieres edita directamente el fichero *AndroidManifest.xml* para que incluya:

```
<activity android:name=".AcercaDe"
          android:label="Acerca de ..."
          android:theme="@android:style/Theme.Dialog"/>
```

13. Ejecuta de nuevo el programa y observa cómo mejora la apariencia:



**Ejercicio paso a paso:** *Un escuchador de evento por código.*

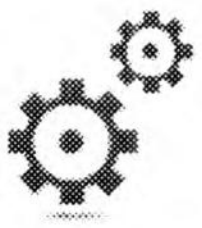
Como acabamos de ver el atributo `android:onClick` nos permite asignar desde un *Layout* xml un método que será ejecutado al hacer clic en una vista. Resulta muy sencillo y además está disponible en cualquier descendiente de la clase `View`. Sin embargo esta técnica presenta dos inconvenientes. En primer lugar, solo está

disponible a partir de la versión 1.6 de Android. En segundo lugar, solo está disponible para el evento `onClick()`. La clase `View` tiene otros eventos (`onLongClick()`, `onFocusChange()`, `onKey()`,...) para los que no se han definido un atributo xml. Entonces, qué hacemos si queremos trabajar con una versión anterior a 1.6 o definir un evento distinto de `onClick()`. La respuesta la encontrarás en el siguiente ejercicio:

1. Abre la clase *Asteroides.java* y añade las líneas que aparecen subrayadas:

```
public class Asteroides extends Activity {

    private Button bAcercaDe;

    @Override public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        bAcercaDe =(Button) findViewById(R.id.button03);
        bAcercaDe.setOnClickListener(new OnClickListener() {
        public void onClick(View view) {
            lanzarAcercaDe(null);
        }
    });
}
...
```

En el capítulo 5 se estudiarán con más detalle los escuchadores de evento.



**Práctica:** *El botón Salir.*

Queremos que cuando se pulse el botón *Salir* se ejecute el código: `finish();` De esta forma la actividad terminará al pulsarlo.

1. Realiza este trabajo utilizando un escuchador de evento por código.
2. Hazlo ahora con el atributo xml `android:onClick`.
3. Verifica que el resultado es el mismo en ambos casos.

## 3.2. Comunicación entre actividades

Cuando una actividad ha de lanzar a otra actividad en muchos casos necesita enviarle cierta información. Android nos permite este intercambio de datos utilizando el mecanismo descrito a continuación:

Cuando lances una actividad usa el siguiente código:

```
Intent intent = new Intent(this, MI_CLASE.class);
intent.putExtra("usuario", "Pepito Perez");
```

```
intent.putExtra("edad", 27);
startActivity(intent);
```

En la actividad lanzada podemos recoger los datos de la siguiente forma:

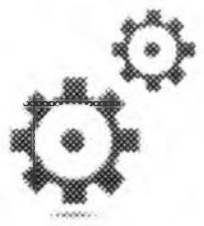
```
Bundle extras = getIntent().getExtras();
String s = extras.getString("usuario");
int i = extras.getInt("edad");
```

Cuando la actividad lanzada termina también podrá devolver datos que podrán ser recogidos por la actividad lanzadora de la siguiente manera.

```
Intent intent = new Intent(this, MI_CLASE.class);
startActivityForResult(intent, 1234);
...
@Override protected void onActivityResult (int requestCode,
                                           int resultCode, Intent data){
  if (requestCode==1234 & resultCode==RESULT_OK) {
     String res = data.getExtras().getString("resultado");
  }
}
```

En la actividad llamada has de escribir:

```
Intent intent = new Intent();
intent.putExtra("resultado","valor");
setResult(RESULT_OK, intent);
finish();
```

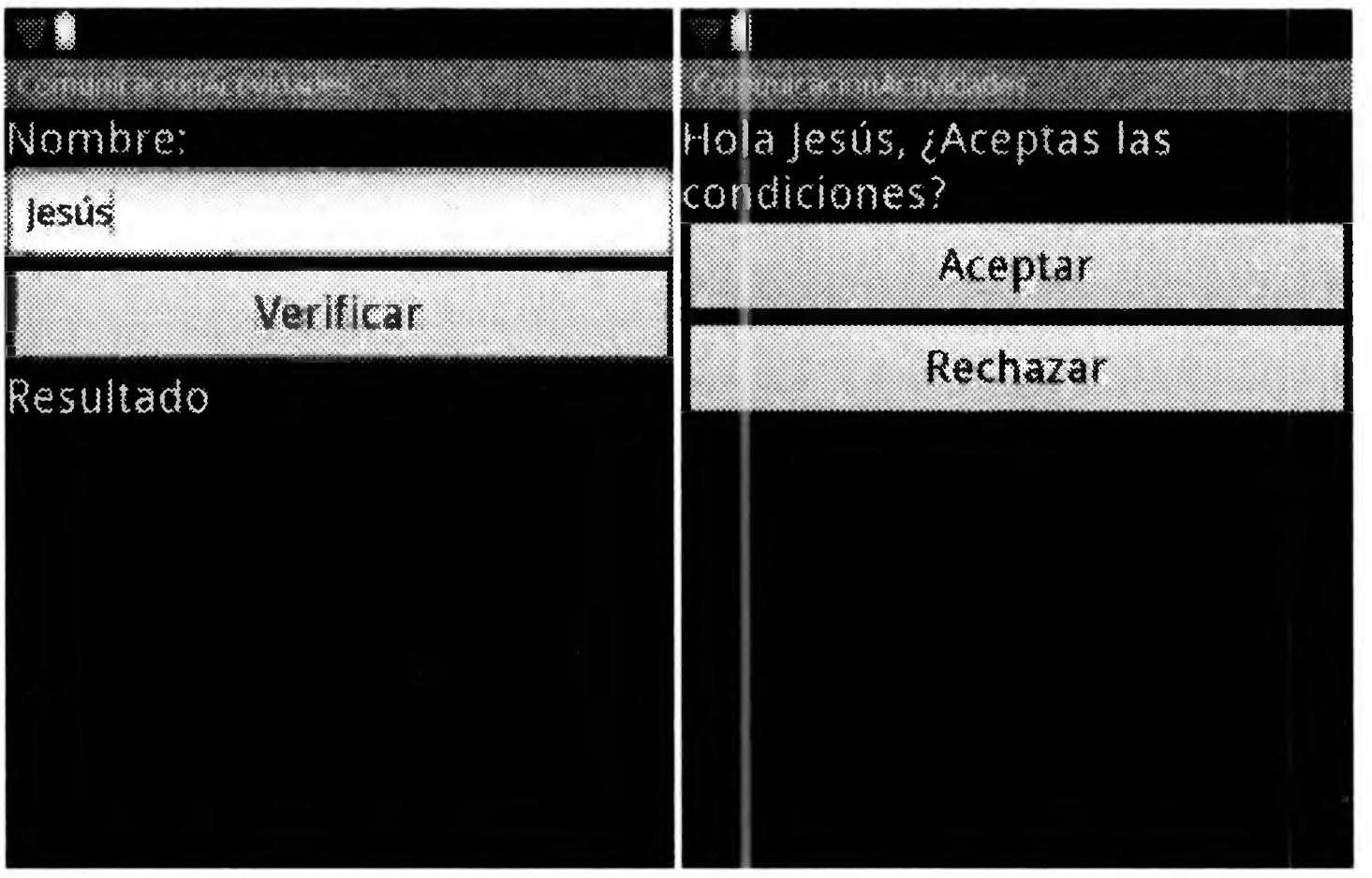


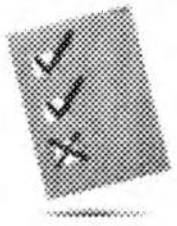
**Práctica:** *Comunicación entre actividades.*

Crea un nuevo proyecto con nombre *ComunicacionActividades*.

1. El *Layout* de la actividad inicial ha de ser similar al que se muestra a la izquierda de la imagen siguiente.
2. Introduce el código para que cuando se pulse el botón "Verificar" se arranque una segunda actividad. A esta actividad se le pasará como parámetro el nombre introducido en el `EditText`.
3. El *Layout* correspondiente a la segunda actividad se muestra a la derecha.
4. Al arrancar la actividad el texto del primer `TextView` ha de modificarse para que ponga "Hola "+nombre recibido+"¿Aceptas las condiciones?"
5. En esta actividad se podrán pulsar dos botones, de forma que se devuelva a la actividad principal el `String` "Aceptado" o "Rechazado", según el botón pulsado. Al pulsar cualquier botón se regresará a la actividad anterior.

6. En la actividad principal se modificará el texto del último `TextView` para que ponga "Resultado: Aceptado" o "Resultado: Rechazado", según lo recibido.



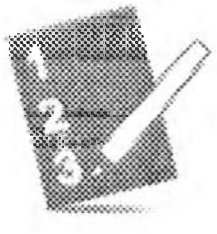



**Preguntas de repaso y reflexión:** *Comunicación entre actividades.*

## 3.3. Añadiendo un menú

Android permite asignar menús a las actividades que se despliegan pulsando el botón menú del terminal. Este tipo de menús resultan muy interesantes en dispositivos con pantallas pequeñas dado que no ocupan parte de la pantalla, es decir, están ocultos hasta que se pulsa la tecla de menú.

**Poli[Media]:** *Añadiendo un menú en Android.*



**Ejercicio paso a paso:** *Añadiendo un menú a Asteroides.*

Android permite asignar menús a las actividades que se despliegan pulsando el botón *menú* del terminal. Podemos asignar un menú de este tipo a nuestra actividad de forma muy sencilla.

1. Abre el proyecto Asteroide-

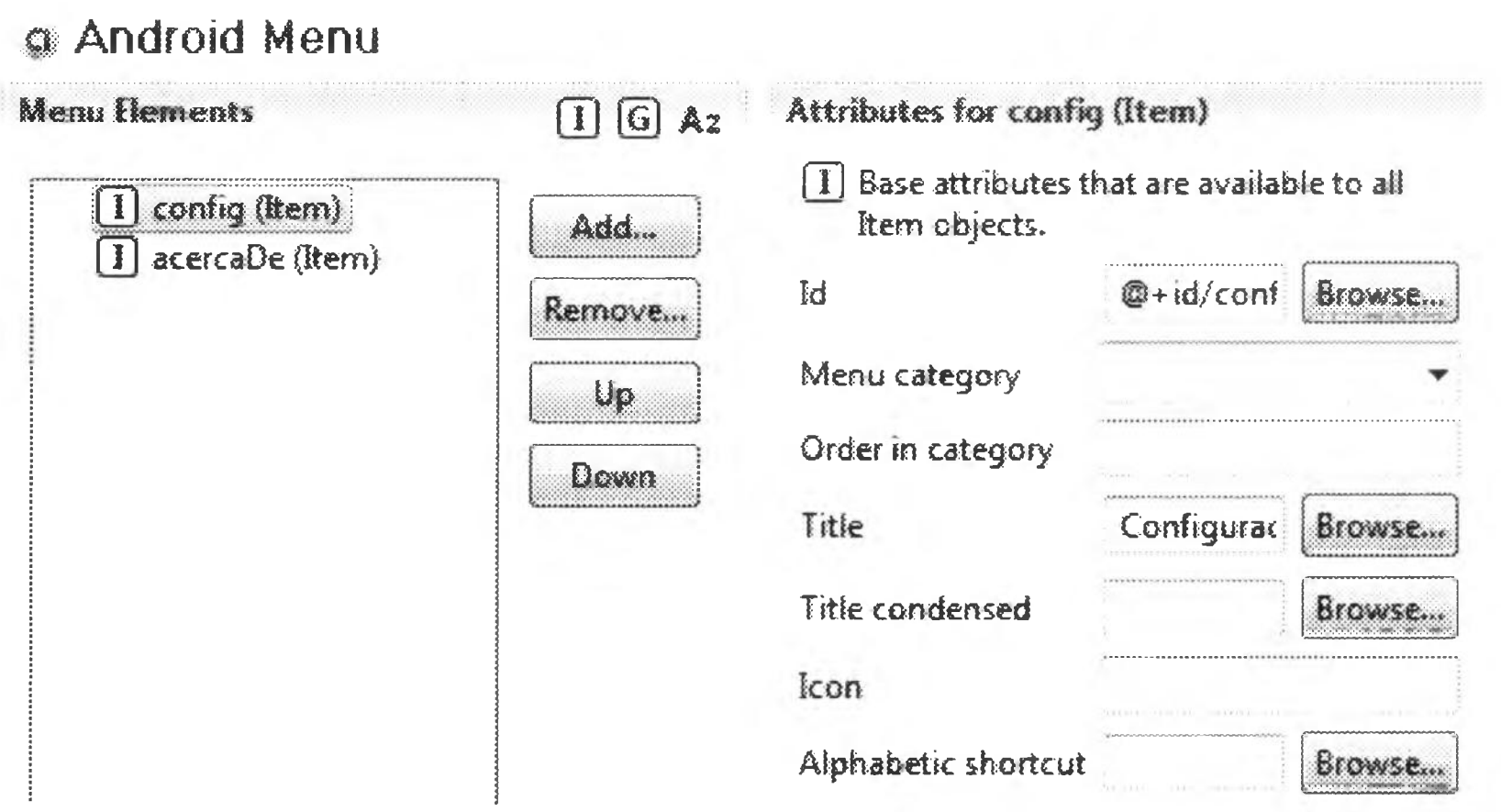
2. Pulsa con el botón derecho sobre esta carpeta de recursos y selecciona *New > Other... Selecciona Android > Android XML File*.

3. Introduce el valor *menu.xml* en el campo *File* y selecciona que queremos crear un fichero de tipo *menú*.

4. Aparecerá una lista vacía de elementos de menú. Pulsa en el botón *Add...*

5. Añade un elemento de tipo *Item*. Introduce el valor *@+id/config* en el campo *Id*, el valor *Configuración* en el campo *Title* y *@android:drawable/ ic_menu_preferences* en el campo *Icon*.

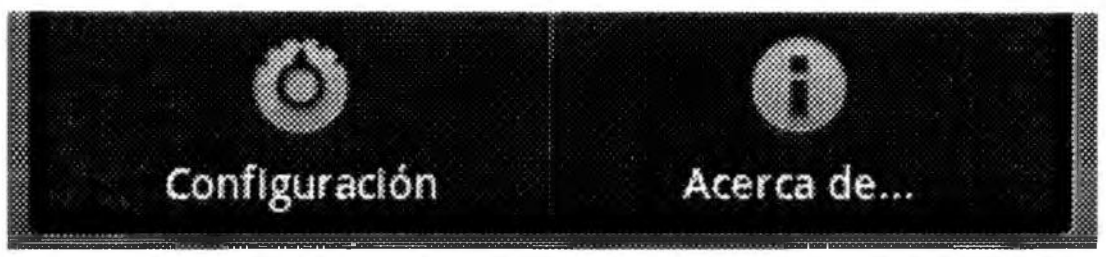
6. Repite el proceso para introducir un nuevo ítem con *Id* tomando el valor *@+id/acercaDe*, *Title* con el valor *Acerca de...* e *Icon* con el valor *@android:drawable/ic_menu_info_details*.




7. Pulsa la lengüeta .xml. El fichero xml resultante se muestra a continuación:

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
  <item android:title="Configuración"
        android:id="@+id/confg"
        android:icon="@android:drawable/ic_menu_preferences"/>
  <item android:title="Acerca de..."
        android:id="@+id/acercaDe"
        android:icon="@android:drawable/ic_menu_info_details"/>
</menu>
```

La apariencia de este menú se muestra a continuación:



8. Para activar el menú has de introducir el siguiente código al final de *Asteroides.java*:

```
@Override public boolean onCreateOptionsMenu(Menu menu) {
    super.onCreateOptionsMenu(menu);
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.menu, menu);
    return true; /** true -> el menú ya está visible */
}

@Override public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
    case R.id.acercaDe:
       lanzarAcercaDe();
       break;
    }
    return true; /** true -> consumimos el ítem, no se propaga*/
}
```

9. Ejecuta el programa y pulsa el botón *MENU* del terminal. En la parte inferior de la pantalla ha de aparecer los dos ítems de menú como se muestra en la figura anterior.

## 3.4. Creación y uso de iconos

En el apartado anterior hemos creado un menú que hacía uso de dos iconos. Se utilizaban iconos disponibles en el sistema Android, es decir, se han utilizado recursos del sistema Android. Otra alternativa es crear tus propios iconos y almacenarlos en la carpeta `res/drawable`.

En Android se utilizan diferentes tipos de iconos según su utilidad. La siguiente tabla muestra los más importantes:

| Tipo de iconos | Finalidad | Ejemplos |
|---|---|---|
| Lanzadores | Representa la aplicación en la pantalla principal del dispositivo. |  |
| Menú | Se muestran cuando presione el botón Menú. Cada actividad puede definir su propio menú. |   |
| Barra de acciones | A partir de la versión 3.0 con el aumento de pantalla de las tabletas las actividades pueden disponer de una barra de iconos siempre visible. |  |

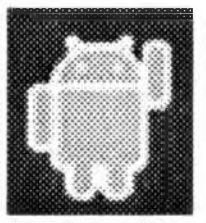
| Tipo de iconos | Finalidad | Ejemplos |
|---|---|---|
| Barra de estado | Barra con pequeños iconos que nos alertan de notificaciones . |  |
| Pestañas | Las pestañas (Tabs) nos permiten seleccionar entre varias vistas. |  |
| Otros | También es muy frecuente el uso de iconos en cuadros de dialogo y en ListView. |  |

***Tabla 1:*** *Tipos de iconos en Android.*

Si estás interesado en utilizar iconos disponibles en el sistema, encontrarás el inconveniente de no poder seleccionar el que te interese de forma sencilla. Recuerda como en el punto 6 del último ejercicio, cuando introducías el campo *Icon* en la definición de un ítem de menú, no disponía del botón *Browse...* para poder seleccionar el icono desde una lista. Para conocer los iconos disponibles te recomendamos que utilices el primer enlace de la lista mostrada a continuación.

Si por el contrario decides diseñar tus propios iconos has de tener en cuenta algunos aspectos. En primer lugar, recuerda que Android ha sido concebido para ser utilizado en dispositivos con gran variedad de densidades gráficas. Este rango puede ir desde 100 pixeles/pulgada (ldpi) hasta 340 pixeles/pulgada (xhdpi). Por lo tanto, resulta interesante que plantees tus diseños con una resolución como mínimo de 300 pixeles/pulgada. Cuando se represente en un dispositivo el sistema reajustará el tamaño del icono para que se ajuste al espacio disponible y, por supuesto, siempre es conveniente realizar una reducción en lugar de tener que ampliar. Ahora bien, en muchos casos este reajuste automático de tamaño puede no ser se satisfactorio (en el siguiente ejercicio se muestra un ejemplo). En estos casos, podremos hacer usos de los recursos alternativos y crear diferentes iconos para diferentes densidades gráficas. Para ayudarte en esta tarea puedes utilizar la herramienta *Android Asset Studio* que se incluye en la siguiente lista de enlaces.

En segundo lugar, tu aplicación va a ser ejecutada dentro de un sistema donde se utilizan ciertas guías de estilo. Si quieres que tus iconos no desentonen lee la documentación que se indica a continuación.

**Enlaces de interés:**

*Lista de recursos drawable:* En la documentación oficial de Android no aparece un listado con los recursos disponibles. Para ver gráficamente todos los recursos *drawabe* del sistema puedes acceder a la siguiente página:

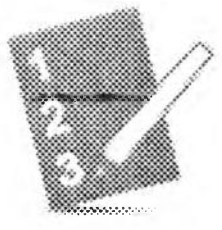
*http://androiddrawableexplorer.appspot.com/*

*Android Asset Studio:* Puedes crear tus propios iconos de forma sencilla utilizando la siguiente herramienta online:

*http://android-ui-utils.googlecode.com/hg/asset-studio/dist/index.html*

*Guía de estilo para iconos:* La siguiente página describe las guías de estilo para los iconos en las distintas versiones de Android:

*http://developer.android.com/guide/practices/ui_guidelines/icon_design.html*



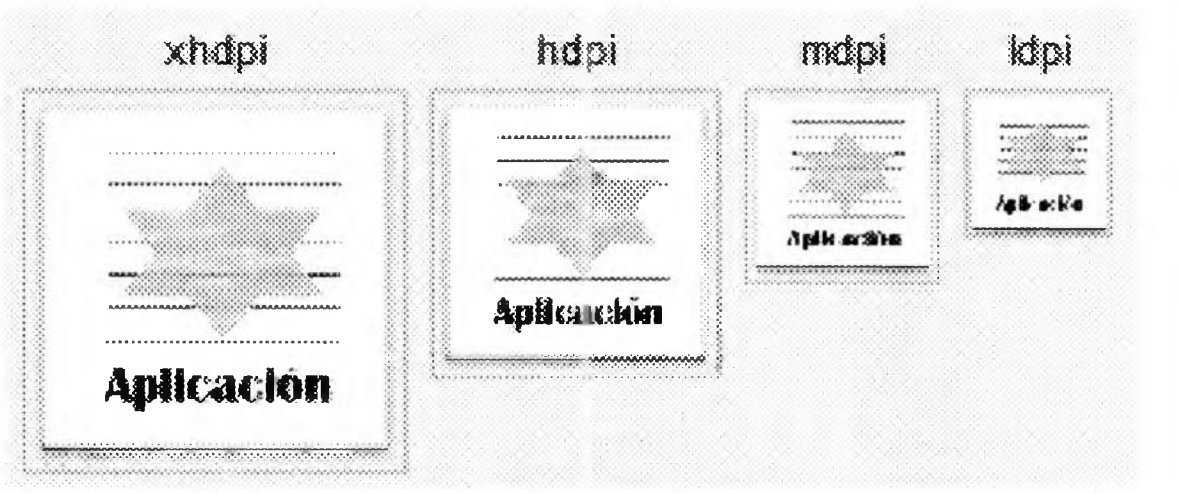
**Ejercicio paso a paso:** *Creación de iconos personalizados.*

Si te gusta el diseño o lo consideras interesante para tu formación, pueden resultar de gran utilidad las herramientas descritas anteriormente. Veamos un ejemplo práctico: El diseñador gráfico de nuestra empresa, nos acaba de pasar el icono asociado para iniciar a la aplicación que estamos diseñando (*launcher icon*).

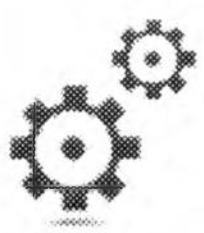

El problema es que no estamos seguros de que se va a ver correctamente para las diferentes densidades gráficas que podemos encontrar en los dispositivos Android. Para asegurarnos que esto sea así realizaríamos los siguientes pasos:

1. Descarga el gráfico anterior de www.androidcurso.com y llámale *icon.png*.
2. Accede a *Android Asset Studio*:

   http://android-ui-utils.googlecode.com/hg/asset-studio/dist/index.html
3. Selecciona la opción *Launcher icons*.
4. Pulsa sobre el botón *Image* y selecciona el fichero anterior.
5. En la sección *Output images* podrás previsualizar como quedarán las imágenes para las cuatro densidades gráficas de Android.



6. El resultado solo es aceptable para la opción xhdpi. En el resto de los casos ha desaparecido alguna línea o el texto no se lee. Va a ser imprescindible retocar estos iconos.
7. Descarga el fichero pulsando en Download .ZIP.
8. Retoca el icono hdpi para que se vean todas las líneas azules.
9. Retoca el icono mdpi y ldpi para que se pueda leer el texto "Aplicación".
10. Asigna los iconos retocados a las carpetas adecuadas (*res/drawable*) de una aplicación (por ejemplo la creada en la sección "Comunicación entre actividades").
11. Visualiza el resultado instalando la aplicación en diferentes dispositivos con diferentes densidades gráficas.



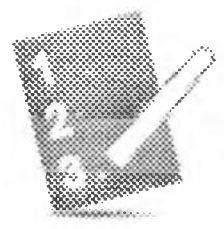
**Práctica:** *Creación de iconos pera Asteroides.*

1. Repite los pasos indicados en el ejercicio anterior para crear un icono que sea utilizada para iniciar la aplicación Asteroides.

## 3.5. Añadiendo preferencias de usuario

Android nos facilita la configuración de nuestro programa, al permitir añadir una lista de preferencias que el usuario podrá modificar. Las preferencias también pueden ser utilizadas para que tu aplicación almacene de forma permanente información. En el capítulo 9 se estudiará cómo realizar esta función.

Polli**[Media]**: *Añadir preferencias en Android.*



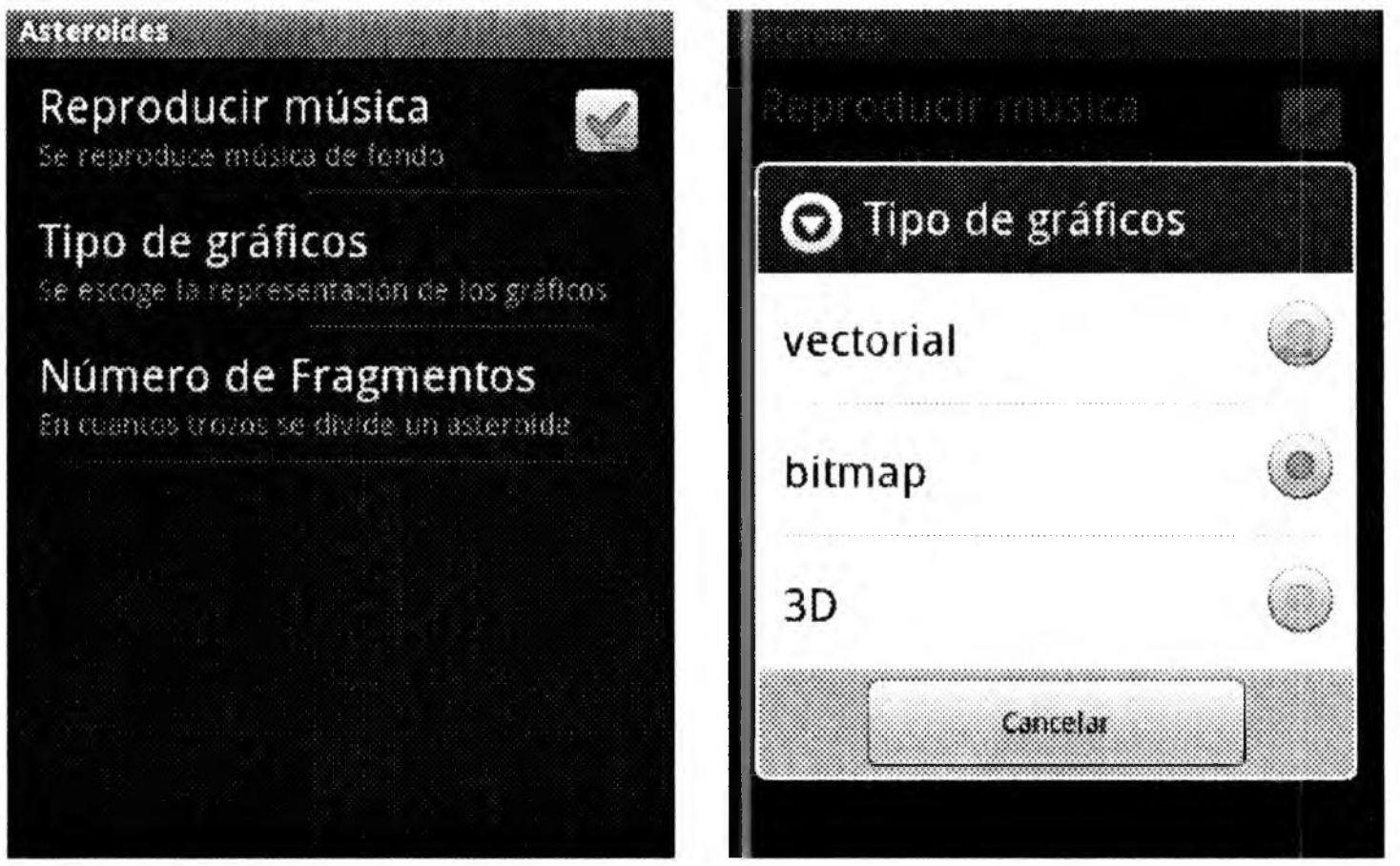
***Ejercicio paso a paso:*** *Añadiendo preferencias a Asteroides.*

1. Abre el proyecto Asteroides.
2. Pulsa con el botón derecho sobre la carpeta *res* y selecciona la opción *New > Other… Selecciona Android > Android XML File*.
3. Completa los siguientes campos *Resource Type: Preference*, *File: preferencias.xml*. Se creará el fichero *res/xml/preferencias.xml*.
4. Selecciona la lengüeta *preferencias.xml* para editar el fichero en xml.

5. Introduce el siguiente código:

```
<?xml version="1.0" encoding="utf-8"?>
<PreferenceScreen
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:key="preferencias_principal" >
  <CheckBoxPreference
    android:key="musica"
    android:title="Reproducir música"
    android:summary="Se reproduce música de fondo"/>
  <ListPreference
    android:key="graficos"
    android:title="Tipo de gráficos"
    android:summary="Se escoge la representación de gráficos"
    android:entries="@array/tiposGraficos"
    android:entryValues="@array/tiposGraficosValores"
    android:defaultValue="1"/>
  <EditTextPreference
    android:key="fragmentos"
    android:title="Número de Fragmentos"
    android:summary="En cuantos trozos se divide un asteroide"
    android:defaultValue="3"/>
</PreferenceScreen>
```

El resultado que obtendremos se muestra a continuación:




6. Para almacenar los valores del desplegable has de crear el fichero */res/values/arrays.xml* con el siguiente contenido:

```
<resources>
    <string-array name="tiposGraficos">
      <item>vectorial</item>
```

```
        <item>bitmap</item>
        <item>3D</item>
    </string-array>
    <string-array name="tiposGraficosValores">
        <item>0</item>
        <item>1</item>
        <item>2</item>
    </string-array>
</resources>
```

7. En lugar de trabajar directamente con xml. También puedes introducir las distintas preferencias utilizando la lengüeta *Structure*. Investiga esta opción.

8. Crea ahora una nueva clase *Preferencias.java* con el siguiente código:

```
package org.example.asteroides;

public class Preferencias extends PreferenceActivity {
    @Override protected void onCreate(Bundle
savedInstanceState) {
        super.onCreate(savedInstanceState);
        addPreferencesFromResource(R.xml.preferencias);
    }
}
```

**Nota sobre Java:** *Pulsa **Ctrl-Shift-O**, para que automáticamente se añadan los paquetes que faltan.*

9. No hay que olvidar registrar toda nueva actividad en `AndroidManifest.xml`.

10. Para activar la configuración desde la opción de menú añade el siguiente código en el fichero *Asteroides.java* en el método `onOptionsItemSelected()` dentro del *switch*.

```
case R.id.config:
    Intent i = new Intent(this, Preferencias.class);
    startActivity(i);
    break;
```

11. Arranca la aplicación y verifica que puedes lanzar las preferencias mediante la opción de menú correspondiente.

12. Modifica *Asteroides.java* para que se pueda acceder a las preferencias tanto desde el menú como desde el botón con texto *Configurar*.

### 3.5.1. Organizando preferencias

Cuando el número de preferencias es grande resulta interesante organizarlas de forma adecuada. Una opción consiste en dividirlas en varias pantallas. De forma que cuando se seleccione una opción en la primera pantalla, se abra una nueva pantalla de preferencias. Para anidar las preferencias usa el siguiente esquema:

```
<PreferenceScreen>
 <CheckBoxPreference …/>
 <EditTextPreference …/>
 …
 <PreferenceScreen android:title="Modo multijugador">
   <CheckBoxPreference …/>
   <EditTextPreference …/>
   …
 </PreferenceScreen>
</PreferenceScreen>
```

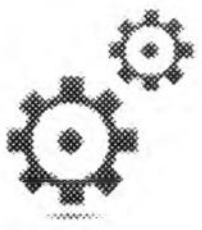
**Práctica:** *Organizando preferencias (I).*

1. Crea una nueva lista de preferencias `<PreferenceScreen>` dentro de la lista de preferencias del fichero *res/xml/preferencias.xml*.
2. Asígnale al parámetro `android:title` el valor "Modo multijugador".
3. Crea tres elementos dentro de esta lista: *Activar multijugador*, *Máximo de jugadores* y *Tipo de conexión*. Para este ultimo han de poder escogerse los valores: *Bluetooth*, *Wi-Fi* e *Internet*.

Otra opción para organizar las preferencias consiste en agruparlas por categorías. Con esta opción se visualizarán en la misma pantalla, pero separadas por grupos. Has de seguir el siguiente esquema:

```
<PreferenceScreen>
  <CheckBoxPreference …/>
  <EditTextPreference …/>
  …
  <PreferenceCategory android:title="Modo multijugador">
    <CheckBoxPreference …/>
    <EditTextPreference …/>
    …
  </PreferenceCategory>
</PreferenceScreen>
```

A continuación se representa la forma en la que Android muestra las categorías:

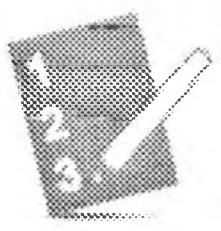



**Práctica:** *Organizando preferencias (II).*

1. Modifica la práctica anterior para que en lugar de mostrar las propiedades en dos pantallas, las muestre en una sola, tal y como se muestra en la imagen anterior.

## 3.5.2. Como se almacenan las preferencias de usuario

Si modificas un valor de una preferencia, este quedará almacenado de forma permanente en el dispositivo. Para conseguir esta persistencia Android utiliza un fichero xml que se almacena en el sistema de ficheros. Se utiliza la carpeta *shared_prefs* para almacenar los ficheros de preferencias. Dentro de esta carpeta, para almacenar las preferencias de usuario se utiliza el fichero *nombre.del.paquete_preferences.xml*, donde hay que remplazar *nombre.del.paquete* por el paquete que contiene la aplicación. Esto significa que solo puede haber unas preferencias de usuario por aplicación. Como se estudiará en el capítulo 9 pueden haber otros ficheros de preferencia, pero a diferencia de las preferencias de usuario no pueden ser editadas directamente por el usuario sino que hay que acceder a ellas por código.

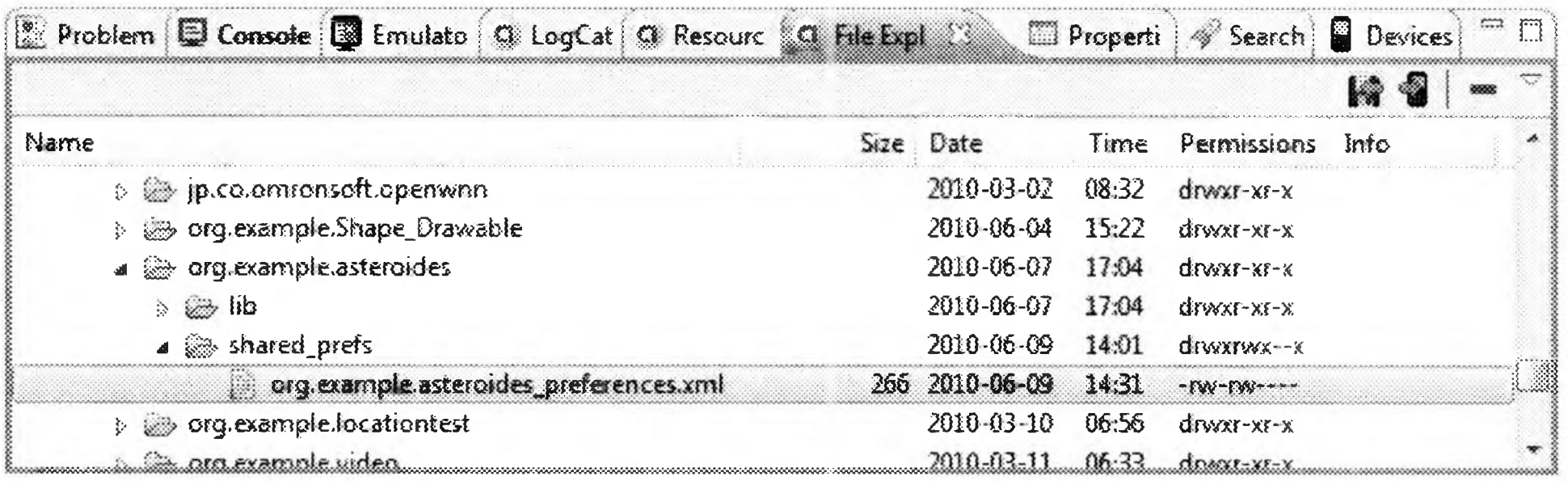


**Ejercicio paso a paso:** *Donde se almacenan las preferencias de usuario.*

Veamos donde se han almacenado las preferencias que acabamos de crear:

1. Para navegar por el sistema de ficheros de un dispositivo (tanto virtual como real) accede al menú *Windows* > *Show View* > *Others...* > *Android* > *File Expl.*

2. Busca el siguiente fichero: */data/data/org.examples.asteroide/shared_prefs/org.examples.asteroide_preferences.xml*.




3. Pulsa el botón del disquete para descargar el fichero.
4. Visualiza su contenido. Tiene que ser similar a:

```
<map>
    <boolean name="musica" value="true" />
    <string name="graficos">1</string>
    <string name="fragmentos">3</string>
</map>
```

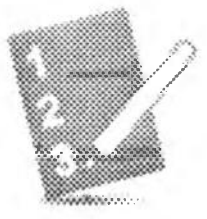
## 3.5.3. Accediendo a los valores de las preferencias

Por supuesto, será necesario acceder a los valores de las preferencias para alterar el funcionamiento de nuestra aplicación. El siguiente ejemplo nos muestra como realizarlo:

```
public void mostrarPreferencias(){
SharedPreferences pref = getSharedPreferences(
"org.example.asteroides_preferences", MODE_PRIVATE);
    String s = "música: " + pref.getBoolean("musica",true)
          +", gráficos: " + pref.getString("graficos","?");
    Toast.makeText(this, s, Toast.LENGTH_SHORT).show();
}
```

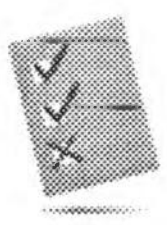
La función comienza creando el objeto `pref` de la clase `SharedPreferences` y le asigna un fichero de preferencias, abriéndolo en modo privado (solo nuestra aplicación tiene acceso). A continuación crea el `String s` y le asigna los valores de dos de las preferencias. Se utilizan los métodos `pref.getBoolean()` y `pref.getString()`, que disponen de dos parámetros: el valor de `key` que queremos buscar (`"musica"` y `"graficos"`) y el valor asignado por defecto en caso de no encontrar esta `key`.

Finalmente se visualiza el resultado utilizando la clase `Toast`. Los tres parámetros indicados son el contexto (nuestra actividad), el `String` a mostrar y el tiempo que se estará mostrando esta información.



**Ejercicio paso a paso:** *Accediendo a los valores de las preferencias.*

1. Copia la función anterior en la clase *Asteroides*.
2. Asígnala a un botón. Por ejemplo, *Jugar*.
3. Visualiza también el resto preferencias que hayas introducido.



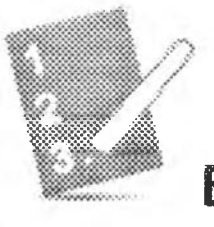
**Preguntas de repaso y reflexión:** *Menús y preferencias.*

## 3.6. Añadiendo una lista de puntaciones en Asteroides

Muchos videojuegos permiten recordar las puntuaciones de partidas anteriores, de esta forma, un jugador puede tratar de superar su propio récord o mejorar el de otros jugadores.

En el capítulo 9 estudiaremos varios métodos para que esta información se almacene permanentemente en el sistema. En el capítulo 10 estudiaremos como podemos compartirlo utilizando Internet. En este capítulo nos centraremos en representar esta lista de puntuaciones de forma atractiva utilizando la vista `ListView`.

Vamos a intentar que el mecanismo de acceso a esta lista de puntuaciones sea lo más independiente posible del método final escogido. Con este propósito, vamos a definir la interfaz `AlmacenPuntuaciones`.



**Ejercicio paso a paso:** *El interfaz AlmacenPuntuaciones.*

1. Abre la aplicación Asteroides.
2. Pulsa con el botón derecho sobre la carpeta de código (*src/com.example.asteroides*) y selecciona *New* > *Interface*.
3. En el campo *Name* introduce `AlmacenPuntuaciones` y pulsa *Finish*.
4. Introduce el código que se muestra a continuación:

```
public interface AlmacenPuntuaciones {
    public void guardarPuntuacion(int puntos,String nombre,long fecha);
  public Vector<String> listaPuntuaciones(int cantidad);
}
```

**Nota sobre Java:** *La interfaz es una clase abstracta pura, es decir una clase donde se indican los métodos pero no se implementa ninguno (en este caso se dice que los métodos son abstractos). Permite al programador de la clase establecer la estructura de esta (nombres de métodos, sus parámetros y tipos que retorna, pero no el código de cada método). Una interfaz también puede contener constantes, es decir campos de tipo static y final.*

Las diferentes clases que definamos para almacenar puntuaciones han de implementar esta interfaz. Como ves tiene dos métodos. El primero para guardar la puntuación de una partida, con los parámetros puntuación obtenida, nombre del jugador y fecha de la partida. La segunda es para obtener una lista de puntuaciones previamente almacenadas. El parámetro `cantidad` indica el número máximo de puntuaciones que ha de devolver.

5. Veamos a continuación una clase que utiliza esta interfaz. Para ello crea en el proyecto la clase `AlmacenPuntuacionesArray`.

6. Introduce el siguiente código:

```
public class AlmacenPuntuacionesArray implements AlmacenPuntuaciones{

    private Vector<String> puntuaciones;

    public AlmacenPuntuacionesArray() {
        puntuaciones = new Vector<String>();
        puntuaciones.add("123000 Pepito Domingez");
        puntuaciones.add("111000 Pedro Martinez");
        puntuaciones.add("011000 Paco Pérez");
    }

    @Override public void guardarPuntuacion(int puntos,
                                String nombre, long fecha) {
        puntuaciones.add(0, puntos + " " + nombre);
    }

    @Override public Vector<String> listaPuntuaciones(int cantidad) {
        return  puntuaciones;
    }
}
```

Esta clase almacena la lista de puntuaciones en un vector de String. Tiene el inconveniente de que al tratarse de una variable local, cada vez que se cierre la aplicación se perderán las puntuaciones. El constructor inicializa el array e introduce tres valores. La idea es que aunque todavía no esté programado el juego y no podamos jugar, tengamos ya algunas puntuaciones para poder representar una lista. El método

`guardarPuntuacion()` se limita a insertar en la primera posición del array un String con los puntos y el nombre. La fecha no es almacenada. El método `listaPuntuaciones()` devuelve el vector de String entero, sin tener en cuenta el parámetro `cantidad` que debería limitar el número de Strings devueltos.

7. En la actividad `Asteroides` tendrás que declarar una variable para almacenar las puntuaciones:

```
public static AlmacenPuntuaciones almacen =
                                    new AlmacenPuntuacionesArray();
```

***Nota sobre Java:*** *El modificador* `static` *permite compartir el valor de una variable entre todos los objetos de la clase. Es decir, aunque se creen varios objetos, solo existirá una única variable* `almacen` *compartida por todos los objetos. El modificador* `public` *permite acceder a la variable desde fuera de la clase. Por lo tanto, no será necesario crear métodos getters y setters. Para acceder a esta variable no tendremos más que escribir el nombre de la clase seguida de un punto y el nombre de la variable. Es decir* `Asteroides.almacen`.

8. Para que los jugadores puedan ver las últimas puntuaciones obtenidas, modifica el cuarto botón del *layout* `main.xml` para que en lugar del texto "Salir" se visualice "Puntuaciones". Para ello modifica los ficheros `res/values/strings`. También sería interesante que cambiaras el fichero `res/values-en/strings`.

9. Modifica el escuchador asociado al cuarto botón para que llame al método:

```
public void lanzarPuntuaciones(View view) {
  Intent i = new Intent(this, Puntuaciones.class);
  startActivity(i);
}
```

10. De momento no te permitirá ejecutar la aplicación. Hasta que en el siguiente apartado no creemos la actividad `Puntuaciones` no será posible.

## 3.7. La vista ListView

Una vista `ListView` visualiza una lista deslizable verticalmente de varios elementos, donde cada elemento puede definirse como un *Layout* .Su utilización es algo compleja, pero muy potente. Un ejemplo lo podemos ver en la siguiente figura:

El uso de un `ListView` conlleva los siguientes cuatro pasos:

- Diseñar un Layout que lo contenga al `ListView`.
- Diseñar un Layout individual que se repetirá en la lista.
- Implementar una actividad que lo visualice el Layout con el `ListView`.
- Personalizar cada una de los *Layouts* individuales según nuestros datos.

Veamos estos pasos con más detalle:

Para utilizar un `ListView` dentro de un *Layout* has de usar la siguiente estructura:

```
<FrameLayout>
  <ListView
      android:id="@android:id/list"
      ... />
  <TextView
      android:id="@android:id/empty"
      ... />
</FrameLayout>
```

Donde tenemos un `FrameLayout` que me permite visualizar dos posibles elementos, uno u otro, pero no los dos simultáneamente. El primero es el `ListView` que se visualizará cuando haya algún elemento en la lista. El segundo puede ser cualquier tipo de vista y se visualizará cuando no existan elementos en la lista. El sistema controla la visibilidad de forma automática, solo has de tener cuidado de identificar cada uno de los elementos con el valor exacto que se muestra.

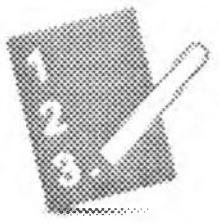
> ***NOTA:*** *Recuerda que para crear nuevos identificadores debes utilizar la expresión "`@+id/nombre_identificador`". El carácter @ significa que se trata de un identificador de recurso (es decir se definirá en la clase R.java). El carácter + significa que el recurso ha de ser creado en este momento. En este caso hemos utilizado identificadores que ya han sido definidos en el sistema (es decir `@android:` significa que es un recurso definido en la clase android.R.java).*

Una vez creado el *Layout* que contiene el `ListView` tendremos que visualizarlo en una actividad. Para este propósito utilizaremos un tipo de actividad especial, `ListActivity`.

También tendremos que indicar al sistema cada uno de los *Layaouts* individuales que contendrá el `ListView`. Esto lo haremos llamando al método `setListAdapter()`. Existen varias alternativas con diferentes grados de dificultad. Para una mejor comprensión iremos mostrando tres ejemplos de uso de `setListAdapter()`, de más sencillo a más complejo.

Las capturas anteriores muestran los tres ListView que vamos construir. El de la izquierda se limita a mostrar una lista de Strings. El del centro visualiza una lista de un *Layout* diseñado por nosotros. Aunque este Layout tiene varios componentes (una imagen y dos textos), solo cambiamos uno de los textos. En el último ejemplo cambiaremos también la imagen de cada elemento.

## 3.7.1. Un *ListView* que visualiza una lista de Strings



**Ejercicio paso a paso:** *Un ListView que visualiza una lista de Strings.*

1. El *Layout* que utilizaremos en Asteroides para mostrar las puntuaciones se llamará `puntuaciones.xml`. En el se incluye una vista `ListView`. Crea el *Layout* con el siguiente código:

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android
"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <TextView
     android:layout_width="fill_parent"
     android:layout_height="wrap_content"
     android:text="Puntuaciones"
     android:gravity="center"
     android:layout_margin="10px"
     android:textSize="10pt""/>
  <FrameLayout
     android:layout_width="fill_parent"
     android:layout_height="0dip"
     android:layout_weight="1">
     <ListView
      android:id="@android:id/list"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:drawSelectorOnTop="false" />
     <TextView
      android:id="@android:id/empty"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:text="No hay puntuaciones" />
    </FrameLayout>
</LinearLayout>
```

2. Necesitamos ahora crear la actividad `Puntuaciones` para visualizar el Layout anterior. Crea una nueva clase en tu proyecto e introduce el siguiente código:

```
public class Puntuaciones extends ListActivity {
    @Override public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.puntuaciones);
        setListAdapter(new ArrayAdapter<String>(this,
                android.R.layout.simple_list_item_1,
                Asteroides.almacen.listaPuntuaciones(10)));
    }
}
```

Toda actividad que vaya a visualizar un `ListView` ha de heredar de `ListActivity`. Además, ha de llamar al método `setListAdapter()` para indicar la lista de elementos a visualizar. Este método funciona de diferente manera según los parámetros que se indiquen (ver sobrecarga en Java). En el ejemplo se ha utilizado una de las posibilidades más sencillas, usar como parámetro un objeto de la clase `ArrayAdapter`. Un `ArrayAdapter` actúa como puente entre un ListView y el suministrador de información, en este caso, cada elemento de la lista se define como un Strings de un array. El constructor de esta clase tiene tres parámetros: El primer parámetro es un `Context` con información sobre el entorno de la aplicación. Utilizaremos como contexto la misma actividad que hace la llamada. El segundo parámetro es un *Layout*, utilizado para representar cada elemento de la lista. En este ejemplo, en lugar de definir uno nuevo, utilizaremos una ya definido en el sistema. El último parámetro es un array con los strings a mostrar. Para ello, llamamos al método `listaPuntuaciones()` que nos devuelve esta lista del objeto estático *almacen* de la clase `Asteroides`.

3. Recuerda que toda nueva actividad ha de ser registrada en `AndroidManifest.xml`.

4. Prueba si funcionan las modificaciones introducidas.

## 3.7.2. Un *ListView* que visualiza *Layouts* personalizados

Vamos a personalizar el `ListView` anterior para que cada elemento de la lista sea un *Layout* definido por nosotros. Para ello sigue los siguientes pasos:

**Ejercicio paso a paso:** *Un ListView que visualiza Layouts personalizados.*

1. Remplaza la clase anterior por:

```
public class Puntuaciones extends ListActivity {
    @Override public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.puntuaciones);
```

```
            setListAdapter(
            new ArrayAdapter<String>(this,
                R.layout.elemento_lista,
                R.id.titulo,
                Asteroides.almacen.listaPuntuaciones(10)));
    }
}
```

Como hemos explicado, la clase `ArrayAdapter<String>` permite insertar los datos desde un array de `String` en nuestro `ListView`. En este ejemplo se utiliza un constructor con cuatro parámetros:

`this`: es el contexto, con información sobre el entorno de la aplicación.

`R.layout.elemento_lista`: es una referencia de recurso a la vista que será utilizada repetidas veces para formar la lista. Se define a continuación.

`R.id.titulo`: identifica un id de la vista anterior que ha de ser un `TextView`. Su texto será reemplazado por el que se indica en el siguiente parámetro.

`Asteroides.almacen.listaPuntuaciones(10)`: vector de `String` con los textos que serán visualizados en cada uno de los `TextView`. Esta lista es obtenida accediendo a la clase `Asteroides` a su variable estática `almacen` llamando a su método `listaPuntuaciones()`.

2. Ahora hemos de definir el *Layout* que representará cada uno de los elementos de la lista. Crea el fichero *res/Layout/elemento_lista.xml* con el siguiente código:

```
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="?android:attr/listPreferredItemHeight">
    <ImageView android:id="@+id/icono"
    android:layout_width="?android:attr/listPreferredItemHeight"
        android:layout_height="fill_parent"
        android:layout_alignParentLeft="true"
        android:src="@drawable/asteroide2"/>
    <TextView android:id="@+id/titulo"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_toRightOf="@id/icono"
        android:layout_alignParentTop="true"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:singleLine="true" />
<TextView android:id="@+id/subtitulo"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
```

```
            android:text="Otro Texto"
            android:layout_toRightOf="@id/icono"
            android:layout_below="@id/titulo"
            android:layout_alignParentBottom="true"
            android:gravity="center"/>
</RelativeLayout>
```

Este *Layout* representa una imagen a la izquierda con dos textos a la derecha, uno de mayor tamaño en la parte superior. Para combinar estos elementos se ha escogido un `RelativeLayout`, donde el ancho se establece a partir de un parámetro de configuración del sistema `?android:attr/listPreferredItemHeight`. El primer elemento que contiene es un `ImageView` alineado a la izquierda. Su alto es la misma que el contenedor (`fill_parent`) mientras que el ancho se establece con el mismo parámetro que el alto del contenedor. Por lo tanto la imagen será cuadrada.

3. Las imágenes utilizadas en la aplicación Asteroides puedes descargarlas de www.androidcurso.com. Copia el fichero *asteriode2.png* a la carpeta *res/drawable*.

4. Ejecuta la aplicación y verifica el resultado.

## 3.7.3. Un *ListView* con nuestro propio adaptador

En el ejercicio anterior hemos visto como podíamos asociar un Layout definido por nosotros al `ListView` y personalizar uno de sus textos. Si queremos algo más adaptable, como cambiar las fotos o cambiar varios textos, tendremos que escribir nuestro propio adaptador de `ListView` extendiendo la clase `BaseAdapter`. En esta clase habrá que sobrescribir los siguientes cuatro métodos:

```
View getView(int position, View convertView, ViewGroup parent)
```

Este método ha de construir un nuevo objeto `View` con el *Layout* correspondiente a la posición `position` y devolverlo. Opcionalmente podemos partir de una vista base `convertView` para generar más rápido las vistas. El último parámetro corresponde al padre al que la vista va a ser añadida.

```
int getCount()
```

Devuelve el número de elementos de la lista.

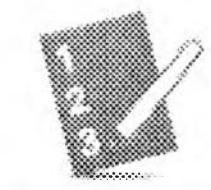
```
Object getItem(int position)
```

Devuelve el elemento en una determinada posición de la lista.

```
long getItemId(int position)
```

Devuelve el identificador de fila de una determinada posición de la lista.

Veamos un ejemplo:



**Ejercicio paso a paso:** *Un ListView con nuestro propio adaptador.*

1. Crea la clase *MiAdaptador.java* en el proyecto con el siguiente código:

```
public class MiAdaptador extends BaseAdapter {
    private final Activity actividad;
    private final Vector<String> lista;

    public MiAdaptador(Activity actividad, Vector<String> lista) {
        super();
        this.actividad = actividad;
        this.lista = lista;
    }

    @Override public View getView(int position, View convertView,
                                  ViewGroup parent) {
        LayoutInflater inflater = actividad.getLayoutInflater();
        View view = inflater.inflate(R.layout.elemento_lista, null,
                                     true);
        TextView textView =(TextView)view.findViewById(R.id.titulo);
        textView.setText(lista.elementAt(position));
        ImageView imageView=(ImageView)view.findViewById(R.id.icono);
        switch (Math.round((float)Math.random()*3)){
        case 0:
            imageView.setImageResource(R.drawable.asteroide1);
            break;
        case 1:
            imageView.setImageResource(R.drawable.asteroide2);
            break;
        default:
            imageView.setImageResource(R.drawable.asteroide3);
            break;
        }
        return view;
    }

    @Override
    public int getCount() {
        return lista.size();
    }

    @Override public Object getItem(int arg0) {
        return lista.elementAt(arg0);
    }

    @Override public long getItemId(int position) {
        return position;
    }
}
```

En el constructor de la clase se indica la actividad donde se ejecutará y la lista de datos a visualizar. El método más importante de esta clase es `getView()` el cual tiene que construir los diferentes *Layouts* que serán añadidos en la lista. Comenzamos construyendo un objeto `View` a partir del código xml definido en *elemento_lista.xml*. Este trabajo se realiza por medio de la clase `LayoutInflater`. Luego, se modifica el texto de uno de los `TextView` según el array que se pasó en el constructor. Finalmente, se obtiene un número al azar (`Math.round()`) y se asigna uno de los tres gráficos de forma aleatoria.

2. Remplaza en la clase `Puntuaciones` la llamada al constructor de `ArrayAdapter<String>` por:

```
setListAdapter(new MiAdaptador(this,
      Asteroides.almacen.listaPuntuaciones(10)));
```

Ejecuta la aplicación y verifica el resultado.

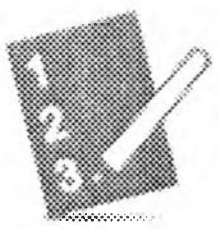
***NOTA:*** *En algunos casos el adaptador ha de trabajar con listas muy grandes o estas listas han de ser creadas desde un servidor. En estos casos es mejor ir solicitando la información a medida que se va representando. Un ejemplo se muestra en la aplicación `ApiDemos` descrita en el capítulo 1, en la actividad:*

`com.example.android.apis.view.List13`

## 3.7.4. Detectar una pulsación sobre un elemento de la lista

Un `ListView` puede tener diferentes componentes que nos permitan interaccionar con el usuario. Por ejemplo, cada elemento definido en `getView()` puede tener botones para diferentes acciones.

Hay un tipo de interacción muy sencilla de definir. La clase `ListActivity` tiene un método que es invocado cada vez que se pulsa sobre un elemento de la lista. El siguiente ejercicio ilustra cómo utilizarlo.



**Ejercicio paso a paso:** *Detectar una pulsación sobre un elemento de la lista.*

1. Añade el siguiente método a la clase *Puntuaciones.java*:

```
@Override protected void onListItemClick(ListView listView,
                          View view, int position, long id) {
   super.onListItemClick(listView, view, position, id);
   Object o = getListAdapter().getItem(position);
   Toast.makeText(this, "Selección: " + Integer.toString(position)
         +  " - " + o.toString(),Toast.LENGTH_LONG).show();
}
```

2. Ejecuta la aplicación y verifica el resultado.

## 3.8. Las Intenciones

Una *intención* representa la voluntad de realizar alguna acción o tarea; como realizar una llamada de teléfono o visualizar una página web. Una intención nos permite lanzar una actividad o servicio de nuestra aplicación o de una aplicación diferente. Existen dos tipos de intenciones:

- **Intenciones explícitas:** se indica exactamente el componente a lanzar. Su utilización típica es la de ir ejecutando los diferentes componentes internos de una aplicación. Por ejemplo, desde la actividad `Asteroides` lanzamos `AcercaDe` por medio de una intención explicita.
- **Intenciones implícitas:** pueden solicitar tareas abstractas, como "quiero tomar una foto" o "quiero enviar un mensaje". Además las intenciones se resuelven en tiempo de ejecución, de forma que el sistema mirará cuantas aplicaciones han registrado la posibilidad de ejecutar ese tipo de actividad. Si encuentra varias el sistema puede preguntar al usuario la actividad que prefiere utilizar.

Además, como se ha estudiado en el apartado *Comunicación entre actividades* las intenciones ofrecen un servicio de paso de mensajes que permite interconectar datos entre actividades. En concreto se utilizan intenciones cada vez que queramos:

- Lanzar una *actividad* (`startActivity()`).
- Lanzar un *servicio* (`startService()`).
- Lanzar un *anuncio de tipo broadcast* (`sendBroadcast()`).
- Comunicarnos con un *servicio* (`bindService()`).

En muchas ocasiones una *intención* no será inicializada por la aplicación, si no por el sistema, por ejemplo, cuando pedimos visualizar una página Web. En otras ocasiones será necesario que la aplicación inicialice su propia *intención*. Para ello se creará un objeto de la clase *Intent*.

Cuando se crea una Intención (es decir, se instancia un objeto de tipo `Intent`) esta contiene información de interés para que el sistema trate adecuadamente la intención o para el componente que recibe la intención. Podemos destacar la siguiente información:

**Nombre del componente:** Identificamos el componente que queremos lanzar con la intención. Hay que utilizar el nombre de clase totalmente cualificado (`org.example.asteroides.AcercaDe`) que queremos lanzar. El nombre del componente es opcional. En caso de no indicarse se utilizará otra información de la intención para obtener el componente a lanzar. A este tipo de intenciones se les conocía como intenciones explícitas.

**Acción:** Una cadena de caracteres donde indicamos la acción a realizar (o en caso de un *Receptor de anuncios (Broadcast receiver)* la acción que tuvo lugar y queremos reportar). La clase `Intent` define una serie de constantes para acciones genéricas que son listadas a continuación. No obstante, además de estas podemos definir nuevas acciones:

| Constante | Componente a lanzar | Acción |
|---|---|---|
| ACTION_CALL | Actividad | Inicializa una llamada de teléfono. |
| ACTION_EDIT | Actividad | Visualiza datos para que el usuario los edite. |
| ACTION_MAIN | Actividad | Arranca como actividad principal de una tarea. (sin datos de entrada y sin devolver datos) |
| ACTION_SYNC | Actividad | Sincroniza datos en un servidor con los datos de un dispositivo móvil. |
| ACTION_BATTERY_LOW | Receptor de anuncios | Advertencia de bacteria baja. |
| ACTION_HEADSET_PLUG | Receptor de anuncios | Los auriculares han sido conectados o desconectados. |
| ACTION_SCREEN_ON | Receptor de anuncios | La pantalla es activada. |
| ACTION_TIMEZONE_CHANGED | Receptor de anuncios | Se cambia la selección de zona horaria. |

***Tabla 2:*** *Acciones estándar de las Intenciones.*

También puedes definir tus propias acciones. En este caso has de indicar el paquete de tu aplicación como prefijo. Por ejemplo: `org.example.asteroides.MUESTRA_PUNTUACIONES`.

**Datos:** Referencia a los datos con los que trabajaremos. Hay que expresar estos datos por medio de una URI (el mismo concepto ampliamente utilizado en Internet). Ejemplos de URIs son: tel:963228525, http://www.androidcurso.com, content://call_log/calls... En muchos casos resulta importante saber el tipo de datos con el que se trabaja. Con este propósito se indica el tipo MIME asociado a la URI, es decir, se utiliza el mismo mecanismo que en Internet. Ejemplos de tipos MIME son `text/xml`, `image/jpeg`, `audio/mp3`,...

**Categoría:** Complementa a la acción. Indica información adicional sobre el tipo de componente que ha de ser lanzado. El número de categorías puede ser arbitrariamente ampliado. No obstante, en la clase `Intent` se definen una serie de categorías genéricas que podemos utilizar.

| Constante | Significado |
|---|---|
| CATEGORY_BROWSABLE | La actividad lanzada puede ser con seguridad invocada por el navegador para mostrar los datos referenciados por un enlace - por ejemplo, una imagen o un mensaje de correo electrónico. |

| Constante | Significado |
|---|---|
| CATEGORY_GADGET | La actividad puede ser embebida dentro de otra actividad que aloja gadgets. |
| CATEGORY_HOME | La actividad muestra la pantalla de inicio, la primera pantalla que ve el usuario cuando el dispositivo está encendido o cuando la tecla HOME es presionada. |
| CATEGORY_LAUNCHER | La actividad puede ser la actividad inicial de una tarea y se muestra en el lanzador de aplicaciones de nivel superior. |
| CATEGORY_PREFERENCE | La actividad a lanzar es un panel de preferencias. |

***Tabla 3:*** *Categorías estándar de las Intenciones.*

Una categoría suele utilizarse junto con una acción para aportar información adicional. Por ejemplo, indicaremos `ACTION_MAIN` a las actividades que pueden utilizarse como puntos de entrada de una aplicación. Indicaremos además `CATEGORY_LAUNCHER` en la actividad que será escogida por defecto para ser lanzada en primer lugar.

**Extras:** Información adicional que será recibida por el componente lanzado. Está formada por un conjunto de pares variable/valor. Estas colecciones de valores se almacenan en un objeto de la clase `Bundle`. Su utilización ha sido descrita en la sección *Comunicación entre actividades*. Recordemos cómo se introducían estos valores en un `Intent`.

```
intent.putExtra("usuario", "Pepito Perez");
intent.putExtra("edad", 27);
```

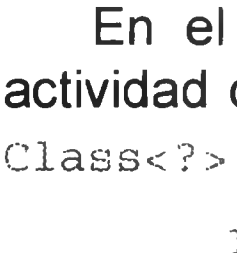
En el apartado *Creación de nuevas actividades* aprendimos a lanzar una actividad de forma explícita utilizando el constructor `Intent(Context contexto, Class<?> clase)`. Por ejemplo, para lanzar la actividad `AcercaDe` escribíamos:

```
Intent intent = new Intent(this, AcercaDe.class);
startActivity(intent);
```

Para lanzar una actividad de forma explícita podemos usar el constructor `Intent(String action, Uri uri)`. Por ejemplo:

```
Intent intent = new Intent(Intent.ACTION_DIAL,
                           URI.parse("tel:962849347");
startActivity(intent);
```

También se puede utilizar `startActivityForResult()` si esperamos que la actividad nos devuelva datos.



**Ejercicio paso a paso:** *Uso de intenciones implícita.*

1. Crea un nuevo proyecto con nombre *Intenciones*.

2. El *Layout* de la actividad inicial ha de estar formado por cinco botones, tal y como se muestra a continuación:


3. Abre la actividad principal e incorpora los siguientes métodos:

```
public void pgWeb(View view) {
    Intent intent = new Intent(Intent.ACTION_VIEW,
                    Uri.parse("http://www.androidcurso.com/"));
    startActivity(intent);
}

public void llamadaTelefono(View view) {
    Intent intent = new Intent(Intent.ACTION_CALL,
                    Uri.parse("tel:962849347"));
    startActivity(intent);
}

public void googleMaps(View view) {
    Intent intent = new Intent(Intent.ACTION_VIEW,
                    Uri.parse("geo:41.656313,-0.877351"));
    startActivity(intent);
}

public void tomarFoto(View view) {
    Intent intent = new Intent("android.media.action.IMAGE_CAPTURE");
    startActivity(intent);
}

public void mandarCorreo(View view) {
    Intent intent = new Intent(Intent.ACTION_SEND);
    intent.setType("text/plain");
    intent.putExtra(Intent.EXTRA_SUBJECT, "asunto");
    intent.putExtra(Intent.EXTRA_TEXT, "texto del correo");
    intent.putExtra(Intent.EXTRA_EMAIL,
                    new String[] { "jtomas@upv.es" });
    startActivity(intent);
}
```

4. Asocia el atributo `onClick` de cada uno de los botones al método correspondiente.

5. Si ejecutas esta aplicación en un emulador es muy posible que el botón *mandar Correo* o *Google Maps* no funcione. La razón es que no hay ninguna aplicación instalada en el emulador que sea capaz de realizar este tipo de acciones. Si tienes estos problemas, Abre el *AVD Manager* y crea un dispositivo virtual con Google API. Estos dispositivos incorporan además de las API de Android, algunas de las API de Google, como la de Google Maps (Estas API se estudiarán más adelante).

6. Ejecuta la aplicación en un terminal real. Observa como el botón *mandar Correo* te permite seleccionar entre diferentes aplicaciones con esta funcionalidad.

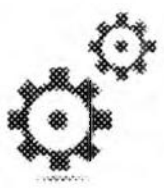

Este resultado puede variar en función de las aplicaciones instaladas.

***NOTA:*** *En el capítulo 7 se estudiará el tema de la seguridad. Aprenderás como has de solicitar el permiso adecuado si quieres que tu aplicación llame por teléfono o acceda a Internet. Cuando estas acciones no las realizas directamente, si no que las pides a través de una intención, no es tu aplicación quien las realiza y por tanto no has de pedir estos permisos.*

**Recursos adicionales:** *Tabla con Intenciones que podemos utilizar de aplicaciones Google.*

| Aplicación | URI | Acción | Resultado |
|---|---|---|---|
| Browser | http://*dirección_web*<br>https://*dirección_web* | VIEW | Abre una ventana de navegador con una URL. |
| | "" (cadena vacía)<br>http://*dirección_web*<br>https://*dirección_web* | WEB_ SEARCH | Abre el fichero en la ubicación indicada en el navegador. |

| Aplicación | URI | Acción | Resultado |
|---|---|---|---|
| Dialer | tel:*número_teléfono* | CALL | Realiza una llamada de teléfono. Los números validos se definen en IETF RFC 3966. Entre estos se incluye: tel:2125551212 y tel:(212)5551212. Necesitamos el premiso `android.permission.CALL_PHONE` |
| | tel:*número_teléfono*<br>voicemail: | DIAL | Introduce un número sin llegar a realizar la llamada. |
| Google Maps | geo:*latitud*,*longitud*<br>geo:*lat*,*long*?z=*zoom*<br>geo:0,0?q=*dirección*<br>geo:0,0?q=*búsqueda* | VIEW | Abre la aplicación Google Maps para una localización determinada. El campo z especifica el nivel de zoom. |
| Google Streetview | google.streetview:<br>cbll=*latitud*,*longitud*&<br>cbp=1,*yaw*,,*pitch*,*zoom*&<br>mz=*mapZoom* | VIEW | Abre la aplicación Street View para la ubicación dada. El esquema de URI se basa en la sintaxis que utiliza Google Maps. Solo el campo *cbll* es obligatorio. |



**Práctica:** *Uso de intenciones implícitas.*

1. Crea nuevos botones en la aplicación del ejercicio anterior y experimenta con otro tipo de acciones y URL. Puedes consultar la tabla anterior. A continuación tienes algunas propuestas:
2. Compara las acciones `VIEW` y `WEB_SEARCH`. ¿Encuentras alguna diferencia?
3. Compara las acciones `CALL` y `DIAL`. ¿Encuentras alguna diferencia?
4. Experimenta con Google Streetview

## 3.8.1. La etiqueta <intent-filter>

Cuando creamos una nueva actividade, servicio o receptor broadcast podemos informar al sistema que tipo de intenciones implícitas pueden ser resueltas con nuestro componente. Para conseguir esto utilizaremos un filtro de intenciones mediante la etiqueta `<intent-filter>` de *AndroidManifest.xml*.

Cuando desarrollamos una aplicación lo habitual es utilizar intenciones explicitas, que son resueltas utilizando el nombre de la clase. Por lo tanto, si vamos a llamar a nuestro componente de forma explicita, no tiene sentido crear un filtro de intenciones.

**Enlaces de interés:** *Aprender más sobre intenciones.*

- *Android Developers - Reference: Class Intent:*
  http://developer.android.com/reference/android/content/Intent.html
- *Android Developers - Dev Gide: Intents and Intent Filters:*
- http://developer.android.com/guide/topics/intents/intents-filters.html

# CAPÍTULO 4.
# Gráficos en Android

Android nos proporciona, a través de su API gráfica, una potente y variada colección de funciones que pueden cubrir prácticamente cualquier necesidad gráfica de una aplicación. Podemos destacar la manipulación de imágenes, gráficos vectoriales, animaciones, trabajo con texto o gráficos en 3D.

En este capítulo se introducen algunas de las características más significativas del API gráfico de Android. Nos centraremos en el estudio de las clases utilizadas para el desarrollo de gráficos en 2D. En el capítulo 2 se describió cómo se utilizaban las vistas como elementos constructivos para el diseño del interfaz de usuario. Disponíamos de una amplia paleta de vistas. No obstante, en muchas ocasiones va a ser interesante diseñar nuestras propias vistas. En este capítulo veremos cómo hacerlo.

Trataremos de aplicar lo aprendido en un ejemplo concreto, la representación de gráficos en Asteroides. Se utilizarán dos técnicas alternativas, los gráficos en mapa de bits y en formato vectorial. Al final del capítulo se describen las herramientas disponibles en Android para realizar animaciones. En concreto se describirán las animaciones Tween y las animaciones de propiedades. Por supuesto, resultaría imposible abarcar todas sus funciones para gráficos, por lo que se recomienda al lector que consulte la documentación de Android para obtener una descripción pormenorizada.

**Objetivos:**

- Enumerar las distintas APIS gráficas para 2D y 3D disponibles en Android.
- Describir cómo se utilizan las principales clases para gráficos en 2D (`Canvas`, `Paint` y `Path`).
- Introducir la clase `Drawable` y utilizar muchos de sus descendientes (`BitmapDrawable`, `GradienDrawable`, etc.).

- Estudiar cómo crear nuevas vistas y utilizarlas en distintas aplicaciones.
- Aplicar gran parte de lo aprendido en un ejemplo concreto: Asteroides.
- Describir las herramientas de Android para crear animaciones.

# 4.1. Clases para gráficos en Android

Antes de empezar a trabajar con gráficos conviene familiarizarse con las clases que nos van a permitir crear y manipular gráficos en Android. A continuación, se introducen algunas de estas clases:

**Poli[Media]**: *APIs para gráficos en Android.*

## 4.1.1. Canvas

La clase `Canvas` representa una superficie donde podemos dibujar. Dispone de una serie de métodos que nos permiten representar líneas, círculos, texto, etc. Para dibujar en un `Canvas` necesitaremos un pincel (`Paint`) donde definiremos el color, el grosor de trazo, la transparencia, etc. También podremos definir una matriz de 3x3 (`Matrix`) que nos permitirá transformar coordenadas aplicando una translación, escala o rotación. Otra opción consiste en definir un área conocida como *Clip*, de forma que los métodos de dibujo afecten solo a esta área.

Veamos, a continuación, algunos métodos de la clase `Canvas`. No se trata de un listado exhaustivo y muchos de estos métodos pueden trabajar con otros parámetros, por lo tanto se recomienda consultar la documentación del SDK para una información más detallada.

Para dibujar figuras geométricas:

```
drawCircle(float cx, float cy, float radio, Paint pincel)
drawOval(RectF ovalo, Paint pincel)
drawRect(RectF rect, Paint pincel)
drawPoint(float x, float y, Paint pincel)
drawPoints(float[] pts, Paint pincel)
```

Para dibujar líneas y arcos:

```
drawLine(float iniX, float iniY, float finX, float finY,
                                 Paint pincel)
drawLines(float[] puntos, Paint pincel)
drawArc(RectF ovalo, float iniAnglulo, float anglulo,
                                 boolean usarCentro, Paint pincel)
drawPath(Path trazo, Paint pincel)
```

Para dibujar texto:

```
drawText(String texto, float x, float y, Paint pincel)
```

```
drawTextOnPath(String texto, Path trazo, float desplazamHor,
                    float desplazamVert, Paint pincel)
drawPosText(String texto, float[] posicion, Paint pincel)
```

Para rellenar todo el `Canvas` (a no ser que se haya definido un *Clip*):

```
drawColor(int color)
drawARGB(int alfa, int rojo, int verde, int azul)
drawPaint(Paint pincel)
```

Para dibujar imagines:

```
drawBitmap(Bitmap bitmap, Matrix matriz, Paint pincel)
```

Si definimos un *Clip*, solo se dibujará en el área indicada:

```
boolean clipRect(RectF rectangulo)
boolean clipRegion(Region region)
boolean clipPath(Path trazo)
```

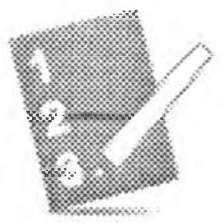
Definir una matriz de transformación (`Matrix`) nos permitirá transformar coordenadas aplicando una translación, escala o rotación:

```
setMatrix(Matrix matriz)
Matrix getMatrix()
concat(Matrix matriz)
translate(float despazX, float despazY)
scale(float escalaX, float escalaY)
rotate(float grados, float centroX, float centroY)
skew(float despazX, float despazY)
```

Para averiguar el tamaño del `Canvas`:

```
int getHeight()
int getWidth()
```

**Poli[Media]**: *La clase canvas en Android.*



**Ejercicio paso a paso**: *Creación de una vista personalizada.*

A continuación, se muestra un ejemplo donde se crea una vista que es dibujada por código por medio de un `Canvas`.

1. Crea un nuevo proyecto con nombre *EjemploGraficos*. El nombre del paquete debe ser `org.example.ejemplograficos`.
2. Remplaza el código de la actividad por:

```
public class EjemploGraficosActivity extends Activity {
```

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(new EjemploView(this));
}

public class EjemploView extends View {
    public EjemploView (Context context) {
        super(context);
    }
        @Override
        protected void onDraw(Canvas canvas) {
            //Dibujar aquí
        }
    }
}
```

Comienza con la creación de una `Activity`, pero en este caso, el objeto `View` que asociamos a la actividad mediante el método `setContentView()` no está definido mediante XML. Por el contrario, es creado mediante código, a partir del constructor de la clase `EjemploView`.

La clase `EjemploView` extiende `View`, modificando solo el método `onDraw()` responsable de dibujar la vista. A lo largo del capítulo, iremos viendo ejemplos de código que pueden ser escritos en este método.

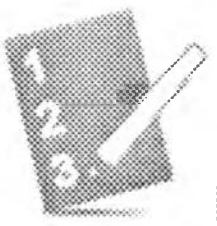
3. Si ejecutas la aplicación no se observará nada. Aprenderemos a dibujar en esta `Canvas` utilizando el objeto `Paint` descrito en la siguiente sección.

**Nota sobre Java:** *Siempre que en un ejemplo aparezca un error indicándote que no se puede resolver un tipo, pulsa **Ctrl-Shift-O** para añadir los import de forma automática.*

### 4.1.2. Paint

Como acabamos de ver, la mayoría de los métodos de la clase `Canvas` utilizan un parámetro de tipo `Paint`. Esta clase nos permite definir el color, estilo o grosor del trazado de un gráfico vectorial.

**Poli[Media]**: *La clase Paint en Android.*

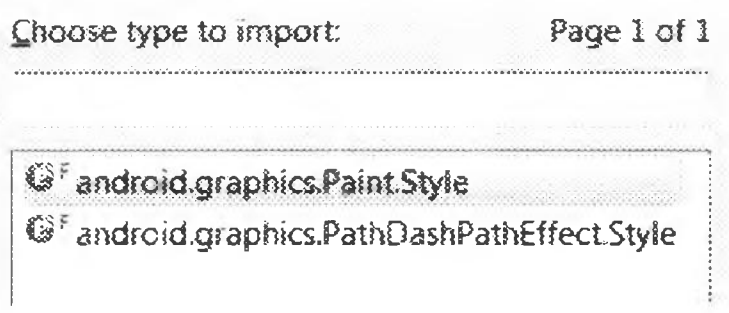


**Ejercicio paso a paso:** *Uso de la clase Paint.*

1. Escribe dentro de `OnDraw` del ejercicio anterior el código siguiente:

```
Paint pincel = new Paint();
pincel.setColor(Color.BLUE);
pincel.setStrokeWidth(8);
pincel.setStyle(Style.STROKE);
canvas.drawCircle(150, 150, 100, pincel);
```

**Nota sobre Java:** *Si pulsas **Ctrl-Shift-O** para añadir los `import` el sistema te advertirá que la clase `Style` está definida en dos paquetes diferentes:*




*Te preguntará cuál de los dos quieres importar. No suele resultar complicado resolver este problema si analizas el contexto en el que estás trabajando. En nuestro caso estamos utilizando la clase `Paint` por lo que la primera opción es la adecuada. Si alguna vez te equivocas en esta selección, lo normal es que aparezcan errores en el código. En este caso, debes deshacer y seleccionar la otra opción.*

2. Ejecuta la aplicación para ver el resultado.
3. Aprovechando la opción de autocompletar de Eclipse, prueba otros valores para `Color.` Y `Style.`
4. Prueba otros métodos de dibujo, como `drawLine()` o `drawPoint()`.

### *4.1.2.1. Definición de colores*

Android representa los colores utilizando enteros de 32 bits. Estos bits son divididos en 4 campos de 8 bits: alfa, rojo, verde y azul (ARGB, usando las iniciales en inglés). Al estar formado cada componente por 8 bits, podrá tomar 256 valores diferentes.

Los componentes rojo, verde y azul son utilizados para definir el color, mientras que el componente alfa define el grado de transparencia con respecto al fondo. Un valor de 255 significa un color opaco y, a medida que vayamos reduciendo el valor, el dibujo se irá haciendo transparente.

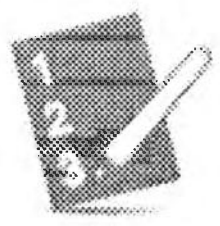
Para definir un color tenemos las siguientes opciones:

```
int color;
color = Color.BLUE;                          //Azul opaco
color = Color.argb(127, 0, 255, 0);          //Verde transparente
color = 0x7F00FF00;                          //Verde transparente
color = getResources().getColor(R.color.color_Circulo);
```

Para conseguir una adecuada separación entre programación y diseño, se recomienda utilizar la última opción. Es decir, no definir los colores directamente en código, sino utilizar el fichero de recursos `res/values/colors.xml`:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="color_circulo">#7fffff00</color>
</resources>
```

Como puedes observar los colores han de definirse en el fichero `colors.xml` mediante sus componentes ARGB en hexadecimal.



**Ejercicio paso a paso:** *Definición de colores.*

1. Modifica el ejercicio anterior, añadiendo al final de `OnDraw` el código siguiente:

```
pincel.setColor(Color.argb(127,255,0,0));
canvas.drawCircle(150, 250, 100, pincel);
```

2. Observa cómo el color rojo seleccionado es mezclado con el color de fondo. Prueba otros valores de alfa.

3. Reemplaza la primera línea que acabas de introducir por:

```
pincel.setColor(0x7FFF0000);
Observa cómo el resultado es idéntico.
```

4. Define en el fichero `res/values/colors.xml` un nuevo color utilizando el siguiente código:

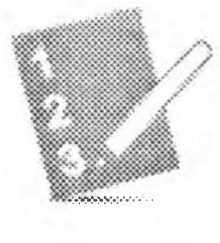
```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="color_circulo">#7fffff00</color>
</resources>
```

5. Modifica el ejemplo anterior para que se utilice este color definido en los recursos:

```
pincel.setColor(getResources().getColor(R.color.color_circulo));
```

### 4.1.3. Path

La clase `Path` permite definir un trazado a partir de segmentos de línea y curvas. Una vez definido puede ser dibujado con `canvas.drawPath(Path, Paint)`. Un `Path` también puede ser utilizado para dibujar un texto sobre el trazado marcado.

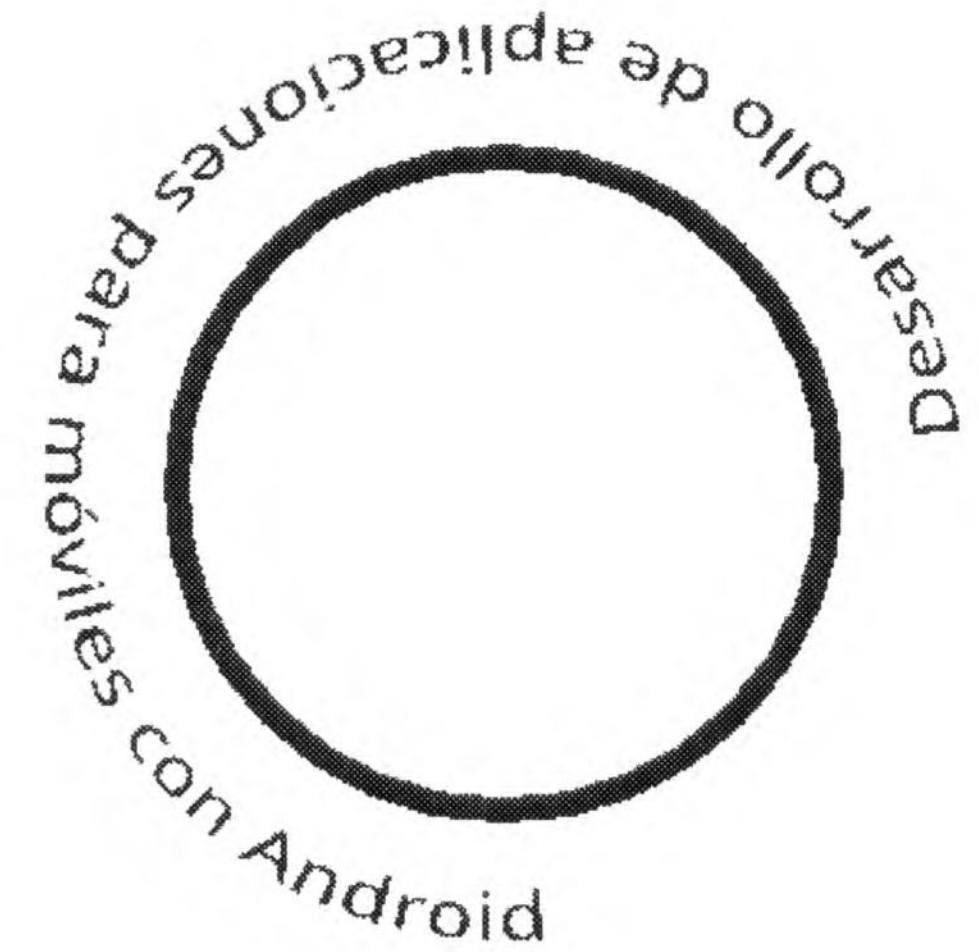


**Ejercicio paso a paso:** *Uso de la clase Path.*

1. Remplaza dentro de `OnDraw` del ejercicio anterior el código siguiente:

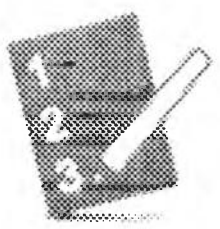
```
Path trazo = new Path();
trazo.addCircle(150, 150, 100, Direction.CCW);
canvas.drawColor(Color.WHITE);
Paint pincel = new Paint();
pincel.setColor(Color.BLUE);
pincel.setStrokeWidth(8);
pincel.setStyle(Style.STROKE);
canvas.drawPath(trazo, pincel);
pincel.setStrokeWidth(1);
pincel.setStyle(Style.FILL);
pincel.setTextSize(20);
pincel.setTypeface(Typeface.SANS_SERIF);
canvas.drawTextOnPath("Desarrollo de aplicaciones para
                móviles con Android", trazo, 10, 40, pincel);
```

2. El resultado obtenido ha de ser:




3. Modifica en la segunda línea el parámetro `Direction.CCW` (contrario a las agujas del reloj) por `Direction.CW` (a favor de las agujas del reloj). Observa el resultado.

4. Modifica los parámetros de `canvas.drawTextOnPath()` hasta que comprendas su significado.
5. ¿Podrías dibujar el texto a favor de las agujas del reloj, pero por fuera del círculo?

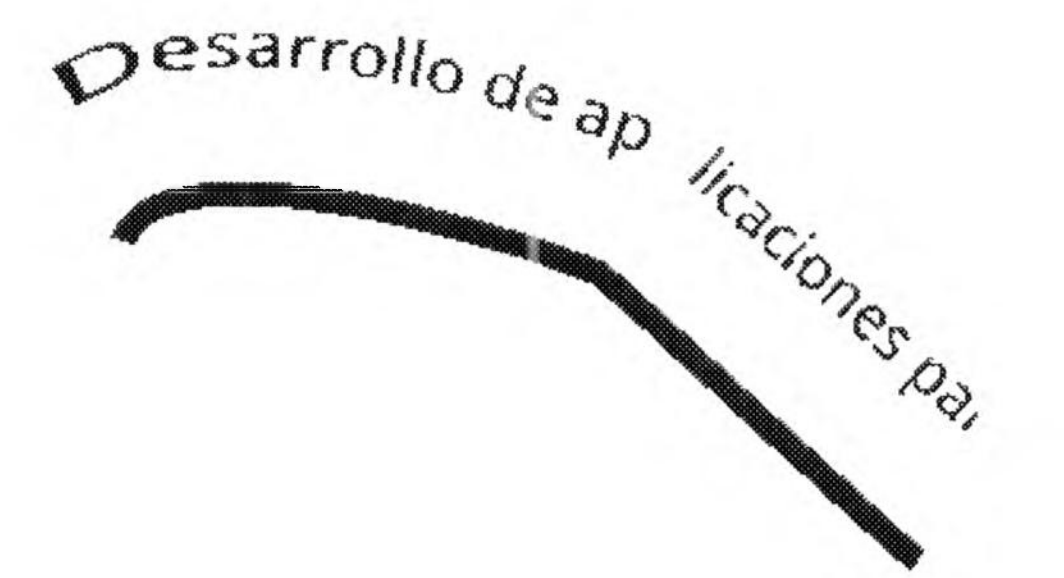


**Ejercicio paso a paso:** *Un ejemplo de Path más complejo.*

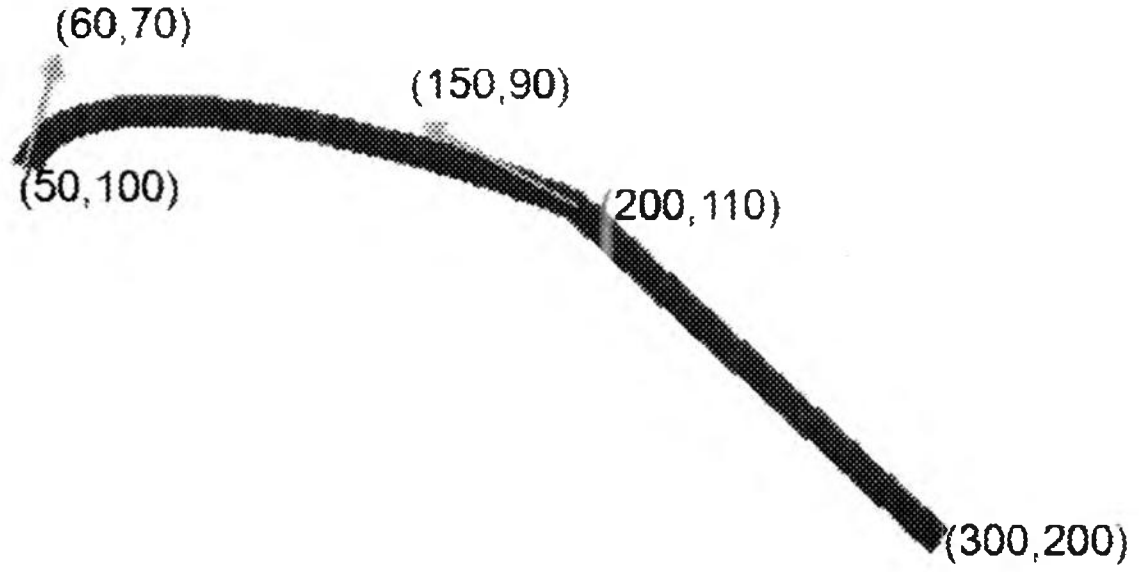
1. Remplaza las dos primeras líneas del ejemplo anterior por:

```
Path trazo = new Path();
trazo.moveTo(50, 100);
trazo.cubicTo(60,70, 150,90, 200,110);
trazo.lineTo(300,200);
```

2. El resultado obtenido debe ser similar a:




El trazo comienza desplazándose a las coordenadas (50,100). Luego introduce una curva cúbica o Bézier hasta la coordenada (200,110). Una curva Bézier introduce dos puntos de control, el primero (60,70) permite controlar cómo arranca la dirección del comienzo de la curva y el segundo (150,90) la dirección del final de la curva. Funciona de la siguiente manera, si trazas una recta desde el comienzo de la curva (50,100) hasta el primer punto de control (60,70), la curva se trazará tangencialmente a esta recta. Finalmente, se añade una línea desde las coordenadas (200,110) hasta (300,200).



3. ¿Por qué aparece una separación entre la "p" y la "l"? ¿Qué dos parámetros del siguiente gráfico tendríamos que modificar para que esta separación no estuviera? (*Solución: (150,90) => (150,65)*)

## 4.1.4. Drawable

La clase `Drawable` es una abstracción que representa "algo que puede ser dibujado". Esta clase se extiende para definir gran variedad de objetos gráficos más específicos. Muchos de ellos pueden ser definidos como recursos usando ficheros XML. Entre ellos tenemos los siguientes:

**BitmapDrawable**: Imagen basada en un fichero gráfico (PNG o JPG). Etiqueta XML `<bitmap>`.

**ShapeDrawable**: Permite realizar un gráfico a partir de primitivas vectoriales, como formas básicas (círculos, cuadrados,...) o trazados (`Path`). No puede ser definido mediante un fichero XML.

**LayerDrawable**: Contiene un *array* de `Drawable` que son visualizados según el orden del *array*. El índice mayor del *array* es el que se representa encima. Cada `Drawable` puede situarse en una posición determinada. Etiqueta XML `<layer-list>`.

**StateListDrawable**: Similar al anterior pero ahora podemos usar una máscara de bits y podemos seleccionar los objetos visibles. Etiqueta XML `<selector>`.

**GradienDrawable**: Degradado de color que puede ser usado en botones o fondos.

**TransitionDrawable**: Una extensión de `LayerDrawables` que permite un efecto de fundido entre la primera y la segunda capa. Para iniciar la transición hay que llamar a `startTransition(int tiempo)`. Para visualizar la primera capa hay que llamar a `resetTransition()`. Etiqueta XML `<transition>`.

**AnimationDrawable**: Permite crear animaciones frame a frame a partir de una serie de objetos `Drawable`. Etiqueta XML `<animation-list>`

También puede ser interesante que uses la clase `Drawable`, o uno de sus descendientes, como base para crear tus propias clases gráficas.

