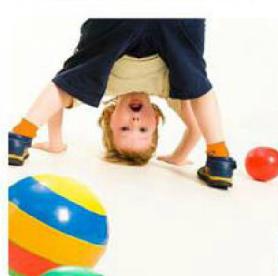


Android



Los propietarios del © tiene reservados todos los derechos. Cualquier reproducción, total o parcial de este texto, por cualquier medio, o soporte sonoro, visual o informático, así como su utilización fuera del ámbito estricto de la información del alumno comprador, sin la conformidad expresa por escrito de los propietarios de los derechos, será perseguida con todo el rigor que prevé la ley y se exigirán las responsabilidades civiles y penales, así como las reparaciones procedentes.

Android

Autor: Joaquín Fanlo González

Imprime: El depositario, con autorización expresa de SEAS, S.A.

ISBN: 978-84-938884-9-7

Depósito Legal: Z-1582-2011

ÍNDICE ASIGNATURA

UNIDAD 1. PRESENTACIÓN DE ANDROID

1.1. ¿Qué es Android?

1.1.1. Historia

1.2. Instalación del entorno

1.2.1. Requisitos del Sistema

1.2.2. Instalación de Java Development Kit

1.2.3. Instalación del SDK de Android

1.2.4. Añadir Variables Path

1.2.5. Instalación de Eclipse

1.2.6. Instalación Android Development Tools (ADT)

1.3. Mi Primer programa

1.3.1. Crear un Emulador AVD

1.3.2. Instalación de Dispositivo Android

1.3.3. Selección del Emulador

1.3.4. DDMS

1.3.5. Hola Mundo en Android

1.4. Explorando la SDK de Android

1.4.1. Iniciar el desarrollo de Android. Componentes de una aplicación

1.4.2. La Interfaz de Usuario

UNIDAD 2. AVANZADO EN ANDROID

2.1. Gráficos 2D

- 2.1.1. Conceptos básicos
- 2.1.2. Aplicando conceptos

2.2. Gráficos 3D

- 2.2.1. Introducción a OpenGL
- 2.2.2. Implementando OpenGL

2.3. Multimedia

- 2.3.1. Reproducción de audio
- 2.3.2. Reproducción de video

2.4. Bases de Datos SQLite

- 2.4.1. Aplicación con SQLite
- 2.4.2. Vinculación de datos
- 2.4.3. Utilizar un ContentProvider

UNIDAD 3. PRÓXIMAS GENERACIONES

3.1. Sistemas de Geolocalización

- 3.1.1. Geolocalizándonos
- 3.1.2. Configurar el Api Key
- 3.1.3. Mostrando Google Maps
- 3.1.4. Aplicación Localización GPS

3.2. Sensores

3.3. Bluetooth

3.4. MultiTouch

3.5. Reconocimiento de voz

UNIDAD 4. WEB MÓVI EN ACCIÓN

4.1. Web Services

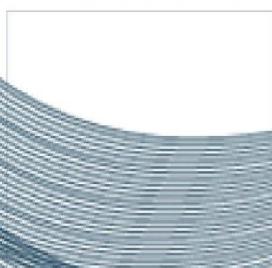