Además de ser dibujada, la clase `Drawable` proporciona una serie de mecanismos genéricos que permiten indicar cómo ha de ser dibujada. No todo `Drawable` ha de implementar todos los mecanismos. Veamos los más importantes:

El método `setBounds(x1,y1,x2,y2)` permite indicar el rectángulo donde ha de ser dibujado. Todo `Drawable` debe respetar el tamaño solicitado por el cliente, es decir, debe permitir el escalado. Podemos consultar el tamaño preferido de un `Drawable` mediante los métodos `getIntrinsicHeight()` y `getIntrinsicWidth()`.

El método `getPadding(Rect)` nos proporciona información sobre los márgenes recomendados para representar contenidos. Por ejemplo, un `Drawable` que intente ser un marco para un botón, debe devolver los márgenes correctos para localizar las etiquetas, u otros contenidos, en el interior del botón.

El método `setState(int[])` permite indicar al `Drawable` en qué estado ha de ser dibujado, por ejemplo "con foco", "seleccionado", etc. Algunos `Drawable` cambiarán su representación según este estado.

El método `setLevel(int)` permite implementar un controlador sencillo para indicar cómo se representará el `Drawable`. Por ejemplo, un nivel puede ser interpretado como una batería de niveles o un nivel de progreso. Algunos `Drawable` modificarán la imagen basándose en el nivel.

Un Drawable puede realizar animaciones al ser llamado desde el cliente a su método `Drawable.Callback`. Todo cliente debe implementar esta interfaz (vía `setCallback(Drawable.Callback)`). El sistema nos facilita realizar esta tarea de forma sencilla a través de `setBackgroundDrawable( Drawable)` e `ImageView`.

**Poli[Media]**: *La clase drawable en Android.*

Existen varias alternativas para crear una instancia de `Drawable`. Puedes crearla a partir de un fichero de imagen almacenado en los recursos del proyecto, también puedes crearla a partir del diseño basado en XML o puedes crearla a partir de código.

Veamos con más detalle algunas de las subclases de `Drawable`.

### *4.1.4.1. BitmapDrawable*

La forma más sencilla de añadir gráficos a tu aplicación es incluirlos en la carpeta `res/drawable` del proyecto. El SDK de Android soporta los formatos PNG, JPG y GIF. El formato aconsejado es PNG, aunque si el tipo de gráfico así lo recomienda, también puedes utilizar JPG. El formato GIF está desaconsejado.

Cada gráfico de esta carpeta es asociado a un ID de recurso. Por ejemplo, para el fichero `mi_imagen.png` se creará el ID `mi_imagen`. Este ID te permitirá hacer referencia al gráfico desde el código o desde XML.

**Ejercicio paso a paso**: *Dibujar un BitmapDrawable de los recursos.*

Busca en Internet un fichero gráfico en codificación *png* o *jpg* (los formatos gráficos usados por defecto en Android).

1. Renombra el fichero para que se llame *mi_imagen.png* o *mi_imagen.jpg* y arrastra a *res/drawable*.
2. Declara la variable `miImagen` en la clase `EjemploView` del ejercicio anterior:

```
private Drawable miImagen;
```

3. Escribe las siguientes tres líneas dentro del constructor de esta clase:

```
Resources res = context.getResources();
miImagen = res.getDrawable(R.drawable.mi_imagen);
miImagen.setBounds(30,30,200,200);
```

4. Escribe la siguiente línea en el método `onDraw`:

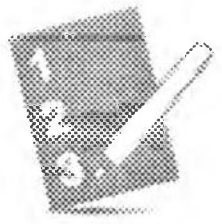
```
miImagen.draw(canvas);
```

### 4.1.4.2. GradienDrawable

También podemos definir en XML otro tipo de `Drawables` como `GradienDrawable`. Por ejemplo, el siguiente fichero define un degradado desde el color blanco (*FFFFFF*) a azul (*0000FF*):

```
<shape xmlns:android="http://schemas.android.com/apk/res/android">
    <gradient
        android:startColor="#FFFFFF"
        android:endColor="#0000FF"
        android:angle="270" />
</shape>
```

Este tipo de objetos gráficos es utilizado con frecuencia como fondo de botones o de pantalla. El parámetro `angle` indica la dirección del degradado. Solo se permiten los ángulos 0, 90, 180 y 270.



**Ejercicio paso a paso:** *Definir un GradienDrawable.*

1. Abre el proyecto Asteroides.
2. Crea el siguiente fichero *res/drawable/degradado.xml*:

```
<shape
xmlns:android="http://schemas.android.com/apk/res/android">
  <gradient
     android:startColor="#FFFFFF"
     android:endColor="#0000FF"
     android:angle="270" />
</shape>
```

3. Introduce la siguiente línea en el constructor de una vista, para conseguir que este *drawable* sea utilizado como fondo.

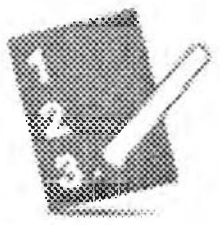
```
setBackgroundResource(R.drawable.degradado);
```

4. Resulta más conveniente definir el fondo de una vista en su *Layout* en XML en lugar de hacerlo por código. Comenta la línea introducida en el punto anterior e introduce el siguiente atributo en el *Layout main.xml*.

```
android:background="@drawable/degradado"
```

### *4.1.4.3. TransitionDrawable*

Un *TransitionDrawable* es una extensión de `LayerDrawables` que permite un efecto de fundido entre la primera y la segunda capa. Para iniciar la transición hay que llamar a `startTransition(int tiempo)`. Para volver a visualizar la primera capa hay que llamar a `resetTransition()`.



**Ejercicio paso a paso:** *Definir un TransitionDrawable.*

1. Crea un nuevo proyecto con nombre *Transition*.
2. Crea el siguiente recursos en *res/drawable/transicion.xml* con el código:

```
<?xml version="1.0" encoding="utf-8"?>
<transition  xmlns:android=
     "http://schemas.android.com/apk/res/android">
  <item android:drawable="@drawable/asteroide1"/>
  <item android:drawable="@drawable/asteroide3"/>
</transition>
```

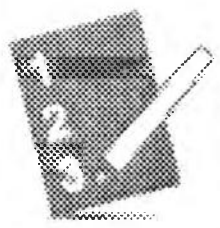
3. Remplaza en la actividad el método `onCreate` por el siguiente código:

```
@Override public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  ImageView image = new ImageView(this);
  setContentView(image);
  TransitionDrawable transition = (TransitionDrawable)
        getResources().getDrawable(R.drawable.transicion);
  image.setImageDrawable(transition);
  transition.startTransition(2000);
}
```

4. Si todo funciona correctamente, verás cómo la llamada a `transition.startTransition(2000)` provoca que la primera imagen se transforme, a lo largo de dos segundos, en la segunda imagen.

### *4.1.4.4. ShapeDrawable*

Cuando quieras crear dinámicamente un gráfico mediante primitivas vectoriales, una buena opción puede ser utilizar `ShapeDrawable`. Esta clase permite dibujar gráficos a partir de formas básicas. Un `ShapeDrawable` es una extensión de `Drawable`, por lo tanto, puedes utilizar todo lo que permite `Drawable`.



**Ejercicio paso a paso:** *Definir un ShapeDrawable.*

Veamos un ejemplo sobre cómo utilizar un objeto `ShapeDrawable` para crear una vista a medida.

1. Abre el proyecto *EjemploGraficos*.
2. En la clase `EjemploView` declara la siguiente variable:

```
private ShapeDrawable miImagen;
```

3. Añade las siguientes tres líneas dentro del constructor de esta clase:

```
miImagen  = new ShapeDrawable(new OvalShape());
miImagen.getPaint().setColor(0xff0000ff);
miImagen.setBounds(10, 10, 310, 60);
```

   En el constructor, un objeto `ShapeDrawable` es definido como un óvalo. Se le asigna un color y se definen sus fronteras.

4. Escribe la siguiente línea en el método `onDraw`:

```
miImagen.draw(canvas);
```

5. Ejecuta la aplicación y observa el resultado.

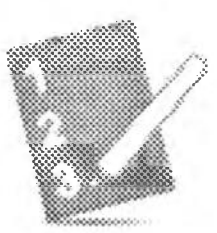
Toda nueva vista creada va a poder ser reutilizada de forma directa por cualquier programa. Incluso podrá ser usada desde un *Layout* en XML. No tienes más que escribir la clase `EjemploView` en un fichero independiente para que la clase sea visible y utilizar la siguiente etiqueta en tu `Layout` cuando quieras dibujar un óvalo.

```
<org.example.ejemplograficos.EjemploView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"/>
```

En el siguiente apartado trataremos de hacerlo.

### *4.1.4.5. AnimationDrawable*

Android nos proporciona varios mecanismos para crear animaciones gráficas. Una ventaja a destacar es que estas animaciones pueden ser creadas en ficheros XML. En este apartado veremos una de las animaciones más sencillas, las creadas a partir de una serie de fotogramas. Para ello utilizaremos la clase `AnimationDrawable`.



**Ejercicio paso a paso:** *Uso de AnimationDrawable*.

1. Crea un nuevo proyecto con nombre *Animacion*.
2. Crea el siguiente recurso *res/drawable/animacion.xml*.

```
<animation-list
    xmlns:android= "http://schemas.android.com/apk/res/android"
                android:oneshot= "false">
      <item android:drawable="@drawable/misil1"
            android:duration="200" />
      <item android:drawable="@drawable/misil2"
            android:duration="200" />
```

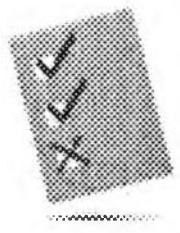
```
        <item android:drawable="@drawable/misil3"
              android:duration="200" />
</animation-list>
```

3. Copia los ficheros *misil1.png*, *misil2.png* y *misil3.png* a la carpeta *res/drawable*. Los encontrarás en www.androidcurso.com.

4. Remplaza el código de la actividad por:

```
public class AnimacionActivity extends Activity {
  AnimationDrawable animacion;

  @Override public void onCreate(Bundle savedInstanceState) {
     super.onCreate(savedInstanceState);
     animacion = (AnimationDrawable)getResources().getDrawable(
                                              R.drawable.animacion);
     ImageView vista = new ImageView(this);
     vista.setBackgroundColor(Color.WHITE);
     vista.setImageDrawable(animacion);
     vista.setOnClickListener(new OnClickListener() {
        public void onClick(View view) {
           animacion.start();
        }
     });
     setContentView(vista);
  }
}
```

En este ejemplo se comienza declarando un objeto `animacion` de la clase `AnimationDrawable`. Se inicializa utilizando el fichero XML incluido en los recursos. Se crea una nueva vista de la clase `ImageView` para ser representada por la actividad y se pone como imagen de esta vista `animacion`. Finalmente, se crea un escuchador de evento `onClick` para que cuando se pulse sobre la vista se ponga en marcha la animación.

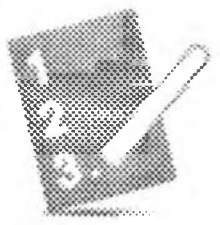


**Preguntas de repaso y reflexión:** *Las clases para gráficos en Android.*

## 4.2. Creación de una vista en un fichero independiente

Como hemos visto en los ejemplos anteriores, para poder dibujar en Android hemos tenido que crear una nueva clase `EjemploView`, descendiente de View. Esta clase era creada dentro de `EjemploGraficosActivity` por lo que solo podía ser utilizada desde esta clase. Va a resultar mucho más interesante crear esta clase en un fichero independiente. De esta forma, podremos utilizarla desde cualquier parte de

nuestro proyecto, o desde otros proyectos. Incluso estará disponible en la paleta de vistas del editor visual. El siguiente ejercicio, te permite verificar este concepto.



**Ejercicio paso a paso**: *Creación de una nueva Vista independiente.*

En este ejercicio vamos a poner la clase `EjemploView` en un fichero independiente para que pueda ser utilizada desde cualquier parte.

1. Abre el proyecto *EjemploGraficos*.
2. En la clase `EjemploGraficosActivity` selecciona todo el texto correspondiente a la definición de la clase `EjemploView`.
3. Pulsa con el botón derecho sobre *src/org.example/ejemplografico* y selecciona la opción *New/Class...*
4. Introduce como nombre de la clase `EjemploView`.
5. Pega el texto que has puesto en el portapapeles.
6. Verifica que, al ejecutar la aplicación, el resultado es idéntico al obtenido en la sección anterior.

En este apartado aprovechamos para introducir otros aspectos que tendrás que tener en cuenta para crear nuevas vistas. Cuando quieras crear una nueva vista, tendrás que extender la clase `View`, escribir un constructor y como mínimo sobreescribir los métodos `onSizeChanged()` y `onDraw()`. Es decir, tendrás que seguir el siguiente esquema:

```
public class MiVista extends View {
    public MiVista(Context context, AttributeSet attrs) {
        super(context, attrs);
        //Inicializa la vista
        //Ojo: Aún no se conocen sus dimensiones
    }

    @Override protected void onSizeChanged(int ancho, int alto,
            int ancho_anterior, int alto_anterior){
        //Te informan del ancho y el alto
    }
    @Override protected void onDraw(Canvas canvas) {
        //Dibuja aquí la vista
    }
}
```

Observa cómo el constructor utilizado tiene dos parámetros: El primero de tipo `Context` te permitirá acceder al contexto de aplicación, por ejemplo para utilizar recursos de esta aplicación. El segundo, de tipo `AttributeSet`, te permitirá acceder a los atributos de esta vista, cuando sea creada desde XML. El constructor es el

lugar adecuado para crear todos los componentes de tu vista, pero, cuidado, en este punto todavía no se conoce las dimensiones que tendrá.

Android realiza un proceso de varias pasadas para determinar el ancho y alto de cada vista dentro de un *Layout*. Cuando finalmente ha establecido las dimensiones de una vista llamará a su método `onSizeChanged()`. Nos indicará como parámetros el ancho y alto asignado. En caso de tratarse de un reajuste de tamaños, por ejemplo si una de las vistas del Layout desaparece y el resto tiene que ocupar su espacio, se nos pasará el ancho y alto anterior. Si es la primera vez que se llama al método estos parámetros valdrán 0.

El último método que siempre tendrás que reescribir es `onDraw()`. Es aquí donde tendrás que dibujar la vista.

**Ejercicio paso a paso**: *Una Vista que pueda ser diseñada desde XML.*

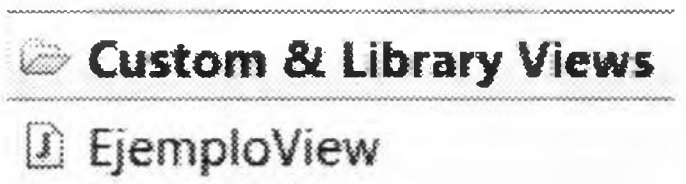
Vamos a modificar la vista anterior para que pueda ser utilizada usando un diseño en XML:

1. Modifica el código de la clase `EjemploView.` para que coincida con el siguiente (en negrita se resaltan las diferencias):

```
public class EjemploView extends View {
  private ShapeDrawable miImagen;

  public EjemploView(Context context, AttributeSet attrs) {
     super(context, attrs);
     miImagen = new ShapeDrawable(new OvalShape());
     miImagen.getPaint().setColor(0xff0000ff);
  }

  @Override protected void onSizeChanged(int ancho, int alto,
        int ancho_anterior, int alto_anterior){
     miImagen.setBounds(0, 0, ancho, alto);
  }

  @Override protected void onDraw(Canvas canvas) {
     miImagen.draw(canvas);
  }
}
```

   La primera diferencia es la utilización de un constructor con el parámetro `AttributeSet`. Este constructor es imprescindible si quieres poder definir la vista en XML. También se ha añadido el método `onSizeChanged`, para que el óvalo se dibuje siempre ajustado al tamaño de la vista.

2. Abre el fichero *main.xml* en modo *Graphical Layout*. En la paleta de vistas dentro de la última sección, observa cómo se ha añadido la nueva vista:

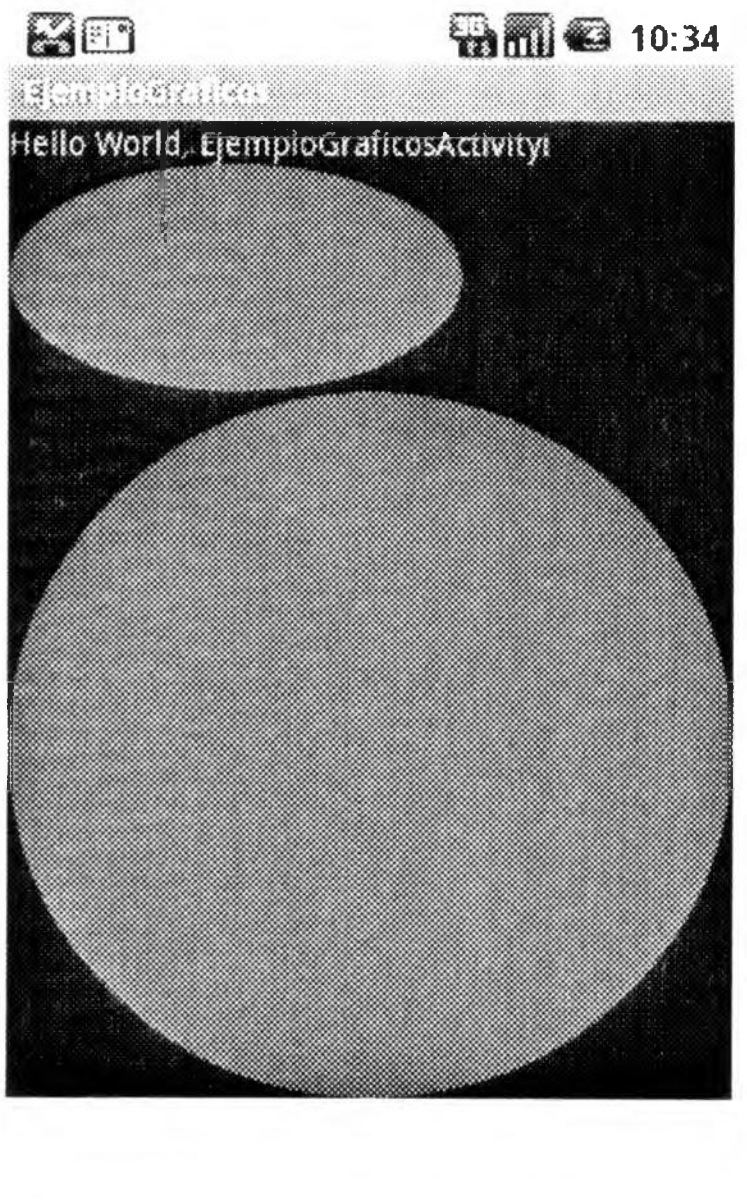



3. Arrastra dos elementos de este tipo al Layout.
4. Modifica las propiedades que definen el ancho y alto de estas vistas, tal y como se muestra a continuación:

```
<org.example.ejemplograficos.EjemploView
    android:id="@+id/ejemploView1"
    android:layout_width="200px"
    android:layout_height="100px" />

<org.example.ejemplograficos.EjemploView
    android:id="@+id/ejemploView2"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent" />
```

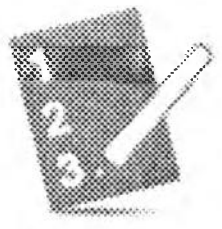
***NOTA:*** *No utilices el valor* `"wrap_content"`. *Nuestra vista no ha sido programada para soportar este tipo de valor.*

5. Ejecuta la aplicación. El resultado ha de ser similar a:




## 4.3. Creando la actividad principal de Asteroides

Una vez que conocemos los rudimentos que nos permiten utilizar gráficos en Android vamos a aplicarlos a nuestro ejemplo.



**Ejercicio paso a paso**: *Creando la actividad principal de Asteroides.*

Lo primero que necesitamos es crear una actividad que controle la pantalla del juego propiamente dicho. A esta actividad la llamaremos `Juego`.

1. Abre el proyecto Asteroides.
2. Lo primero que necesitamos es crear una actividad que controle la pantalla del juego propiamente dicho. Crea la clase `Juego` con el siguiente código.

```
public class Juego extends Activity {
  @Override public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.juego);
  }
}
```

3. Necesitaremos un *Layout* para la pantalla del juego. Crea el fichero `res/layout/juego.xml` con el siguiente contenido:

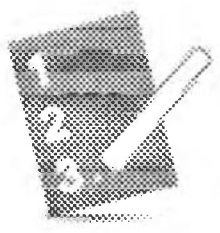
```
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent" >
    <org.example.asteroides.VistaJuego
     android:id="@+id/VistaJuego"
     android:layout_width="fill_parent"
     android:layout_height="fill_parent"
     android:focusable="true"
     android:background="@drawable/fondo" />
</LinearLayout>
```

Como puedes observar, cuando diseñamos un *Layout* en XML, no estamos limitados a escoger los elementos que tenemos en la paleta de vistas; vamos a utilizar vistas creadas por nosotros. En este ejemplo utilizaremos la vista `org.example.asteroides.VistaJuego` que será descrita más adelante. Va a ser esta vista la que lleve el peso de la aplicación.

4. Observa que se ha indicado el parámetro `android:background` asociado al recurso `@drawable/fondo`. Por lo tanto, tendremos que poner el recurso *fondo.jpg* en la carpeta correspondiente. Copia también los gráficos correspondientes a los asteroides, la nave y el misil a la carpeta `res/drawable`. Estos gráficos serán utilizados en los siguientes apartados. Los puedes encontrar en www.androidcurso.com.
5. De momento no podremos ejecutar la aplicación hasta haber definido la clase `VistaJuego`.

### 4.3.1. La clase Gráfico

El juego que estamos desarrollando va a desplazar muchos tipos de gráficos por pantalla: asteroides, nave, misiles, etc. Dado que el comportamiento de todos estos elementos es muy similar, con el fin de reutilizar el código y mejorar su comprensión, vamos a crear una clase que represente un gráfico a desplazar por pantalla. A esta nueva clase la llamaremos `Grafico` y presentará las siguientes características. El elemento a dibujar será representado en un objeto `Drawable`. Como hemos visto, esta clase presenta una gran versatilidad, lo que nos permitirá trabajar con gráficos en bitmap (`BitmapDrawable`), vectoriales (`ShapeDrawable`), gráficos con diferentes representaciones (`StateListDrawable`), gráficos animados (`AnimationDrawable`). Además, un `Grafico` dispondrá de posición, velocidad de desplazamiento, ángulo de rotación y velocidad de rotación. Para finalizar, un gráfico que salga por uno de los extremos de la pantalla reaparecerá por el extremo contrario, tal y como ocurría en el juego original de Asteroides.



**Ejercicio paso a paso:** *La clase Gráfico.*

1. Crea una nueva clase `Grafico` en el proyecto *Asteroides* y copia el siguiente código.

```
class Grafico {
  private Drawable drawable;    //Imagen que dibujaremos
private double posX, posY;    //Posición
private double incX, incY;    //Velocidad desplazamiento
private int angulo, rotacion;//Ángulo y velocidad rotación
private int ancho, alto;      //Dimensiones de la imagen
private int radioColision;    //Para determinar colisión
 //Donde dibujamos el gráfico (usada en view.ivalidate)
private View view;
 // Para determinar el espacio a borrar (view.ivalidate)
public static final int MAX_VELOCIDAD = 20;

public Grafico(View view, Drawable drawable){
   this.view = view;
   this.drawable = drawable;
   ancho = drawable.getIntrinsicWidth();
   alto = drawable.getIntrinsicHeight();
   radioColision = (alto+ancho)/4;
}
public void dibujaGrafico(Canvas canvas){
   canvas.save();
   int x=(int) (posX+ancho/2);
   int y=(int) (posY+alto/2);
   canvas.rotate((float) angulo,(float) x,(float) y);
```

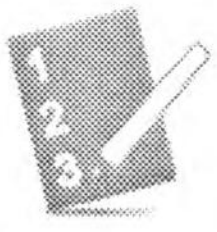
```
        drawable.setBounds((int)posX, (int)posY,
                           (int)posX+ancho, (int)posY+alto);
        drawable.draw(canvas);
        canvas.restore();
        int rInval = (int) Math.hypot(ancho,alto)/2 + MAX_VELOCIDAD;
        view.invalidate(x-rInval, y-rInval, x+rInval, y+rInval);
    }

    public void incrementaPos(double factor){
        posX+=incX * factor;
        // Si salimos de la pantalla, corregimos posición
        if(posX<-ancho/2) {posX=view.getWidth()-ancho/2;}
        if(posX>view.getWidth()-ancho/2) { posX=-ancho/2;}
        posY+=incY * factor;
        if(posY<-alto/2) {posY=view.getHeight()-alto/2;}
        if(posY>view.getHeight()-alto/2) {posY=-alto/2;}
        angulo += rotacion * factor; //Actualizamos ángulo
    }

    public double distancia(Grafico g) {
        return Math.hypot(posX-g.posY, posy-g.posX);
    }

    public boolean verificaColision(Grafico g) {
        return (distancia(g) < (radioColision+g.radioColision));
    }
}
```

2. Al principio de la clase hemos definido varios campos con el modificador `private`. Vamos a necesitar acceder a estos campos desde fuera de la clase, por lo que resulta necesario declarar los métodos *get* y *set* correspondientes. Vamos a realizar esta tarea de forma automática utilizando una herramienta de Eclipse. Sitúa el cursor al final de la clase (justo antes de la última llave) y pulsa con el botón derecho. Selecciona en el menú desplegable *Source/Generate Getters and Setters...* En la ventana de diálogo marca todos los campos (botón *Sellect All*) y pulsa *OK*. Eclipse hará el trabajo por nosotros.

**Nota sobre Java:** *En el tutorial Java Esencial / Encapsulamiento y visibilidad puedes aprender más sobre los métodos get y set. Lo encontrarás en www.androidcurso.com.*

## 4.3.2. La clase VistaJuego

Pasemos a describir la creación de `VistaJuego` que, como hemos indicado, es la responsable de la ejecución del juego. En una primera versión solo se representarán los asteroides de forma estática:



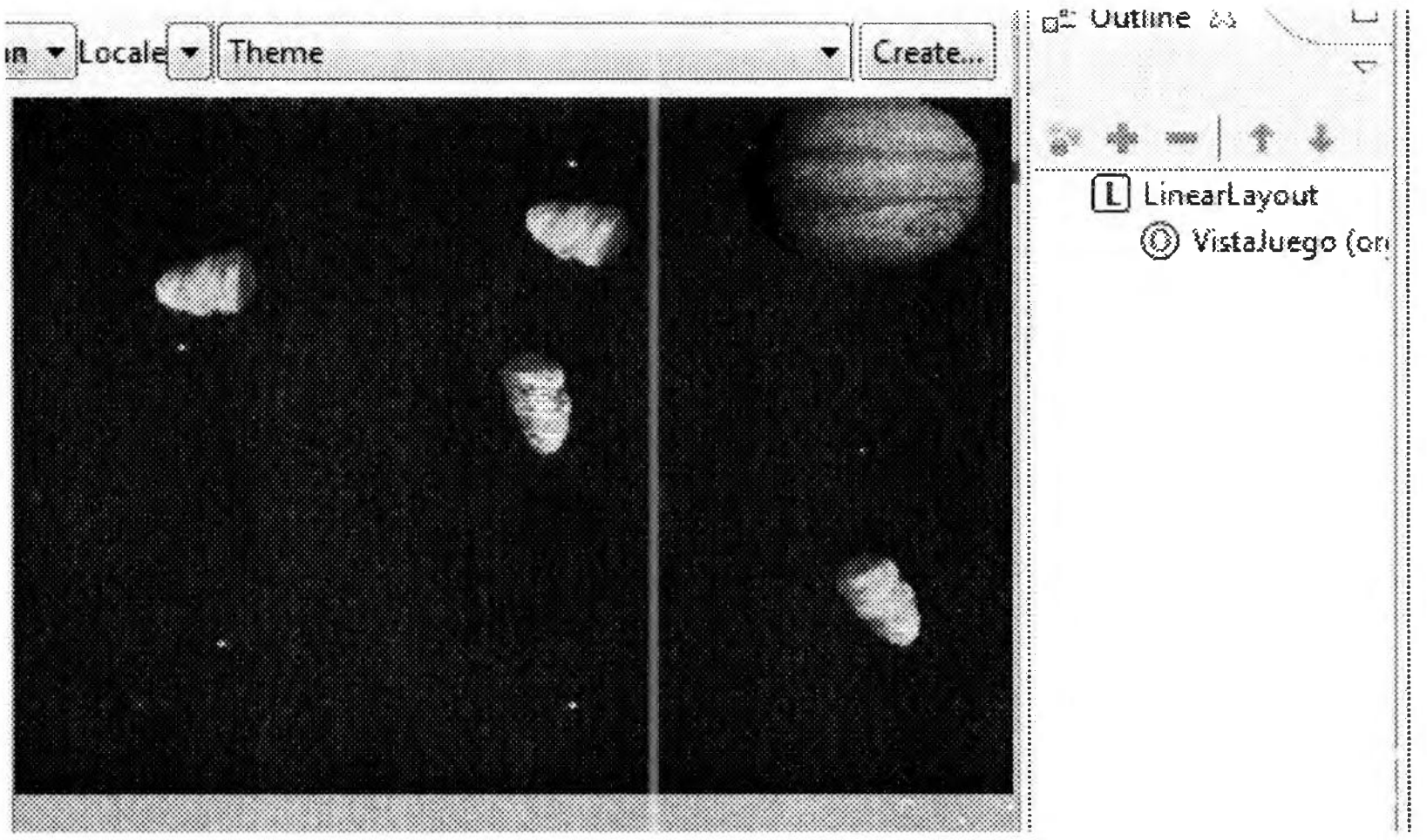
**Ejercicio paso a paso:** *La clase VistaJuego.*

1. Crea una nueva clase `VistaJuego` en el proyecto *Asteroides* y copia el siguiente código:

```
public class VistaJuego extends View {
  // //// ASTEROIDES //////
  private Vector<Grafico> Asteroides; // Vector con los Asteroides
  private int numAsteroides = 5; // Número inicial de asteroides
  private int numFragmentos = 3; // Fragmentos en que se divide

  public VistaJuego(Context context, AttributeSet attrs) {
     super(context, attrs);
     Drawable drawableNave, drawableAsteroide, drawableMisil;
     drawableAsteroide = context.getResources().getDrawable(
                                        R.drawable.asteroide1);
     Asteroides = new Vector<Grafico>();
     for (int i = 0; i < numAsteroides; i++) {
        Grafico asteroide = new Grafico(this, drawableAsteroide);
        asteroide.setIncY(Math.random() * 4 - 2);
        asteroide.setIncX(Math.random() * 4 - 2);
        asteroide.setAngulo((int) (Math.random() * 360));
        asteroide.setRotacion((int) (Math.random() * 8 - 4));
        Asteroides.add(asteroide);
     }
  }

  @Override protected void onSizeChanged(int ancho, int alto,
                                int ancho_anter, int alto_anter) {
     super.onSizeChanged(ancho, alto, ancho_anter, alto_anter);
     // Una vez que conocemos nuestro ancho y alto.
     for (Grafico asteroide: Asteroides) {
        asteroide.setPosX(Math.random()*
                                (ancho-asteroide.getAncho()));
        asteroide.setPosY(Math.random()*
                                (alto-asteroide.getAlto()));
     }
  }

  @Override protected void onDraw(Canvas canvas) {
     super.onDraw(canvas);
     for (Grafico asteroide: Asteroides) {
        asteroide.dibujaGrafico(canvas);
     }
  }
}
```

Como ves se han declarado tres métodos. En el constructor creamos los asteroides e inicializamos su velocidad, ángulo y rotación. Sin embargo, resulta imposible iniciar su posición, dado que no conocemos el alto y ancho de la pantalla. Esta información será conocida cuando se llame a `onSizeChanged()`. Observa cómo en esta función los asteroides están situados de forma aleatoria. El último método, `onDraw()`, es el más importante de la clase `View`, dado que es el responsable de dibujar la vista.

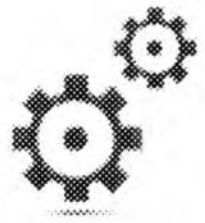
2. Hemos creado una vista personalizada. No tendría demasiado sentido, pero podrá ser utilizada en cualquier otra aplicación que desarrolles. Visualiza el *Layout* `juego.xml` y observa cómo la nueva vista se integra perfectamente en el entorno de desarrollo.




3. Registra la actividad Juego en *AndroidManifest.xml*.
4. Ejecuta la aplicación. Has de ver cinco asteroides repartidos al azar por la pantalla:

## 4.3.3. Introduciendo la nave en VistaJuego

El siguiente paso consiste en dibujar la nave que controlará el usuario para destruir los asteroides.



**Práctica:** *Introduciendo la nave en VistaJuego*

1. Declara las siguientes variables al comienzo de la clase `VistaJuego`:

```
// //// NAVE //////
private Grafico nave; // Gráfico de la nave
private int giroNave; // Incremento de dirección
private float aceleracionNave; // aumento de velocidad
// Incremento estándar de giro y aceleración
```

```
private static final int PASO_GIRO_NAVE = 5;
private static final float PASO_ACELERACION_NAVE = 0.5f;
```

Algunas de estas variables serán utilizadas en el siguiente capítulo.

2. En el constructor de la clase instancia la variable `drawableNave` de forma similar a como se ha hecho en `drawableAsteroide`.

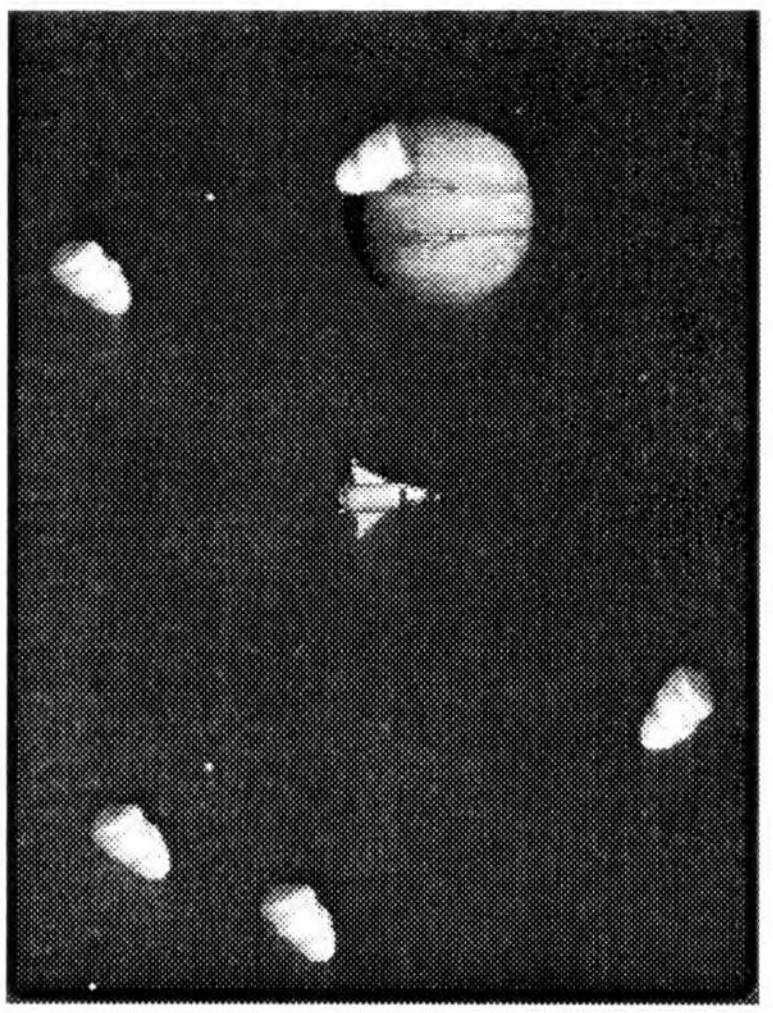
3. Inicializa también en el constructor la variable `nave` de la siguiente forma:

```
nave = new Grafico(this, drawableNave);
```

4. En el método `onSiceChange()` posiciona la nave justo en el centro de la vista.

5. En el método `onDraw()` dibuja la nave en el `Canvas`.

6. Ejecuta la aplicación. La nave ha de aparecer centrada:



7. Si cuando situamos los asteroides, alguno coincide con la posición de la nave, el jugador no tendrá ninguna opción de sobrevivir. Sería más interesante asegurarnos de que al posicionar los asteroides estos se encuentran a una distancia adecuada de la nave y, en caso contrario, tratar de obtener otra posición. Para conseguirlo puedes utilizar el siguiente código en sustitución de las dos líneas `asteroide.setPosX(...)` y `asteroide.setPosY(...)`.

```
do {
  asteroide.setPosX(Math.random()*(ancho-asteroide.getAncho()));
  asteroide.setPosY(Math.random()*(alto-asteroide.getAlto()));
} while(asteroide.distancia(nave) < (ancho+alto)/5);
```

**Ejercicio paso a paso:** *Evitando que VistaJuego cambie su representación con el dispositivo en horizontal y en vertical.*

1. Ejecuta la aplicación.

2. Cambia de orientación la pantalla del dispositivo. En el emulador se consigue pulsando la tecla *Ctrl-F11*.

3. Observa cómo cada vez se reinicializa la vista, regenerando los asteroides. Esto nos impediría jugar de forma adecuada. Para solucionarlo edita *AndroidManifet.xml*. En la lengüeta *Application* selecciona la actividad Juego. En los parámetros de la derecha selecciona en *Screen orientation: landscape*.

4. Ejecuta de nuevo la aplicación. Observa cómo la actividad Juego será siempre representada en modo horizontal, de forma independiente a la posición del teléfono.

5. Abre de nuevo las propiedades de la actividad Juego. En *Theme* selecciona el valor `@android:style/Theme.NoTitleBar.Fullscreen`. Este tema visualizará la vista ocupando toda la pantalla, sin la barra de notificaciones ni el nombre de la aplicación.

6. Si en *Theme* pulsas el botón *Browse...* y seleccionas el botón circular *System Resources* puedes ver una lista de estilos definidos en el sistema.

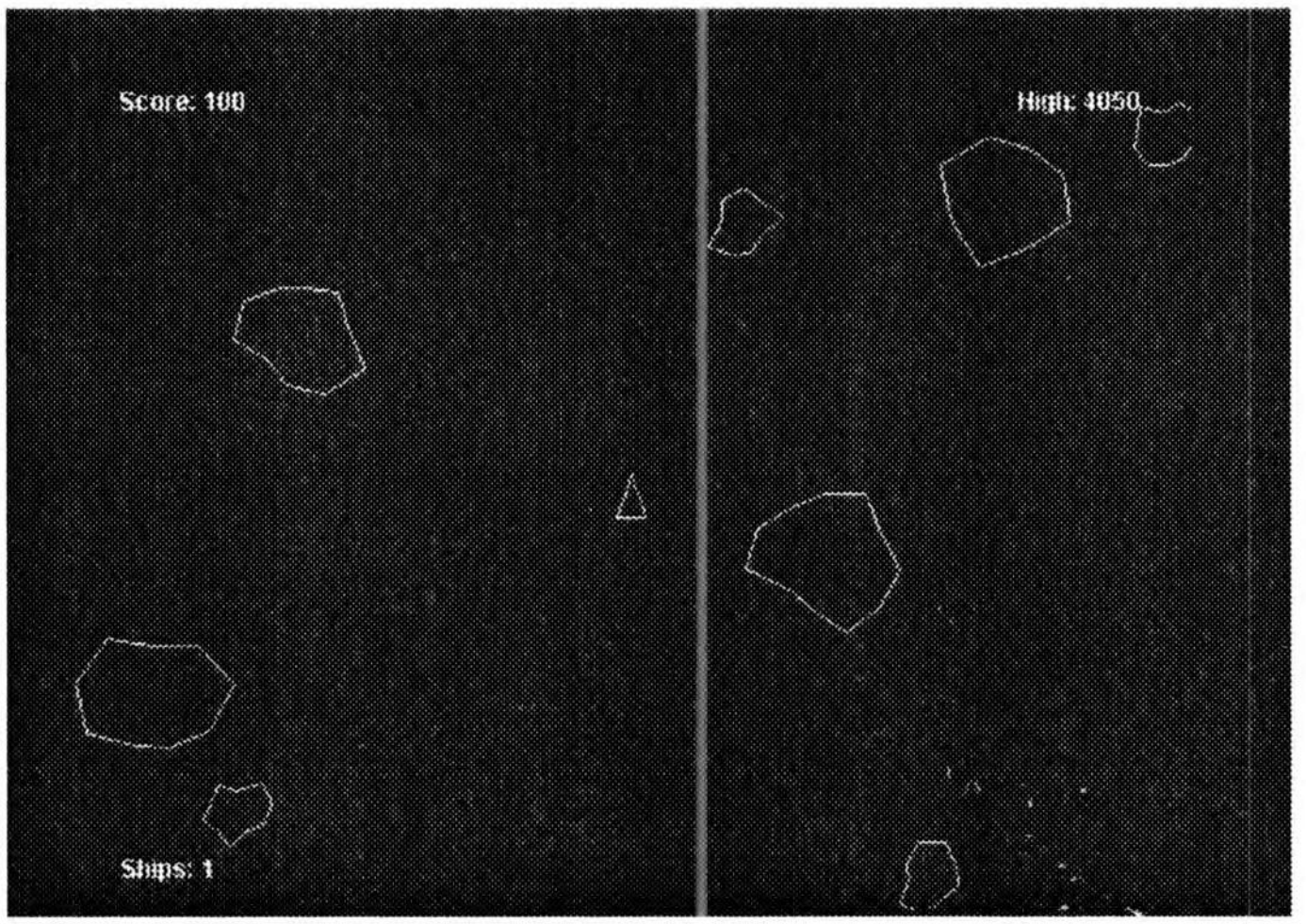
***NOTA**. En algunas instalaciones esta lista puede que no te funcione.*

7. Ejecuta la aplicación en un terminal real y verifica el resultado.

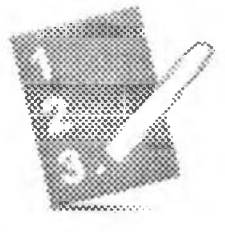
***NOTA**: En un emulador si cambias la orientación (Crtl-F11) esta cambiará igualmente. Se trata de un error de simulación, al no soportar esta configuración.*

## 4.4. Representación de gráficos vectoriales en Asteroides

La versión original del juego Asteroides se ejecutaba sobre ordenadores con escasa potencia gráfica. Tal y como puedes ver a continuación, nuestra nave se representaba con un simple triángulo.



Dado que cuando hemos diseñado la clase `Grafico`, la representación del mismo la hemos delegado en un objeto `Drawable`, va a resultar muy fácil cambiar los gráficos de la aplicación para que tengan una apariencia más retro. Simplemente utilizando la subclase de `Drawable`, `ShapeDrawable`, en lugar de `BitmapDrawable`, para cambiar la forma de dibujar los gráficos.



**Ejercicio paso a paso:** *Representación vectorial de los Asteroides.*

1. Abre la clase `VistaJuego`.
2. En el constructor remplaza la línea:

```
drawableAsteroide = context.getResources().getDrawable(
                                        R.drawable.asteroide1);
```

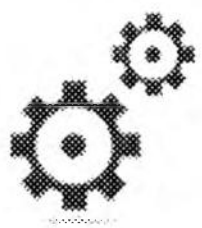
por el siguiente código:

```
SharedPreferences pref = context.getSharedPreferences(
    "org.example.asteroides_preferences", Context.MODE_PRIVATE);
if (pref.getString("graficos", "0").equals("0")) {
  Path pathAsteroide = new Path();
  pathAsteroide.moveTo((float) 0.3, (float) 0.0);
  pathAsteroide.lineTo((float) 0.6, (float) 0.0);
  pathAsteroide.lineTo((float) 0.6, (float) 0.3);
  pathAsteroide.lineTo((float) 0.8, (float) 0.2);
  pathAsteroide.lineTo((float) 1.0, (float) 0.4);
  pathAsteroide.lineTo((float) 0.8, (float) 0.6);
  pathAsteroide.lineTo((float) 0.9, (float) 0.9);
  pathAsteroide.lineTo((float) 0.8, (float) 1.0);
  pathAsteroide.lineTo((float) 0.4, (float) 1.0);
  pathAsteroide.lineTo((float) 0.0, (float) 0.6);
  pathAsteroide.lineTo((float) 0.0, (float) 0.2);
  pathAsteroide.lineTo((float) 0.3, (float) 0.0);
  ShapeDrawable dAsteroide = new ShapeDrawable(
                          new PathShape(pathAsteroide, 1, 1));
  dAsteroide.getPaint().setColor(Color.WHITE);
  dAsteroide.getPaint().setStyle(Style.STROKE);
  dAsteroide.setIntrinsicWidth(50);
  dAsteroide.setIntrinsicHeight(50);
  drawableAsteroide = dAsteroide;
  setBackgroundColor(Color.BLACK);
} else {
  drawableAsteroide = context.getResources().getDrawable(
                                        R.drawable.asteroide1);
}
```

Lo primero que hace este código es consultar en las preferencias para ver si el usuario ha escogido gráficos vectoriales. En caso negativo, se

realizará la misma inicialización de `drawableAsteroide` que teníamos antes. En caso afirmativo, comenzaremos creando la variable `pathAsteroide` de la clase `Path`. En este objeto se introducen todas las órdenes de dibujo necesarias para dibujar un asteroide. Luego se crea la variable `dAsteroide` de la clase `ShapeDrawable` para crear un *drawable* a partir del *path*. Los últimos dos parámetros (..., `1,1`) significan el valor de escala aplicado al eje x y al eje y. Luego se debe indicar el color y el estilo del pincel y el alto y ancho por defecto del *drawable*. Finalmente asignamos el objeto creado a `drawableAsteroide`.

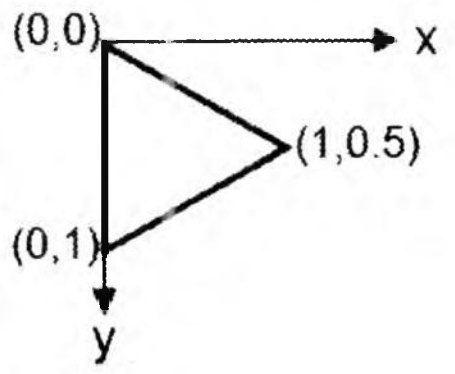
3. Ejecuta la aplicación y selecciona el tipo de gráficos adecuado en las preferencias.



**Práctica:** *Representación vectorial de la nave.*

Como habrás comprobado en el ejercicio anterior, la nave se representa siempre utilizando un fichero png. En esta práctica has de intentar que también pueda representarse vectorialmente.

1. Crea un nuevo objeto de la clase `Path` para representar la nave dentro de la sección `if` introducida en el ejercicio anterior. Como puedes ver en la ilustración siguiente, debe ser un simple triángulo. Como el ángulo de rotación inicial es cero, la nave debe mirar a la derecha.




2. Crea un nuevo `ShapeDrawable` a partir de `Path` anterior. Unas dimensiones adecuadas para que la nave pueden ser 20 de ancho y 15 de alto.

3. Inicializa la variable `drawableNave` de forma adecuada.

## 4.5. Animaciones

El entorno de programación Android incorpora tres mecanismos para crear animaciones en nuestras aplicaciones:

- La clase `AnimationDrawable`: Permite crear *drawables* que reproducen una animación fotograma a fotograma. Se ha descrito su uso en la sección de Drawables.
- Animaciones Tween: Permiten crear efectos de translación, rotación, zoom y alfa a cualquiera vista de nuestra aplicación.

- Animaciones de propiedades: Nuevo mecanismo incorporado en Android 3.0. Permite animar cualquier propiedad de cualquier objeto, sea una vista o no. Además, modifica el objeto en sí, no solamente cambia su representación en pantalla como ocurre en un animación Tween.

Las siguientes secciones describen con más detalle estos tipos de animación.

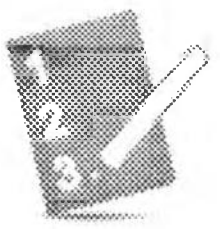
### 4.5.1. Animaciones Tween

Una "animación tween" puede realizar series de transformaciones simples (posición, tamaño, rotación y transparencia) en el contenido de un objeto View. Por ejemplo, si tienes un objeto `TextView` puedes moverlo, rotarlo, aumentarlo, disminuirlo o cambiarle la transparencia al texto.

La secuencia de órdenes que define la "animación tween" puede estar escrita mediante xml o código, pero es recomendable el fichero xml, al ser mas legible, reutilizable e intercambiable.

Las instrucciones de la animación definen las transformaciones que quieres que ocurran, cuándo ocurrirán y cuánto tiempo tardarán en completarse. Las transformaciones pueden ser secuenciales o simultáneas. Cada tipo de transformación tiene unos parámetros específicos, y también existen unos parámetros comunes a todas las transformaciones, como el tiempo que durarán y el tiempo de inicio.

El fichero XML que define a la animación debe pertenecer al directorio *res/anim/* en tu proyecto Android. El archivo debe tener solo un único elemento raíz: este debe ser uno de los siguientes: `<translate>`, `<rotate>`, `<scale>`, `<alpha>` o elemento `<set>` que puede contener grupos de estos elementos (además de otro `<set>`). Por defecto, todas las instrucciones de animación ocurren a partir del instante inicial. Si quieres que una animación comience más tarde debes especificar el atributo `startOffset`.



**Ejercicio paso a paso:** *Creación de una animación Tween para animar una vista.*

1. Crea un nuevo proyecto con nombre AnimacionTween.
2. Crea el fichero *res/anim/tweenanimation.xml* y pega el siguiente código:

```
<set xmlns:android="http://schemas.android.com/apk/res/android">
    <scale
        android:duration="2000"
        android:fromXScale="2.0"
        android:fromYScale="2.0"
        android:toXScale="1.0"
        android:toYScale="1.0" />
    <rotate
        android:startOffset="2000"
        android:duration="2000"
```

```
            android:fromDegrees="0"
            android:toDegrees="360"
            android:pivotX="50%"
            android:pivotY="50%"/>
        <translate
            android:startOffset="4000"
            android:duration="2000"
            android:fromXDelta="0"
            android:fromYDelta="0"
            android:toXDelta="50"
            android:toYDelta="100" />
        <alpha
            android:startOffset="4000"
            android:duration="2000"
            android:fromAlpha="1"
            android:toAlpha="0" />
    </set>
```

3. Abre el fichero *res/Layout/main.xml* y añade el siguiente atributo a la vista de tipo `TextView`:

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

4. Abre la actividad del proyecto y añade las líneas marcadas en negrita al método `OnCreate()`.

```
@Override public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.main);
  TextView texto = (TextView) findViewById(R.id.textView);
  Animation animacion = AnimationUtils.loadAnimation(this,
                                                R.anim.animacion);
  texto.startAnimation(animacion);
}
```

5. Ejecuta la aplicación.

Como podrás ver, el `TextView` comienza haciéndose más pequeño (etiqueta `<scale>`), después rota sobre sí mismo (etiqueta `<rotate>`) y, finalmente, se desplaza (etiqueta `<translate>`) a la vez que se hace transparente (etiqueta `<alpha>`). Al finalizar la animación, vuelve a su posición y estado inicial, sin importar dónde ni cómo haya acabado.



**Recursos adicionales:** *Lista de etiquetas de las animaciones tween y sus atributos*

Los siguientes atributos son aplicables a todas las transformaciones:

`startOffset` – Instante inicial de la transformación en milisegundos.

`duration` – Duración de la transformación en milisegundos.

`repeatCount` – Número de repeticiones adicionales de la animación.

`interpolator` – En lugar de realizar una transformación lineal se aplica algún tipo de interpolación. Alguno de los valores posibles son:

```
accelerate_decelerate_interpolator, accelerate_interpolator,
anticipate_interpolator, anticipate_overshoot_interpolator,
bounce_interpolator, cycle_interpolator, decelerate_interpolator,
linear_interpolator, overshoot_interpolator
```

Lista de las transformaciones con sus atributos específicos:

**`<translate>`** – Desplaza la vista.

`fromXDelta`, `toXDelta` – Valor inicial y final del desplazamiento en eje X.

`fromYDelta`, `toYDelta` – Valor inicial y final del desplazamiento en eje Y.

**`<rotate>`** – Rota la vista.

`fromDegrees`, `toDegrees` – Valor inicial y final en grados de la rotación en grados. Si quieres un giro completo en sentido antihorario elige de 0 a 360, y si lo quieres en sentido horario, de 360 a 0 o de 0 a -360. Si quieres dos giros elige de 0 a 720.

`pivotX`, `pivotY` – Punto sobre el que se realizará el giro. Este quedará fijo en la pantalla.

**`<scale>`** – Cambia el tamaño de la vista.

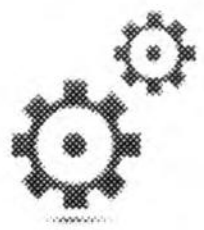
`fromXScale`, `toXScale` – Valor inicial y final para la escala del eje X (0.5=50%, 1=100%)

`fromYScale`, `toYScale` – Valor inicial y final para la escala del eje Y.

`pivotX`, `pivotY` – Punto sobre el que se realizará el zoom. Este quedará fijo en la pantalla.

**`<alpha>`** – Cambia la opacidad de la vista.

`fromAlpha`, `toAlpha` – Valor inicial y final de la opacidad.



**Práctica:** *Introduciendo animaciones en Asteroides*

En esta práctica has de conseguir que los diferentes elementos del Layout inicial de Asteroides vayan apareciendo, uno tras otro, con diferentes efectos.

1. Abre el proyecto Asteroides.
2. Crea una nueva animación con nombre *giro_con_zoom.xml*. Debe durar dos segundos y de forma simultánea debe hacer un zoom de escala 3 a 1 y un giro de dos vueltas (720º). El punto de anclaje de la rotación y el zoom ha de ser el centro de la vista.
3. Selecciona el Layout *main.xml* y pon un id al `TextView` correspondiente al título.

4. En la actividad inicial de Asteroides, crea un objeto correspondiente a este `TextView` y aplícale la animación anterior.
5. Crea una nueva animación con nombre *aparecer.xml*. Debe comenzar a los dos segundos, durar un segundo y modificar el valor de *alpha* de 0 hasta 1.
6. Aplica esta animación al primer botón.
7. Crea una nueva animación con nombre *desplazamiento_derecha.xml*. Debe comenzar a los tres segundos, durar un segundo y modificar el valor de *desplazamiento x de* 400 hasta 0. Prueba también algún tipo de interpolación.
8. Aplica esta animación al segundo botón.
9. Si dispones de tiempo, crea dos nuevas animaciones a tu gusto y aplícalas al tercer y cuarto botón.
10. Aplica la animación *giro_con_zoom.xml* al botón "Acerca de" cuando sea pulsado. Observa cómo al lanzar la nueva actividad `AcercaDe`, la actividad principal continúa ejecutándose.

## 4.5.2. Animaciones de propiedades

A partir de la versión 3.0 de Android (nivel de API 11), se ha incorporado un nuevo tipo de animaciones. A diferencia de las animaciones Tween que solo es aplicable a vistas, una animación de propiedades puede animar cualquier tipo de objetos. Además, no está restringido a las cuatro transformaciones antes vistas, podemos animar cualquier propiedad del objeto. Por ejemplo, podemos hacer una animación que cambie progresivamente el color de fondo de una vista.

Otra diferencia, con respecto a las animaciones Tween, es que estas solo modifican la forma en que la vista es representada, pero no sus propiedades. Por ejemplo, si aplicas una animación Tween para que un texto se desplace por la pantalla, se visualizará correctamente, pero al acabar la animación el texto estará en el lugar inicial. Lo que te obligará a implementar tu propia lógica para manejar este cambio de posición. En una animación de propiedades estará cambiando el objeto en sí, no solamente cómo se representa.

Desventajas de las animaciones Tween:

- Solo podemos animar objetos de la clase `View`.
- Está limitado a estas cuatro transformaciones y no puede aplicarse a otros aspectos como cambiar el color de fondo.
- Solo modifica la forma en que la vista es representada, pero no sus propiedades en sí.

Desventajas de las animaciones de propiedades:

- Solo disponible a partir de la versión 3.0. En la actualidad existen muy pocos dispositivos que la soporten.
- Requiere más tiempo para inicializarse y hay que escribir más código.

Para aprender más sobre este tipo de animación puedes leer la siguiente sección de Android Developers: ***Property Animation***.

# CAPÍTULO 5.
# Entradas en Android: teclado, pantalla táctil y sensores

La forma más habitual para interactuar con un ordenador es el teclado y el ratón. Por desgracia, estos dispositivos de entrada no existen, o están muy limitados, en un teléfono móvil. Afortunadamente, los nuevos móviles permiten nuevas formas de interacción con el usuario, por lo que el diseño de nuestras aplicaciones ha de adaptarse a estas nuevas formas de interacción. A lo largo de este capítulo se estudiarán diferentes alternativas para recoger las acciones que los usuarios realizan sobre la aplicación.

Tras una visión general del manejo de eventos en Android, comenzaremos con el dispositivo más tradicional, el teclado, luego estudiaremos los eventos de la pantalla táctil, y finalizaremos con los sensores. Estos tres mecanismos de interacción serán aplicados al manejo de nuestra nave en la aplicación Asteroides. De forma adicional se tocarán otros aspectos importantes, como el manejo de hilos de ejecución (*threads*) y las *gestures*.

**Objetivos:**

- Mostrar las distintas alternativas para manejar los eventos de usuario en Android.
- Describir cómo se manejan los eventos del teclado.
- Aprender a interaccionar con la pantalla táctil.
- Descubrir qué son las *Gestures* y como pueden ayudarte en el diseño del interfaz de usuario.
- Enumerar los sensores disponibles en muchos terminales Android y aprender a utilizarlos.
- Describir el uso de hilos de ejecución (`Thread`).
- Seguir mejorando la aplicación Asteroides.

# 5.1. Manejando eventos de usuario

Android captura los distintos eventos de usuario de forma homogénea y se los pasa a la clase encargada de recogerlos. Por lo general, va a ser un objeto tipo `View` el que recogerá estos eventos por medio de dos técnicas alternativas. Los escuchadores de eventos (*Event Listener*) y los manejadores de eventos (*Event Handler*).

## 5.1.1. Escuchador de eventos

Un Escuchador de eventos o *Event Listener* es una interfaz de la clase `View` que contiene un método *callback* que ha de ser registrado. Cada *Escuchador de Eventos* tiene solo un método *callback*, que será llamado por Android cuando se produzca la acción correspondiente. Tenemos los siguientes escuchadores de eventos:

`onClick()`

Método de la interfaz `View.OnClickListener`. Se llama cuando el usuario selecciona un elemento. Se puede utilizar cualquier medio como la pantalla táctil, las teclas de navegación o el *trackball*.

`onLongClick()`

Método de la interfaz `View.OnLongClickListener`. Se llama cuando el usuario selecciona un elemento durante más de un segundo.

`onFocusChange()`

Método de la interfaz `View.OnFocusChangeListener`. Se llama cuando el usuario navega dentro o fuera de un elemento.

`onKey()`

Método de la interfaz `View.OnKeyListener`. Se llama cuando se pulsa o se suelta una tecla del dispositivo.

`onTouch()`

Método de la interfaz `View.OnTouchListener`. Se llama cuando se pulsa o se suelta o se desplaza en la pantalla táctil.

`onCreateContextMenu()`

Método de la interfaz `View.OnCreateContextMenuListener`. Se llama cuando se crea un menú de contexto.

Existen dos alternativas para crear un escuchador de evento. La primera es crear un objeto anónimo por ejemplo de la clase `OnClickListener()`:

```
protected void onCreate(Bundle savedValues) {
   ...
   Button boton = (Button)findViewById(R.id.boton);
   boton.setOnClickListener( new OnClickListener() {
      public void onClick(View v) {
         // Acciones a realizar
      }
   ...
}
```

La segunda alternativa consiste en implementar la interfaz `OnClickListener` como parte de tu clase y recoger los eventos en el método `onClick()`. Esta alternativa es la recomendada por Android, al tener menos gasto de memoria. A continuación, se muestra un ejemplo:

```
public class Ejemplo extends Activity implements OnClickListener{
    protected void onCreate(Bundle savedValues) {
        ...
        Button boton = (Button)findViewById(R.id.boton);
        boton.setOnClickListener(this);
    }

    public void onClick(View v) {
     // Acciones a realizar
    }
    ...
}
```

### 5.1.2. Manejadores de eventos

Si estás creando un descendiente de la clase *View*, podrás utilizar varios métodos (*callback*) directamente usados como manejadores de eventos por defecto (*Event Handlers*). En esta lista se incluye:

`onKeyDown(int keyCode, KeyEvent e)` Llamado cuando una tecla es pulsada.

`onKeyUp(int keyCode, KeyEvent e)` Cuando una tecla deja de ser pulsada.

`onTrackballEvent(MotionEvent me)` Llamado cuando se mueve el trackball.

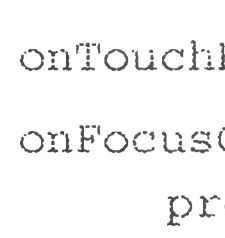
`onTouchEvent(MotionEvent me)` Cuando se pulsa en la pantalla táctil.

`onFocusChanged(boolean obtengoFoco, int direccion, Rect prevRectanguloFoco)` Llamado cuando cambia el foco.

Es la forma más sencilla, dado que no hace falta usar una interfaz, ni registrar el método *callback*. Como en nuestro ejemplo estamos creando `JuegoView`, que es un descendiente de `View`, podremos utilizar directamente manejadores de evento.

## 5.2. El teclado

Aunque cada vez existen menos terminales Android con teclado físico, siempre es interesante aprender a gestionar los eventos procedentes del teclado. Su manejo se ilustra en el siguiente ejercicio.



**Ejercicio paso a paso:** *Manejo de la nave con el teclado.*

Veamos cómo podemos utilizar un manejador de eventos de teclado para maniobrar la nave de Asteroides:

1. Abre el proyecto *Asteroides*.
2. Inserta este código en la clase `VistaJuego`.

```
@Override
public boolean onKeyDown(int codigoTecla, KeyEvent evento) {
  super.onKeyDown(codigoTecla, evento);
  // Suponemos que vamos a procesar la pulsación
  boolean procesada = true;
  switch (codigoTecla) {
  case KeyEvent.KEYCODE_DPAD_UP:
     aceleracionNave = +PASO_ACELERACION_NAVE;
     break;
  case KeyEvent.KEYCODE_DPAD_LEFT:
     giroNave = -PASO_GIRO_NAVE;
     break;
  case KeyEvent.KEYCODE_DPAD_RIGHT:
     giroNave = +PASO_GIRO_NAVE;
     break;
  case KeyEvent.KEYCODE_DPAD_CENTER:
  case KeyEvent.KEYCODE_ENTER:
     ActivaMisil();
     break;
  default:
     // Si estamos aquí, no hay pulsación que nos interese
     procesada = false;
     break;
  }
  return procesada;
}
```

3. Cada vez que se pulse una tecla se realizará una llamada al método `onKeyDown()` con los siguientes parámetros: El primero es un entero que nos identifica el código de la tecla pulsada. El segundo es de la clase `KeyEvent` y nos permite obtener información adicional sobre el evento como, por ejemplo, cuándo se produjo. Este método ha de devolver un valor booleano, verdadero, si consideramos que la pulsación ha sido procesada por nuestro código, y falso, si queremos que otro manejador de evento, siguiente al nuestro, reciba la pulsación.
4. Antes de ponerlo en marcha comenta la llamada a `ActivaMisil()`, dado que esta función aún no está implementada.
5. Verifica si funciona correctamente.

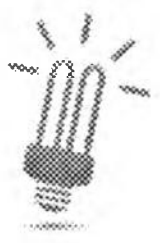
***NOTA:*** *Para poder recoger eventos de teclado desde una vista es necesario que esta tenga el foco y para que esto sea posible verifica que tiene la propiedad* `focusable="true"`.

**Práctica**: *Manejo de la nave con el teclado.*

El ejercicio anterior no funciona de forma satisfactoria. Cuando pulsamos una tecla para girar la nave se pone a girar pero ya no hay manera de pararla. El manejador de eventos `onKeyDown` solo se activa cuando se pulsa una tecla, pero no cuando se suelta.

Trata de escribir el manejador de eventos `onKeyUp` para que la nave atienda a las órdenes de forma correcta. Puedes partir del siguiente código:

```
@Override
public boolean onKeyUp(int codigoTecla, KeyEvent evento) {
    super.onKeyUp(codigoTecla, evento);
    // Suponemos que vamos a procesar la pulsación
    boolean procesada = true;
    ...
    return procesada;
}
```



**Solución:** Una posible solución al ejercicio se muestra a continuación:

```
@Override
public boolean onKeyUp(int codigoTecla, KeyEvent evento) {
    super.onKeyUp(codigoTecla, evento);
    // Suponemos que vamos a procesar la pulsación
    boolean procesada = true;
    switch (codigoTecla) {
    case KeyEvent.KEYCODE_DPAD_UP:
        aceleracionNave = 0;
        break;
    case KeyEvent.KEYCODE_DPAD_LEFT:
    case KeyEvent.KEYCODE_DPAD_RIGHT:
        giroNave = 0;
        break;
    default:
        // Si estamos aquí, no hay pulsación que nos interese
        procesada = false;
        break;
    }
    return procesada;
}
```

## 5.3. La pantalla táctil

Los teléfonos Android suelen incorporar una pantalla táctil, que es utilizada como dispositivo principal de entrada. El uso más importante de la pantalla táctil es como sustituto del ratón de un ordenador de sobremesa. De esta forma podemos seleccionar, arrastrar y soltar cualquier elemento de la pantalla de forma sencilla. No obstante el uso de este dispositivo no acaba aquí. Suele utilizarse en sustitución del teclado en aquellos dispositivos que no disponen de teclado físico. También puede ser utilizada como entrada de un videojuego, como se verá en este apartado. Otra alternativa para usar la pantalla táctil consiste en el uso de *gestures* soportado a partir del SDK 1.6. Las *gestures* serán estudiadas en el siguiente punto. Otro abanico de nuevas posibilidades se abre con el *multi-touch*, soportado a partir del SDK 2.0.

El manejo básico de la pantalla táctil pasa por definir el método `onTouchEvent` en una clase `View` (o implementar la interfaz `OnTouchListener` en otras clases). Este método nos devolverá en un parámetro, un objeto de la clase `MotionEvent`.

Los métodos más interesantes de la clase `MotionEvent` se indican a continuación:

`getAction()` Tipo de acción realizada. En API level 1 puede ser: `ACTION_DOWN`, `ACTION_MOVE`, `ACTION_UP` o `ACTION_CANCEL`.

`getX()`, `getY()` Posición de la pulsación.

`getDownTime()` Tiempo en ms en que el usuario presionó, por primera vez, en una cadena de eventos de posición.

`getEventTime()` Tiempo en ms del evento actual.

`getPression()` Estima la presión de la pulsación. El valor 0 es el mínimo, el valor 1 representa una pulsación normal.

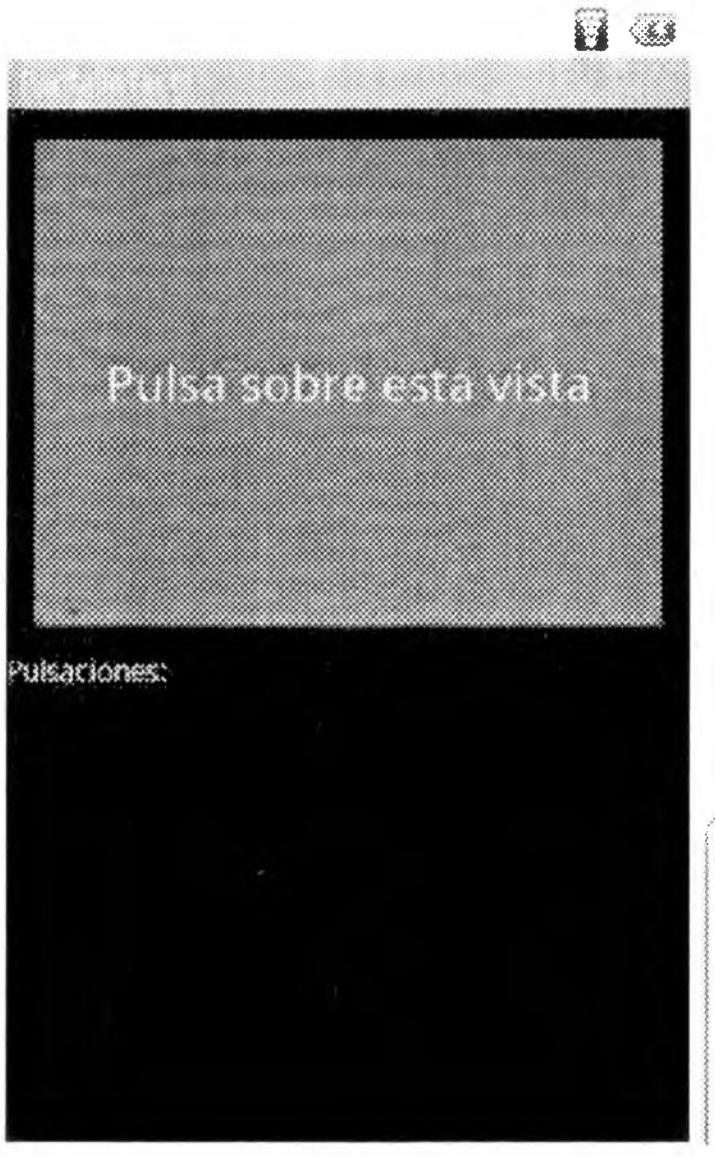
`getSize()` Valor escalado en 0 y 1 que estima el grosor de la pulsación.

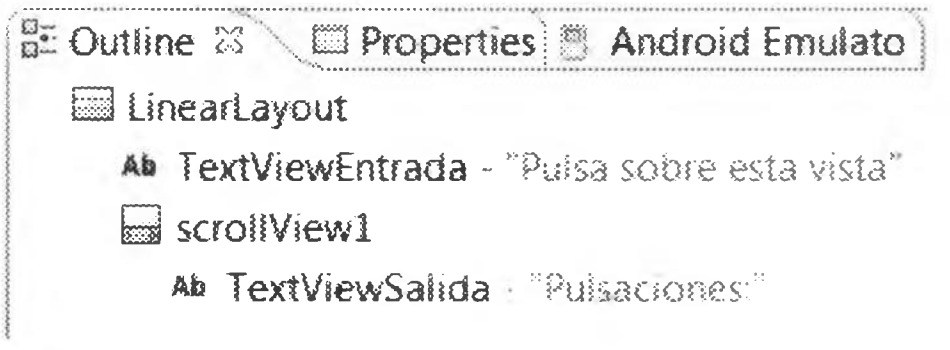
A partir del API level 5 estos métodos pueden indicar como parámetro un índice de puntero para decirle al sistema sobre cuál de los distintos punteros estamos consultando.

**Ejercicio paso a paso:** *Uso de la pantalla táctil.*

En este ejercicio se mostrará cómo podemos capturar los eventos procedentes de la pantalla táctil. También se aprovechará para repasar otros conceptos como: Creación de Layouts y herramientas de revisión de código en Eclipse.

1. Crea un nuevo proyecto con nombre *PantallaTactil*.
2. Modifica el Layout *main.xml* para que tenga una apariencia similar a la siguiente. De esta forma practicarás la creación de Layouts. A la derecha se muestra la estructura de vistas que contiene.




3. Una posible solución se muestra a continuación:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >
    <TextView
        android:id="@+id/TextViewEntrada"
        android:layout_width="fill_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:text="Pulsa sobre esta vista"
        android:gravity="center"
        android:background="#0000FF"
        android:layout_margin="2mm"
        android:textSize="10pt"/>
    <ScrollView
    android:id="@+id/scrollView1"
    android:layout_width="fill_parent"
    android:layout_height="0dp"
    android:layout_weight="1" >
    <TextView
        android:id="@+id/TextViewSalida"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:text="Pulsaciones:"/>
```

```
    </ScrollView>
</LinearLayout>
```

4. Introduce las siguientes dos líneas al final del método `onCreate()`:

```
TextView entrada = (TextView) findViewById(R.layout.
                                                TextViewEntrada);
entrada.setOnTouchListener(this);
```

5. Pulsa *Shift-Ctrl-O* para añadir los *imports*.

6. Observa cómo el método `setOnTouchListener` está marcado como erróneo. Si pones el cursor encima, te indicará que el parámetro de este método (`this`) es de la clase `PantallaTactilActivity` y es necesario que sea de tipo `OnTouchListener`.

7. Para evitar el error te mostrará una lista de posibles soluciones. Selecciona la última "`Lef 'PantallaTactilActivity' implement 'OnTouchListener'`" de esta forma implementaremos este interfaz y nuestra clase podrá ser considerada de este tipo. La declaración de la clase cambiará a:

```
public class PantallaTactilActivity extends Activity
                                        implements OnTouchListener {
```

8. Se ha solucionado el problema anterior, pero ha aparecido otro. Ahora, la `PantallaTactilActivity` está marcada como errónea. El problema consiste en que estamos diciendo que implementamos el interfaz `OnTouchListener` pero no hemos implementado ninguno de los métodos de este interfaz.

9. Para evitar el error selecciona en la lista de posibles soluciones: "`Add unimplemented methods`" de esta forma se añadirán todos los métodos necesarios de este interfaz. La declaración de la clase cambiará a:

```
@Override
public boolean onTouch(View arg0, MotionEvent arg1) {
  // TODO Auto-generated method stub
  return false;
}
```

10. Reemplaza el nombre de los parámetros por otros más expresivos. Por ejemplo: `arg0` por `vista` y `arg1` por `evento`.

11. Observa como este método ha de devolver un parámetro. Actualmente es `false`, que significa que no nos hemos hecho cargo de la pulsación, el sistema seguirá pasando este evento a otras vistas. En este caso el `LinearLayout` que contiene la vista. Cámbialo a `true`, para que el sistema no siga propagando este evento.

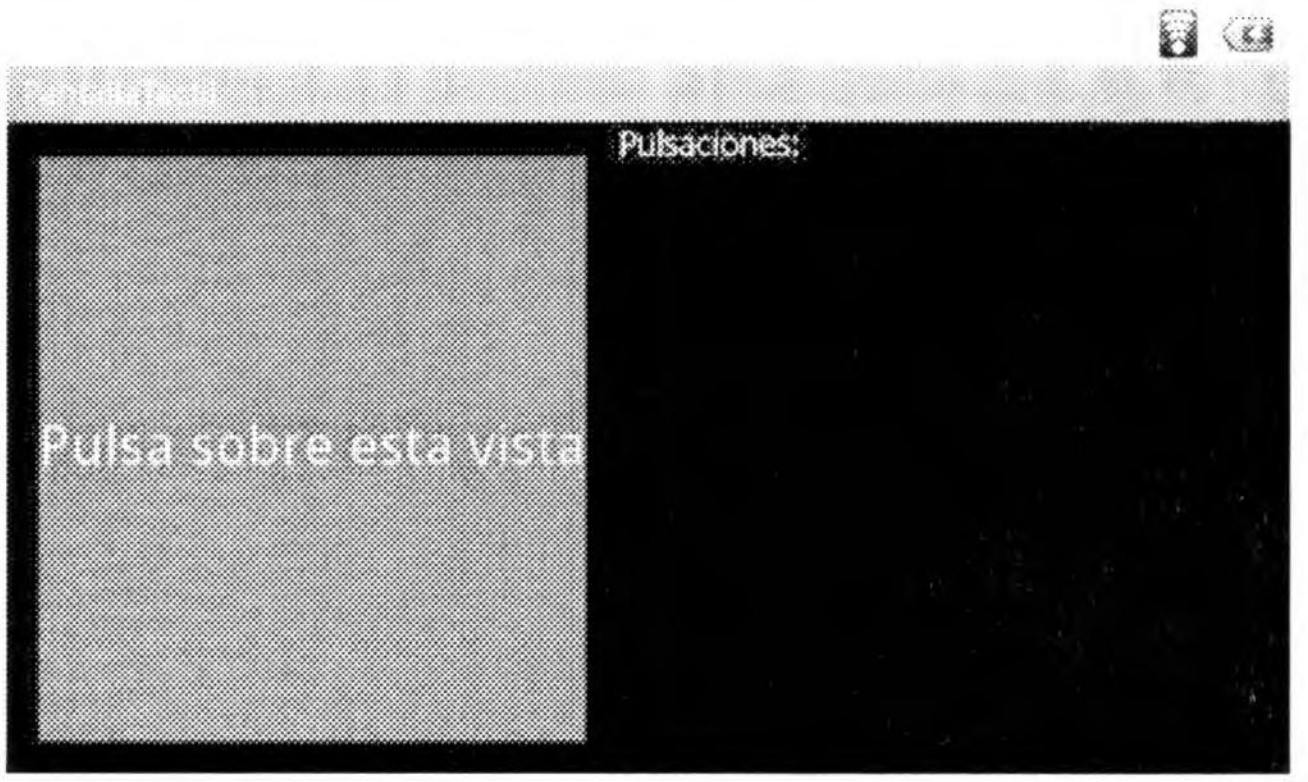
12. Reemplaza la línea "`// TODO Auto-generated method stub`" por:

```
TextView salida = (TextView) findViewById(R.id.TextViewSalida);
salida.append(evento.toString()+"\n");
```

13. Ejecuta el proyecto y verifica el resultado.

14. `accion=0` significa que se ha pulsado sobre la pantalla, `accion=1` significa que se ha soltado y `accion=2` que se está desplazando el dedo (estos tres valores corresponden con las constantes `MotionEvent.ACTION_DOWN`, `MotionEvent.ACTION_UP` y `MotionEvent.ACTION_MOVE`).

15. Modifica el proyecto para que cuando el móvil se ponga en apaisado el Layout que se visualice sea:




16. Verifica el resultado en un dispositivo real.

17. No todas las pantallas táctiles soportan los métodos `getPression()` y `getSize()`. Prueba si tu terminal lo soporta y, en tal caso, observa el rango de valores que obtienes.

18. Conéctate a www.androidcurso.com y localiza el ejercicio que acabas de realizar. Comprueba los valores obtenidos con otros terminales y comparte los valores obtenidos por tu terminal.

## 5.3.1. Manejo de la pantalla táctil con *multi-touch*

Las pantallas táctiles más modernas tienen la posibilidad de indicar la posición de varios punteros sobre la pantalla a un mismo tiempo. Para averiguar si el dispositivo tiene esta capacidad puedes utilizar el siguiente código:

```
boolean multiTouch = getPackageManager().hasSystemFeature(
        PackageManager.FEATURE_TOUCHSCREEN_MULTITOUCH);
```

A partir de la versión 2.0 (API level 5), un objeto de la clase `MotionEvent` contendrá información de todos estos punteros. Un puntero estará activo desde que se pulsa sobre la pantalla hasta que se deja de presionar. El número de punteros activos puede consultarse llamando al método `getPointerCount()`. Cada puntero tiene un id para identificarlo que es asignado cuando se produce la primera pulsación.

La clase `MotionEvent` amplia la lista de constantes para identificar las acciones posibles para adaptarse a multi-touch a partir de la versión 2.0.

Veamos una lista:

`ACTION_DOWN` – Se pulsa en la pantalla sin haber otro puntero activo.

`ACTION_UP` – Se deja de presionar el último puntero activo.

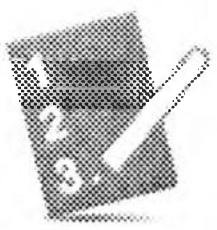
`ACTION_MOVE` – Cualquiera de los punteros activos se desplaza.

`ACTION_CANCEL` – Se cancela un gesture.

`ACTION_OUTSIDE` – El puntero se sale de la vista.

`ACTION_POINTER_DOWN` – Se pulsa un nuevo puntero distinto al primero.

`ACTION_POINTER_UP` – Se deja de presionar un puntero pero no es el último.



**Ejercicio paso a paso:** *Uso de la pantalla táctil multi-touch.*

1. Ejecuta el ejercicio anterior en un dispositivo real con capacidad de multi-touch (si no dispones de uno te será imposible realizar este ejercicio).
2. Pulsa simultáneamente con dos dedos en la pantalla: Si lo haces, sin desplazar los dedos, recibirás 4 eventos. Los dos primeros por las pulsaciones de cada dedo y los dos siguientes cuando se levanten. El resultado puede simular al siguiente:

```
Pulsaciones:MotionEvent{4050b730 action=0
x=119.0 y=237.0 pressure=0.32156864
size=0.13333334}
MotionEvent{4050b730 action=261 x=287.0
y=91.0 pressure=0.3803922 size=0.20000002}
MotionEvent{4050b730 action=262 x=287.0
y=91.0 pressure=0.3803922 size=0.20000002}
MotionEvent{4050b730 action=1 x=119.0 y=237.0
pressure=0.32156864 size=0.13333334}
```

3. Como puedes ver, cuando hay más de un puntero en pantalla la acción resulta compleja de interpretar. Veremos cómo hacerlo a continuación.
4. En primer lugar asegúrate que tu proyecto utiliza las APIs 2.0 o superior. Para ello utiliza la opción de menú *Project/Properties/Android* y selecciona en *Project Build Target* la opción *Android 2.0*.
5. Remplaza la siguiente línea del método `onTouch()`:

```
salida.append(evento.toString()+"\n");
```

por:

```
String acciones[] = { "ACTION_DOWN", "ACTION_UP",
            "ACTION_MOVE", "ACTION_CANCEL", "ACTION_OUTSIDE",
            "ACTION_POINTER_DOWN", "ACTION_POINTER_UP" };
int accion = evento.getAction();
int codigoAccion = accion & MotionEvent.ACTION_MASK;
```

```
salida.append(acciones[codigoAccion]);
for (int i = 0; i < evento.getPointerCount(); i++) {
salida.append(" puntero:" + evento.getPointerId(i) +
        " x:" + evento.getX(i) + " y:" + evento.getY(i));
}
salida.append("\n");
return true;
```

6. Para visualizar cada posible acción hemos creado un array con sus nombres. A continuación, averiguamos la acción en la variable `accion`. Esta nueva acción la ha podido hacer cualquier puntero de los activos o uno nuevo. A partir de la versión 2.0, en esta variable se codifica simultáneamente el código de la acción (8 bits menos significativos) e índice de puntero que la ocasiona (siguientes 8 bits). Para obtener esta información por separado puedes utilizar el siguiente código:

```
int codigoAccion = accion & MotionEvent.ACTION_MASK;
int iPuntero = (accion & MotionEvent.ACTION_POINTER_ID_MASK) >>
MotionEvent.ACTION_POINTER_ID_SHIFT;
```

7. A partir de la versión 2.2 (API level 8) las dos últimas constantes quedan obsoletas y se definen otras dos equivalentes cuyos nombres son más adecuados:

```
int iPuntero = (accion & MotionEvent.ACTION_POINTER_INDEX_MASK)
                        >> MotionEvent.ACTION_POINTER_INDEX_SHIFT;
```

8. Una vez obtenido el código de la acción mostramos su nombre en la vista `salida`. Luego hacemos un bucle para mostrar información de todos los punteros activos. El método `getPointerCount()` nos permite averiguar su número. Vamos a recorrer los punteros activos con la variable `i`. Al principio de este apartado vimos una serie de métodos para averiguar información sobre el puntero (`getX()`, `getSize()`,...). A partir de la versión 2.0, estos métodos siguen dando información sobre el primer puntero activo que se pulsó, ahora también disponemos de los mismos métodos pero indicando un índice de puntero (`getX(i)`, `getSize(i)`,...) para averiguar información del resto de punteros.

9. El método `getPointerId(int indice)` nos permite averiguar el identificador del puntero. No hay que confundir el índice de puntero con su identificador. El índice se asigna en función del orden en que fueron pulsados. El índice cero siempre es el más antiguo. El índice de un puntero decrece a medida que los punteros anteriores a él dejan de estar activos. Por el contrario, el identificador de un puntero es asignado cuando se crea y permanece constante durante toda su vida. Nos será muy útil para seguir la pista de un determinado puntero. El método `findPointerIndex(int id)` nos permite averiguar el índice de un puntero a partir de su identificador.

10. Ejecuta de nuevo el proyecto y vuelve a pulsar con dos dedos. El resultado ha de ser similar al siguiente:


11. Prueba con otras combinaciones de pulsaciones e investiga la relación entre el índice y el id de puntero.

12. Modifica el programa para que, además, se muestre en cada evento, el índice de puntero que lo ocasionó.

## 5.3.2. Manejo de la nave con la pantalla táctil

**Ejercicio paso a paso:** *Manejo de la nave con la pantalla táctil.*

Veamos cómo podemos utilizar un manejador de eventos de la pantalla táctil para maniobrar la nave de Asteroides. El código que se muestra permite manejar la nave de la siguiente forma: un desplazamiento del dedo horizontal hace girar la nave, un desplazamiento vertical produce una aceleración y, si al soltar la pulsación no hay movimiento, se provoca un disparo.

1. Abre el proyecto Asteroides.

2. Inserta este código en la clase `VistaJuego`.

```
private float mX=0, mY=0;
private boolean disparo=false;

@Override
public boolean onTouchEvent (MotionEvent event) {
  super.onTouchEvent(event);
  float x = event.getX();
  float y = event.getY();
  switch (event.getAction()) {
  case MotionEvent.ACTION_DOWN:
      disparo=true;
      break;
  case MotionEvent.ACTION_MOVE:
      float dx = Math.abs(x - mX);
      float dy = Math.abs(y - mY);
      if (dy<6 && dx>6){
          giroNave = Math.round((x - mX) / 2);
          disparo = false;
      } else if (dx<6 && dy>6){
          aceleracionNave = Math.round((mY - y) / 25);
          disparo = false;
```

```
        }
        break;
    case MotionEvent.ACTION_UP:
        giroNave = 0;
        aceleracionNave = 0;
        if (disparo){
        ActivaMisil();
        }
        break;
    }
      mX=x; mY=y;
       return true;
    }
```

3. Las variables globales `mX` y `mY` van a ser utilizadas para recordar las coordenadas del último evento. Comparándolas con las coordenadas actuales (`x`, `y`) podremos verificar si se trata de un desplazamiento horizontal o vertical. Por otra parte, la variable `disparo` es activada cada vez que comienza una pulsación (`ACTION_DOWN`). Si esta pulsación es continuada con un desplazamiento horizontal o vertical, `disparo` es desactivado. Si por el contrario, se levanta el dedo (`ACTION_UP`) sin haberse producido estos desplazamientos, `disparo` no estará desactivado y se llamará a `ActivaMisil()`.

4. Antes de ponerlo en marcha comenta la llamada a `ActivaMisil()`, dado que esta función aún no está implementada.

5. Verifica si funciona correctamente.

6. Modifica los parámetros de ajuste (`<6` ,`>6`, `/2`, `/25`), para que se adapten de forma adecuada a tu terminal.

7. En el juego original podíamos acelerar pero no desacelerar. Si queríamos detener la nave teníamos que dar un giro de 180 grados y acelerar lo justo. Modifica el código anterior para que no sea posible desacelerar.

## 5.4. *Gestures*

La pantalla táctil es uno de los mecanismos más cómodos para interaccionar con un teléfono Android. No obstante el reducido tamaño de la pantalla en los teléfonos móviles hace que el diseño de la interfaz de usuario sea complejo. Por ejemplo, tratar de introducir las decenas de botones y menús que incorporan la mayoría de aplicaciones de sobremesa sería imposible en una pantalla de 3 pulgadas. Para tratar de dar nuevas alternativas en el diseño de interfaz de usuario, a partir del SDK 1.6, se incorporan las *gestures*.

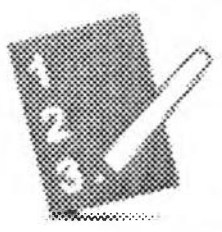
Una *gesture* es un movimiento pregrabado sobre la pantalla táctil, que la aplicación puede reconocer. De esta forma, la aplicación podrá realizar acciones

especiales en función de la *gesture* introducida por el usuario. Esto permite simplificar mucho una interfaz de usuario al poder reducir el número de botones.

Si quieres ver un ejemplo de cómo es utilizado *gestures* en una aplicación real, te recomendamos que bajes "Google Gesture Search" del Market. Como puedes ver en la siguiente ilustración, esta aplicación te permite introducir una secuencia de letras o dígitos escrita directamente en la pantalla. A partir de estos caracteres realiza una búsqueda global en tu teléfono (aplicaciones, música, contactos...).

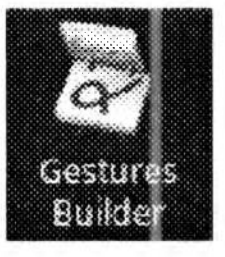

## 5.4.1. Creación y uso de una librería de gestures

El primer paso para usar *gestures* en nuestra aplicación es crear una librería que contenga algunas de ellas. Con este propósito, puedes utilizar la aplicación *Gesture Builder*, que está pre-instalada en la versión 1.6 del emulador.

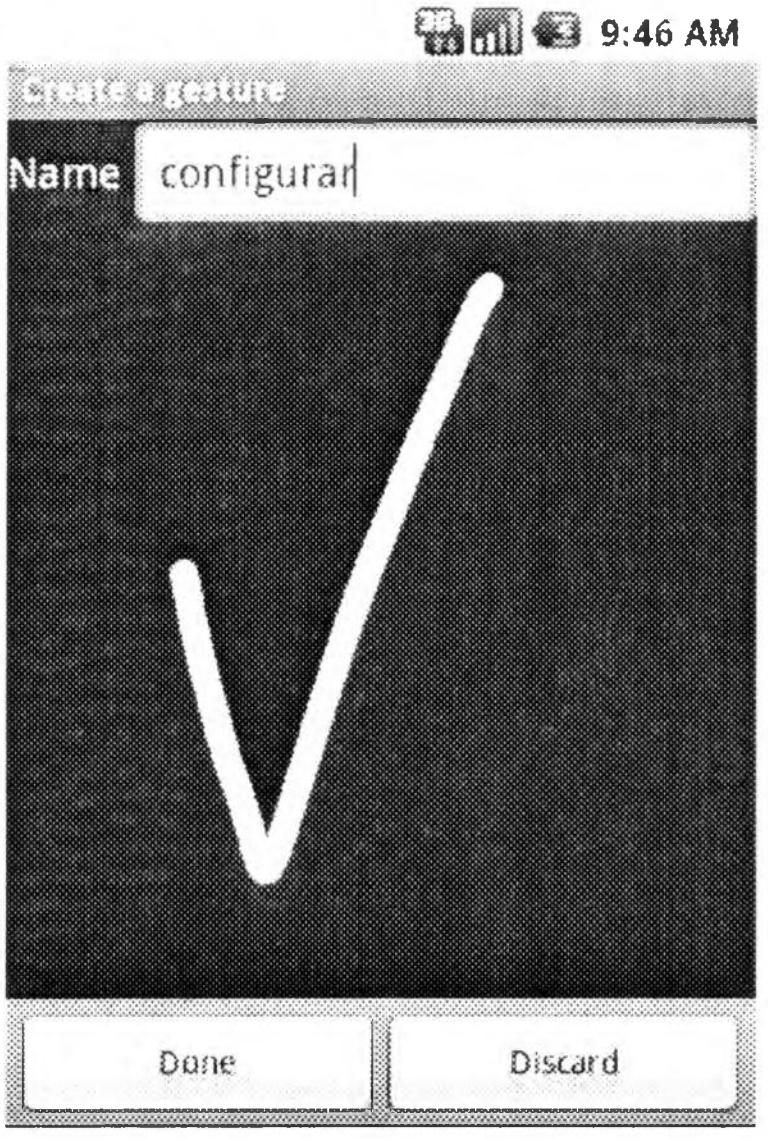


**Ejercicio paso a paso:** *Creación de una librería de gestures.*

1. Abre un emulador con versión 1.6 y con memoria externa.
2. En el menú de programas busca el siguiente icono y abre la aplicación.




3. Para añadir una nueva *gesture* a tu librería pulsa el botón "Add gesture" y realiza un trazado sobre la pantalla (por ejemplo, un visto bueno), luego introduce un nombre asociado a esta *gesture* (por ejemplo, "configurar"). El resultado se muestra a continuación:




4. Si no te gusta como ha quedado el trazado, no tienes más que realizar uno nuevo. El anterior será descartado.

5. Pulsa el botón "*Done*" para aceptar la *gesture*. Tras pulsar este botón aparecerá un cuadro de texto indicándote donde se acaba de guardar el fichero con la librería de *gestures*.

***NOTA**: Si intentas crear con el emulador una gesture formada por varios trazos (por ejemplo el símbolo "X") es posible que solo quede almacenado el último trazo. Para que ambos trazos sean reconocidos en la misma gesture, has de introducir el segundo justo a continuación del primero. Puede resultar algo difícil, pero tras un par de intentos lo conseguirás. No te preocupes, introducir una gesture de varios trazos en un dispositivo real no resulta tan complicado como en el emulador. Concretamente, el problema está en el valor* `FadeOffset` *que indica el tiempo máximo en milisegundos entre dos trazos de la misma gesture. Si al introducir dos trazos el tiempo entre ellos es mayor que* `FadeOffset`*, se considerará que se han introducido dos gestures diferentes. Por defecto, este valor es asignado a 420 milisegundos. El valor resulta adecuado con un dispositivo real, pero muy pequeño para el emulador. En el ejemplo descrito más adelante daremos un valor más alto a* `FadeOffset` *si queremos trabajar de forma más cómoda con el emulador.*

6. Trata de introducir las *gestures* mostradas en la siguiente captura. Para mejorar el porcentaje de reconocimientos correctos, puede ser interesante introducir varias veces la misma *gesture* con trazados alternativos. Esto se consigue dándole el mismo nombre a dos *gestures*.

7. Cada vez que una nueva *gesture* es introducida aparece una ventana de texto que indica el fichero donde está almacenanda nuestra librería. Seguramente será "/sdcard/gestures". Utiliza la vista *File Explorer* de Eclipse para localizar este fichero.


8. Selecciónalo y pulsa el botón para guardarlo en tu ordenador.

**Ejercicio paso a paso:** *Añadiendo la librería de gestures a nuestra aplicación.*

1. Para utilizar la librería que acabas de guardar, crea un nuevo proyecto con los siguientes datos:

```
Project Name: Gestures
Build Target: Android1.6
Application Name: Gestures
Package Name: org.example.gestures
Create Activity: Gestures
```

2. El siguiente paso va a consistir en crear la carpeta `res/raw` en el proyecto y copiar en ella el fichero que has guardado (`gestures`) en el ejercicio anterior.

3. Remplaza `res/layout/main.xml` por el siguiente código:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
  <TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
```

```
    android:gravity="center_horizontal"
    android:text="Introduce una gesture"
    android:textSize="8pt"
    android:layout_margin="10dip"/>
<TextView
    android:id="@+id/salida"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"/>
<android.gesture.GestureOverlayView
    android:id="@+id/gestures"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:gestureStrokeType="multiple"
    android:fadeOffset="800"/>
</LinearLayout>
```

4. El *layout* anterior está formado por un `LinearLayot` que contiene: un `TextView` con un título, un `TextView` para mostrar la salida del programa y un `GestureOverlayView` que será utilizado para introducir los *gestures*. En esta última etiqueta el parámetro `gestureStrokeType` indica que permitimos *gestures* formados por varios trazos. El parámetro `fadeOffset` ha sido explicado en el apartado anterior.

5. Reemplaza el código de la actividad por:

```
public class Gestures extends Activity implements
                                        OnGesturePerformedListener {
    private GestureLibrary libreria;
    private TextView osalida;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        libreria = GestureLibraries.fromRawResource(this,
                                                R.raw.gestures);
        if (!libreria.load()) {
           finish();
        }
       GestureOverlayView gesturesView = (GestureOverlayView)
                                        findViewById(R.id.gestures);
        gesturesView.addOnGesturePerformedListener(this);
        salida = (TextView) findViewById(R.id.salida);
}

public void onGesturePerformed(GestureOverlayView ov,
                                        Gesture gesture) {
    ArrayList<Prediction> predictions =
                                        libreria.recognize(gesture);
```

```
                salida.setText("");
                for (Prediction prediction : predictions){
                    salida.append(prediction.name+" " +
                                        prediction.score+"\n");
                }
            }
        }
```

6. En este código se comienza declarando dos campos de la clase `librería` que contendrá la librería de *gestures*, creada en el ejercicio anterior, y `salida` que corresponde al `TextView`, donde escribiremos los resultados.

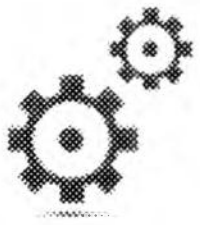
7. En el constructor, tras realizar las operaciones habituales, se carga la librería de *gestures* desde los recursos y en caso de no ser cargada finaliza la aplicación. A continuación, asocia el `GestureOverlayView` de `main.xml` al objeto `gestureView` y se indicará que nuestra clase será el escuchador de este elemento. Finalmente se asocia el `TextView` donde queremos sacar la salida al objeto `salida`.

8. El método `onGesturePerformed` se introduce para implementar la interfaz `OnGesturePerformedListener`. Este método tiene dos parámetros, el `GestureOverlayView` donde se ha introducido el *gesture* y el objeto `Gesture` que ha sido introducido. El primer paso consiste en reconocer el *gesture* comparándolo con la lista de nuestra librería. El resultado es una lista ordenada de `Predictions` con las *gestures* que considere más parecidas a la introducida. Tras borrar `salida`, se recorrerán todos los elementos de esta lista mostrando el nombre del *gesture* (`prediction.name`) y la puntuación de reconocimiento (`prediction.score`). Resulta complicado fijar un umbral, pero una puntuación inferior a 1 se suele considerar demasiado baja para tenerla en cuenta como predicción.

9. Ejecuta la aplicación y estudia las puntuaciones obtenidas.

## 5.4.2. Añadiendo *gestures* a Asteroides

En el siguiente ejercicio trataremos de aplicar lo aprendido a la aplicación Asteroides. La idea es que como una forma alternativa a usar el menú de cuatro botones que se muestra al arrancar la aplicación, se pueda utilizar *gestures*.



**Práctica**: *Añadiendo gestures a Asteroides.*

1. Si el proyecto Asteroides ha sido con una versión del SDK anterior a la 1.6, tendrás que actualizarlo como mínimo a esta versión. Para ello pulsa sobre el proyecto con el botón derecho y selecciona "properties", selecciona en la lista de la izquierda "Android" y marca la versión adecuada.

2. Crea la carpeta `res/raw` y copia el fichero `gestures`, que contiene la librería creada anteriormente.
3. Modifica el *Layout main.xml* para que disponga de un `GestureOverlayView`.
4. Cuando el usuario esté utilizando este Layout ha de poder introducir alguna de las cuatro gestures de la librería de forma que se ejecute la acción correspondiente.

**Solución:** Los pasos a seguir para realizar el ejercicio anterior se describen a continuación:

1. Añade al principio de `res/layout/main.xml` el siguiente código. Cierra la etiqueta al final del fichero.

```
<android.gesture.GestureOverlayView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/gestures"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:gestureStrokeType="multiple"
    android:fadeOffset="800">
```

2. En el fichero `Asteroides.java` añade en la definición de la clase:

```
public class Asteroides extends Activity
implements OnGesturePerformedListener{
    private GestureLibrary libreria;
    …
```

3. Añade al final del método onCreate:

```
libreria = GestureLibraries.fromRawResource(this,
                                            R.raw.gestures);
if (!libreria.load()) {
  finish();
}
GestureOverlayView gesturesView =
  (GestureOverlayView) findViewById(R.id.gestures);
gesturesView.addOnGesturePerformedListener(this);
```

4. Añade el siguiente método:

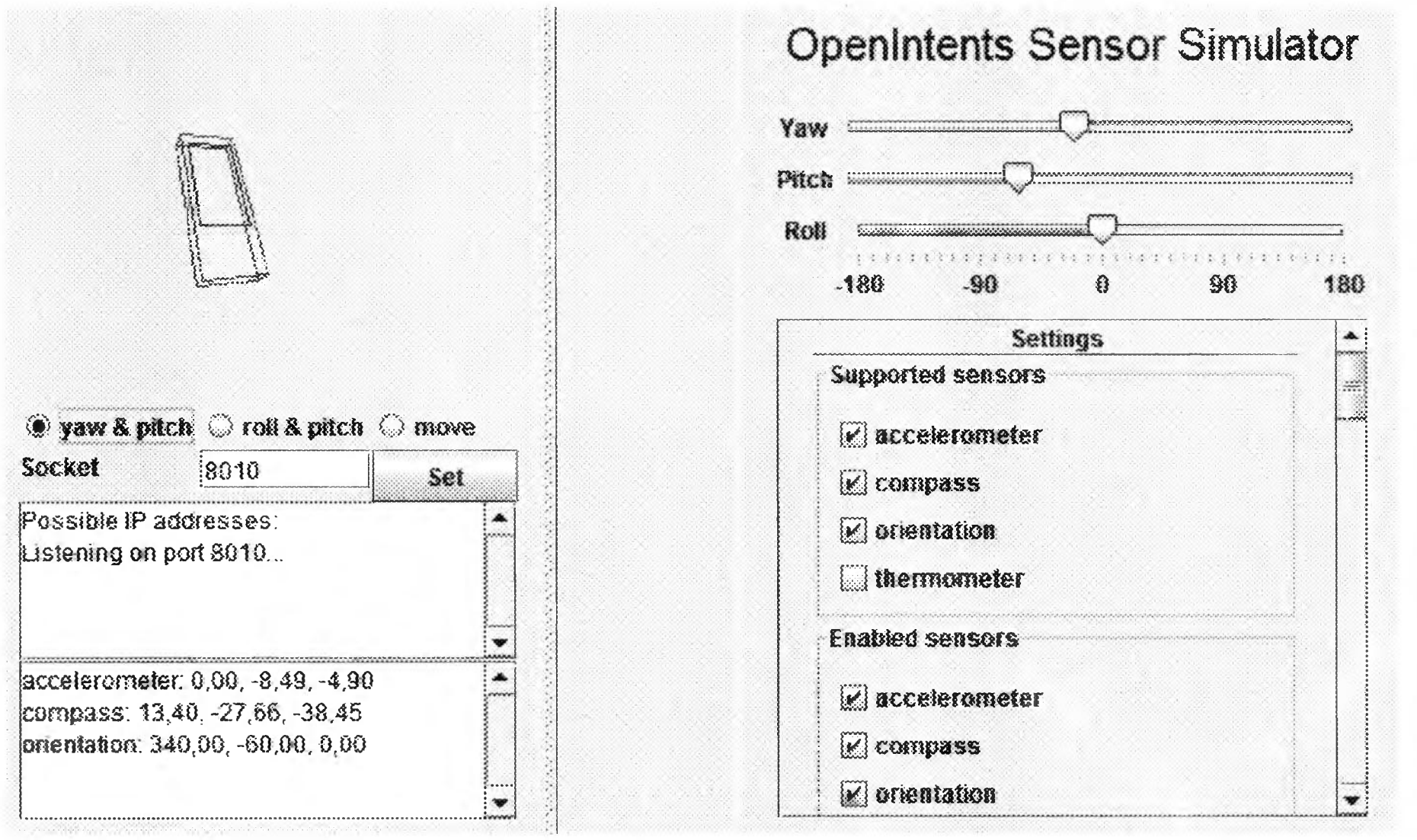
```
public void onGesturePerformed(GestureOverlayView ov,
                               Gesture gesture) {
  ArrayList<Prediction> predictions=libreria.recognize(gesture);
  if (predictions.size()>0){
     String comando = predictions.get(0).name;
     if (comando.equals("play")){
```

```
                lanzarJuego();
            } else if (comando.equals("configurar")){
                lanzarPreferencias();
            } else if (comando.equals("acerca_de")){
                lanzarAcercaDe();
            } else if (comando.equals("cancelar")){
                finish();
            }
        }
    }
```

## 5.5. Los sensores

Bajo la denominación de sensores se engloba un conjunto de dispositivos con los que podremos obtener información del mundo exterior (en este conjunto no se incluye la cámara, el micrófono o el GPS). Como se verá en este apartado, todos los sensores se manipulan de forma homogénea. Son los dispositivos de entrada más novedosos que incorpora Android y con ellos podremos implementar formas atractivas de interacción con el usuario.

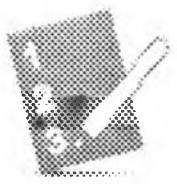
Si no dispones de un terminal físico, puedes instalar un software de emulación que te permitirá realizar las pruebas sobre el emulador, mientras desde una aplicación de tu PC cambias la orientación de un teléfono ficticio mediante el ratón. Puedes descargarte el *software* de http://www.openintents.org/en/node/6, no obstante, el proceso resulta bastante laborioso. Además, tendrás que cambiar el código de los ejemplos para adaptarlos a los requerimientos del emulador. Por lo tanto, tendrás que realizar dos programas diferentes: uno para sensores reales y otro para el emulador.



Android permite acceder a los sensores internos del dispositivo a través de las clases `Sensor`, `SensorEvent`, `SensorManager` y la interfaz `SensorEventListener`, del paquete `android.hardware`.

La clase Sensor acepta ocho tipos de sensores. Aunque, los sensores disponibles varían en función del dispositivo utilizado:

| | |
|---|---|
| `TYPE_ACCELEROMETER` | Acelerómetro |
| `TYPE_GYROSCOPE` | Sensor de giro |
| `TYPE_LIGHT` | Sensor de luz |
| `TYPE_MAGNETIC_FIELD` | Brújula |
| `TYPE_ORIENTATION` | Sensor de orientación |
| `TYPE_PRESSURE` | Sensor de presión |
| `TYPE_PROXIMITY` | Sensor de proximidad |
| `TYPE_TEMPERATURE` | Sensor de temperatura |



**Ejercicio paso a paso:** *Listar los sensores del dispositivo.*

No todos los dispositivos disponen de los mismos sensores. Por lo tanto, la primera tarea consiste en averiguar los sensores disponibles.

1. Crea un nuevo proyecto con nombre Sensores.
2. Añade la siguiente propiedad al `TextView` de *res/layout/main.xml*:

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

3. Inserta este código en la actividad principal:

```
public class SensoresActivity extends Activity {
    private TextView salida;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        salida = (TextView) findViewById(R.id.salida);
        SensorManager sensorManager = (SensorManager)
                                getSystemService(SENSOR_SERVICE);
        List<Sensor> listaSensores = sensorManager.
                                getSensorList(Sensor.TYPE_ALL);
        for(Sensor sensor: listaSensores) {
            log(sensor.getName());
        }
    }
    private void log(String string) {
        salida.append(string + "\n");
    }
}
```

4. El método comienza indicando el *Layout* de la actividad y obteniendo el *TextView* `salida`, donde mostraremos los resultados. A continuación, vamos a utilizar el método `getSystemService` para solicitar el sistema servicios específicos. Este método pertenece a la clase `Context` (como somos `Activity` también somos `Context`) y será muy utilizado para acceder a gran cantidad de servicios del sistema. Al indicar como parámetro `SENSOR_SERVICE`, indicamos que queremos utilizar los sensores. Lo haremos a través del objeto `sensorManager`. En primer lugar llamamos al método `getSensorList()` del objeto para que nos de `listaSensores`, una lista de objetos `Sensor`. La siguiente línea recorre todos los elementos de esta lista para llamar a su método `getName()` y mostrar el nombre de sensor.

5. Ejecuta el programa. Esta es una lista de los valores devueltos por el código anterior ejecutándose en el HTC Magic:

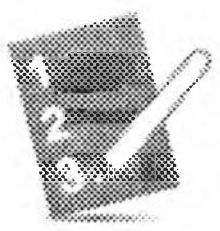
```
AK8976A 3-axis Accelerometer
AK8976A 3-axis Magnetic field sensor
AK8976A Orientation sensor
AK8976A Temperature sensor
```

6. El `AK8976A` es una combinación de acelerómetro de tres ejes y magnetómetro de tres ejes. Combinando la lectura de los campos gravitatorio y magnético terrestres proporciona también información de orientación. Incluye, además, un sensor interno de temperatura, útil para comprobar si el móvil se está calentado demasiado.

Como hemos visto la clase `Sensor` nos permite manipular los sensores. A continuación, se listan los métodos públicos de la clase `Sensor`:

| | |
|---|---|
| `public float getMaximumRange()` | Rango máximo en las unidades del sensor. |
| `public String getName()` | Nombre del sensor. |
| `public float getPower()` | Potencia (mA) usada por el sensor mientras está en uso. |
| `public float getResolution ()` | Resolución en las unidades del sensor. |
| `public int getType()` | Tipo genérico del sensor. |
| `Public String getVendor()` | Fabricante del sensor. |
| `public int getVersion()` | Versión del sensor. |

La clase `SensorManager` tiene además tres métodos (`getInclination`, `getOrientation` y `getRotationMatrix`), usados para calcular transformaciones de coordenadas.



**Ejercicio paso a paso:** *Acceso a los datos del sensor.*

Veamos ahora cómo obtener la lectura de cada uno de los sensores.

1. Copia el siguiente código al final de `onCreate`.

```
listaSensores = sensorManager.getSensorList(Sensor.TYPE_ORIENTATION);
if (!listaSensores.isEmpty()) {
  Sensor orientationSensor = listaSensores.get(0);
  sensorManager.registerListener(this, orientationSensor,
                                 SensorManager.SENSOR_DELAY_UI);}
listaSensores = sensorManager.getSensorList(Sensor.TYPE_ACCELEROMETER);
if (!listaSensores.isEmpty()) {
  Sensor acelerometerSensor = listaSensores.get(0);
  sensorManager.registerListener(this, acelerometerSensor,
                                 SensorManager.SENSOR_DELAY_UI);}
listaSensores = sensorManager.getSensorList(Sensor.TYPE_MAGNETIC_FIELD);
if (!listaSensores.isEmpty()) {
  Sensor magneticSensor = listaSensores.get(0);
  sensorManager.registerListener(this, magneticSensor,
                                 SensorManager.SENSOR_DELAY_UI);}
listaSensores = sensorManager.getSensorList(Sensor.TYPE_TEMPERATURE);
if (!listaSensores.isEmpty()) {
  Sensor temperatureSensor = listaSensores.get(0);
  sensorManager.registerListener(this, temperatureSensor,
                                 SensorManager.SENSOR_DELAY_UI);}
```

2. Comenzamos consultando si disponemos de un sensor de orientación. Para ello solicitamos al sistema que nos brinde todos los sensores de este tipo llamando a `getSensorList()`. Si la lista no está vacía obtenemos el primer elemento (el 0). Es necesario registrar cada tipo de sensor por separado para poder obtener información de él. El método `registerListener()` toma como primer parámetro un objeto que implemente el interface `SensorEventListener`, veremos a continuación cómo se implementa esta interfaz (se indica `this` porque la clase que estamos definiendo implementará este interfaz para recoger eventos de sensores). El segundo parámetro es el sensor que estamos registrando. Y el tercero indica al sistema con qué frecuencia nos gustaría recibir actualizaciones del sensor. Acepta cuatro posibles valores, de menor a mayor frecuencia tenemos: `SENSOR_DELAY_NORMAL`, `SENSOR_DELAY_UI`, `SENSOR_DELAY_GAME` y `SENSOR_DELAY_FASTEST`. Esta indicación sirve para que el sistema estime cuánta atención necesitan los sensores, pero no garantiza una frecuencia concreta.

3. Para que nuestra clase implemente el interface que hemos comentado añade a la declaración de la clase:

```
implements SensorEventListener
```

4. Para recibir los datos de los sensores tenemos que implementar dos métodos de la interfaz `SensorEventListener`:

```
@Override
public void onAccuracyChanged(Sensor sensor, int precision) {}
```

```
@Override
public void onSensorChanged(SensorEvent evento) {
  //Cada sensor puede provocar que un thread principal pase por
  //aquí, así que sincronizamos el acceso (se verá más adelante)
  synchronized (this) {
     switch(evento.sensor.getType()) {
     case Sensor.TYPE_ORIENTATION:
        for (int i=0 ; i<3 ; i++) {
           log("Orientación "+i+": "+evento.values[i]);
        }
        break;
     case Sensor.TYPE_ACCELEROMETER:
        for (int i=0 ; i<3 ; i++) {
           log("Acelerómetro "+i+": "+evento.values[i]);
        }
        break;
     case Sensor.TYPE_MAGNETIC_FIELD:
        for (int i=0 ; i<3 ; i++) {
           log("Magnetismo "+i+": "+evento.values[i]);
        }
        break;
     default:
        for (int i=0 ; i<evento.values.length ; i++) {
           log("Temperatura "+i+": "+evento.values[i]);
        }
     }
  }
}
```

5. Cuando implementamos un interface estamos obligados a implementar todos sus métodos. En este caso son dos. Para `onAccuracyChanged` no queremos ninguna acción específica, pero lo tenemos que incluir. Cuando un sensor cambie se llamará al método `onSensorChanged`, aquí comprobaremos qué sensor ha causado la llamada y leeremos los datos.

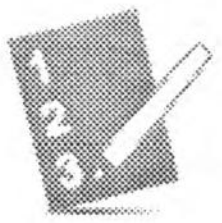
6. Verifica que el programa funcione correctamente.

Cuando el evento se dispara en el método `onSensorChanged` comprobamos qué sensor lo ha causado y leemos los datos. Los posibles valores devueltos se indican en la documentación de la clase `SensorEvent`[1].

[1] http://developer.android.com/reference/android/hardware/SensorEvent.html

### 5.5.1. Un programa que muestra los sensores disponibles y sus valores en tiempo real

La aplicación realizada en el ejercicio anterior resulta algo difícil de utilizar. En primer lugar, presupone que se dispone de cuatro sensores, cosa que no será cierta en muchos dispositivos. Además, los datos de los sensores cambian demasiado rápido para poder leerlos en un listado. El ejercicio mostrado a continuación es un resumen de los ejemplos anteriores, pero que permite mostrar en la misma pantalla los valores actuales de todos los sensores de un dispositivo. Además, es un buen ejemplo para ilustrar cómo crear una vista desde código.



**Ejercicio paso a paso:** *Creación de una vista desde código para mostrar los datos de los sensores.*

1. Crea un nuevo proyecto con nombre *Sensores2*.
2. Remplaza el código de la actividad por el siguiente:

```
public class Sensores extends Activity implements SensorEventListener {
    private LinearLayout mLinearLayout;
    private TextView aTextView[][] = new TextView[20][3];;
    private SensorManager mSensorManager;
    private Sensor aSensor[] = new Sensor[20];
    private List<Sensor> listSensors;

    @Override public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);
      mLinearLayout = (LinearLayout) findViewById(R.id.LinearLayout01);
      mSensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);
      listSensors = mSensorManager.getSensorList(Sensor.TYPE_ALL);
      int n = 0;
      for (Sensor sensor : listSensors) {
          aSensor[n] = sensor;
          TextView mTextView = new TextView(this);
          mTextView.setText(sensor.getName());
          mLinearLayout.addView(mTextView);
          LinearLayout nLinearLayout = new LinearLayout(this);
          mLinearLayout.addView(nLinearLayout);
          for (int i = 0; i < 3; i++) {
              aTextView[n][i] = new TextView(this);
              aTextView[n][i].setText("?");
              aTextView[n][i].setWidth(87);
          }
          TextView xTextView = new TextView(this);
          xTextView.setText("  X: ");
          nLinearLayout.addView(xTextView);
```

```
        nLinearLayout.addView(aTextView[n][0]);
        TextView yTextView = new TextView(this);
        yTextView.setText("  Y: ");
        nLinearLayout.addView(yTextView);
        nLinearLayout.addView(aTextView[n][1]);
        TextView zTextView = new TextView(this);
        zTextView.setText("  Z: ");
        nLinearLayout.addView(zTextView);
        nLinearLayout.addView(aTextView[n][2]);
        mSensorManager.registerListener(this, sensor,
          SensorManager.SENSOR_DELAY_UI);
            n++;
        }
    }

    @Override public void onAccuracyChanged(Sensor sensor, int
    accuracy) {}

    @Override public void onSensorChanged(SensorEvent event) {
      synchronized (this) {
          int n = 0;
          for (Sensor sensor: listSensors) {
              if (event.sensor == sensor) {
                  for (int i=0; i<event.values.length; i++) {
    aTextView[n][i].setText(Float.toString(event.values[i]));
                  }
              }
              n++;
          }
      }
    }
    }
```

3. Como puedes observar esta actividad utiliza el *Layout* creado por defecto que, básicamente, es un `LinearLayot` (en el código corresponde a la variable `mLinearLayout`) con un `TextView` en su interior (*"Hello Word ..."*). A `mLinearLayout` se le va a ir añadiendo una serie de vistas adicionales según los sensores encontrados en el dispositivo. Por cada sensor se añade: un `TextView` con el nombre del sensor, un `LinearLayot` de tipo horizontal para contener a su vez un `TextView` con " X: ", un `TextView` con el valor del sensor en el eje X, un `TextView` con " Y ", un `TextView` con el valor del sensor en el eje Y, un `TextView` con " Z: " y un `TextView` con el valor del sensor en el eje Z. Las referencias a los `TextView` donde se visualizarán los valores de los sensores se almacenan en el array `aTextView[][]` donde el primer índice identifica el número de sensor y el segundo la dimensión X, Y o Z. Por otra parte, los sensores encontrados son almacenados en el array `aSensor[]`.

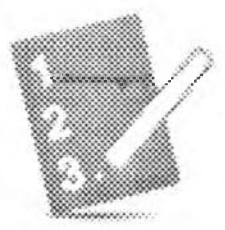
4. En el método `onSensorChanged()` se hace un bucle para localizar el índice del sensor que ha cambiado y se modifican los `TextView` correspondientes al sensor con los valores leídos.

***NOTA:*** *No todos los sensores tienen tres dimensiones. Por ejemplo, en el caso del sensor de temperatura solo se cambiará en el valor de X.*

5. Verifica sobre un dispositivo real que el programa funciona correctamente.

## 5.5.2. Utilización de los sensores en Asteroides

A continuación, proponemos una serie de ejercicios y prácticas para manejar la nave de Asteroides utilizando sensores.



**Ejercicio paso a paso:** *Manejo de la nave con el sensor de orientación.*

1. En primer lugar, implementa la interfaz `SensorEventListener`.

```
public class VistaJuego extends View implements SensorEventListener {
```

2. En el constructor registra el sensor e indica que nuestro objeto recogerá la llamada *callback*:

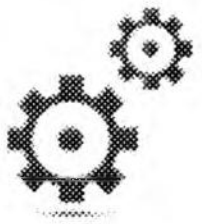
```
SensorManager mSensorManager = (SensorManager)
context.getSystemService(Context.SENSOR_SERVICE);
List<Sensor> listSensors = mSensorManager.getSensorList(
                                        Sensor.TYPE_ORIENTATION);
if (!listSensors.isEmpty()) {
  Sensor orientationSensor = listSensors.get(0);
  mSensorManager.registerListener(this, orientationSensor,
                SensorManager.SENSOR_DELAY_GAME);
}
```

3. Añade los siguientes dos métodos que implementan la interfaz `SensorEventListener`:

```
@Override
public void onAccuracyChanged(Sensor sensor, int accuracy){}

private boolean hayValorInicial = false;
private float valorInicial;

@Override
public void onSensorChanged(SensorEvent event) {
  float valor = event.values[1];
  if (!hayValorInicial){
     valorInicial = valor;
     hayValorInicial = true;
```

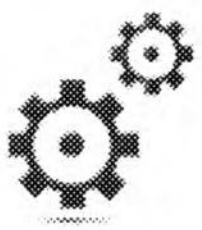
```
    }
    giroNave=(int) (valor-valorInicial)/3 ;
}
```

4. Prueba la aplicación. Has de tener cuidado de que el terminal esté en una posición cómoda al entrar en la actividad `Juego`, dado que el movimiento de la nave se obtiene con la diferencia de la posición del terminal con respecto a la posición inicial.



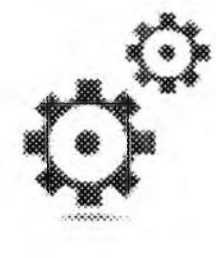
**Práctica**: *Manejo de la nave con sensor de aceleración.*

Modifica el ejemplo anterior para utilizar el sensor de aceleración en lugar del de orientación. Gracias a la fuerza de gravedad que la Tierra ejerce sobre el terminal podremos saber si este está horizontal. En caso de que la nave esté horizontal (o casi) no ha de girar, pero cuando el terminal se incline, la nave debe girar proporcionalmente a esta inclinación. Utiliza los programas anteriores para descubrir qué eje (x, y o z) es el que te interesa y el rango de valores que proporciona.



**Práctica**: *Aceleración de la nave con sensores.*

¿Te animarías a controlar la aceleración de la nave con los sensores? Ten cuidado de que no acelere con mucha facilidad, este juego resulta muy difícil cuando la nave está en movimiento. Puede ser una buena idea que permitas también desacelerar la nave.



**Práctica**: *Configuración de tipo de entrada en preferencias.*

Todos los controles de la nave (teclado, pantalla táctil y sensores) están activados simultáneamente. El teclado y la pantalla táctil no interfieren cuando el usuario no quiere utilizarlos. Sin embargo, la activación de los sensores sí que molestará a los usuarios que no quieran utilizar este método de entrada.

1. Crea nuevas entradas en la configuración para activar o desactivar cada tipo de entrada (o al menos la de los sensores).
2. Modifica el código anterior para que se desactiven las entradas que el usuario no haya seleccionado.

## 5.6. Uso de hilos de ejecución (Threads)

Cuando se lanza una nueva aplicación el sistema crea un nuevo hilo de ejecución (*thread*) para esta aplicación conocida como hilo principal. Este hilo es muy importante dado que se encarga de atender los eventos de los distintos

componentes. Es decir, este hilo ejecuta los métodos `onCreate()`, `onDraw()`, `onKeyDown()`. Por esta razón al hilo principal también se lo conoce como hilo del interfaz de usuario.

El sistema no crea un hilo independiente cada vez que se crea un nuevo componente. Es decir, todas las actividades y servicios de una aplicación son ejecutados por el hilo principal.

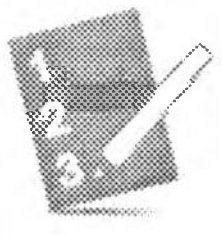
Cuando tu aplicación ha de realizar trabajo intensivo como respuesta a una interacción de usuario, hay que tener cuidado porque es posible que la aplicación no responda de forma adecuada. Por ejemplo, imagina que has de esperar para descargar unos datos de Internet, si lo haces en el hilo de ejecución principal este quedará bloqueado a la espera de que termine la descarga. Por lo tanto, no se podrá redibujar la vista (`onDraw()`) o atender eventos del usuario (`onKeyDown()`). Desde el punto de vista del usuario, se tendrá la impresión de que la aplicación se ha colgado. Mas todavía, si el hilo principal es bloqueado más de 5 segundos, el sistema mostrará un cuadro de diálogo al usuario *"La aplicación no responde"* para que el usuario decida si quieres esperar o detener la aplicación.

La solución en estos casos es crear un nuevo hilo de ejecución, para que realice este trabajo intensivo. De esta forma no bloqueamos el hilo principal, que puede seguir atendiendo los eventos de usuario. Es decir, cuando estés implementando un método del hilo principal (empieza por `on...`) nunca realices una acción que pueda bloquear este hilo, como cálculos largos o que requieran esperar mucho tiempo. En estos casos hay que crear un nuevo hilo de ejecución y encomendarle esta tarea. En el siguiente apartado describiremos cómo hacerlo.

Las herramientas del interfaz de usuario de Android han sido diseñadas para ser ejecutadas desde un único hilo de ejecución, el hilo principal. Por lo tanto, no se permite manipular el interfaz de usuario desde otros hilos de ejecución.

## 5.6.1. Introduciendo movimiento en Asteroides

Para que el juego cobre vida será necesario animar todos los gráficos introducidos. En el siguiente ejercicio veremos cómo hacerlo.



**Ejercicio paso a paso:** *Introduciendo movimiento en Asteroides.*

1. Comienza declarando las siguientes variables en la clase `VistaJuego`:

```
// //// THREAD Y TIEMPO //////
// Thread encargado de procesar el juego
private ThreadJuego thread = new ThreadJuego();
// Cada cuanto queremos procesar cambios (ms)
private static int PERIODO_PROCESO = 50;
// Cuando se realizó el último proceso
private long ultimoProceso = 0;
```

2. La animación del juego la llevará a cabo el método `actualizaFisica()` que será ejecutado a intervalos regulares, definidos por la constante `PERIODO_PROCESO`. Esta constante ha sido inicializada a 50 ms. En `ultimoProceso` se almacena el instante en que se llamó, por última vez, a `actualizaFisica()`.

3. Copia el siguiente método dentro de la clase `VistaJuego`:

```
protected void actualizaFisica() {
  long ahora = System.currentTimeMillis();
  // No hagas nada si el período de proceso no se ha cumplido.
  if (ultimoProceso + PERIODO_PROCESO > ahora) {
     return;
  }
  // Para una ejecución en tiempo real calculamos retardo
  double retardo = (ahora - ultimoProceso) / PERIODO_PROCESO;
  ultimoProceso = ahora; // Para la próxima vez
  // Actualizamos velocidad y dirección de la nave a partir de
  // giroNave y aceleracionNave (según la entrada del jugador)
  nave.setAngulo((int) (nave.getAngulo() + giroNave * retardo));
  double nIncX = nave.getIncX() + aceleracionNave *
        Math.cos(Math.toRadians(nave.getAngulo())) * retardo;
  double nIncY = nave.getIncY() + aceleracionNave *
        Math.sin(Math.toRadians(nave.getAngulo())) * retardo;
  // Actualizamos si el módulo de la velocidad no excede el máximo
  if (Math.hypot(nIncX,nIncY) <= Grafico.getMaxVelocidad()){
     nave.setIncX(nIncX);
     nave.setIncY(nIncY);
  }
  // Actualizamos posiciones X e Y
  nave.incrementaPos(retardo);
  for (Grafico asteroide : Asteroides) {
     asteroide.incrementaPos(retardo);
  }
}
```

4. Como veremos a continuación, este método será llamado de forma continua. Como queremos desplazar los gráficos cada `PERIODO_PROCESO` milisegundos, verificamos si ya ha pasado este tiempo desde la última vez que se ejecutó (`ultimoProceso`).

5. Como también es posible que el sistema esté ocupado y no nos haya podido llamar hasta un tiempo superior a `PERIODO_PROCESO`, vamos a calcular el factor de retardo en función del tiempo adicional que haya pasado. Si por ejemplo, desde la última llamada ha pasado dos veces `PERIODO_PROCESO`, la variable `retardo` ha de valer 2. Lo que significará que los gráficos han de desplazarse el doble que en circunstancias normales. De esta forma conseguiremos un desplazamiento continuo en tiempo real.

6. A continuación, se actualizan las variables que controlan la aceleración y cambios de dirección de la nave. Se consiguen por medio de las variables `aceleracionNave` y `giroNave`. En el siguiente capítulo, modificaremos estas variables para que el jugador pueda pilotear la nave. A partir de estas variables se obtiene una nueva velocidad de la nave, descompuesta en sus componentes X e Y. Si el módulo de estos componentes es mayor que la velocidad máxima permitida, no se actualizará la velocidad.

7. Finalmente, se actualizan las posiciones de todos los gráficos (nave y asteroides) a partir de sus velocidades. Esto se consigue llamando al método `incrementaPos()` definido en la clase `Grafico`.

8. Ahora necesitamos que esta función sea llamada continuamente, para lo que utilizaremos un *Thread*. Crea la siguiente clase dentro de la clase `VistaJuego`:

```
class ThreadJuego extends Thread {
  @Override
  public void run() {
     while (true) {
        actualizaFisica();
     }
  }
}
```

9. Introducir esta línea al final del método `onSizeChanged()`:

```
thread.start();
```

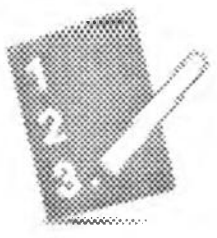
10. Esto ocasionará que se llame al método `run()` del hilo de ejecución. Este método es un bucle infinito que, continuamente, llama al `actualizaFisica()`.

11. Ejecuta la aplicación y observa cómo el juego cobra vida.

El trabajo con hilos de ejecución es especialmente delicado. Como veremos en próximos capítulos, este código nos va a ocasionar varios quebraderos de cabeza. Un problema es que seguirá ejecutándose aunque nuestra aplicación esté en segundo plano. Veremos cómo detener el hilo de ejecución, cuando estudiemos el ciclo de vida de una actividad (***NOTA:*** *Si ejecutas el programa en el terminal real y detectas que este funciona más lentamente, puede ser buena idea detener la aplicación*). Un segundo problema, aparecerá cuando dos hilos de ejecución traten de acceder a la misma variable a la vez. También se resolverá más adelante.

## 5.7. Introduciendo un misil en Asteroides

Para poder disparar a los asteroides va a ser necesario introducir un misil en el juego. En el siguiente ejercicio aprenderemos a hacerlo:



**Ejercicio paso a paso:** *Introduciendo un misil en Asteroides.*

1. En primer lugar, añade las siguientes variables a la clase `VistaJuego`:

```
// //// MISIL //////
private Grafico misil;
private static int PASO_VELOCIDAD_MISIL = 12;
private boolean misilActivo = false;
private int tiempoMisil;
```

2. Para trabajar con gráficos vectoriales, puedes crear en el constructor la variable `drawableMisil` de la siguiente forma:

```
ShapeDrawable dMisil = new ShapeDrawable(new RectShape());
dMisil.getPaint().setColor(Color.WHITE);
dMisil.getPaint().setStyle(Style.STROKE);
dMisil.setIntrinsicWidth(15);
dMisil.setIntrinsicHeight(3);
drawableMisil = dMisil;
```

3. Crea la variable `drawableMisil` para el caso en que se deseen gráficos en *bitmap*, utilizando el fichero `misil1.png`.

4. Inicializa el objeto `misil` de forma similar a como se ha hecho en `nave`.

5. En el método `onDraw()` dibuja `misil`, solo si lo indica la variable `misilActivo`.

6. Quita los comentarios de las llamadas a `activaMisil()`

7. En el método `actualizaFisica()` añade las siguientes líneas:

```
// Actualizamos posición de misil
if (misilActivo) {
  misil.incrementaPos(retardo);
  tiempoMisil--;
  if (tiempoMisil < 0) {
     misilActivo = false;
  } else {
     for (int i = 0; i < Asteroides.size(); i++)
     if (misil.verificaColision(Asteroides.elementAt(i))) {
        destruyeAsteroide(i);
        break;
     }
  }
}
```

8. Añade los siguientes dos métodos:

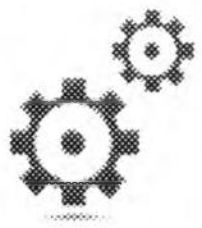
```
private void destruyeAsteroide(int i) {
  Asteroides.remove(i);
  misilActivo = false;
}

private void ActivaMisil() {
```

```
        misil.setPosX(nave.getPosX()+ nave.getAncho()/2-misil.getAncho()/2);
        misil.setPosY(nave.getPosY()+ nave.getAlto()/2-misil.getAlto()/2);
        misil.setAngulo(nave.getAngulo());
        misil.setIncX(Math.cos(Math.toRadians(misil.getAngulo())) *
                                                    PASO_VELOCIDAD_MISIL);
        misil.setIncY(Math.sin(Math.toRadians(misil.getAngulo())) *
                                                    PASO_VELOCIDAD_MISIL);
        tiempoMisil = (int) Math.min(this.getWidth() / Math.abs( misil.
            getIncX()), this.getHeight() / Math.abs(misil.getIncY())) - 2;
        misilActivo = true;
    }
```

9. Este último método requiere alguna explicación. Cuando se quiere activar un nuevo misil este debe partir del centro de la nave. Como las coordenadas X, Y de `Grafico`, corresponden a la esquina superior izquierda hay que hacer algunos ajustes. El ángulo del misil ha de ser el mismo que actualmente tiene la nave. El módulo de la velocidad de la nave nos lo indica la constante `PASO_VELOCIDAD_MISIL`. Para descomponerlo en sus componentes X e Y utilizamos el coseno y el seno. Dada la naturaleza del espacio del juego (lo que sale por un lado aparece por el otro) si disparáramos un misil este podría acabar chocando contra la nave. Para solucionarlo, vamos a dar un tiempo de vida al misil para impedir que pueda llegar de nuevo a la nave (`tiempoMisil`). Para obtener este tiempo nos quedamos con el mínimo entre el ancho dividido la velocidad en X y el alto dividido entre la velocidad en Y. Luego le restamos una constante. Terminamos activando el misil.

10. Verifica que todo funciona correctamente.

> ***NOTA:*** *Este código es posible que de algún problema de acceso concurrente a los datos desde dos threads diferentes. Se resolverá en el siguiente capítulo.*



**Práctica**: *Disparando varios misiles a la vez.*

Tal y como se ha planteado en el código solo es posible lanzar un misil cada vez. Si disparamos un segundo misil el primero desaparece. ¿Podrías modificar el código para que se pudieran lanzar tantos misiles cómo quisieras? Consejo: Observa como se ha resuelto para el caso de los asteroides, la solución para los misiles puede ser muy similar.

# CAPÍTULO 6.
# Multimedia y ciclo de vida de una actividad

Una de las funciones más habituales de los modernos teléfonos móviles es su utilización como reproductores multimedia. Su uso más frecuente es como reproductores MP3, para la reproducción de vídeos y televisión a través de Internet. El API de Android viene preparado con excelentes características de reproducción multimedia, permite la reproducción de gran variedad de formatos, tanto de audio como de vídeo. El origen de los datos puede ser tanto un fichero local como un *stream* obtenido desde Internet. Todo este trabajo lo realiza principalmente una clase `MediaPlayer`.

Antes de comenzar la descripción de estos contenidos, comenzaremos el capítulo con un aspecto de vital importancia en el desarrollo de aplicaciones en Android, el ciclo de vida de una actividad. Es decir, cómo las aplicaciones son creadas, puestas en espera y finalmente destruidas.

**Objetivos**:

- Comprender el ciclo de vida de una actividad Android.
- Utilizar de forma correcta los diferentes eventos relacionados con el ciclo de vida.
- Aprender cuándo y cómo guardar el estado de una actividad.
- Repasar las facilidades multimedia disponibles en Android, qué formatos soporta y las clases que hemos de utilizar.
- Describir la clase MediaPlayer, utilizada para la reproducción de audio y video.
- Dotar a la aplicación Asteroides de un control adecuado de su ciclo de vida y de varios efectos de audio.

## 6.1. Ciclo de vida de una actividad

El ciclo de vida de una aplicación Android es bastante diferente del ciclo de vida de una aplicación en otros S.O., como Windows. La mayor diferencia es que, en Android el ciclo de vida es controlado principalmente por el sistema, en lugar de ser controlado directamente por el usuario.

Una aplicación en Android va a estar formada por un conjunto de elementos básicos de visualización, conocidos como actividades. Además de varias actividades una aplicación también puede contener servicios. El ciclo de vida de los servicios será estudiado en el capítulo 8. Son las actividades las que realmente controlan el ciclo de vida de las aplicaciones, dado que el usuario no cambia de aplicación, sino de actividad. El sistema va a mantener una pila con las actividades previamente visualizadas, de forma que el usuario va a poder regresar a la actividad anterior pulsando la tecla "atrás".

Una aplicación Android corre dentro de su propio proceso Linux. Este proceso es creado para la aplicación y continuará vivo hasta que ya no sea requerido y el sistema reclame su memoria para asignársela a otra aplicación.

Una característica importante, y poco usual, de Android es que la destrucción de un proceso no es controlado directamente por la aplicación. En lugar de esto, es el sistema quien determina cuando destruir el proceso, basándose en el conocimiento que tiene el sistema de las partes de la aplicación que están corriendo (actividades y servicios), acerca de qué tan importantes son para el usuario y cuánta memoria disponible hay en un determinado momento.

Si tras eliminar el proceso de una aplicación, el usuario vuelve a ella, se creará de nuevo el proceso, pero se habrá perdido el estado que tenía esta aplicación. En estos casos, va a ser responsabilidad del programador almacenar el estado de las actividades, si queremos que cuando sea reiniciada conserve su estado.

Como vemos, Android es sensible al ciclo de vida de una actividad, por lo tanto necesitas comprender y manejar los eventos relacionados con el ciclo de vida si quieres crear aplicaciones estables.

Una actividad en Android puede estar en uno de estos cuatro estados:

**Activa** (*Running*): La actividad está encima de la pila, lo que quiere decir que es visible y tiene el foco.

**Visible** (*Paused*): La actividad es visible pero no tiene el foco. Se alcanza este estado cuando pasa a activa otra actividad con alguna parte transparente o que no ocupa toda la pantalla. Cuando una actividad está tapada por completo, pasa a estar parada.

**Parada** (*Stopped*): Cuando la actividad no es visible, se recomienda guardar el estado de la interfaz de usuario, preferencias, etc.

**Destruida** (*Destroyed*): Cuando la actividad termina al invocarse el método *finish()*, o es matada por el sistema Android, sale de la pila de actividades.

Cada vez que una actividad cambia de estado se van a producir eventos que podrán ser capturados por ciertos métodos de la actividad. A continuación, se muestra un esquema que ilustra los métodos que capturan estos eventos.

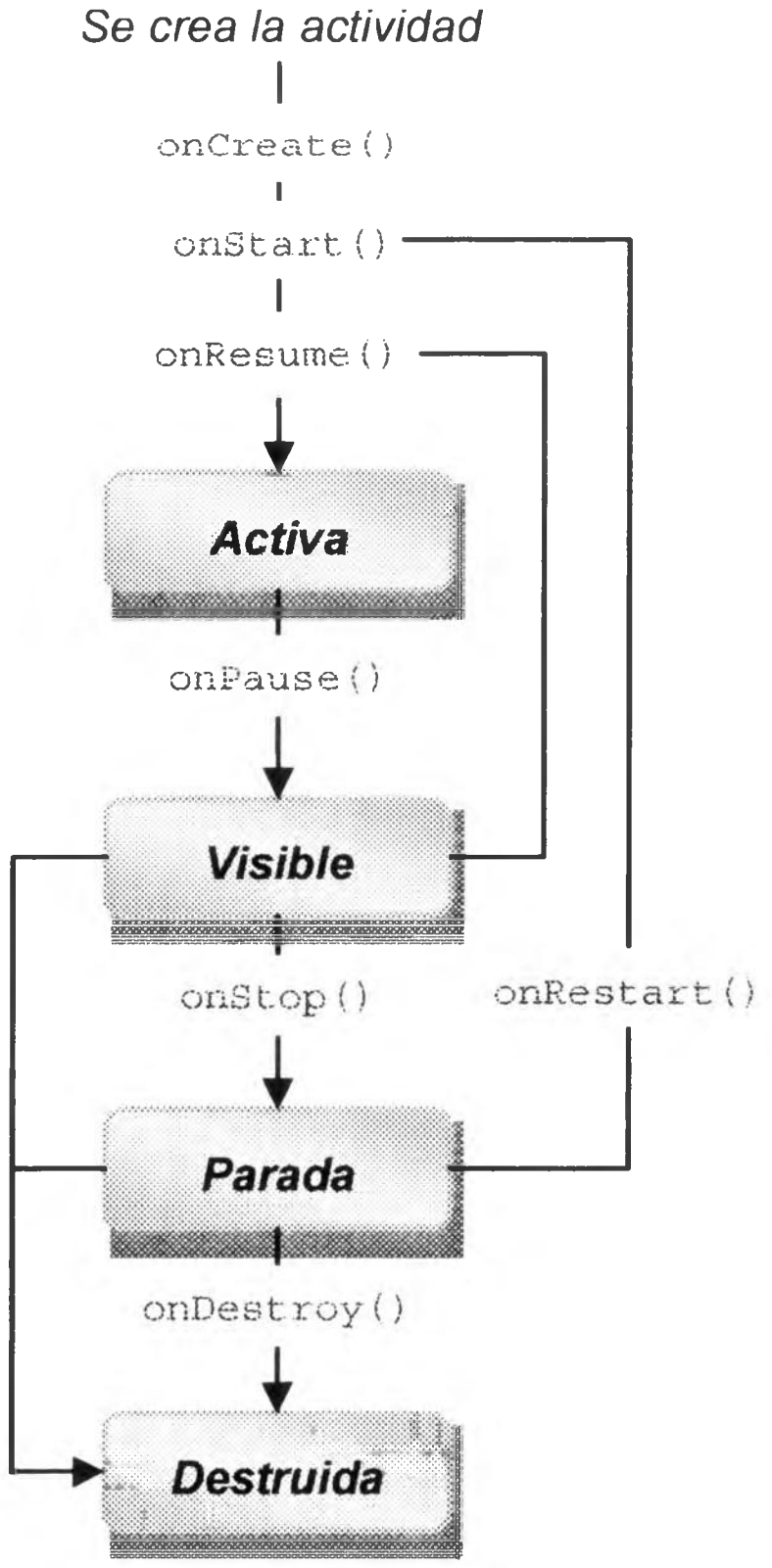



***Figura 1:*** *Ciclo de vida de una actividad.*

**onCreate(Bundle)**: Se llama en la creación de la actividad. Se utiliza para realizar todo tipo de inicializaciones, como la creación de la interfaz de usuario o la inicialización de estructuras de datos. Puede recibir información de estado de instancia (en una instancia de la clase `Bundle`), por si se reanuda desde una actividad que ha sido destruida y vuelta a crear.

**onStart()**: Nos indica que la actividad está a punto de ser mostrada al usuario.

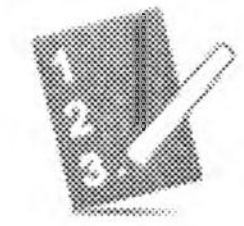
**onResume()**: Se llama cuando la actividad va a comenzar a interactuar con el usuario. Es un buen lugar para lanzar las animaciones y la música.

**onPause()**: Indica que la actividad está a punto de ser lanzada a segundo plano, normalmente porque otra aplicación es lanzada. Es el lugar adecuado para detener animaciones, música o almacenar los datos que estaban en edición.

**onStop()**: La actividad ya no va a ser visible para el usuario. Ojo si hay muy poca memoria, es posible que la actividad se destruya sin llamar a este método.

**onRestart()**: Indica que la actividad va a volver a ser representada después de haber pasado por *onStop()*.

**onDestroy()**: Se llama antes de que la actividad sea totalmente destruida. Por ejemplo, cuando el usuario pulsa el botón <volver> o cuando se llama al método *finish()*. Ojo si hay muy poca memoria, es posible que la actividad se destruya sin llamar a este método.



**Ejercicio paso a paso:** *¿Cuándo se llama a los eventos del ciclo de vida en una actividad?*

En este ejercicio vamos a implementar todos los métodos del ciclo de vida de la actividad principal de Asteroides y añadiremos un `Toast` para mostrar cuando son ejecutados. De esta forma comprenderemos mejor cuándo se llama a cada método.

1. Abre la actividad `Asteroides` del proyecto *Asteroides*.
2. Añade en el método `onCreate()` el siguiente código:

```
Toast.makeText(this, "onCreate", Toast.LENGTH_SHORT).show();
```

3. Añade los siguientes métodos:

```
@Override protected void onStart() {
  super.onStart();
  Toast.makeText(this, "onStart", Toast.LENGTH_SHORT).show();
}

@Override protected void onResume() {
  super.onResume();
  Toast.makeText(this, "onResume", Toast.LENGTH_SHORT).show();
}

@Override protected void onPause() {
  Toast.makeText(this, "onPause", Toast.LENGTH_SHORT).show();
  super.onPause();
}

@Override protected void onStop() {
  Toast.makeText(this, "onStop", Toast.LENGTH_SHORT).show();
  super.onStop();
}

@Override protected void onRestart() {
  super.onRestart();
  Toast.makeText(this, "onRestart", Toast.LENGTH_SHORT).show();
}
```

```
@Override protected void onDestroy() {
  Toast.makeText(this, "onDestroy", Toast.LENGTH_SHORT).show();
  super.onDestroy();
}
```

4. Ejecuta la aplicación y observa la secuencia de `Toast`.
5. Pulsa el botón *Acerca de* y luego regresa a la actividad. Observa la secuencia de `Toast`.
6. Pulsa el botón *Jugar* y luego regresa a la actividad. Observa la secuencia de `Toast`.
7. Sal de la actividad y observa la secuencia de `Toast`.

**Ejercicio paso a paso:** *Aplicando eventos del ciclo de vida en la actividad Juego de Asteroides.*

Asteroides gestiona el movimiento de los objetos gráficos por medio de un *thread* que se ejecuta continuamente. Cuando la aplicación pasa a segundo plano, este *thread* continúa ejecutándose, por lo que puede hacer que nuestro teléfono funcione más lentamente. Este problema aparece por una gestión incorrecta del ciclo de vida. En el siguiente ejercicio veremos cómo solucionarlo:

1. Abre el proyecto *Asteroides* y ejecútalo en un terminal con una versión 2.3.x.
2. Pulsa el botón "Jugar" y cuando esté en mitad de la partida pulsa con el botón de inicio para dejar a la actividad en estado *parada*. Las actividades en este estado no tendrían que consumir demasiados recursos. Sin embargo, Asteroides sí que lo hace, para verificarlos utiliza la aplicación "Administrador de tareas" (disponible en las últimas versiones de Android). El resultado puede ser similar al que se muestra a continuación:

Como puedes ver la aplicación Asteroides está consumiendo casi el 50% del uso de CPU.

Evidentemente algo hemos hecho mal. No es lógico que cuando la actividad `Juego` esté en segundo plano se siga llamando a `actualizaFisica()`. En este ejercicio aprenderemos a solucionarlo.

3. Incluye la siguiente variable en la actividad `Juego`.

```
private VistaJuego vistaJuego;
```

4. Al final de `onCreate` añade:

```
vistaJuego = (VistaJuego) findViewById(R.id.VistaJuego);
```

5. Incorpora los siguientes métodos a la actividad:

```
@Override protected void onPause() {
  super.onPause();
  vistaJuego.getThread().pausar();
}

@Override protected void onResume() {
  super.onResume();
  vistaJuego.getThread().reanudar();
}

@Override protected void onDestroy() {
  vistaJuego.getThread().detener();
  super.onDestroy();
}
```

Lo que intentamos hacer con este código es poner en pausa el thread secundario, cuando la actividad deje de estar activa, y reanudarlo, cuando la actividad recupere el foco. Además, detener el thread cuando la actividad vaya a ser destruida.

***NOTA:*** *realmente el thread será destruido al destruirse la actividad que lo ha lanzado. No obstante, puede resultar interesante hacerlo lo antes posible.*

Observa cómo los eventos del ciclo de vida solo pueden ser escritos en un `Activity`, por lo que no sería válido hacerlo en `VistaJuego`.

6. Abre la clase `VistaJuego` y busca la definición de la clase `ThreadJuego` y remplázala por la siguiente:

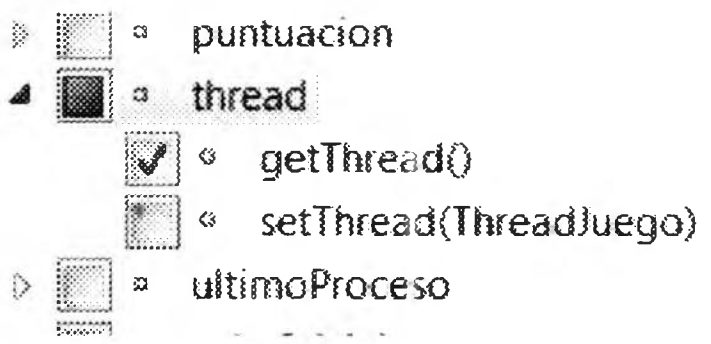
```
class ThreadJuego extends Thread {

  private boolean pausa,corriendo;

  public synchronized void pausar() {
     pausa = true;
  }
```

```
public synchronized void reanudar() {
    pausa = false;
    notify();
}

public void detener() {
    corriendo = false;
    if (pausa) reanudar();
}

@Override
public void run() {
    corriendo = true;
    while (corriendo) {
        actualizaFisica();
        synchronized (this) {
            while (pausa) {
                try {
                    wait();
                } catch (Exception e) {
                }
            }
        }
    }
}
```

Comenzamos declarando las variables `pausa`, `corriendo`. Estas pueden ser modificadas mediante los métodos `pausar()`, `reanudar()` y `detener()`.

7. La palabra reservada `synchronized` impide el acceso concurrente a una sección del código. En el siguiente capítulo explicaremos su utilización. El método `run()` se ha modificado de manera que en lugar de ser un bucle infinito, permitimos que termine poniendo la variable `corriendo` a `false`. Luego, tras llamar a `actualizaFisica()`, se comprueba si se ha activado `pausa`. En tal caso, se entra en un bucle donde ponemos en espera el *thread* llamando al método `wait()`. Este quedará bloqueado hasta que se llame a `notify()`. Esta acción se realizará desde el método `reanudar()`. La llamada a `wait()` puede lanzar excepciones por lo que es obligatorio escribirla dentro de un bloque `try {…} catch {…}`.

8. Dado que la variable `thread` es de tipo `private`, no puede ser manipulada desde fuera de `VistaJuego`. Para poder llamar a los métodos de este objeto (pausar, reanudar) vamos a incluir un método *getter*. Para ello, sitúa el cursor justo antes de la última llave de `VistaJuego`. Pulsa con el botón derecho en el código y selecciona la opción *Source / Generate Getters and Setters…* Marca solo el método `getThread()` tal y como se muestra a continuación:




Se insertará el siguiente código:

```
public ThreadJuego getThread() {
  return thread;
}
```

9. Ejecuta de nuevo la aplicación y repite el segundo punto de este ejercicio. En este caso el resultado ha de ser similar al siguiente:


## 6.1.1. ¿Qué proceso se elimina?

Como hemos comentado Android mantiene en memoria todos los procesos que quepan aunque estos no se estén ejecutando. Una vez que la memoria está llena y el usuario decide ejecutar una nueva aplicación, el sistema ha de determinar qué proceso de los que están en ejecución ha de ser eliminado. Android ordena los procesos en una lista jerárquica, asignándole a cada uno una determinada "importancia". Esta lista se confecciona basándose en los componentes de la aplicación que están corriendo (actividades y servicios) y el estado de estos componentes.

Para establecer esta jerarquía de importancia se distinguen los siguientes tipos de procesos:

**Proceso de primer plano** (*Foreground process*): Hospeda una actividad en la superficie de la pantalla, con la cual el usuario está interactuando (su método `onResume()` ha sido llamado). Debería haber solo uno o unos pocos procesos de este tipo. Sólo serán eliminados cómo último recurso,

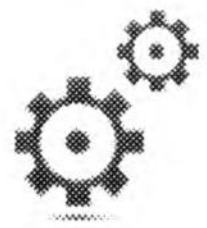
si es que la memoria está tan baja que ni siquiera estos procesos pueden continuar corriendo.

**Proceso visible** (*Visible process*): Hospeda una actividad que está visible en la pantalla, pero no en el primer plano (su método `onPause()` ha sido llamado). Considerado importante, no será eliminado a menos que sea necesario para mantener los procesos de primer plano.

**Proceso de servicio** (*Service process*): Hospeda un servicio que ha sido inicializado con el método `startService()`. Aunque estos procesos no son directamente visibles al usuario, generalmente están haciendo tareas que para el usuario son importantes (tales como reproducir un archivo mp3 o mantener una conexión con un servidor de contenidos). El sistema siempre tratará de mantener esos procesos corriendo, a menos que los niveles de memoria comiencen a comprometer el funcionamiento de los procesos de primer plano o visibles.

**Proceso de fondo** (*Background process*): Hospeda una actividad que no es actualmente visible al usuario (su método `onStop()` ha sido llamado). Si estos procesos son eliminados no tendrán un directo impacto en la experiencia del usuario. Generalmente, hay muchos de estos procesos, así el sistema asegura que el último proceso visto por el usuario sea el último en ser eliminado.

**Proceso vacío** (*Empty process*): No hospeda a ningún componente de aplicación activo. La única razón para mantener ese proceso es tener un "caché" que permita mejorar el tiempo de activación en la próxima vez que un componente de su aplicación sea ejecutado.



**Práctica**: *Aplicando eventos del ciclo de vida en la actividad inicial de Asteroides.*

Los conceptos referentes al ciclo de vida de una actividad son imprescindibles para el desarrollo de aplicaciones estables en Android. Para reforzar estos conceptos te proponemos el siguiente ejercicio en el que vamos a reproducir una música de fondo.

1. Abre el proyecto Asteroides.
2. Busca un fichero de audio (los formatos soportados por Android se listan en este capítulo). Renombra este fichero a *audio.xxx* y cópialo a la carpeta *res/raw*.

***NOTA:*** *Cada vez que ejecutes el proyecto este fichero será añadido al paquete .apk. Si este fichero es muy grande la aplicación también lo será, lo que ralentizará su instalación. Para agilizar la ejecución te recomendamos un fichero muy pequeño, por ejemplo un .mp3 de corta duración o un fichero MIDI (.mid). Si no encuentras ninguno puedes descargar uno de la Web del curso.*

3. Abre la actividad `Asteroides` y añade las siguientes líneas en el método `onCreate()`:

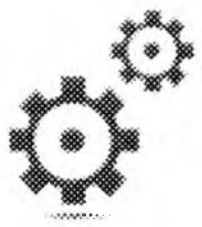
```
MediaPlayer mp = MediaPlayer.create(this, R.raw.audio);
mp.start();
```

4. Ejecuta el proyecto y verifica que cuando sales de la actividad la música sigue sonando un cierto tiempo.

5. Utilizando los eventos del ciclo de vida queremos que cuando la actividad deje de estar **activa** el audio deje de escucharse. Puedes utilizar los métodos:

```
mp.pause();
mp.start();
```

6. Verifica que funciona correctamente.



**Práctica:** *Aplicando eventos del ciclo de vida en la actividad inicial de Asteroides (II).*

1. Tras realizar el ejercicio anterior, ejecuta la aplicación y pulsa en el botón *Acerca de...* La música ha de detenerse.

2. Nos interesa que, mientras parte de esta actividad esté visible (como ha ocurrido en el punto anterior), la música se escuche. Es decir, utilizando los eventos del ciclo de vida, queremos que cuando la actividad deje de estar **visible** el audio deje de escucharse.

3. Verifica que funciona correctamente.

## 6.1.2. Guardando el estado de una actividad

Cuando el usuario ha estado utilizando una actividad, y tras cambiar a otras, regresa a la primera, lo habitual es que esta permanezca en memoria y continúe su ejecución sin alteraciones. Como hemos explicado, en situaciones de escasez de memoria, es posible que el sistema haya eliminado el proceso que ejecutaba la actividad. En este caso, el proceso será ejecutado de nuevo, pero se habrá perdido su estado, es decir, se habrá perdido el valor de sus variables y el puntero de programa. Como consecuencia, si el usuario estaba rellenando un cuadro de texto o estaba reproduciendo un audio en un punto determinado perderá esta información. En este apartado estudiaremos un mecanismo sencillo que nos proporciona Android para resolver este problema.

> ***NOTA:*** *Cuando se ejecuta una actividad sensible a la inclinación del teléfono, es decir que puede verse en horizontal o en vertical, se presenta un problema similar al anterior. La actividad es destruida y vuelta a construir con las nuevas dimensiones de pantalla y, por lo tanto, se llama de nuevo al método* `onCreate()`. *Antes de que la actividad sea destruida también resulta fundamental guardar su estado.*

Dependiendo de lo sensible que sea la pérdida de la información de estado se podrá actuar de diferentes maneras. Una posibilidad es, por ejemplo, no hacer nada. Siguiendo con el ejemplo anterior, el usuario tendría que volver a rellenar el cuadro de texto o el audio volvería a reproducirse desde el principio.

En caso de tratarse de información extremadamente sensible, se recomienda el uso del método `onPause()` para guardar el estado y `onResume()` para recuperarlo. Por supuesto, la información sensible ha de almacenarse en un medio permanente, como un fichero o una base de datos. Se trata del método más fiable, como vimos en el ciclo de vida de una actividad, los métodos `onStop()` y `onDestroy()` no tienen la garantía de ser llamados.

También tenemos una tercera posibilidad que, aunque no tenga una fiabilidad tan grande, resulta mucho más sencilla. Consiste en utilizar los siguientes dos métodos:

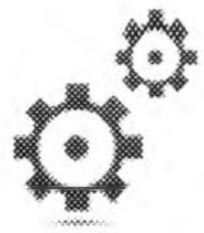
**onSaveInstanceState(Bundle)**: Se invoca para permitir a la actividad guardar su estado. Ojo, si hay muy poca memoria, es posible que la actividad se destruya sin llamar a este método.

**onRestoreInstanceState(Bundle)**: Se invoca para recuperar el estado guardado por `onSaveInstanceState()`.

Veamos un ejemplo de lo sencillo que resulta guardar la información de una variable tipo cadena de caracteres y entero.

```
String cadena;
int pos;

@Override
protected void onSaveInstanceState(Bundle guardarEstado) {
    super.onSaveInstanceState(guardarEstado);
    guardarEstado.putString("variable", var);
    guardarEstado.putInt("posicion", pos);
}

@Override
protected void onRestoreInstanceState(Bundle recEstado) {
    super.onRestoreInstanceState(recEstado);
    var = recEstado.getString("variable");
    pos = recEstado.getInt("posicion");
}
```

La ventaja de usar estos métodos es que el programador no debe buscar un método de almacenamiento permanente, es el sistema quien hará este trabajo. El inconveniente es que el sistema no nos garantiza que sean llamados. En casos de extrema necesidad de memoria se podría destruir nuestro proceso sin llamarlos.

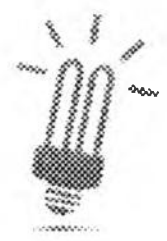


**Práctica**: *Guardando el estado en Asteroides.*

1. Ejecuta el proyecto Asteroides.
2. Cambia de orientación el teléfono. Observarás como la música se reinicia cada vez que lo haces.
3. Utilizando los métodos para guardar el estado de una actividad, trata de que cuando se voltea el teléfono, el audio continúe en el mismo punto de reproducción. Puedes utilizar los siguientes métodos:

```
int pos = mp.getCurrentPosition();
mp.seekTo(pos);
```

4. Verifica el resultado.



**Solución**: Una posible solución al ejercicio anterior se muestra a continuación:

```
@Override
protected void onSaveInstanceState(Bundle estadoGuardado){
    super.onSaveInstanceState(estadoGuardado);
    if (mp != null) {
        int pos = mp.getCurrentPosition();
        estadoGuardado.putInt("posicion", pos);
    }
}

@Override
protected void onRestoreInstanceState(Bundle estadoGuardado){
    super.onRestoreInstanceState(estadoGuardado);
    if (estadoGuardado != null && mp != null) {
        int pos = estadoGuardado.getInt("posicion");
        mp.seekTo(pos);
    }
}
```

Para reforzar el uso de estos métodos te recomendamos el ejercicio planteado en el apartado 6.4.1 donde se pide recordar el punto de reproducción de un vídeo.

## 6.2. Utilizando multimedia en Android

La integración de contenido multimedia en nuestras aplicaciones resulta muy sencilla gracias a la gran variedad de facilidades que nos proporciona el API.

Concretamente podemos reproducir audio y vídeo desde orígenes distintos:

- Desde un fichero almacenado en el dispositivo.

- Desde un recurso que está incrustado en el paquete de la aplicación (fichero `.apk`).
- Desde un *stream* que es leído desde una conexión de red. En este punto admite dos posibles protocolos (`http://` y `rstp://`)

También resulta sencilla la grabación de audio y vídeo, siempre que el *hardware* del dispositivo lo permita.

En la siguiente lista se muestran las clases de Android que nos permitirán acceder a los servicios Multimedia:

**MediaPayer**: Reproducción de audio/vídeo desde ficheros o *streams*.

**MediaControler**: Visualiza controles estándar para mediaPlayer (pausa, stop).

**VideoView**: Vista que permite la reproducción de vídeo.

**SoundPool**: Maneja y reproduce una colección de recursos de audio.

**AsyncPlayer**: Reproduce listas de audios desde un *thread* distinto al nuestro.

**MediaRecorder**: Permite grabar audio y vídeo.

**AudioManager**: Gestiona varias propiedades del sistema (volumen, tonos, etc.).

**AudioTrack**: Reproduce un búfer de audio PCM directamente por *hardware*.

**JetPlayer**: Reproduce audio y video interactivo creado con JetCreator.

**Camera:** Cómo utilizar la cámara para tomar fotos y video.

**FaceDetector**: Identifica la cara de la gente en un bitmap.

La plataforma Android soporta una gran variedad de formatos, muchos de los cuales pueden ser tanto decodificados como codificados. A continuación, mostramos una tabla con los formatos multimedia soportados. No obstante, algunos modelos de móviles pueden soportar formatos adicionales que no se incluyen en la tabla, como por ejemplo DivX.

Cada desarrollador es libre de usar los formatos incluidos en el núcleo del sistema o aquellos que solo se incluyen en algunos dispositivos.

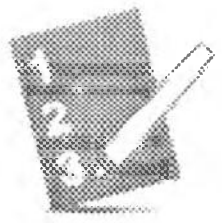
| Tipo | Formato | Codifica | Decodifica | Detalles | Fichero soportado |
|---|---|---|---|---|---|
| Audio | AAC LC/LTP | X | X | Mono/estéreo con cualquier combinación estándar de frecuencias por encima de 160 Kbps y ratios de muestreo de 8 a 48kHz | 3GPP (`.3gp`)<br>MPEG-4 (`.mp4`)<br>No soporta raw AAC (`.aac`) |
| | HE-AACv1 (AAC+) | | X | | |
| | HE-AACv2 (enhanced AAC+) | | X | | |
| | AMR-NB | X | X | 4.75 a 12.2 Kbps muestreada a @ 8kHz | 3GPP (`.3gp`) |

| Tipo | Formato | Codifica | Decodifica | Detalles | Fichero soportado |
|---|---|---|---|---|---|
| | AMR-WB | X | X | 9 ratios de 6.60 Kbps a 23.85 Kbps muestreada a @ 16kHz | 3GPP (.3gp) |
| | MP3 | | X | Mono/estéreo de 8 a 320 Kbps, frecuencia de muestreo constante (CBR) o variable (VBR) | MP3 (.mp3) |
| | MIDI | | X | 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 | | X | | Ogg (.ogg) Matroska (.mkv a partir 4.0) |
| | FLAC | | a partir 3.1 | mono/estereo (no multicanal) | FLAC (.flac) |
| | PCM/WAVE | | X | 8 y 16 bits PCM lineal (frecuencias limitadas por el hardware) | WAVE (.wav) |
| Imagen | JPEG | X | X | Base + progresivo | JPEG (.jpg) |
| | GIF | | X | | GIF (.gif) |
| | PNG | X | X | | PNG (.png) |
| | BMP | | X | | BMP (.bmp) |
| | WEBP | a partir 4.0 | a partir 4.0 | | WebP (.webp) |
| Video | H.263 | X | X | | 3GPP (.3gp) MPEG-4 (.mp4) |
| | H.264 AVC | a partir 3.0 | X | Baseline Profile (BP) | 3GPP (.3gp) MPEG-4 (.mp4) |
| | MPEG-4 SP | | X | | 3GPP (.3gp) |
| | WP8 | | a partir 2.3.3 | Streaming a partir 4.0 | WebM (.webm) Matroska (.mkv a partir 4.0) |

***Tabla 1:*** *Formatos multimedia soportados en Android.*

## 6.3. La vista VideoView

La forma más fácil de incluir un vídeo en tu aplicación es incluir una vista de tipo `VideoView`. Veamos cómo hacerlo en el siguiente ejercicio.



**Ejercicio paso a paso:** *Reproducir un vídeo con VideoView.*

1. Crea una nueva aplicación con los siguientes datos:

```
Project name: videoView
Build Target: Android 1.5
Application name: videoView
Package name: org.example.videoview
Create Activity: videoView
```

2. Remplaza el fichero `ref/layout/main.xml` por:

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <VideoView
        android:id="@+id/surface_view"
        android:layout_width="320px"
        android:layout_height="240px"
    />
</LinearLayout>
```

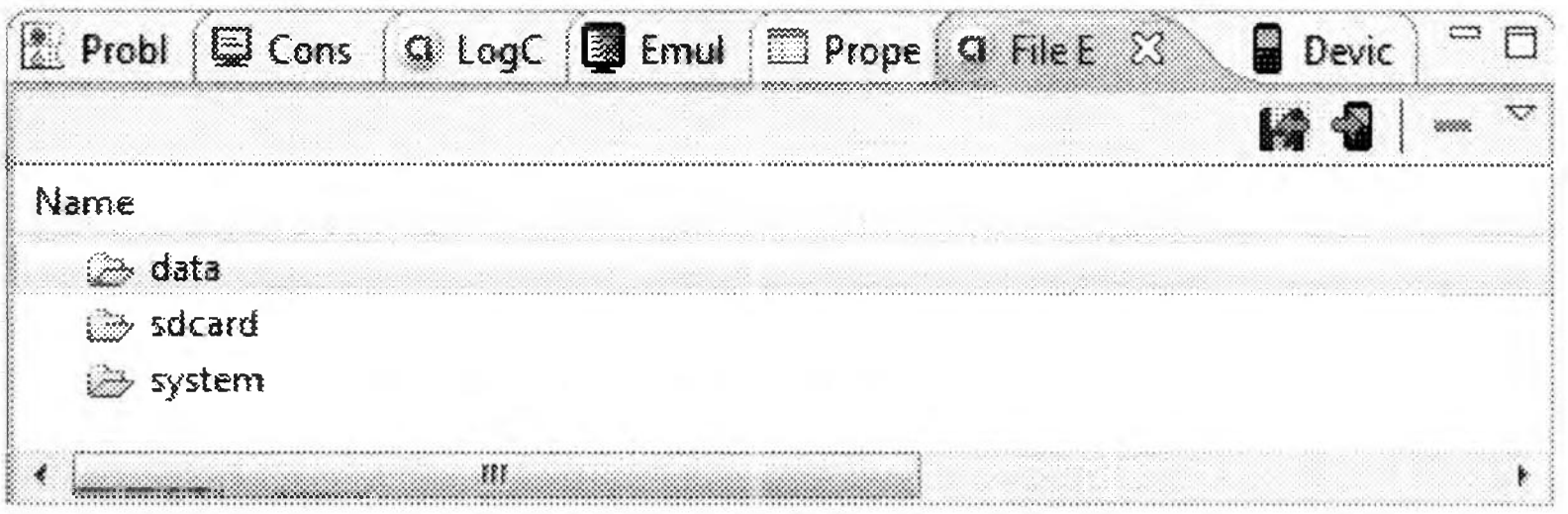
3. Remplaza el siguiente código en la clase `videoView`:

```
public class videoView extends Activity {
  private VideoView mVideoView;

  @Override
  public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);
      mVideoView =(VideoView)findViewById(R.id.surface_view);
      //de forma alternative si queremos un streaming usar
      //mVideoView.setVideoURI(Uri.parse(URLstring));
      mVideoView.setVideoPath("/data/video.mp4");
      mVideoView.start();
      mVideoView.requestFocus();
  }
}
```

4. En el método `setVideoPath()` estamos indicando un fichero local.

5. Busca un fichero de vídeo en codificación mp4. Renombra este fichero a *video.mp4*.

6. Es necesario almacenar un fichero de vídeo en el simulador. Para ello, utiliza la opción del menú *Window* > *Show View* >*Others...* > *Android* > *File Explorer*. Te mostrará el sistema de ficheros del emulador.




7. Selecciona la carpeta *data* y utiliza el botón donde aparece un teléfono con una flecha para almacenar un nuevo fichero. Indica el fichero *video.mp4*.

8. Ejecuta la aplicación y observa el resultado. Recuerda que con *Ctrl-F11* puedes cambiar la orientación. Si no funciona, trata de usar otro tipo de emulador.


9. Modifica el fichero xml para que el vídeo aparezca centrado y ocupe toda la pantalla del teléfono, tal y como se muestra en la imagen de arriba.

10. Añade la siguiente línea antes de la llamada al método `start()`:

```
mVideoView.setMediaController(new MediaController(this));
```

De esta forma permitimos que el usuario pueda controlar la reproducción del vídeo mediante el objeto `MediaController`.

11. Observa el resultado.

## 6.4. La clase MediaPlayer

La reproducción multimedia en Android se lleva a cabo principalmente por medio de la clase `MediaPlayer`. Veremos, a continuación, las características más importantes de esta clase y cómo podemos sacarle provecho.

Un objeto `MediaPlayer` va a poder pasar por gran variedad de estados: inicializados sus recursos (*initialized*), preparando la reproducción (*preparing*), preparado para reproducir (*prepared*), reproduciendo (*started*), en pausa (*paused*), parado (*stoped*), reproducción completada (*playback completed*), finalizado (*end*) y con error (*error*). Resulta importante conocer en qué estado se encuentra dado que muchos de los métodos solo pueden ser llamados desde ciertos estados. Por ejemplo, no podremos ponerlo en reproducción (método `start()`) sino está en estado preparado. O no podremos ponerlo en pausa (`pause()`), si está parado. Si llamamos a un método no admitido para un determinado estado se producirá un error de ejecución.

La siguiente tabla permite conocer los métodos que podemos invocar desde cada uno de los estados y cuál es el nuevo estado al que iremos tras invocarlo. Existen dos tipos de métodos, los que no están subrayados representan métodos llamados de forma síncrona desde nuestro código, mientras que los que están subrayados representan métodos llamados de forma asíncrona por el sistema.

| **Estado salida** / **Estado entrada** | **Idle** | **Initialized** | **Preparing** | **Prepared** | **Started** | **Paused** | **Stoped** | **Playback Completed** |
|---|---|---|---|---|---|---|---|---|
| **Initialized** | setDataSource | | | | | | | |
| **Preparing** | | prepareAsinc | | | | | prepareAsinc | |
| **Prepared** | | prepare | onPrepared | seekTo | | | prepare | |
| **Started** | | | | start | seekTo<br>start | start | | start |
| **Paused** | | | | | pause | seekTo<br>pause | | |
| **Stoped** | | | | stop | stop | stop | stop | stop |
| **Playback Completed** | | | | | onCompletion | | | seekTo |
| **End** | Release | release | release | release | release | release | release | release |
| **Error** | onError | onError | onError | onError | onError | onError | onError | onError |

***Tabla 2:*** *Transiciones entre estados de la clase MediaPlayer.*

### 6.4.1. Reproducción de audio con MediaPlayer

Si el audio o vídeo se va a reproducir siempre en nuestra aplicación resulta interesante incluirlo en el paquete `.apk` y tratarlo como un recurso. Este uso ya ha sido ilustrado al comienzo del capítulo. Recordemos como se hacía:

1. Crea una nueva carpeta con nombre raw dentro de la carpeta res.

2. Arrastra a su interior el fichero de audio. Por ejemplo audio.mp3.
3. Añade las siguientes líneas de código:

```
MediaPlayer mp = MediaPlayer.create(this, R.raw.audio);
mp.start();
```

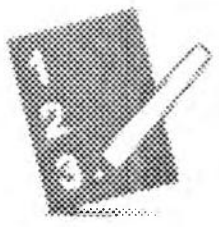
Si deseas parar la reproducción tendrás que utilizar el método `stop()`. Si, a continuación, quieres volver a reproducirlo utiliza el método `prepare()` y, luego, `start()`. También puedes usar `pause()` y `start()` para ponerlo en pausa y reanudarlo.

Si en lugar de un recurso prefieres reproducirlo desde un fichero utiliza el siguiente código. Observa cómo en este caso es necesario llamar al método `prepare()`. En el caso anterior no ha sido necesario dado que esta llamada se hace desde `create()`.

```
MediaPlayer mp = new MediaPlayer();
mp.setDataSource(CAMINO_AL_FICHERO);
mp.prepare();
mp.start();
```

## 6.5. Un reproductor multimedia paso a paso

En el siguiente ejercicio vamos a profundizar en el objeto *MediaPlayer* por medio de un ejemplo, donde trataremos de realizar un reproductor de vídeos personalizado.



**Ejercicio paso a paso:** *Un reproductor multimedia paso a paso.*

1. Crea una nueva aplicación con los siguientes datos:

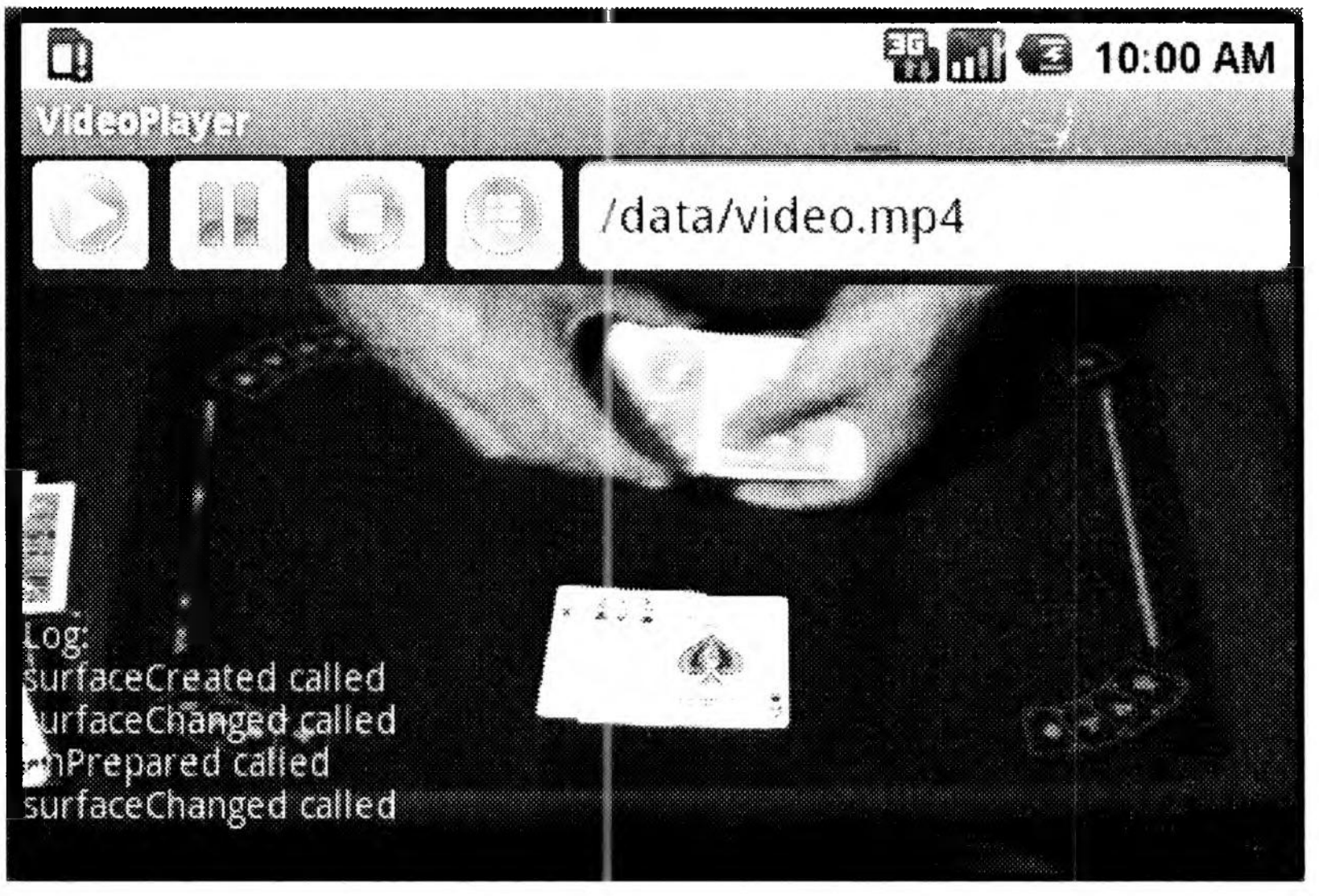
```
Project name: VideoPlayer
Build Target: Google APIs 1.5
Application name: VideoPlayer
Package name: org.example.videoplayer
Create Activity: VideoPlayer
Min SDK Version: 3
```

2. En la carpeta `res/drawable` arrastra los cuatro ficheros de iconos: `play.jpeg`, `pause.jpeg`, `reset.jpeg` *y* `stop.jpeg`. Los puedes encontrar en www.androidcurso.com.
3. Remplaza el fichero *res/layout/main.xml* por:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_height="fill_parent"
  android:layout_width="fill_parent"
  android:orientation="horizontal">
```

```
<LinearLayout
        android:id="@+id/ButonsLayout"
        android:layout_height="wrap_content"
        android:layout_width="fill_parent"
        android:orientation="horizontal"
        android:layout_alignParentTop="true">
    <ImageButton android:id="@+id/play"
                 android:layout_height="wrap_content"
                 android:layout_width="wrap_content"
                 android:src="@drawable/play"/>
    <ImageButton android:id="@+id/pause"
                 android:layout_height="wrap_content"
                 android:layout_width="wrap_content"
                 android:src="@drawable/pause"/>
    <ImageButton android:id="@+id/stop"
                 android:layout_height="wrap_content"
                 android:layout_width="wrap_content"
                 android:src="@drawable/stop"/>
    <ImageButton android:id="@+id/logButton"
                 android:layout_height="wrap_content"
                 android:layout_width="wrap_content"
                 android:src="@drawable/log"/>
    <EditText    android:id="@+id/path"
                 android:layout_height="fill_parent"
                 android:layout_width="fill_parent"
                 android:text="/data/video.3gp"/>
</LinearLayout>
<VideoView android:id="@+id/surfaceView"
           android:layout_height="fill_parent"
           android:layout_width="fill_parent"
           android:layout_below="@+id/ButonsLayout"/>
<ScrollView android:id="@+id/ScrollView"
           android:layout_height="100px"
           android:layout_width="fill_parent"
           android:layout_alignParentBottom="true">
    <TextView    android:id="@+id/Log"
                 android:layout_height="wrap_content"
                 android:layout_width="fill_parent"
                 android:text="Log:"/>
</ScrollView>
</RelativeLayout>
```

La apariencia del *Layout* anterior se muestra a continuación:




4. Remplaza el código de la clase `VideoPlayer` por:

```
public class VideoPlayer extends Activity implements
        OnBufferingUpdateListener, OnCompletionListener,
        MediaPlayer.OnPreparedListener, SurfaceHolder.Callback {
  private MediaPlayer mediaPlayer;
  private SurfaceView surfaceView;
  private SurfaceHolder surfaceHolder;
  private EditText editText;
  private ImageButton bPlay, bPause, bStop, bLog;
  private TextView logTextView;
  private boolean pause;
  private String path;
  private int savePos = 0;

  public void onCreate(Bundle bundle) {
     super.onCreate(bundle);
     setContentView(R.layout.main);
     surfaceView = (SurfaceView) findViewById(R.id.surfaceView);
     surfaceHolder = surfaceView.getHolder();
     surfaceHolder.addCallback(this);
     surfaceHolder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
     editText = (EditText) findViewById(R.id.path);
     editText.setText(
           "http://personales.gan.upv.es/~jtomas/video.3gp");
     logTextView = (TextView) findViewById(R.id.Log);
     bPlay = (ImageButton) findViewById(R.id.play);
     bPlay.setOnClickListener(new OnClickListener() {
        public void onClick(View view) {
```

```
            if (mediaPlayer != null) {
                if (pause) {
                    mediaPlayer.start();
                } else {
                    playVideo();
                }
            }
        }
    });
    bPause = (ImageButton) findViewById(R.id.pause);
    bPause.setOnClickListener(new OnClickListener() {
        public void onClick(View view) {
            if (mediaPlayer != null) {
                pause = true;
                mediaPlayer.pause();
            }
        }
    });
    bStop = (ImageButton) findViewById(R.id.stop);
    bStop.setOnClickListener(new OnClickListener() {
        public void onClick(View view) {
            if (mediaPlayer != null) {
                pause = false;
                mediaPlayer.stop();
            }
        }
    });
    bLog = (ImageButton) findViewById(R.id.logButton);
    bLog.setOnClickListener(new OnClickListener() {
        public void onClick(View view) {
            if (logTextView.getVisibility()==TextView.VISIBLE) {
                logTextView.setVisibility(TextView.INVISIBLE);
            } else {
                logTextView.setVisibility(TextView.VISIBLE);
            }
        }
    });
        log("");
    }
```

5. Como puedes ver, la aplicación extiende la clase `Activity`. Además, implementamos cuatro interfaces que corresponden a varios escuchadores de eventos. Luego continúa la declaración de variables. Las primeras corresponden a diferentes elementos de la aplicación y su significado resulta obvio. La variable `pause` nos indica si el usuario ha pulsado el botón correspondiente, la variable `path` nos indica dónde está el vídeo en reproducción y la variable `savePos` almacena la posición de reproducción.

6. Añade:

```
private void playVideo() {
  try {
      pause = false;
      path = editText.getText().toString();
      mediaPlayer = new MediaPlayer();
      mediaPlayer.setDataSource(path);
      mediaPlayer.setDisplay(surfaceHolder);
      mediaPlayer.prepare();
      // mMediaPlayer.prepareAsync(); Para streaming
      mediaPlayer.setOnBufferingUpdateListener(this);
      mediaPlayer.setOnCompletionListener(this);
      mediaPlayer.setOnPreparedListener(this);
      mediaPlayer.setAudioStreamType(AudioManager.STREAM_MUSIC);
      mediaPlayer.seekTo(savePos);
  } catch (Exception e) {
      log("ERROR: " + e.getMessage());
  }
}
```

7. El código continúa con la definición del método `playVideo()`. Este método se encarga de obtener la ruta de reproducción, crear un nuevo objeto `MediaPlayer`, luego se le asigna la ruta y la superficie de visualización, a continuación, se prepara la reproducción del vídeo. En caso de querer reproducir un *stream* desde la red, esta función puede tardar bastante tiempo, en tal caso es recomendable utilizar en su lugar el método `prepareAsync()` que permite continuar con la ejecución del programa, aunque sin esperar a que el vídeo esté preparado. Las siguientes tres líneas asignan a nuestro objeto varios escuchadores de eventos que serán descritos más adelante. Tras preparar el tipo de audio, se sitúa la posición de reproducción a los milisegundos indicados en la variable `savePos`. Si se trata de una nueva reproducción, esta variable será cero.

8. Añade el código:

```
public void onBufferingUpdate(MediaPlayer arg0, int percent)
{
  log("onBufferingUpdate percent:" + percent);
}

public void onCompletion(MediaPlayer arg0) {
  log("onCompletion called");
}
```

9. Los métodos anteriores implementan los interfaces `OnBufferingUpdate Listener` y `OnCompletionListener`. El primero irá mostrando el porcentaje de obtención de búfer de reproducción, mientras que el segundo será invocado cuando el vídeo en reproducción llegue al final.

10. Añade el código:

```
public void onPrepared(MediaPlayer mediaplayer) {
  log("onPrepared called");
  int mVideoWidth = mediaPlayer.getVideoWidth();
  int mVideoHeight = mediaPlayer.getVideoHeight();
  if (mVideoWidth != 0 && mVideoHeight != 0) {
     surfaceHolder.setFixedSize(mVideoWidth, mVideoHeight);
     mediaPlayer.start();
  }
}
```

11. El método anterior implementa la interfaz `OnPreperedListener`. Es invocado una vez que el vídeo ya está preparado para su reproducción. En este momento podemos conocer el alto y el ancho del vídeo y ponerlo en reproducción.

12. Añade el código:

```
public void surfaceCreated(SurfaceHolder holder) {
  log("surfaceCreated called");
  playVideo();
}

public void surfaceChanged(SurfaceHolder surfaceholder,
                           int i, int j, int k) {
  log("surfaceChanged called");
}

public void surfaceDestroyed(SurfaceHolder surfaceholder) {
  log("surfaceDestroyed called");
}
```

13. Los métodos anteriores implementan la interfaz `SurfaceHolder.Callback`. Se invocarán cuando nuestra superficie de visualización se cree, cambie o se destruya. Los métodos que siguen corresponden a acciones del ciclo de vida de una actividad:

14. Añade el código:

```
@Override protected void onDestroy() {
  super.onDestroy();
  if (mediaPlayer != null) {
     mediaPlayer.release();
     mediaPlayer = null;
  }
}
```

15. Este método se invoca cuando la actividad va a ser destruida. Dado que un objeto de la clase `MediaPlayer` consume muchos recursos, resulta interesante liberarlos lo antes posible.

16. Añade el código:

```
@Override public void onPause() {
  super.onPause();
  if (mediaPlayer != null & !pause) {
     mediaPlayer.pause();
  }
}

@Override public void onResume() {
  super.onResume();
  if (mediaPlayer != null & !pause) {
     mediaPlayer.start();
  }
}
```

17. Los dos métodos anteriores se invocan cuando la actividad pasa a un segundo plano y cuando vuelve a primer plano. Dado que queremos que el vídeo deje de reproducirse y continúe reproduciéndose en cada uno de estos casos, se invocan a los métodos `pause()` y `start()`, respectivamente. No hay que confundir esta acción con la variable `pause` que lo que indica es que el usuario ha pulsado el botón correspondiente.

18. Añade el código:

```
@Override
protected void onSaveInstanceState(Bundle guardarEstado) {
  super.onSaveInstanceState(guardarEstado);
  if (mediaPlayer != null) {
     int pos = mediaPlayer.getCurrentPosition();
     guardarEstado.putString("ruta", path);
     guardarEstado.putInt("posicion", pos);
  }
}

@Override
protected void onRestoreInstanceState(Bundle recEstado) {
  super.onRestoreInstanceState(recEstado);
  if (recEstado != null) {
     path = recEstado.getString("ruta");
     savePos = recEstado.getInt("posicion");
  }
}
```

19. Cuando este sistema necesita memoria, puede decidir eliminar nuestra actividad. Antes de hacerlo llamará al método `onSaveInstanceState` para darnos la oportunidad de guardar información sensible. Si, más adelante, el usuario vuelve a la aplicación, esta se volverá a cargar, invocándose el método `onRestoreInstanceState`, donde podremos restaurar el estado perdido. En nuestro caso la información a guardar son las variables `path` y

savePos, que representan el vídeo y la posición que estamos reproduciendo.

20. Ocurre el mismo proceso cuando el usuario cambia la posición del teléfono. Es decir, cuando el teléfono se voltea las actividades son destruidas y vueltas a crear, por lo que también se invocan estos métodos.

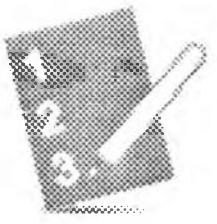
21. Añade el código:

```
private void log(String s) {
  logTextView.append(s + "\n");
}
```

22. El último método es utilizado por varios escuchadores de eventos para mostrar información sobre lo que está pasando. Esta información puede visualizarse o no, utilizando el botón correspondiente.

## 6.6. Introduciendo efectos de audio con SoundPool

Hemos aprendido a utilizar la clase `MediaPlayer` para reproducir audio y vídeo. En un primer ejercicio vamos a aprender como introducir efectos de audio en Asteroides. Como veremos esta clase no resulta adecuada para este uso. A continuación, se introducirá la clase `SoundPool` y se mostrará, mediante un ejercicio, cómo esta sí que resulta eficiente para nuestro juego.



**Ejercicio paso a paso:** *Introduciendo efectos de audio con MediaPlayer en Asteroides.*

1. Abre el proyecto Asteroides.
2. Copia los siguientes ficheros a la carpeta *res/raw*. disparo.mp3 y explosion.mp3.
3. Abre la clase `VistaJuego` y añade las siguientes variables:

```
MediaPlayer mpDisparo, mpExplosion;
```

4. En el constructor de la clase inicialízalas de la siguiente forma:

```
mpDisparo = MediaPlayer.create(context, R.raw.disparo);
mpExplosion = MediaPlayer.create(context, R.raw.explosion);
```

5. Añade en el método `ActivaMisil()` de `VistaJuego`:

```
mpDisparo.start();
```

6. Añade en el método `destruyeAsteroide()` de `VistaJuego`:

```
mpExplosion.start();
```

7. Ejecuta el programa y verifica el resultado. ¿Qué pasa cuando disparamos o destruimos un asteroide? ¿El sonido se oye de forma inmediata?

La clase `SoundPool` maneja y reproduce de forma rápida recursos de audio en las aplicaciones.

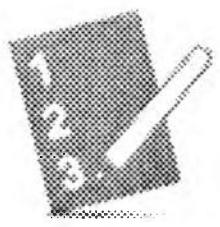
Un `SoundPool` es una colección de pistas de audio que se pueden cargar en la memoria desde un recurso (dentro de la APK) o desde el sistema de archivos. `SoundPool` utiliza el servicio de la clase `MediaPlayer` para decodificar el audio en un formato crudo (PCM de16 bits), lo que después permite reproducirlo rápidamente por el hardware sin tener que decodificarlas cada vez.

Los sonidos pueden repetirse en un bucle una cantidad establecida de veces, definiendo el valor al reproducirlo, o dejarse reproduciendo en un bucle infinito con -1. En este caso, será necesario detenerlo con el método `stop()`.

La velocidad de reproducción también se puede cambiar. El rango de velocidad de reproducción va de 0.5 a 2.0. Una tasa de reproducción de 1.0 hace que el sonido se reproduzca en su frecuencia original. Una tasa de reproducción de 2.0 hace que el sonido se reproduzca al doble de su frecuencia original, y una tasa de reproducción de 0.5 hace que se reproduzca a la mitad de su frecuencia original.

Cuando se crea un `SoundPool` hay que establecer en un parámetro el máximo de pistas que se pueden reproducir simultáneamente. Este parámetro no tiene por qué coincidir con el número de pistas cargadas. Cuando se pone una pista en reproducción (método `play()`) hay que indicar una prioridad. Esta prioridad se utiliza para decidir qué se hará cuando el número de reproducciones activas exceda el valor máximo establecido en el constructor. En este caso, se detendrá el flujo con la prioridad más baja, y en caso de que haya varios, se detendrá el más antiguo. En caso de que el nuevo flujo sea el de menor prioridad, este no se reproducirá.

Una lista de todos los métodos de esta clase la puedes encontrar en: http://developer.android.com/reference/android/media/SoundPool.html



**Ejercicio paso a paso:** *Introduciendo efectos de audio con SoundPool en Asteroides.*

1. El proyecto Asteroides elimina todo el código introducido a partir del punto 3, del ejercicio anterior.
2. Abre la clase `VistaJuego` y añade las siguientes variables:

```
// //// MULTIMEDIA //////
SoundPool soundPool;
int idDisparo, idExplosion;
```

3. En el constructor de la clase inicializalas de la siguiente forma:

```
soundPool = new SoundPool( 5, AudioManager.STREAM_MUSIC , 0);
idDisparo = soundPool.load(context, R.raw.disparo, 0);
idExplosion = soundPool.load(context, R.raw.explosion, 0);
```

4. En el constructor de *SoundPool* hay que indicar tres parámetros. El primero corresponde al máximo de reproducciones simultáneas. El segundo es el tipo

de stream de audio (normalmente `STREAM_MUSIC`). El tercero es la calidad de reproducción, aunque actualmente no se implementa.

5. Cada una de las pistas ha de ser cargada mediante el método `load()`. Existen muchas sobrecargas para este método. La versión utilizada en el ejemplo permite cargar las pistas desde los recursos. El último parámetro corresponde a la prioridad, aunque actualmente no tiene ninguna utilidad.

6. Añade en el método `ActivaMisil()` de `VistaJuego`:

```
soundPool.play(idDisparo, 1, 1, 1, 0, 1);
```

7. El método `play()` permite reproducir una pista. Hay que indicarle el identificador de pista; el volumen para el canal izquierdo y derecho (0.0 a 1.0); La prioridad; El número de repeticiones (-1=siempre, 0=solo una vez, 1=repetir una vez, ...) y el ratio de reproducción, con el que podremos modificar la velocidad o pitch (1.0 reproducción normal, rango: 0.5 a 2.0).

8. Añade en el método `destruyeAsteroide()` de `VistaJuego`:

```
soundPool.play(idExplosion, 1, 1, 0, 0, 1);
```

9. Los parámetros utilizados para la explosión son similares, solo hemos introducido una prioridad menor. La consecuencia será que si ya hay un total de 5 pistas (ver constructor) reproduciéndose y se pide la reproducción de un nuevo disparo, el sistema detendrá la reproducción de la explosión más antigua, por tener esta menos prioridad.

10. Ejecuta el programa y verifica el resultado. ¿Qué pasa cuando disparamos o destruimos un asteroide? ¿El sonido se oye de forma inmediata?

11. Modifica algunos de los parámetros del método `play()` y verifica el resultado.

## 6.7. Grabación de audio

Las APIs de Android disponen de facilidades para capturar audio y video, permitiendo su codificación en diferentes formatos. La clase `MediaRecorder` te permitirá de forma sencilla integrar esta funcionalidad en tu aplicación.

La mayoría de los dispositivos disponen de micrófono para capturar audio, sin embargo esta facilidad no ha sido integrada en el emulador. Por lo tanto, deberás probar los ejercicios de este apartado en un dispositivo real.

La clase `MediaRecorder` dispone de una serie de métodos que podrás utilizar para configurar la grabación:

`setAudioSource(int audio_source)` – Dispositivo que se utilizará como fuente del sonido. Normalmente `MediaRecorder.AudioSource.MIC`. También se pueden utilizar otras constantes como `DEFULT`, `CAMCORDER`, `VOICE_CALL`, `VOICE_COMUNICATION`,...

`setOutputFile (String fichero)` – Nombre del fichero de salida.

`setOutputFormat(int output_forma)` – Formato del fichero de salida. Se pueden utilizar constantes de la clase `MediaRecorder.OutputFormat`: `DEFAULT`, `AMR_NB`, `AMR_WB`, `RAW_AMR` (ARM), `MPEG_4` (MP4) y `THREE_GPP` (3GPP).

`setAudioEncoder(int audio_encoder)` – Codificación del audio. Cuatro posibles constantes de la clase `MediaRecorder.AudioEncoder`: `AAC`, `AMR_NB`, `AMR_WB` y `DEFAULT`.

`setAudioChannels(int numeroCanales)` – Especificamos el número de canales 1: mono y 2: estéreo.

`setAudioEncodingBitRate (int bitRate)` (desde nivel de API 8) – Especificamos los bits por segundo (bps) a utilizar en la codificación.

`setAudioSamplingRate (int samplingRate)` (desde nivel de API 8) – Especificamos el número de muestras por segundo a utilizar en la codificación.

`setMaxDuration (int max_duration_ms)` (desde nivel de API 3) – Indicamos una duración máxima para la grabación. Tras este tiempo se detendrá.

`setMaxFileSize (long max_filesize_bytes)` (desde nivel de API 3) – Indicamos un tamaño máximo para el fichero. Al alcanzar el tamaño se detendrá.

`prepare()` – Prepara la grabación para la captura de datos. Antes de llamarlo hay que configurar la grabación y después ya podemos invocar al método `start()`.

`start()` – Comienza la captura.

`stop()` – Finaliza la captura.

`reset()` – Reinicia el objeto como si lo acabáramos de crear. Hay que volver a configurarlo.

`release()` – Libera todos los recursos utilizados de forma inmediata. Si no llamas al método se liberarán cuando el objeto sea destruido.

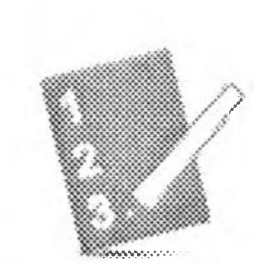
La clase `MediaRecorder` también dispone de métodos que podrás utilizar para configurar la grabación de video. Más información en: http://developer.android.com/reference/android/media/MediaRecorder.html.

La siguiente tabla muestra los diferentes métodos que pueden ser ejecutados en cada estado y el estado que alcanzaremos tras la llamada:

| Estado salida | Estado entrada: Initial | Initialized | DataSource Configured | Prepared | Recording | Error |
|---|---|---|---|---|---|---|
| **Initial** | | Reset | reset | reset | reset<br>stop | reset |

| Estado salida \ Estado entrada | Initial | Initialized | DataSource Configured | Prepared | Recording | Error |
|---|---|---|---|---|---|---|
| **Initialized** | setAudioSource<br>setVideoSource | setAudioSource<br>setVideoSource | | | | |
| **DataSource Configured** | | setOutputFormat | setAudioEncoder<br>setVideoEncoder<br>setOutputFile<br>setVideoSize<br>setVideoFrameRate<br>setPreviewDisplay | | | |
| **Prepared** | | | prepare | | | |
| **Recording** | | | | start | | |
| **Released** | releas | | | | | |
| **Error** | llamada incorrecta | llamada incorrecta | llamada incorrecta | llamada incorrecta | llamada incorrecta | |

***Tabla 3:*** *Transiciones entre estados de la clase MediaRecorder*



**Ejercicio paso a paso:** *Grabación de audio utilizando MediaRecorder.*

1. Crea un nuevo proyecto con nombre *Grabadora*. El nombre del paquete ha de ser `org.example.grabadora`.
2. Remplaza el Layout *main.xml* por el siguiente código:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="horizontal">
    <Button
        android:id="@+id/bGrabar"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Grabar"
        android:onClick="grabar"/>
<Button
        android:id="@+id/bDetener"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Detener Grabacion"
        android:onClick="detenerGrabacion"/>
<Button
        android:id="@+id/bReproducir"
```

```
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Reproducir"
            android:onClick="reproducir"/>
    </LinearLayout>
```

3. Remplaza el código de la actividad por:

```
public class GrabadoraActivity extends Activity {
  private static final String LOG_TAG = "Grabadora";
  private MediaRecorder mediaRecorder;
  private MediaPlayer mediaPlayer;
  private static String fichero = Environment.
  getExternalStorageDirectory().getAbsolutePath()+"/audio.3gp";

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

  public void grabar(View view) {
     mediaRecorder = new MediaRecorder();
     mediaRecorder.setOutputFile(fichero);
     mediaRecorder.setAudioSource(MediaRecorder.AudioSource.MIC);
     mediaRecorder.setOutputFormat(MediaRecorder.
                                   OutputFormat.THREE_GPP);
     mediaRecorder.setAudioEncoder(MediaRecorder.AudioEncoder.
                                   AMR_NB);
     try {
        mediaRecorder.prepare();
     } catch (IOException e) {
        Log.e(LOG_TAG, "Fallo en grabación");
     }
     mediaRecorder.start();
  }

  public void detenerGrabacion(View view) {
     mediaRecorder.stop();
     mediaRecorder.release();
  }

  public void reproducir(View view) {
     mediaPlayer = new MediaPlayer();
     try {
        mediaPlayer.setDataSource(fichero);
        mediaPlayer.prepare();
        mediaPlayer.start();
     } catch (IOException e) {
```

```
            Log.e(LOG_TAG, "Fallo en reproducción");
        }
    }
}
```

4. Comenzamos declarando dos objetos de las clases `MediaRecorder` y `MediaPlayer` que serán usados para la grabación y reproducción respectivamente. También se declaran dos `String`. La contante `LOG_TAG` será utilizada como etiqueta para identificar nuestra aplicación en el fichero de Log. La variable `fichero` identifica el nombre del fichero donde se guardará la grabación. Este fichero se almacenará en una carpeta especialmente creada para nuestra aplicación. En el método `onCreate()` no se realizará ninguna acción especial.

5. A continuación, se han introducido tres métodos que serán ejecutados al pulsar los botones de nuestro Layout. El significado de cada uno de los métodos que se invocan acaba de ser explicado.

6. Abre AndroidManifest.xml y en la pestaña *Permissions* pulsa el botón *Add...* selecciona *Uses Permissions*. En el desplegable *Name* indica `RECORD_AUDIO`. Repite este proceso para usar el permiso `WRITE_EXTERNAL_STORAGE`.

7. Ejecuta de nuevo la aplicación y verifica el resultado.

**Ejercicio paso a paso:** *Grabación de audio utilizando MediaRecorder (II)*

La aplicación anterior resulta algo confusa de utilizar. Sería más sencillo si los botones que no pudiéramos utilizar en un determinado momento estuvieran deshabilitados. Para conseguirlo vamos a utilizar el método `Button.setEnabled(boolean)`. Veamos cómo conseguirlo:

1. Declara las siguientes tres variables al principio de la Actividad:

   ```
   private Button bGrabar, bDetener, bReproducir;
   ```

2. En el método `onCreate()` inicializa estos tres botones. Además, comenzaremos deshabilitando dos de los botones que, al principio de la aplicación, no deben ser pulsados:

   ```
   bGrabar = (Button) findViewById(R.id.bGrabar);
   bDetener = (Button) findViewById(R.id.bDetener);
   bReproducir = (Button) findViewById(R.id.bReproducir);
   bDetener.setEnabled(false);
   bReproducir.setEnabled(false);
   ```

3. En el método `grabar()` añade el siguiente código:

   ```
   bGrabar.setEnabled(false);
   bDetener.setEnabled(true);
   ```

```
bReproducir.setEnabled(false);
```

4. En el método `detenerGrabacion()` añade el siguiente código:

```
bGrabar.setEnabled(true);
bDetener.setEnabled(false);
bReproducir.setEnabled(true);
```

5. Ejecuta de nuevo la aplicación y verifica el resultado.

# CAPÍTULO 7.
# Seguridad y posicionamiento

En este capítulo abordaremos dos de los aspectos más novedosos de Android: la seguridad y el API de posicionamiento.

El capítulo comienza estudiando los fundamentos del sistema de seguridad que incorpora Android. Sin embargo, a pesar de lo que cabría esperar, el sistema de seguridad no va a impedir que las aplicaciones puedan realizar cualquier tipo de acción con el dispositivo.

En la segunda parte del capítulo, se describe el API que incorpora Android para permitir conocer la posición geográfica del dispositivo. Estos servicios se basan principalmente en el GPS, pero también disponemos de novedosos servicios de localización basados en telefonía móvil y redes Wi-Fi. A lo largo de este capítulo mostraremos una serie de ejemplos que te permitirán aprender a utilizar estas funciones.

Terminaremos el capítulo describiendo cómo podemos incorporar a nuestra aplicación servicios realizados por terceros. En concreto instalaremos una vista que permite representar un mapa de Google Maps.

**Objetivos**:

- Mostrar los pilares de la seguridad en Android.
- Describir cómo Android crea un usuario Linux asociado a cada aplicación.
- Describir el esquema de permisos en Android y enumerar los permisos más importantes.
- Mostrar cómo los permisos de Android pueden ser ampliados con permisos definidos por el usuario y enumerar los pasos a seguir para crear un nuevo permiso.
- Describir las APIs de Android para la geolocalización y los diferentes tipos de sistemas de posicionamiento disponibles.

- Ver lo sencillo que resulta incorporar en nuestra aplicación un servicio de un tercero. En concreto, Google Maps.
- Seguir mejorando Asteroides al introducir el problema del acceso concurrente a datos y cómo se puede solucionar el Java mediante la palabra reservada `synchronized`.

## 7.1. Los tres pilares de la seguridad en Android

**Poli[Media]**: *Seguridad en Android.*

La seguridad es un aspecto clave de todo sistema. Si nos descargáramos una aplicación maliciosa de Internet o del Play Store, esta podría leer nuestra lista de contactos, averiguar nuestra posición GPS, mandar toda esta información por Internet y terminar enviando 50 mensajes SMS.

En algunas plataformas, como Windows Mobile, estamos prácticamente desprotegidos ante este tipo de aplicaciones. Por lo tanto, los usuarios han de ser muy cautos antes de instalar una aplicación.

En otras plataformas, como en *iOS* del *iPhone*, toda aplicación ha de ser validada por Apple antes de poder ser instalada en un teléfono. Esto limita a los pequeños programadores y da un poder excesivo a Apple. Se trata de un planteamiento totalmente contrario al *software* libre.

Android propone un esquema de seguridad que protege a los usuarios, sin la necesidad de imponer un sistema centralizado y controlado por una única empresa. La seguridad en Android se fundamenta en los siguientes tres pilares:

Como se comentó en el primer capítulo Android está basado en Linux, por lo tanto, vamos a poder aprovechar la seguridad que incorpora este SO. De esta forma Android puede impedir que las aplicaciones tengan acceso directo al *hardware* o interfieran con recursos de otras aplicaciones.

Toda aplicación ha de ser firmada con un certificado digital que identifique a su autor. La firma digital también nos garantiza que el fichero de la aplicación no ha sido modificado. Si se desea modificar la aplicación está tendrá que ser firmada de nuevo, y esto solo podrá hacerlo el propietario de la clave privada. No es preciso (ni frecuente) que el certificado digital sea firmado por una autoridad de certificación.

Si queremos que una aplicación tenga acceso a partes del sistema que pueden comprometer la seguridad del sistema hemos de utilizar un modelo de permisos, de forma que el usuario conozca los riesgos antes de instalar la aplicación.

En los siguientes apartados se describe con más detalle el primer y tercer punto. El proceso de firmar una aplicación será descrito en el último capítulo.

### 7.1.1. Usuario Linux y acceso a ficheros

Para proteger el acceso a recursos utilizados por otras aplicaciones, Android crea una cuenta de usuario Linux (user ID) nueva por cada paquete (.apk) instalado en el sistema. Este usuario es creado cuando se instala la aplicación y permanece constante durante toda su vida en el dispositivo.

Cualquier dato almacenado por la aplicación será asignado a su usuario Linux, por lo que normalmente no tendrán acceso otras aplicaciones. No obstante, cuando crees un fichero puedes usar los modos `MODE_WORLD_READABLE` y/o `MODE_WORLD_WRITEABLE` para permitir que otras aplicaciones puedan leer o escribir en el fichero. Aunque otras aplicaciones puedan escribir el fichero, el propietario siempre será el usuario asignado a la aplicación que lo creó.

Dado que las restricciones de seguridad se garantizan a nivel de proceso, el código de dos paquetes no puede, normalmente, ejecutarse en el mismo proceso. Para ello sería necesario usar el mismo usuario. Puedes utilizar el atributo `sharedUserId` en `AndroidManifest.xml` para asignar un mismo usuario Linux a dos aplicaciones. Con esto conseguimos que a efectos de seguridad ambas aplicaciones sean tratadas como una sola. Por razones de seguridad, ambas aplicaciones han de estar firmadas con el mismo certificado digital.

### 7.1.2. El esquema de permisos en Android

Para proteger ciertos recursos y características especiales del *hardware*, Android define un esquema de permisos. Toda aplicación que acceda a estos recursos está obligada a declarar su intención de usarlos. En caso de que una aplicación intente acceder a un recurso del que no ha solicitado permiso, se generará una excepción de permiso y la aplicación será interrumpida inmediatamente.

Cuando el usuario instala una aplicación, este podrá examinar la lista de permisos que solicita la aplicación y decidir si considera oportuno instalar dicha aplicación. Veamos la descripción de algunos permisos. Abajo se describe el esquema utilizado:

> NOMBRE_DEL_PERMISO ⚠ *Grupo al que pertenece – Descripción del permiso que se mostrará al usuario antes de instalar la aplicación* (grado de relevancia de seguridad a juicio del autor). Descripción y comentarios.

A continuación, se muestra una lista con algunos de los permisos más utilizados en Android:

- CALL_PHONE ⚠ *Servicios por los que tienes que pagar – Llamar directamente a números de teléfono* (muy alta). Permite realizar llamadas sin la intervención del usuario.
- SEND_SMS ⚠ *Servicios por los que tienes que pagar – Envío SMS/MMS* (muy alta). Permite a la aplicación mandar SMS.

- WRITE_EXTERNAL_STORAGE ⚠ Almacenamiento — Modificar/borrar archivos en SD (alta). Permite la lectura de archivos y su modificación, típico en aplicaciones de *backup*.

- READ_OWNER_DATA ⚠ Tu información personal — Leer datos de contacto (alta). Permite leer información sobre el propietario del teléfono (nombre, correo electrónico, número de teléfono). Muy peligroso, algunas aplicaciones podrían utilizar esta información de forma no lícita.

- READ_CALENDAR ⚠ Tu información personal — Leer datos de calendario (moderada). Solo otorga este permiso si consideras que la aplicación realmente lo necesita.

- WRITE_CALENDAR ⚠ Tu información personal — Escribir datos de calendario (moderada). Este permiso no permite leer el calendario. Por las mismas razones que el anterior permiso, hay que plantearse si una aplicación nos pide este permiso con sentido o no.

- READ_PHONE_STATE ⚠ Llamadas de teléfono - Leer estado del teléfono e identidad (alta). Muchas aplicaciones piden este permiso para ponerse en pausa mientras hablas por teléfono. Sin embargo, también permite el acceso al IMEI, IMSI y al identificador único de 64 bits que Google asigna a cada terminal. En las primeras versiones de SDK este permiso no era necesario.

- ACCESS_FINE_LOCATION ⚠ Tu ubicación — Precisar la ubicación (GPS) (moderada). Localización basada en GPS.

- ACCESS_COARSE_LOCATION ⚠ Tu ubicación — Ubicación común (basada en red) (moderada). Localización basada en telefonía móvil (Cel-ID) y Wi-Fi.

- BLUETOOTH ⚠ Comunicación de red — Crear conexión Bluetooth (baja). Conexión con otro dispositivo *Bluetooth*.

- INTERNET ⚠ Comunicación de red — Acceso íntegro a Internet (muy alta). Permite establecer conexiones a través de Internet. Este es posiblemente el permiso más importante, en el que hay que fijarse más a quién se le otorga. Muchas aplicaciones lo piden, pero no todas lo necesitan. Cualquier *malware* necesita una conexión para poder enviar datos de nuestro dispositivo.

- ACCESS_WIFI_STATE ⚠ Comunicación de red — Ver estado de conexión, ver estado de Wi-Fi (baja). Información sobre redes WiFi disponibles.

- ⚠ Herramientas del sistema — Impedir que el teléfono entre en modo de suspensión (baja). Algunas aplicaciones pueden necesitar este permiso, y realmente a lo único que puede afectar es a nuestra batería.

- CHANGE_CONFIGURATION ⚠ Herramientas del sistema — Modificar la configuración global del sistema (moderada). Permite cambiar la configuración (como *locale*). Pese a que es importante en sí, es muy común, así por ejemplo los *widgets* lo necesitan.

- READ_SYNC_SETTINGS ⚠ Herramientas del sistema – Leer ajustes de sincronización (baja). Tan solo permite saber si tienes sincronización en segundo plano con alguna aplicación (como con un cliente de Twitter o Gmail).

- WRITE_APN_SETTINGS ⚠ Herramientas del sistema – Escribir configuración del Punto de Acceso (moderada). Permite la configuración del punto de acceso a la red. Por ejemplo, encender y apagar tu conexión de red o Wi-Fi.

- MANAGE_APP_TOKENS ⚠ Herramientas del sistema – Recuperar aplicaciones en ejecución (moderada). Permite saber qué aplicaciones están corriendo en segundo plano y cambiar el orden.

- SET_PREFERRED_APPLICATIONS ⚠ Herramientas del sistema – Establecer aplicaciones preferidas (moderada). Permite la asignación a una aplicación para que haga determinada tarea. Por ejemplo, la reproducción de vídeos.

- VIBRATE ⚠ Controles de hardware – Control de la vibración (baja). Permite hacer vibrar al teléfono. Los juegos lo suelen utilizar bastante.

- CAMERA ⚠ Controles de hardware – Realizar fotografías (baja). Permite acceso al control de la cámara y a la toma de imágenes.

- RECORD_AUDIO ⚠ Controles de hardware – Grabar audio (moderada). Permite grabar sonido desde el micrófono del teléfono.

Para solicitar un determinado permiso en tu aplicación, no tienes más que incluir una etiqueta `<uses-permission>` en el fichero `AndroidManifest.xml` de tu aplicación. En el siguiente ejemplo se solicitan dos permisos:

```
<manifest package="org.example.mi_aplicacion" >
    ...
    <uses-permission android:name="android.permission.RECEIVE_SMS"/>
    <uses-permission android:name="android.permission.SEND_SMS"/>
</manifest>
```

## 7.1.3. Permisos definidos por el usuario en Android

Además de los permisos definidos por el sistema, los desarrolladores vamos a poder crear nuevos permisos para restringir el acceso a elementos de nuestro *software*.

**Poli[Media]**: *Permisos definidos por el usuario en Android.*

Abordaremos el estudio de la creación de nuevos permisos utilizando el siguiente ejemplo. Somos la empresa *PayPerView* especializada en ofrecer servicios de reproducción de vídeos bajo demanda. Queremos crear un *software* que permita a cualquier desarrollador reproducir nuestros vídeos desde sus aplicaciones. No obstante, este servicio no es gratuito, por lo que nos interesa que el usuario sea advertido cuando se instale la aplicación del tercero, indicándole que esta aplicación va hacer uso de un servicio no gratuito.

Para definir el nuevo permiso utilizaremos el tag `<permission>` en el fichero `AndroidManifest.xml` de nuestro *software*. A continuación, se muestra un ejemplo:

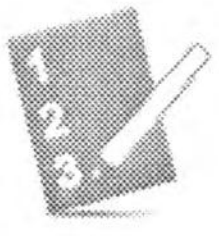
```
<manifest package="com.payperview.servicios">
  ...
  <permission android:name="com.payperview.servicios.VER_VIDEOS"
     android:label="@string/etiqueta"
     android:description="@string/descripcion"
     android:permissionGroup="android.permission-group.COST_MONEY"
     android:protectionLevel="dangerous" />
</manifest>
```

El atributo `android:name` indica el nombre del permiso. Como ves ha de estar dentro del mismo dominio que nuestra aplicación. El atributo `android:permissionGroup` es opcional y permite incluir nuestro permiso en un grupo. En el ejemplo se ha incluido en el grupo de permisos que pueden suponer un coste económico al usuario. El atributo `android:protectionLevel` informa al sistema cómo el usuario ha de ser informado y qué aplicaciones tienen acceso al permiso. Los valores posibles se indican a continuación:

| | |
|---|---|
| `normal` | El usuario no es advertido de que se va utilizar el permiso. |
| `dangerous` | El usuario es advertido. |
| `signature` | Solo aplicaciones firmadas con la misma firma digital que la aplicación que declara el permiso. |
| `signatureOrSystem` | Igual que `signature` pero además puede ser usado por el sistema. Caso poco frecuente, donde varios fabricantes necesitan compartir características a través de la imagen del sistema. |

Los atributos `android:label` y `android:description` son opcionales y han de ser introducidos a través de un recurso de cadena. En estas cadenas hay que describir el permiso de forma abreviada y extensa, respectivamente. Veamos cómo podría ser en el ejemplo:

```
<string name="etiqueta">reproducción de videos bajo
    demanda</string>
<string name="descripcion">Permite a la aplicación reproducir
    videos de la empresa PayPerView sin tu intervención. Se trata
    de un servicio no gratuito, por lo que puede afectar al saldo
    de tu cuenta con esta empresa. Si no tienes una cuenta abierta
    los videos no podrán ser reproducidos. </string>
```



**Ejercicio paso a paso:** *Creando tus propios permisos.*

Vamos a comprobar cómo usar este permiso con este ejercicio:

1. Crea un nuevo proyecto con los siguientes datos:

```
Project name: NuevoPermiso
Build Target: Android 1.5
Application name: Nuevo Permiso
Package name: com.payperview.servicios
Create Activity: NuevoPermiso
Min SDK Version: 3
```

2. Crea una nueva actividad en este proyecto con nombre `VerVideo` y copia el mismo código de la actividad `NuevoPermiso`. En el ejemplo real esta actividad sería la responsable de la reproducción de vídeos.

3. Modifica el fichero `AndroidManifest.xml` según el código mostrado a continuación:

```
<manifest…>
  <application…>
     …
     <activity
          android:name="VerVideo"
          android:permission=
"com.payperview.servicios.VER_VIDEOS">
          <intent-filter>
              <action android:name="android.intent.action.MAIN" />
              <category android:name=
                              "android.intent.category.LAUNCHER"/>
          </intent-filter>
      </activity>
  </application>
</manifest>
```

4. Copia detrás de `</application>` la etiqueta `<permission …/>` del ejemplo anterior.

5. Recuerda definir los recursos de cadena `etiqueta` y `descripcion`.

6. Ejecuta el proyecto. Es imprescindible para registrar en el teléfono el nuevo permiso y la nueva actividad que queremos lanzar desde otras aplicaciones.

7. Para usar este servicio crea un nuevo proyecto con los siguientes datos:

```
Project name: UsarPermiso
Build Target: Android1.5
Application name: UsarPermiso
```

```
Package name: org.example.usarpermiso
Create Activity: UsarPermiso
Min SDK Version: 3
```

8. Abre el fichero `main.xml` e inserta el siguiente botón dentro del `<LineraLayout>`:

```
<Button android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="Ver Vídeo"
   android:id="@+id/Button01"/>
```

9. Abre el fichero `UsarPermisos.java` y añade el siguiente código al final de la función `onCreate()`:

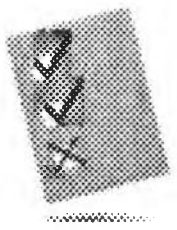
```
Button b = (Button)findViewById(R.id.Button01);
b.setOnClickListener(new OnClickListener(){
   @Override
   public void onClick(View view){
      Intent i = new Intent();
      i.setClassName("com.payperview.servicios",
            "com.payperview.servicios.VerVideo");
      startActivity(i);
   }
});
```

**Nota sobre Java:** *En este código no se han incluido los import, pulsa **Ctrl-Shift-O** para añadirlos de forma automática.*

10. Ejecuta la aplicación. Cuando pulses el botón la aplicación provocará un error.
11. Visualiza la ventana *LogCat* para verificar que se trata de un error de permiso.
12. Para solucionar el problema tendrás que incluir el siguiente código al final del fichero `AndroidManifest.xml`:

```
<manifest ...>
   ...
   <uses-permission
         android:name="com.payperview.servicios.VER_VIDEOS"/>
</manifest>
```

13. Comprueba cómo ahora se accede al servicio sin problemas.



**Preguntas de repaso y reflexión:** *Los permisos en Android.*

## 7.2. Localización

La plataforma Android dispone de un interesante sistema de posicionamiento que combina varias tecnologías:

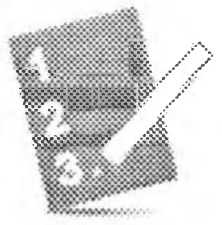
Sistema de localización global basado en GPS. Este sistema solo funciona si disponemos de visibilidad directa de los satélites.

Sistema de localización basado en la información recibida de las torres de telefonía celular y de puntos de acceso Wi-Fi. Funciona en el interior de los edificios.

Estos servicios se encuentran totalmente integrados en el sistema y son usados por gran variedad de aplicaciones. Por ejemplo, la aplicación *Locale*[1] de Android puede adaptar la configuración del teléfono según donde se encuentre. Podría por ejemplo poner el modo de llamada en vibración cuando estemos en el trabajo.

El sistema de posicionamiento global, GPS, fue diseñado inicialmente con fines militares pero hoy en día es ampliamente utilizado para uso civil. Gracias al desfase temporal de las señales recibidas por varios de los 31 satélites desplegados, este sistema es capaz de posicionarnos en cualquier parte del planeta con una precisión de 15 metros.

El GPS presenta un inconveniente; solo funciona cuando tenemos visión directa de los satélites. Para solventar este problema, Android combina esta información con la recibida de las torres de telefonía celular y de puntos de acceso Wi-Fi.



### Ejercicio paso a paso: *El API de localización de Android.*

En este ejercicio crearemos una aplicación que es capaz de leer información de localización del dispositivo y actualizarla cada vez que se produce un cambio.

1. Crea un nuevo proyecto con los siguientes datos:

```
Project name: Localizacion
Build Target: Android 1.6
Application name: Localizacion
Package name: org.example.localizacion
Create Activity: Localizacion
Min SDK Version: 4
```

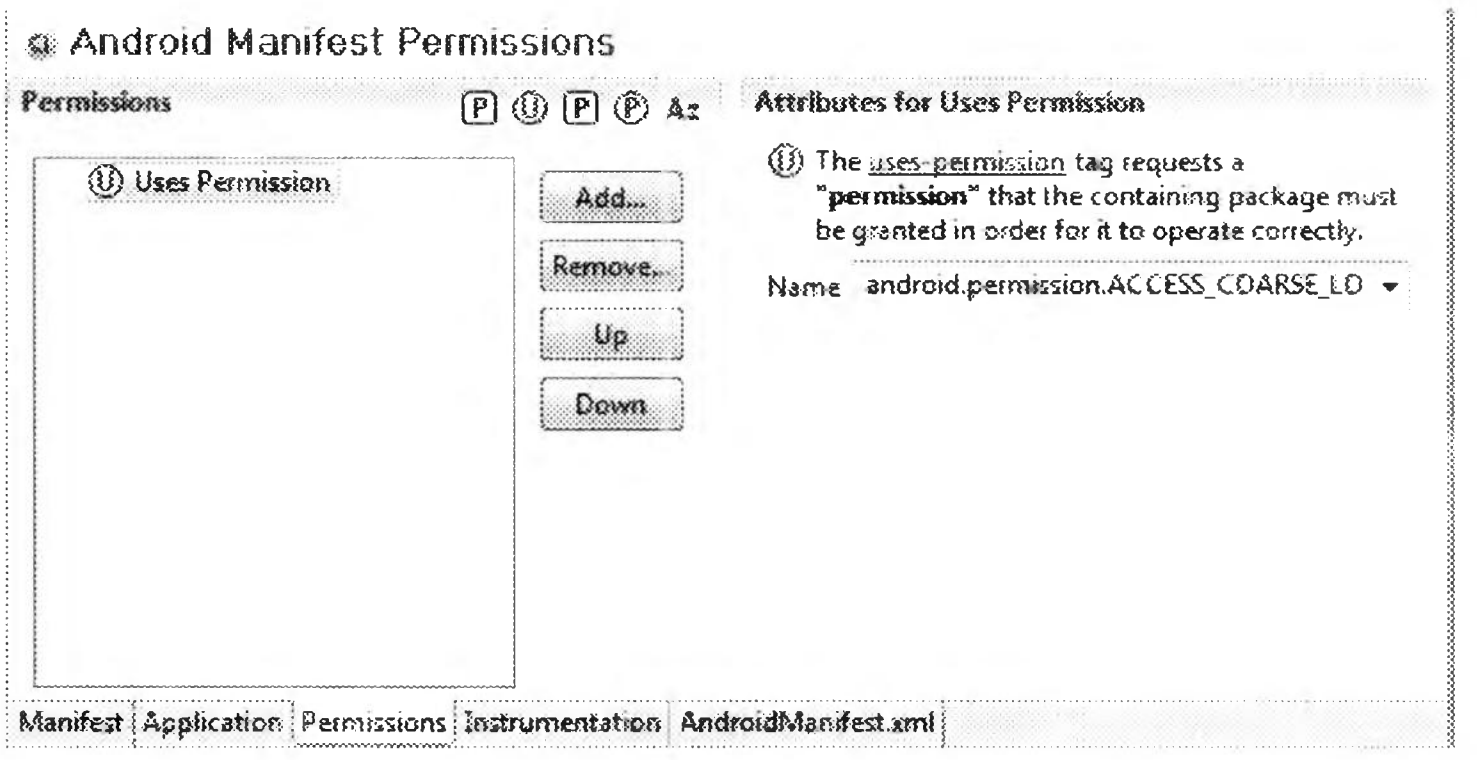
2. Por razones de seguridad acceder a la información de localización está, en principio, prohibido a las aplicaciones. Si estas desean hacer uso de dicho servicio han de solicitar permisos especiales. Estos permisos hay que indicarlos en el fichero `AndroidManiest.xml`. En concreto, esta aplicación necesita los permisos de localización precisa y localización imprecisa:

---

[1] http://www.androidlocale.com

```
ACCESS_FINE_LOCATION
ACCESS_COARSE_LOCATION
```

Puedes hacerlo a través de los cuadros de diálogos, como se muestra a continuación:

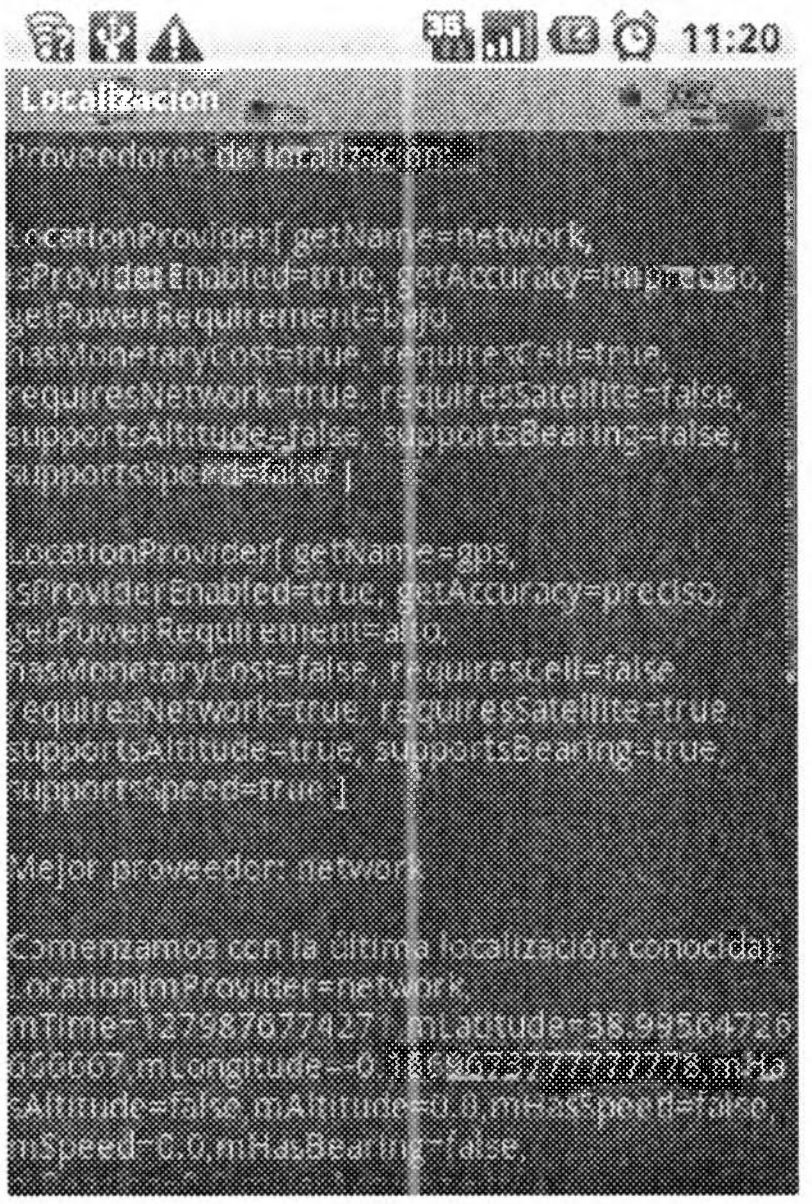



Añadiendo las siguientes líneas en el fichero XML.

```
<uses-permissionandroid:name="android.permission.ACCESS_COARSE_LOCATION"/>
<uses-permissionandroid:name="android.permission.ACCESS_FINE_LOCATION"/>
```

En este ejemplo de posicionamiento vamos a utilizar tanto la localización fina, que nos proporciona el GPS, como una menos precisa, proporcionada por las torres de telefonía celular y WiFi.

3. En este ejemplo nos limitaremos a mostrar en modo de texto la información obtenida desde el API de localización, tal y como se muestra en la siguiente pantalla:



Como puede observarse nos limitaremos a mostrar un `TextView` con *scroll*. Sustituye el fichero `res/layout/main.xml` por:

```
<ScrollView
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent">
     <TextView
        android:id="@+id/salida"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content" />
</ScrollView>
```

4. Ve ahora al fichero `LocationTest.java` y copia el siguiente código:

```
public class Localizacion extends Activity implements LocationListener {

   private static final String[] A = { "n/d", "preciso", "impreciso" };
   private static final String[] P = { "n/d", "bajo", "medio","alto" };
   private static final String[] E = { "fuera de servicio",
                        "temporalmente no disponible ", "disponible" };
   private LocationManager manejador;
   private TextView salida;
   private String proveedor;

   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);
      salida = (TextView) findViewById(R.id.salida);

      manejador = (LocationManager) getSystemService(LOCATION_SERVICE);
      log("Proveedores de localización: \n ");
      muestraProveedores();

      Criteria criteria = new Criteria();
      proveedor = manejador.getBestProvider(criteria, true);
      log("Mejor proveedor: " + proveedor + "\n");

      log("Comenzamos con la última localización conocida:");
      Location localizacion = manejador.getLastKnownLocation(proveedor);
      muestraLocaliz(localizacion);
   }
```

La primera línea que nos interesa es la llamada a `getSystemService (LOCATION_SERVICE)` que crea el objeto `manejador` de tipo `LocationManager`.

La siguiente línea hace una llamada al método `log()` que será definido más adelante. Simplemente saca por el `TextView` el texto indicado.

La siguiente llamada a `muestraProveedores()` también es un método definido por nosotros, que listará todos los proveedores de localización disponibles.

En las tres siguientes líneas vamos a seleccionar uno de estos proveedores de localización. Para ello usaremos el método `getBestProvider()`. En este método hay que indicar un criterio de selección. Aquí se podría indicar restricciones de coste, potencia, precisión, etc. En este ejemplo no indicamos ninguna restricción.

Dependiendo del proveedor puede tardar un cierto tiempo en darnos una primera posición. No obstante, Android recuerda la última posición que fue devuelta por este proveedor. Es lo que se hace en las últimas líneas del programa. El método `muestraLocaliz()` será definido más tarde y muestra en pantalla una determinada localización.

5. Copia, a continuación, el resto del código:

```
// Métodos del ciclo de vida de la actividad
@Override  protected void onResume() {
   super.onResume();
   // Activamos notificaciones de localización
   manejador.requestLocationUpdates(proveedor, 10000, 1, this);
}

@Override  protected void onPause() {
   super.onPause();
   // Desactivamos notificaciones para ahorrar batería
   manejador.removeUpdates(this);
}

// Métodos de la interfaz LocationListener
public void onLocationChanged(Location location) {
   log("Nueva localización: ");
   muestraLocaliz(location);
}

public void onProviderDisabled(String proveedor) {
   log("Proveedor deshabilitado: " + proveedor + "\n");
}

public void onProviderEnabled(String proveedor) {
   log("Proveedor habilitado: " + proveedor + "\n");
}

public void onStatusChanged(String proveedor, int estado,
   Bundle extras) {
   log("Cambia estado proveedor: " + proveedor + ", estado="
         + E[Math.max(0,estado)] + ", extras=" + extras + "\n");
}
```

```
// Métodos para mostrar información
private void log(String cadena) {
  salida.append(cadena + "\n");
}

private void muestraLocaliz(Location localizacion) {
  if (localizacion == null)
     log("Localización desconocida\n");
  else
     log(localizacion.toString() + "\n");
}

private void muestraProveedores() {
  List<String> proveedores = manejador.getAllProviders();
  for (String proveedor : proveedores) {
     muestraProveedor(proveedor);
  }
}

private void muestraProveedor(String proveedor) {
  LocationProvider info = manejador.getProvider(proveedor);
  log("LocationProvider[ "+"getName=" + info.getName()+
     ", isProviderEnabled=" +
                    manejador.isProviderEnabled(proveedor)+
     ", getAccuracy=" + A[Math.max(0, info.getAccuracy())]+
     ", getPowerRequirement=" +
                    P[Math.max(0, info.getPowerRequirement())]+
     ", hasMonetaryCost=" + info.hasMonetaryCost()+
     ", requiresCell=" + info.requiresCell()+
     ", requiresNetwork=" + info.requiresNetwork()+
     ", requiresSatellite=" + info.requiresSatellite()+
     ", supportsAltitude=" + info.supportsAltitude()+
     ", supportsBearing=" + info.supportsBearing()+
     ", supportsSpeed=" + info.supportsSpeed()+" ]\n");
}
```

Para conseguir que se notifiquen cambios de posición hay que llamar al método `requestLocationUpdates()` y para indicar que se dejen de hacer las notificaciones hay que llamar a `removeUpdates()`. Dado que queremos ahorrar batería, nos interesa que se reporten notificaciones solo cuando la aplicación esté activa. Por lo tanto, tenemos que reescribir los métodos `onResume()` y `onPause()`.

El método `requestLocationUpdates()` dispone de 4 parámetros: el nombre del proveedor, el tiempo entre actualizaciones en ms (se recomiendan valores mayores de 60.000 ms), la distancia mínima (de manera que si es menor, no se notifica) y un objeto `LocationListener`.

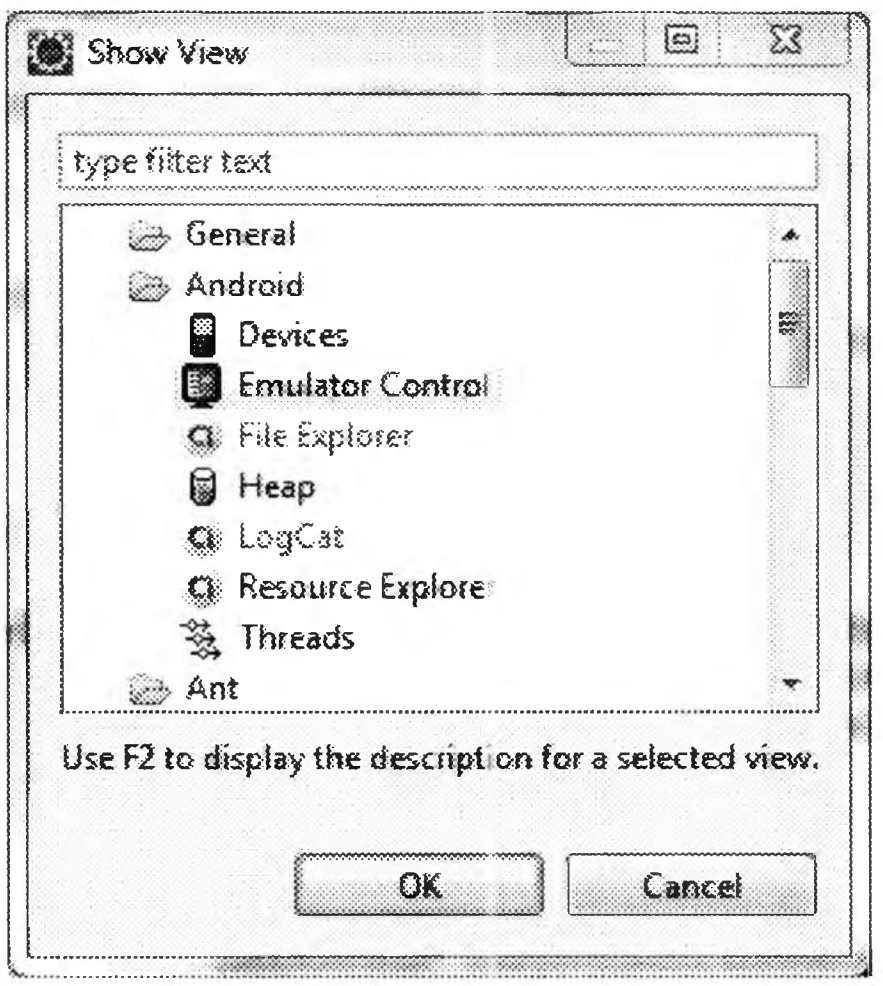
A continuación, implementamos los métodos de un `LocationListener`: `onLocationChanged` se activará cada vez que cambie la posición. Los otros tres métodos pueden ser usados para cambiar de proveedor en caso de que se active uno mejor o deje de funcionar el actual. Sería buena idea llamar de nuevo aquí al método `getBestProvider()`.

El resto del código resulta fácil de interpretar.

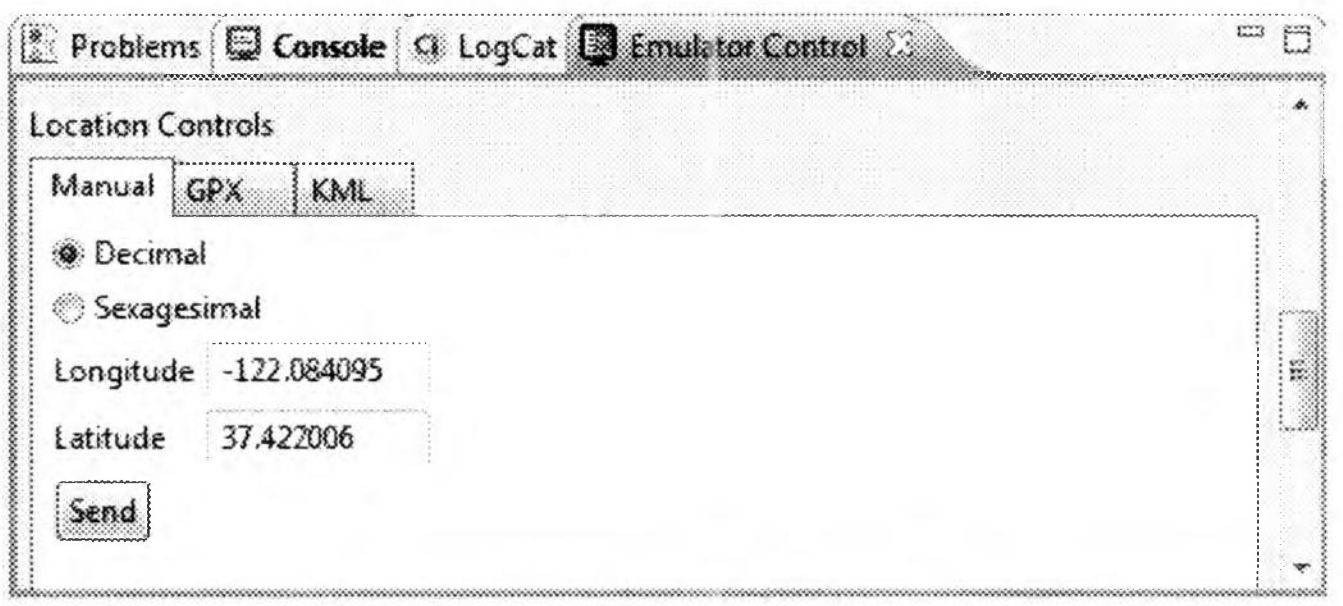
6. Verifica el funcionamiento del programa, si es posible con un dispositivo real.

### 7.2.1. Emulación del GPS con Eclipse

Muy probablemente el ordenador donde estés trabajando no disponga de GPS, por lo que sería muy difícil que este programa funcione en el emulador. No hay problema, el plug-in de Android para Eclipse proporciona un sistema de emulación del GPS. Para activarlo sigue los siguientes pasos. Accede al menú *Window* > *Show View* >*Others*... > *Android* > *Emulator Control*.




Aparecerá una ventana como la siguiente:



***NOTA:*** *Es posible que el GPS no reciba ninguna señal. El problema está relacionado con el formato de las coordenadas enviadas al emulador. Por algún motivo (probablemente relacionado con el carácter usado para separar los decimales, la coma en español y punto en inglés) el emulador solo recibe correctamente las actualizaciones si la configuración regional está establecida en el idioma inglés. Para solucionar este problema basta cambiar el* locale *del runtime de java. Si utilizas el plug-in para Eclipse, añade '-Duser.language=en' al archivo eclipse.ini. Más información en* [2].

## 7.3. Google Maps

Google Maps nos proporciona un servicio de cartografía *online* que podremos utilizar en nuestras aplicaciones Android. Veamos las claves necesarias para utilizarlo.

En primer lugar, para hacer uso de este servicio necesitas una clave de Google Maps. Puedes encontrar información en http://code.google.com/intl/es/apis/maps/signup.html. Siempre es una buena idea revisar los términos y condiciones. He aquí algunos ejemplos:

- Hay un límite en el número de solicitudes de codificación geográfica por día, 15.000.
- Publicidad (*AdSense / AdWords*) no está incluida en la API de Google Maps.
- Estás obligado a ofrecer mapas de Google como un servicio gratuito para tus usuarios.

### 7.3.1. Obtención de una clave Google Maps

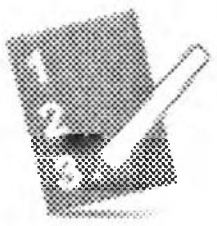
Para poder utilizar este servicio de Google, igual que como ocurre cuando se utiliza desde una página web, va a ser necesario registrar la aplicación que lo utilizará. Tras registrar la aplicación, se nos entregará una clave que tendremos que indicar en la aplicación.

Realmente vamos a necesitar dos claves diferentes, una durante el proceso de desarrollo y otra para la aplicación final. La razón es que se genera una clave diferente en función del certificado digital con la que se firma la aplicación. En la fase de desarrollo las aplicaciones también han de ser firmadas digitalmente, pero en este caso el SDK utiliza un certificado especial utilizado solo en la fase de desarrollo.

Veamos cómo obtener la clave *Google Maps* para el certificado de depuración. En caso de querer distribuir tu aplicación, una vez terminada tendrás que firmarla con un certificado digital propio. Este proceso se explica en el último capítulo. Recuerda que será necesario remplazar la clave *Google Maps* por otra, esta última asociada a tu certificado digital.

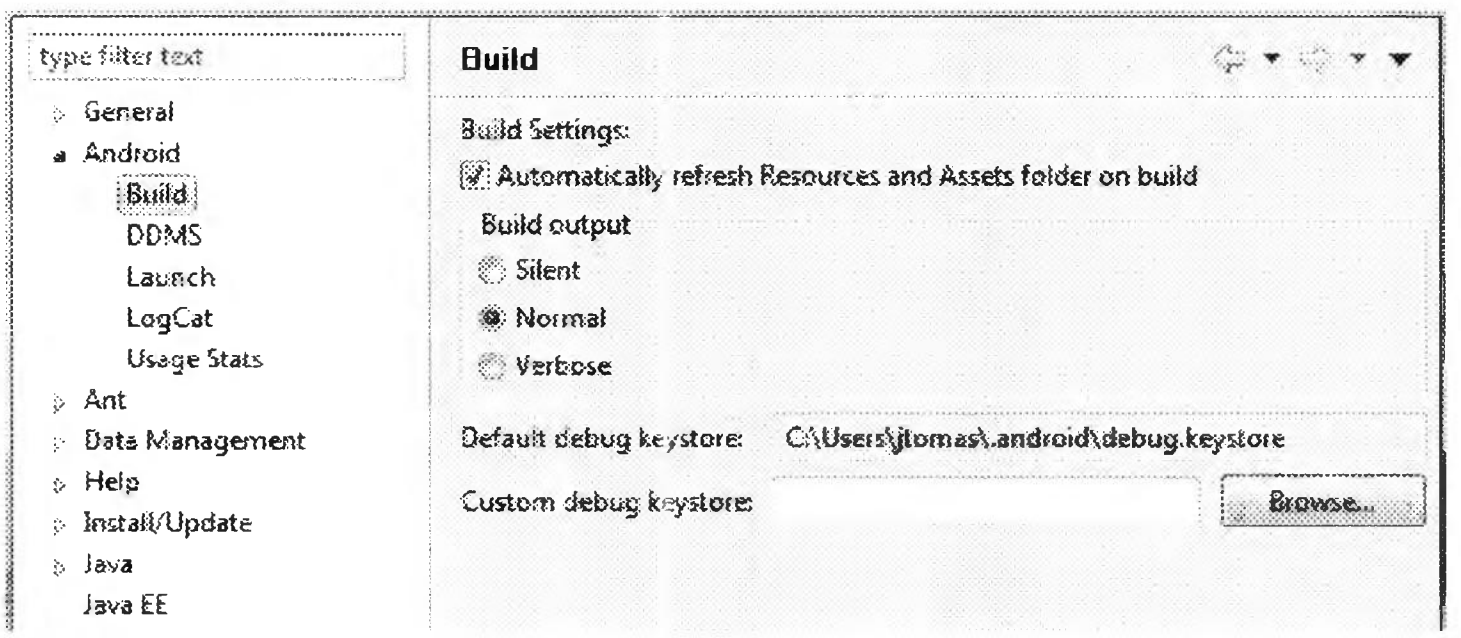
---

[2] http://code.google.com/p/android/issues/detail?id=915



**Ejercicio paso a paso:** *Obtención de una clave Google Maps.*

1. El primer paso va a consistir en descubrir donde está almacenado el certificado digital de depuración. Utilizando el entorno Eclipse accede al menú *Windows* > *Preferences* > *Android* > *Build*. Aparecerá el siguiente cuadro de diálogo:




   En el cuadro informativo *Default debug keystore:* aparece la ruta del fichero donde se almacena el certificado digital de depuración.

2. Copia en el portapapeles esta ruta.

3. Ahora necesitamos extraer la huella digital MD5 de este fichero. Para extraer la huella digital puedes utilizar el programa `keytool`. En Windows este programa se encuentra en la carpeta `C:\Program Files\Java\jre6\bin\` o en una similar. Abre un intérprete de comandos y sitúate en la carpeta anterior.

4. Ejecuta el siguiente comando remplazando el nombre del fichero por el que acabas de copiar en el portapapeles.

   ```
   keytool -v -list -keystore [Tu debug.keystore path]
   ```

   En nuestro ejemplo:

   ```
   keytool -v -list -keystore C:\Users\jtomas\.android\debug.keystore
   ```

El programa te solicitará una contraseña para proteger el almacén de claves. La contraseña es `android` o no introduzcas nada. A continuación, te indicará la huella digital del certificado DM5. Como puedes ver en la captura anterior está formado por los siguientes 16 bytes expresados en hexadecimal: `FD:AF:7A:5F:52:AE:51:62:84:18:60:1F:D1:00:FE:79`

5. Copia en el portapapeles esta secuencia de dígitos.

6. Para obtener la clave *Google Maps* entra en la siguiente página Web:

   http://code.google.com/android/maps-api-signup.html

7. Tendrás que introducir tu huella digital y el usuario de Google Mail que realiza la solicitud. Si no dispones de un usuario en Google puedes crear uno nuevo en un par de minutos. El resultado final se muestra a continuación.

**Gracias por suscribirte a la clave del API de Android Maps.**

Tu clave es:

```
0MbSafHcRHUdA0ASRI36rUqst6U2PM5ATMjUkKg
```

Esta clave es válida para todas las aplicaciones firmadas con el certificado cuya huella dactilar sea:

```
FD:AF:7A:5F:52:AE:51:62:84:18:60:1F:D1:00:FE:79
```

8. Recuerda copiar en el portapapeles la clave obtenida, la necesitaremos en el siguiente ejercicio.

**Ejercicio paso a paso:** *Un programa de ejemplo con Google Maps.*

Veamos un sencillo ejemplo que nos permite visualizar un mapa centrado en las coordenadas geográficas detectadas por el sistema de posicionamiento.

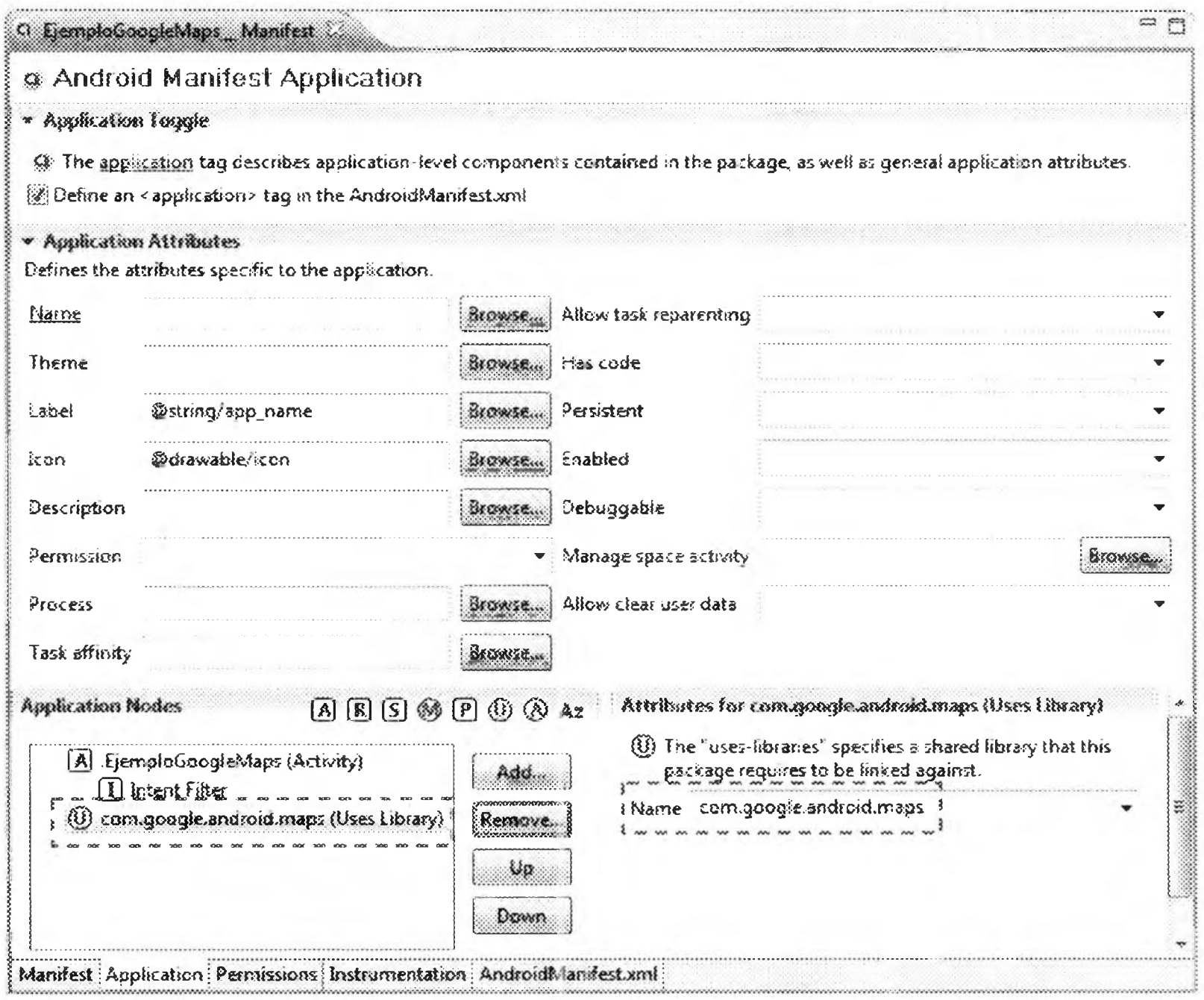
1. Crea un nuevo proyecto con los siguientes datos:

```
Project name: EjemploGoogleMaps
Build Target: Google APIs 1.6
Application name: Ejemplo Google Maps
Packagename: org.example.ejemplogooglemaps
Create Activity: EjemploGoogleMaps
Min SDK Version: 4
```

2. Añade los siguientes permisos a tu aplicación en Android.xml:

```
INTERNET
ACCESS_FINE_LOCATION
ACCESS_COARSE_LOCATION
```

3. Indica que necesitamos la librería de Google Maps en tu aplicación. Entra en `AndroidManifest.xml` y selecciona la lengüeta *Application*. Utilizando el botón "*Add...*" añade un "*Uses library*" con la librería: `com.google.android.maps`




En el fichero `AndroidManifest.xml` se añadirá la siguiente línea:

```
<uses-library android:name="com.google.android.maps"/>
```

4. Remplaza el contenido del *layout* `main.xml` por el siguiente código:

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

    <com.google.android.maps.MapView
        android:id="@+id/mapa"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:clickable="true"
        android:apiKey=">>> Tu clave de Google Maps <<<"/>
</RelativeLayout>
```

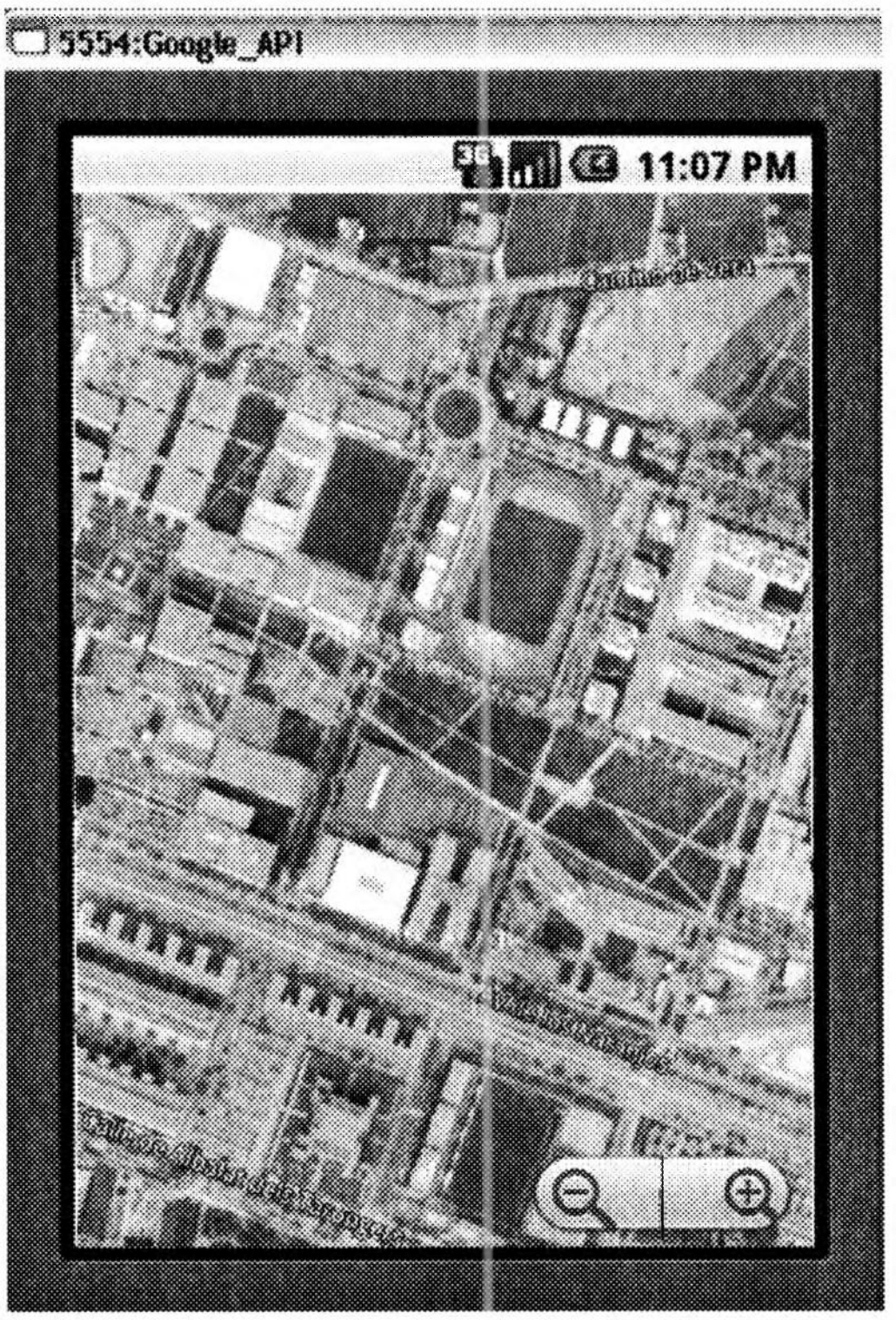
5. Sustituye "`>>> Tu clave de Google Maps <<<`" por la clave de Google Maps obtenida en el apartado anterior.

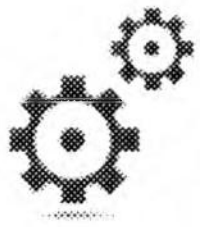
6. Remplaza el código de `EjemploGoogleMaps.java` por:

```
public class EjemploGoogleMaps extends MapActivity
                                    implements LocationListener {
  private MapController mapController;
  private MapView mapView;
  private LocationManager manejador;

@Override
public void onCreate(Bundle bundle) {
  super.onCreate(bundle);
  setContentView(R.layout.main);
  mapView = (MapView) findViewById(R.id.mapa);
  mapView.setBuiltInZoomControls(true);      //Activa controles zoom
  mapView.setSatellite(true);                //Activa vista satélite
  mapView.setStreetView(false);              //Desactiva StreetView
  mapView.setTraffic(false);      //Desactiva información de tráfico
  mapController = mapView.getController();
  mapController.setZoom(14);      // Zoom 1 ver todo el mundo
  manejador = (LocationManager)
                          getSystemService(Context.LOCATION_SERVICE);
  manejador.requestLocationUpdates(LocationManager.GPS_PROVIDER,
                                                    10000, 1, this);
}
@Override protected boolean isRouteDisplayed() {
  return false;
}

@Override
public void onLocationChanged(Location location) {
     int lat = (int) (location.getLatitude() * 1E6);
     int lng = (int) (location.getLongitude() * 1E6);
     GeoPoint point = new GeoPoint(lat, lng);
     mapController.setCenter(point);
}

@Override
public void onProviderDisabled(String provider) {}

@Override
public void onProviderEnabled(String provider) {}

@Override
public void onStatusChanged(String provider, int status,
                                                  Bundle extras){}
}
```

Si estás utilizando el emulador, en lugar de un teléfono real, utiliza la vista "Emulator Control" para indicar las coordenadas geográficas a visualizar.

Por ejemplo, la Universidad Politécnica de Valencia se encuentra en Latitud: 39.47987 y Longitud: -0.33874. El resultado se muestra a continuación:



**Práctica:** *Un programa de ejemplo con Google Maps.*

1. En el ejemplo anterior, modifica algunos parámetros de configuración, como visualizar recorrido *StreetView* o nivel inicial de zoom. Verifica los resultados.
2. Si estás utilizando un teléfono real reemplaza en el código anterior `LocationManager.GPS_PROVIDER` por `LocationManager.NETWORK_PROVIDER`. Ejecuta de nuevo el programa y compara las dos posiciones geográficas obtenidas.

## 7.4. Fragmentando los asteroides

Siguiendo con el juego Asteroides, queremos que cuando el misil alcance un asteroide, este se divida en varios fragmentos. Para conseguirlo puedes seguir las instrucciones del siguiente ejercicio:

## Ejercicio paso a paso: *Fragmentando los asteroides.*

1. Convierte la variable local `drawableAsteroide` declarada en el constructor de la clase `VistaJuego`, en una variable global, que será un *array* de tres elementos:

```
private Drawable drawableAsteroide[]= new Drawable[3];
```

2. En el constructor cuando se quiera trabajar con bitmaps inicializaremos esta variable de la siguiente forma:

```
drawableAsteroide[0] = context.getResources().
                       getDrawable(R.drawable.asteroide1);
drawableAsteroide[1] = context.getResources().
                       getDrawable(R.drawable.asteroide2);
drawableAsteroide[2] = context.getResources().
                       getDrawable(R.drawable.asteroide3);
```

3. Y en caso de querer trabajar con gráficos vectoriales:

```
for (int i=0; i<3; i++) {
  ShapeDrawable dAsteroide = new ShapeDrawable(new PathShape(
                                        pathAsteroide, 1, 1));
  dAsteroide.getPaint().setColor(Color.WHITE);
  dAsteroide.getPaint().setStyle(Style.STROKE);
  dAsteroide.setIntrinsicWidth(50 - i * 14);
  dAsteroide.setIntrinsicHeight(50 - i * 14);
  drawableAsteroide[i] = dAsteroide;
}
```

4. Añade al principio del método `destruyeAsteroide(int i)` el código:

```
int tam;
if(Asteroides.get(i).getDrawable()!=drawableAsteroide[2]){
  if(Asteroides.get(i).getDrawable()==drawableAsteroide[1]){
     tam=2;
  } else {
     tam=1;
  }
  for(int n=0;n<numFragmentos;n++){
     Grafico asteroide = new Grafico(this,drawableAsteroide[tam]);
     asteroide.setPosX(Asteroides.get(i).getPosX());
     asteroide.setPosY(Asteroides.get(i).getPosY());
     asteroide.setIncX(Math.random()*7-2-tam);
     asteroide.setIncY(Math.random()*7-2-tam);
     asteroide.setAngulo((int)(Math.random()*360));
     asteroide.setRotacion((int)(Math.random()*8-4));
     Asteroides.add(asteroide);
  }
}
```

5. Corrige algún error adicional ocasionado por este cambio.

6. Prueba los cambios propuestos anteriormente y verifica que cuando se destruye un asteroide no siempre aparece el mismo número de fragmentos. También es posible que el programa se interrumpa. ¿A qué puede deberse este problema? El siguiente ejercicio trata de explicar este extraño comportamiento.

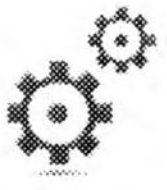
**Ejercicio paso a paso:** *Introduciendo secciones críticas en Java (synchronized).*

Cuando se realiza una aplicación que ejecuta varios hilos de ejecución hay que prestar un especial cuidado a que ambos hilos pueden acceder de forma simultánea a los datos. Cuando se limitan a leer las variables, no suele haber problemas. El problema aparece cuando un hilo está modificando algún dato y justo en este instante se pasa a ejecutar un segundo hilo que ha de leer estos datos. Este segundo hilo va a encontrar unos datos a mitad de modificar, lo que posiblemente cause errores en su interpretación. El método más común para evitar que dos hilos accedan al mismo tiempo a un recurso es el de la exclusión mutua. En Java, se consigue utilizando la palabra reservada `synchronized`.

1. Prueba a introducir la palabra reservada `synchronized` delante del método `onDraw()` y `actualizaFisica()`.
2. Verifica si se ha corregido el problema.

***Nota sobre Java:*** *La palabra clave* `synchronized` *permite definir una sección crítica en Java. Expliquemos en qué consiste: Cada vez que un hilo de ejecución (thread) entra en un método o bloque de instrucciones marcado con* `synchronized` *se pregunta al objeto si ya hay algún otro thread que haya entrado en la sección crítica de ese objeto. La sección crítica está formada por todos los bloques de instrucciones marcados con* `synchronized`. *Si nadie ha entrado en la sección crítica, se entrará normalmente. Si ya hay otro thread dentro, entonces el thread actual es suspendido y ha de esperar hasta que la sección crítica quede libera. Esto ocurrirá cuando el thread, que está dentro de la sección crítica, salga.*

*Dos matizaciones importantes: La primera es que la sección crítica se define a nivel de objeto no de clase. Es decir, cada objeto instanciado no influye en las secciones críticas de otros objetos. En segundo lugar, solo se define una sección crítica por clase. Aunque se haya utilizado* `synchronized` *en varios métodos, realmente solo hay una sección crítica.*



**Práctica:** *Mejorando preferencias en Asteroides*

1. Modifica el programa para que el número de fragmentos generados corresponda con el valor introducido en las preferencias.
2. Puedes aprovechar para que la reproducción de música de fondo y los efectos de audio sean también configurables por el usuario.

# CAPÍTULO 8.
# Servicios, notificaciones y receptores de anuncios

Las aplicaciones que hemos creado hasta el momento estaban formadas por una serie de actividades, cada una de las cuales permitía construir un elemento de interacción con el usuario. Una aplicación en Android va a disponer de otros tipos de componentes; estos serán estudiados en este capítulo.

Cuando sea necesario que parte de una aplicación se ejecute en segundo plano, debajo de otras actividades y, además, no precise de ningún tipo de interacción con el usuario, la opción más adecuada es crear un servicio. Un servicio puede estar en ejecución indefinidamente o puede ser controlado desde una actividad. A lo largo de este capítulo aprenderemos las facilidades proporcionadas por Android para la creación de servicios.

Por otra parte, las notificaciones de la barra de estado constituyen un mecanismo de comunicación vital en Android. Permiten a las aplicaciones que corren en un segundo plano advertir al usuario sobre alertas, avisos o cualquier tipo de información. Las notificaciones se representan como pequeños iconos en la barra superior de la pantalla y se utilizan, habitualmente, para indicar al usuario la llegada de un mensaje, una cita de calendario, una llamada perdida o cualquier otra incidencia de interés. Se trata de una comunicación que no requiere una interacción inmediata del usuario; este puede estar utilizando otra aplicación sin ser interrumpido o puede no estar utilizando el teléfono en ese momento. Este hecho hace de las notificaciones un mecanismo de comunicación ideal para un servicio o para receptores de anuncios. Por lo tanto, este capítulo parece el sitio ideal para describir cómo podemos crear nuestras propias notificaciones y utilizarlas desde nuestras aplicaciones.

Terminaremos el capítulo estudiando otro componente de una aplicación Android, los receptores de anuncios. Un *receptor de anuncios* (*BroadcastReceiver* en inglés) permite realizar acciones cuando se producen anuncios globales de tipo *broadcast*. Existen muchos anuncios originados por el sistema; como por ejemplo *Batería baja, llamada entrante,...* aunque, las aplicaciones también pueden lanzar un *anuncio*

*broadcast* o incluso crear nuevos tipos. Los receptores de anuncios te permitirán crear aplicaciones mucho más integradas en el entorno donde se ejecutan.

**Objetivos:**

- Describir el uso de servicios en Android.
- Enumerar los pasos a seguir cuando queramos crear un servicio para que una tarea se ejecute en segundo plano.
- Mostrar cómo las notificaciones de la barra de estado pueden ser utilizadas como mecanismo de comunicación eficaz con el usuario.
- Repasar los tipos de avisos que pueden utilizar las notificaciones.
- Describir el uso de un servicio como mecanismo de comunicación entre aplicaciones.
- Enumerar los pasos a seguir para crear un receptor de anuncios.
- Enumerar los receptores de anuncios más importantes disponibles en Android

# 8.1. Introducción a los servicios en Android

Polii[Media]: *Los servicios en Android.*

Polii[Media]: *Un servicio para ejecución en segundo plano.*

En muchos casos, será necesario añadir un nuevo componente a tu aplicación para ejecutar algún tipo de acción que se ejecute en segundo plano, es decir, que no requiera una interacción directa con el usuario, pero que queramos que permanezca activo aunque el usuario cambie de actividad. Este es el momento de crear un servicio.

En Android los servicios tienen una doble función:

La primera función permite indicar al sistema que el elemento que estamos creando ha de ejecutarse en segundo plano, normalmente durante un largo período de tiempo. Este tipo de servicios son iniciados mediante el método `startService()`, que indica al sistema que lo ejecute de forma indefinida hasta que alguien le indique lo contrario.

La segunda función permite que nuestra aplicación se comunique con otras aplicaciones, para lo cual ofreceremos ciertas funciones que podrán ser llamadas desde otras aplicaciones. Este tipo de servicios son iniciados mediante el método `bindService()`, que permite establecer una conexión con el servicio e invocar alguno de los métodos que son ofrecidos.

Cada vez que un servicio es creado por alguna de las razones anteriores, el sistema instancia el servicio y llama al método `onCreate()`. Corresponde al servicio implementar el comportamiento adecuado; habitualmente creará un hilo de ejecución (*thread*) secundario donde se realizará el trabajo.

Un servicio en sí es algo muy simple; en este capítulo se verán ejemplos de servicios locales escritos en muy pocas líneas. No obstante, también pueden complicarse, como veremos al final del capítulo cuando tratemos de invocar servicios remotos por medio de una interfaz AIDL.

Un servicio, como el resto de componentes de una aplicación, se ejecuta en el hilo principal del proceso de la aplicación. Por lo tanto, si el servicio necesita un uso intensivo de la CPU o puede quedar bloqueado en ciertas operaciones, como uso de redes, debes crear un hilo diferente para ejecutar estas acciones. También puedes utilizar la clase `IntentService` para lanzar un servicio en su propio hilo.

## 8.1.1. Ciclo de vida de un servicio.

Es importante que recuerdes que un servicio tiene un ciclo de vida diferente a una actividad. A continuación, podemos ver un gráfico que ilustra el ciclo de vida de los servicios:

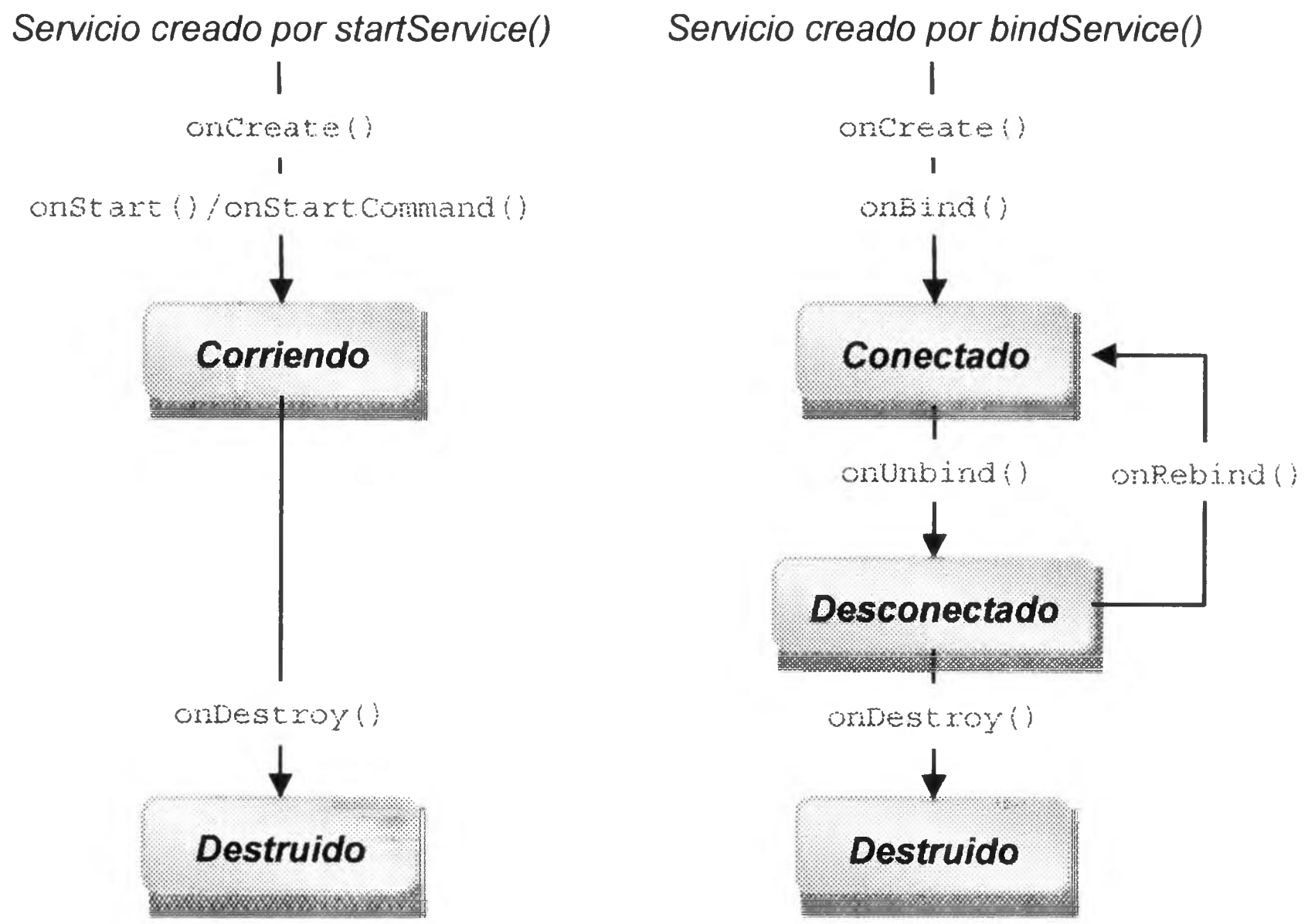



***Figura 1:*** *Ciclo de vida de los servicios.*

Como acabamos de explicar existen dos tipos de servicios según como hayan sido creados. Las funciones de estos servicios son diferentes y, por lo tanto, también su ciclo de vida.

Si el servicio es iniciado mediante `startService()` el sistema comenzará creándolo y llamando a su método `onCreate()`. A continuación llamará a su método `onStartCommand(Intent intent, int flags, int startId)`[1] con los argumentos proporcionados por el cliente. El servicio continuará en ejecución hasta que sea invocado el método `stopService()` o `stopSelf()`.

> ***NOTA: Si se producen varias llamadas a*** `startService()` ***esto no supondrá la creación de varios servicios, aunque sí que se realizarán múltiples llamadas a*** `onStartCommand()`***. No importa cuántas veces el servicio haya sido creado, parará con la primera invocación de*** `stopService()` ***o*** `stopSelf()`***. Sin embargo, podemos utilizar el método*** `stopSelf(int startId)` ***para asegurarnos de que el servicio no parará hasta que todas las llamadas hayan sido procesadas.***

Cuando se inicia un servicio para realizar alguna tarea en segundo plano, el proceso donde se ejecuta podría ser eliminado ante una situación de baja memoria. Podemos configurar la forma en que el sistema reaccionará ante esta circunstancia según el valor que devolvamos en `onStartCommand()`. Existen dos modos principales: devolveremos `START_STICKY` si queremos que el sistema trate de crear de nuevo el servicio cuando disponga de memoria suficiente. Devolveremos `START_NOT_STICKY` si queremos que el servicio sea creado de nuevo sólo cuando llegue una nueva solicitud de creación.

Teniendo en cuenta que los servicios pueden estar largo tiempo en ejecución, el ciclo de vida del proceso que contiene nuestro servicio es un asunto de gran importancia. Conviene aclarar que en situaciones donde el sistema necesite memoria para conservar un servicio, este será siempre menos prioritario que la actividad visible en pantalla, aunque más prioritario que otras actividades en segundo plano. Dado que el número de actividades visibles es siempre reducido, un servicio sólo será eliminado en situaciones de extrema necesidad de memoria. Por otra parte, si un cliente visible está conectado a un servicio, éste también será considerado como visible, siendo tan prioritario como el cliente. En el caso de un proceso que contenga varios componentes, por ejemplo una actividad y un servicio, su prioridad se obtiene como el máximo de sus componentes.

Podemos también utilizar `bindService(Intent servicio, ServiceConnection conexion, int flags)` para obtener una conexión persistente con un servicio. Si dicho servicio no está en ejecución, será creado (siempre que el *flag* `BIND_AUTO_CREATE` esté activo), llamándose al método `onCreate()`, pero no se llamará a `onStartCommand()`. En su lugar se llamará al método `onBind(Intent intencion)` que ha de devolver al cliente un objeto `IBinder` a través del cual se podrá establecer una comunicación entre cliente y servicio. Esta comunicación se establece por medio de

---

[1] En versiones del API inferiores a 2.0 el método llamado será `onStart()`. En versiones recientes se mantiene por razones de compatibilidad.

una interfaz escrita en AIDL, que permite el intercambio de objetos entre aplicaciones que corren en procesos separados. El servicio permanecerá en ejecución tanto tiempo como la conexión esté establecida, independientemente de que se mantenga o no la referencia al objeto `IBinder`.

También es posible diseñar un servicio que pueda ser arrancado de ambas formas (`startService()` y `bindService()`). Este servicio permanecerá activo se ha sido creado desde la aplicación que lo contiene o si recibe conexiones desde otras aplicaciones.

Todo servicio terminará llamando al método `ancestro()` cuando vaya a terminar de forma efectiva.

### 8.1.2. Permisos

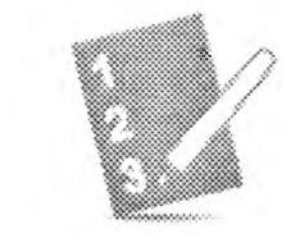
Podemos conseguir que el acceso global a un servicio declarándolo en la etiqueta `<service>` de `AndroidManifest.xml`. También podemos definir un permiso para restringir su acceso. En este caso, las aplicaciones han de declarar este permiso, con el correspondiente `<uses-permission>` en su propio manifiesto.

Podemos definir un permiso para arrancar, parar o conectarse a un servicio. De forma adicional, podemos restringir el acceso a funciones específicas de las ofertadas por un servicio. Para este propósito, podemos llamar al principio de nuestra función a `checkCallingPermission(String)` para verificar si el cliente dispone de un permiso en concreto.

Para más información sobre permisos se recomienda la lectura del capítulo 7.

## 8.2. Un servicio para ejecución en segundo plano

Dentro de los dos usos de un servicio, el más frecuente es permitirnos ejecutar parte de nuestra aplicación en segundo plano.



**Ejercicio paso a paso:** *Un servicio para ejecución en segundo plano de reproducción de música.*

Veamos un ejemplo de servicio que corre en el mismo proceso de la aplicación que lo utiliza. El servicio será creado con la finalidad de reproducir una música de fondo y podrá ser arrancado y detenido desde la actividad principal.

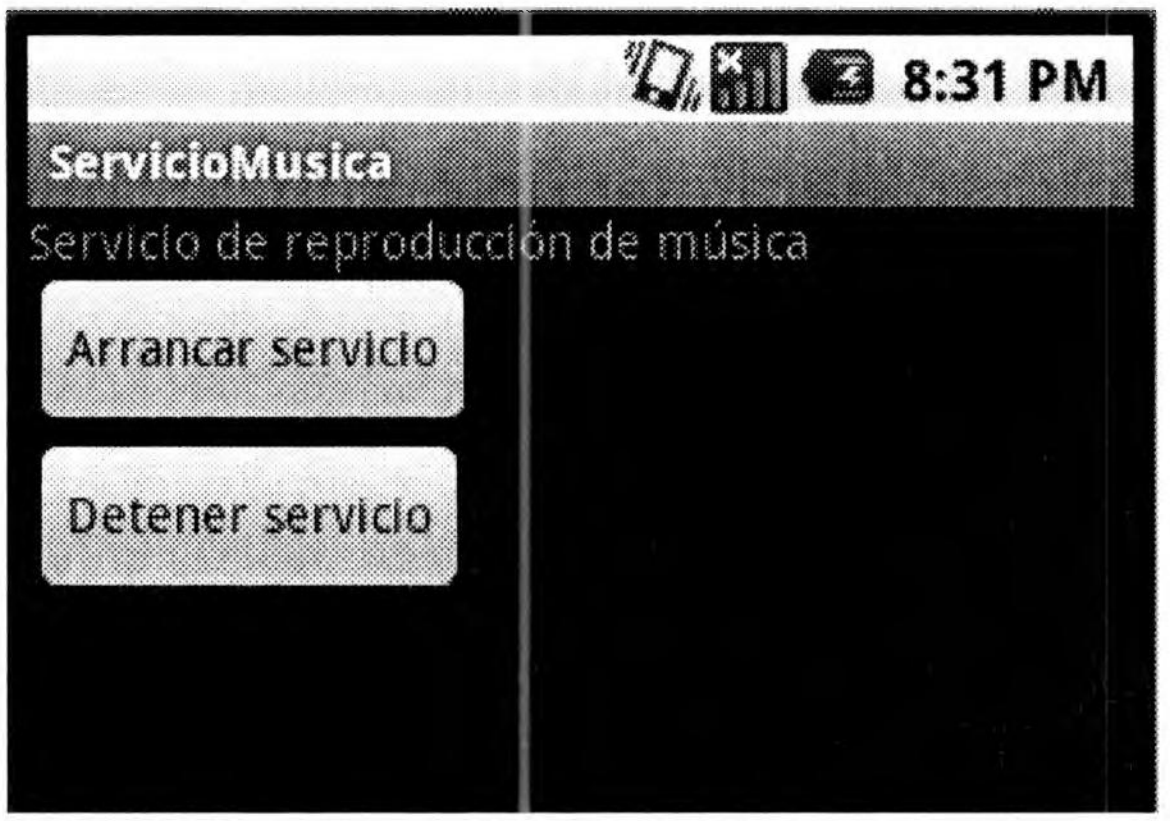
1. Crea un nuevo proyecto con los siguientes datos:

```
Project name: ServicioMusica
Build Target: Android 2.0
Application name: Servicio de Música
Package name: org.example.serviciomusica
Create Activity: ActividadPrincipal
Min SDK Version: 5
```

2. Remplaza el código del *layout* `main.xml` por:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent">
   <TextView  android:layout_width="fill_parent"
              android:layout_height="wrap_content"
              android:text="Servicio de reproducción de
música"/>
   <Button android:id="@+id/boton_arrancar"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content"
              android:text="Arrancar servicio"/>
   <Button android:id="@+id/boton_detener"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content"
              android:text="Detener servicio"/>
</LinearLayout>
```

3. Como puedes ver se trata de un *layout* muy sencillo, con un texto y dos botones:




4. Remplaza el código de la actividad por:

```
public class ActividadPrincipal extends Activity {
  @Override
  public void onCreate(Bundle savedInstanceState) {
     super.onCreate(savedInstanceState);
     setContentView(R.layout.main);

     Button arrancar = (Button) findViewById(R.id.boton_arrancar);
     arrancar.setOnClickListener(new OnClickListener() {
        public void onClick(View view) {
```

```
            startService(new Intent(ActividadPrincipal.this,
                  ServicioMusica.class));
        }
    });
    Button detener = (Button) findViewById(R.id.boton_detener);
    detener.setOnClickListener(new OnClickListener() {
        public void onClick(View view) {
            stopService(new Intent(ActividadPrincipal.this,
                  ServicioMusica.class));
        }
    });
  }
}
```

5. Crea la nueva clase, `ServicioMusica`, con el siguiente código:

```
public class ServicioMusica extends Service {
  MediaPlayer reproductor;

  @Override
  public void onCreate() {
      Toast.makeText(this,"Servicio creado",
                                        Toast.LENGTH_SHORT).show();
      reproductor = MediaPlayer.create(this, R.raw.audio);
  }

  @Override
  public int onStartCommand(Intent intencion, int flags, int idArranque) {
      Toast.makeText(this,"Servicio arrancado "+ idArranque,
                                        Toast.LENGTH_SHORT).show();
      reproductor.start();
      return START_STICKY;
  }

  @Override
  public void onDestroy() {
      Toast.makeText(this,"Servicio detenido",
                                        Toast.LENGTH_SHORT).show();
      reproductor.stop();
  }

  @Override
  public IBinder onBind(Intent intencion) {
      return null;
  }
}
```

6. Edita el fichero `AndroidManifest.xml` y añade la siguiente línea dentro de la etiqueta `<application>`.

```
<service android:name=".ServicioMusica" />
```

7. Crea una nueva carpeta con nombre `raw` dentro de la carpeta `res`. Arrastra a su interior el fichero `audio.mp3`.

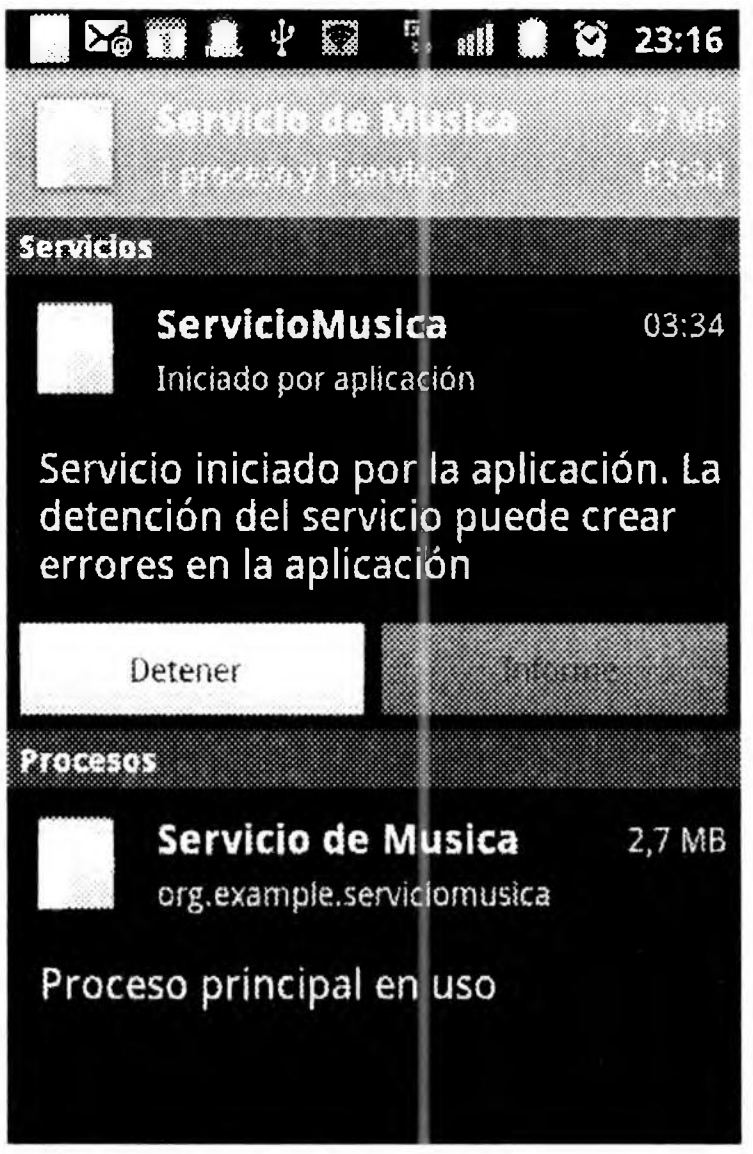
***NOTA:*** *puedes utilizar cualquier fichero de música compatible con Android siempre que el nombre de fichero sea de audio.*

8. Ejecuta la aplicación y comprueba su funcionamiento. Puedes terminar la actividad pulsando el botón para retroceder y verificar que el servicio continúa en marcha.

9. Verifica que aunque pulses varias veces el botón "Arrancar servicio", este no vuelve a crearse, pero sí que vuelve a llamarse al método `onStartCommand()`. Además, con solo una vez que pulses en "Detener servicio" este parará.

10. Arranca la aplicación Ajustes/Aplicaciones/Servicios en ejecución/Servicio de Música. Desde aquí puedes obtener información y detener el servicio.

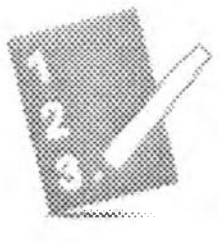
***NOTA:*** *Esta aplicación no está disponible en todas las versiones.*




## 8.2.1. Los métodos onStartCommad() y onStart()

Cuando se crea un servicio hay que tener en cuenta una cuestión de compatibilidad. El método `onStartCommand()` aparece a partir del nivel de API 5, en sustitución de

`onStart()`. Si trabajas con una versión inferior a la 2.0, remplaza este método por el código siguiente:



**Ejercicio paso a paso:** *Los métodos onStartCommand() y onStart().*

1. Comenta en el ejercicio anterior el método `onStartCommand()`
2. Añade el siguiente método:

```
@Override
public void onStart(Intent intent, int startId) {
  Toast.makeText(this, "Servicio arrancado " + startId,
                                        Toast.LENGTH_SHORT).show();
  reproductor.start();
}
```

3. Verifica que el programa funciona exactamente igual.
4. Comenta el método `onStart()` y quita el comentario al método `onStartCommand()`.

Si lo comparamos con `onStartCommand()`, este último tiene un parámetro más y permite devolver un resultado. Veamos con más detalle como sus parámetros pueden ser utilizados para obtener información valiosa:

`public int onStartCommand (Intent intencion, int flags, int idArranque)`

Llamado cada vez que un cliente inicializa un servicio mediante el método `startService()`. Los parámetros se detallan a continuación:

`intencion` Un objeto `Intent` que se indicó en la llamada `startService(Intent)`.

`flags` Información sobre cómo comienza la solicitud. Puede ser 0, `START_FLAG_REDELIVERY` o `START_FLAG_RETRY`. Un valor distinto de 0 se utiliza para reiniciar un servicio tras detectar algún problema.

`idArranque` Un entero único representando la solicitud de arranque específica. Usar este mismo estero en el método `stopSelfResult(int idArranque)`.

En el retorno, describe cómo ha de comportarse el sistema cuando el proceso del servicio "sea matado" una vez que el servicio ya se ha inicializado. Esto puede ocurrir en situaciones de baja memoria. Los siguientes valores están permitidos:

`START_STICKY`: Cuando sea posible el sistema tratará de recrear el servicio, se realizará una llamada a `onStartCommand()` pero con el parámetro `intencion` igual a `null`. Esto tiene sentido cuando el servicio puede arrancar sin información adicional, como por ejemplo, el servicio mostrado para la reproducción de música de fondo.

`START_NOT_STICKY`: El sistema no tratará de volver a crear el servicio, por lo tanto el parámetro `intencion` nunca podrá ser igual a `null`. Esto tiene sentido cuando el servicio no puede reanudarse una vez interrumpido.

`START_REDELIVER_INTENT`: El sistema tratará de volver a crear el servicio. El parámetro `intencion` será el que se utilizó en la última llamada `startService(Intent)`.

`START_STICKY_COMPATIBILITY`: Versión compatible de `START_STICKY`, que no garantiza que `onStartCommand()` sea llamado después de que el proceso "sea matado".

## 8.3. Las notificaciones de la barra de estado

**Poli[Media]**: *Notificaciones en Android.*

La barra de estado de Android se encuentra situada en la parte superior de la pantalla. La parte izquierda de esta barra está reservada para visualizar notificaciones. Cuando se crea una nueva notificación, aparece un texto desplazándose en la barra y, a continuación, un pequeño icono permanecerá en la barra para recordar al usuario la notificación.


El usuario puede arrastrar la barra de notificaciones hacia abajo, para mostrar el listado de las notificaciones por leer. Un posible ejemplo se muestra a continuación:


Una notificación puede ser creada por un servicio o por una actividad. Aunque dado que la actividad dispone de su propio interfaz de usuario, parece que las notificaciones son el mecanismo de interacción más interesante del que disponen

los servicios. Las notificaciones pueden crearse desde un segundo plano, sin interferir con la actividad que en ese momento esté utilizando el usuario.

**Ejercicio paso a paso:** *Las notificaciones de la barra de estado.*

1. Abre el proyecto *ServicioMusica*.
2. Declara las siguientes variables al comienzo de la clase:

```
private NotificationManager nm;
private static final int ID_NOTIFICACION_CREAR = 1;
```

3. Vamos a obtener una referencia al `NotificationManager` del sistema. Para ello, declara las siguientes variables y añade al método `onCreate()` la siguiente línea:

```
nm = (NotificationManager)getSystemService(NOTIFICATION_SERVICE);
```

4. Vamos a crear una nueva notificación. Añade la siguiente línea en el método `onStartCommand()`:

```
notificacion = new Notification(
        R.drawable.icon,
        "Creando Servicio de Música",
        System.currentTimeMillis() );
```

Como puedes ver, en el constructor de una notificación hay que indicar 3 parámetros: el icono a visualizar (en el ejemplo usamos el mismo que el de la aplicación. *En MOTODEV* se llama `ic_laucher`), el texto a mostrar y cuándo queremos que se visualice (en el ejemplo indicamos que ahora mismo).

5. Define información adicional que será utilizada en la ventana de notificaciones. Esta información incluye el mensaje expandido y la actividad a ejecutar cuando se pulse sobre la notificación:

```
PendingIntent intencionPendiente = PendingIntent.getActivity(
        this, 0, new Intent(this, ActividadPrincipal.class), 0);
notificacion.setLatestEventInfo(this, "Reproduciendo música",
        "información adicional", intencionPendiente);
```

Cuando el usuario abra la ventana de notificaciones, podrá ver información adicional formada por un título y un texto explicativo. Además se podrá asociar una actividad para que se ejecute cuando el usuario pulse sobre la notificación. En el ejemplo se crea un `PendingIntent` asociado a la actividad `ActividadPrincipal`. Por supuesto, también puedes crear una nueva actividad para usarla exclusivamente con este fin. En un ejemplo más complejo, puedes pasar los parámetros adecuados a través del `Intent`, para que la actividad conozca los detalles específicos que provocaron la notificación (por ejemplo, el número de teléfono que provocó la llamada perdida).

Una notificación puede tener otros parámetros, por ejemplo, puede reproducir un sonido, puede hacer vibrar el teléfono o puede hacer parpadear un LED del teléfono. Puedes consultar el siguiente punto si estás interesado en alguno de estos aspectos.

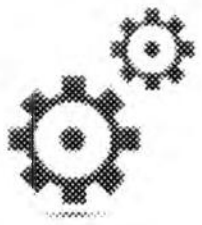
6. Vamos a pasar la notificación creada al `NotificationManager`. Para ello añade:

```
nm.notify(ID_NOTIFICACION_CREAR, notificacion);
... // resto de código de onCreate()
```

7. Queremos que si el servicio deja de estar activo, se elimine la notificación. Para ello añade en `onDestroy()`:

```
nm.cancel(ID_NOTIFICACION_CREAR);
```

Este paso es opcional. Muchas notificaciones han de permanecer visibles aunque el servicio que las creó sea destruido. En nuestro caso, dado que estamos anunciando que un servicio de reproducción de música está activado, la notificación deja de tener sentido al desaparecer el servicio.



**Práctica:** *Uso del servicio de música en Asteroides.*

1. Copia la clase `ServicioMusica` realizada en el ejercicio anterior al proyecto Asteroides.
2. Corrige los errores que hayan aparecido para adaptarla al nuevo proyecto.

***NOTA:*** *Mantén la versión del SDK de Asteroides en 1.6.*

3. En el ejercicio anterior, cuando se visualizan los detalles de la notificación se podía lanzar la actividad `ActividadPrincipal`. Modifica el código para que se lance la actividad `Asteroides`.
4. Si realizas el punto anterior simplemente lanzando la actividad `Asteroides`, cuando el usuario pulse sobre la notificación el sistema lanzará una nueva tarea, aunque ya exista una previa. Si te interesa que no se lance una nueva tarea cuando ya exista una previa, añade la línea en negrita en *AndroidManifest.xml*.

```
<activity
    android:name=".Asteroides"
    android:label="@string/app_name"
    android:launchMode="singleTask">
```

5. Lanza el servicio en el método `onCreate()` de la actividad `Asteroides`. Para el servicio en el método `onDestroy()`.

## 8.3.1. Configurando tipos de avisos en las notificaciones

Como hemos comentado, una notificación puede utilizar diferentes métodos para alertar al usuario de que se ha producido. Veamos algunas opciones.

### *8.3.1.1. Asociar un sonido*

Si consideras que una notificación es muy urgente y deseas que el usuario pueda conocerla de forma inmediata, puedes asociarle un sonido que será reproducido cuando aquélla se produzca.

Si quieres asociar un sonido de notificaciones por defecto, utiliza la siguiente sentencia:

```
notificacion.defaults |= Notification.DEFAULT_SOUND;
```

Si prefieres reproducir un sonido personalizado para la notificación, puedes almacenarlo en una carpeta y usar:

```
notificacion.sound = Uri.parse("file:///sdcard/carpeta/tono.mp3");
```

Si el fichero de audio se encuentra almacenado en el ContentProvider MediaStore, puedes utilizar la siguiente sentencia:

```
notificacion.sound =
    Uri.withAppendedPath(Audio.Media.INTERNAL_CONTENT_URI, "6");
```

Tendrás que sustituir el parámetro `"6"` por el ID del elemento que quieras reproducir. Si desconoces este ID, puedes realizar una consulta al *ContentProvider*. Para más información consulta el apartado 9.7.

### *8.3.1.2. Añadiendo vibración*

También es posible alertar al usuario haciendo vibrar el teléfono.

Puedes utilizar la vibración por defecto:

```
notificacion.defaults |= Notification.DEFAULT_VIBRATE;
```

O, por el contrario, tu propio patrón de de vibración:

```
long[] vibrate = {0,100,200,300};
notificacion.vibrate = vibrate;
```

El *array* define un patrón de longitudes expresadas en milisegundos, donde el primer valor es el tiempo sin vibrar, el segundo es el tiempo vibrado, el tercero sin vibrar y así sucesivamente. Este *array* puede ser tan largo como queramos, pero solo será activado una vez, no se repetirá de forma cíclica.

### *8.3.1.3.* Añadiendo parpadeo de LED

Algunos móviles disponen de diodos LED que pueden ser utilizados para avisar al usuario que se ha producido una notificación. Este método es muy interesante si el grado de urgencia del aviso no es lo suficientemente alto para usar uno de los métodos anteriores.

Podemos utilizar el aviso de LED configurado por defecto:

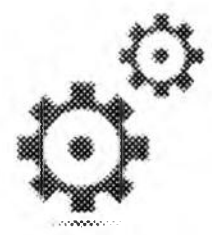
```
notificacion.defaults |= Notification.DEFAULT_LIGHTS;
```

O podemos definir una cadencia de tiempo y color específica para nuestra notificación:

```
notificacion.ledARGB = 0xff00ff00;
notificacion.ledOnMS = 300;
notificacion.ledOffMS = 1000;
notificacion.flags |= Notification.FLAG_SHOW_LIGHTS;
```

En el ejemplo anterior se empieza indicando que queremos que el LED se ilumine en color verde durante 300 ms y luego esté apagado durante 1 segundo. Esta secuencia se repetirá de forma cíclica hasta que el usuario atienda la notificación.

Conviene destacar que no todos los móviles disponen de un LED para este propósito. Además, no todos los colores pueden ser utilizados, siendo el color verde el más habitual para indicar una notificación.



**Práctica:** *Una notificación de socorro.*

1. En el proyecto anterior, crea un nuevo botón.
2. Al pulsar este botón se lanzará una nueva notificación que será mostrada 5 segundos más tarde.
3. Esta notificación mostrará el texto "¡SOCORRO!"
4. El audio de la notificación será una grabación de voz que diga "¡SOCORRO!"
5. La notificación hará vibrar el teléfono con el mensaje internacional de socorro SOS codificado en Morse. Para conseguir esto haz vibrar el teléfono con una sucesión de tres pulsos cortos, tres largos y otros tres cortos (. . . - - - . . .).

## 8.4. Receptores de anuncios

Un receptor de anuncios (*BroadcastReceiver*) recibe y reacciona ante anuncios globales de tipo *broadcast,* aunque las aplicaciones también pueden lanzar un *anuncio broadcast*.. Existen muchos originados por el sistema; como por ejemplo *Batería baja, llamada entrante,...* (se muestra una tabla más adelante). Los receptores de anuncios no tienen interfaz de usuario, aunque pueden iniciar una actividad o crear una notificación para informar al usuario.

El ciclo de vida de un *BroadcastReceiver* es muy sencillo pues solo dispone del método `onReceive()`. De hecho un objeto *BroadcastReceiver* sólo existe durante la llamada a `onReceive()`. El sistema crea el *BroadcastReceiver,* llama a este método y cuando termina destruye el objeto. Un detalle importante es que no hace falta tener en marcha la aplicación donde se define el *BroadcastReceiver* para que este se active.

El método `onReceive()` es ejecutado por el hilo principal de la aplicación; por lo tanto no puede bloquear al sistema (ver ciclo de vida de una actividad). Si tienes que realizar una acción que puede bloquear el sistema tendrás que lanzar un hilo secundario. Si queremos una acción persistente en el tiempo resulta muy frecuente lanzar un servicio. Desde un *BroadcastReceiver* no puedes mostrar un cuadro de diálogo o unirte a un servicio (`bindService()`). Para lo primero, en su lugar puedes lanzar una notificación. Para lo segundo, puedes utilizar `startService()` para arrancar un servicio.

Una aplicación puede registrar un receptor de anuncios de dos maneras: en *AndroidManifest.xml* y en tiempo de ejecución mediante el método `registerReceiver()`.

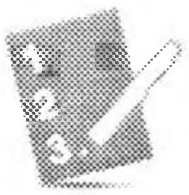
## 8.4.1. Receptor de anuncios registrado en *AndroidManifest.xml*

Registrar un receptor de anuncios desde *AndroidManifest.xml* es muy sencillo. No tienes más que introducir las siguientes líneas en *AndroidManifest.xml* dentro de la etiqueta `<aplication>`:

```
<receiver android:name=".ReceptorAnuncio" >
    <intent-filter>
        <action android:name="android.intent.BATTERY_LOW" />
    </intent-filter>
</receiver>
```

En segundo lugar tienes que crear la clase `ReceptorAnuncio`. El método `onReceive()` será llamado cuando el sistema lance el anuncio *broadcast* `BATTERY_LOW`, esto ocurrirá cuando detecte un nivel bajo de batería.

```
public class ReceptorAnuncio extends BroadcastReceiver {

    @Override
    public void onReceive(Context context, Intent intent) {
        //...
    }
}
```



**Ejercicio paso a paso:** *Un receptor de anuncios.*

Sigue las instrucciones de la sección del libro: "8.3. Las notificaciones de la barra de estado".

1. Crea un nuevo proyecto con los siguientes datos:

```
Project name: LlamadaEntrante
Build Target: Android 1.6
Application name: Llamada Entrante
Package name: org.example.llamadaentrante
Create Activity: LlamadaEntranteActivity
```

```
Min SDK Version: 4
```

2. Edita *AndroidManifest.xml* y añade dentro de la etiqueta `<aplication>` las siguientes líneas:

```
<receiver android:name="ReceptorLlamada">
      <intent-filter>
            <action android:name="android.intent.
                                        action.PHONE_STATE" />
      </intent-filter>
</receiver>
```

   De esta forma registramos un receptor de anuncios que se activará cuando se produzca una llamada

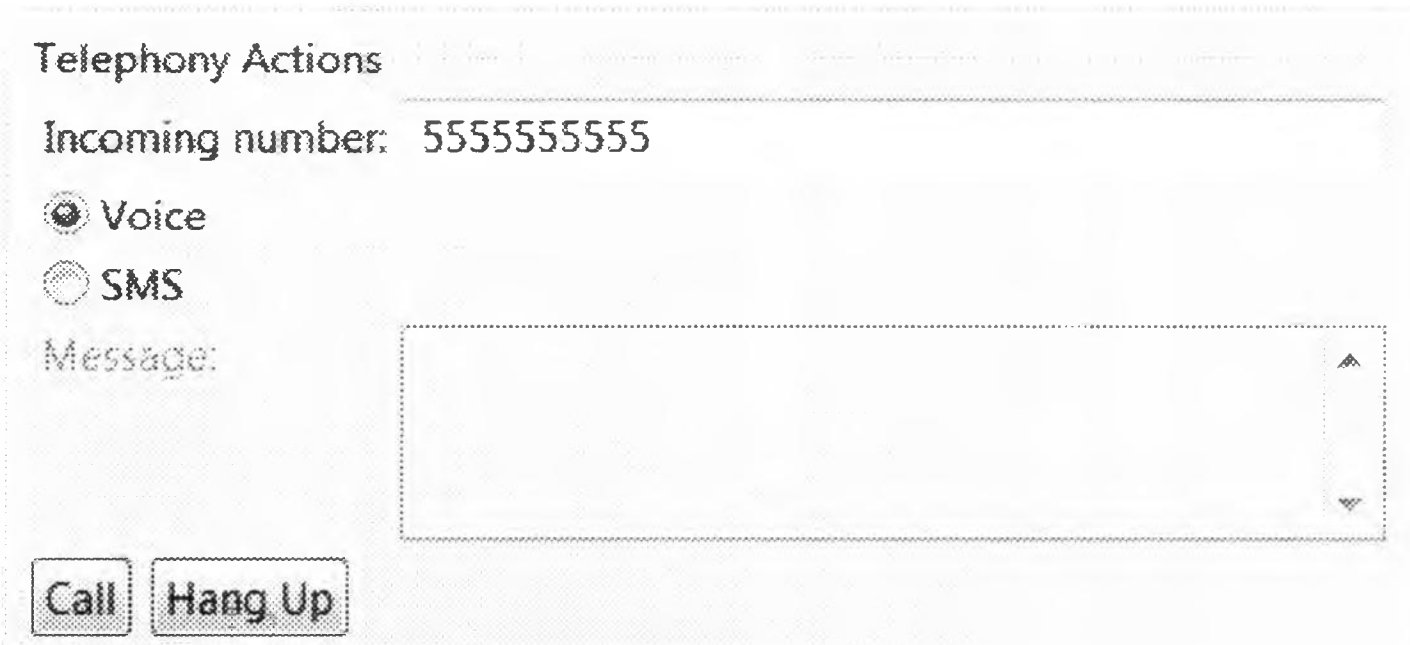
3. Tenemos que pedir permiso para leer el estado del teléfono. Añade la siguiente línea dentro de `<manifest>`.

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

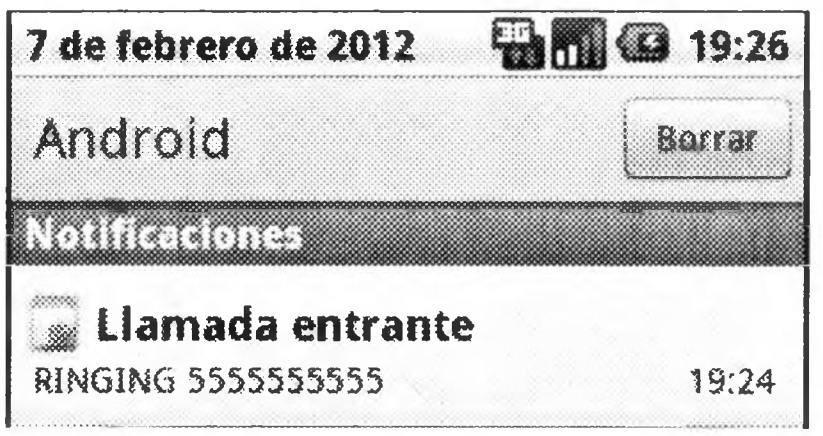
4. Crea una nueva clase `ReceptorLlamadas` y el siguiente código:

```
public class ReceptorLlamadas extends BroadcastReceiver {

  @Override
  public void onReceive(Context context, Intent intent) {
     // Sacamos información del intent
     String estado = "", numero = "";
     Bundle extras = intent.getExtras();
     if (extras != null) {
        estado = extras.getString(TelephonyManager.EXTRA_STATE);
        if (estado.equals(TelephonyManager.EXTRA_STATE_RINGING)) {
           numero = extras.getString(TelephonyManager.
                                        EXTRA_INCOMING_NUMBER);
        }
     }
     String info = estado + " " + numero;
     Log.d("ReceptorAnuncio", info + " intent=" + intent);
     // Creamos Notificación
     NotificationManager nm = (NotificationManager)
context.getSystemService(Context.NOTIFICATION_SERVICE);
     Notification notificacion = new Notification(R.drawable.ic_launcher,
                "Llamada entrante", System.currentTimeMillis());
     PendingIntent intencionPendiente = PendingIntent.getActivity(context,
        0, new Intent(context, LlamadaEntranteActivity.class), 0);
     notificacion.setLatestEventInfo(context, "Llamada entrante",
                                        info, intencionPendiente);
     nm.notify(1, notificacion);
  }
}
```

5. Ejecuta la aplicación e introduce una llamada. Si utilizas el emulador puedes utilizar la vista *EmulatorControl*.




6. Verifica que se crea la notificación.
7. Abre la notificación y verifica la información mostrada:




8. Con el administrador de tareas detén la aplicación. Verifica que el receptor de anuncios funciona igual aunque la aplicación no esté en marcha.

**Recursos adicionales:** *Algunos anuncios broadcast*

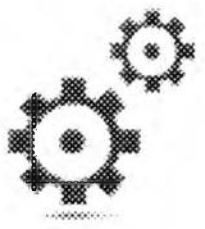
Lista de los anuncios *broadcast* más importantes organizados por temas. (*No Manifest)*: no se puede declarar el receptor de anuncios en *AndroidManifest.xml*. Solo se puede utilizar `registerReceiver()`. *(Solo sistema)*: intención protegida que solo puede ser lanzada por el sistema.

| *Nombre de la acción /(*CONSTANTE*)* | **Descripción/Permiso (**INFORMACIÓN EXTRA EN INTENT**)** |
|---|---|
| ***Batería*** | |
| *android.intent.action.BATTERY_LOW (*ACTION BATTERY LOW*)* | **Batería baja** *(Solo sistema).* |

| | |
|---|---|
| android.intent.action.BATTERY_OKAY (ACTION_BATTERY_OKAY) | **Batería correcta después de haber estado baja** *(Solo sistema).* |
| android.intent.action.ACTION_POWER_CONNECTED (ACTION_POWER_CONNECTED) | **La alimentación se ha conectado** *(Solo sistema).* |
| android.intent.action.ACTION_POWER_DISCONNECTED (ACTION_POWER_DISCONNECTED) | **La alimentación se ha desconectado** *(Solo sistema).* |
| android.intent.action.BATTERY_CHANGED (ACTION_BATTERY_CHANGED) | **Cambia el estado de la batería** (*No Manifest*) *(Solo sistema).* |
| **Sistema** | |
| *android.intent.action.BOOT_COMPLETED* (ACTION_BOOT_COMPLETED) | **Sistema operativo cargado. Permiso** RECEIVE_BOOT_COMPLETED *(Solo sistema).* |
| android.intent.action.ACTION_SHUTDOWN (ACTION_SHUTDOWN) | **El dispositivo va a ser desconectado** *(Solo sistema).* |
| *android.intent.action.AIRPLANE_MODE* (ACTION_AIRPLANE_MODE_CHANGED) | **Modo vuelo activo** *(Solo sistema).* |
| android.intent.action.TIME_TICK (ACTION_TIME_TICK) | **Se envía cada minuto.** (*No Manifest*) *(Solo sistema).* |
| android.intent.action.TIME_SET (ACTION_TIME_CHANGED) | **La fecha/hora es modificada** *(Solo sistema).* |
| android.intent.action.CONFIGURATION_CHANGED (ACTION_CONFIGURATION_CHANGED) | **Cambia la configuración del dispositivo (orientación, idioma,..)** (*No Manifest*) *(Solo sistema).* |
| ***Entradas y pantalla*** | |
| *android.intent.action.SCREEN_OFF* (ACTION_SCREEN_OFF) | **La pantalla se apaga** *(Solo sistema).* |
| *android.intent.action.SCREEN_ON* (ACTION_SCREEN_ON) | **La pantalla se enciende** *(Solo sistema).* |
| android.intent.action.CAMERA_BUTTON (ACTION_CAMERA_BUTTON) | **Se pulsa el botón de la cámara** (EXTRA_KEY_EVENT) |
| android.intent.action.HEADSET_PLUG (ACTION_HEADSET_PLUG) | **Se conectan los auriculares (extras: *state*, *name*, *microphone*)** |

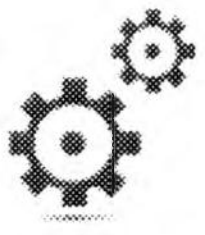
| | |
|---|---|
| android.intent.action.INPUT_METHOD_CHANGED (ACTION_INPUT_METHOD_CHANGED) | **Cambia método de entrada.** |
| android.intent.action.USER_PRESENT (ACTION_USER_PRESENT) | **El usuario está presente después de que se active el dispositivo** *(Solo sistema).* |
| **Memoria y Escáner Multimedia** | |
| android.intent.action.DEVICE_STORAGE_LOW (ACTION_DEVICE_STORAGE_LOW) | **Queda poca memoria** *(Solo Sistema) (Solo sistema).* |
| android.intent.action.DEVICE_STORAGE_OK (ACTION_DEVICE_STORAGE_OK) | **Salimos de la condición de poca memoria** *(Solo Sistema) (Solo sistema).* |
| android.intent.action.MEDIA_EJECT (ACTION_MEDIA_EJECT) | **El usuario pide extraer almacenamiento exterior.** |
| android.intent.action.MEDIA_MOUNTED (ACTION_MEDIA_MOUNTED) | **Almacenamiento exterior disponible.** |
| android.intent.action.MEDIA_REMOVED (ACTION_MEDIA_REMOVED) | **Almacenamiento exterior no disponible.** |
| android.intent.action.MEDIA_SCANNER_FINISHED (ACTION_MEDIA_SCANNER_FINISHED) | **El escáner de medios termina un directorio (se indica en Intent.mData).** |
| android.intent.action.MEDIA_SCANNER_SCAN_FILE (ACTION_MEDIA_SCANNER_SCAN_FILE) | **El escáner de medios encuentra un fichero (se indica en Intent.mData).** |
| android.intent.action.MEDIA_SCANNER_STARTED (ACTION_MEDIA_SCANNER_STARTED) | **El escáner de medios comienza un directorio (se indica en Intent.mData).** |
| **Aplicaciones** | |
| android.intent.action.MY_PACKAGE_REPLACED (ACTION_MY_PACKAGE_REPLACED) | **Una nueva version de tu aplicación ha sido instalada** *(Solo sistema).* |
| android.intent.action.PACKAGE_ADDED (ACTION_PACKAGE_ADDED) | **Una nueva aplicación instalado (**EXTRA_UID, EXTRA_REPLACING**)** *(Solo sistema).* |
| android.intent.action.PACKAGE_FIRST_LAUNCH (ACTION_PACKAGE_FIRST_LAUNCH) | **Primera vez que se lanza una aplicación** *(Solo sistema).* |

| | |
|---|---|
| android.intent.action.PACKAGE_REMOVED (ACTION_PACKAGE_REMOVED) | **Se desinstala una aplicación** *(Solo sistema).* |
| **Comunicaciones y redes** | |
| *android.intent.action.PHONE_STATE* | **Llamada de teléfono.** Permiso: READ_PHONE_STATE (EXTRA_STATE, EXTRA_STATE_RINGING). |
| android.intent.action.NEW_OUTGOING_CALL (ACTION_NEW_OUTGOING_CALL) | **Se va a hacer una llamada.** Permiso PROCESS_OUTGOING_CALLS (*EXTRA_PHONE_NUMBER*) *(Solo sistema).* |
| *android.provider.Telephony.SMS_RECEIVED* | **SMS recibido**. Permiso: RECEIVE_SMS. |
| *android.bluetooth.intent.action.DISCOVERY_STARTED* | **Comienza escáner Bluetooth.** |
| *android.bluetooth.intent.action.ENABLED* | **Bluetooth habilitado.** |
| *android.net.wifi.NETWORK_IDS_CHANGED* (NETWORK_IDS_CHANGED_ACTION) | **Cambia la red WiFi.** |
| *android.net.wifi.STATE_CHANGE* (NETWORK_STATE_CHANGED_ACTION) | **Cambia la conectividad WiFi** (EXTRA_NETWORK_INFO, EXTRA_BSSID, EXTRA_WIFI_INFO). |
| android.net.wifi.RSSI_CHANGED (RSSI_CHANGED_ACTION) | **Cambia el nivel de señal WiFi** (EXTRA_NEW_RSSI). |



**Práctica:** *Arranque de una actividad desde un receptor de anuncio.*

Modifica el proyecto *Asteroides* para que arranque automáticamente la actividad AcercaDe al llegar un SMS cualquiera.



**Práctica:** *Arranque de una tarea desde un receptor de anuncio.*

Modifica el proyecto *Asteroides* para que arranque automáticamente la actividad principal Asteroides al llegar un SMS cualquiera.

***NOTA:*** *Si utilizas el código anterior modificando* `AcercaDe.class` *por* `Asteroides.class` *comprobarás que no funciona. Para realizar esta práctica has de tener en cuenta que Android trata de forma diferente a las actividades principales de una tarea (marcada en* `<action>` *como* `MAIN`*). Si intentas arrancar una actividad de este tipo, Android considera que estás tratando de arrancar una nueva tarea. El problema surge porque para lanzar una tarea desde una intención resulta imprescindible activar el flag* `FLAG_ACTIVITY_NEW_TASK`*. Para hacer esto, siendo* `i` *de la clase* `Intent`*, utiliza el siguiente código:*

```
i.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
```

Cuando se utiliza este *flag*, si una tarea ya se está ejecutando no se lanza una nueva actividad; el *flag* se pondrá al frente de la tarea, mostrándose la última vista activa de ésta. Si no nos interesa este comportamiento podemos utilizar, además, el *flag* `FLAG_ACTIVITY_MULTIPLE_TASK` para permitir lanzar varias tareas

## 8.4.2. Arrancar un servicio tras cargar el sistema operativo

En muchas ocasiones puede ser interesante que un servicio de nuestra aplicación esté siempre activo, incluso aunque el usuario no haya arrancado nuestra aplicación. Imagina, por ejemplo, un servicio de mensajería instantánea que ha de estar siempre atento a la llegada de mensajes. Conseguirlo es muy fácil, no tenemos más que crear un receptor de anuncios que se active ante el anuncio `android.intent.action.BOOT_COMPLETED`. Desde este receptor podremos crear el servicio. Para poder registrar el receptor es obligatorio solicitar el permiso `RECEIVE_BOOT_COMPLETED`. Veamos los tres pasos a seguir:

1. Creamos un recetor de anuncios:

```
public class ReceptorArranque extends BroadcastReceiver {
   @Override
   public void onReceive(Context context, Intent intent) {
        context.startService(new Intent(context, Servicio.class));
   }
}
```

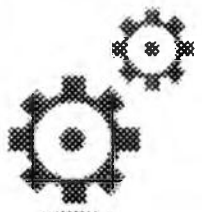
2. Creamos el servicio

```
public class Servicio extends Service {

  @Override
  public void onCreate() {…}

  @Override
  public int onStartCommand(Intent intencion, int flags,
                                                  int idArranque) {…}

  @Override
  public void onDestroy() {…}
```

```
    @Override
    public IBinder onBind(Intent intencion) {
      return null;
    }
}
```

3. En *AndroidManifest.xml* registramos el receptor de anuncios y pedimos el permiso adecuado.

```
...
<aplication>
  ...
   <receiver android:name="ReceptorArranque" >
      <intent-filter >
         <action android:name="android.intent.action.BOOT_COMPLETED" />
      </intent-filter>
   </receiver>
</aplication>
...
<uses-permission android:name="android.permission.
                                        RECEIVE_BOOT_COMPLETED"/>
...
```



**Práctica:** *Arranque automático del Servicio de Música.*

Modifica el proyecto *ServicioMusica* para que el servicio se active desde el arranque del sistema operativo.

## 8.5. Un servicio como mecanismo de comunicación entre aplicaciones

Como hemos comentado, un servicio tiene una doble funcionalidad: permitir la ejecución de código en segundo plano y poder ser utilizado como un mecanismo de comunicación entre aplicaciones[2].

Cuando una aplicación quiere compartir algún tipo de información con otra se presenta un problema. En Android las aplicaciones se ejecutan en procesos separados y, por tanto, tienen espacios de memorias distintos. Esto nos impide, por ejemplo, que ambas aplicaciones compartan un mismo objeto.

[2] http://developer.android.com/guide/topics/fundamentals.html#rpc

Como respuesta a este problema Android nos propone un mecanismo de comunicación entre procesos, que se basa en un lenguaje de especificación de interfaces, AIDL, que son publicados por medio de servicios.

AIDL (*Android Interface Definition Language*) es un lenguaje de especificación de interfaces que permite que un proceso en Android pueda llamar a un método de un objeto situado en un proceso diferente al suyo. Se trata de un mecanismo de comunicación entre procesos similar a COM o Corba, aunque algo más ligero.

Si queremos comunicar dos aplicaciones a través de este mecanismo seguiremos los siguientes pasos:

1) Escribiremos un fichero AIDL: en él se define la interfaz, es decir los métodos y los parámetros que luego podremos utilizar.
2) Implementaremos los métodos de la interfaz: para ello, habrá que crear una clase en Java que implemente estos métodos.
3) Publicar la interfaz a los clientes: para ello, se extenderá la clase `Service` y sobrescribiremos el método `onBind(Intent)` de forma que devuelva una instancia de la clase que implementa la interfaz.

**Poli[Media]**: *Un servicio como mecanismo de comunicación entre aplicaciones.*

Veamos estos tres pasos más detenidamente por medio de un ejemplo. Para ello, crea la siguiente aplicación:

```
Project name: ServicioRemoto
Build Target: Android 2.0
Application name: Servicio Remoto
Package name: org.example.serviciocliente
Create Activity: ActividadPrincipal
Min SDK Version: 5
```

Reemplaza el código del *Layout* `main.xml` por el mismo utilizado en `ServicioMusica`. Remplaza los textos de los botones *"Arrancar servicio"* por *"Conectar servicio"* y *"Detener servicio"* por *"Desconectar servicio"*. Crea dos botones más. Uno con texto "*Reproducir*" e id *"@+id/boton_reproducir*" y otro con texto "*Avanzar*" e id *"@+id/boton_avanzar*".

Copia el fichero `res/raw/audio.mp3` a la nueva aplicación.

## 8.5.1. Crear la interfaz en AIDL

El lenguaje de especificación de interfaces AIDL, tiene una sintaxis similar a Java, aunque permite únicamente identificar la interfaz de un objeto, no su implementación.

Una interfaz estará formada por una secuencia de métodos cada uno con una serie de parámetros y un valor devuelto. Tanto los parámetros como el valor devuelto han de tener un tipo. Los tipos permitidos se indican a continuación:

- Tipos primitivos: `int`, `short`, `byte`, `char`, `float`, `double`, `long`, `Boolean`.
- Uno de los siguientes tipos: `String`, `CharSequence`, `List`, `Map`.
- Una interfaz escrita en AIDL.
- Un objeto `Parcelable`.

Para seguir con el ejemplo, crea un nuevo fichero con nombre `IServicioMusica.aidl` dentro de `src/org.example.servicioaidl/` con el siguiente código:

```
package org.example.servicioremoto;

interface IServicioMusica {
    String reproduce(in String mensaje);
    void setPosicion(int ms);
    int getPosicion();
}
```

Como puedes observar la sintaxis es similar a Java aunque existen diferencias. La más destacable consiste en que los parámetros de los métodos cuyos tipos no sean primitivos, han de indicar la etiqueta `in`, `out` o `inout`, según sean parámetros de entrada, salida o de las dos cosas a la vez. Para los tipos primitivos sólo se permite que actúen como entrada, por lo tanto se procesan de forma predeterminada como `in`.

***NOTA:*** *Igual que ocurre con las clases en Java los interfaces en AIDL han de escribirse en fichero con igual nombre que la interfaz.*

## 8.5.2. Implementar la interfaz

Una vez escrita esta interfaz y almacenada en el fichero `.aidl`, el *plug-in* de Eclipse generará de forma automática el fichero `gen/org.example.servicioremoto/IServicioMusica.java`.

Este fichero implementa la interfaz Java `IServicioMusica` como descendiente de `IInterface`. En `IServicioMusica` se define internamente la clase abstracta `Stub` que implementa la interfaz escrita en AIDL. Es decir, la clase `Stub` contiene tantos métodos abstractos como métodos declaramos en la interfaz AIDL. La clase `Stub` también define una serie de métodos que le permiten el intercambio de información cuando se invoquen estos métodos entre procesos remotos.

Ahora tenemos que darle funcionalidad a la interfaz y, para ello, tendremos que crear una clase que extienda `IServicioMusica.Stub` y que implemente todos los métodos abstractos de esta clase, o lo que es lo mismo, los métodos declarados en la interfaz AIDL. A continuación se muestra un ejemplo de cómo podríamos implementar estos métodos. Más tarde se indica dónde hay que introducir este código:

```
private final IServicioMusica.Stub binder = new IServicioMusica.Stub() {

    public String reproduce(String mensaje) {
```

```
        reproductor.start();
        return mensaje;
    }

    public void setPosicion(int ms) {
        reproductor.seekTo(ms);
    }

    public int getPosicion() {
        return reproductor.getCurrentPosition();
    }
}
```

La variable `reproductor` será declarada posteriormente de tipo `MediaPlayer`. Como puedes ver, en el método `reproduce`, tanto el parámetro de entrada como el valor devuelto no tienen ninguna utilidad. Se han introducido para ilustrar el paso de una variable no primitiva.

Cuando implementes los métodos de una interfaz AIDL has de tener en cuenta lo siguiente:

Si generas una excepción desde uno de estos métodos, ésta no pasará a la aplicación que hizo la llamada.

Las llamadas son síncronas. Por lo tanto, has de tener cuidado de que cuando se haga una llamada que tarde cierto tiempo en responder, nunca se realice desde el hilo principal de la aplicación. Si este hilo queda bloqueado demasiado tiempo, aparecerá el incómodo cuadro de diálogo "La aplicación no responde". En estos casos, crea un hilo secundario desde donde se haga la llamada.

Solo es posible declarar métodos; no puedes declarar campos estáticos en una interfaz AIDL

### 8.5.3. Publicar la interfaz en un servicio

Otras aplicaciones han de tener visible esta interfaz para poder comunicarse con nosotros. Esto se consigue creando un servicio que contenga nuestra interfaz.

Para ello, has de crear una clase que herede de `Service` y que reescriba el método `onBind()`. Este método será utilizado para devolver un objeto que implementa nuestra interfaz. Veamos cómo se escribiría este servicio:

```
public class ServicioRemoto extends Service{

    MediaPlayer reproductor;

    @Override
    public void onCreate() {
        super.onCreate();
        reproductor = MediaPlayer.create(ServicioRemoto.this, R.raw.audio);
```

```
    }

    private final IServicioMusica.Stub binder = new IServicioMusica.Stub(){
        // Copia aquí el código anterior
    };

    @Override
    public IBinder onBind(Intent intent) {
        return this.binder;
    }
```

Crea una nueva clase en el proyecto e introduce este código. Para que el servicio sea visible a otras aplicaciones hay que publicarlo declarándolo en `AndroidManifest.xml`. Para ello, copia el siguiente código dentro de la etiqueta `<application>`.

```
<service android:name=".ServicioRemoto"  android:process=":remoto">
    <intent-filter>
        <action android:name="org.example.servicioaremoto.IServicioMusica"/>
    </intent-filter>
</service>
```

El atributo `name` ha de coincidir con la clase que implementa el servicio. El atributo `process` permite que el servicio se ejecute en un proceso propio, diferente al resto de los componentes de la aplicación. A continuación, indicaremos dentro de la etiqueta `<intent-filter>` una etiqueta `<action>` por cada interfaz que queremos publicar. Un servicio podría publicar más de una interfaz, para lo que habría que escribir un fichero AIDL por cada interfaz. Puedes encontrar un ejemplo en la aplicación ApiDemos.

### 8.5.4. Llamar a una interfaz remota

Para llamar a la interfaz creada anteriormente, sigue los siguientes pasos:

1) Declara un objeto de tipo `IServicioMusica`. Es el que podrás utilizar para hacer las llamadas al objeto remoto.
2) Implementa la clase `ServiceConnection`. Nos permitirá controlar cuándo se produce una conexión (`onServiceConnected()`) y cuándo se produce una desconexión del servicio (`onServiceDisconnected()`).
3) Para conectarte al servicio llama al método `bindService()` pasándole un objeto de la clase `ServiceConnection` que acabas de implementar.
4) Si se puede realizar la conexión se llamará al método `ServiceConnection.onServiceConnected()` y recibirás en uno de sus parámetros una instancia de `IBinder`. Utiliza el método `IServicioMusica.Stub.asInterface()` pasándole como parámetro esta instancia de `IBinder` para inicializar el objeto declarado en el primer punto.

5) Ya puedes llamar a los métodos del objeto remoto declarados en el punto 1). En nuestro caso `reproducir()`, `setPosicion()` y `getPosicion()`.
6) Puedes desconectarte del servicio utilizando el método `unbindService()`.

Realizaremos este trabajo en la clase `ActividadPrincipal`. Remplaza su código por el siguiente:

```
public class ActividadPrincipal extends Activity {

    private IServicioMusica servicio;                                    (1)

    private ServiceConnection conexion = new ServiceConnection() {       (2)
        public void onServiceConnected(ComponentName className,          (4)
                                       IBinder iservicio) {
            servicio = IServicioMusica.Stub.asInterface(iservicio);
            Toast.makeText(ActividadPrincipal.this,
                    "Conectado a Servicio", Toast.LENGTH_SHORT).show();
        }
        public void onServiceDisconnected(ComponentName className) {
            servicio = null;
            Toast.makeText(ActividadPrincipal.this,
                        "Se ha perdido la conexión con el Setvicio",
                        Toast.LENGTH_SHORT).show();
        }
    };

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Button botonConectar = (Button) findViewById(R.id.boton_arrancar);
        botonConectar.setOnClickListener(new OnClickListener() {
            public void onClick(View v) {
                ActividadPrincipal.this.bindService(                     (3)
                    new Intent(ActividadPrincipal.this, ServicioRemoto.class),
                    conexion,    Context.BIND_AUTO_CREATE);
            }
        });
        Button botonReproducir=(Button) findViewById(R.id.boton_reproducir);
        botonReproducir.setOnClickListener(new OnClickListener() {
            public void onClick(View v) {
                try {
                    servicio.reproduce("titulo");                        (5)
                } catch (Exception e) {
                    Toast.makeText(ActividadPrincipal.this, e.toString(),
                                                    Toast.LENGTH_SHORT).show();
                }
            }
        });
        Button botonAvanzar = (Button) findViewById(R.id.boton_avanzar);
```

```
        botonAvanzar.setOnClickListener(new OnClickListener() {
            public void onClick(View v) {
                try {
                    servicio.setPosicion(servicio.getPosicion()+1000);  (5)
                } catch (Exception e) {
                    Toast.makeText(ActividadPrincipal.this, e.toString(),
                                                    Toast.LENGTH_SHORT).show();
                }
            }
        });
        Button botonDetener = (Button) findViewById(R.id.boton_detener);
        botonDetener.setOnClickListener(new OnClickListener() {
            public void onClick(View v) {
                try{
                    ActividadPrincipal.this.unbindService(conexion);    (6)
                } catch (Exception e) {
                    Toast.makeText(ActividadPrincipal.this, e.toString(),
                                                    Toast.LENGTH_SHORT).show();
                }
                servicio = null;
            }
        });
    }
}
```

La actividad está formada por 4 botones para realizar las acciones de: conectarse al servicio, desconectarse, reproducir música y avanzar 1 segundo (1.000 ms). Si los botones no se activan por el orden adecuado podemos provocar excepciones. Estas son capturadas y visualizadas mediante un `toast`, por lo que la aplicación continuará funcionando aunque se produzcan. Prueba a llamar a un método antes de establecer la conexión o trata de desconectarte dos veces para observar las excepciones generadas.

# CAPÍTULO 9.
# Almacenamiento de datos

Las aplicaciones descritas hasta este capítulo representaban la información a procesar en forma de variables. El problema de estas variables es que dejan de existir en el momento en que la aplicación es destruida. En muchas ocasiones vamos a necesitar almacenar información de manera permanente. Las alternativas más habituales para conservar esta información son los ficheros, las bases de datos o servicios a través de la red. Estas técnicas no solo permiten mantener a buen recaudo los datos de la aplicación, sino también compartir estos datos con otras aplicaciones y usuarios. De forma adicional, el sistema Android pone a nuestra disposición dos nuevos mecanismos para almacenar datos: las preferencias y *ContentProvider*.

A lo largo de este capítulo estudiaremos cómo utilizar estas técnicas. Comenzaremos describiendo el uso de las preferencias como un mecanismo sencillo para guardar de forma permanente algunas variables. Seguiremos describiendo las características del sistema de ficheros que incorpora Android. Se puede acceder a los ficheros a través de las clases estándar incluidas en Java; de forma adicional se incluyen nuevas clases para cubrir las peculiaridades de Android.

Como tercera alternativa se estudiará el uso de XML para almacenar la información de manera estructurada. Se describirán dos herramientas alternativas, las librerías SAX y DOM. Como cuarta alternativa al almacenamiento de datos se estudiarán las bases de datos. Android incorpora la librería SQLite, que nos permitirá crear y manipular nuestras propias bases de datos de forma muy sencilla. Para finalizar, se describirá la clase `ContentProvider`; consiste en un mecanismo introducido en Android para poder compartir datos entre aplicaciones.

En el capítulo siguiente se describe otra alternativa, el uso de Internet como recurso para almacenar y compartir información. Concretamente, se describirá el uso de *sockets* TCP, HTML y los servicios web.

Todas estas alternativas serán ilustradas a través del mismo ejemplo. Trataremos de almacenar la lista con las mejores puntuaciones obtenidas en Asteroides descrita en el capítulo 3.

Objetivos:

- Repasar las alternativas para el almacenamiento de datos en Android.
- Describir el uso de ficheros.
- Ilustrar la utilización de dos herramientas para manipular ficheros XML, las librerías SAX y DOM.
- Mostrar cómo desde Android podemos utilizar SQLite para trabajar con bases de datos.
- Describir qué es un ContentProvider y cómo podemos utilizar algunos ContentProvider disponibles en Android.
- Aprender a crear nuestros propios ContentProvider.

## 9.1. Alternativas para guardar datos permanentemente en Android

Existen muchas alternativas para almacenar información de forma permanente en un sistema informático. A continuación mostramos una lista de las más habituales utilizadas en Android:

- **Preferencias:** es un mecanismo liviano que permite almacenar y recuperar datos primitivos en la forma de pares clave/valor. Este mecanismo es típicamente usado para almacenar los parámetros de configuración de una aplicación.
- **Ficheros:** puedes almacenar los ficheros en la memoria interna del dispositivo o en un medio de almacenamiento removible como una tarjeta SD. También puedes utilizar ficheros añadidos a tu aplicación como recursos.
- **XML:** se trata de un tipo de archivo que sigue un determinado estándar. Ampliamente utilizado en Internet y en muchos otros sitios (como en el Android SDK). Disponemos de las librerías SAX y DOM para manipular estos ficheros desde Android.
- **Base de datos:** las APIs de Android contienen soporte para SQLite. Tu aplicación puede crear y usar base de datos SQLite de forma muy sencilla y con toda la potencia que nos da el lenguaje SQL.
- **Proveedores de contenidos:** un proveedor de contenidos es un componente opcional de una aplicación que expone el acceso de lectura/escritura de sus datos a otras aplicaciones. Está sujeto a las restricciones de seguridad que quieras imponer. Los proveedores de contenidos implementan una sintaxis estándar para solicitar datos (URIs) y un mecanismo de acceso para devolver los datos (similar a SQL). Android provee algunos suministradores de contenidos para tipos de datos estándar, tales como contactos personales, ficheros multimedia, etc.
- **Internet:** no te olvides que también puedes usar la nube para almacenar y recuperar datos. Se estudia en el siguiente capítulo.

**Poli[Media]**: *Almacenamiento de datos en Android.*

## 9.2. Añadiendo puntuaciones en Asteroides

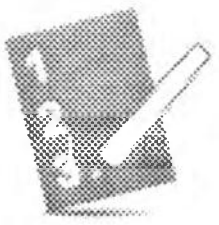
A modo de ejemplo, se va a implementar la posibilidad de guardar las mejores puntuaciones obtenidas en Asteroides. Se utilizarán mecanismos alternativos que serán desarrollados a lo largo de este capítulo y el siguiente:

- Array (implementado en el capítulo 3).
- Preferencias.
- Ficheros en memoria interna.
- Fichero en memoria externa.
- Fichero en recursos.
- XML con SAX.
- XML con DOM.
- Base de datos SQLite.
- Content Provider.
- Sockets.
- Servicios Web.

Para facilitar la sustitución del método de almacenamiento, en el capítulo 3 se creó el siguiente *interface*:

```
public interface AlmacenPuntuaciones {
    public void guardarPuntuacion(int puntos, String nombre,
                                  long fecha);
    public Vector<String> listaPuntuaciones(int cantidad);
}
```

También se declaró la variable `almacen` de tipo `AlmacenPuntuaciones` y se creó la actividad `Puntuaciones` que visualizaba un `ListView` con las puntuaciones. Dado que en el capítulo 3 todavía no teníamos la opción de jugar, no se podían añadir nuevas puntuaciones a `almacen`. En el siguiente ejercicio trataremos de calcular una puntuación en el juego y almacenarla en `almacen`.



**Ejercicio paso a paso:** *Calculando la puntuación en Asteroides.*

1. Crea una variable global en la clase `VistaJuego` con nombre `puntuacion` e inicialízala a cero:

```
private int puntuacion = 0;
```

2. Cada vez que se destruya un asteroide hay que incrementar esta variable. Añade dentro de `destruyeAsteroide()` la siguiente línea:

```
puntuacion += 1000;
```

3. Cuando desde la actividad inicial `Asteroides`, se llame a la actividad `Juego` nos interesa que ésta nos devuelva la puntuación obtenida. Recuerda cómo en el capítulo 3 estudiamos la comunicación entre actividades. Para pasar la información entre las actividades añade en siguiente código en `Asteroides`.

```
public void lanzaJuego(View view) {
   Intent i = new Intent(this, Juego.class);
   startActivityForResult(i, 1234);
}

@Override protected void onActivityResult (int requestCode,
                                    int resultCode, Intent data){
   super.onActivityResult(requestCode, resultCode, data);
   if (requestCode==1234 & resultCode==RESULT_OK & data!=null) {
     int puntuacion = data.getExtras().getInt("puntuacion");
     String nombre = "Yo";
     // Mejor leerlo desde un Dialog o una nueva actividad
     // AlertDialog.Builder
     almacen.guardarPuntuacion(puntuacion, nombre,
                                    System.currentTimeMillis());
     lanzarPuntuaciones();
   }
}
```

4. Para realizar la respuesta de la actividad va a ser más sencillo hacerlo desde `VistaJuego` en lugar que desde `Juego`. El problema es que esta clase es una vista, no una actividad. Para solucionar el problema puedes usar el siguiente truco. Introduce en `VistaJuego` el siguiente código:

```
private Activity padre;

public void setPadre(Activity padre) {
    this.padre = padre;
}
```

5. Cuando se detecte una condición de victoria o derrota es un buen momento para almacenar la puntuación y salir de la actividad. Para ello, crea el siguiente método dentro de `Juego`:

```
private void salir() {
   Bundle bundle = new Bundle();
   bundle.putInt("puntuacion", puntuacion);
   Intent intent = new Intent();
   intent.putExtras(bundle);
   padre.setResult(Activity.RESULT_OK, intent);
```

```
        padre.finish();
    }
```

6. Al final del método `destruyeAsteroide()` introduce:

```
if (Asteroides.isEmpty()) {
    salir();
}
```

7. Al final del método `actualizaFisica()` introduce:

```
for (Grafico asteroide : Asteroides) {
    if (asteroide.verificaColision(nave)) {
        salir();
    }
}
```

8. En el método `onCreate` de `Juego` introduce:

```
vistaJuego.setPadre(this);
```

## 9.3. Preferencias

Las preferencias (clase `SharedPreferences`) pueden ser usadas como un mecanismo para que los usuarios modifiquen algunos parámetos de configuración de la aplicación. Este uso fue estudiado en el capítulo 3, donde se describe cómo podíamos crear una actividad dependiente de `PreferenceActivity` para que el usuario consultara y modificara estas preferencias.

Las preferencias también pueden ser utilizadas como un mecanismo liviano para almacenar ciertos datos que tu aplicación quiera conservar de forma permanente. Es un mecanismo sencillo que te permite almacenar una serie de variables con su nombre y su valor. Puedes almacenar variables de tipo booleano, real, entero y `String`. En este apartado describimos su utilización.

Las preferencias son almacenadas en ficheros XML dentro de la carpeta `shared_prefs` en los datos de la aplicación. Recuerda que en el capítulo 3 vimos cómo las preferencias de usuario siempre se almacenaban en el fichero *`paquete`*`_preferences` , donde el paquete ha de ser reemplazado por el paquete de la aplicación (en Asteroides el fichero es `org.example.asteroides_preferences`). Cuando utilices las preferencias para almacenar otros valores podrás utilizar otros ficheros. Tienes dos alternativas según utilices uno de los siguientes métodos:

- `getSharedPreferences()` Te permite indicar de forma explícita el nombre de un fichero de preferencias. Puedes utilizarlo cuando necesites varios ficheros de preferencias o acceder al mismo fichero desde varias actividades.
- `getPreferences()` No tienes que indicar ningún nombre. Puedes utilizarlo cuando solo necesites un fichero de preferencias en la actividad donde se usa.

Estos dos métodos necesitan como parámetro el tipo de permiso que queramos dar al fichero de preferencias. Los valores posibles son `MODE_PRIVATE`, `MODE_WORLD_READABLE` o `MODE_WORLD_WRITEABLE` según queramos tener acceso exclusive, permitir la lectura o permitir lectura y escritura.

Una llamada a uno de estos dos métodos te devolverá un objeto de la clase `SharedPreferences`.

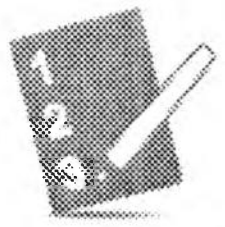
Para escribir las preferencias puedes utilizar el siguiente código:

```
SharedPreferences preferencias= gePreferences(MODE_PRIVATE);
SharedPreferences.Editor editor = preferencias.edit();
editor.putString("nombre", "Juan");
editor.putInt("edad", 35);
editor.commit();
```

Para leer las preferencias puedes utilizar el siguiente código:

```
SharedPreferences preferencias = gePreferences(MODE_PRIVATE);
String nombre = preferencias.getString("nombre",
                                       "valor por defecto");
int edad = preferencias.getInt("edad", -1);
```

El ejemplo anterior puede ser modificado reemplazando `getPreferences()` por `getSharedPreferences()`. En este caso, tendrás que indicar el fichero donde se almacenarán las preferencias.



**Ejercicio paso a paso:** *Almacenando la última puntuación en un fichero de preferencias.*

Veamos un ejemplo de cómo podemos crear un fichero de preferencias (*MiFicheroDePreferencias.xml*) para almacenar la última puntuación obtenida en Asteroides.

1. Abre el proyecto *Asteroides*.
2. Crea una nueva clase `AlmacenPuntuacionesPreferencias`.
3. Remplaza el código por el siguiente:

```
public class AlmacenPuntuacionesPreferencias implements
AlmacenPuntuaciones {
  private static String PREFERENCIAS = "puntuaciones";
  private Context context;

  public AlmacenPuntuacionesPreferencias(Context context) {
    this.context = context;
  }

  public void guardarPuntuacion(int puntos, String nombre,
                                long fecha) {
```

```
    SharedPreferences preferencias =context.getSharedPreferences(
                        PREFERENCIAS, Context.MODE_PRIVATE);
    SharedPreferences.Editor editor = preferencias.edit();
    editor.putString("puntuacion", puntos + " " + nombre);
    editor.commit();
  }

  public Vector<String> listaPuntuaciones(int cantidad) {
    Vector<String> result = new Vector<String>();
    SharedPreferences preferencias =context.getSharedPreferences(
                                    PREFERENCIAS, Context.MODE_PRIVATE);
        String s = preferencias.getString("puntuacion", "");
        if (s != "") {
            result.add(s);
        }
        return result;
    }
  }
```

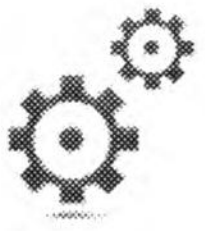
4. Abre el fichero *Asteroides.java* y en el método `onCreate()` reemplaza la línea adecuada por:

```
almacen = new AlmacenPuntuacionesPreferencias(this);
```

5. Ejecuta el proyecto y verifica que la última puntuación se guarda correctamente.

6. Abre la vista File Explorer: utilizando el menú *Window/Show View/Other.../Android/File Explorer* y verifica que se ha creado el fichero: */data/data/org.example.asteroides/shared_prefs/puntuaciones.xml*

7. Descarga este fichero a tu ordenador (botón ) y observa su contenido.



**Práctica:** *Almacenando las últimas 10 puntuación en un fichero de preferencias.*

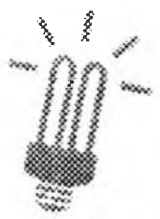
En el ejercicio anterior solo guardamos la última puntuación, lo cual no coincide con la idea que habíamos plantado en un principio –nos interesaba guardar una lista con las últimas puntuaciones–. En esta práctica tratarás de solucionar este inconveniente.

***NOTA:*** *Básicamente es una práctica de programación en Java. Si no estás interesado puedes consultar directamente la solución.*

1. Las preferencias solo están preparadas para almacenar variables de tipo simple, por lo que no permiten almacenar un vector. Para solucionar este inconveniente, te recomendamos que crees 10 preferencias con nombres `puntuacion0`, `puntuacion1`, ... , `puntuacion9`.

2. Cuando se llame a `guardarPuntuacion()` almacena la nueva puntuación en `puntuacion0`. Pero, antes, ten la precaución de copiar el valor de esta preferencia a `puntuacion1`; y `puntuacion1` a `puntuacion2` y así hasta la penúltima. La última se perderá. Esta operación puede realizarse por medio de un bucle con un índice entero, `n`, de forma que el nombre de la preferencia a mover puedes expresarlo como `"puntuacion"+n`.

3. Utiliza el mismo truco para implementar el método `ListaPuntuaciones()`.



**Solución:** *Almacenando las últimas 10 puntuación en un fichero de preferencias.*

4. Remplaza en `guardarPuntuacion()`:

```
editor.putString("puntuacion", puntos + " " + nombre);
```

por:

```
for (int n = 9; n >= 1; n--) {
  editor.putString("puntuacion" + n,
        preferencias.getString("puntuacion" + (n - 1), ""));
}
editor.putString("puntuacion0", puntos + " " + nombre);
```

5. Remplaza en `ListaPuntuaciones()`:

```
String s = preferencias.getString("puntuacion", "");
if (s != "") {
  result.add(s);
}
```

por:

```
for (int n = 0; n <= 9; n++) {
  String s = preferencias.getString("puntuacion" + n, "");
  if (s != "") {
     result.add(s);
  }
}
```

## 9.4. Accediendo a ficheros

Existen tres tipos de ficheros donde podemos almacenar información en Android: ficheros almacenados en la memoria interna del teléfono, ficheros almacenados en la memoria externa (normalmente una tarjeta SD) y ficheros almacenados en los recursos. Estos últimos son de solo lectura por lo que no son útiles para almacenar información desde la aplicación.

Poli[Media]: *Gestión de ficheros en Android.*

## 9.4.1. Sistema interno de ficheros

Android permite almacenar ficheros en la memoria interna del teléfono. Por defecto los ficheros almacenados solo son accesibles por la aplicación que los creó, no pueden ser leídos por otras aplicaciones, ni siquiera por el usuario del teléfono. Cada aplicación dispone de una carpeta especial para almacenar ficheros (*/data/data/nombre_del_paquete/files*). La ventaja de utilizar esta carpeta es que cuando se desinstala la aplicación los ficheros que has creado se eliminarán. Cuando trabajes con ficheros en Android, ten siempre en cuenta que la memoria disponible de los teléfonos móviles es limitada.

Recuerda que el sistema de ficheros se sustenta en la capa Linux, por lo que Android hereda su estructura. Cuando se instala una nueva aplicación, Android crea un nuevo usuario Linux asociado a la aplicación y es este usuario el que podrá, o no, acceder a los ficheros.

Puedes utilizar cualquier rutina del paquete `java.io` para trabajar con ficheros. Adicionalmente, se han creado métodos adicionales asociados a la clase `Contex` para facilitarte el trabajo con ficheros almacenados en la memoria interna. En particular los métodos `openFileInput()` y `openFileOutput()` te permiten abrir un fichero para lectura o escritura respectivamente. Si utilizas estos métodos el nombre del archivo no puede contener subdirectorios. De hecho, el fichero siempre es almacenado en la carpeta reservada para tu aplicación (*/data/data/nombre_del_paquete/files*). Recuerda siempre cerrar los ficheros con el método `close()`. El siguiente ejemplo muestra cómo crear un fichero y escribir en él un texto:

```
String fichero = "fichero.txt";
String texto = "texto almacenado";
FileOutputStream fos;
try {
    fos = openFileOutput(fichero,Context.MODE_PRIVATE);
    fos.write(texto.getBytes());
    fos.close();
} catch (FileNotFoundException e) {
    Log.e("Mi Aplicación",e.getMessage(),e);
} catch (IOException e) {
    Log.e("Mi Aplicación",e.getMessage(),e);
}
```

Es muy importante hacer un manejo cuidadoso de los errores. De hecho, el acceso a ficheros ha de realizarse de forma obligatoria dentro de una sección `try/catch`.

Además de los dos métodos indicados pueden serte útiles algunos de los siguientes: `getFilesDir()` devuelve la ruta absoluta donde se están guardando los

ficheros; `getDir()` crea un directorio en tu almacenamiento interno (o lo abre si existe; `deleteFile()` borra un fichero; `fileList()` devuelve un *array* con los ficheros almacenados por tu aplicación.

**Ejercicio paso a paso:** *Almacenando puntuaciones en un fichero de la memoria interna.*

El siguiente ejercicio muestra una clase que implementa la interfaz `AlmacenPuntuaciones` utilizando los métodos antes descritos.

1. Abre el proyecto *Asteroides*.
2. Crea una nueva clase `AlmacenPuntuacionesFicheroInterno`.
3. Remplaza el código por el siguiente:

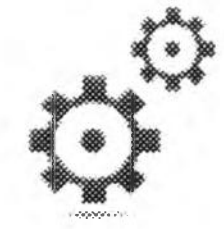
```
public class AlmacenPuntuacionesFicheroInterno implements
AlmacenPuntuaciones {
    private static String FICHERO = "puntuaciones.txt";
    private Context context;

    public AlmacenPuntuacionesFicheroInterno(Context context) {
     this.context = context;
    }

    public void guardarPuntuacion(int puntos, String nombre, long fecha){
       try {
           FileOutputStream f = context.openFileOutput(FICHERO,
                                                Context.MODE_APPEND);
           String texto = puntos + " " + nombre + "\n";
           f.write(texto.getBytes());
           f.close();
       } catch (Exception e) {
           Log.e("Asteroides", e.getMessage(), e);
       }
    }

    public Vector<String> listaPuntuaciones(int cantidad) {
       Vector<String> result = new Vector<String>();
       try {
          FileInputStream f = context.openFileInput(FICHERO);
          BufferedReader entrada = new BufferedReader(
                                   new InputStreamReader(f));
          int n = 0;
          String linea;
          do {
             linea = entrada.readLine();
```

```
            if (linea != null) {
                result.add(linea);
                n++;
            }
        } while (n < cantidad && linea != null);
        f.close();
    } catch (Exception e) {
        Log.e("Asteroides", e.getMessage(), e);
    }
    return result;
}
```

4. Abre el fichero *Asteroides.java* y en el método `onCreate()` reemplaza la línea adecuada por:

```
almacen = new AlmacenPuntuacionesFicheroInterno(this);
```

5. Ejecuta el proyecto y verifica que la última puntuación se guarda correctamente.
6. Abre la vista *File Explorer* y verifica que se ha creado el fichero: */data/data/org.example.asteroides/files/puntuaciones.txt*
7. Descarga este fichero a tu ordenador (botón ) y observa su contenido.



**Practica:** *Configurar almacenamiento de puntuaciones desde preferencias.*

Modifica las preferencias de la aplicación Asteroides para que el usuario pueda seleccionar donde se guardarán las puntuaciones. De momento incluye tres opciones: "Constante", "Preferencias" y "Fichero en memoria interna"

1. Abre el fichero Asteroides.java y en el método `onCreate()` remplaza:

```
almacen = new AlmacenPuntuacionesFicheroInterno(this);
```

   por el código necesario para que se inicialice la variable `almacen` de forma adecuada según el valor introducido en preferencias.
2. Verifica el resultado.
3. Cada vez que añadas un nuevo método de almacenamiento inclúyelo en la lista de preferencias.

## 9.4.2. Sistema de almacenamiento externo

Los teléfonos Android suelen disponer de memoria adicional de almacenamiento, conocido como almacenamiento externo. Éste suele ser de mayor capacidad por lo que resulta ideal para almacenar ficheros de música o vídeo. Suele ser una memoria extraíble, como una tarjeta SD, o una memoria interna no extraíble (algunos modelos incorporan los dos tipos de memoria, es decir almacenamiento externo extraíble y no

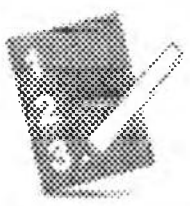
extraíble). Cuando conectamos el dispositivo Android a través del cable USB permitimos el acceso a esta memoria externa, de forma que los ficheros aquí escritos podrán ser leídos, modificados o borrados por cualquier usuario.

Para acceder a la memoria externa lo habitual es utilizar la ruta `/sdcard/...`

En esta carpeta es donde el sistema suele montar la tarjeta SD. No obstante, resulta más conveniente utilizar el método `Environment.getExternalStorageDirectory()` para que el sistema nos indique la ruta exacta.

A partir de la versión 1.6 resulta necesario declarar el permiso `WRITE_EXTERNAL_STORAGE` en `AndroidManifest.xml` para poder escribir en la memoria externa.

Poli[**Media**]: *Almacenamiento externo en Android.*



**Ejercicio paso a paso:** *Almacenando puntuaciones en la memoria externa.*

1. Abre el proyecto del ejercicio anterior.
2. Selecciona el fichero *AlmacenPuntuacionesFicheroInterno.java*, y cópialo en el portapapeles (Ctrl-C).
3. Pega el fichero sobre el proyecto (Ctrl-V) y renómbralo con el nombre *AlmacenPuntuacionesFicheroExterno.java*.
4. Abre la nueva clase creada y remplaza la inicialización de la variable `FICHERO` por:

```
private static String FICHERO = Environment.
        getExternalStorageDirectory() + "/puntuaciones.txt";
```

   Normalmente esta variable valdrá: `"/sdcard/puntuaciones.txt"`.

5. En el método `guardarPuntuacion()` remplaza la inicialización de la variable `fos` por:

```
FileOutputStream fos = new FileOutputStream(FICHERO, true);
```

6. En el método `listaPuntuacion()` remplaza la inicialización de la variable `fis` por:

```
FileInputStream fis = new FileInputStream(FICHERO);
```

7. Abre el fichero *AndroidManifest.xml* y solicita el permiso `WRITE_EXTERNAL_STORAGE`.
8. Ejecuta la aplicación y crea nuevas puntuaciones.
9. Verifica con la vista *File Explorer* que dentro de la carpeta sdcard aparece el fichero.

### *9.4.2.1. Verificando acceso a la memoria externa*

La memoria externa puede haber sido extraída o estar protegida contra escritura.


Puedes utilizar el método `Environment.getExternalStorageState()` para verificar el estado de la memoria. Veamos como se utiliza:

```
String stadoSD = Environment.getExternalStorageState();

if (stadoSD.equals(Environment.MEDIA_MOUNTED)) {
    // Podemos leer y escribir
    ...
} else if (stadoSD.equals(Environment.MEDIA_MOUNTED_READ_ONLY)) {
    // Podemos leer
    ...
} else {
    // No podemos leer y ni escribir
    ...
}
```

**Práctica:** *Verificando acceso a la memoria externa.*

1. Modifica la clase `AlmacenPuntuacionesFicheroExterno` para que, antes de acceder a la memoria externa, verifique que la operación es posible. En caso contrario mostrará un `Toast` y saldrá del método.
2. Ejecuta el programa en un dispositivo real con memoria externa y verifica que se almacena correctamente.
3. Para que la aplicación Asteroides ya no tenga acceso a esta memoria la solución más sencilla consiste en conectar el dispositivo con el cable USB y activar el almacenamiento por USB.
4. Regresa a Asteroides y comprueba el resultado.

**Solución:** *Verificando acceso a la memoria externa.*

1. En `guardarPuntuacion()` añade:

```
String stadoSD = Environment.getExternalStorageState();
if (!stadoSD.equals(Environment.MEDIA_MOUNTED)) {
    Toast.makeText(context, "No puedo escribir en la memoria
    externa", Toast.LENGTH_LONG).show();
    return;
}
```

2. En `listaPuntuacion()` añade:

```
String stadoSD = Environment.getExternalStorageState();
if (!stadoSD.equals(Environment.MEDIA_MOUNTED) &&
    !stadoSD.equals(Environment.MEDIA_MOUNTED_READ_ONLY)) {
      Toast.makeText(context, "No puedo leer en la memoria
      externa", Toast.LENGTH_LONG).show();
      return result;
}
```

### 9.4.2.2. Almacenando ficheros específicos de tu aplicación en el almacenamiento externo

A partir de la versión 2.2 (nivel de API 8) las aplicaciones pueden almacenar los ficheros en una carpeta específica del sistema de almacenamiento externo, de forma que cuando la aplicación sea desinstalada se borren automáticamente estos ficheros. En concreto esta carpeta ha de seguir esta estructura:

```
/Android/data/<nombre_del_paquete>/files/
```

donde el paquete *<nombre_del_paquete>* ha de cambiarse por el nombre del paquete de la aplicación, por ejemplo `org.example.asteroides`.

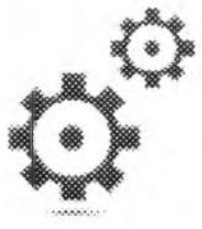
A partir del nivel de API 8 puedes utilizar el método `getExternalFilesDir(null)` para obtener esta ruta. Si en lugar de `null` indicas alguna de las constantes que se indican más abajo, se devolverá la ruta a una carpeta específica según el tipo de contenido que nos interese. Este método crea la carpeta en caso de no existir previamente. Indicando la carpeta garantizamos que el escáner de medios de Android categoriza los ficheros de forma adecuada. Por ejemplo, un tono de llamada será identificado como tal y no como un fichero de música. De esta forma, no aparecerá en la lista de música que puede reproducir el reproductor multimedia. Estas carpetas también son eliminadas cuando se desinstala la aplicación.

| Constante | carpeta | descripción |
|---|---|---|
| `DIRECTORY_MUSIC` | `Music` | Ficheros de música. |
| `DIRECTORY_PODCASTS` | `Podcasts` | Descargas desde podcast. |
| `DIRECTORY_RINGTONES` | `Ringtones` | Tono de llamada de teléfono. |
| `DIRECTORY_ALARMS` | `Alarms` | Sonidos de alarma. |
| `DIRECTORY_NOTIFICATIONS` | `Notifications` | Sonidos para notificaciones. |
| `DIRECTORY_PICTURES` | `Pictures` | Ficheros con fotografías. |
| `DIRECTORY_DOWNLOADS` | `Download` | Descargas de cualquier tipo. |
| `DIRECTORY_DCIM` | `DCIM` | Carpeta que tradicionalmente crean las cámaras. |

***NOTA.*** *Si tu aplicación es creada para el nivel de API inferior a 8 no tendrás acceso al método* `getExternalFilesDir(null)`. *No obstante, puedes almacenar a mano los ficheros en una carpeta con la estructura:*

```
getExternalStorageDirectory() +
            "/Android/data/<nombre_del_paquete>/files/"
```

Cuando se desinstale tu aplicación esta carpeta será eliminada si se ha instalado en dispositivos con una versión 2.2 o superior.



**Práctica:** *Almacenando puntuaciones en una carpeta de la aplicación de la memoria externa.*

1. Selecciona el fichero *AlmacenPuntuacionesFicheroExterno.java*, y cópialo en el portapapeles (Ctrl-C).
2. Pega el fichero sobre el proyecto (Ctrl-V) y renómbralo con el nombre *AlmacenPuntuacionesFicheroExtApl.java*.
3. Modifica los método `listaPuntuaciones()` y `guardarPuntuacion()` para que las puntuaciones se almacenen en la SD pero en una carpeta de tu aplicación. Realiza esta tarea con una versión de SDK para Asteroides inferior a la 2.2.
4. Puedes utilizar el siguiente código para crear el directorio:

```
File ruta = new
File(Environment.getExternalStorageDirectory() +
            "/Android/data/org.example.asteroides/files/");
if (!ruta.exists()) {
    ruta.mkdirs();
}
```

5. Verifica el resultado ejecutando la aplicación en un terminal con una versión inferior a la 2.2. Desinstala la aplicación y verifica si el fichero ha sido eliminado.
6. Ejecuta ahora la aplicación en un terminal con una versión igual o superior a la 2.2. Desinstala la aplicación y verifica si el fichero ha sido eliminado.

### *9.4.2.3. Almacenando ficheros compartidos en el almacenamiento externo*

Si quieres crear un fichero que no sea específico para tu aplicación y quieres que no sea borrado cuando tu aplicación sea desinstalada, puedes crearlo en cualquier directorio del almacenamiento externo.

Lo ideal es que utilices alguno de los directorios públicos creados para almacenar diferentes tipos de ficheros. Estos directorios parten de la raíz del almacenamiento externo y siguen con alguna de las carpetas listadas en la tabla anterior.

A partir del nivel de API 8, para obtener la ruta de un directorio compartido, puedes utilizar el método `getExternalStoragePublicDirectory(String tipo)`. Como

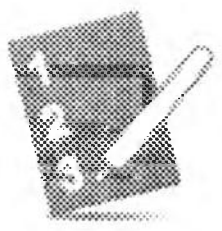
parámetro utiliza alguna de las constantes que se indican en la tabla anterior. Guardando los ficheros en las carpetas adecuadas garantizamos que el escáner de medios de Android categoriza los ficheros de forma adecuada.

Si utilizas un nivel de API anterior al 8, lo recomendable es crear estas carpetas manualmente.

> ***NOTA:*** *Si quieres que tus ficheros estén ocultos al escáner de medios incluye un fichero vacío con nombre .nomedia en la carpeta donde estén almacenados.*

### 9.4.3. Acceder a un fichero de los recursos

También tienes la posibilidad de almacenar ficheros en los recursos, es decir, adjuntos al paquete de la aplicación. Has de tener en cuenta que estos ficheros no podrán ser modificados. Por ejemplo, si arrastras cualquier fichero con nombre `datos` y cualquier nombre extensión a la carpeta `res/raw`, podrás acceder a él usando `Resources.openRawResource(R.raw.datos)`. Recuerda que ningún fichero en la carpeta `raw` será comprimido.



**Ejercicio paso a paso:** *Almacenando puntuaciones en un fichero de recursos.*

1. Con el explorador de ficheros busca en el terminal un fichero de texto con nombre *puntuaciones.txt*, creado en alguno de los ejercicios anteriores.
2. Extráelo del terminal y pégalo en el proyecto Asteroides en la carpeta *res/raw*.
3. Selecciona el fichero *AlmacenPuntuacionesFicheroInterno.java*, y cópialo en el portapapeles (Ctrl-C).
4. Pega el fichero sobre el proyecto (Ctrl-V) y renómbralo con el nombre *AlmacenPuntuacionesRecurso.java*.
5. Elimina de esta clase todo el código del método `guardarPuntuacion()`. Al igual que en la clase *AlmacenPuntuacionesConstante* no se realiza ninguna acción en este método.
6. Remplaza en el método `listaPuntuaciones()` para que las puntuaciones se lean del fichero de los recursos:

   ```
   FileInputStream f = context.openFileInput(FICHERO);
   ```

   por:

   ```
   InputStream f = context.getResources().openRawResource(
                                              R.raw.puntuaciones);
   ```

7. La siguiente línea ya no tiene sentido. Elimínala:

   ```
   private static String FICHERO = "puntuaciones.txt";
   ```

8. Verifica el resultado.

## 9.5. Trabajando con XML

Como sabrás, XML es uno de los estándares más utilizados en la actualidad para codificar información. Es ampliamente utilizado en Internet; además, como hemos mostrado a lo largo de este libro se utiliza para múltiples usos en el SDK de Android. Entre otras cosas es utilizado para definir *Layouts*, animaciones, `AndroidManifest.xml`,...

Una de las mayores fortalezas de la plataforma Android es que se aprovecha del lenguaje de programación Java y sus librerías. El SDK de Android no acaba de ofrecer todo lo disponible para su estándar del entorno de ejecución Java (JRE), pero es compatible con una fracción muy significativa de la misma. Lo mismo ocurre en lo referente a trabajar con XML; Java dispone de gran cantidad de APIs con este propósito pero no todas están disponibles desde Android.

Librerías disponibles:

*Java's Simple API for XML* (SAX) (paquetes `org.xml.sax.*`)

*Document Object Model* (DOM) (paquetes `org.w3c.dom.*`)

Librerías no disponibles:

*Streaming API for XML* (StAX). Aunque se dispone de otra librería con funcionalidad equivalente (paquete `org.xmlpull.v1.XmlPullParser`).

*Java Architecture for XML Binding* (JAXB). Resultaría demasiado pesada para Android.

Como podrás ver al estudiar los ejemplos, leer y escribir ficheros XML es muy laborioso y necesitarás algo de esfuerzo para comprender el código empleado. Vamos a explicar las dos alternativas más importantes, SAX y DOM. El planteamiento es bastante diferente. Tras ver los ejemplos podrás decidir qué herramienta se adapta mejor a tus gustos personales o al problema en concreto que tengas que resolver.

El ejemplo utilizado para ilustrar el trabajo con XML va a ser el mismo que el utilizado en el resto del capítulo: almacenar las mejores puntuaciones obtenidas. El formato XML que se utilizará para este propósito se muestra a continuación:

```
<?xml version="1.0" encoding="UTF-8"?>
<lista_puntuaciones>
  <puntuacion fecha="1288122023410">
     <nombre>Mi nombre</nombre>
     <puntos>45000</puntos>
  </puntuacion>
  <puntuacion fecha="1288122428132">
     <nombre>Otro nombre</nombre>
     <puntos>31000</puntos>
  </puntuacion>
</lista_puntuaciones>
```

### 9.5.1. Procesando XML con SAX

El uso de la API SAX (*Simple API for XML*) se recomienda cuando se desea un programa de análisis rápido y se quiere reducir al mínimo el consumo de memoria de la aplicación. Eso hace que sea muy apropiado para un dispositivo móvil con Android. También resulta ventajoso para procesar ficheros de gran tamaño.

SAX nos facilita realizar un *parser* (analizador) sobre un documento XML para así poder analizar su contenido. Ha de quedar claro que SAX no almacena los datos. Por lo tanto, necesitaremos una estructura de datos donde guardar la información contenida en el XML. Para realizar este *parser* se van a ir generando una serie de eventos a medida que se vaya leyendo el documento secuencialmente. Por ejemplo, al analizar el documento XML anterior, SAX generará los siguientes eventos:

```
Comienza elemento: lista_puntuaciones
Comienza elemento: puntuacion, con atributo fecha="1288122023410"
Comienza elemento: nombre
Texto de nodo: Mi nombre
Finaliza elemento: nombre
Comienza elemento: puntos
Texto de nodo: 45000
Finaliza elemento: puntos
Finaliza elemento: puntuacion
Comienza elemento: puntuacion, con atributo fecha="1288122428132"
Comienza elemento: nombre
Texto de nodo: Otro nombre
Finaliza elemento: nombre
Comienza elemento: puntos
Texto de nodo: 31000
Finaliza elemento: puntos
Finaliza elemento: puntuacion
Finaliza elemento: lista_puntuaciones
```

Para analizar un documento mediante SAX, vamos a escribir métodos asociados a cada tipo de eventos. Este proceso se realiza extendiendo la clase `DefaultHandler` que nos permite rescribir 5 métodos. Los métodos listados a continuación serán llamados a medida que ocurran los eventos del proceso de lectura secuencial del documento.

`startDocument()`: comienza el Documento XML.

`endDocument()`: finaliza documento XML.

`startElement(String uri, String nombreLocal, String nombreCualif, Attributes atributos)`: comienza una nueva etiqueta; se indican los parámetros:

`uri`: La uri del espacio de nombres o vacío, si no se ha definido.

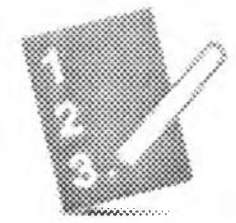
`nombreLocal`: nombre local de la etiqueta sin prefijo.

`nombreCualif`: nombre cualificado de la etiqueta con prefijo.

`atributos`: lista de atributos de la etiqueta.

`endElement(String uri, String nombreLocal, String nombreCualif)`: termina una etiqueta.

`characters(char ch[], int comienzo, int longitud)`: devuelve los caracteres dentro de una etiqueta. Es decir en `<etiqueta> caracteres </etiqueta>` Devolvería `caracteres`. Para obtener un *String* con estos caracteres: `String s = new String(ch,comienzo,longitud)`. Más adelante veremos un ejemplo de cómo utilizar este método.



**Ejercicio paso a paso:** *Almacenando puntuaciones en XML con SAX.*

Una vez descritos los principios de trabajo con SAX, pasemos a implementar la interfaz `AlmacenPuntuaciones` mediante esta API.

1. Crea la clase `AlmacenPuntuacionesXML_SAX` en la aplicación `Asteroides` y escribe el siguiente código:

```
public class AlmacenPuntuacionesXML_SAX implements
AlmacenPuntuaciones {
    private static String FICHERO = "puntuaciones.xml";
    private Context contexto;
    private ListaPuntuaciones lista;
    private boolean cargadaLista;

    public AlmacenPuntuacionesXML_SAX(Context contexto) {
        this.contexto = contexto;
        lista = new ListaPuntuaciones();
        cargadaLista = false;
    }

    @Override
    public void guardarPuntuacion(int puntos, String nombre,
                                  long fecha) {
        try {
            if (!cargadaLista){
                lista.leerXML(contexto.openFileInput(FICHERO));
            }
        } catch (FileNotFoundException e) {
        } catch (Exception e) {
            Log.e("Asteroides", e.getMessage(), e);
        }
        lista.nuevo(puntos, nombre, fecha);
        try {
            lista.escribirXML(contexto.openFileOutput(FICHERO,
                              Context.MODE_PRIVATE));
        } catch (Exception e) {
```

```
            Log.e("Asteroides", e.getMessage(), e);
        }
    }

    @Override
    public Vector<String> listaPuntuaciones(int cantidad) {
        try {
            if (!cargadaLista){
                lista.leerXML(contexto.openFileInput(FICHERO));
            }
        } catch (Exception e) {
            Log.e("Asteroides", e.getMessage(), e);
        }
        return lista.aVectorString();
    }
```

La nueva clase comienza definiendo una serie de variables y constantes. En primer lugar el nombre del fichero donde se guardarán los datos. Con el valor indicado, el fichero se almacenará en `/data/data/org.example.asteroides/files/puntuaciones.xml`. Pero puedes almacenarlos en otro lugar, como por ejemplo en la memoria SD indicando `/sdcard/puntuaciones.xml`. La variable más importante es `lista` de la clase `ListaPuntuaciones`. En ella guardaremos la información contenida en el fichero XML. Esta clase es definida a continuación. La variable `cargadaLista` nos indica si `lista` ya ha sido leída desde el fichero.

El código continúa sobrescribiendo los dos métodos de la interfaz. En `guardarPuntuacion()` comenzamos verificando si `lista` ya ha sido cargada, para hacerlo en caso necesario. Es posible que el programa se esté ejecutando por primera vez, en tal caso el fichero no existirá. En este caso se producirá una excepción de tipo `FileNotFoundException` al tratar de abrir el fichero. Esta excepción es capturada por nuestro código, pero no realizamos ninguna acción dado que no se trata de un verdadero error. A continuación se añade un nuevo elemento a `lista` y se escribe de nuevo el fichero XML. El siguiente método, `listaPuntuacion()`, resulta sencillo de entender al limitarse a métodos definidos en la clase `ListaPuntuaciones`.

2. Pasemos a mostrar el comienzo de la clase `ListaPuntuaciones`. No es necesario almacenarla en un fichero aparte, puedes definirla dentro de la clase anterior. Para ello, copia el siguiente código justo antes del último `}` de la clase `AlmacenPuntuacionesXML_SAX`:

```
private class ListaPuntuaciones {

private class Puntuacion {
    int puntos;
    String nombre;
    long fecha;
}
```

```
private List<Puntuacion> listaPuntuaciones;

public ListaPuntuaciones() {
    listaPuntuaciones = new ArrayList<Puntuacion>();
}

public void nuevo(int puntos, String nombre, long fecha) {
    Puntuacion puntuacion = new Puntuacion();
    puntuacion.puntos = puntos;
    puntuacion.nombre = nombre;
    puntuacion.fecha = fecha;
    listaPuntuaciones.add(puntuacion);
}

public Vector<String> aVectorString() {
    Vector<String> result = new Vector<String>();
    for (Puntuacion puntuacion : listaPuntuaciones) {
        result.add(puntuacion.nombre+" "+puntuacion.puntos);
    }
    return result;
}
```

El objetivo de esta clase es mantener una lista de objetos `Puntuacion`. Dispone de métodos para insertar un nuevo elemento (`nuevo()`) y devolver un listado con todas las puntuaciones almacenadas (`aVectorString()`).

3. Lo verdaderamente interesante de esta clase es que permite la lectura y escritura de los datos desde un documento XML (`leerXML()` y `escribirXML()`). Veamos primero cómo leer un documento XML usando SAX. Escribe el siguiente código a continuación del anterior:

```
public void leerXML(InputStream entrada) throws Exception {
    SAXParserFactory fabrica = SAXParserFactory.newInstance();
    SAXParser parser = fabrica.newSAXParser();
    XMLReader lector = parser.getXMLReader();
    ManejadorXML manejadorXML = new ManejadorXML();
    lector.setContentHandler(manejadorXML);
    lector.parse(new InputSource(entrada));
    cargadaLista = true;
}
```

Para leer un documento XML comenzamos creando una instancia de la clase `SAXParserFactory`, lo que nos permite crear un nuevo *parser* XML de tipo `SAXParser`. Luego creamos un lector, de la clase `XMLReader`, asociado a este *parser*. Creamos `manejadorXML` de la clase `XMLHadler` y asociamos este manejador al `XMLReader`. Para finalizar le indicamos al `XMLReader` qué entrada tiene para que realice el proceso de *parser*. Una vez finalizado el proceso, marcamos que el fichero está cargado.

Como ves, el proceso es algo largo pero siempre se realiza igual. Donde sí que tendremos que trabajar algo más es en la creación de la clase `XMLHadler`, dado que va a depender del formato del fichero que queramos leer. Esta clase es listada a continuación.

4. Escribir este código a continuación del anterior:

```
class ManejadorXML extends DefaultHandler {
  private StringBuilder cadena;
  private Puntuacion puntuacion;

  @Override
  public void startDocument() throws SAXException {
     listaPuntuaciones = new ArrayList<Puntuacion>();
     cadena = new StringBuilder();
  }

  @Override
  public void startElement(String uri, String nombreLocal, String
          nombreCualif, Attributes atr) throws SAXException {
     cadena.setLength(0);
     if (nombreLocal.equals("puntuacion")) {
        puntuacion = new Puntuacion();
        puntuacion.fecha = Long.parseLong(atr.getValue("fecha"));
     }
  }

  @Override
  public void characters(char ch[], int comienzo, int lon) {
     cadena.append(ch, comienzo, lon);
  }

  @Override
  public void endElement(String uri, String nombreLocal,
        String nombreCualif) throws SAXException {
     if (nombreLocal.equals("puntos")) {
        puntuacion.puntos = Integer.parseInt(cadena.toString());
     } else if (nombreLocal.equals("nombre")) {
        puntuacion.nombre = cadena.toString();
     } else if (nombreLocal.equals("puntuacion")) {
        listaPuntuaciones.add(puntuacion);
     }
  }

  @Override
  public void endDocument() throws SAXException {
  }
}
```

Esta clase define un manejador que captura los cinco eventos generados en el proceso de *parsing* en SAX. En `startDocument()` nos limitamos a inicializar variables. En `startElement()` verificamos que hemos llegado a una etiqueta `<puntuación>`. El tal caso, creamos un nuevo objeto de la clase `Puntuacion` e inicializamos el campo `fecha` con el valor indicado en uno de los atributos.

El método `characters()` es llamado cuando aparece texto dentro de una etiqueta (`<etiqueta> caracteres </etiqueta>`). Nos limitamos a almacenar este texto en la variable `cadena` para utilizarlo en el siguiente método. SAX no nos garantiza que nos pasará todo el texto en un solo evento; si el texto es muy extenso se realizarían varias llamadas a este método. Por esta razón, el texto se va acumulando en `cadena`.

El método `endElement()` resulta más complejo, dado que en función de qué etiqueta esté acabando realizaremos una tarea diferente. Si se trata de `</puntos>` o de `</nombre>` utilizaremos el valor de la variable `cadena` para actualizar el valor correspondiente. Si se trata de `</puntuacion>` añadimos el objeto `puntuacion` a la lista.

5. Introduce a continuación el último método de la clase `ListaPuntuaciones`, que nos permite escribir el documento XML:

```
public void escribirXML(OutputStream salida) {
    XmlSerializer serializador = Xml.newSerializer();
    try {
        serializador.setOutput(salida, "UTF-8");
        serializador.startDocument("UTF-8", true);
        serializador.startTag("", "lista_puntuaciones");
        for (Puntuacion puntuacion : listaPuntuaciones) {
            serializador.startTag("", "puntuacion");
            serializador.attribute("", "fecha",
                    String.valueOf(puntuacion.fecha));
            serializador.startTag("", "nombre");
            serializador.text(puntuacion.nombre);
            serializador.endTag("", "nombre");
            serializador.startTag("", "puntos");
            serializador.text(String.valueOf(puntuacion.puntos));
            serializador.endTag("", "puntos");
            serializador.endTag("", "puntuacion");
        }
        serializador.endTag("", "lista_puntuaciones");
        serializador.endDocument();
        } catch (Exception e) {
            Log.e("Asteroides", e.getMessage(), e);
        }
    }
}
```

Como puedes ver todo el trabajo se realiza por medio de un objeto de la clase `XmlSerializer` al que se le asigna como salida el `OuputStream` que hemos pasado como parámetros.

## 9.5.2. Procesando XML con DOM

DOM (Modelo de Objetos del Documento) es un API creado por W3C (*World Wide Web Consortium*) que nos permite manipular dinámicamente documentos XML y HTML. Android soporta el nivel de especificación 3, por lo que permite trabajar con definición de tipo de documento (DTD) y validación de documentos. Para no extender en exceso los ejemplos no vamos a entrar en la definición y validación de documentos.

Como ya hemos comentado, el planteamiento de DOM es muy diferente al de SAX. DOM permite cargar cualquier tipo de documento XML y manipularlo directamente en memoria (RAM). Podremos crear nuevos nodos, o modificar los existentes. Una vez dispongamos de la nueva versión podremos almacenarlo en un fichero o mandarlo por Internet.

Trabajar con DOM tiene sus ventajas frente a SAX, por ejemplo nos evitamos definir a mano el proceso de *parser* (clase `ManejadorXML`) y crear una estructura para almacenar los datos (clase `ListaPuntuaciones`). Pero también tiene sus inconvenientes: recorrer un documento DOM puede ser algo complejo, además, al tener que cargarse todo el documento en memoria puede consumir excesivos recursos para un dispositivo como un teléfono móvil. Este inconveniente cobra especial relevancia al trabajar con documentos grandes. Para terminar, DOM procesa la información de forma más lenta.

**Ejercicio paso a paso:** *Almacenando puntuaciones en XML con DOM.*

Veamos cómo se implementa el ejemplo anterior mediante el API DOM.

1. Crea la clase `AlmacenPuntuacionesXML_DOM` y escribe el siguiente código:

```
public class AlmacenPuntuacionesXML_DOM implements AlmacenPuntuaciones{
private static String FICHERO = "puntuaciones.xml";
private Context contexto;
private Document documento;
private boolean cargadoDocumento;

public AlmacenPuntuacionesXML_DOM(Context contexto) {
    this.contexto = contexto;
    cargadoDocumento = false;
}

@Override
public void guardarPuntuacion(int puntos, String nombre, long fecha){
```

```
try {
    if (!cargadoDocumento) {
        leerXML(contexto.openFileInput(FICHERO));
    }
} catch (FileNotFoundException e) {
    crearXML();
} catch (Exception e) {
    Log.e("Asteroides", e.getMessage(), e);
}
nuevo(puntos, nombre, fecha);
    try {
        escribirXML(contexto.openFileOutput(FICHERO,
                                    Context.MODE_PRIVATE));
    } catch (Exception e) {
        Log.e("Asteroides", e.getMessage(), e);
    }
}

@Override
public Vector<String> listaPuntuaciones(int cantidad) {
    try {
        if (!cargadoDocumento) {
            leerXML(contexto.openFileInput(FICHERO));
        }
    } catch (FileNotFoundException e) {
        crearXML();
    } catch (Exception e) {
        Log.e("Asteroides", e.getMessage(), e);
    }
    return aVectorString();
}
```

La clase comienza definiendo una serie de variables y constantes. Son iguales que en el ejemplo anterior con la excepción de `documento` de la clase `org.w3c.dom.Document.Document`. En este objeto mantendremos en memoria nuestro documento XML. Tras el constructor, se definen los dos métodos de la interfaz. Su funcionamiento es similar al ejemplo anterior, aunque ahora en lugar de definir una clase nueva, definiremos los métodos: `crearXML()`, `leerXML()`, `nuevo()`, `aVectorString()` y `escribirXML()`. Estos métodos tienen por objeto interactuar con el objeto `documento`.

2. Veamos los dos primeros. Añade el siguiente código:

```
public void crearXML() {
    try {
        DocumentBuilderFactory fabrica =
                        DocumentBuilderFactory.newInstance();
        DocumentBuilder constructor = fabrica.newDocumentBuilder();
```

```
            documento = constructor.newDocument();
            Element raiz = documento.createElement("lista_puntuaciones");
            documento.appendChild(raiz);
            cargadoDocumento = true;
        } catch (Exception e) {
            Log.e("Asteroides", e.getMessage(), e);
        }
    }

    public void leerXML(InputStream entrada) throws Exception {
        DocumentBuilderFactory fabrica =
                            DocumentBuilderFactory.newInstance();
        DocumentBuilder constructor = fabrica.newDocumentBuilder();
        documento = constructor.parse(entrada);
        cargadoDocumento = true;
    }
```

En `crearXML()` comenzamos construyendo objetos `DocumentBuilderFactory` y `DocumentBuilder` para poder crear una nueva instancia de `documento`. Creamos un nuevo elemento que es añadido en la raíz de `documento`. Finalizamos marcando que el documento está creado.

En `leerXML()` el proceso es similar, aunque ahora llamamos al método `parse()` que se encargará de procesar la entrada. Como puedes comprobar, el proceso de lectura del documento es mucho más sencillo en DOM que en SAX.

3. Veamos los siguientes dos métodos. Añade el siguiente código:

```
    public void nuevo(int puntos, String nombre, long fecha) {
        Element puntuacion = documento.createElement("puntuacion");
        puntuacion.setAttribute("fecha", String.valueOf(fecha));
        Element e_nombre = documento.createElement("nombre");
        Text texto = documento.createTextNode(nombre);
        e_nombre.appendChild(texto);
        puntuacion.appendChild(e_nombre);
        Element e_puntos = documento.createElement("puntos");
        texto = documento.createTextNode(String.valueOf(puntos));
        e_puntos.appendChild(texto);
        puntuacion.appendChild(e_puntos);
        Element raiz = documento.getDocumentElement();
        raiz.appendChild(puntuacion);
    }

    public Vector<String> aVectorString() {
        Vector<String> result = new Vector<String>();
        String nombre = "", puntos = "";
        Element raiz = documento.getDocumentElement();
        NodeList puntuaciones = raiz.getElementsByTagName("puntuacion");
```

```
for (int i = 0; i < puntuaciones.getLength(); i++) {
    Node puntuacion = puntuaciones.item(i);
    NodeList propiedades = puntuacion.getChildNodes();
    for (int j = 0; j < propiedades.getLength(); j++) {
       Node propiedad = propiedades.item(j);
       String etiqueta = propiedad.getNodeName();
       if (etiqueta.equals("nombre")) {
          nombre = propiedad.getFirstChild().getNodeValue();
       } else if (etiqueta.equals("puntos")) {
          puntos = propiedad.getFirstChild().getNodeValue();
       }
    }
    result.add(nombre + " " + puntos);
  }
  return result;
}
```

El método `nuevo()` tiene por objeto insertar dentro de `documento` una nueva etiqueta `<puntuacion>` con los atributos y etiquetas interiores necesarios. Comienza creando el elemento `puntuacion` al que le añade el atributo `fecha`. Luego, crea el elemento `nombre` y le añade el texto correspondiente. Este elemento es añadido como hijo a `puntuacion`. El mismo proceso se repite para el elemento `puntos`. Para finalizar el elemento `puntuacion` es añadido a la raíz del documento.

El método `aVectorString()` devuelve un vector de *strings* con las puntuaciones almacenadas. Para ello, iremos recorriendo todo el documento comenzando por el elemento `raiz`. Obtenemos la lista de nodos `puntuaciones` para recorrerla en un bucle `for`. Para cada uno de estos nodos, obtenemos en `propiedades` los nodos hijos. Recorremos esta segunda lista en un nuevo bucle, donde analizamos si el nombre del nodo es `nombre` o `puntos`, y guardamos el valor asociado al nodo en la variable correspondiente. Antes de pasar al siguiente nodo `puntuacion`, añadimos lo obtenido al resultado.

4. Veamos el último método. Añade el siguiente código:

```
public void escribirXML(OutputStream salida) throws Exception {
   TransformerFactory fabrica = TransformerFactory.newInstance();
   Transformer transformador = fabrica.newTransformer();
   transformador.setOutputProperty(OutputKeys.OMIT_XML_DECLARATION,"yes");
   transformador.setOutputProperty(OutputKeys.INDENT, "yes");
   DOMSource fuente = new DOMSource(documento);
   StreamResult resultado = new StreamResult(salida);
   transformador.transform(fuente, resultado);
}
```

Este método permite escribir el documento XML utilizando un objeto de la clase `javax.xml.transform.Transformer`. Tras configurarlo se le indica

como fuente `documento` y como resultado de la trasformación el `OuputStream` pasado como parámetro.

5. Por desgracia, la clase `Transformer` solo está disponible a partir del nivel de API 8. Si quieres implementar una aplicación que sea compatible con la totalidad de teléfonos te proponemos que reemplaces el método anterior por el siguiente:

```
public void escribirXML(OutputStream salida) throws Exception {
    String s = serializa(documento.getDocumentElement());
    salida.write(s.getBytes("UTF-8"));
}

public static String serializa(Node raiz) throws IOException {
    StringBuilder resultado = new StringBuilder();
    if (raiz.getNodeType() == Node.TEXT_NODE)
        resultado.append(raiz.getNodeValue());
    else {
        if (raiz.getNodeType() != Node.DOCUMENT_NODE) {
            StringBuffer atributos = new StringBuffer();
            for (int i=0; i < raiz.getAttributes().getLength(); ++i) {
                atributos.append(" ")
                .append(raiz.getAttributes().item(i).getNodeName())
                .append("=\"")
                .append(raiz.getAttributes().item(i).getNodeValue())
                .append("\" ");
            }
            resultado.append("<").append(raiz.getNodeName())
                .append(" ").append(atributos).append(">");
        } else {
            resultado.append("<?xml version=\"1.0\"
                                encoding=\"UTF-8\"?>");
        }
        NodeList listaNodos = raiz.getChildNodes();
        for (int i = 0; i < listaNodos.getLength(); i++) {
            Node nodo = listaNodos.item(i);
            resultado.append(serializa(nodo));
        }
        if (raiz.getNodeType() != Node.DOCUMENT_NODE) {
            resultado.append("</").append(raiz.getNodeName())
                                                .append(">");
        }
        return resultado.toString();
    }
```

El método `serializa()` convierte un documento DOM a un `String`. No es específica de nuestro ejemplo pero puedes usarla para convertir cualquier documento XML. Se trata de una función recursiva. A partir de un nodo que se pasa como parámetro, se encarga de realizar una nueva llamada por cada uno de los nodos que contiene. Si queremos convertir todo el documento, en la primera llamada hay que indicar en el parámetro el nodo raíz del documento.

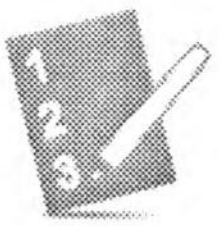
## 9.6. Bases de datos

Las bases de datos son una herramienta de gran potencia en la creación de aplicaciones informáticas. Hasta hace muy poco resultaba muy costoso y complejo incorporar bases de datos a nuestras aplicaciones. No obstante, Android incorpora la librería SQLite que nos permitirá utilizar bases de datos mediante el lenguaje SQL, de una forma sencilla y utilizando muy pocos recursos del sistema. Como verás en este apartado, almacenar tu información en una base de datos no es mucho más complejo que almacenarlos en un fichero y, además, resulta mucho más potente.

SQL es el lenguaje de programación más utilizado para bases de datos. No resulta complejo entender los ejemplos que se mostrarán en este libro. No obstante, si deseas hacer cosas más complicadas te recomiendo que consultes alguno de los muchos manuales que se han escrito sobre el tema.

Para manipular una base de datos en Android usaremos la clase `SQLiteOpenHelper` que nos facilita tanto la creación de la base de datos, como el trabajar con futuras versiones de esta base de datos. Para crear un descendiente de esta clase hay que implementar los métodos `onCreate()`, y `onUpgrade()` y opcionalmente `onOpen()`. La gran ventaja de utilizar esta clase es que ella se preocupará de abrir la base de datos si existe o de crearla si no existe. Incluso de actualizar la versión si decidimos crear una nueva estructura de la base de datos. Además, esta clase tiene dos métodos `getReadableDatabase()` y `getWritableDatabase()` que abren la base de datos en modo solo lectura o lectura y escritura. En caso de todavía no existir la base de datos, estos métodos se encargarán de crearla.

**Poli[Media]**: *Uso de bases de datos en Android.*



**Ejercicio paso a paso:** *Utilizando una base de datos para guardar puntuaciones.*

Pasemos a demostrar cómo guardar las puntuaciones obtenidas en Asteroides en una base de datos. Si comparas la solución propuesta con las anteriores verás cómo el código necesario es menor. Además, una base de datos te da mucha más

potencia; puedes por ejemplo ordenar la salida por puntuación, eliminar filas antiguas, etc. Todo esto sin aumentar apenas el uso de recursos.

1. Crea la clase `AlmacenPuntuacionesSQLite` en el proyecto `Asteroides` y escribe el siguiente código:

```
public class AlmacenPuntuacionesSQLite extends SQLiteOpenHelper
                                        implements AlmacenPuntuaciones{
    //Métodos de SQLiteOpenHelper
    public AlmacenPuntuacionesSQLite(Context context) {
        super(context, "puntuaciones", null, 1);
    }

    @Override public void onCreate(SQLiteDatabase db) {
        db.execSQL("CREATE TABLE puntuaciones ("+
            "_id INTEGER PRIMARY KEY AUTOINCREMENT, "+
            "puntos INTEGER, nombre TEXT, fecha LONG)");
    }

    @Override    public void onUpgrade(SQLiteDatabase db,
                                int oldVersion, int newVersion) {
    // En caso de una nueva versión habría que actualizar las tablas
    }

    //Métodos de AlmacenPuntuaciones
    public void guardarPuntuacion(int puntos, String nombre,
                                                long fecha) {
        SQLiteDatabase db = getWritableDatabase();
        db.execSQL("INSERT INTO puntuaciones VALUES ( null, "+
            puntos+", '"+nombre+"', "+fecha+")");
    }

    public Vector<String> listaPuntuaciones(int cantidad) {
        Vector<String> result = new Vector<String>();
        SQLiteDatabase db = getReadableDatabase();
        Cursor cursor = db.rawQuery("SELECT puntos, nombre FROM " +
         "puntuaciones ORDER BY puntos DESC LIMIT " +cantidad, null);
        while (cursor.moveToNext()){
            result.add(cursor.getInt(0)+" " +cursor.getString(1));
        }
        cursor.close();
        return result;
    }
}
```

El constructor de la clase se limita a llamar al constructor heredado con el perfil:

```
SQLiteOpenHelper(Context contexto, String nombre,
      SQLiteDatabase.CursorFactory cursor, int version).
```

Los parámetros se describen a continuación:

`contexto`: contexto usado para abrir o crear la base de datos.

`nombre`: nombre de la base de datos que se creará. En nuestro caso "puntuaciones".

`cursor`: se utiliza para crear un objeto de tipo cursor. En nuestro caso no lo necesitamos.

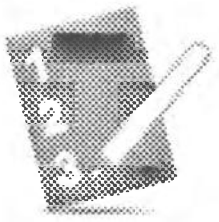
`version`: número de versión de la base de datos empezando desde 1. En el caso de que la base de datos actual tenga una versión más antigua se llamará a `onUpgrade()` para que actualice la base de datos.

El método `onCreate()` será invocado cuando sea necesario crear la base de datos. En nuestro caso tendrá solo la tabla `puntuaciones` que es creada por medio del comando SQL `CREATE TABLE puntuaciones...`

El método `onUpgrade()` no ha sido implementado. Si más adelante decidiéramos crear una nueva estructura para la base de datos, tendríamos que indicar un número de versión superior, por ejemplo la 2. Cuando se ejecute el código sobre un sistema donde se dispone de una base de datos con la versión 1, será invocado el método `onUpgrade()`. En él tendríamos que escribir los comandos necesarios para transformar la antigua base de datos en la nueva, tratando de conservar la información de la versión anterior.

Pasemos a describir los dos métodos de la interfaz `AlmacenaPuntuaciones`. El método `guardarPuntuacion()` comienza obteniendo una referencia a nuestra base de datos utilizando `getWritableDatabase()`. Con ella ejecuta el comando SQL para almacenar un nueva fila en la tabla `INSERT INTO puntuaciones ...` El método `listaPuntuaciones()` comienza obteniendo una referencia a nuestra base de datos utilizando `getReadableDatabase()`. Realiza una consulta utilizando el método `rawQuery()`, con la que obtiene un cursor que utiliza para leer todas las filas devueltas en la consulta.

2. Modifica el código correspondiente en *Asteroides.java* para que este método pueda ser seleccionado para almacenar las puntuaciones.
3. Verifica su funcionamiento.



**Ejercicio paso a paso:** *Verificación de los ficheros creados.*

1. Abre la vista *File Explorer*: utilizando el menú *Window/Show View/Other.../ Android/File Explorer*.
2. Si estás utilizando varios dispositivos abre la vista *Devices* y asegúrate de seleccionar el dispositivo donde has ejecutado Asteroides.

3. Busca la ruta data/data/org.example.asteroides
4. Observa los ficheros creados y compara el tamaño de cada uno.


## 9.6.1. Los métodos query() y rawQuery()

En el ejemplo anterior hemos utilizado el método `rawQuery()` para hacer una consulta. Este método tiene una versión alternativa con la misma función el método `query()`. El método `query()` es el descrito en la documentación oficial y, además, es el único disponible en otras clase, como por ejemplo para hacer una consulta en un *ContentProvider*. Sin embargo, tiene un inconveniente respecto al método `rawQuery()`, has de rellenar gran cantidad de parámetros para controlar la búsqueda, lo que lo hace confuso de utilizar. Si estás acostumbrado a trabajar con SQL es posible que este método te resulte incómodo. A continuación se describen los parámetros de ambos métodos:

```
Cursor SQLiteDatabase.query (
    String table,           //tabla a consultar (SELECT)
    String[] columns,       //columnas a devolver (FROM)
    String selection,       //consulta (WHERE)
    String[] selectionArgs,//reemplaza "?" de la consulta
    String groupBy,         //agrupado por (GROUPBY)
    String having,          //condición de f. aritmética
    String orderBy,         //ordenado por
    String limit)           //cantidad máx. de registros

Cursor SQLiteDatabase.rawQuery(
    String sql,                 //comando SQL
    String[] selectionArgs)//reemplaza "?" de la consulta
```

Veamos un ejemplo de cómo se podrían utilizar estos métodos. Supongamos que hemos creado la tabla, `tabla`, y que tiene las columnas `texto`, `entero` y `numero`. Si quisiéramos seleccionar las columnas `texto` y `entero` de las filas con el valor de `numero` mayor que 2, ordenados según el valor de `entero` y que además el número de filas seleccionadas estuviera limitado a un máximo de `cantidad` (donde cantidad ha de ser una variable de tipo entero previamente definida), escribiríamos:

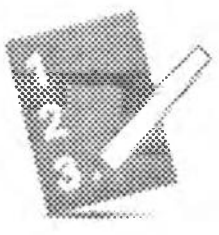
```
Cursor cursor = db.rawQuery("SELECT texto, entero FROM tabla" +
    " WHERE numero>2 ORDER BY entero LIMIT " + cantidad, null);
```

Cuando uno está acostumbrado al lenguaje SQL esta puede ser la forma más sencilla de hacer la consulta. De forma alternativa podemos hacer uso del segundo parámetro. Este ha de ser un *array* de String, de forma que estos Strings remplazan cada una de las apariciones del carácter "?" en la cadena del primer parámetro. Veamos un ejemplo que sería equivalente al anterior:

```
String[] param  = new String[1];
param[0]= Integer.toString(cantidad,10);
Cursor cursor = db.rawQuery("SELECT texto, entero FROM tabla" +
            " WHERE numero>2 ORDER BY entero LIMIT ?", param);
```

Si en lugar del método `rawQuery()` queremos utilizar el método `query()` usaríamos el siguiente código equivalente a los dos anteriores:

```
String[] CAMPOS = {"texto", "entero"};
Cursor cursor = db.query("tabla", CAMPOS, "numero>2", null,
        null, null, "entero", Integer.toString(cantidad));
```



**Ejercicio paso a paso:** *Utilización del método query() para guardar puntuaciones.*

1. Remplaza la llamada al método `rawQuery()` del ejercicio anterior por el siguiente código:

```
String[] CAMPOS = {"puntos", "nombre"};
Cursor cursor=db.query("puntuaciones", CAMPOS, null, null,
            null, null, "puntos", Integer.toString(cantidad));
```

2. Verifica que el funcionamiento es idéntico.

## 9.7. Utilizando la clase ContentProvider

Los proveedores de contenidos (*content providers*) son uno de los bloques constructivos más importantes de Android. Nos van a permitir acceder a información proporcionada por otras aplicaciones, o a la inversa, compartir nuestra información con otras aplicaciones.

Tras describir los principios en los que se basan los *ContentProviders*, pasaremos a demostrar cómo acceder a *ContentProviders* creados por otras aplicaciones. Esta operación resulta muy útil dado que te permitirá tener acceso a información interesante, como la lista de contactos, el registro de llamadas o los ficheros multimedia almacenados en el dispositivo. Terminaremos este apartado mostrando cómo crear tu propio *ContentProvider*, de forma que otras aplicaciones puedan acceder a tu información.

## 9.7.1. Conceptos básicos

### *9.7.1.1.* El modelo de datos

La forma en la que un *ContentProvider* almacena la información es un aspecto de diseño interno, de forma que podríamos utilizar cualquiera de los métodos descritos en este capítulo. No obstante, cuando hagamos una consulta al *ContentProvider* se devolverá un objeto de tipo `Cursor`. Esto hace que la forma más práctica para almacenar la información sea en una base de datos.

Utilizando términos del modelo de bases de datos, un *ContentProvider* proporciona sus datos a través de una tabla simple, donde cada fila es un registro y cada columna es un tipo de datos con un significado particular. Por ejemplo, el *ContentProvider* que utilizaremos en el siguiente ejemplo se llama `CallLog`, y permite acceder al registro de llamadas del teléfono. La información que nos proporciona tiene la estructura que se muestra a continuación:

| _id | date | number | Duration | Type |
|---|---|---|---|---|
| 1 | 12/10/10 16:10 | 555123123 | 65 | INCOMING_TYPE |
| 3 | 12/11/10 20:42 | 555453455 | 356 | OUTGOING_TYPE |
| 4 | 13/11/10 12:15 | 555123123 | 90 | MISSED_TYPE |
| 5 | 14/11/10 22:10 | 555783678 | 542 | OUTGOING_TYPE |

***Tabla 1:*** *Ejemplo de estructura de datos de un ContentProvider.*

Cada registro incluye el campo numérico `_id` que lo identifica de forma única. Como veremos a continuación, podremos utilizar este campo para identificar una llamada en concreto.

La forma de acceder a la información de un *ContentProvider* es muy similar al proceso descrito para las bases de datos. Es decir, vamos a poder realizar una consulta (incluso utilizando el lenguaje SQL), tras la cual se nos proporcionará un objeto de tipo `Cursor`. Este objeto, contiene una información con una estructura similar a la mostrada en la tabla anterior.

### *9.7.1.2.* Las URI

Para acceder a un *ContentProvider* en particular va a ser necesario identificarlo con una URI. Una URI (ver estándar RFC 2396) es una cadena de texto que permite identificar un recurso de información. Suele utilizarse frecuentemente en Internet, por ejemplo para acceder a una página web. Una URI está formada por cuatro partes, tal y como se muestra a continuación:

```
<standard_prefix>://<authority>/<data_path>/<id>
```

La parte `<standard_prefix>` de todos los *ContentProvider* ha de ser siempre `content`. En el primer ejemplo de este apartado, utilizaremos un *ContentProvider* donde Android almacena el registro de llamadas. Para acceder a este *ContentProvider* utilizaremos la siguiente URI:

```
content://call_log/calls
```

En la Tabla 2 encontrarás otros ejemplos de URI que te permitirán acceder a otras informaciones almacenadas en Android.

Para acceder a un elemento concreto has de indicar un `<id>` en la URI. Por ejemplo, si te interesa solo acceder a la llamada con identificador 4 (normalmente corresponderá a la cuarta llamada de la lista), has de indicar:

```
content://call_log/calls/4
```

Un mismo *ContentProvider* puede contener múltiples conjuntos de datos identificados por diferentes URI. Por ejemplo, para acceder a los ficheros multimedia almacenados en el móvil utilizaremos el ContentProvider `MediaStore`, utilizando algunos de los siguientes ejemplos de URI:

```
content://media/internal/images
content://media/external/video/5
content://media/*/audio
```

Cada *ContentProvider* suele definir constantes con sus correspondientes URI.

Por ejemplo, `android.provider.CallLog.Calls.CONTENT_URI` corresponde con `content://call_log/calls`.

Y la constante:

```
android.provider.MediaStore.Audio.Media.INTERNAL_CONTENT_URI.
```

Corresponde con `content://media/internal/audio`.

## 9.7.2. Acceder a la información de un ContentProvider

Android utiliza los *ContentProvider* para almacenar diferentes tipos de información. En la tabla siguiente los *ContentProvider* más importantes disponibles en Android:

| Clase | Información almacenada | Ejemplos de URIs |
|---|---|---|
| Browser | Enlaces favoritos, historial de navegación, historial de búsquedas | content://browser/bookmarks |
| CallLog | Llamadas entrantes, salientes y pérdidas. | content://call_log/calls |
| Contacts | Lista de contactos del usuario. | content://contacts/people |
| MediaStore | Ficheros de audio, vídeo e imágenes, almacenados en dispositivos de almacena-miento internos y externos. | content://media/internal/images<br>content://media/external/video<br>content://media/*/audio |
| Setting | Preferencias del sistema. | content://settings/system/ringtone<br>content://settings/system/notificatio n_sound |

| Clase | Información almacenada | Ejemplos de URIs |
|---|---|---|
| UserDictionary (a partir de 1.5) | Palabras definidas por el usuario, utilizadas en los métodos de entrada predictivos. | content://user_dictionary/words |
| Telephony (a partir de 1.5) | Mensajes SMS y MMS mandados o recibidos desde el teléfono. | content://sms<br>content://sms/inbox<br>content://sms/sent<br>content://mms |
| Calendar (a partir de 4.0) | Permite consultar y editar los eventos del calendario | content://com.android.calendar/time<br>content://com.android.calendar/events |

***Tabla 2:*** *ContentProviders disponibles en Android.*

### *9.7.2.1. Leer información de un ContentProvider*

Veamos un ejemplo que permite leer el registro de llamadas del teléfono. Crea una nueva aplicación con los siguientes datos:

```
Project name: ContentCallLog
Build Target: Android 1.5
Application name: ContentCallLog
Package name: org.example.contentcalllog
Create Activity: ContentCallLog
Min SDK Version: 3
```

Añade en el fichero `res/layout/main.xml` la línea subrayada:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<TextView
    android:id="@+id/salida"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/hello"
    />
</LinearLayout>
```

De esta forma podremos identificar el `TextView` desde el programa y utilizarlo para mostrar la salida.

Añade al final del fichero `AndroidManifest.xml` las líneas subrayadas:

```
    ...
    <uses-permission
      android:name="android.permission.READ_CONTACTS">
    </uses-permission>
</manifest>
```

Al solicitar el permiso `READ_CONTATS` podremos acceder al registro de llamadas. Añade al final del método `onCreate` de la actividad principal el siguiente código:

```
...
String[] TIPO_LLAMADA = {"","entrante","saliente","perdida"};
TextView salida = (TextView) findViewById(R.id.salida);

Uri llamadas = Uri.parse("content://call_log/calls");
Cursor c = managedQuery(llamadas, null, null, null, null);
while (c.moveToNext()) {
   salida.append("\n"
     + DateFormat.format("dd/MM/yy k:mm (",
               c.getLong(c.getColumnIndex(Calls.DATE)))
     + c.getString(c.getColumnIndex(Calls.DURATION)) + ") "
     + c.getString(c.getColumnIndex(Calls.NUMBER)) + ", "
     + TIPO_LLAMADA[Integer.parseInt(c.getString(
                          c.getColumnIndex(Calls.TYPE)))]);
}
```

En primer lugar se define un *array* de *strings*, de forma que `TIPO_LLAMADA[1]` corresponda a `"entrante"`, `TIPO_LLAMADA[2]`, corresponda a `"saliente"`, etc. Luego se crea el objeto `salida` que es asignado al `TextView` correspondiente del *Layout*.

Ahora comienza lo interesante, creamos la URI, `llamadas` asociada a `content://call_log/calls`. Para realizar la consulta creamos el `Cursor`, `c`. Se trata de la misma clase que hemos utilizado para hacer una consulta en una base de datos, pero ahora la información será suministrada por un *ContentProvider* por medio del método `manageQuery()`. Este método permite varios parámetros para indicar exactamente los elementos que nos interesan, de forma similar a como se hace en una base de datos. Estos parámetros serán estudiados más adelante. Al no indicar ninguno se devolverán todas las llamadas registradas.

No tenemos más que desplazarnos por todos los elementos del cursor (`c.moveToNext()`) e ir añadiendo la salida (`salida.append()`) la información correspondiente a cada registro. En concreto fecha, duración, número de teléfono y tipo de llamada. Una vez que el cursor se encuentra situado en una fila determinada, podemos obtener la información de una columna utilizando los métodos `getString()`, `getInt()`, `getLong()` y `getFloat()` dependiendo del tipo de dato almacenado. Estos métodos necesitan como parámetros el índice de columna. Para averiguar el índice de cada columna utilizaremos el método `getColumnIndex()` indicando el nombre de la columna. En nuestro caso estos nombres son `"date"`, `"duration"`, `"number"` y `"type"`. En el ejemplo, en lugar de utilizar estos nombres se han utilizado cuatro constantes definidas con estos valores.

Especial mención tiene la columna `"date"` que nos devuelve un entero largo que representa un instante concreto de una fecha. Para mostrarla en el formato deseado hemos utilizado el método estático `format()` de la clase `DateFormat`.

El resultado de ejecutar este programa se muestra a continuación:


Veamos con más detalle el método `managedQuery()`:

```
Cursor managedQuery(Uri uri, String[] proyeccion, String
seleccion, String[] argsSelecc, String orden)
```

Donde los parámetros corresponden a:

| | |
|---|---|
| `uri` | URI correspondiente al *ContentProvider* a consultar. |
| `proyeccion` | Lista de columnas que queremos que nos devuelva. |
| `seleccion` | Cláusula SQL correspondiente a `WHERE`. |
| `argsSelecc` | Lista de argumentos utilizados en el parámetro `seleccion`. |
| `orden` | Cláusula SQL correspondiente a `ORDER BY`. |

Para ilustrar el uso de estos parámetros remplaza en el ejemplo anterior la línea:

```
Cursor c = managedQuery(llamadas, null, null, null, null);
```

por:

```
String[] proyeccion = new String[] {
    Calls.DATE, Calls.DURATION, Calls.NUMBER, Calls.TYPE };
String[] argsSelecc = new String[] {"1"};
Cursor c = managedQuery(
    llamadas,        // Uri del ContentProvider
    proyeccion,      // Columnas que nos interesan
    "type = ?",      // consulta WHERE
    argsSelecc,      // parámetros de la consulta anterior
    "date DESC");    // Ordenado por fecha, orden ascendente
```

De esta forma nos devolverán solo las columnas indicadas en `proyeccion`. Esto supone un ahorro de memoria en caso de que existan muchas columnas. En el siguiente parámetro se indica las filas que nos interesan por medio de una consulta de tipo `WHERE`. En caso de encontrar algún carácter `?` éste es sustituido por el primer `string` del parámetro `argSelecc`. En caso de haber más caracteres `?` se irían sustituyendo siguiendo el mismo orden. Cuando se sustituyen los interrogantes cada elemento de `argSelecc` es introducido entre comillas. Por lo tanto, en el ejemplo, la consulta `WHERE` resultante es `"WHERE type = '1'"`. Esta consulta implica que solo se mostrarán las llamadas entrantes. El último parámetro sería equivalente a indicar en SQL `"SORTED BY date DESC"`, es decir, el resultado estará ordenado por fecha en orden ascendiente.

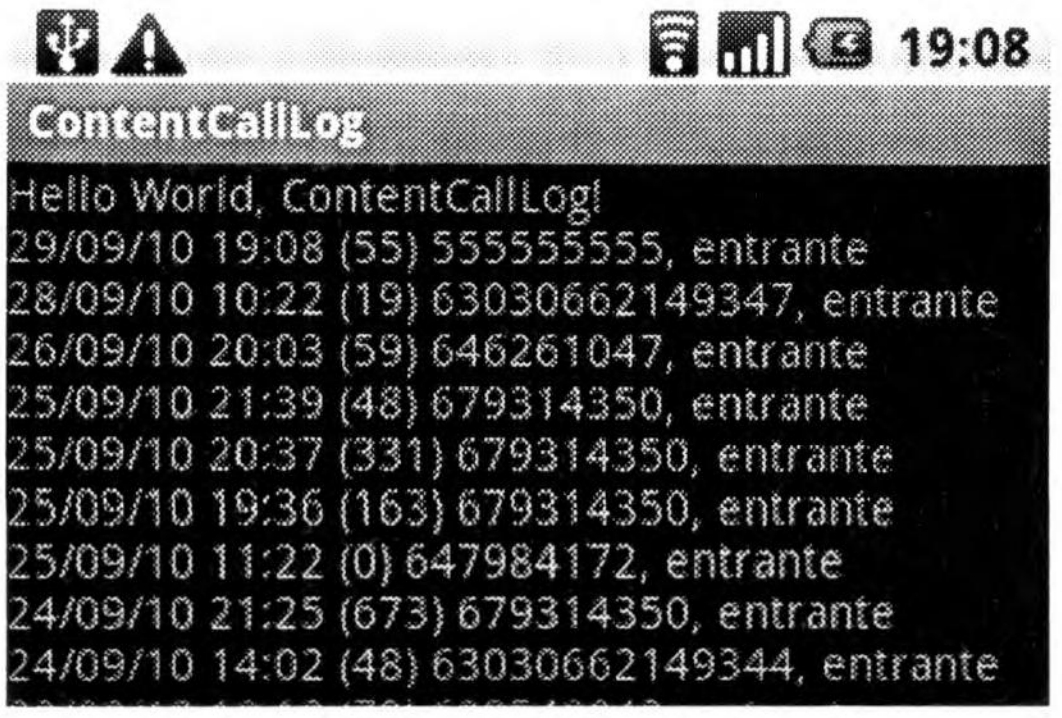
### *9.7.2.2. Escribir información en un ContentProvider*

Añadir un nuevo elemento en un *ContentProvider* resulta muy sencillo. Para ilustrar cómo se hace escribe el siguiente código al principio del ejemplo anterior:

```
ContentValues valores = new ContentValues();
valores.put(Calls.DATE, new Date().getTime() );
valores.put(Calls.NUMBER, "555555555");
valores.put(Calls.DURATION, "55");
valores.put(Calls.TYPE, Calls.INCOMING_TYPE);

Uri nuevoElemento = getContentResolver().insert(
                                    Calls.CONTENT_URI, valores);
```

Como puedes ver comenzamos creando un objeto `ContentValues` donde vamos almacenado una serie de pares de valores, nombre de columna y valor asociado a la columna. A continuación, se llama a `getContentResolver().insert()` pasándole la URI del *ContentProvider* y los valores a insertar. Este método nos devuelve una URI que apunta de forma específica al elemento que acabamos de insertar. Podrías utilizar esta URI para hacer una consulta y obtener un cursor al nuevo elemento y así poder modificarlo, borrarlo u obtener el `_ID`. Recuerda que has de pedir el permiso `WRITE_CONTACTS`.

Si ejecutas ahora el programa, la nueva llamada insertada ha de aparecer en primer lugar.



Estamos modificando el registro de llamadas del sistema, por lo tanto, también puedes verificar esta información desde las aplicaciones del sistema.


### *9.7.2.3. Borrar y modificar elementos de un ContentProvider*

Puedes utilizar el método `delete()` para eliminar elementos de un *ContentProvider*:

```
int ContentProvider.delete(Uri uri, String seleccion,
                           String[] argsSelecc)
```

Este método devuelve el número de elementos eliminados. Los tres parámetros del método se detallan a continuación:

| | |
|---|---|
| `uri` | URI correspondiente al *ContentProvider* a consultar. |
| `seleccion` | Cláusula SQL correspondiente a `WHERE`. |
| `argsSelecc` | Lista de argumentos utilizados en el parámetro `seleccion`. |

Si quisiéramos eliminar un solo elemento podríamos obtener su URI e indicarlo en el primer parámetro, dejando los otros dos a `null`. Si por el contrario quieres eliminar varios elementos puedes utilizar el parámetro `seleccion`. Por ejemplo, si quisiéramos eliminar todos los registros de llamada del número 555555555, escribiríamos:

```
getContentResolver().delete(Calls.CONTENT_URI,
                            "number='555555555'", null);
```

También puedes utilizar el método `update()` para modificar elementos de un *ContentProvider*:

```
int ContentProvider.update(Uri uri, ContentValues valores,
                           String seleccion, String[] argsSelecc)
```

Por ejemplo, si quisiéramos modificar los registros con número 555555555, por el número 444444444, escribiríamos:

```
ContentValues valores2 = new ContentValues();
valores2.put(Calls.NUMBER, "444444444");
getContentResolver().update(Calls.CONTENT_URI, valores2,
                            "number='555555555'", null);
```

## 9.7.3. Creación de un ContentProvider

En este apartado vamos a describir los pasos necesarios para crear tu propio *ContentProvider*. En concreto vamos a seguir tres pasos:

1) Definir una estructura de almacenamiento para los datos. En nuestro caso usaremos una base de datos SQLite.
2) Crear nuestra clase extendiendo `ContentProvider`.
3) Declarar el *ContetProvider* en el `AndroidManifest.xml` de nuestra aplicación.

Si no deseas compartir tu información con otras aplicaciones, no es necesario crear un *ContentProvider*. En este caso resulta mucho más sencillo usar una base de datos directamente, tal y como se ha explicado en el apartado anterior. En este apartado vamos a seguir el mismo ejemplo descrito en los anteriores apartados del capítulo, es decir, vamos a crear un *ContentProvider* que nos permita compartir la lista de puntuaciones con otras aplicaciones. Posiblemente, no se trate de una situación muy realista; es de suponer, que ninguna aplicación esté interesada en esta información. No obstante, hemos pensado que va a resultar más sencillo continuar con el mismo ejemplo.

Crea una nueva aplicación con los siguientes datos:

```
Project name: PuntuacionesProvider
Build Target: Android 1.5
Application name: PuntuacionesProvider
Package name: org.example.puntuacionesprovider
Create Activity: ActividadPrincipal
Min SDK Version: 3
```

### *9.7.3.1. Definir la estructura de almacenamiento del ContentProvider*

El objetivo último de un `ContentProvider` es almacenar información de forma permanente. Por lo tanto, resulta imprescindible utilizar alguno de los mecanismos descritos en este tema, o en el siguiente, para almacenar datos. Como se ha estudiado en el apartado anterior, podemos realizar consultas a un *ContentProvider* de forma similar a una base de datos (podemos hacer consultas SQL y nos devuelve un objeto de tipo `Cursor`). Por lo tanto, la forma más sencilla de almacenar los datos de un *ContentProvider* es en una base de datos. De esta forma, si nos solicitan una consulta SQL, no tendremos más que trasladarla a nuestra base de datos, y el objeto `Cursor` que nos devuelva será el resultado que nosotros devolveremos.

Para crear la base de datos de nuestro *ContentProvider* añade una nueva clase con nombre `PuntuacionesSQLiteHelper` e introduce el siguiente código:

```
public class PuntuacionesSQLiteHelper extends SQLiteOpenHelper {

    public PuntuacionesSQLiteHelper(Context context) {
        super(context, "puntuaciones", null, 1);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL("CREATE TABLE puntuaciones ("
            + "_id INTEGER PRIMARY KEY AUTOINCREMENT, "
```

```
            + "puntos INTEGER, "
            + "nombre TEXT, "
            + "fecha LONG)");
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion,
                                                int newVersion) {
    // En caso de una nueva versión habría que actualizar las tablas
    }
```

La clase que acabamos de añadir al proyecto se encarga de crear la base de datos `puntuaciones` extendiendo la clase `SQLiteOpenHelper`. Este proceso fue descrito en el apartado dedicado a las bases de datos. Dado que estamos trabajando sobre el mismo ejemplo, la tabla creada es idéntica y lo mismo ocurre con el código. Para una explicación más detallada recomendamos consultar este apartado.

### *9.7.3.2. Extendiendo la clase ContentProvider*

Ahora abordamos la parte más laboriosa: la creación de una clase descendiente de `ContentProvider`. Los métodos principales que tenemos que implementar son:

`getType()` – Devuelve el tipo MIME de los elementos del *ContentProvider*.

`query()` – Permite realizar consultas al *ContentProvider*.

`insert()` – Inserta nuevos datos.

`delete()` – Borra elementos del *ContentProvider*.

`update()` – Permite modificar los datos existentes.

La clase `ContentProvider` toma las precauciones necesarias para evitar problemas con las llamadas simultáneas de varios procesos, por lo tanto, en la creación de una subclase no nos tenemos que preocupar de este aspecto.

Añade una nueva clase con nombre `PuntuacionesProvider` e introduce el siguiente código:

```
public class PuntuacionesProvider extends ContentProvider {

  public static final String AUTORIDAD =
                      "org.example.puntuacionesprovider";
  public static final Uri CONTENT_URI = Uri.parse("content://"
                          + AUTORIDAD + "/puntuaciones");
  public static final int TODOS_LOS_ELEMENTOS = 1;
  public static final int UN_ELEMENTO = 2;
  private static UriMatcher URI_MATCHER = null;
  static {
      URI_MATCHER = new UriMatcher(UriMatcher.NO_MATCH);
      URI_MATCHER.addURI(AUTORIDAD, "puntuaciones",
                                        TODOS_LOS_ELEMENTOS);
```

```
        URI_MATCHER.addURI(AUTORIDAD, "puntuaciones/#", UN_ELEMENTO);
    }
    public static final String TABLA = "puntuaciones";
    private SQLiteDatabase baseDeDatos;

    @Override
    public boolean onCreate() {
        PuntuacionesSQLiteHelper dbHelper = new
                        PuntuacionesSQLiteHelper(getContext());
        baseDeDatos = dbHelper.getWritableDatabase();
        return baseDeDatos != null && baseDeDatos.isOpen();
    }
}
```

Como no podría ser de otra forma la clase extiende `ContentProvider`. A continuación creamos constantes, `AUTORIDAD` y `Uri`, que identificarán nuestro *ContentProvider* mediante la URI:

```
content://org.example.puntuacionesprovider/puntuaciones
```

Las siguientes líneas permiten crear el objeto estático `URI_MATCHER` de la clase `UriMatcher`. En Java es habitual utilizar variables estáticas finales para albergar objetos que no van a cambiar en toda la vida de la aplicación, es decir que son constantes. Conviene recordar que solo se creará un objeto `URI_MATCHER` aunque se instancie varias veces la clase `PuntuacionesProvider`. La clase `UriMatcher` permite diferenciar entre distintos tipos de URI que vamos a manipular. En nuestro caso permitimos dos tipos de URI: acabada en `/puntuaciones`, que identifica todas las puntuaciones almacenadas y acabada en `/puntuaciones/#`, donde # hay que reemplazarlo por un código numérico que coincida con el campo `_id` de nuestra tabla. Cada tipo de URI ha de tener un código numérico asociado; en nuestro caso `NO_MATCH` (-1), `TODOS_LOS_ELEMENTOS` (1) y `UN_ELEMENTO` (2).

La declaración de variables termina con la constante `TABLA`, que identifica la tabla que usaremos para almacenar la información y el objeto `baseDeDatos` donde se almacenará la información.

A continuación está el método `onCreate()` que es llamado cuando se crea una instancia de esta clase. Básicamente, se crea un `SQLiteHelper` a partir de la clase descrita en el apartado anterior (`PuntuacionesSQLiteHelper`) y se asigna la base de datos resultante a la variable `baseDeDatos`. Devolvemos `true` solo en el caso de que no haya habido problemas en su creación.

Veamos la implementación del método `getType()` que a partir de una URI nos devuelve el tipo MIME que le corresponde:

```
@Override
public String getType(final Uri uri) {
    switch (URI_MATCHER.match(uri)) {
    case TODOS_LOS_ELEMENTOS:
        return "vnd.android.cursor.dir/vnd.org.example.puntuacion";
    case UN_ELEMENTO:
```

```
            return "vnd.android.cursor.item/vnd.org.example.puntuacion";
        default:
            throw new IllegalArgumentException("URI incorrecta: " + uri);
        }
    }
```

***NOTA:*** *Los tipos MIME fueron creados para identificar un tipo de datos concreto que añadíamos como anexo a un correo electrónico, aunque en la actualidad son utilizados por muchos sistemas y protocolos. Un tipo MIME tiene dos partes* `tipo_genérico/tipo_específico`, *por ejemplo* `image/gif`, `image/jpeg`, `text/html`, *etc.*

Existe un convenio para identificar los tipos MIME que proporciona un *ContentProvider*. Si se trata de un recurso único, utilizamos:

`vnd.android.cursor.item/vnd.`*`ELEMENTO`*

y si se trata de una colección de recursos utilizamos:

`vnd.android.cursor.dir/vnd.`*`ELEMENTO`*

donde *`ELEMENTO`* ha de ser remplazado por un identificador que describa el tipo de datos. En nuestro caso hemos elegido `org.example.puntuacion`. Utilizar prefijos para definir el elemento minimiza el riesgo de confusión con otro ya existente.

A continuación hemos de sobrescribir los métodos `query`, `insert`, `delete` y `update` que permitan consultar, insertar, borrar y actualizar elementos de nuestro *ContentProvider*. Comencemos por el primer método:

```
@Override
public Cursor query(Uri uri, String[] projection, String seleccion,
                    String[] argSeleccion, String orden) {
    SQLiteQueryBuilder queryBuilder = new SQLiteQueryBuilder();
    queryBuilder.setTables(TABLA);
    switch (URI_MATCHER.match(uri)) {
    case TODOS_LOS_ELEMENTOS:
        break;
    case UN_ELEMENTO:
        String id = uri.getPathSegments().get(1);
        queryBuilder.appendWhere("_id = " + id);
        break;
    default:
        throw new IllegalArgumentException("URI incorrecta "+uri);
    }
    return queryBuilder.query(baseDeDatos, proyeccion, seleccion,
                              argSeleccion, null, null, orden);
}
```

El método `query()` que hemos de implementar tiene parámetros similares que `SQLiteQueryBuilder.query()`. Por lo tanto, no tenemos más que trasladar la consulta a nuestra base de datos. No obstante existe un pequeño inconveniente, si

nos pasan una URI que identifique un solo elemento, como la mostrada a continuación:

```
content://org.example.puntuacionesprovider/puntuaciones/324
```

Hemos de asegurarnos que solo se devuelve el elemento con `_id`=324. Para conseguirlo se introduce una cláusula `switch`, que en caso de tratarse de una URI de tipo `UN_ELEMENTO`, añada a la cláusula `WHERE` de la consulta la condición correspondiente mediante el método `apendWhere()`. La cláusula `WHERE` puede tener más condiciones si se ha utilizado el parámetro `seleccion`.

```
@Override
public Uri insert(Uri uri, ContentValues valores) {
   long IdFila = baseDeDatos.insert(TABLA, null, valores);
   if (IdFila > 0) {
       return ContentUris.withAppendedId(CONTENT_URI, IdFila);
   } else {
       throw new SQLException("Error al insertar registro en "+uri);
   }
}

@Override
public int delete(Uri uri, String seleccion, String[] argSeleccion) {
   switch (URI_MATCHER.match(uri)) {
   case TODOS_LOS_ELEMENTOS:
       break;
   case UN_ELEMENTO:
       String id = uri.getPathSegments().get(1);
       if (TextUtils.isEmpty(seleccion)) {
        seleccion = "_id = " + id;
       } else {
        seleccion = "_id = " + id + " AND (" + seleccion + ")";
       }
       break;
   default:
       throw new IllegalArgumentException("URI incorrecta: " + uri);
   }
   return baseDeDatos.delete(TABLA, seleccion, argSeleccion);
}
```

El método `insert()` no requiere explicaciones adicionales.

El método `delete()`, igual como ocurrió con el método `query()`, presenta el inconveniente de que pueden habernos indicado una URI que identifique un solo elemento (…/`puntuaciones/324`). En el método `query()`, solucionamos este problema llamando a `SQLiteQueryBuilder.appendWhere()`. Sin embargo, ahora no disponemos de un objeto de esta clase, por lo que nos vemos obligados a realizar este trabajo a mano. En caso de no haberse indicado nada en `seleccion`, este parámetro valdrá `"_id = 324"`; y en caso de haberse introducido una condición, por ejemplo `"numero = '555'"`, `seleccion` valdrá `"_id = 324 AND (numero = '555')"`.

```
@Override
public int update(Uri uri, ContentValues valores, String seleccion,
                                        String[] ArgumentosSeleccion) {
   switch (URI_MATCHER.match(uri)) {
   case TODOS_LOS_ELEMENTOS:
       break;
   case UN_ELEMENTO:
       String id = uri.getPathSegments().get(1);
       if (TextUtils.isEmpty(seleccion)) {
        seleccion = "_id = " + id;
       } else {
        seleccion = "_id = " + id + " AND (" + seleccion + ")";
       }
       break;
   default:
       throw new IllegalArgumentException("URI incorrecta: " + uri);
   }
   return baseDeDatos.update(TABLA, valores, seleccion,
                                              ArgumentosSeleccion);
}
```

Finalizamos con el método `update()` que es muy similar a `delete()`.

### *9.7.3.3. Declarar el ContentProvider en AndroidManifest.xml*

Si queremos que nuestro `ContentProvider` sea visible a otras aplicaciones resulta imprescindible hacérselo saber al sistema declarándolo en el `AndroidManifest.xml`. Para conseguirlo no tienes más que añadir el siguiente código dentro de la etiqueta `<application>`:

```
<provider
  android:authorities="org.example.puntuacionesprovider"
  android:name="org.example.puntuacionesprovider.PuntuacionesProvider"
/>
```

La declaración de un `ContentProvider` requiere que se especifiquen dos atributos:

`name`: nombre cualificado de la clase donde hemos implementado nuestro *ContentProvider*.

`authorities`: parte de correspondiente a la autoridad de las URI que vamos a publicar. Puede indicarse más de una autoridad.

También se pueden indicar otros atributos en la etiqueta `<provider>`. Veamos los más importantes:

`label`: etiqueta describiendo el *ContentProvider* que será mostrada al usuario. Esta etiqueta como una referencia a un recurso de tipo *string*.

`icon`: una referencia a un recurso de tipo *drawable* con un icono que represente nuestro *ContentProvider*.

`enable`: indica si está habilitado. El valor por defecto es `true`.

`exported`: indica si puede ser accedido desde otras aplicaciones. Valor por defecto `true`. Si está a `false` solo podrá ser utilizado por aplicaciones con el mismo ID de usuario que la aplicación donde se crea.

`readPermision`: indica si los clientes podrán consultar la información.

`writePermision`: indica si los clientes podrán modificar la información.

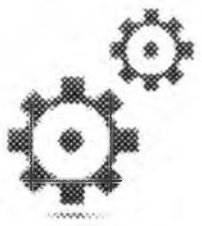
`permission`: el nombre de un permiso que el cliente ha de solicitar para acceder al *ContentProvider*. Para más información consultar el capítulo sobre seguridad.

`multiprocess`: el valor por defecto es `false`, en este caso se creará un único proceso para contener el *ContentProvider*. Si se indica `true`, se creará un nuevo proceso por cada cliente que quiera hacer uso del *ContentProvider*.

`process`: nombre del proceso en el que el *ContentProvider* ha de ejecutarse. Habitualmente, todos los componentes de una aplicación se ejecutan en el mismo proceso creado para la aplicación. Si no se indica lo contrario, el nombre del proceso coincide con el nombre del paquete de la aplicación (si lo deseas puedes cambiar este nombre con el atributo `process` de la etiqueta `<aplication>`). Si prefieres que un componente de la aplicación se ejecute en su propio proceso, has de utilizar este atributo.

`initOrder`: orden en que el *ContentProvider* ha de ser instalado en relación a otros *ContentProviders* del mismo proceso.

`syncable`: indica si la información del *ContentProvider* está sincronizada con el servidor.



**Práctica:** *La etiqueta provider.*

1. Trata de introducir nuevos atributos a la etiqueta `<provider>`.
2. Observa los resultados obtenidos.

## 9.7.4. Acceso a PuntuacionesProvider desde Asteroides

Una vez hemos creado y declarado nuestro *ContentProvider* vamos a probarlo desde la aplicación Asteroides. Como hemos hecho en ejemplos anteriores, vamos a crear una nueva clase que implemente la interfaz `AlmacenPuntuaciones`.

Crea una nueva clase en la aplicación `Asteroides` con el nombre `AlmacenPuntuacionesProvider`. Introduce el siguiente código:

```
public class AlmacenPuntuacionesProvider
                                    implements AlmacenPuntuaciones {
private Activity activity;
```

```
    public AlmacenPuntuacionesProvider(Activity activity) {
        this.activity = activity;
    }

    public void guardarPuntuacion(int puntos, String nombre, long fecha) {
        Uri uri = Uri.parse(
                "content://org.example.puntuacionesprovider/puntuaciones");
        ContentValues valores = new ContentValues();
        valores.put("nombre", nombre);
        valores.put("puntos", puntos);
        valores.put("fecha", fecha);
        try {
            activity.getContentResolver().insert(uri, valores);
        } catch (Exception e) {
            Toast.makeText(
                    activity,
                    "Verificar que está instalado "+
                            "org.example.puntuacionesprovider",
                    Toast.LENGTH_LONG).show();
            Log.e("Asteroides", "Error: " + e.toString(), e);
        }
    }

    public Vector<String> listaPuntuaciones(int cantidad) {
        Vector<String> result = new Vector<String>();
        Uri uri = Uri.parse(
                "content://org.example.puntuacionesprovider/puntuaciones");
        Cursor cursor = activity.managedQuery(uri, null, null,
                                              null, "fecha DESC");
        if (cursor != null) {
            while (cursor.moveToNext()) {
                String nombre = cursor.getString(
                        cursor.getColumnIndex("nombre"));
                int puntos = cursor.getInt(
                        cursor.getColumnIndex("puntos"));
                result.add(puntos + " " + nombre);
            }
            cursor.close();
        }
        return result;
    }
}
```

Para utilizar esta nueva clase, modifica en el principio de la clase `Asteroides` y de la clase `Puntuaciones` la declaración de la variable `almacen` por:

```
AlmacenPuntuaciones almacen = new AlmacenPuntuacionesProvider(this);
```

# CAPÍTULO 10.
# Internet: *sockets*, HTTP y servicios web

Los teléfonos Android suelen disponer de conexión a Internet. Esto nos permite no solo almacenar los datos en nuestro dispositivo, sino compartirlos con otros usuarios. En el primer punto del capítulo trataremos de resolver el problema de comunicar dos aplicaciones en Internet mediante la herramienta básica: los *sockets*. Existen otras alternativas de más alto nivel, como el uso del protocolo HTTP, que será estudiada en el segundo punto del capítulo. Para terminar, se tratará una tercera alternativa, todavía de más alto nivel, los servicios web.

En este capítulo implementaremos el mismo ejemplo que en el capítulo anterior, es decir, almacenaremos las puntuaciones obtenidas en Asteroides, pero ahora en un servidor de Internet. Utilizaremos las tres alternativas descritas en el párrafo anterior. No obstante, has de tener claro que estos mecanismos están relacionados entre sí. Por ejemplo, si utilizas servicios web, internamente se utilizará el protocolo HTTP y además este protocolo utiliza *sockets* para establecer la comunicación.

**Objetivos:**

- Repasar las alternativas principales para intercambiar datos por Internet.
- Describir el uso de socket como herramienta básica de una aplicación para comunicarse con otras aplicaciones por Internet.
- Mostrar cómo se programaría un protocolo basado en sockets sobre TCP.
- Describir el funcionamiento del protocolo HTTP
- Mostrar cómo se pueden programa peticiones HTTP desde Android.
- Definir el concepto de servicio Web y comparar las alternativas más importantes.
- Mostrar el acceso a servicios web de terceros desde un cliente Android.
- Aprender a crear nuestro propio servidor de servicios Web.

# 10.1. Comunicaciones en Internet mediante *sockets*

Antes de definir que es un *socket* conviene aclarar los roles o configuraciones que pueden tomar las aplicaciones en un proceso de comunicación. Las dos configuraciones más importantes que pueden tomar son: las llamadas "arquitectura igual a igual" y la "arquitectura cliente/servidor". Esta segunda es la que utilizaremos en los ejemplos de este capítulo, por tanto, conviene aclarar este aspecto.

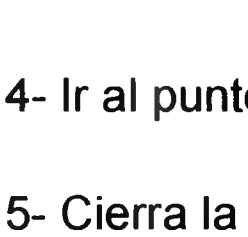
## 10.1.1. La arquitectura cliente/servidor

Las aplicaciones en Internet suelen seguir la arquitectura cliente/servidor. Esta arquitectura se caracteriza por descomponer el trabajo en dos partes (es decir dos programas): el servidor, que centraliza el servicio, y el cliente, que controla la interacción con el usuario. El servidor ha de ofrecer sus servicios a través de una dirección conocida. Algunos ejemplos de aplicaciones basadas en la arquitectura cliente/servidor son WWW o el correo electrónico. Se suelen seguir las siguientes pautas de comportamiento en esta arquitectura:

| Cliente | Servidor |
|---|---|
| 1- Se conecta al servidor. | 1- A la espera de que algún cliente se conecte. |
| 2- Solicita alguna información al servidor. | 2- Recibe solicitud. |
| 3- Recibe la respuesta. | 3- Envía respuesta. |
| 4- Ir al punto 2. | 4- Ir al punto 2. |
| 5- Cierra la conexión | 5- Cierra la conexión. |
| | 6- Ir al punto 1. |

## 10.1.2. ¿Qué es un *socket*?

Cada una de las diferentes aplicaciones en Internet (web, correo electrónico,...) ha de poder intercambiar información entre programas situados en diferentes ordenadores o dispositivos. Con este propósito, se va a hacer uso del nivel de transporte de la pila de protocolos TCP/IP, cuyo objetivo final es permitir el intercambio de información a través de la red de forma fiable y transparente.



**Polii[Media]**: *El Interfaz Socket.*

La interfaz *socket* define las reglas que un programa ha de seguir para utilizar los servicios del nivel de transporte en una red TCP/IP. Esta interfaz se basa en el concepto de *socket*. Un *socket* es el punto final de una comunicación bidireccional

entre dos programas que intercambian información a través de Internet (*socket* se traduce literalmente como enchufe).

Dado que en un mismo dispositivo/ordenador podemos estar ejecutando de forma simultánea diferentes aplicaciones que utilizan Internet para comunicarse, resulta imprescindible identificar cada *socket* con una dirección diferente. Un *socket* se va a identificar por la dirección IP del dispositivo donde está, mas un número de puerto (de 16 bits). En Internet se suele asociar a cada aplicación un número de puerto concreto (por ejemplo: 80 para la web, 25 para el correo electrónico, 7 para ECHO o 4661 para eDonkey). Una conexión está determinada por un par de *sockets*, que son los extremos de la conexión.

Existen dos tipos de *socket*, *socket stream* y *socket datagram*. Veamos en qué se diferencian:

### *10.1.2.1.Sockets stream (TCP)*

Los *sockets stream* ofrecen un servicio orientado a conexión, donde los datos se transfieren como un flujo continuo sin encuadrarlos en registros o bloques. Este tipo de *socket* se basa en el protocolo TCP, que es un protocolo orientado a conexión. Esto implica que, antes de transmitir información, hay que establecer una conexión entre los dos *sockets*. Mientras uno de los *sockets* atiende peticiones de conexión (servidor), el otro solicita la conexión (cliente). Una vez que los dos *sockets* están conectados, ya se puede transmitir datos en ambas direcciones. El protocolo incorpora al programador, de forma transparente, la corrección de errores. Es decir, si detecta que parte de la información no llegó a su destino correctamente, ésta volverá a ser trasmitida. Además, no limita el tamaño máximo de información a transmitir.

### *10.1.2.2.Sockets datagram (UDP)*

Los *sockets datagram* se basan en el protocolo UDP y ofrecen un servicio de transporte sin conexión. Es decir, podemos mandar información a un destino sin necesidad de realizar una conexión previa. El protocolo UDP es más eficiente que TCP, pero tiene el inconveniente que no se garantiza la fiabilidad. Además, los datos se envían y reciben en datagramas (paquetes de información) de tamaño limitado. La entrega de un datagrama no está garantizada: estos pueden ser duplicados, perdidos o llegar en un orden diferente al que se envió.

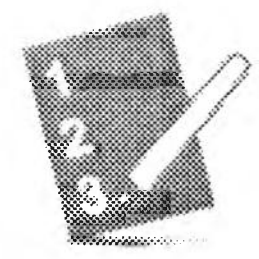
La gran ventaja de este tipo de *sockets* es que apenas introduce sobrecarga sobre la información transmitida. Además, los retrasos introducidos son mínimos, lo que los hace especialmente interesantes para aplicaciones en tiempo real, como la transmisión de audio y vídeo sobre Internet. Sin embargo, presenta muchos inconvenientes al programador: cuando transmitimos un datagrama no tenemos la certeza de que este llegue a su destino, por lo que, si fuera necesario, tendríamos que implementar nuestro propio mecanismo de control de errores. Otro inconveniente es el hecho de que existe un tamaño máximo de datagrama, unos 1.500 bytes dependiendo de la implementación. Si la información a enviar es mayor, tendríamos que fraccionarla y enviar varios datagramas independientes. En el destino tendríamos que concatenarlos en el orden correcto.

En conclusión, si deseas una comunicación libre de errores y sin preocupaciones para el programador es más conveniente que utilices *sockets stream*. Es el tipo de *socket* que utilizaremos en los siguientes ejemplos.

### 10.1.3. Un ejemplo de un cliente/servidor de ECHO

El servicio ECHO suele estar instalado en el puerto 7 de máquinas Unix y permite comprobar que la máquina está operativa y que se puede establecer una conexión con ella.

El funcionamiento de un servidor ECHO es muy sencillo: cuando alguien se conecta espera que le envíe algo y le responde exactamente con la misma información recibida. El cliente actúa de forma contraria; envía datos al servidor y luego comprueba que los datos recibidos son idénticos a los transmitidos.



**Ejercicio paso a paso:** *Un cliente de ECHO.*

El siguiente ejemplo muestra cómo podrías desarrollar un cliente de ECHO que utiliza un *socket stream*.

> ***NOTA:*** *Para que el ejemplo funcione en la dirección IP* `158.42.146.127` *ha de haber un servidor de ECHO en funcionamiento. En caso de no ser así, puedes remplazar la IP por la de un servidor de ECHO en funcionamiento o realizar el siguiente ejercicio.*

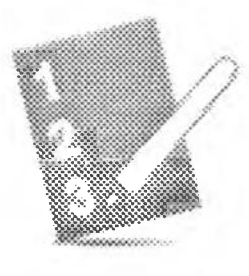
1. Crea la aplicación *ClienteECHO*, donde se cree una actividad con nombre `ClienteECHO`.
2. Remplaza el código por:

```
public class ClienteECHO extends Activity {
  private extView output;

  @Override public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    output = (TextView) findViewById(R.id.TextView01);
    ejecutaCliente();
  }

  private void ejecutaCliente() {
    String ip="158.42.146.127";
    int puerto=7;
    log(" socket " + ip + " " + puerto);
    try {
        Socket sk = new Socket(ip,puerto);
        BufferedReader entrada = new BufferedReader(
            new InputStreamReader(sk.getInputStream()));
```

```
            PrintWriter salida = new PrintWriter(
                new OutputStreamWriter(sk.getOutputStream()),true);
            log("enviando...");
            salida.println("Hola Mundo");
            log("recibiendo ... " + entrada.readLine());
            sk.close();
        } catch (Exception e) {
            log("error: " + e.toString());
        }
    }

    private void log(String string) {
        output.append(string + "\n");
    }
}
```

La parte interesante se encuentra en el método `ejecutarCliente()`. En primer lugar, todo cliente ha de conocer la dirección del *socket* del servidor; en este caso los valores se indican en el par de variables `ip` y `puerto`. Nunca tenemos la certeza de que el servidor admita la conexión, por lo que es obligatorio utilizar una sección `try/catch`. La conexión propiamente dicha se realiza con el constructor de la clase `Socket`. Siempre hay que tener previsto que ocurra algún problema, en tal caso, se creará la excepción y se pasará a la sección `catch`. En caso contrario, continuaremos obteniendo el `InputStream` y el `OutputStream` asociado al *socket*. Lo cual nos permitirá obtener las variables, `entrada` y `salida` mediante las que podremos recibir y transmitir información. El programa transmite la cadena `"Hola Mundo"`, tras lo que visualiza la información recibida. Si todo es correcto, ha de coincidir con lo transmitido.

3. Solicita en la aplicación el permiso INTERNET.
4. Ejecuta la aplicación y verifica si el servidor responde. Recuerda que es posible que no se haya arrancado este servicio en el servidor.



**Ejercicio paso a paso:** *Un servidor de ECHO.*

Vamos a implementar un servidor de ECHO en tu ordenador.

1. Crea un nuevo proyecto Java (no para Android) con nombre *ServidorECHO*.
2. Crea en este proyecto la clase `ServidorECHO` con el siguiente código:

```
public class ServidorECHO {

    public static void main(String args[]) {
        try {
            ServerSocket sk = new ServerSocket(7);
            while (true) {
```

```
                Socket cliente = sk.accept();
                BufferedReader entrada = new BufferedReader(
                    new InputStreamReader(cliente.getInputStream()));
                PrintWriter salida = new PrintWriter(
                    new OutputStreamWriter(cliente.getOutputStream()),true);
                String datos = entrada.readLine();
                salida.println(datos);
                cliente.close();
            }
        } catch (IOException e) {
            System.out.println(e);
        }
    }
}
```

En este caso utilizaremos la clase `ServerSocket` asociada al puerto 7 para crear un *socket* que acepta conexiones. Luego, se introduce un bucle infinito para que el servidor esté perpetuamente en servicio. En la siguiente línea el servidor no se detendrá hasta que un cliente se conecte. Cuando ocurra esto, todo el intercambio de información se realizará a través de un nuevo *socket* creado con este propósito, el *socket* `cliente`. El resto del código es similar al cliente, aunque en este caso primero se recibe y luego se transmite lo mismo que ha recibido.

3. Sustituye la dirección IP en *ClienteECHO* por la IP de tu ordenador. El comando `ipconfig` (Windows) te permite averiguar la dirección IP de tu ordenador. No utilices como IP 127.0.0.1 (*localhost*) dado que, aunque se ejecuten en la misma máquina, la IP del emulador es diferente a la del PC.
4. Ejecuta primero el servidor y luego el cliente para verificar que funciona.

***NOTA:*** *Si la IP de tu ordenador es privada no podrás crear un servidor accesible desde cualquier parte de Internet. En este caso, utiliza el emulador o un dispositivo Android real que se conecte por WiFi a la misma red de tu ordenador. Si no, el terminal no encontrará el servidor.*

### 10.1.4. Un servidor por *sockets* para las puntuaciones

Siguiendo la estructura básica de un cliente y un servidor TCP que acabamos de ver, va a resultar muy sencillo implementar la interfaz `AlmacenPuntuaciones` para que varios clientes con dispositivos Asteroides puedan conectarse a un servidor para intercambiar puntuaciones.

El primer lugar tenemos que diseñar un protocolo que permita realizar las dos operaciones que ha de implementar la interfaz. Un posible ejemplo de interacción cliente/servidor se muestra a continuación:

Para consultar puntuaciones:

Cliente: `PUNTUACIONES\n`

Servidor: `19000 Pedro Perez\n`
`17500 María Suarez\n`
`13000 Juan García\n`

Para almacenar puntuaciones:

Cliente: `32000 Eva Gutierrez\n`

Servidor: `OK\n`

En segundo lugar, hay que elegir un número de puerto para realizar la comunicación; por ejemplo el 1234.

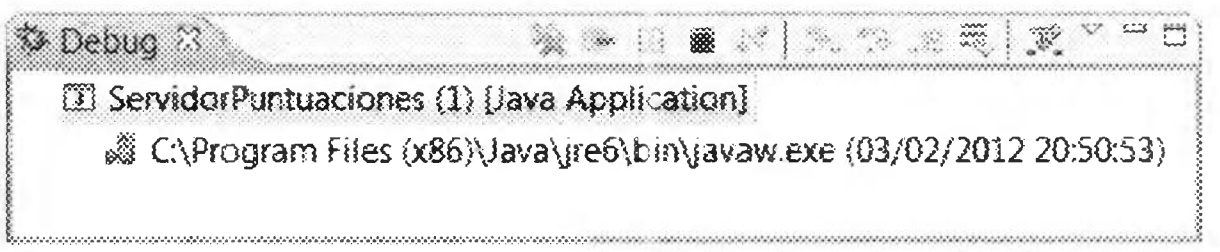
**Ejercicio paso a paso:** *Almacenando las puntuaciones mediante un protocolo basado en sockets.*

1. Crea un nuevo proyecto Java para el servidor (Archivo/Nuevo/Proyecto.../ Proyecto Java) con nombre ServidorPuntuacionesSocket.
2. Pulsa con el botón derecho en la carpeta *src* de este proyecto y selecciona *Nueva/Clase*. Introduce como nombre *ServidorPuntuaciones*.
3. Remplaza en código por el que se muestra a continuación:

```
public class ServidorPuntuaciones {

    public static void main(String args[]) {
        Vector<String> puntuaciones = new Vector<String>();

        try {
            ServerSocket s = new ServerSocket(1234);
            System.out.println("Esperando conexiones...");

            while (true) {
                Socket cliente = s.accept();
                BufferedReader entrada = new BufferedReader(
                        new InputStreamReader(cliente.getInputStream()));
                PrintWriter salida = new PrintWriter(
                        new OutputStreamWriter(cliente.getOutputStream()),
                                                                    true);
                String datos = entrada.readLine();
                if (datos.equals("PUNTUACIONES")){
                    for (int n=0; n<puntuaciones.size(); n++ ){
                        salida.println(puntuaciones.get(n));
                    }
                } else {
                    puntuaciones.add(0, datos);
                    salida.println("OK");
```

```
                }
                cliente.close();
            }
        } catch (IOException e) {
            System.out.println(e);
        }
    }
}
```

4. Ejecuta el proyecto como Aplicación Java.
5. Verifica que en la vista *Consola* aparece: `"Esperando conexiones..."`
6. Cambia la perspectiva a modo *Debug* y Observa la vista de la esquina superior izquierda.




Desde esta vista podrás detener la aplicación pulsando el cuadro rojo.

***NOTA:*** *Si ejecutas de nuevo la aplicación dará un error. Esto es debido a que la aplicación ya lanzada no es detenida y esta aplicación tiene asociado el puerto 1234. El sistema no permitirá que una nueva aplicación escuche este puerto.*

7. Abre el proyecto Asteroides y crea la siguiente clase:

```
public class AlmacenPuntuacionesSocket implements AlmacenPuntuaciones{

    public void guardarPuntuacion(int puntos, String nombre, long fecha){
        try {
            Socket sk = new Socket("X.X.X.X", 1234);
            BufferedReader entrada = new BufferedReader(
                    new  InputStreamReader(sk.getInputStream()));
            PrintWriter salida = new PrintWriter(
                    new OutputStreamWriter(sk.getOutputStream()),true);
            salida.println(puntos + " " + nombre);
            String respuesta = entrada.readLine();
            if (!respuesta.equals("OK")) {
                Log.e("Asteroides", "Error: respuesta de servidor
                                                    incorrecta");
            }
            sk.close();
        } catch (Exception e) {
            Log.e("Asteroides", e.toString(), e);
        }
    }
```

```
public Vector<String> listaPuntuaciones(int cantidad) {
   Vector<String> result = new Vector<String>();
   try {
      Socket sk = new Socket("X.X.X.X", 1234);
      BufferedReader entrada = new BufferedReader(
            new InputStreamReader(sk.getInputStream()));
      PrintWriter salida = new PrintWriter(
            new OutputStreamWriter(sk.getOutputStream()),true);
      salida.println("PUNTUACIONES");
      int n = 0;
      String respuesta;
      do {
         respuesta = entrada.readLine();
         if (respuesta != null) {
            result.add(respuesta);
            n++;
         }
      } while (n < cantidad && respuesta != null);
      sk.close();
   } catch (Exception e) {
      Log.e("Asteroides", e.toString(), e);
   }
   return result;
}
```

8. Has de sustituir las dos apariciones de `"X.X.X.X"` por la dirección IP donde esté ejecutándose el servidor.

   ***NOTA:*** *El comando* `ipconfig` *(Windows) te permite averiguar la dirección IP de tu ordenador. No utilices como IP 127.0.0.1 (localhost) dado que, aunque se ejecuten en la misma máquina, la IP del emulador es diferente a la del PC.*

9. También has de recordar que ahora la aplicación `Asteroides` necesita el permiso `INTERNET`.

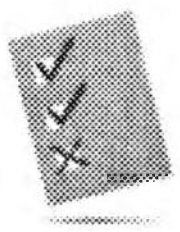
10. Ejecuta Asteroides y accede a visualizar la lista de puntuaciones y luego echa una partida nueva.

11. Verifica que en la vista *Consola* aparecen las consultas.

12. Para terminar, reemplaza la IP por la siguiente `"158.42.146.127"` para conectarte a un servidor compartido.

    ***NOTA:*** *Es posible que este servicio no haya sido iniciado.*

13. Comprueba si otros usuarios han accedido a este servidor y aparecen sus puntuaciones.



**Preguntas de repaso y reflexión:** *El interfaz socket.*

## 10.2. La web y el protocolo HTTP

Dentro del mundo de Internet destaca una aplicación, que es con mucho la más utilizada: Word Wide Web o, coloquialmente, WWW; nos referiremos a esta aplicación como la Web. Su gran éxito se debe a la facilidad de uso, dado que simplifica el acceso a todo tipo de información y a que esta información es presentada de forma atractiva. Básicamente, la web nos ofrece un servicio de acceso a información distribuida en miles de servidores en todo Internet, que nos permite ir navegando por todo tipo de documentos multimedia gracias a un sencillo sistema de hipervínculos.

Para la comunicación entre los clientes y los servidores de esta aplicación, se emplea el protocolo HTTP (*HyperText Transfer Protocol*) que será el objeto de estudio de este apartado.

### 10.2.1. El protocolo HTTP

HTTP es un sencillo protocolo cliente-servidor que articula los intercambios de información entre los navegadores web y los servidores web. Fue propuesto por Tim Berners-Lee, atendiendo a las necesidades de un sistema global de distribución de información como la World Wide Web. En la web los servidores han de escuchar en el puerto 80, esperando la conexión de algún cliente Web.

**Polii[Media]:** *El protocolo http.*

A continuación describimos los pasos habituales que se siguen en un ejemplo de interacción en este protocolo:

1) El usuario quiere acceder a la página http://www.upv.es/dir/pag.html. Para lo cual pulsa en un enlace de un documento HTML o la introduce directamente en el campo *Dirección* del navegador Web.
2) El navegador averigua la dirección IP de www.upv.es.
3) El navegador establece una conexión con el puerto 80 de esta IP.
4) El navegador envía por esta conexión (↵ carácter de salto de línea):

```
GET /dir/pag.html ↵
```

5) El servidor envía la página a través de la conexión:

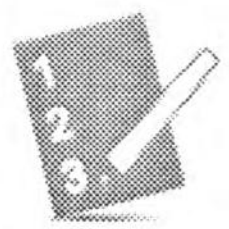
```
<HTML> ↵
<HEAD> ↵
<TITLE>Página de ... </TITLE> ↵
```

```
...
</HTML>
```

6) El servidor cierra la conexión.

Este proceso se repite cada vez que el navegador necesita un fichero del servidor. Por ejemplo, si se ha bajado un documento HTML en cuyo interior están insertadas cuatro imágenes, el proceso anterior se repite un total de cinco veces, una para el documento HTML y cuatro para las imágenes.

Como ves, se trata de un protocolo sin estado. Cada petición contiene la información necesaria para ser atendida. Si deseamos mantener un estado tendrá que ser implementado usando algún mecanismo adicional (por ejemplo las *cookies*).



**Ejercicio paso a paso:** *Estudio del protocolo HTTP utilizando el comando telnet.*

1. Abre un navegador web y accede a la página: (en caso de que el servidor no responda, puedes realizar el ejercicio con cualquier página de otro servidor). El carácter ~ se obtiene pulsando simultáneamente <Alt Gr> y <4>.

   `http://www.dcomg.upv.es/~jtomas/corta.html`

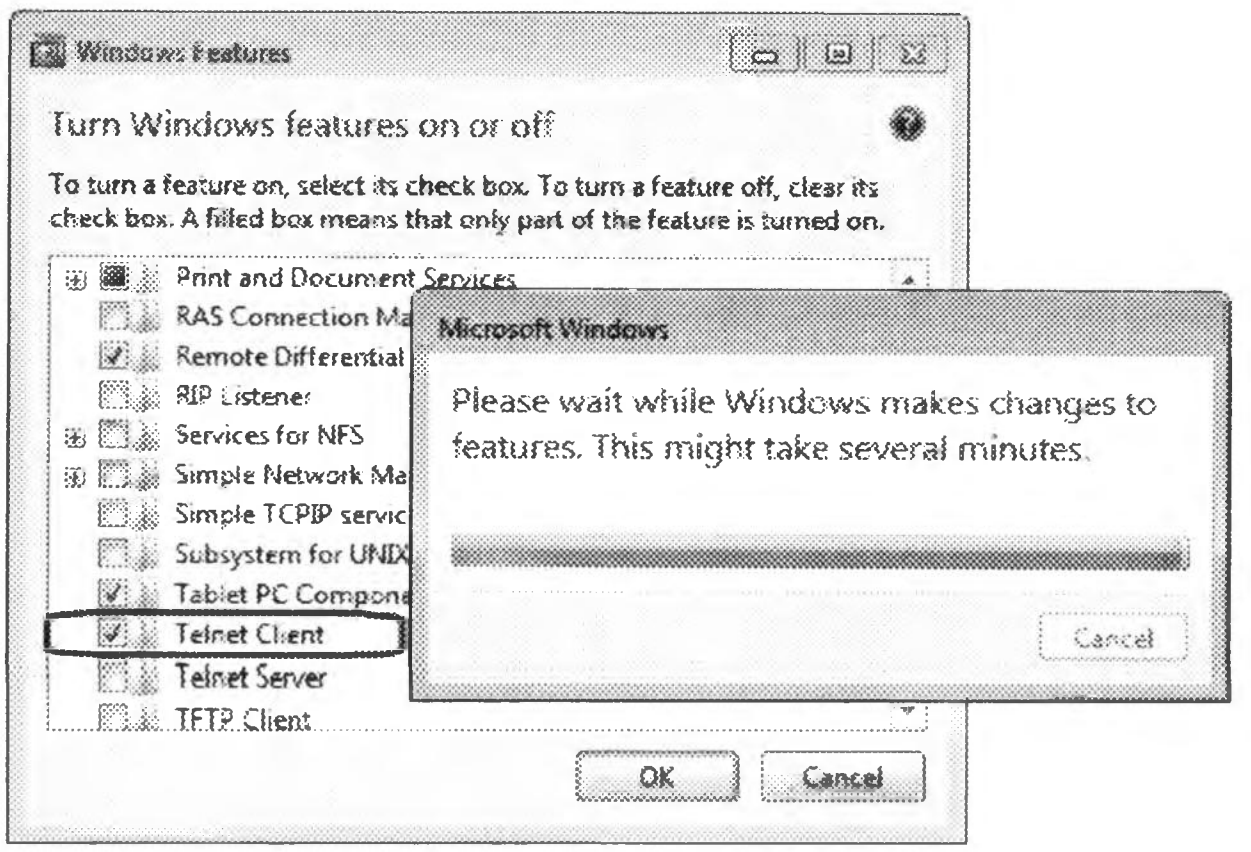
2. Visualiza el contenido HTML de la página (menú "Ver/Código fuente", "Herramientas/ Ver código fuente", o similar).

3. Desde un intérprete de comandos (símbolo del sistema/shell) escribe:

   ```
   telnet www.dcomg.upv.es 80
   ```

   Este comando permite establecer una conexión TCP con el puerto 80 del servidor. A partir de ahora todo lo que escribas será enviado al servidor (aunque en muchos casos no lo veas en pantalla) y todo lo que el servidor envíe será impreso en pantalla.

***NOTA:*** *Windows 7 tiene desactivado por defecto el comando Telnet. Para habilitarlo:*



*Instrucciones:*

*Clic en el menú Inicio>Panel de control>Todos los programas.*

*Haz clic en la curva característica de Windows, selecciona (marca) el cliente Telnet (si hay diferentes servicios que desees permitir puedes seleccionarlas).*

*Haz clic en Aceptar (puede que tengas que reiniciar después de agregar el componente a fin de que los cambios surtan efecto).*

4. Cuando se establezca la conexión teclea exactamente:

```
GET /~jtomas/corta.html↵
```

Si te equivocas no uses la tecla de borrar. En tal caso, repite el ejercicio desde el punto 3.

5. Observa que la respuesta obtenida coincide con el contenido HTML del apartado 2.

## 10.2.2. Versión 1.0 del protocolo HTTP

Con la popularización de la aplicación WWW, pronto se vio la necesidad de ampliar este sencillo protocolo para permitir nuevas funcionalidades. Se definió la versión 1.0 del protocolo que añadía nuevos métodos (PUT, POST,..) además de permitir el intercambio de cabeceras entre cliente y servidor.

**Poli[Media]**: *El protocolo HTTP v1.0.*

A continuación se muestra un ejemplo de interacción para la versión 1.0:

*Cliente*:
```
GET /dir/pag.html HTTP/1.0 ↵
User-Agent: Internet Explorer v3.2 ↵
Host: mi_ordenador.upv.es ↵
Accept: text/html, image/gif, image/jpeg ↵
↵                                          <línea en blanco>
```

*Servidor*:
```
HTTP/1.1 200 OK ↵
Server: Microsoft-IIS/5.0 ↵
Last-Modified: Mon, 25 Feb 2002 15:49:22 GMT ↵
Content-Type: text/html↵
↵                                          <línea en blanco>
<HTML> ↵
<HEAD> ↵
<TITLE>Página de ... </TITLE> ↵
...
</HTML>
```

En esta nueva versión el navegador se conecta al puerto 80 del servidor y normalmente le envía el comando "GET" seguido de la página que desea obtener. Ahora el navegador añadirá la palabra "HTTP/1.0" para indicar al servidor que

quiere utilizar esta nueva versión del protocolo. A continuación, el navegador introducirá un salto de línea seguido, si lo desea, de alguna cabecera. Una cabecera consta de un identificador de cabecera, seguido de dos puntos y el valor de la cabecera, más un salto de línea (↵). Estas cabeceras van a permitir que el navegador indique al servidor ciertos datos que pueden serle de utilidad. Por ejemplo, con el identificador de cabecera "User-Agent:" se puede indicar el tipo y versión de navegador con el que se realiza la solicitud. "Host:" permite indicar el ordenador donde se ejecuta este navegador. O la cabecera "Accept:", que permite indicar al servidor qué tipo de documentos es capaz de visualizar el navegador en formato MIME. Cuando el navegador ya no quiera insertar más cabeceras introducirá una línea en blanco. Es decir, tras el salto de línea de la última cabecera, manda un nuevo salto de línea.

Cuando el servidor detecte dos saltos de línea seguidos sabrá que le ha llegado su turno y tiene que contestar. En esta versión del protocolo el servidor no transmitirá la página solicitada directamente. Antes contesta con una línea indicando: primero la versión más alta que soporta (normalmente "HTTP/1.1"), a continuación un espacio y un código de tres dígitos que informa si se puede realizar la operación solicitada, finalizando con una frase explicativa sobre este código. Algunos códigos de respuesta posible son: `200 OK`, `401 no autorizado`, `404 fichero no encontrado`,... Tras esta primera línea el servidor podrá enviar alguna cabecera con información sobre el servidor o el documento que va a transmitir. Cuando ya no quiera insertar más cabeceras introducirá una línea en blanco seguido del documento.

Aunque el método GET es el más utilizado, en la versión 1.0 se añaden nuevos métodos. A continuación se incluye una descripción:

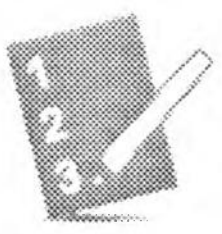
**GET**: Petición de lectura de un recurso.

**POST**: Envío de información asociada a un recurso del servidor.

**PUT**: Creación de un nuevo recurso en el servidor.

**DELETE**: Eliminación de un recurso.

**HEAD**: El servidor solo transmitirá las cabeceras, no la página



**Ejercicio paso a paso:** *Estudio del protocolo HTTP v1.0 utilizando el comando telnet.*

1. Desde un intérprete de comandos (símbolo del sistema/shell) escribe:

```
telnet www.dcomg.upv.es 80
```

2. Cuando se establezca la conexión teclea:

```
GET /~jtomas/corta.html HTTP/1.0↵
↵
```

3. Observa que la respuesta obtenida es similar al ejercicio anterior, pero ahora el servidor ha incluido cabeceras. ¿Qué información puedes sacar

de estas cabeceras? ¿Por qué has tenido que introducir dos saltos de línea?

4. Repite el ejercicio utilizando el comando HEAD.

Aunque ya se ha publicado la versión 1.2, la 1.1 es la más utilizada en la actualidad. Incorpora algunas mejoras como las conexiones persistentes activadas por defecto o la gestión de caché del cliente. También permite al cliente enviar múltiples peticiones a la vez.

### 10.2.3. Utilizando HTTP desde Android

Tras el gran éxito de la Web, el protocolo HTTP está siendo utilizado con finalidades diferentes a las que tenía en un principio, la descarga de páginas Web. Por ejemplo, hoy en día es frecuente su uso para el intercambio de ficheros o la comunicación entre aplicaciones. A continuación describiremos las herramientas disponibles en Android para utilizar el protocolo HTTP.

Para este propósito tenemos dos alternativas principales desde Android: el uso de las librerías `java.net.*` o `org.apache.commons.httpclient.*`. En el siguiente ejemplo utilizaremos las primeras.

En el ejemplo mostrado en este apartado vamos a extraer información de una de las páginas web más utilizadas en la actualidad, el servicio de búsqueda de Google. En concreto, nos interesa conocer el número de apariciones de una determinada secuencia de palabras en la web. En ciertas ocasiones esta información puede resultar muy interesante. Por ejemplo, tenemos dudas sobre el uso de una preposición en inglés, ¿Se escribe *travel in bus* o *travel by bus*? Si buscamos ambas secuencias de palabras en Google, hay que ponerlas entre comillas para que busque la secuencia de forma literal, obtenemos 26.000 apariciones para la primera y 161.000 para la segunda. Nuestra duda ha sido resuelta. Esta aplicación también puede ser utilizada para averiguar quién es más popular en Internet Antonio Banderas o Rafael Nadal.

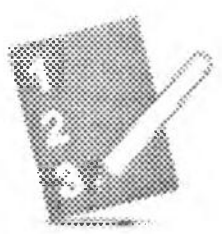
Básicamente la aplicación que mostramos a continuación funciona de la siguiente forma:

1) El usuario introduce una secuencia de palabras, por ejemplo "Antonio Banderas".
2) Accedemos al servidor mediante la siguiente URL:

```
http://www.google.es/search?hl=es&q="Antonio+Banderas"
```

3) En este caso las peticiones se atienten mediante el método GET. En este método si queremos enviar información al servidor hemos de incluirla tras un carácter "?" seguido de un nombre de parámetro seguido del carácter "=" seguido del valor. Los diferentes parámetros se separan mediante el carácter "&". Los espacios en blanco han de ser sustituidos por el carácter "+". En este ejemplo el parámetro hl corresponde al idioma de búsqueda y q a las palabras a buscar.
4) Obtenemos la respuesta del servidor en una variable de tipo String.
5) Buscamos la primera aparición de "Aproximadamente" en la respuesta.
6) Tras esta palabra se encuentra la información que buscamos.

Obviamente, el correcto funcionamiento de esta aplicación está sujeto a que no se produzcan cambios en la forma en que Google visualiza los resultados. En caso de que no funcione correctamente va a ser necesaria una adaptación. Hoy en día existe otra alternativa más interesante para consultar un motor de búsqueda desde una aplicación. En el siguiente apartado mostraremos cómo utilizar un servicio web de Yahoo con este propósito.



**Ejercicio paso a paso:** *Utilizando HTTP desde Android.*

1. Crea una nueva aplicación con los siguientes datos:

```
Project name: HTTP
Build Target: Android 1.5
Application name: HTTP
Package name: org.example.http
Create Activity: HTTP
Min SDK Version: 3
```

2. Asegúrate que la aplicación solicita el permiso de acceso a Internet, añadiendo la siguiente línea en `AndroidManifest.xml`.

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

3. Remplaza el código del *Layout* `main.xml` por:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent">
```

```
    <EditText android:id="@+id/EditText01"
        android:layout_height="wrap_content"
        android:layout_width="fill_parent"
        android:text="palabra a buscar"/>
    <Button android:id="@+id/Button01"
        android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        android:text="buscar en Google"/>
    <TextView android:id="@+id/TextView01"
        android:layout_height="fill_parent"
        android:layout_width="fill_parent"
        android:textSize="8pt"/>
</LinearLayout>
```

La apariencia de este *Layout* se muestra a continuación:


4. Remplaza el código de `HTTP.java` por:

```
package org.example.http;

import java.net.HttpURLConnection;
import java.net.URL;
import java.net.URLEncoder;
import ...;

public class HTTP extends Activity {

    private EditText entrada;
    private Button boton;
    private TextView salida;

    @Override
     public void onCreate(Bundle savedInstanceState) {
         super.onCreate(savedInstanceState);
         setContentView(R.layout.main);
         entrada = (EditText)findViewById(R.id.EditText01);
```

```
boton = (Button)findViewById(R.id.Button01);
salida = (TextView)findViewById(R.id.TextView01);
boton.setOnClickListener(new OnClickListener(){
public void onClick(View view) {
   try {
      String palabras = entrada.getText().toString();
      String resultado = resultadosGoogle(palabras);
      salida.append(palabras+"--"+resultado+"\n");
   } catch (Exception e) {
      salida.append("Error al conectar\n");
         e.printStackTrace();
      }
   }
  } );
 }

String resultadosGoogle(String palabras) throws Exception {
   String pagina = "", devuelve = "";
   URL   url = new URL("http://www.google.es/search?hl=es&q=\""+
                URLEncoder.encode(palabras, "UTF-8")+"\"");
   HttpURLConnection conexion =
                (HttpURLConnection) url.openConnection();
   conexion.setRequestProperty("User-Agent",
        "Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1)");
   if (conexion.getResponseCode()==HttpURLConnection.HTTP_OK ){
   BufferedReader reader = new BufferedReader(
        new InputStreamReader(conexion.getInputStream()));
   String linea = reader.readLine();
   while (linea != null) {
      pagina+=linea;
      linea = reader.readLine();
   }
   reader.close();
   int ini=pagina.indexOf("Aproximadamente");
   if (ini!=-1){
      int fin=pagina.indexOf(" ", ini+16);
      devuelve = pagina.substring(ini+16, fin);
   }else{
      devuelve ="no encontrado";
      }
   } else {
      salida.append("ERROR :"+conexion.getResponseMessage()+"\n");
   }
   conexion.disconnect();
   return devuelve;
  }
}
```

Pasemos a describir el método `resultadosGoogle()`. El resto del código no presenta mayor interés. Este método toma como entrada una secuencia de palabras y devuelve el número de veces que Google las ha encontrado en alguna página web. Lo primero que llama la atención es el modificador `throws Exception`. Estamos obligados a incluirlo si utilizamos la clase `HttpURLConnection`. Este modificador obliga a utilizar el método dentro de una sección `try ... catch ...`. La razón de esto es que toda conexión HTTP es susceptible de no poder realizarse, por lo que tenemos la obligación de tomar las acciones pertinentes en caso de problemas.

Tras la declaración de variables, creamos la URL que utilizaremos para la conexión. El método `URLEncoder.encode()` se encargará de codificar las palabras en el formato esperado por el servidor. Entre otras cosas remplaza espacios en blanco, caracteres no ASCII, etc. A continuación, creamos la conexión por medio de la clase `HttpURLConnection`. Mediante el método `setRequestProperty()` podemos añadir cabeceras HTTP. En el siguiente ejercicio se demuestra la necesidad de insertar la cabecera `User-Agent`.

En la siguiente línea se utiliza el método `getResponseCode()` para establecer la conexión. Si se establece sin problemas (`HTTP_OK`) leemos la respuesta línea a línea, concatenándola en el *String* `pagina`. De esta forma podemos buscar en `pagina` la primera aparición de la palabra "Aproximadamente". Si la encontramos buscamos el primer espacio detrás del número que ha de aparecer tras "Aproximadamente" y devolvemos los caracteres entre ambas posiciones. En caso de no encontrar "Aproximadamente" puede que la secuencia buscada no se haya encontrado, aunque también es posible que Google haya cambiado la forma de devolver los resultados. En el caso de que la conexión no haya sido satisfactoria, devolvemos el mensaje de respuesta que nos dio el servidor `getResponseMessage()`.

5. En el programa anterior comenta la línea:

```
conexion.setRequestProperty("User-Agent", ...);
```

6. Ejecuta el programa. Ahora el resultado ha de ser:

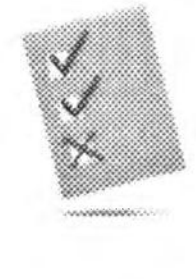
```
ERROR: Forbidden
```

En este caso, al tratar de establecer la conexión, el servidor en lugar de devolvernos el código de respuesta `200: OK`, nos ha devuelto el código `403: Forbidden` ¿Por qué? La cabecera `User-Agent` informa al servidor de qué tipo de cliente ha establecido la conexión. Según se demuestra en este ejercicio, el servicio de búsquedas de Google prohibe la respuesta a aquellos clientes que no se identifican.

7. Analiza el valor asignado a la cabecera "Mozilla/4.0 ...". Puedes comprobar que la información que estamos dando al servidor es totalmente errónea.

8. Modifica este valor por "Ejemplo de El gran libro de Android" y comprueba el resultado es `403: Forbidden` ¿Por qué no quiere responder? La

respuesta es que desde finales de 2011, el servidor de Google exige que el tipo de navegador que se conecte sea conocido.



**Preguntas de repaso y reflexión:** *El protocolo http.*

## 10.3. Servicios web

La W3C define "Servicio web" como un sistema de software diseñado para permitir la interoperabilidad máquina a máquina en una red. Se trata de API que son publicadas, localizadas e invocadas a través de la web. Es decir, una vez desarrolladas, son instaladas en un servidor, y otras aplicaciones (u otros servicios Web) pueden descubrirlas desde otros ordenadores de Internet e invocar uno de sus servicios.

Como norma general, el transporte de los datos se realiza a través del protocolo HTTP y la representación de los datos mediante XML. Sin embargo, no hay reglas fijas en los servicios web y en la práctica no tiene por qué ser así.

Una de las grandes ventajas de este planteamiento es que es tecnológicamente neutral. Es decir, podemos utilizar un servicio web sin importarnos el sistema operativo o el lenguaje en el que fue programado. Además, al apoyarse sobre el protocolo HTTP, puede utilizar los sistemas de seguridad (https) y presenta pocos problemas con cortafuegos al utilizar puertos que suelen estar abiertos (80 ó 8080).

Como inconveniente resaltar que, dado que el intercambio de datos se realiza en formato de texto (XML), tiene menor rendimiento que otras alternativas como RMI (*Remote Method Invocation*), CORBA (*Common Object Request Broker Architecture*) o DCOM (*Distributed Component Object Model*). Además el hecho de apoyarse en HTTP, hace que resulte complicado a un cortafuego filtrar este tipo de tráfico. ¿No acabamos de decir que esto era una ventaja? Es posible que, lo que para un desarrollador sea una ventaja, para un administrador de red sea un inconveniente.

### 10.3.1. Alternativas en los servicios web

Como acabamos de ver, el término servicio web resulta difícil de definir de forma precisa. En torno a este concepto se han desarrollado varias propuestas bastante diferentes entre sí. En este apartado estudiaremos las dos alternativas que están teniendo más relevancia en la actualidad: SOAP y REST. No obstante, dada la complejidad que surge de estas propuestas, resulta interesante centrar antes algunos conceptos antes de empezar a describir estas alternativas. Comenzaremos indicando que existen tres enfoques diferentes a la hora definir un servicio Web, es lo que se conoce como arquitectura del servicio web.

- **Llamadas a Procedimiento Remotos (RPC):** se enfoca el servicio web como una colección de operaciones o procedimientos que pueden ser

invocados desde una máquina diferente a donde se ejecutan. Resulta una visión sencilla de entender para un programador, por lo que fue una de las primeras que se implementó en lo que se conoce como servicios web de primera generación.

- **Arquitectura Orientada a Servicios (SOA):** en el planteamiento anterior, RPC, la unidad básica de interacción es la operación; en este nuevo planteamiento, la unidad de interacción pasa a ser el mensaje. Por lo tanto, en muchos casos se conoce como servicios orientados a mensaje. Cada uno de los mensajes que vamos a utilizar ha de ser definido siguiendo una estricta sintaxis expresada en XML. En la actualidad se trata de la arquitectura más extendida, soportada por la mayoría del software de servicios web.
- **Transferencia de Estado Representacional (REST):** en los últimos años se está popularizando este nuevo planteamiento, que se caracteriza principalmente por su simplicidad. En REST se utiliza directamente el protocolo HTTP, por medio de sus operaciones GET, POST, PUT y DELETE. En consecuencia, esta arquitectura se centra en la solicitud de recursos, en lugar de las operaciones o mensajes de las alternativas anteriores.

### *10.3.1.1.Servicios web basados en SOAP*

SOAP es el protocolo más utilizado en la actualidad para implementar servicios Web. Fue creado por Microsoft, IBM y otros, aunque en la actualidad está bajo el auspicio de la W3C.

Utiliza como transporte HTTP, aunque también es posible utilizar otros métodos de transporte, como el correo electrónico.

Los mensajes del protocolo se definen utilizando un estricto formato XML, que ha de ser consensuado por ambas partes. A continuación, se muestra un posible ejemplo de mensaje SOAP.

```
<soapenv:Envelope
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing">
    <soapenv:Header>
        <wsa:MessageID>
            uuid:920C5190-0B8F-11D9-8CED-F22EDEEBF7E5
        </wsa:MessageID>
        <wsa:To>
            http://localhost:8081/axis/services/BankPort
        </wsa:To>
    </soapenv:Header>
    <soapenv:Body>
        <axis2:echo xmlns:axis2="http://ws.apache.org/axis2">
            Hello World
        </axis2:echo>
    </soapenv:Body>
</soapenv:Envelope>
```

Un mensaje SOAP contiene una etiqueta `<Envelope>`, que encapsula las etiquetas `<Header>` y `<Body>`. La etiqueta `<Header>` es opcional y encapsula aspectos relativos a calidad de servicio, como seguridad, esquemas de direccionamiento,... La cabecera `<Body>` es obligatoria y contiene la información que las aplicaciones quieren intercambiar.

SOAP proporciona una descripción completa de las operaciones que puede realizar un nodo mediante una descripción WSDL (*Web Services Description Language*), por supuesto codificada en XML. En uno de los siguientes apartados crearemos un servicio web y podremos estudiar el fichero WSDL correspondiente.

Aunque SOAP está ampliamente extendido como estándar para el desarrollo de servicios web, no resulta muy adecuado para ser utilizado en Android. Esto es debido a la complejidad introducida, supone una sobrecarga que implica un menor rendimiento frente a otras alternativas como REST. Además, Android no incorpora las librerías necesarias para trabajar con SOAP.

No obstante, es posible que ya dispongas de un servidor basado en SOAP y necesites implementar un cliente en Android. En tal caso puedes utilizar las librerías kSOAP2 (http://ksoap2.sourceforge.net/).

### *10.3.1.2.Servicios web basados en REST*

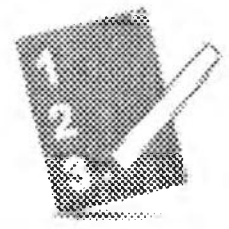
En primer lugar conviene destacar que el término REST se refiere a una arquitectura en lugar de a un protocolo en concreto como es el caso de SOAP. A diferencia de SOAP no vamos a añadir una capa adicional a la pila de protocolos, sino que utilizaremos directamente el protocolo HTTP. Siendo estrictos, la arquitectura REST no impone el uso de HTTP; no obstante, en la práctica se entiende que un servicio web basado en REST, es aquel que se implementa directamente sobre la web.

Este planteamiento supone seguir los principios de la aplicación WWW, pero en lugar de solicitar páginas web solicitaremos servicios web. Los principios básicos de la aplicación WWW y, por tanto, los de REST son:

- Transporte de datos mediante HTTP, utilizando las operaciones de este protocolo, que son GET, POST, PUT y DELETE.
- Los diferentes servicios son invocados mediante el espacio de URI unificado. Como ya se ha tratado en este libro una URI identifica un recurso en Internet. Este sistema ha demostrado ser flexible, sencillo y potente a un mismo tiempo. Se cree que fue uno de los principales factores que motivó el éxito de WWW.
- La codificación de datos es identificada mediante tipos MIME (text/html, image/gif,...). Aunque el tipo de codificación preferido es XML (text/xml).

Las ventajas de REST derivan de su simplicidad. Entre estas podemos destacar mejores tiempos de respuesta y disminución de sobrecarga tanto en cliente como en servidor. Mayor estabilidad frente a futuros cambios. Y también, una gran sencillez en el desarrollo de clientes, estos solo han de ser capaces de realizar interacciones HTTP y codificar información en XML.

Como inconveniente indicar que, al igual que ocurre con el protocolo HTTP, no se mantiene el estado. Es decir, cada solicitud es tratada por el servidor de forma independiente sin recordar solicitudes anteriores.



**Ejercicio paso a paso:** *Comparativa entre una interacción SOAP y REST.*

La empresa WebserviceX.NET ofrece un servicio Web, GetIPService, que nos permite conocer, a partir de una dirección IP, el país al que pertenece. Este servicio puede ser utilizado tanto con REST como con SOAP, lo cual nos va a permitir comparar ambos mecanismos. Más todavía, dentro de REST tenemos dos opciones para mandar datos al servidor, el método GET y el método POST. El servicio que vamos a probar nos permite las dos variantes, lo que nos permitirá comparar ambos mecanismos.

1. Abre un navegador web y accede a la URL:

   http://www.webservicex.net/geoipservice.asmx/GetGeoIP?IPAddress=158.42.38.1

2. Verifica que el resultado es similar al siguiente:

```
<?xml version="1.0" encoding="utf-8"?>
<GeoIP xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="http://www.webservicex.net/">
    <ReturnCode>1</ReturnCode>
    <IP>158.42.38.1</IP>
    <ReturnCodeDetails>Success</ReturnCodeDetails>
    <CountryName>European Union</CountryName>
    <CountryCode>EU</CountryCode>
</GeoIP>
```

3. Prueba otras IP al azar y verifica a que países pertenecen.

4. Vamos a emular el protocolo HTTP de forma similar a como lo hemos hecho en secciones anteriores. Desde un intérprete de comandos (símbolo del sistema/shell) escribe:

```
telnet www.webservicex.net 80
```

5. Cuando se establezca la conexión teclea exactamente el siguiente código seguido de un salto de línea adicional (↵):

```
GET /geoipservice.asmx/GetGeoIP?IPAddress=158.42.38.1 HTTP/1.1
Host: www.webservicex.net
```

***NOTA:*** *También puedes cortar el texto y pegarlo. Para pegar sobre la ventana de símbolo de sistema de Windows has de pulsar con el botón derecho del ratón y seleccionar Pegar.*

El resultado ha de parecerse al anterior aunque al principio el servidor enviará sus cabeceras:

```
HTTP/1.1 200 OK
Cache-Control: private, max-age=0
Content-Length: 374
Content-Type: text/xml; charset=utf-8
Server: Microsoft-IIS/7.0
X-AspNet-Version: 4.0.30319
X-Powered-By: ASP.NET
Date: Mon, 30 Jan 2012 19:28:55 GMT
```

Como acabamos de ver, el protocolo HTTP permite enviar información al servidor utilizando el método GET e introduciendo un carácter "?" al final de la URL seguido de los parámetros.

6. El protocolo HTTP también permite mandar información con el método POST. El servicio Web que estamos utilizando nos permite las dos alternativas. Veamos en qué consiste el método POST. Escribe en el intérprete de comandos:

```
telnet www.webservicex.net 80
```

7. Cuando se establezca la conexión pega los siguientes caracteres:

```
POST /geoipservice.asmx/GetGeoIP HTTP/1.1
Host: www.webservicex.net
Content-Type: application/x-www-form-urlencoded
Content-Length: 21
IPAddress=158.42.38.1
```

Como puedes observar, la información enviada es la misma, aunque ahora en lugar de adjuntarla a la URL se manda tras las cabeceras separada por una línea en blanco.

***NOTA:*** *La cabecera* `Content-Length:` *es obligatoria. Indica el número de caracteres enviados. En caso de que cambiara la longitud de la dirección IP tendrías que ajustarlo.*

8. El servidor está esperando nuevos comandos. Pulsa <Ctrl+C> para cerrarla conexión.

9. Ahora vamos a usar el mismo servicio aunque mediante el protocolo SOAP 1.1 (***NOTA:*** *también es posible utilizar SOAP 1.2*). Escribe en el intérprete de comandos:

```
telnet www.webservicex.net 80
```

10. Cuando se establezca la conexión pega los siguientes caracteres:

```
POST /geoipservice.asmx HTTP/1.1
Host: www.webservicex.net
Content-Type: text/xml; charset=utf-8
Content-Length: 371
SOAPAction: "http://www.webservicex.net/GetGeoIP"
```

```
<?xml version="1.0" encoding="utf-8"?>

<soap:Envelope
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <GetGeoIP xmlns="http://www.webservicex.net/">
      <IPAddress>158.42.38.1</IPAddress>
    </GetGeoIP>
  </soap:Body>
</soap:Envelope>
```

11. Compara la información mandada en SOAP con el caso anterior.

12. Pulsa <Ctrl-C> para cerrar la conexión. El resultado obtenido ha de ser similar al siguiente:

```
HTTP/1.1 200 OK
Cache-Control: private, max-age=0
Content-Length: 514
Content-Type: text/xml; charset=utf-8
Server: Microsoft-IIS/7.0
X-AspNet-Version: 4.0.30319
X-Powered-By: ASP.NET
Date: Mon, 30 Jan 2012 20:07:55 GMT

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <soap:Body>
    <GetGeoIPResponse xmlns="http://www.webservicex.net/">
      <GetGeoIPResult>
        <ReturnCode>1</ReturnCode>
        <IP>158.42.38.1</IP>
        <ReturnCodeDetails>Success</ReturnCodeDetails>
        <CountryName>European Union</CountryName>
        <CountryCode>EU</CountryCode>
      </GetGeoIPResult>
    </GetGeoIPResponse>
  </soap:Body>
</soap:Envelope>
```

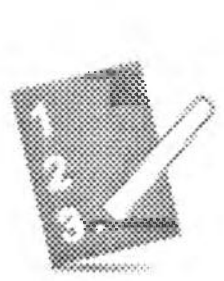
**Recursos adicionales:** *Ejemplos de algunos servicios Web gratuitos.*

En la siguiente tabla te mostramos una lista con algunos servicios Web interesantes. Además de estas te recomendamos que visites la Web de la empresa WebserviceX.NET que ofrece decenas de servicios Web gratuitos.

| Nombre | Descripción | Empresa | Tipo (codific.) |
|---|---|---|---|
| Web Search API | Búsqueda el la Web. Obsoleto. No hay que registrar clave. | Google | REST (JSON) |
| XML Custom Search API | Búsqueda el la Web. Hay que registrar clave. Máximo 100 cosultas/dia. | Google | REST (XML) |
| JSON/Atom Custom Search API | Igual que el anterior con respuesta JSON o Atom. Ejemplo | Google | REST (JSON/ Atom) |
| Books API | Búsqueda y altas de libros. Obsoleto. No hay que registrar clave. Ejemplo | Google | REST (XML) |
| Books API (nueva) | Búsqueda y altas de libros. Ejemplo | Google | REST (JSON) |
| GetIPService | A partir de una IP nos indica el país. Ejemplo | WebservicX.NET | SOAP /REST |
| Currency Convertor | Convertidor de divisas según cambio actual. Ejemplo: ratio euro-dolar | WebservicX.NET | SOAP /REST |

## 10.3.2. Acceso a servicios web de terceros

En este apartado nos centraremos en cómo utilizar un servicio REST publicado por un tercero. Como se ha comentado, el acceso a un servicio SOAP resulta algo más complicado y Android no dispone de librerías para facilitarnos el trabajo.



**Ejercicio paso a paso:** *Acceso a servicios Web de búsqueda de libros.*

En concreto vamos a utilizar el servicio Books API que permite buscar y gestionar libros de la base de datos de Google. Es un servicio obsoleto aunque actualmente operativo. No usaremos el nuevo servicio que ofrece Google (nueva Books API),

dado que este nos da el resultado en JSON en lugar de en XML y en este libro no hemos estudiado el trabajo con JSON.

1. Para probar el servicio Web abre un navegador web y accede a la siguiente URL:

   http://books.google.com/books/feeds/volumes?q=antonio+banderas

   El resultado ha de ser similar a:

```
<?xml version='1.0' encoding='UTF-8'?>
<feed xmlns:openSearch="http://a9.com/-/spec/opensearchrss/1.0/"
    xmlns:gbs="http://schemas.google.com/books/2008"
    xmlns:dc="http://purl.org/dc/terms"
    xmlns:batch=http://schemas.google.com/gdata/batch
    xmlns:gd="http://schemas.google.com/g/2005"
    xmlns="http://www.w3.org/2005/Atom" >
      <id >http://www.google.com/books/feeds/volumes</id>
      <updated >2012-01-30T23:56:34.000Z</updated>
      <category scheme=http://schemas.google.com/g/2005#kind
          term="http://schemas.google.com/books/2008#volume" />
      <title type="text" > Search results for antonio
                                                banderas</title>
      <link href="http://www.google.com" rel="alternate"
                                                type="text/html"/>
  <link  href="http://www.google.com/books/feeds/volumes"
          rel="http://schemas.google.com/g/2005#feed"
          type="application/atom+xml" />
  <link href="http://www.google.com/books/feeds/volumes?q=
     antonio+banderas"    rel="self" type="application/atom+xml" />
  <link href="http://www.google.com/books/feeds/volumes?q=
          antonio+banderas&amp;start-index=11&amp;max-results=10"
          rel="next" type="application/atom+xml" />
  <author><name>Google Books Search</name>
      <uri>http://www.google.com</uri></author>
  <generator version="beta" >Google Book Search data API</generator>
  <openSearch:totalResults >596</openSearch:totalResults>
  <openSearch:startIndex >1</openSearch:startIndex>
  <openSearch:itemsPerPage >10</openSearch:itemsPerPage>
  <entry >
      <id >http://www.google.com/books/feeds/volumes/_Y81OAAACAAJ</id>
      <updated >2012-01-30T23:56:34.000Z</updated>
...
```

2. En el resultado obtenido localiza la etiqueta `totalResults`. Representan los libros encontrados que contienen la búsqueda “antonio banderas”.

3. Pasamos a crear una nueva aplicación, funcionalmente igual a la del ejercicio “Utilizando HTTP desde Android”, pero internamente ahora utilizaremos un servicio web de búsqueda de libros.

En el explorador de aplicaciones, pulsa con el botón derecho sobre la aplicación HTTP. En el menú contextual selecciona *Copy*.

4. Pulsa de nuevo con el botón derecho, pero esta vez selecciona *Paste*. Te solicitará un nombre para la nueva aplicación, llámale `ServicioWebLibros`.

5. Usando el explorador de aplicaciones abre en la nueva aplicación la carpeta `src/org.example.http` dentro estará la clase `HTTP.java`. Con el botón derecho selecciona la opción *Refactor/Rename* e introduce el nombre `ServicioWebLibros`.

6. En esta clase reemplaza el método `resultadosGoogle()` por el código que se muestra a continuación.

```
String resultadosSW(String palabras) throws Exception {
   URL url = new URL("http://books.google.com/books/feeds/volumes?q=\""
                        + URLEncoder.encode(palabras, "UTF-8") + "\"");
   SAXParserFactory fabrica = SAXParserFactory.newInstance();
   SAXParser parser = fabrica.newSAXParser();
   XMLReader lector = parser.getXMLReader();
   ManejadorXML manejadorXML = new ManejadorXML();
   lector.setContentHandler(manejadorXML);
   lector.parse(new InputSource(url.openStream()));
   return manejadorXML.getTotalResults();
}
```

El método comienza creando la URL de acceso. El resto del código utiliza las librerías `org.xml.sax.*` para realizar un proceso de *parser* sobre el código XML de la URL y así extraer la información que nos interesa. Este proceso ha sido explicado en el capítulo anterior. El trabajo que sí tendremos que realizar en función del formato XML específico, será la creación de la clase `XMLHadler`. Una vez finalizado el *parser*, podemos llamar al método `getTotalResults()` de nuestro manejador para que nos devuelva la información que nos interesa.

7. También has de sustituir la llamada a este método que se hace desde `onCreate()`.

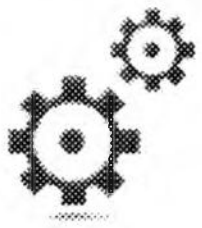
8. En `res/layout/main.xml` puedes cambiar el texto asociado al botón `"buscar en Google"` por `"buscar Libros"`.

9. A continuación mostramos la definición de la clase `ManejadorXML`. Copia este código dentro de la clase `ServicioWebLibros`.

```
public class ManejadorXML extends DefaultHandler {

   private String totalResults;
   private StringBuilder cadena = new StringBuilder();

public String getTotalResults() {
      return totalResults;
   }
```

```
    @Override
    public void startElement(String uri, String nombreLocal, String
          nombreCualif, Attributes atributos) throws SAXException {
       cadena.setLength(0);
    }

    @Override
    public void characters(char ch[], int comienzo, int lon){
       cadena.append(ch, comienzo, lon);
    }

    @Override
    public void endElement(String uri, String nombreLocal,
                  String nombreCualif) throws SAXException {
       if (nombreLocal.equals("totalResults")) {
          totalResults = cadena.toString();
       }
    }
}
```

Para procesar el fichero XML extendemos la clase `DefaultHandler` y reescribimos muchos de sus métodos: En `startElement()` inicializamos la variable `cadena` cada vez que empieza una etiqueta. En el método `characters()` añadimos el contenido que aparece dentro de la etiqueta. Finalmente en `endElement()` recogemos el valor acumulado en `cadena` cada vez que termina una etiqueta. Como hemos comentado, de todo el código XML que vamos a procesar solo nos interesa el contenido de la etiqueta `totalResults`.



**Práctica:** *Convertidor de divisas mediante un servicio Web.*

La empresa WebserviceX.NET ofrece un servicio Web, Currency Convertor[1], que permite obtener el ratio de conversión entre de divisas según cambio actual. Un ejemplo de uso para obtener el ratio euro-dólar se muestra a continuación:

http://www.webservicex.net/CurrencyConvertor.asmx/ConversionRate?FromCurrency=EUR&ToCurrency=USD

1. Retoma el diseño de *Layouts* de la euro-calculadora, realizado en el capítulo 2, para un nuevo proyecto con nombre "Convertidor de divisas". En una primera fase solo se realizará la conversión de euros a dólares, aplicando el ratio obtenido a través del servicio Web indicado.

[1] http://www.webservicex.net/WS/WSDetails.aspx?WSID=10&CATID=2

2. En una segunda fase puedes permitir que el usuario seleccione la divisa de entrada y la de salida. Una lista de las divisas disponibles la puedes encontrar en la información del servicio Currency Convertor.

## 10.3.3. Diseño e implantación de nuestro servicio web

A la hora de desarrollar una aplicación distribuida, una de las alternativas más utilizadas en la actualidad son los servicios web. A lo largo de este apartado aprenderás cómo crear tus propios servicios web e instalarlos en un servidor Web. Como no podría ser de otra manera, el ejemplo seleccionado es el mismo que ya hemos implementado en varias ocasiones, un almacén de las mejores puntuaciones de Asteroides.

No resultaría demasiado complicado rescribir el ejemplo del servidor de ECHO descrito en el apartado anterior para crear nuestro propio servidor web. No obstante, si estamos trabajando en un entorno empresarial esta alternativa no sería la más adecuada. En este entorno es mucho más recomendable utilizar un servidor web de uso comercial, como Apache Tomcat, e instalar un soporte para servicios web, como Axis2. Esta solución resulta mucho más segura y escalable.

A continuación veremos cómo instalar estos servidores y cómo se integran perfectamente con Eclipse. Por lo tanto, la creación de los servicios web será algo extremadamente sencillo; de hecho, todo el proceso será realizado de forma automática por Eclipse, creándonos los servicios web tanto en REST como en SOAP. Por supuesto, se trata solo de una de las muchas alternativas para crear un servicio Web. Es posible que tengas experiencia en algún lenguaje para realizar aplicaciones Web, como PHP o ASP; en tal caso, te resultará muy sencillo crear un servicio Web para acceder a tus bases de datos.

Para finalizar el apartado, describiremos la parte del cliente que hay que incluir dentro de Asteroides para que pueda obtener el listado de puntuaciones y publicar nuevas puntuaciones. Solo se incluye el cliente REST; implementar un cliente SOAP en Android es una tarea no trivial dado que no incorpora las librerías necesarias. No obstante, a juicio del autor, implementar el cliente SOAP en lugar de REST, no obtendría ninguna ventaja. Ocasionaría mayor flujo de datos y un código más pesado.

### *10.3.3.1. Instalación del servidor de servicios web*

Para realizar la aplicación que describimos a continuación necesitamos tener instalado el siguiente *software*:

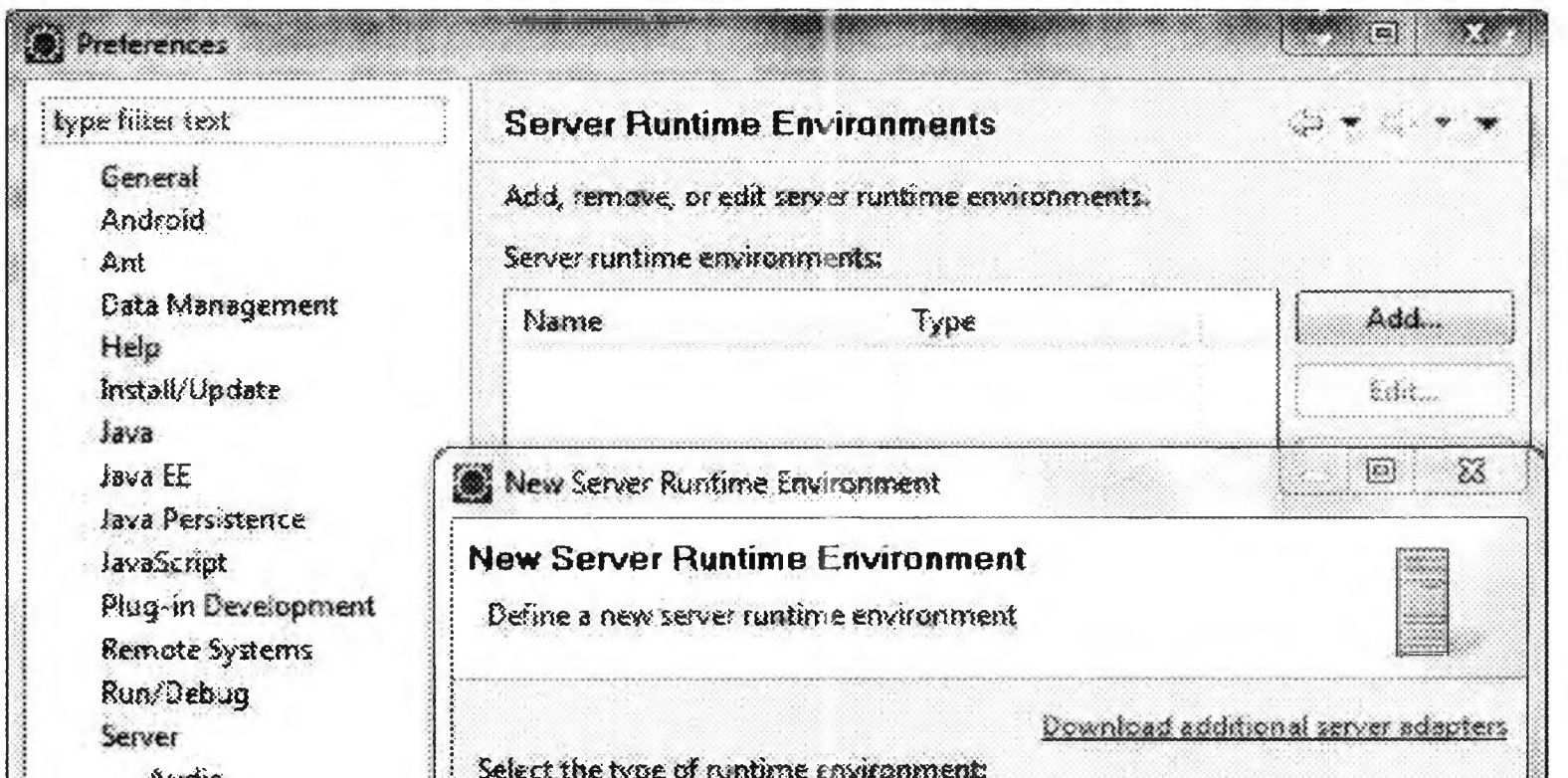
- **Apache Tomcat:** El más que conocido Servidor web con soporte para Java. Puedes descargarlo de: http://tomcat.apache.org. Si trabajas en Windows te recomendamos la versión ejecutable con asistente de instalación.
- **Axis2:** Soporte para Servicios web que funciona sobre Tomcat. Lo encontrarás en http://ws.apache.org/axis2
- **Eclipse con WTP (*Web Tools Plataform*):** si instalaste la versión "Eclipse for Java developers" no tendrás instaladas las herramientas web. En este caso te

recomendamos instales la versión "Eclipse J2EE" que encontrarás en http://www.eclipse.org/downloads.

***NOTA:*** *El software utilizado en este ejercicio ha sido: Tomcat 7.0, Axis2-1.5.2 y Eclipse JEE Helios R1. Te recomendamos que utilices estas versiones, dado que hemos comprobado que algunas versiones más recientes provocan algunos problemas al realizar el ejercicio.*

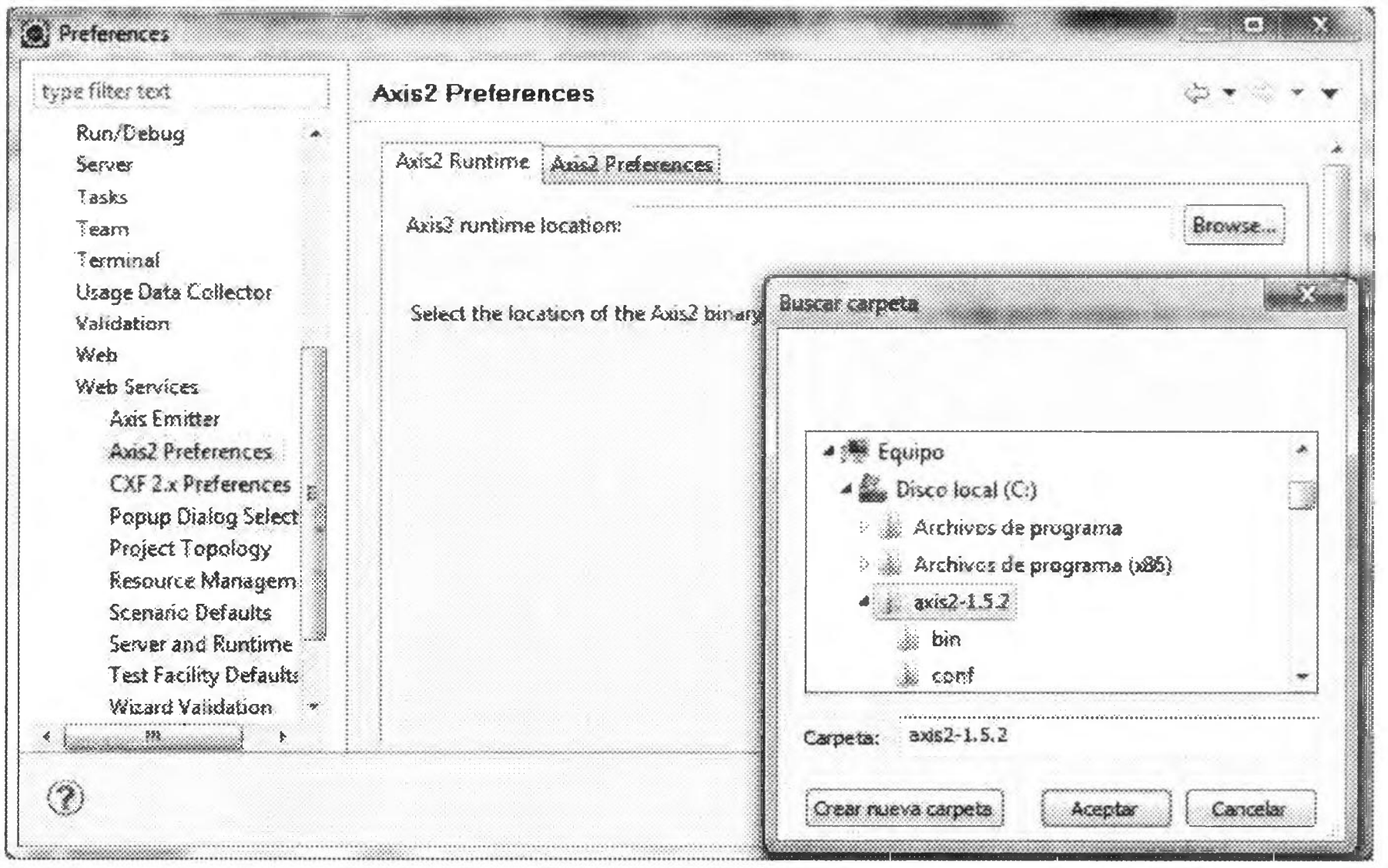
Una vez hayas descargado todo el *software* comienza instalando Apache Tomcat. Sigue las indicaciones el asistente o, en su defecto, las instrucciones de instalación. Para instalar Axis2 y Eclipse no tienes más que descomprimir los ficheros comprimidos en la carpeta que quieras.

A continuación, configura Eclipse para que pueda interactuar con Apache Tomcat y Axis2. Abre el menú *Windows* > *Preference* de Eclipse. Busca la opción *Server* > *Runtime Envirioment*. Pulsa el botón *Add...* y selecciona "el tipo de servidor" y "servidor a instalar". En nuestro caso Apache Tomcat v7.0.




Pulsa *Next* y luego *Browse...* para informar a Eclipse en qué directorio has instalado Apache Tomcat. Para terminar pulsa en *Finish* y te aparecerá el nuevo servidor en la lista *Entornos de tiempo real con servidor*.

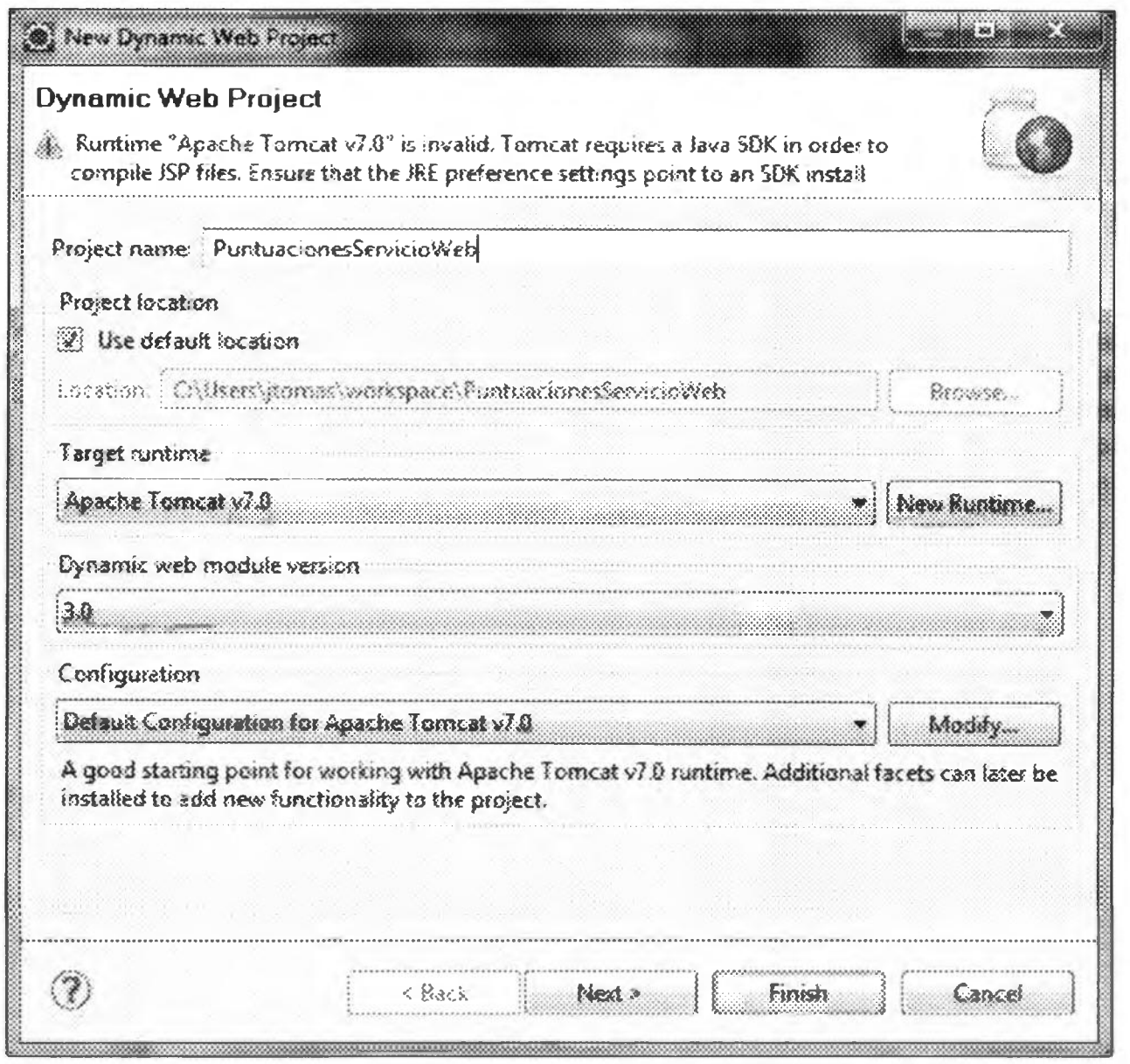
Siguiendo en la ventana *Preferences*, busca la opción *Web Services/Axis2 Preferences*. Pulsa el botón *Next* e indica directorio has instalado Axis2.



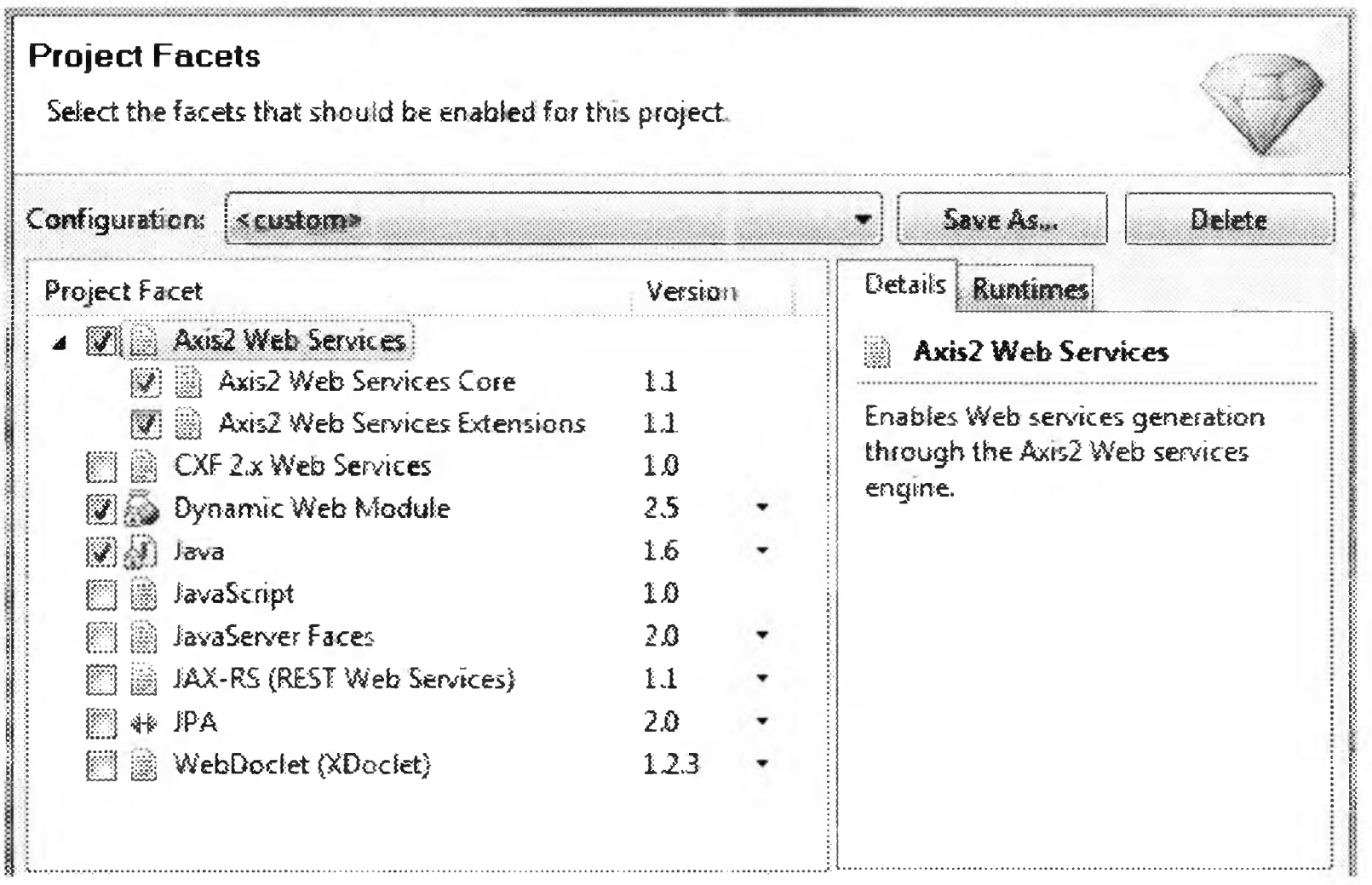

Pulsa el botón *OK* y ya habrás acabado de configurar Apache Tomcat y Axis2 dentro de Eclipse.

### *10.3.3.2. Creación un servicio web en Eclipse*

Para crear nuestro servicio web desde Eclipse comienza creando un nuevo proyecto. Para ello, selecciona la opción de menú *File/New/Dynamic Web Project* e indica en *Projet Name:* `PuntuacionesServicioWeb`.



Pulsa el botón *Modidy…* para cambiar las opciones de configuración. Añade la opción *Axis2 Web Services* tal y como se muestra a continuación:




Una vez regreses a la ventana de creación del proyecto pulsa el botón *Finish* para terminar.

Despliega el proyecto que acabas de crear y encontrarás la carpeta *Java Resources:src*. Pulsa con el botón derecho y selecciona en el menú contextual *New* > *Class*. En el campo *Package* introduce `org.example.PuntuacionesServicioWeb` y en el campo *Name* introduce `PuntuacionesServicioWeb`. Pulsa *Finish* para acabar.

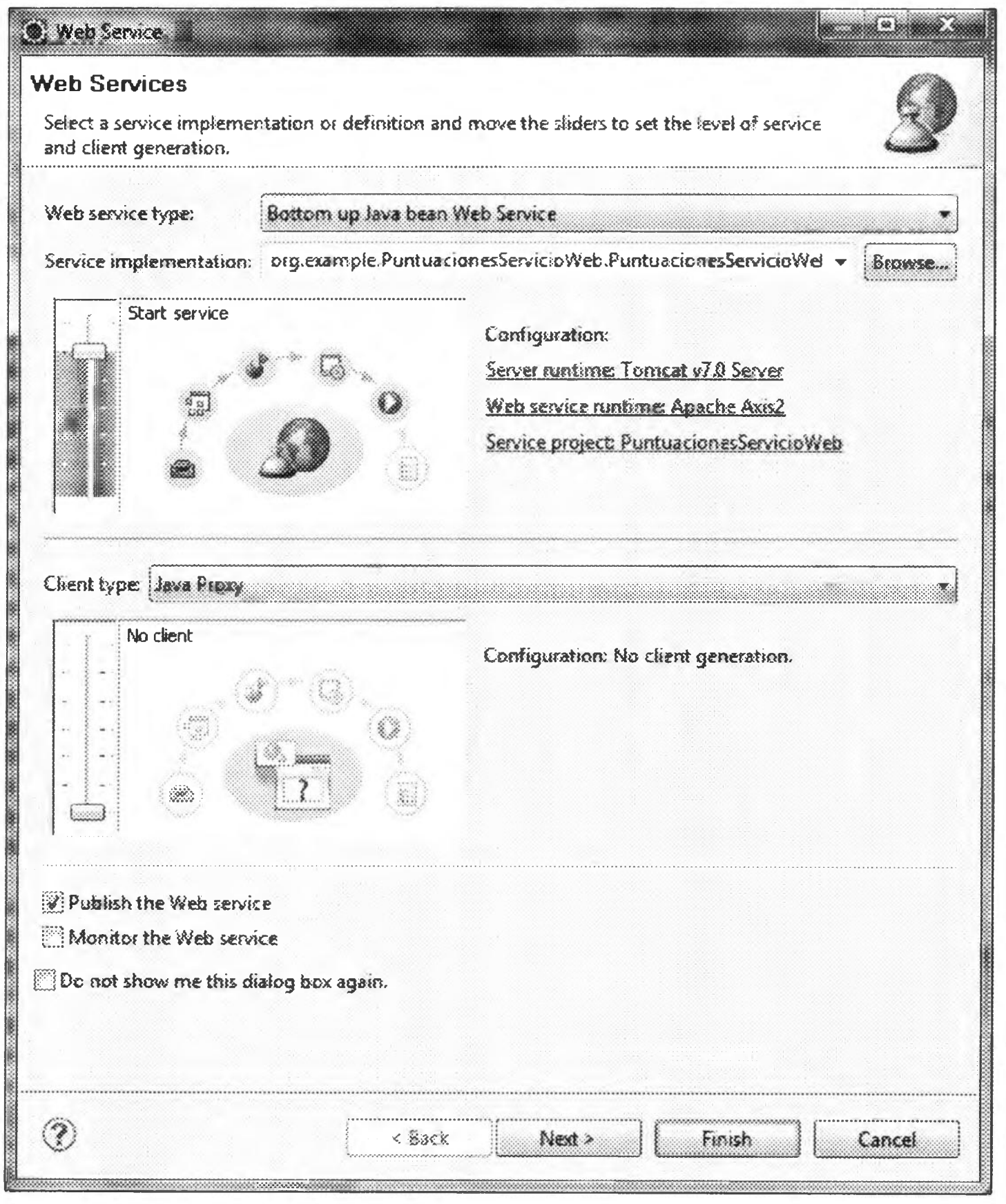
El código de esta clase ha de ser el siguiente:

```
public class PuntuacionesServicioWeb {

    static Vector<String> puntuaciones = new Vector<String>();

    public String nueva(int puntos, String nombre, long fecha) {
        puntuaciones.add(0, puntos + " " + nombre);
        return "OK";
    }

    public List<String> lista(int maximo) {
        return puntuaciones.subList(0,
                                    Math.min(puntuaciones.size(),maximo));
    }
}
```

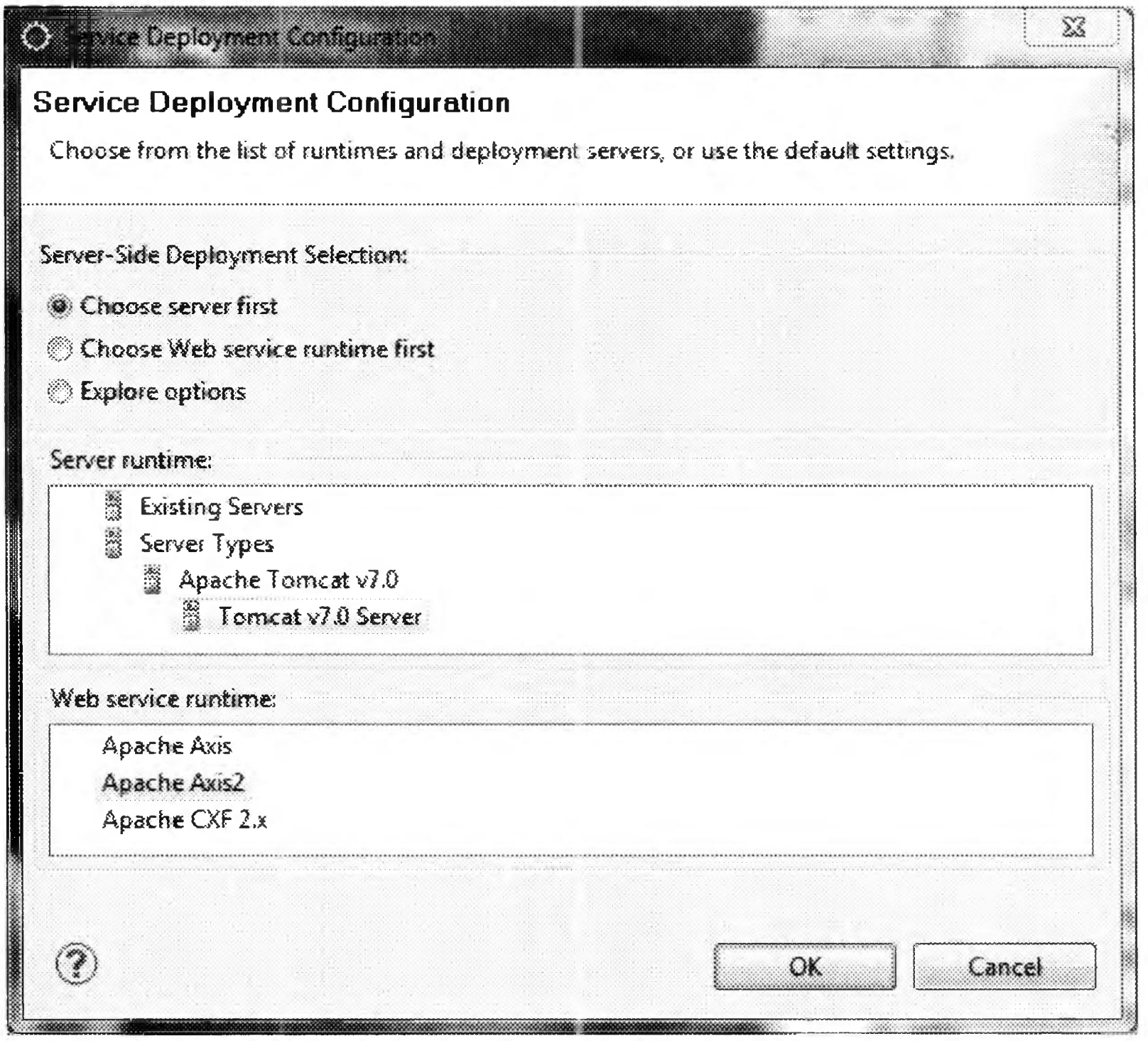
A partir de esta clase pide a Eclipse que genere el servicio web. Cada uno de sus métodos corresponderá con una operación del servicio. El método `nueva` para almacenar una nueva puntuación, y `lista` que devuelve las últimas puntuaciones almacenadas, hasta un máximo de `maximo`.

Para no alargar en exceso el código no hemos utilizado un almacenamiento permanente, como una base de datos. En su lugar hemos utilizado un simple vector de cadenas de caracteres con nombre `puntuaciones`. Esta solución también fue utilizada en el ejemplo del servidor de *sockets*. Aunque ahora el modificador `static` tiene `puntuaciones`. ¿Por qué esta diferencia? En el servidor de *sockets* solo se hace una instancia de la clase, mientras que en este ejemplo se hace una nueva instancia por cada llamada a los servicios web. Como queremos que exista una única `puntuaciones` compartida por todas las instancias de la clase, hemos de usar el modificador `static`.

Veamos ya cómo Eclipse puede convertir esta clase en un servicio web. En el explorador de archivos pulsa con el botón derecho sobre la clase que acabas de crear para desplegar el menú contextual. Selecciona *Web Services* > *Create Web Service*. Aparecerá la siguiente ventana:




Marca la opción *Publish the Web service*. Selecciona también *Web service runtime* y marca las opciones que se muestra a continuación:




Al regresar a la ventana anterior pulsa *Next*. En la siguiente ventana deja la opción que aparece por defecto *Generate a default services.xml* (más tarde veremos el contenido de este fichero) y pulsa de nuevo *Next*. La siguiente ventana te permite arrancar el servidor web pulsando el botón *Start server*. Si no está ya arrancado, arráncalo y pulsa *Next*. La siguiente ventana te permite arrancar la herramienta *Web Services Explorer*. Nosotros lo haremos a continuación manualmente, por lo que ya puedes pulsar *Finish*. Nuestro servicio web ya está creado y publicado.

### *10.3.3.3. Explorando el servicio web desde Eclipse*

Aunque tras el proceso anterior podíamos lanzar directamente la herramienta *Web Services Explorer*, creemos que es más interesante explicar el proceso de ejecución y exploración desde el principio.

En primer lugar haz click en el *Explorador de Proyectos* con el botón derecho sobre nuestro proyecto `PuntuacionesServicioWeb`. Selecciona la opción *Run As* > *Run on Server*. Aparecerá un cuadro de diálogo donde te pregunta el servidor para ejecutar el proyecto y otros detalles. Utiliza las opciones por defecto y pulsa *Finish*. Seguidamente te preguntará si quieres arrancar o rearrancar el servidor. Utiliza las opciones por defecto.

A continuación aparecerá un navegador web mostrando la dirección: http://localhost:8080/PuntuacionesServicioWeb/. Desde esta página puedes realizar varias tareas, como administrar el servidor de servicios web Axis2. Pulsa en el link

*Services* para que el servidor liste los servicios disponibles. Aparecerá la siguiente pantalla:


Como puedes ver hay dos servicios. Uno que se instala por defecto, `Version`, y el que acabamos de crear `PuntuacionesServicioWeb`. Para cada uno de los servicios se lista las operaciones disponibles.

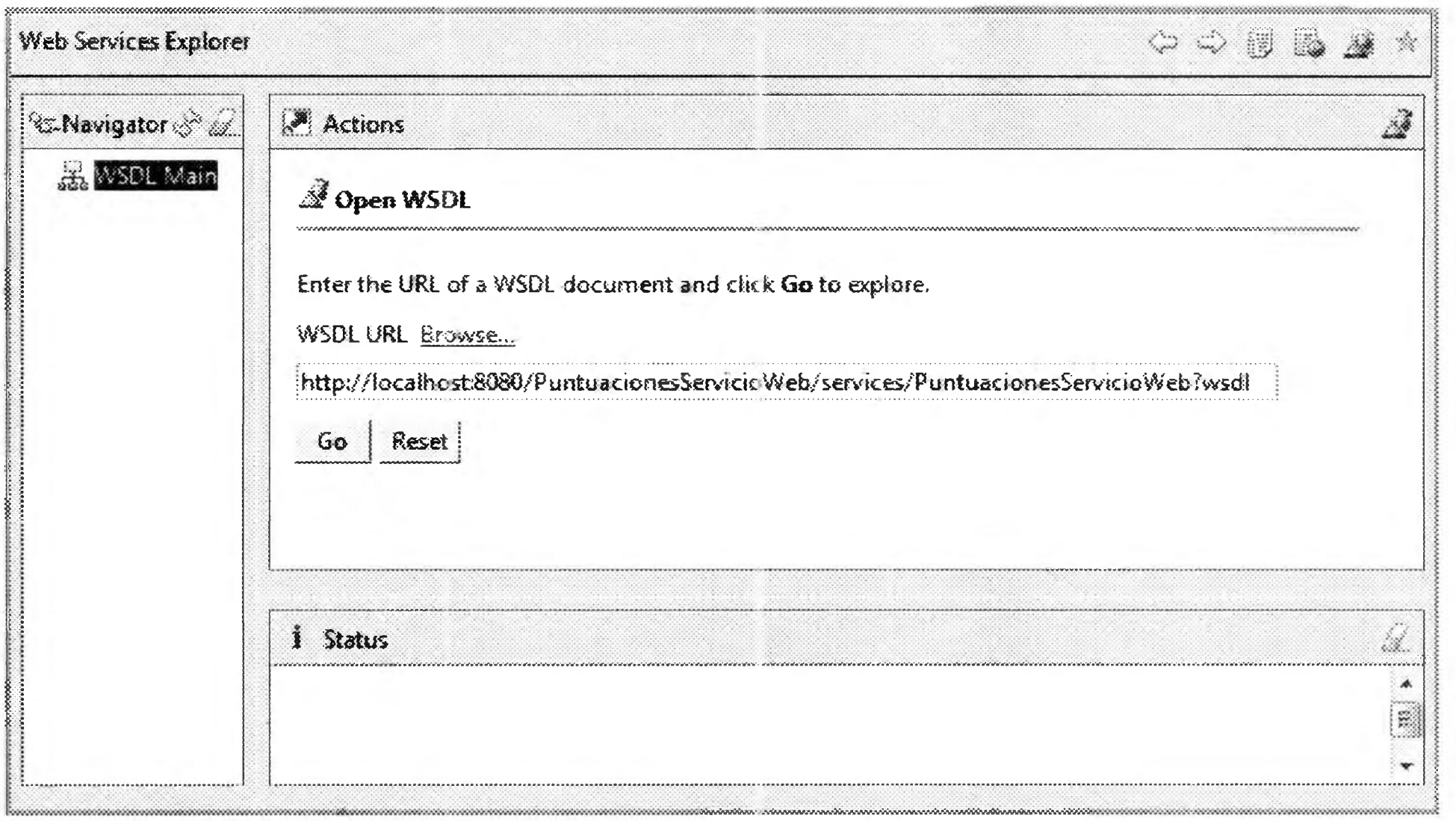
Si pulsas sobre el link *PuntuacionesServicioWeb* se visualizará el fichero WSDL asociado al servicio. WSDL se utiliza sobre todo en SOAP para describir la interfaz pública de un servicio web. Si te detienes a leerlo verás que su comprensión no es trivial. Copia en el portapapeles el link asociado a la descripción WSDL de nuestro servicio web:

http://localhost:8080/PuntuacionesServicioWeb/services/PuntuacionesServicioWeb?wsdl

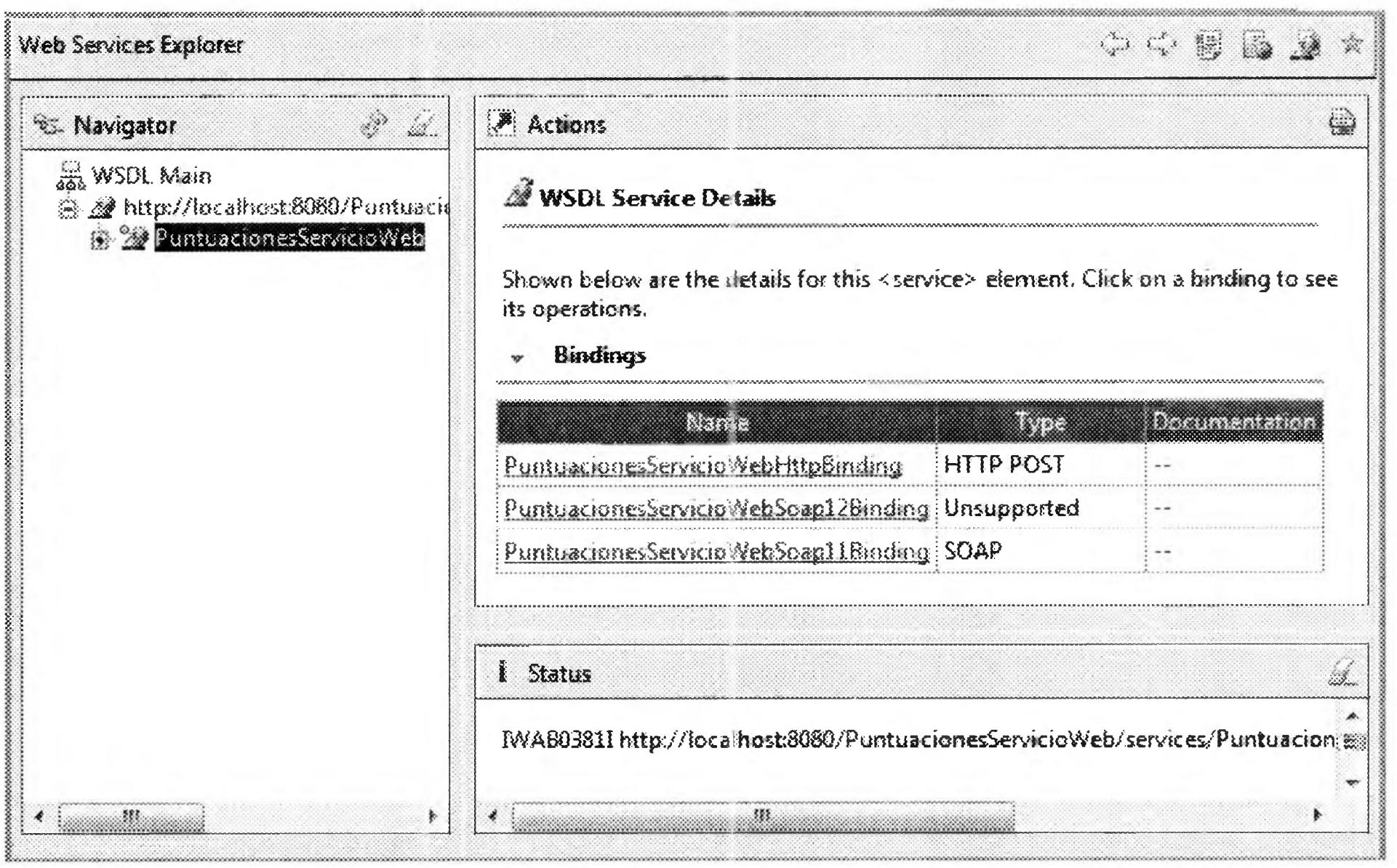
Si no tienes activa la perspectiva *JavaEE*, actívala pulsando el botón en la esquina superior derecha. En esta perspectiva disponemos del botón *Launch the Web Services Explorer* en la barra de botones; púlsalo.

Se abrirá el de exploración de servicios web. En la ventana de la izquierda tenemos el navegador. Pulsa primero en icono *WSDL Page*. Pulsa ahora con el botón derecho sobre *WSDL* y se abrirá un menú desplegable. Selecciona la opción

*Abrir*. Aparecerá la ventana *Open WSDL*. Selecciona el campo *WSDL URL* y pega el contenido del portapapeles. El resultado se muestra a continuación:

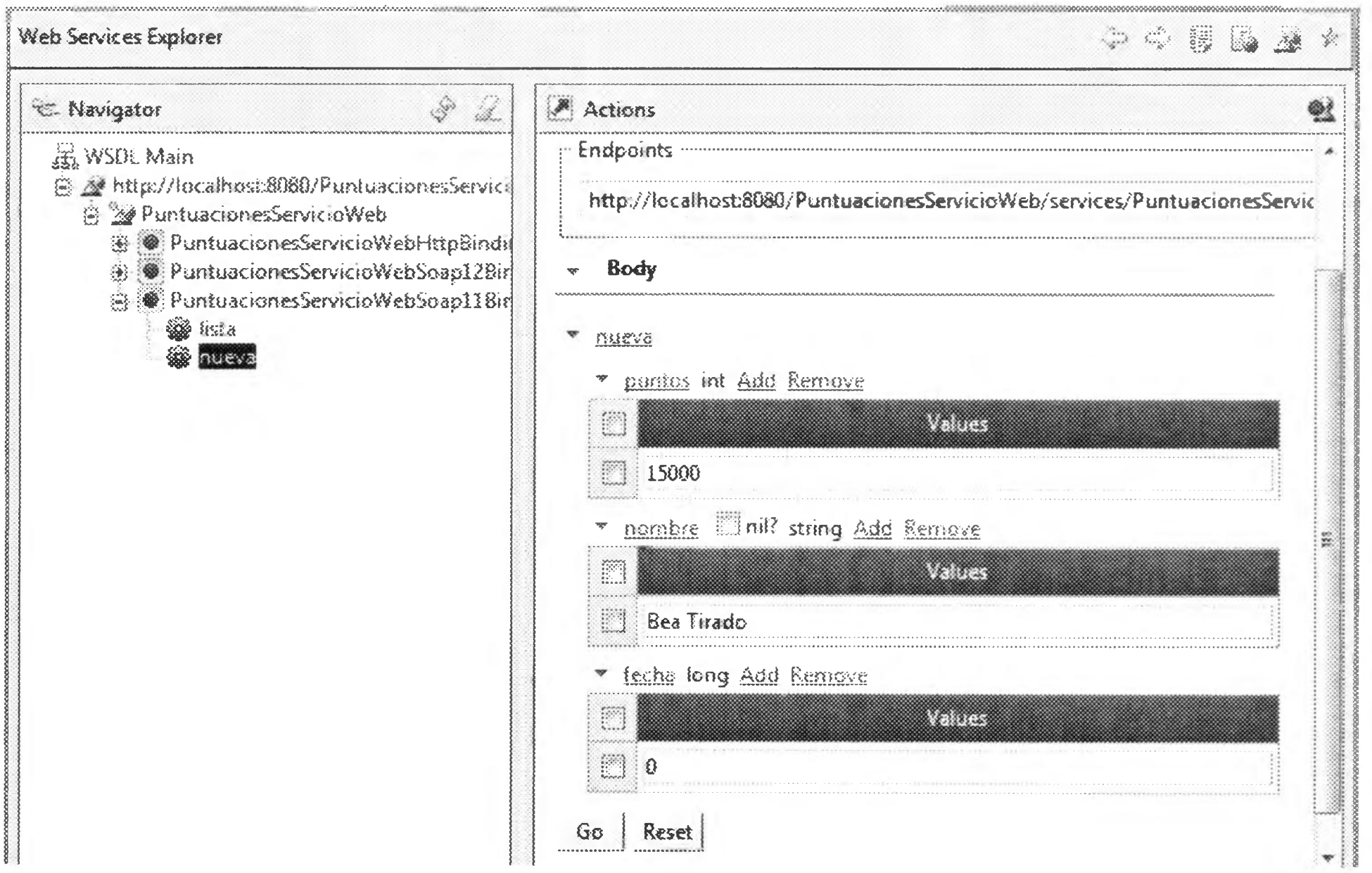



Si pulsas *Go* aparecerá la siguiente información:

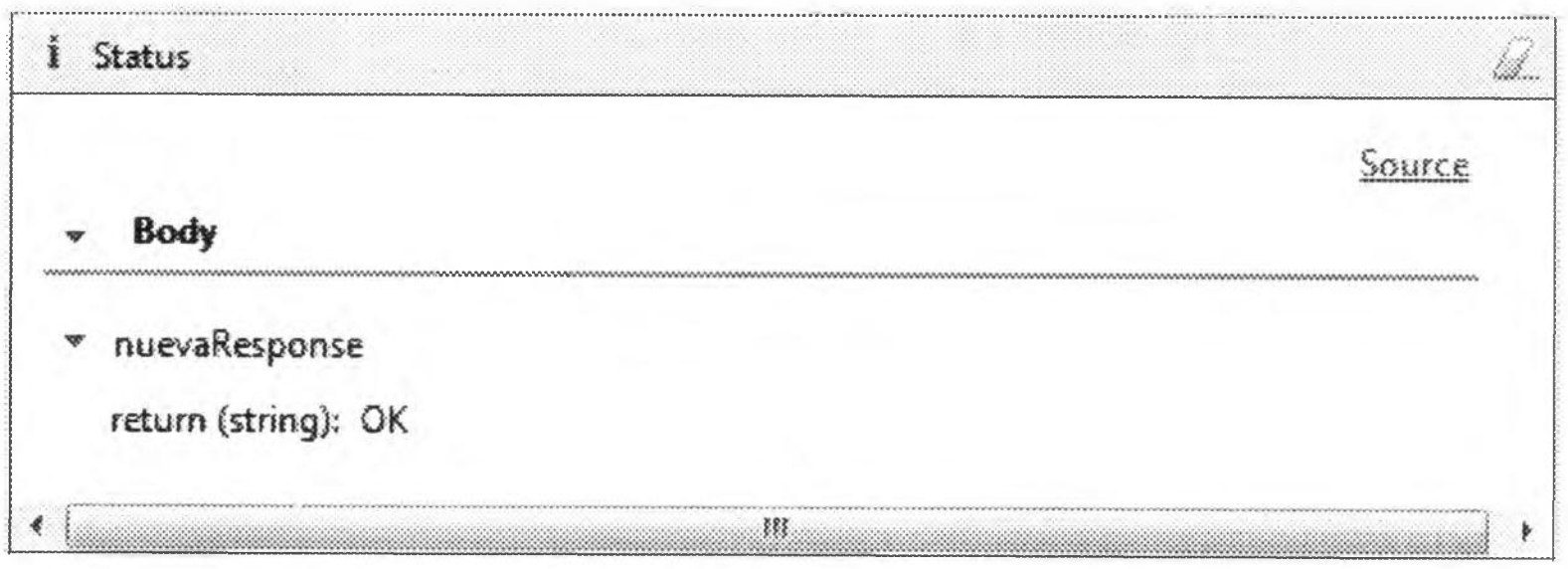



Como puedes observar en la tabla *Bindings*, nuestro servicio web es ofrecido por medio de tres mecanismos alternativos. De arriba abajo tenemos, HTTP POST (correspondería a lo que hemos llamado REST), SOAP v1.2 (no soportado por *Web Services Explorer*) y SOAP v1.1.

Pulsa sobre el último link de la tabla para explorar el servicio por medio de SOAP v1.1. Te aparecerá una tabla con las operaciones disponibles. Pulsa sobre `nueva` y aparecerá una lista con los tres parámetros de la operación. Pulsa sobre el link *Add* para añadir un valor al parámetro.




Pulsa sobre el botón *Go* para invocar el servicio web. El resultado se muestra en la ventana *Status*.




Tanto en la ventana *Actions,* como en la ventana *Status*, podrás encontrar el link *Source*. Al pulsarlo se muestra cómo se codifican los parámetros de entrada y la salida de la operación. Los parámetros de entrada del ejemplo anterior se codifican mediante el siguiente código XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
    xmlns:q0="http://PuntuacionesServicioWeb.example.org"
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
```

```
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<soapenv:Header>
</soapenv:Header>
<soapenv:Body>
    <q0:nueva>
        <q0:puntos>15000</q0:puntos>
        <q0:nombre>Bea Tirado</q0:nombre>
        <q0:fecha>0</q0:fecha>
    </q0:nueva>
    </soapenv:Body>
</soapenv:Envelope>
```

Y la salida del ejemplo anterior se codifica mediante el siguiente código XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
>
    <soapenv:Body>
        <ns:nuevaResponse
        xmlns:ns="http://PuntuacionesServicioWeb.example.org">
            <ns:return>OK</ns:return>
        </ns:nuevaResponse>
    </soapenv:Body>
</soapenv:Envelope>
```

Si lo deseas puedes usar varias veces la operación `nueva` para añadir más puntuaciones. Para ver las últimas puntuaciones añadidas utiliza la operación `lista`.

### *10.3.3.4. Explorando el servicio web desde HTML*

En el siguiente punto vamos a utilizar el servicio web que acabamos de crear desde Android. Lo haremos al estilo REST, dado que Android no incorpora las librerías necesarias para utilizar el protocolo SOAP. Empezaremos el apartado tratando de ilustrar en qué consiste una llamada al estilo REST. Para ello, nada más intuitivo que mostrarlo mediante un sencillo código HTML ejecutado en un navegador.

Crea un nuevo fichero HTML en cualquier carpeta de tu ordenador e introduce el siguiente código:

```
<html>
  <body>
    <h2>Introducir puntuación</h2>
    <form method="POST"
          action="http://localhost:8080/PuntuacionesServicioWeb/services/
            PuntuacionesServicioWeb.PuntuacionesServicioWebHttpEndpoint/nueva">
        puntos: <input name="puntos" type="text" value="1234">
        nombre: <input name="nombre" type="text" value="tu nombre">
        fecha: <input name="fecha" type="text" value="0">
```

```
        <input type="submit" value="Invocar el Servicio Web">
    </form>
    <h2>Listar puntuaciones</h2>
    <form method="POST"
          action="http://localhost:8080/PuntuacionesServicioWeb/services/
           PuntuacionesServicioWeb.PuntuacionesServicioWebHttpEndpoint/lista">
        maximo: <input name="maximo" type="text" value="6">
        <input type="submit" value="Invocar el Servicio Web">
    </form>
  </body>
</html>
```

***NOTA:** La URL de los parámetros `action` ha de encontrarse en una misma línea.*

Esta página web está formada por dos formularios independientes (`<form> ... </form>`) para invocar a las dos operaciones de nuestro servicio web. El atributo `method` puede tomar el valor GET o POST, que corresponde al método HTTP utilizado. Nuestro servicio solo es atendido mediante el método POST. El atributo `action` corresponde a la URL que será llamada cuando se pulsa el botón de tipo `submit`. Con esta llamada se incluirá como parámetros cada una de las entradas del formulario, identificadas mediante atributo `name`.

Abre el fichero HTML con un navegador web. El resultado se muestra a continuación:


Introduce algún valor en las primeras tres entradas y pulsa el primer botón. Se mostrará el siguiente código XML:

```
<ns:nuevaResponse>
  <ns:return>OK</ns:return>
</ns:nuevaResponse>
```

Retrocede con el navegador a la página anterior y pulsa el segundo botón. Dependiendo de los datos introducidos el resultado podría ser el siguiente:

```
<ns:nuevaResponse>
    <ns:return>1234 tu+nombre</ns:return>
```

```
        <ns:return>3500 Maya Tomas</ns:return>
        <ns:return>1500 Bea Tirado</ns:return>
    </ns:nuevaResponse>
```

***NOTA:*** *En los elementos introducidos desde el navegador web los espacios en blanco son remplazados por el carácter "+". Este hecho se debe a que cuando se utiliza el método GET del protocolo HTTP, la URL resultante no puede contener espacios en blanco, por lo que son remplazados por este carácter. Este hecho no afecta a nuestro ejemplo, dado que solo accederemos al servicio web desde Android.*

En resumen, invocar nuestro servicio web resulta muy sencillo. No tenemos más que pedir al servidor la URL asociada a una operación con los parámetros adecuados y el servidor nos devuelve el resultado codificado en XML.

En el siguiente punto realizaremos esta tarea desde Android.

### *10.3.3.5. Utilizando el servicio web desde Asteroides*

Como en todos los ejemplos de este tema asegúrate que la aplicación solicita el permiso de acceso a Internet, añadiendo la siguiente línea en `AndroidManifest.xml`.

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

Pasemos a implementar la interfaz `AlmacenPuntuaciones` accediendo al servidor de servicios web que acabamos de desarrollar. Para ello, crea una nueva clase en la aplicación Asteroides y copia el siguiente código:

```
public class AlmacenPuntuacionesSerWeb implements AlmacenPuntuaciones {

public Vector<String> listaPuntuaciones(int cantidad) {
    try {
        URL url = new URL("http:// X.X.X.X:8080/"
          + "PuntuacionesServicioWeb/services/PuntuacionesServicioWeb."
          + "PuntuacionesServicioWebHttpEndpoint/lista");
        HttpURLConnection conexion = (HttpURLConnection) url
                                              .openConnection();
        conexion.setRequestMethod("POST");
        conexion.setDoOutput(true);
        OutputStreamWriter sal = new OutputStreamWriter(
                                          conexion.getOutputStream());
        sal.write("maximo=");
        sal.write(URLEncoder.encode(String.valueOf(cantidad),"UTF-8"));
        sal.flush();
        if (conexion.getResponseCode() == HttpURLConnection.HTTP_OK) {
            SAXParserFactory fabrica = SAXParserFactory.newInstance();
            SAXParser parser = fabrica.newSAXParser();
            XMLReader lector = parser.getXMLReader();
            ManejadorSerWeb manejadorXML = new ManejadorSerWeb();
            lector.setContentHandler(manejadorXML);
```

```
            lector.parse(new InputSource(conexion.getInputStream()));
            return manejadorXML.getLista();
        } else {
            Log.e("Asteroides", conexion.getResponseMessage());
            return null;
        }
    } catch (Exception e) {
        Log.e("Asteroides", e.getMessage(), e);
        return null;
    }
}

class ManejadorSerWeb extends DefaultHandler {
    private Vector<String> lista;
    private StringBuilder cadena;

    public Vector<String> getLista() {
        return lista;
    }

    @Override
    public void startDocument() throws SAXException {
        cadena = new StringBuilder();
        lista = new Vector<String>();
    }

    @Override
    public void characters(char ch[], int comienzo, int longitud){
        cadena.append(ch, comienzo, longitud);
    }

    @Override
    public void endElement(String uri, String nombreLocal,
            String nombreCualif) throws SAXException {
        if (nombreLocal.equals("return")) {
            try {
                lista.add(URLDecoder.decode(cadena.toString(),"UTF8"));
            } catch (UnsupportedEncodingException e) {
                Log.e("Asteroides", e.getMessage(), e);
            }
        }
        cadena.setLength(0);
    }
}
```

El primer método se encarga de invocar la operación `lista` del servicio web que acabamos de implementar. Comienza definiendo la URL correspondiente al servicio web. En el código hay que reemplazar `"X.X.X.X"` por la dirección IP de tu ordenador. Recuerda que este programa lo ejecutarás desde el emulador o desde un teléfono real y, en ambos casos, la IP será diferente a la de tu ordenador. Esto imposibilita utilizar como dirección `localhost`, como sí hicimos con otros clientes que ejecutábamos desde el mismo ordenador.

Una vez creada la URL se establece la conexión y se manda mediante el método POST el parámetro correspondiente. La respuesta está en XML, por lo que utilizamos un objeto de la clase `SAXParser` para procesarla. Con este fin es creada la clase `ManejadorSerWeb`, que ha de esperar el siguiente formato XML:

```
<ns:nuevaResponse>
    <ns:return>resultado 1</ns:return>
    <ns:return>resultado 2</ns:return>
    ...
</ns:nuevaResponse>
```

La clase utiliza la variable `lista` para ir añadiendo los resultados encontrados. Una vez procesado el fichero, el método `getLista()` permite obtener esta lista.

Pasemos a ver el segundo método de la clase:

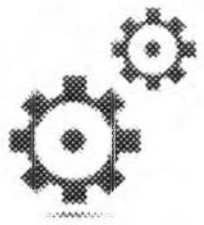
```
public void guardarPuntuacion(int puntos, String nombre, long fecha) {
    try {
        URL url = new URL("http://158.42.38.166:8080/"
        + "PuntuacionesServicioWeb/services/PuntuacionesServicioWeb."
        + "PuntuacionesServicioWebHttpEndpoint/nueva");
    HttpURLConnection conexion = (HttpURLConnection) url
                                                .openConnection();
    conexion.setRequestMethod("POST");
    conexion.setDoOutput(true);
    OutputStreamWriter sal = new OutputStreamWriter(
                                        conexion.getOutputStream());
    sal.write("puntos=");
    sal.write(URLEncoder.encode(String.valueOf(puntos), "UTF-8"));
    sal.write("&nombre=");
    sal.write(URLEncoder.encode(nombre, "UTF-8"));
    sal.write("&fecha=");
    sal.write(URLEncoder.encode(String.valueOf(fecha), "UTF-8"));
    sal.flush();
    if (conexion.getResponseCode() == HttpURLConnection.HTTP_OK) {
        SAXParserFactory fabrica = SAXParserFactory.newInstance();
        SAXParser parser = fabrica.newSAXParser();
        XMLReader lector = parser.getXMLReader();
        ManejadorSerWeb manejadorXML = new ManejadorSerWeb();
        lector.setContentHandler(manejadorXML);
        lector.parse(new InputSource(conexion.getInputStream()));
        if (manejadorXML.getLista().size() != 1
                || !manejadorXML.getLista().get(0).equals("OK")) {
```

```
                Log.e("Asteroides","Error en respuesta servicio Web nueva");
            }
        } else {
            Log.e("Asteroides", conexion.getResponseMessage());
        }
        conexion.disconnect();
    } catch (Exception e) {
        Log.e("Asteroides", e.getMessage(), e);
    }
    }
}
```

La estructura de este método es similar al anterior, pero ahora llamamos a la operación `nueva`. Igual que antes recuerda poner tu dirección IP. Recuerda también que en caso de una llamada satisfactoria la respuesta ha de ser:

```
<ns:nuevaResponse>
    <ns:return>OK</ns:return>
</ns:nuevaResponse>
```

Dado que este formato es muy similar al anterior vamos a poder utilizar el mismo manejador definido en la clase `ManejadorSerWeb`. Como hemos visto, este manejador nos devuelve una lista de *Strings* con los resultados. Consideraremos que ha habido un error si el tamaño de la lista es diferente de uno, o si habiendo un elemento, este no es `"OK"`.



**Práctica:** *Acceso a servicio Web de puntuaciones con DOM.*

Modifica la clase `AlmacenPuntuacionesSerWeb` para que en lugar de procesar la respuesta XML mediante SAX, lo haga mediante DOM. Comenta las ventajas e inconvenientes de las dos implementaciones.

En este capítulo hemos utilizado dos alternativas: *sockets* y servicios web, para resolver un mismo problema. En la mayoría de los casos es más recomendable utilizar servicios web. Veamos las ventajas de un servicio web frente a un servidor de *sockets*:

- La principal ventaja de los servicios web es la **claridad de diseño**. Si comparas los dos códigos resulta mucho más sencillo de entender el que utiliza servicios web. Además, para acceder al servicio resulta también mucho más sencillo utilizar un método estándar muy conocido basado en URL, que tener que crear nuestro propio protocolo.
- Otra ventaja es el **aprovechamiento de las cabeceras HTTP**. Como se comentó en el apartado anterior, el protocolo HTTP incorpora una serie de cabeceras para ofrecer información adicional en el intercambio. Mediante éstas podemos controlar aspectos muy importantes como solicitar la autentificación del cliente, utilizar un modo seguro de transferencia (https), definir el tipo de información transmitida o controlar si queremos que las

peticiones a nuestros servicios sean recordadas, y por cuánto tiempo, en la caché del cliente.

- El uso de **servidores comerciales** en los servicios web nos proporcionan grandes ventajas, que serían complejas de implementar en nuestro servidor de *sockets.* Por ejemplo, la seguridad, la escalabilidad o facilidades de gestión se incluyen en un servidor web como Apache.

- Ambos servicios han de ofrecerse a través de un puerto. Los servicios web suelen utilizar el **mismo puerto que los servidores web**, el 80. Esto presenta la ventaja de tratarse de un puerto que raramente es filtrado por los cortafuegos. Esta ventaja puede también utilizarse en un servidor por *sockets* si le asignamos este puerto. Pero en este caso, ya no podrás instalar en la misma máquina un servidor web

# CAPÍTULO 11.
# Publicar aplicaciones

Una vez terminada tu aplicación puedes plantearte la posibilidad de publicarla para que otros usuarios puedan utilizarla. Antes de esto, tendrás que preparar y testear tu aplicación y, a continuación, firmarla con un certificado digital. Para difundir tu aplicación puedes utilizar Internet o el servicio de *Google Play Store*. De esta forma millones de usuarios de todo el mundo podrán descargar tu aplicación. Incluso puedes cobrar por ella. Para poder realizar este proceso hay que seguir una serie de pasos que son descritos a lo largo de este capítulo.

**Objetivos:**

- Insistir en la necesidad de preparar y testear una aplicación antes de publicarla.
- Mostrar cómo se crea un certificado digital para firmar la aplicación.
- Comparar la publicación de aplicaciones en Internet y en Google Play Store.

# 11.1. Preparar y testear tu aplicación

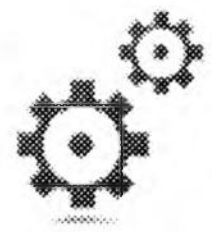
Antes de dar por finalizada tu aplicación tendrás que asegurarte que va a funcionar correctamente en diferentes entornos y dispositivos. Los aspectos más importantes que tendrás que considerar son: elección de la versión de SDK, traducción a diferentes idiomas, adaptar los gráficos a diferentes resoluciones/densidades gráficas y seleccionar métodos de entrada (sensores, pantalla táctil, teclado,...) para así cubrir la mayoría de dispositivos.

Una vez considerados estos aspectos tendrás que verificar sobre diferentes dispositivos físicos y virtuales que todo funciona correctamente.

## 11.1.1. Preparar la aplicación para distintos tipos de dispositivo

Una de las primeras decisiones que has de tomar es la versión de la plataforma que vas a utilizar para compilar la aplicación. A mayor versión de plataforma, menor será el número de usuarios que podrán instalarla, aunque en contrapartida dispondrás de un API más rico. A no ser que sea imprescindible utilizar alguna de las características de las últimas versiones de SDK, conviene ser conservador y utilizar versiones antiguas. Para consultar el porcentaje de utilización de cada versión de la plataforma, puedes acceder a http://developer.android.com.

Como hemos estudiado en el capítulo 2, Android dispone de herramientas que facilitan la difusión de aplicaciones en múltiples lenguas. Si quieres abarcar un gran número de usuarios has de traducir la aplicación, tanto al inglés, como al mayor número de lenguas posibles.



**Práctica:** *Traducción de Asteroides.*

1. Verifica que toda información de texto de la aplicación Asteroides está almacenada en un recurso dentro de la carpeta `res/values/strings.xml`. En caso contrario, crea el recurso de texto correspondiente
2. Asegúrate que todas estas cadenas están traducidas en la carpeta `res/values-en/strings.xml`.

Una importante y frecuente fuente de problemas la encontraremos en lo relativo al tamaño, y resolución gráfica de la pantalla. Sería interesante que nuestra aplicación funcionara sin problemas en las resoluciones más frecuentes.

Android distingue las siguientes caracteriza en una pantalla[1]:

**Tamaño de ventana:** medida física de la diagonal de la pantalla en pulgadas. Por simplicidad se definen de forma genérica 4 posibles valores:

[1] http://developer.android.com/guide/practices/screens_support.html

`small`: 2 - 3,5 pulgadas (teléfonos pequeños)
`normal`: 3 - 4,5 pulgadas (teléfonos grandes, PDAs)
`large`: 4,2 - 7 pulgadas (tabletas)
`xlarge`: 7 - 10,5 pulgadas (network PC, tabletas grandes)

**Ratio de aspecto:** relación entre las medidas físicas de alto y ancho de la pantalla. Se definen dos valores: `long` y `notlong`.

**Resolución:** número total de píxeles en pantalla. Algunas de las resoluciones se indican a continuación:

`QVGA` – (200x320)
`FWQVGA` – (240x432)
`HVGA` – (320x480)
`WVGA` – (480x800)
`FWVGA` – (480x854)
`WXGA` – (1280×800)

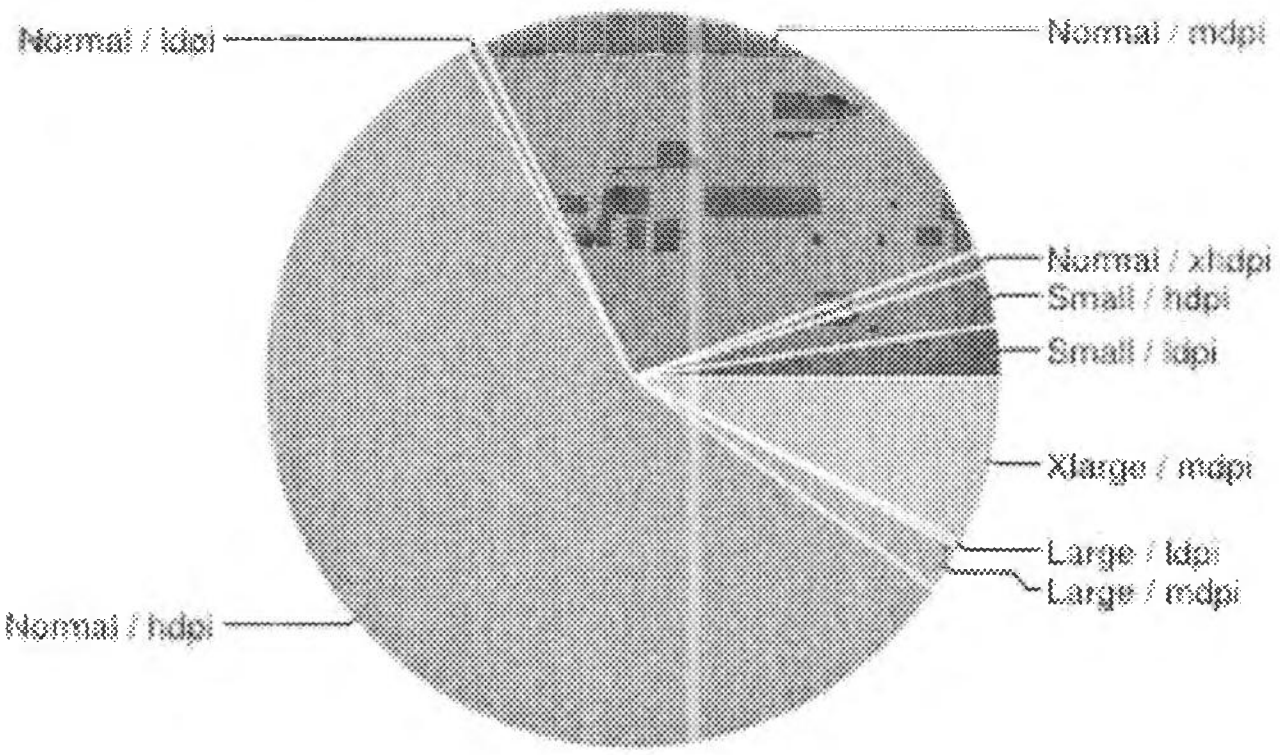
**Densidad:** cantidad de píxeles por unidad de superficie. Se mide en dpi (o en castellano ppp, puntos por pulgada) Resulta un parámetro de vital importancia cuando trabajamos con imágenes en mapa de bits. Por ejemplo, un mismo icono definido con una determinada resolución puede quedar muy pequeño en una pantalla con una alta densidad y muy grande si tiene baja densidad. Para solucionar este problema es frecuente definir el mismo gráfico con diferentes resoluciones adaptándose a la densidad.

`ldpi`: 100 - 130 dpi
`mdpi`: 120 - 190 dpi
`hdpi`: 180 - 270 dpi
`xhdpi`: 260 - 340 dpi

En algunos casos será necesario disponer de *Layouts* o imágenes de fondo alternativas en función de la resolución o la densidad. En estos casos has de hacer uso de los recursos alternativos estudiados en el capítulo 2. En caso de que tu aplicación no pueda soportar todo tipo de pantallas, será necesario indicar qué resoluciones o densidades gráficas no están soportadas. Puedes utilizar la etiqueta `<support-screen>` de `AndroidManifest.xml` para indicar estas restricciones. Por ejemplo, si quieres que tu aplicación solo esté disponible para terminales con pantallas grades utiliza el siguiente código:

```
<manifest ... >
    ...
    <supports-screens android:smallScreens="false"
                      android:normalScreens="false"
                      android:largeScreens="false"
                      android:xlargeScreens="true" />
    <application ... >
        ...
    <application>

</manifest>
```

En *Android Developers*[2] puedes encontrar estadísticas sobre los tamaños de pantalla y densidades gráficas de los dispositivos que accedieron a *Google Play Store* en las últimas dos semanas:




| | ldpi | mdpi | hdpi | xhdpi |
|---|---|---|---|---|
| **small** | 2.3% | | 2.4% | |
| **normal** | 0.7% | 26.2% | 57.8% | 0.9% |
| **large** | 0.3% | 2% | | |
| **xlarge** | | 7.4% | | |

***Figura 1:*** *Tamaños de pantalla y densidades gráficas de los dispositivos Android que han accedido a Google Play Store durante dos semanas terminando el 2 de abril de 2012.*

**Práctica:** *Adaptando los gráficos en Asteroides.*

1. Remplaza el fondo de pantalla de la actividad `Juego` en la aplicación Asteroides. Este ha de estar disponible en varias resoluciones para adaptarse a los tamaños de pantalla.

Los métodos de entrada también son muy variados en los dispositivos Android. Algunos disponen de teclado físico, *TrackBall*, teclas de cursor, etc. Por otra parte, no todos disponen del mismo número de sensores. Una buena idea puede consistir en utilizar diferentes alternativas para recibir entradas del usuario. Dispones de algunos ejemplos en el capítulo 5.

## 11.1.2. Testear la aplicación

Antes de tomar la decisión de publicar una aplicación hay que reflexionar sobre si está lo suficientemente madura. Si nos precipitamos, y la aplicación tiene fallos, los

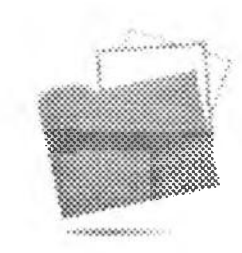
[2] http://developer.android.com/resources/dashboard/screens.html

usuarios quedarán decepcionados y seguramente la desinstalen y asocien a su creador o nuestra empresa con mala calidad.

El entorno de desarrollo Android incluye un marco de testeo integrado que puede ayudarnos en este trabajo. El marco de testeo está basado en JUnit y nos proporciona una arquitectura y útiles herramientas muy interesantes a la hora de planificar y realizar los test. Para más información consultar: http://developer.android.com/guide/topics/testing.

Existen infinidad de dispositivos capaces de ejecutar aplicaciones Android: teléfonos móviles, tabletas, network PC, Google TV,... Las características de estos dispositivos son muy variadas. Con el fin de verificar que la aplicación funciona sin problemas en los distintos tipos de dispositivo que te interese, va a ser imprescindible realizar pruebas en varios dispositivos reales.

También podemos usar el emulador para probar nuestra aplicación en varios dispositivos virtuales. Algunos fabricantes han publicado dispositivos virtuales que emulan ciertos dispositivos reales. Por ejemplo, si accedes en Eclipse al menú *Windows > Android SDK and AVD manager > Available packages > Third party Add-ons > Samsung Electronics add-ons > Galaxy Tab*, podrás instalarte un AVD para la tableta "Galaxy Tab".



**Recursos adicionales:** *A tener en cuenta para preparar y testear tu aplicación.*

- Si lo estimas oportuno traduce los textos de tu aplicación a varios idiomas. Utiliza el inglés como idioma por defecto.
- Verifica que el atributo `android:label` identifica tu aplicación adecuadamente. Ha de ser adecuado en todos los idiomas.
- Crea un icono para tu aplicación.
- Elimina los Log de la aplicación.
- Decide el nombre del paquete y cómo se codificarán las versiones. Para ello edita los siguientes parámetros de AndroidManifest.xml.

```
<manifest
    package="com.empresa.aplicacion"
    android:versionCode="1"
    android:versionName="1.0" >
```

`package` Se utiliza como identificador único de la aplicación. Piensa con cuidado el nombre del paquete, no podrá ser modificado una vez publicado. No puede empezar por com.example/org.example o prefijos utilizados por otras empresas como com.google.

`versionCode` Ha de ser un entero que aumente con cada nueva versión de la aplicación. Puedes utilizar la codificación que prefieras, siempre que sigas la norma anterior. No se muestra al usuario.

`versionName` Solo se utiliza para mostrar la versión al usuario. Puede ser cualquier cadena de caracteres.

`installLocation` A partir del nivel de API 8 puedes decidir si la aplicación se instala en la memoria interna, externa o según prefiera el usuario. Existen tres valores posibles:

"`internalOnly`" Solo en memoria interna (valor por defecto)

"`preferExternal`" Se prefiere en memoria externa (no se garantiza)

"`auto`" Si hay espacio se instalará en la memoria interna, si no en la externa. Luego el usuario puede moverla.

Cuando se instala en la memoria externa los datos de la aplicación se almacenan en la memoria interna.

- Verifica que se ha escogido la versión menor posible de SDK según las APIs utilizadas en la aplicación (1.6 o 2.0 son valores suficientemente pequeños)

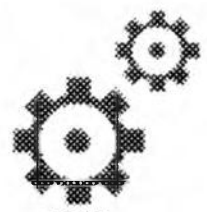
```
<uses-sdk android:minSdkVersion="4" />
```

Puede ser interesante crear varias versiones para diferentes niveles de API.

- Verifica que los permisos solicitados en etiquetas `<uses-permission>` son realmente necesarios.
- Si utilizas servicios de terceros es posible que necesites una nueva clave para la aplicación. Por ejemplo, con Google Maps o otros servicios Web.
- Puede ser interesante preparar la licencia de uso (End User License Agreement - EULA) para proteger tu persona, tu empresa o la propiedad intelectual. Si publicas en Play Store dispones de un servicio sencillo de utilizar (http://developer.android.com/guide/publishing/licensing.html).
- Prueba la aplicación en varios dispositivos (tanto teléfonos como tabletas). Conviene verificar: cambios de orientación, cambios de configuración, ciclo de vida de las actividades, que no se consume excesiva batería, acceso a redes, ...

  (http://developer.android.com/guide/topics/testing/what_to_test.html)

- Prepara los recursos para la promoción: Capturas de pantalla, videos, textos explicativos, ...



**Práctica:** *Preparar y testear Asteroides.*

Aplica los pasos de la lista anterior para verificar que la aplicación Asteroides cumple las condiciones para ser publicada.

## 11.2. Crear un certificado digital y firmar la aplicación

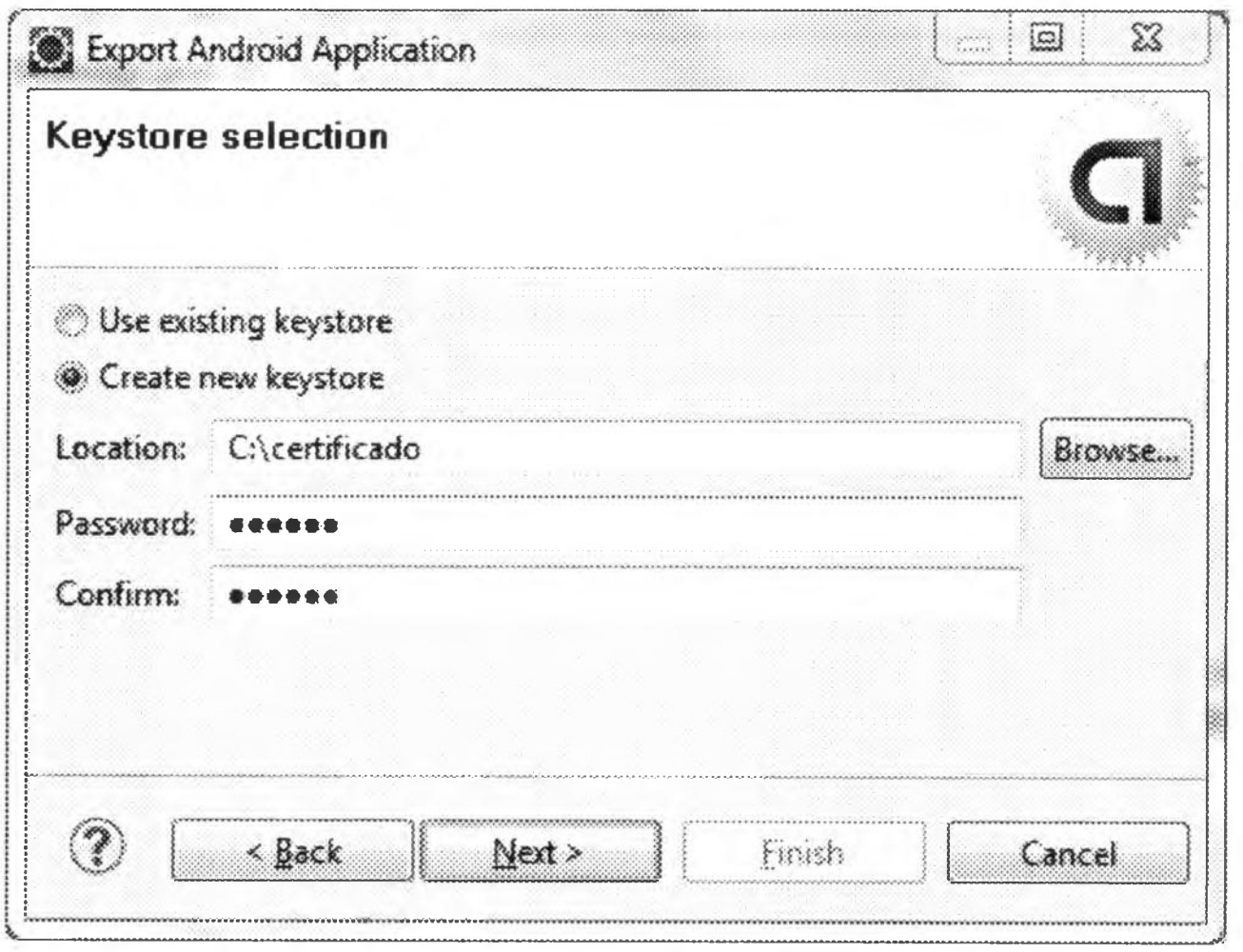
Las aplicaciones han de ser firmadas con un certificado digital para que puedan ser instaladas en un dispositivo Android. La firma digital, permite identificar al autor e

impide que la aplicación pueda ser manipulada. Puedes encontrar información detallada en esta página de Android[3].

Durante la fase de desarrollo sí que puedes ejecutar las aplicaciones en el terminal; esto se consigue usando un certificado digital que genera el SDK para este propósito. Sin embargo, no puedes usar este certificado para publicar tu aplicación. Es imprescindible que generes tu propio certificado. Crear este certificado es muy sencillo. Además no es necesario que lo valide una autoridad de certificación.

Todo certificado digital tiene una fecha de expiración. La fecha de expiración solo afecta al momento de instalación. Una vez instalada nunca caduca la aplicación. Google exige un período de 25 años de validez del certificado para poder publicar la aplicación en Play Store.

La forma más sencilla de crear tu certificado digital va a ser desde Eclipse. No tienes más que seguir los siguientes pasos: accede a la opción de menú *File>Export>Android>Export Android Aplication* y selecciona la aplicación que deseas firmar; a continuación te pregunta si deseas utilizar un certificado ya creado o crear uno nuevo. En nuestro caso seleccionamos crear uno nuevo e indicamos el fichero donde queremos crearlo:

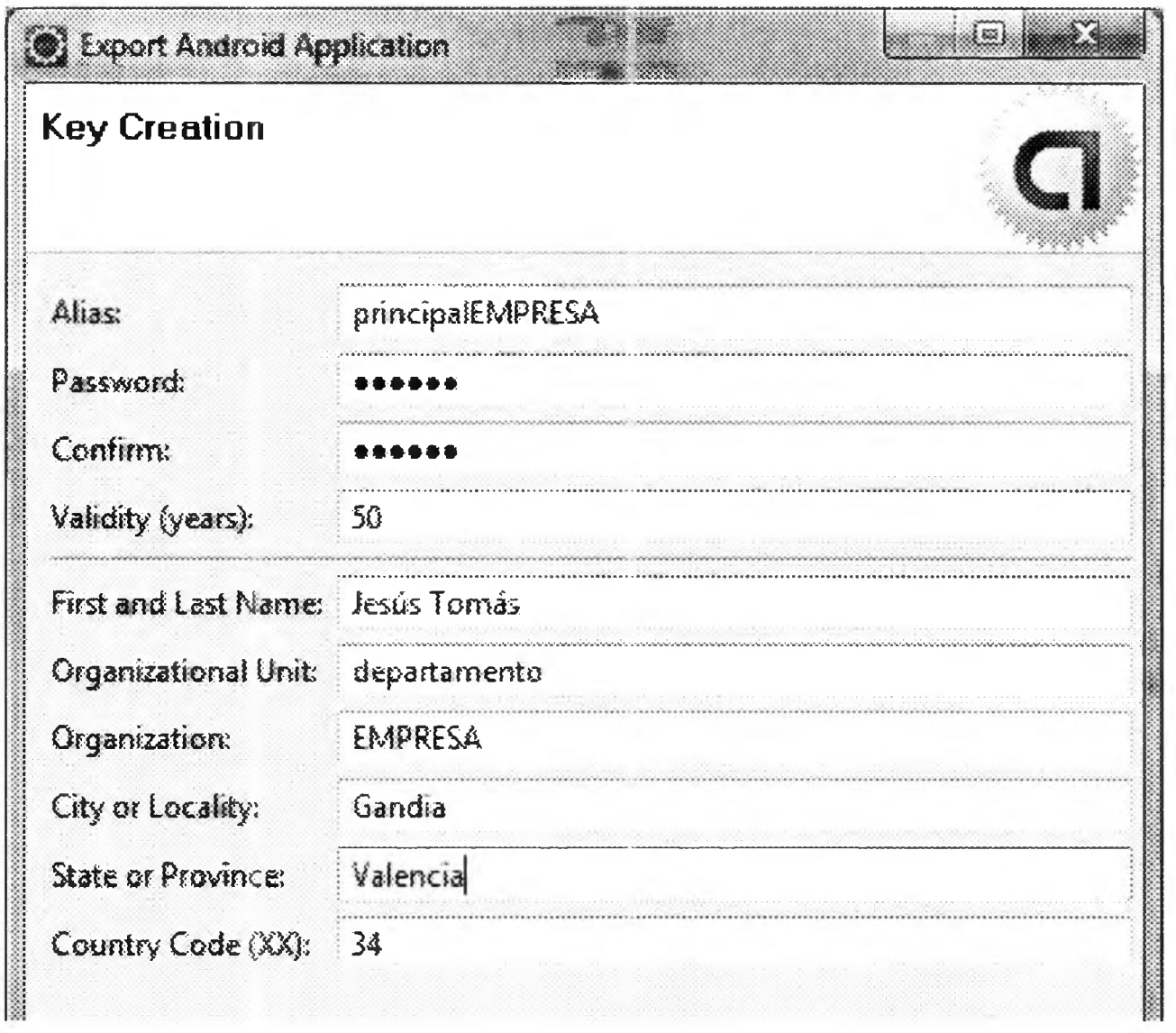



Como medida de seguridad te pide una contraseña para poder acceder a este fichero. La siguiente pantalla te pide información para crear una nueva clave que será incluida en el fichero anterior.

Como puedes comprobar, te pide una nueva contraseña específica de esta clave. Recuerda que Android exige un período de validez para la clave de al menos 25 años. Con respecto a los campos siguientes, no resulta imprescindible rellenarlos todos.

---

[3] http://developer.android.com/guide/publishing/app-signing.html




Para terminar, te pedirá el nombre de un fichero `.apk` donde guardará el paquete de tu aplicación ya firmada con el certificado digital que acabamos de crear. Este es el fichero que tendrás que distribuir.

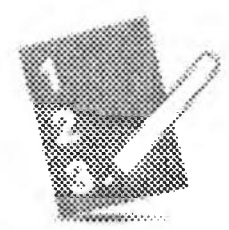
Verifica que se han creado ambos ficheros: el certificado digital y el paquete firmado. Es muy importante que guardes el certificado en lugar seguro. Nadie puede tener acceso a él. También es muy importante que no lo pierdas y recuerdes las contraseñas. Tendrás que utilizarlo cada vez que quieras publicar actualizaciones de tu aplicación. También resulta conveniente, aunque no obligatorio, que firmes todas tus aplicaciones con el mismo certificado.

## 11.3. Publicar la aplicación

### 11.3.1. Publicar en Internet

Una posibilidad para dar a conocer tu aplicación es publicarla en Internet. Es decir, subir el archivo compilado (`.apk`) a alguna web y publicarlo en un foro o blog.

Tienes que tener en cuenta que esta es una opción no apta para usuarios novatos. El usuario deberá bajar la aplicación, pasarla a una memoria exterior e instalarla desde un navegador de archivos. Además tendrá que haber configurado su móvil para aceptar aplicaciones de terceros no provenientes de Play Store.



**Ejercicio paso a paso:** *Instalar una aplicación.*

1. Copia el fichero `.apk` creado en el apartado anterior a tu teléfono móvil. Puedes usar una tarjeta micro SD, el cable USB o una conexión de red.
2. Utiliza un programa que permita la instalación de aplicaciones para instalar la aplicación. En las versiones antiguas de Android, no se incluye ninguno en el sistema, puedes descargar ASTRO desde Play Store.

### 11.3.2. Publicar en Google Play Store

Los desarrolladores de Android disponen de otra alternativa para publicar sus aplicaciones, *Google Play Store*[4].

*Play Store* antes llamado *Android Market* fue lanzado al público el 22 de octubre del 2008. El soporte para las aplicaciones de pago fue añadido para los usuarios de los Estados Unidos y para desarrolladores también en el Reino Unido a mediados del mes de febrero de 2009. El soporte para los usuarios en España fue lanzado el 13 de marzo de 2009 y para desarrolladores a finales de mayo del mismo año.

Android Market tiene un acceso fácil y rápido a sus aplicaciones. Las aplicaciones son creadas por desarrolladores de todo el mundo y posteriormente puntuadas por los usuarios de Android.

El usuario dispone de las categorías de juegos y aplicaciones, proporcionan submenús para que la búsqueda sea más sencilla. En "Mis descargas", visualiza las aplicaciones que están instaladas en el dispositivo.

Los usuarios tienen la posibilidad de valorar las aplicaciones mediante un sistema similar al YouTube, con una escala del 1 al 5, también ofrece la posibilidad de poner comentarios sobre la aplicación.

Android Market distribuye tanto aplicaciones gratuitas como de pago. Los precios permitidos son entre 0,99 y 200 dólares. El precio de la aplicación se puede cambiar en cualquier momento siempre y cuando no la hayas publicado anteriormente como gratuita. El autor recibe un 70% del precio total de la aplicación, mientras que el 30% restante es para Google. El intercambio económico se realiza a través de cuentas Google Checkout. Para poder publicar aplicaciones en Android Market es necesario registrarse y pagar 25 dólares (18€ aproximadamente). Este importe solo se paga una vez, independientemente del número de aplicaciones.

Las tiendas de aplicaciones para los teléfonos móviles están cada vez más en alza. Además de Android Market, podemos destacar App Store de para iPhone y otras para Symbiam, Blackberry, Windows Phone o Palm. Hoy en día, App Store es la tienda que dispone de un mayor número de aplicaciones. La política de admisión de aplicaciones aplicada por Apple en esta tienda es más restrictiva que en Android Market. Apple ha de dar el visto bueno a cada aplicación, además se exige una cuota mayor para poder publicar, que tendrá que pagarse anualmente. A continuación se muestra una comparativa entre ambas tiendas:

---

[4] *http://market.android.com*

| | Play Store | App Store |
|---|---|---|
| Número de aplicaciones | 250.000 | 465.000 (iPhone+iPad) |
| Países con disponibilidad | 48 | 122 |
| Visto bueno a las aplicaciones | No | Si |
| Cuota a desarrolladores | 25$ una vez | 99$ al año |
| Reparto de beneficios | 30% para Google | 30% para Apple |
| Política de devolución | se pueden devolver antes de 15 minutos | solo si demostramos problemas |
| Reembolso en devoluciones | Google devuelve su parte | El desarrollador desembolsa la parte de Apple |

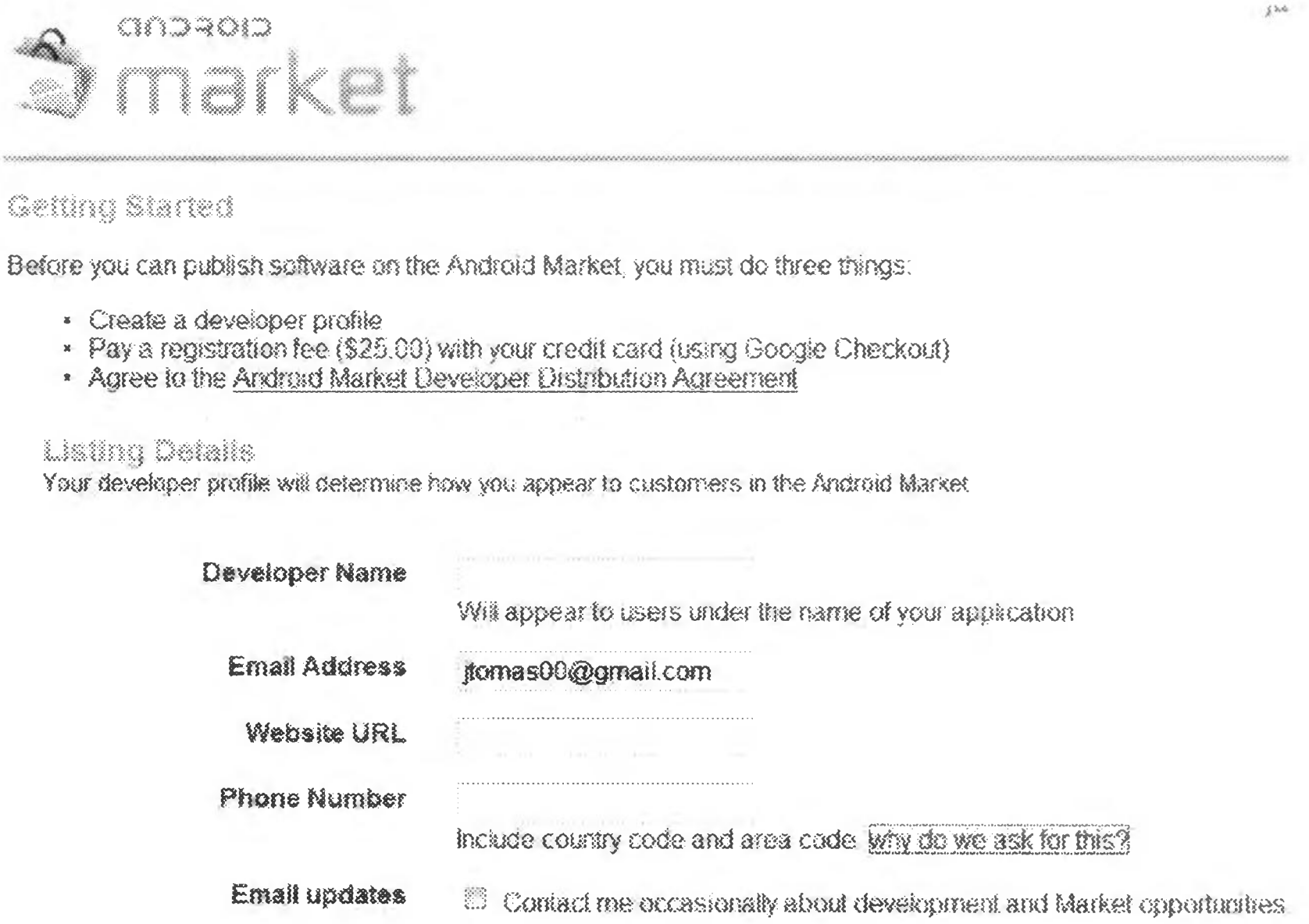
***Tabla 1:*** *Comparativa Play Store y App Store.*
*Fuente: Estudio DISTIMO, junio 2011*

Las ventajas de utilizar Play Store frente a publicarlas directamente en Internet son muchas:

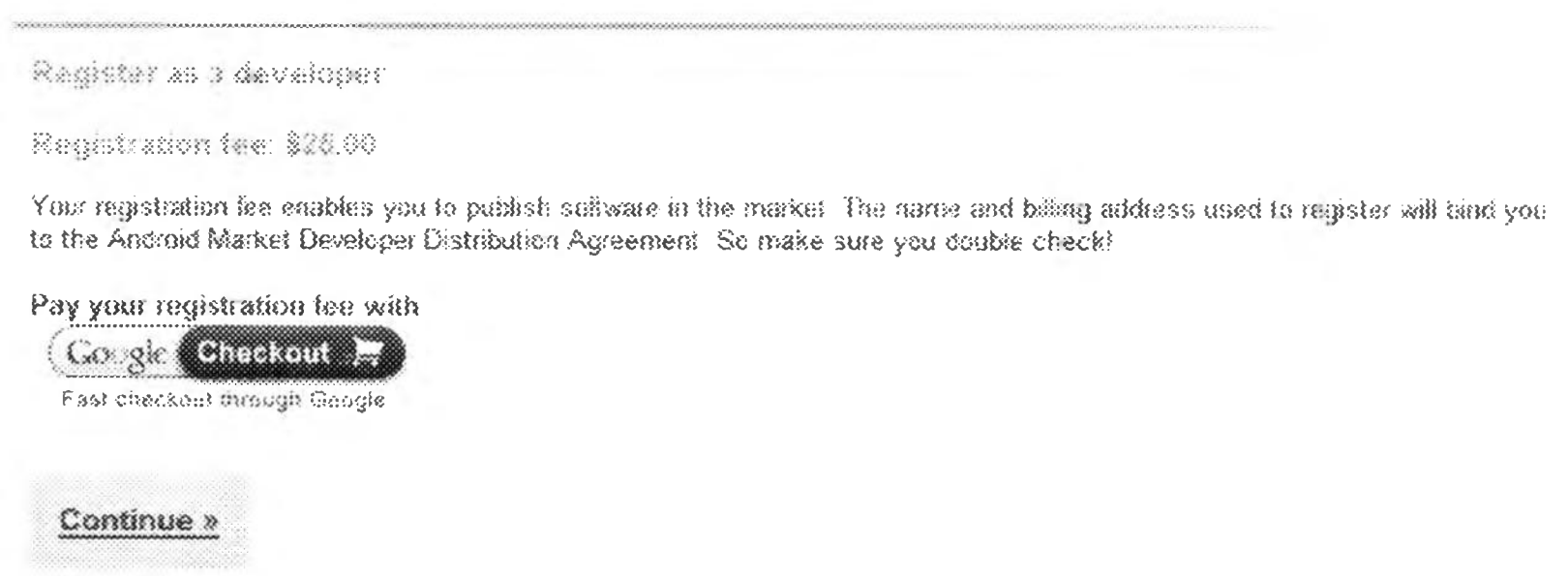
- El número de posibles usuarios es muy grande,
- Podrás ver cuánta gente tiene instalada la aplicación, cuánta se la han descargado o cómo la han valorado.
- Podrás cobrar de tus aplicaciones sin preocuparte por la gestión.
- Podrás actualizar a una nueva versión de forma automática.

Veamos los pasos a seguir para abrir una cuenta de *Play Store* para así poder publicar tus aplicaciones, una vez publicada la aplicación, automáticamente será visible en el Play Store.

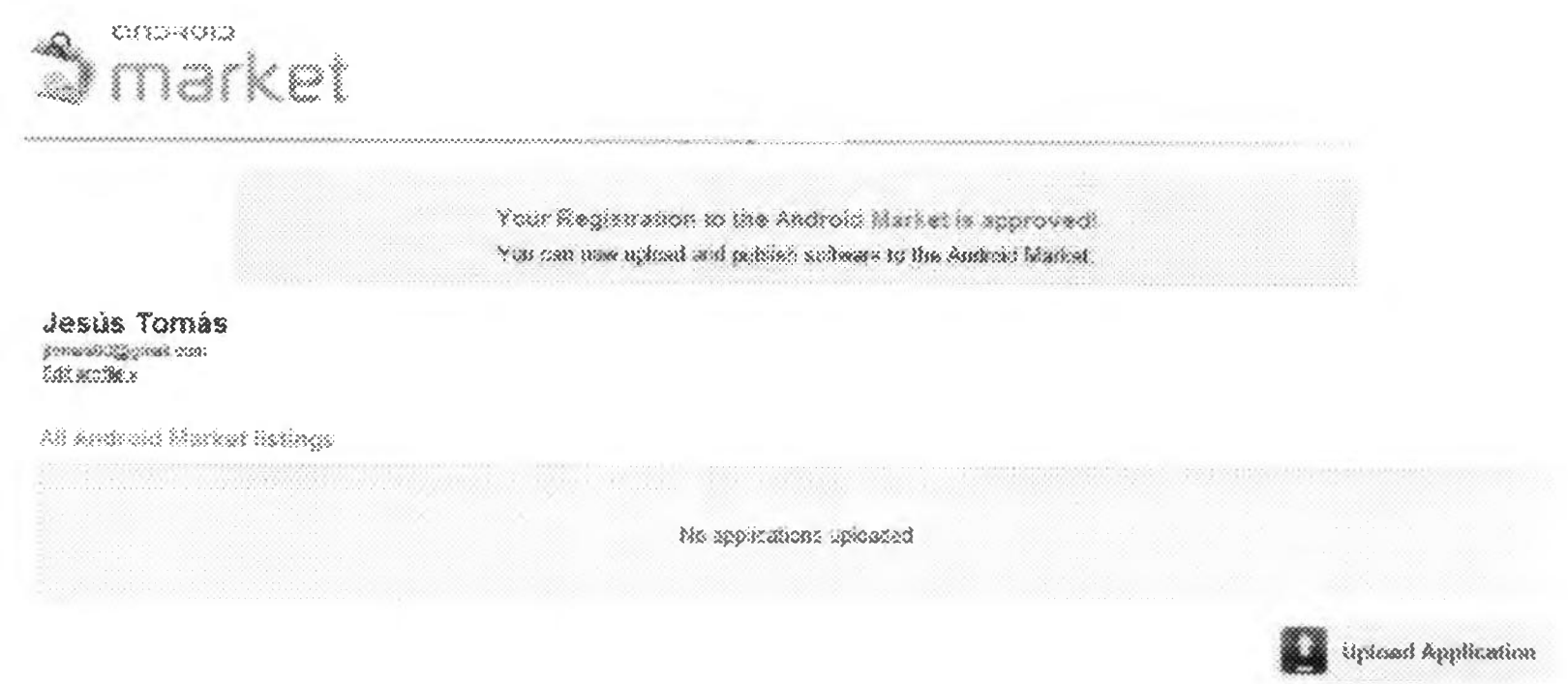
Para abrir una cuenta entra en https://play.google.com/apps/publish/ y selecciona la opción adecuada. Has de disponer de una cuenta Google Gmail, sino la tienes, crea una nueva. En primer lugar, te solicitarán una serie de datos que identifican al desarrollador:



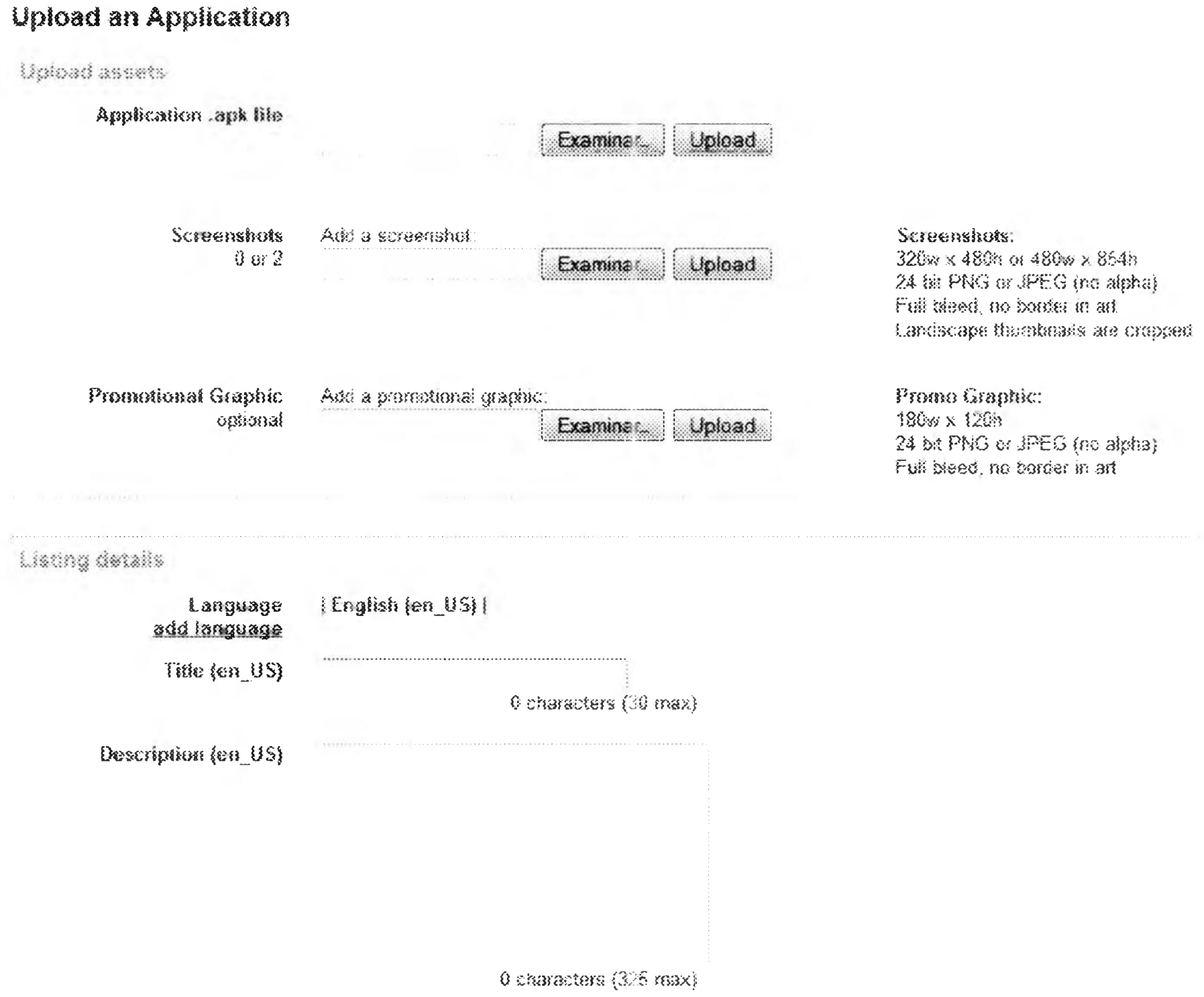

A continuación tendrás que pagar 25$ a través de Google Checkout:




Luego entrarás en el gestor de aplicaciones publicadas. Inicialmente, el listado está vacío:

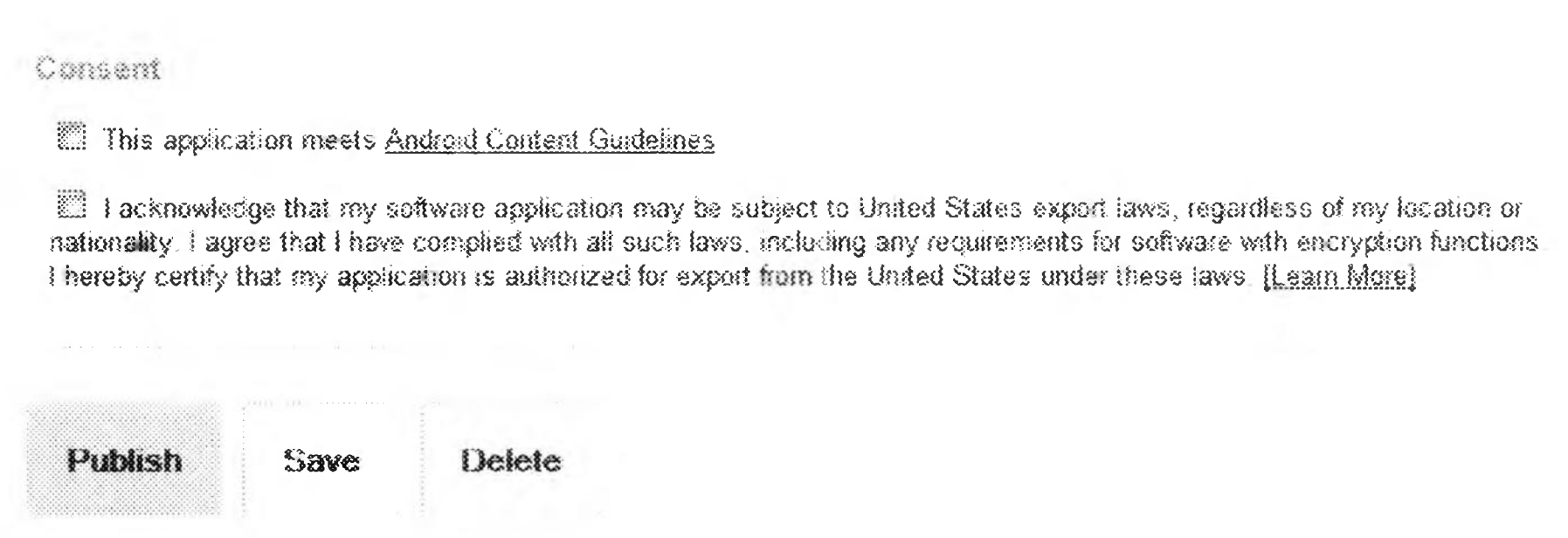


Si pulsas el botón *Update Application* te solicitarán la siguiente información:




Tras subir el .apk con tu aplicación podrás adjuntar dos capturas de pantalla y un gráfico promocional. A continuación indica los idiomas soportados e introduce el título y una breve descripción.

Indica un texto de promoción y el tipo de aplicación (juego o aplicación) y subtipo. Por defecto, la aplicación será gratuita, de no ser así puedes indicarle el precio y la cuenta Google Checkout para realizar los ingresos. En el siguiente bloque puedes indicar que se incluya un mecanismo de protección anticopia, para impedir que, una vez instalada tu aplicación en un dispositivo, su usuario pueda copiar la aplicación a otro dispositivo. Luego puedes indicar el área geográfica donde será distribuida tu aplicación. En el último bloque introduce la información de contacto.



Para terminar, indica que no infringes ninguna norma y pulsa el botón *Publish*. Como puedes ver, publicar una aplicación en Play Store resulta rápido y sencillo.

## 11.4. Asteroides: detectar victoria y derrota

Para concluir la aplicación Asteroides, va a ser interesante detectar las condiciones en que el jugador ha ganado o en las que ha perdido, para así poder detener el juego y mostrar el correspondiente mensaje al usuario.

En primer lugar declararemos las siguientes variables en la clase `VistaJuego`:

```
//// ESTADOS DE JUEGO /////
public static final int ESTADO_JUGANDO = 0;
public static final int ESTADO_VICTORIA = 1;
public static final int ESTADO_DERROTA = 2;

private int estado;
private View vistaVictoria;
private View vistaDerrota;
```

Añade los siguientes métodos a la clase:

```
public void setVistaDerrota(View vista){
vistaDerrota = vista;
}

public void setVistaVictoria(View vista){
vistaVictoria = vista;
}
```

Añade al método `onDraw()` el código:

```
if (estado == ESTADO_VICTORIA){
    vistaDerrota.setVisibility(VISIBLE);
}
```

Remplaza `res/Layout/juego.xml` por el siguiente código:

```
<FrameLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
android:layout_width="fill_parent"
android:layout_height="fill_parent">
<org.example.asteroides.VistaJuego
    android:id="@+id/VistaJuego"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="@drawable/fondo" />
<TextView
   android:id="@+id/Victoria"
```

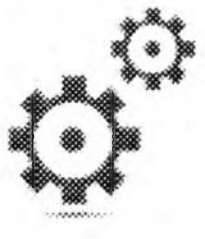
```
            android:text="¡Has ganado!"
            android:layout_width="fill_parent"
            android:layout_height="fill_parent"
            android:gravity="center"
            android:textSize="35sp "
            android:textColor="#FF0000"
            android:background="#7F000000"
            android:visibility="invisible"/>
</FrameLayout>
```

Declara el siguiente atributo en la clase `Juego`:

```
VistaJuego vistaJuego;
```

Escribe las siguientes líneas en el método `onCreate()` de esta clase:

```
vistaJuego = (VistaJuego)findViewById(R.id.VistaJuego);
vistaJuego.setVistaVictoria(findViewById(R.id.Victoria));
```



**Práctica:** *Detectar victoria y derrota en Asteroides.*

1. Escribe dentro del método `actualizaFisica()` una condición que haga pasar la variable `estado` a *ESTADO_VICTORIA*. Puedes utilizar el método `nave.verificaColision()`.
2. Escribe dentro del método `destruyeAsteroide()` una condición que haga pasar la variable `estado` a *ESTADO_DERROTA*. Completa el programa para que se visualice correctamente que hemos pasado a este estado.
3. Una vez que hemos perdido, el programa nos permite seguir jugando. ¿Podrías evitarlo?

# ANEXO A.
# Referencia Java

## *Variables*

### Constantes

**`final`**

```
final <tipo> <CONSTANTE> = <valor>;
```

donde en `<valor>` se escribe: byte: `(byte)64`, short: `(short)64`, int: `64`, long: `64L`, float: `64.0f`, double: `64.0`, char: `'@'` ó `'\u0040'`, boolean: `true/false` objetos: **`null`**, String: `"64"`, vectores: `{<valor>, <valor>, …}`

*Ejemplo:* **`final int`** `MAX_ELEM = 20;`

### Tipos simple o primitivos

```
<tipo_simple> <variable> [= <valor>];
```

Ejemplo: **`int`** `i = 0;`

| tipo | tamaño | rango | | envolvente |
|---|---|---|---|---|
| **byte** | 8 bits | -128 | 127 | Byte |
| **short** | 16 bits | -32.768 | 32.767 | Short |
| **int** | 32 bits | -2.147.483.648 | 2.147.483.647 | Integer |
| **long** | 64 bits | $-9.223.372 \cdot 10^{12}$ | 9.223.372.036.854.775.807 | Long |
| **float** | 32 bits | $-3.4 \cdot 10^{38}$ | $3.4 \cdot 10^{38}$ (mínimo $1.4 \cdot 10^{-45}$) | Float |
| **double** | 64 bits | $-1.8 \cdot 10^{308}$ | $1.8 \cdot 10^{308}$ (mínimo $4.9 \cdot 10^{-324}$) | Double |
| **boolean** | | false | true | Boolean |
| **char** | 16 bits | Unicode 0 | Unicode $2^{16}-1$ | Character |
| **void** | 0 bits | - | - | Void |

### Tipos compuestos

**`new`**

```
<tipo_compuesto> <variable> = new <tipo_compuesto>(<param>);
```

Pueden ser: arrays o clases. Dentro de las clases existen algunas especiales: *envolventes*, `String`, *colecciones* y *enumerados*

Son referencias (punteros)

### Arrays

```
<tipo><array>[]..[] = new <tipo>[<num>]..[<num>];
```

El primer elemento es el 0, al crearlos (new) hay que saber su tamaño

```
float v[] = new float[10]; //Una dimensión y 10 elementos
float M[][]= new float[3][4]; //Dos dimensiones
String s[] = {"hola", "adios"}; // elementos inicializados
for (int i = 0; i < M.length; i++)
  for (int j = 0; j < M[i].length; j++)
    M[i][j] = 0;
```

**Envolventes (*wrappers*)**

Para cada uno de los tipos simples existe una clase equivalente, con constantes y métodos que nos pueden ser útiles. Ver tabla en *variable simple*. Ver *conversión de tipos*.

**Cadena caracteres `String`**

```
String <nombre_variable> [= "<cadena de caracteres>"];
```

*Ejemplo:* `String s = "Hola";` ó `String s = new String("Hola");`

Métodos de la clase `String`:

```
.equals(String s2) //compara dos Strings
.clone() //crea una copia de un String
.charAt(int pos) //retorna el carácter en una posición
.concat(String s2) //concatena con otro Strings
.indexOf(char c, int pos) //devuelve posición de un carácter
.length() //devuelve la longitud del String
.replace(char c1, char c2) // reemplaza un carácter por otro
.substring(int pos1, int pos2) // extrae una porción
.toLowerCase() // convierte el String a minúsculas
.toUpperCase() // convierte el String a mayúsculas
.valueOf(int/float/… numero)  //convierte un número a String
```

**Colecciones**

El API de Java nos proporciona colecciones donde guardar series de datos de cualquier tipo. Dichas colecciones no forman parte del lenguaje, sino que son clases definidas en el paquete java.util.

```
<tipo_colecc><<tipo>> <colección> =
                              new <tipo_colecc><<tipo>>();
```

Hay tres tipos, cada uno con un interfaz común y diferentes implementaciones:

**Listas** – estructura secuencial, donde cada elemento tiene un índice o posición: interfaz: `List<E>` implement.: `ArrayList<E>` (acceso rápido), `LinkedList<E>` (inserciones/borrado rápidas) , `Stack<E>` (pila) , `Vector<E>` (obsoleto)

**Conjunto** – los elementos no tienen un orden y no se permiten duplicados: interfaz: `Set<E>` implement.: `HashSet<E>` (implementación usa tabla hash), `LinkedHashSet<E>`(+doble lista enlazada), `TreeSet<E>` (implem. usa árbol)

**Diccionario o Matriz asociativa** – cada elemento tiene asociado una clave que usaremos para recuperarlo (en lugar del índice de un vector): interfaz: `Map<K,V>` implement.: `HashMap<K,V>`, `TreeMap<K,V>`, `LinkedHashMap<K,V>`

Los interfaces `Iterator` y `ListIterator` facilitan recorres colecciones. La clase estática `Collections` nos ofrece herramientas para ordenar y buscar en colecciones.

```
ArrayList<Complejo> lista = new ArrayList<Complejo>();
lista.add( new Complejo(1.0, 5.0) );
```

```
lista.add( new Complejo(2.0, 4.2) );
lista.add( new Complejo(3.0, 0.0) );
for (Complejo c: lista) {
  System.out.println( c.getNombre() );
}
```

**Enumerados**

**enum** (Java 5)[1]

```
enum <nombre_enumeracion> { <CONSTANTE>, ..., < CONSTANTE> }
```

*Ejemplo:*

```
enum estacion { PRIMAVERA, VERANO, OTOÑO, INVIERNO };
        estacion a = estacion. VERANO;
```

**Ámbito**

Indica la vida de una variable, se determina por la ubicación de llaves { } donde se ha definido.

```
{
  int  a  =  10;  // sólo a  disponible
  {
    int b =  20; // a y b disponibles
  }
  //  sólo a  disponible
}
```

## *Expresiones y sentencias*

**Comentarios**

```
// Comentario de una línea
/* Comentario de
   varias líneas */
/** Comentario javadoc: para crear automáticamente la
    documentación de tu clase */
```

**Operadores**

asignación: `=`
aritméticos: `++, --, +, -, *, /, %`
comparación: `==, ¡=, <, <=, >, >=, !, &&, ||, ?:`
manejo bits: `&, |, ^, ~, <<, >>, >>>`
conversión: `(<tipo>)`

**Conversión de tipos**

Entre tipos compatibles se puede hacer asignación directa o utilizar un typecast.

```
byte b = 3;
int i = b; float f = i; //int a byte y float a int
b =(byte)i;  // hay que hacer un typecast
String s = Integer.toString(i);
b = Byte.parseByte(s);
```

[1] Solo disponible a partir de la versión 5 de Java.

**Sentencias condicional**

**if**

**else**

```
if (<condición>) {
  <instrucciones>;
} else {
  <instrucciones>;
}
```

```
if (b != 0) {
   System.out.println(
     "x= "+a/b);
} else {
   System.out.println(
              "Error");
}
```

**switch**

**case default**

```
switch (<expresión>) {
  case <valor>:
    <instrucciones>;
    [break;]
  case <valor>:
    <instrucciones>;
    [break;]
  ...
  [default:
    <instrucciones>;]
}
```

```
switch (opcion) {
  case 1:
    x = x * Math.sqrt(y);
    break;
  case 2:
  case 3:
    x = x/Math.log(y);
    break;
  default:
  System.out println(
              "Error");
}
```

**Sentencias iterativas**

**while**

```
while (<condición>) {
  <instrucciones>;
}
```

```
i = 0;
while (i < 10) {
   v[i]=0;
   i++;
}
```

**do**

```
do {
  <instrucciones>;
} while (<condición>)
```

```
i = 0;
do {
   v[i]=0;
   i++;
} while (i < 10)
```

**for**

```
for (<inicialización>;
   <comparación>; <incremento>) {
  <instrucciones>;
}
```

```
for (i = 0; i < 10; i++){
   v[i]=0;
}
```

```
for (<tipo> <variable>
                  :<colección>) {
  <instrucciones>;
}
```

```
for (Complejo c: lista){
   c.toString();
}
```

**Sentencias de salto**

**break** `break;` fuerza la terminación inmediata de un bucle ó de un switch

**continue** `continue;` fuerza una nueva iteración del bucle y salta cualquier código label:

**return** `return [<valor>];` sale de la función, puede devolver un valor

**exit** `exit([int código_retorno]);` sale del programa, puede devolver un código

## *Clases y objetos*

Clases

`class`

poli[Media][2]

Cada clase ha de estar en un fichero separado con el mismo nombre de la clase y con extensión `.class`. Por convenio los identificadores de clase se escriben en mayúscula.

```
class <Clase> [extends <Clase_padre>][implement <interfaces>]
{
  //declaración de atributos
  [visibilidad] [modificadores] <tipo> <atributo> [= valor];
  ...
  //declaración de constructor
  public <Clasee>(<argumentos>) {
    <instrucciones>;
  }
  //declaración de métodos
  [visibilidad] [modificadores] <tipo> <método>(<argumentos>)
{
    <instrucciones>;
  }
  ...
}
```

donde: `[visibilidad]` = public, protected o private
`[modificadores]` = final, static y abstract

*Ejemplo:*

```
class Complejo {
  private double re, im;
  public Complejo(double re, double im) {
   this.re = re;  this.im = im;
  }
  public String toString() {
   return(new String(re + "+" + im + "i"));
  }
  public void suma(Complejo v) {
   re = re + v.re;
   im = im + v.im;
  }
}
```

*Uso de objetos:*

```
Complejo z, w;
          z = new Complejo(-1.5, 3.0);
```

[2] Puedes encontrar un video explicativo en www.androidcurso.com

```
w = new Complejo(-1.2, 2.4);
z.suma(w);
System.out.println("Complejo: " + z.toString());
```

**Sobrecarga**

Podemos escribir dos métodos con el mismo nombre si cambian sus parámetros.

```
public <tipo> <método>(<parámetros>) {
    <instrucciones>;
}
public <tipo> <método>(<otros parámetros>) {
    <otras instrucciones>;
}
```

*Ejemplo:*

```
public Complejo sumar(Complejo c) {
  return new Complejo(re + c.re, im + c.im);
}
public Complejo sumar(double r, double i) {
  return new Complejo(re + r, im + i);
}
```

**Herencia:**

`extends`
`@Override`
`super.`
poli[Media]

```
class <Clase_hija> extends <Clase_padre> {
  ...
}
```

La clase hija va a heredar los atributos y métodos de la clase padre. Un objeto de la clase hija también lo es de la clase padre y de todos sus antecesores. La clase hija puede volver a definir los métodos de la clase padre, en tal caso es recomendable (no obligatorio) indicarlo con `@Override`; de esta forma evitamos errores habituales cuando cambiamos algún carácter o parámetro. Si un método ha sido sobrescrito podemos acceder al de la clase padre con el siguiente prefijo `super.<método>(<parámetros>)`. Ver ejemplo del siguiente apartado.

**Constructor**

`super()`

Método que se ejecuta automáticamente cuando se instancia un objeto de una clase. Ha de tener el mismo nombre que la clase. Cuando se crea un objeto todos sus atributos se inicializan en memoria a cero y las referencias serán `null`. Una clase puede tener más de un constructor (ver sobrecarga). Un constructor suele comenzar llamando al constructor de la clase padre, para ello escribiremos como primera línea de código: `super(<parámetros>);`

```
class ComplejoAmpliado extends Complejo {
  private Boolean esReal;
  public ComplejoAmpliado(double re, double im) {
    super(re, im);
    esReal = im ==0;
  }
  public ComplejoAmpliado(double re) {
```

```
    super(re, 0);
    esReal = true;
  }
  @Override
  public Complejo sumar(double re, double im) {
    esReal = im == -this.im;
    return super.sumar(re,im);
  }
  public boolean esReal(){
    return esReal;
  }
}
```

**Visibilidad**

`public`
`private`
`protected`

poli|Media|

La visibilidad indica quién puede acceder a un atributo o métodos. Se define antecediendo una de las palabras. (por defecto `public`)

`public`: accesibles por cualquier clase.
`private`: sólo son accesibles por la clase actual.
`protected`: sólo por la clase que los ha declarado y por sus descendientes.
`<si no indicamos nada>` sólo son accesibles por clases de nuestro paquete.

**Modificadores**

`final`
`abstract`
`static`

`final`: Se utiliza para declarar una constante (delante de un atributo), un método que no se podrá redefinir (delante de un método), o una clase de la que ya no se podrá heredar (delante de una clase).

`abstract`: Denota un método del cual no se escribirá código. Las clases con métodos abstractos no se pueden instanciar. Las clases descendientes deberán escribir el código de sus métodos abstractos.

`static`: Se aplica a los atributos y métodos de una clase que pueden utilizarse sin crear un objeto que instancie dicha clase. El valor de un atributo estático, además, es compartido por todos los objetos de dicha clase.

**Comparación y asignación de objetos**

`equals` y `==`

`clone` y `=`

Podemos comparar valores de variables con el operador ==, y asignar un valor a una variable con el operador =. En cambio, el nombre de un objeto de una clase no contiene los valores de los atributos, sino la posición de memoria donde residen dichos valores de los atributos (referencia indirecta). Utilizaremos el operador == para saber si dos objetos ocupan la misma posición de memoria (son el mismo objeto), mientras que utilizaremos el método `equals(<Obeto>)` para saber si sus atributos tienen los mismos valores. Utilizaremos el operador = para asignar a un objeto otro objeto que ya existe (serán el mismo objeto) y `clone()` para crear una copia idéntica en un nuevo objeto.

**Polimorfismo**

`instanceof`

poli|Media|

Se trata de declarar un objeto de una clase, pero instanciarlo como un descendiente de dicha clase (lo contrario no es posible):

```
<Clase_padre> <objeto> = new <Clase_hija>(<parámetros>);
```

Podemos preguntar si un objeto es de una determinada clase con:

```
<objeto> instanceof <Clase>
```

Podemos hacer un *tipecast* a un objeto para considerarlo de otra clase:

```
(<Clase>) <objeto>
```

*Ejemplo:*

```
Complejo c = new ComplejoAmpliado(12.4);
if (c instanceof Complejo)… //true
if (c instanceof ComplejoAmpliado)… //true
if (((ComplejoAmpliado)c).esReal())…
```

**Recolector de basura**

`finalize()`

Cuando termina el ámbito de un objeto (ver sección Ámbito) y no existen más referencias a él, el sistema lo elimina automáticamente.

```
{
  Complejo a;  // sólo a disponible, pero no inicializado
  {
    Complejo b = new Complejo(1.5,1.0); // Se crea un
objeto
    a = b;  // Dos referencias a un mismo objeto
  }
  //  sólo a  disponible
} //  el objeto es destruido liberando su memoria
```

Para eliminar un objeto el sistema llama a su método `finalize()`. Podemos rescribir este método en nuestras clases:

```
@Override protected void finalize() throws Throwable {
  try {
    close();        // cerramos fichero
  } finally {
    super.finalize();
  }
}
```

**Métodos con argumentos variables en número**

(Java 5)

```
[visibilidad] [modificadores] <tipo> <método>(<Clase>...
args) {
  <instrucciones>;
}
```

*Ejemplo:*

```
public void sumar(Complejo... args) {
  for (int i = 0; i < args.length; i++) {
    re = re + args[i].re;
    im = im + args[i].im;
  }
}
```

**Interfaces**

`interface`

Clase completamente abstracta. No tiene atributos y ninguno de sus métodos tiene código. (En Java no existe la herencia múltiple, pero una clase puede implementar uno o más interfaces, adquiriendo sus tipos).

```
interface <interface> [extends <interface_padre>] {
  [visibilidad] [modificadores] <tipo>
```

```
<metodo1>(<argumentos>);
  [visibilidad] [modificadores] <tipo>
<metodo2>(<argumentos>);
  ...
}
```

Para heredar de un interface:

```
class <Nombre_clase> extends <clase_padre> implements
                        <interface1>, <interface2>, ... {
  ...
}
```

## *Otros*

**Paquetes**

`package`
`import`

Los paquetes son una forma de organizar grupos de clases. Resuelven el problema del conflicto entre los nombres de las clases. Por ejemplo, la clase `Font` se ha creado en cientos de paquetes Java. Para referirnos a una de ellas es obligatorio indicar el paquete al que pertenece. Por ejemplo `java.awt.Font`.

Al inicio de. fichero de una clase se debe indicar su paquete:

```
package carpeta.subcarpeta...;
```

Para usar una clase de otro paquete se indica su paquete:

```
java.awt.Font fuente=new java.awt.Font(...);
```

Para abreviar, tambien podemos importar la clase de un paquete:

```
import java.awt.Font;
```

y utilizar solo el nombre de la clase:

```
Font fuente=new Font(...);
```

Para importar todas las clases de un paquete:

```
import java.awt.*;
```

**Excepciones**

`try`
`catch`
`finally`

polilMedial

```
try {
   código donde se pueden producir excepciones
}
catch (TipoExcepcion1 nombreExcepcion) {
   código a ejecutar si se produce una excepción TipoExcepcion1
}
catch (Excepcion nombreExcepcion) {
   código a ejecutar si se produce una excepción de cualquier tipo
}
finally {
   código a ejecutar tanto si se produce una excepción como si no
}
```

*Ejemplo:*

```
String salario; BufferedReader ficherol;
try {
  ficherol = new BufferedReader(new FileReader("\file"));
  salario = ficherol.readLine();
  salario = (new Integer(Integer.parseInt(salario)*10)
                                          .toString());
}
catch (IOException e) {
  System.err.println(e);
}
catch (NumberFormatException e) {
  System.err.println("No es un número");
}
catch (Exception e) {
  System.err.println("Excepción de cualquier tipo");
}
finally {
  ficherol.close();
}
```

Hilos de ejecución

`Thread`

Creación de un nuevo hilo que llama una vez al método `hazTrabajo()`:

```
class MiHilo extends Thread {
  @Override public void run() {
    hazTrabajo();
  }
}
```

Para ejecutarlo:

```
MiHilo hilo = new MiHilo ();
hilo.start();
```

Creación de un nuevo hilo que llama continuamente al método `hazTrabajo()` y que puede ser pausado y detenido:

```
class MiHilo extends Thread {
  private boolean pausa,corriendo;
  public synchronized void pausar() {
    pausa = true;
  }
  public synchronized void reanudar() {
    pausa = false;
    notify();
  }
  public void detener() {
    corriendo = false;
    if (pausa) reanudar();
  }
  @Override public void run() {
```

```
corriendo = true;
while (corriendo) {
  hazTrabajo();
  synchronized (this) {
    while (pausa) {
      try {
        wait();
      } catch (Exception e) {}
    }
  }
}
```

**Secciones críticas**

`synchronized`

Cada vez que un hilo de ejecución va a entrar en un método o bloque de instrucciones marcado con `synchronized` se comprueba si ya hay otro hilo dentro de la sección crítica de este objeto (formada por todos los bloques de instrucciones marcados con `synchronized`). Si ya hay otro hilo dentro, entonces el hilo actual es suspendido y ha de esperar hasta que la sección crítica quede libre. Para que un método pertenezca a la sección critica de objeto escribe:

```
public synchronized void metodo() {...}
```

o, para que un bloque pertenezca a la sección critica de objeto:

```
synchronized (this) {...}
```

Recuerda: La sección crítica se define a nivel de objeto no de clase. Solo se define una sección crítica por objeto.

Para conseguir una sección critica por clase escribe:

```
public static synchronized void metodo() {...}
```

o

```
synchronized (MiClase.class) {...}
```

**Inicio de la aplicación**

`main`

Una aplicación *Java* tiene que tener al menos una clase con un tipo de método denominado `main`. Será el primer método es ser ejecutado:

```
class <Clase> {
  public static void main(String[] main) {
    <instrucciones>;
  }
}
```

# ANEXO B.

# Referencia: la clase View y sus descendientes

| | |
|---|---|
| `View`<br>  | **Clase base de la jerarquía.**<br>*Posición de la vista dentro del Layout:*<br>`layout_width`, `layout_height` Permite ajustar el ancho y alto de la vista. Se puede indicar una dimensión concreta, por ejemplo 200 px, aunque lo habitual es utilizar uno de los valores:<br>`wrap_content` Ajusta el tamaño a las dimensiones necesarias para representar el contenido.<br>`fill_parent` Ajusta el tamaño al máximo posible según el *Layout* padre que la contiene. Ha sido renombrado `match_parent` a partir del nivel de API 8, aunque podemos utilizar también el nombre anterior.<br>`layout_margin`, `layout_margin_botton`, `layout_margin_left`, `layout_margin_right`, `layout_margin_top` Establece un margen exterior a la vista.<br>`layout_gravity` Centra o justifica la vista dentro del *Layout*.<br>`layout_weight` Cuando estamos en un `LinealLayout` y se dispone de espacio libre sin utilizar, podemos repartirlo entre las vistas del Layout de forma que este se reparte proporcionalmente al valor indicado en este parámetro.<br>*Definen el comportamiento:*<br>`id` Define el identificador que nos permitirá acceder a la vista. Para crear nuevos |

identificadores utilizar la expresión "`@+id/nombre_identificador`". El carácter @ significa que se trata de un identificador de recurso (es decir se definirá en el fichero *R.java*). El carácter + significa que el recurso ha de ser creado en este momento. También existen ciertos identificadores que ya han sido definidos en el sistema. Por ejemplo, utilizaremos "`@android:id/list`" para crear un `ListView`.

`tag` Permite almacenar un String que podrá ser utilizado para cualquier fin. Es decir, una información extra que el programador podrá usar para fines específicos.

`content_description` Cadena de caracteres que describe el contenido de la vista.

`clickable` Indica si la vista reacciona ante eventos de tipo onClick (se pulsa sobre la vista).

`on_click` Nombre del método que será invocado cuando ocurra un evento onClick (a partir de la versión 1.6).

`long_clickable` Indica si la vista reacciona a eventos de tipo pulsación larga (más de un segundo).

`focusable` Indica si la vista puede tomar el foco.

`focusable_in_touch_mode` Establece que cuando el dispositivo tenga capacidades de pantalla táctil y se pulse sobre la vista, ésta tomará el foco. Hay que diferenciarlo de `clickable`. Por ejemplo, nos suele interesar que un botón pueda recibir evento *onClick* pero no que coja el foco.

`next_focus_down`, `next_focus_left`, `next_focus_right`, `next_focus_up` Permite especificar el movimiento del foco cuando usamos las cuatro teclas de cursor. En la mayoría de los casos no hace falta indicarlo, ya que se ajustará automáticamente según la posición de las vistas.

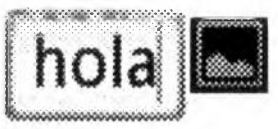
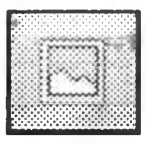
*Aspectos visuales:*

`visibility` Permite hacer invisible una vista:

- `visible` La vista es visible.
- `invisible` La vista es invisible pero ocupa lugar.
- `gone` La vista es invisible pero no ocupa lugar.

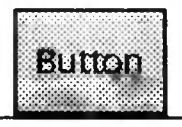
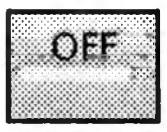
`background` Permite establecer una imagen de fondo.

`style` Permite aplicar un estilo a la vista. Ver apartado estilos y temas.

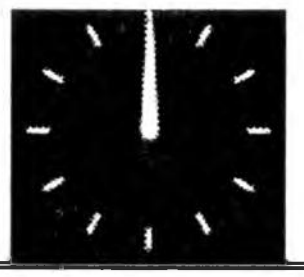
| | |
|---|---|
| | `min_width, min_height` Ancho y alto mínimo de la vista.<br>`padding, paddingBottom, paddingTop, paddingLeft, paddingRigh` Establece un margen interior en la vista. Tiene sentido en vistas como *Button* para establecer un margen entre el texto y el borde del botón. Por el contrario, `layout_margin` establece la separación entre el borde del botón y otras vistas. |
| `ImageView`[1]   | **Muestra una imagen arbitraria, como un icono. Puede cargar imágenes de varias fuentes (como los recursos o los proveedores de contenido).**<br>`adjustViewBounds` Ajustar sus límites para preservar la relación de aspecto.<br>`baseline` Donde se sitúa la línea base. Por ejemplo, en un texto la línea base suele coincidir con la base del texto.<br>`baselineAlignBottom` Posiciona nuestra línea base con la línea base de la vista indicada. Ver ejemplo a la izquierda.<br>`cropToPadding` La imagen será recortada para que quepa en padding.<br>`MaxHeight` Proporcionar una altura máxima de este punto de vista.<br>`MaxWidth` Proporcionar una anchura máxima de este punto de vista.<br>`scaleType` Controla como la imagen debe ser redimensionado o movido para que coincida con el tamaño de este ImageView.<br>`src` Origen de la imagen. |
| `ImageButton`[2]  | **Representa un botón normal pero con una imagen en vez de texto.** |
| `TextView`  | **Muestra un texto y opcionalmente permite su edición.**<br>`text` Texto que se mostrará.<br>`text_size` Tamaño del texto.<br>`text_style` Estilo del texto (negrita o itálica).<br>`typeface` Tipo de fuente usada en el texto.<br>`gravity` Cómo el texto es alineado dentro de la vista. |

[1] *Los descendientes directos de* `View` *los marcaremos con una línea doble.*

[2] Para indicar que una clase es descendiente de otra se usa la tabulación.

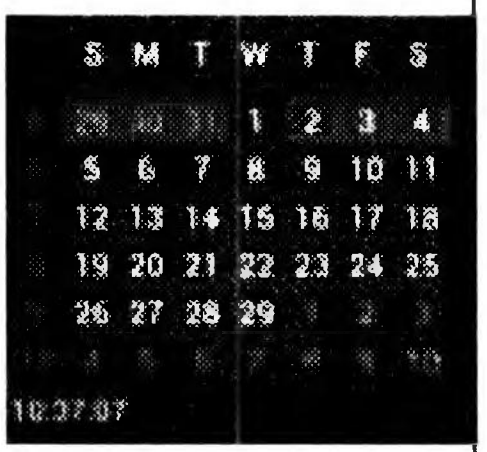
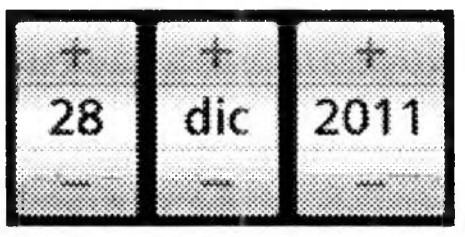
| | |
|---|---|
| | `text_appearance` Permite definir conjuntamente el tipo de fuente, tamaño del texto, color,...<br>`text_color` Color del texto.<br>`text_color_link` Color del texto para hipervínculos.<br>`text_color_highlight` Color del texto cuando es seleccionado.<br>`text_color_hint` Color del texto de indicación (ver `hint`).<br>`text_scale_x` Deforma el texto con un factor de escala horizontal.<br>`width, height` Hace que el texto tenga exactamente el ancho o alto especificado.<br>`hint` Texto que se mostrará, normalmente dentro de un `EditText`, aunque en otro color para indicar algún tipo de instrucciones. Por ejemplo "Introduzca aquí su nombre". |
| `Button`<br>  | **Representa un botón que puede ser pulsado.** |
| `CompoundButton` | **Un botón con dos estados, marcado o no marcado.**<br>`checked` Si está marcado inicialmente.<br>`button` Drawable usado para el botón gráfico. |
| `CheckBox`<br>  | **Botón tipo caja. El usuario puede marcarlo o desmarcarlo.** |
| `RadioButton`<br>  | **Botón circular. Una vez marcado el usuario no puede desmarcarlo. Suele formar parte de un `RadioGroup`.** |
| `ToggleButton`<br>  | **Botón con una "luz" que indica si está marcado o desmarcado.**<br>`disabledAlpha` Valor de alpha cuando está desmarcado.<br>`textOn` Texto del botón cuando está marcado.<br>`textOff` Texto del botón cuando está desmarcado. |
| `Switch`<br>(API 14)[3] | **Interruptor de dos estados. El usuario puede arrastrar el dedo para cambiar de estado.** |
| `EditText`<br>  | **Entrada de texto que puede ser editable.** |
| `CheckedTextView`<br>  | **Extensión de `TextView` que soporta en interface `Checkabe`. Util para permitir selecciones en `ListView`.** |

[3] Solo disponible a partir del nivel de API 14

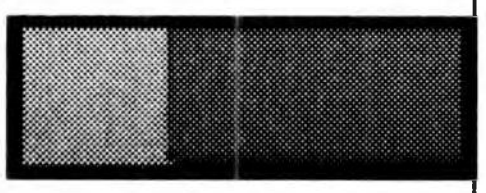
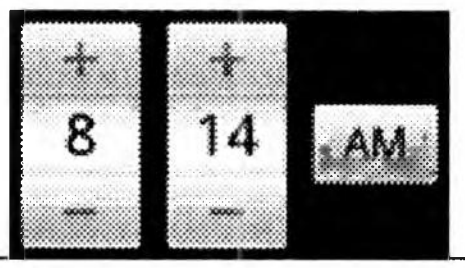
| | |
|---|---|
| `Chronometer`   | **Implementa un cronómetro simple.**<br>`format` Si está definido, mostrará el string, con el primer "%s" reemplazado por el valor actual. |
| `DigitalClock`   | **Muestra un reloj analógico.** |
| `SurfaceView`   | **Proporciona una superficie de dibujo dedicado incrustado dentro de una jerarquía de vistas. Puede controlar el formato de dicha superficie y, si se quiere, su tamaño, el SurfaceView se encarga de la colocación de la superficie en el lugar correcto en la pantalla.** |
| `GLSurfaceView` | **Es una implementación de SurfaceView que utiliza la superficie dedicada a mostrar el renderizado de OpenGL.** |
| `RSSurfaceView` | **La vista de la superficie de un renderScript gráficos (RenderScriptGL) para dibujo.** |
| `VideoView`   | **Muestra un archivo de vídeo. La clase VideoView puede cargar imágenes de varias fuentes (como los recursos o los proveedores de contenido), se encarga de calcular la medida del video para que pueda ser usado en cualquier gestor de layout, y ofrece varias opciones de visualización, como la escala y tinte.** |
| `ViewStub`   | **Es una vista de tamaño cero que se utiliza para aumentar el tiempo de ejecución de los recursos.**<br>`inflatedId` Reemplaza el ID de la vista de inflado con este valor.<br>`Layout` Un identificador para cuando el ViewStub se hace visible. |
| `AnalogClock`  | **Este widget muestra un reloj analógico con las dos manecillas horas y minutos.** |
| `ProgressBar`  | **Indicador visual del progreso de una operación. Muestra una barra que representa al usuario en qué medida la operación se ha procesado.**<br>`animationResolution` Tiempo de espera entre los marcos de la animación en milisegundos. Debe ser un valor entero, tales como " 100 ".<br>`indeterminate` Permite activar el modo indeterminado.<br>`indeterminateBehavior` Define el modo de comportarse cuando el progreso llega a máximo.<br>`indeterminateDrawable` *Drawable* que usan el modo *indeterminate*. |

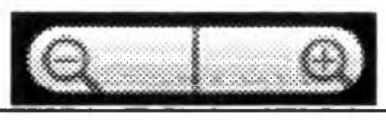
| | |
|---|---|
| | `indeterminateDuration` Duración de la animación *indeterminate*.<br>`indeterminateOnly` Limita a SÓLO modo *indeterminate*.<br>`interpolator` Método relativo.<br>`maxHeight` Un argumento opcional para proporcionar una altura máxima de la vista.<br>`maxWidth` Un argumento opcional para proporcionar una anchura máxima de la vista.<br>`progress` Define el valor por defecto el progreso, entre 0 y máximo.<br>`secondaryProgress` Define el valor de los avances secundarios entre 0 y máximo. |
| `ViewGroup` | **Puede contener otras vistas (como hijos). Es la clase base para los Layouts de los contenedores y View. Esta clase también define el `ViewGroup.LayoutParams` que sirve como clase base para los parámetros de Layouts.**<br>`addStatesFromChildren` Establece los estados *drawable* incluyendo los estados de sus hijos.<br>`alwaysDrawnWithCache` Define si el ViewGroup siempre debe llamar a sus hijos con su caché de drawable o no.<br>`animateLayoutChanges` Define si los cambios en el layouts (causada por agregar y quitar ítems) provoca que se ejecute LayoutTransition.<br>`animationCache` Define si los layouts animados deben crear de dibujo de la caché para sus hijos.<br>`clipChildren` Define si un hijo se limita a dibujar dentro de sus límites o no.<br>`clipToPadding` Define si el ViewGroup recortará su superficie de dibujo con el fin de excluir el área de *padding*.<br>`descendantFocusability` Define la relación entre el ViewGroup y sus descendientes en la búsqueda de una vista para tomar el foco.<br>`layoutAnimation` Define la animación de layout para el uso de la primera vez que se llevaron a cabo las ViewGroup.<br>`persistentDrawingCache` Esta propiedad permite conservar el caché en la memoria después de su uso inicial. |
| `AdapterView` | **Es una vista cuyos hijos están determinados por un adapter.** |
| `ListView`<br>ListView<br>Subitem | **Muestra los items de una lista de desplazamiento vertical. Los artículos provienen de la `ListAdapter` asociados con esta vista.** |

| | |
|---|---|
| `Spinner` | **Clase base abstracta para spinner widgets.**<br>`entries` Hace referencia a un recurso de matriz. |
| `Gallery`<br> | **Muestra items de una lista centrada (center-locked), que se desplaza horizontalmente.**<br>`animationDuration` Establece el tiempo de ejecución de una animación (en milisegundos), cuando el layout ha cambiado.<br>`Gravity` Especifica la forma de colocar el contenido de un objeto, tanto en la X y eje Y, dentro del propio objeto. |
| `GridView`<br>  | **Muestra los ítems de la cuadrícula de desplazamiento en dos dimensiones. Los ítems provienen de la `ListAdapter` asociados con esta vista.** |
| `AdapterViewAnimator` | **Realiza animaciones al cambiar entre sus vistas.**<br>`animateFirstView` Define si animar la vista actual, cuando utilizamos `ViewAnimation` por primera vez.<br>`Inanimación` Identificador de la animación que se utiliza cuando se muestra una vista.<br>`loopViews` Define si el animador de bucles de la primera vista, una vez que ha llegado al final de la lista.<br>`outAnimation` Identificador de la animación para utilizar cuando el fin es oculto. |
| `AbsoluteLayout`<br>  | **Especifica la ubicación exacta (coordenadas x/y) de sus hijos. Los Layouts absolutos son menos flexibles y más difícil de mantener que otros tipos de Layouts, sin posicionamiento absoluto.**<br>*Esta clase está obsoleta. Utilizar* `FrameLayout`, `RelativeLayout` *o un diseño personalizado en su lugar.* |
| `WebView`<br>  | **Utiliza el motor de renderizado WebKit para mostrar las páginas web e incluye métodos para desplazarse hacia delante y hacia atrás a través de un histórico, acercar y alejar, realizar búsquedas de texto y mucho más.** |
| `FrameLayout`<br>  | **Bloquea un área en la pantalla para mostrar en ella varias vistas (de una en una o varias a la vez).**<br>`foreground` Define el drawable para dibujar sobre el contenido.<br>`foregroundGravity` Define la gravity a aplicar al drawable del primer plano.<br>`measureAllChildren` Determina si se debe medir a todos los hijos o sólo a aquellos en el estado visible o invisible en la medición. Por defecto es false. |

| | |
|---|---|
| `AppWidgetHostView` | **Proporciona el pegamento para mostrar vistas `AppWidget`. Esta clase ofrece animación automática entre las actualizaciones, y tratará de reciclar viejos vistas para cada entrada.** |
| `CalendarView`<br>  | **Es un widget de calendario para visualizar y seleccionar las fechas. El rango de fechas con el apoyo de este calendario se puede configurar**.<br>`dateTextAppearance` El aspecto del texto de las fechas del calendario.<br>`firstDayOfWeek` El primer día de la semana de acuerdo al Calendario.<br>`focusedMonthDateColor` El color de las fechas del mes seleccionado.<br>`MaxDate` La fecha mínima muestra esta vista de calendario en formato dd/mm/aaaa.<br>`MinDate` La fecha mínima muestra esta vista de calendario en formato dd/mm/aaaa.<br>`selectedDateVerticalBar` Disponibles para la barra vertical que aparece al principio y al final de una fecha seleccionada.<br>`selectedWeekBackgroundColor` El color de fondo para la semana seleccionada.<br>`showWeekNumber` Si se muestran números de la semana.<br>`shownWeekCount` El número de semanas que se muestran.<br>`unfocusedMonthDateColor` El color de las fechas de un mes fuera de foco.<br>`weekDayTextAppearance` El aspecto del texto de la abreviatura del día de la semana en el encabezado del calendario.<br>`weekNumberColor` El color de los números de semana.<br>`weekSeparatorLineColor` El color de la línea de separación entre semanas. |
| `DatePicker`<br>  | **Es un widget para seleccionar una fecha. La fecha puede ser seleccionada por año, mes, día y los spinners.**<br>`calendarViewShown` Si muestra la vista del calendario.<br>`endYear` El año pasado (inclusive), por ejemplo, "2010".<br>`MaxDate` La fecha mínima muestra esta vista de calendario en formato dd/mm/aaaa.<br>`MinDate` La fecha mínima muestra esta vista de calendario en formato dd/mm/aaaa.<br>`spinnersShown` Si los spinners se muestran.<br>`startYear` El primer año (inclusive), por ejemplo, "1940". |

| | |
|---|---|
| GestureOverlayView<br>  | **Una capa superpuesta transparente para la entrada de un gesture que puede ser colocado sobre otros controles o contener otros widgets.**<br>`eventsInterceptionEnabled` Define si la plantilla debe interceptar los eventos de movimiento cuando la acción se reconoce.<br>`fadeDuration` Duración, en milisegundos, de los efectos que se desvanecen después de que el usuario se lleva a cabo un *gesture*.<br>`fadeEnabled` Define si el *gesture* de forma automática se apagará después de ser reconocido.<br>`fadeOffset` El tiempo, en milisegundos, que se espera antes de que el *gesture* se desvanece después de que el usuario ha terminado de dibujarlo.<br>`gestureColor` Color usado para dibujar un gesture.<br>`gestureStrokeAngleThreshold` Ángulo de curvatura mínimo debe contener antes de que sea reconocido como un *gesture*.<br>`gestureStrokeLengthThreshold` La longitud antes de que sea reconocido como un *gesture*.<br>`gestureStrokeSquarenessThreshold` Cuadratura del umbral antes de que sea reconocido como un *gesture*.<br>`gestureStrokeType` Define el tipo de trazos que definen un *gesture*.<br>`gestureStrokeWidth` Ancho del trazo utilizado para dibujar el *gesture*.<br>`orientation` Indica si los movimientos horizontales (cuando la orientación es vertical) o vertical (cuando la orientación es horizontal) para definir automáticamente un *gesture*.<br>`uncertainGestureColor` Color que se utiliza para dibujar trazos del usuario hasta que estemos seguros que es un *gesture*. |
| HorizontalScrollView<br>  | **Layouts de un container para una jerarquía de view que se pueden desplazar por el usuario, permitiendo que sea más grande que la pantalla física.**<br>`fillViewport` Define si el ScrollView debe estirar su contenido para que ocupe la ventana. |
| MediaController<br>  | **Una vista que contiene los controles de un reproductor multimedia. Por lo general contiene los botones como "Play/Pause", "Rewind", "Fast Forward", y una barra de progreso. Se encarga de la sincronización de los controles con el estado de la MediaPlayer.** |

| | |
|---|---|
| TabHost  | **Vista de ventana con pestañas. Este objeto tiene dos elementos, un conjunto de etiquetas de ficha en el que el usuario hace clic para seleccionar una ficha específica, y un objeto `FrameLayout` que muestra el contenido de esa página.** |
| TimePicker   | **Se usa para la selección de la hora del día, ya sea en 24 horas o AM/PM.** |
| ViewAnimator   | **Realiza animaciones al cambiar entre vistas.**<br>`animateFirstView` Define si se anima la vista actual, cuando se llama a ViewAnimation por primera vez.<br>`inAnimation` Identificador de la animación para utilizar cuando se muestra una vista.<br>`outAnimation` Identificador de la animación para utilizar cuando el fin es oculto. |
| RelativeLayouts   | **Layouts donde las posiciones de los hijos pueden ser descritas en relación con los demás o para los padres.**<br>`Gravity` Especifica la forma de colocar el contenido de un objeto, tanto en la X y eje Y, dentro del propio objeto.<br>`ignoreGravity` Indica qué vista no debería verse afectada por la gravedad. |
| DialerFilter   | |
| TwoLineListItem   | **Un grupo con dos hijos, para uso en listviews. Tiene dos TextViews elementos (o subclases) con los valores ID text1 y text2.** |
| LinearLayout   | **Ordena a sus hijos en una sola columna o una fila.**<br>`baselineAligned` Cuando se establece en false, evita la alineación de los layouts.<br>`baselineAlignedChildIndex` Cuando una disposición lineal es parte de otro layout.<br>`Gravity` Especifica la forma de colocar el contenido de un objeto, tanto en la X y eje Y, dentro del propio objeto.<br>`measureWithLargestChild` Cuando se establece en true, todos los child con un peso, se considera el tamaño mínimo del más largo.<br>`Orientation` Si la disposición es una columna o una fila Usa "horizontal" de una fila, "vertical" de una columna.<br>`weightSum` Define la suma del peso. |

| | |
|---|---|
| NumberPicker<br> | **Un widget que permite al usuario elegir un número entre un rango predefinido.** |
| RadioGroup<br> | **Usado para crear un grupo de `radioButton` de los cuales solo puede ser seleccionado uno a la vez, es decir, al seleccionar uno se deseleccionará cualquier anterior.** |
| SearchView<br>  | **Proporciona una interfaz de usuario para que el usuario introduzca una consulta de búsqueda y presentar una solicitud a un proveedor de búsquedas. Muestra una lista de sugerencias de consulta o los resultados, si está disponible.**<br>`iconifiedByDefault` El estado por defecto de la `SearchView`.<br>`imeOptions` Las opciones IME para establecer en el campo de texto de la consulta.<br>`InputType` El tipo de entrada para establecer en el campo de texto de la consulta.<br>`MaxWidth` Un ancho máximo opcional de la `SearchView`.<br>`queryHint` Cadena de consulta opcional que se muestra en el campo vacío de consulta. |
| TabWidget<br>  | **Muestra una lista de las etiquetas de la ficha que representa. Cuando el usuario selecciona una pestaña, este objeto envía un mensaje al contenedor principal, TabHost, para indicarle que debe cambiar la página mostrada.**<br>`divider` Disponibles utilizado para dibujar la división entre las pestañas.<br>`tabStripEnabled` Determina si la tira en los indicadores de la ficha se dibuja o no.<br>`tabStripLeft` Se utiliza para dibujar la parte izquierda de la tira debajo de las pestañas.<br>`tabStripRight` Se utilizado para dibujar la parte derecha de la tira debajo de las pestañas. |
| TableLayout<br>  | **Ordena a sus hijos en filas y columnas.**<br>`collapseColumns` El índice de las columnas de colapso.<br>`shrinkColumns` El índice de las columnas para reducir el tamaño.<br>`stretchColumns` El índice de las columnas para estirar. |
| TableRow<br> | **Ordena a sus hijos en horizontal dentro de una tabla.** |
| ZoomControls<br>  | **Muestra un conjunto de controles que se utilizan para hacer zoom y ofrece devoluciones de llamada para registrar los eventos.** |

*Esta edición se terminó de imprimir en* **julio** *de* **2012.** *Publicada por*
**ALFAOMEGA GRUPO EDITOR, S.A. de C.V.** *Pitagoras No. 1139*
*Col. Del Valle, Benito Juárez, C.P. 03311, México, D.F.*

*La impresión y encuadernación se realizó en*
**CARGRAPHICS, S.A. de C.V.** *Calle Aztecas No.27*
*Col. Santa Cruz Acatlán, Naucalpan, Estado de México, C.P. 53150. México*

En los últimos años, los teléfonos móviles están transformando la forma en que las personas se comunican. Los nuevos terminales ofrecen unas capacidades similares a las de un ordenador personal. Un teléfono móvil siempre está en el bolsillo del usuario, esto le convierte en el nuevo ordenador personal del siglo XXI.
Android es la plataforma libre para el desarrollo de aplicaciones móviles creada por Google. En la actualidad se ha convertido en la plataforma líder frente a otras como iPhone o Windows Phone. Las aplicaciones Android están ampliando su rango de influencia a nuevos dispositivos tales como tabletas, net-books o Google TV.
Este libro pretende ser una guía para aquellos lectores que quieran introducirse en la programación en Android. Todos los capítulos son descritos por medio de sencillos ejemplos, aumentando su nivel de complejidad a medida que avanzan los capítulos. La obra está recomendada tanto para usuarios con poca experiencia, como para programadores experimentados. A lo largo del libro se desarrolla una aplicación de ejemplo, el mítico videojuego Asteroides. Comienza con una versión sencilla, que se irá completando capítulo a capítulo, para que incluya gráficos vectoriales y en mapa de bits, control mediante pantalla táctil y sensores, geolocalización, multimedia, ficheros, XML, SQL, acceso a Internet, servicios Web... Así cómo publicar en el Google Play Store.

El libro propone un aprendizaje activo, con actividades, muchas a través de Internet:

- **poli[Media]** Más de 50 videos elaborados por el autor.
- **Ejercicio paso a paso:** La mejor forma de aprender es practicando.
- **Práctica:** Para afianzar lo aprendido hay que practicar.
- **Solución:** Te será de ayuda si tienes problemas en las prácticas.
- **Recursos adicionales:** Localiza rápidamente la información clave
- **Preguntas de repaso y reflexión:** ¿Lo has entendido todo correctamente?

**Jesús Tomás** es doctor en informática, profesor titular del Departamento de Comunicaciones en la Universidad Politécnica de Valencia. Trabaja en múltiples proyectos de investigación y transferencia de tecnología relacionados con inteligencia artificial. Ha publicado gran cantidad de trabajos en revistas científicas y varios libros didácticos sobre nuevas tecnologías. Tiene una amplia experiencia impartiendo cursos de formación en empresas.

**www.alfaomega.com.mx**

ÁREA: Computación
SUBÁREA: Sistemas Operativos

ISBN 978-607-707-506-6

"Te acerca al conocimiento"

Alfaomega Grupo Editor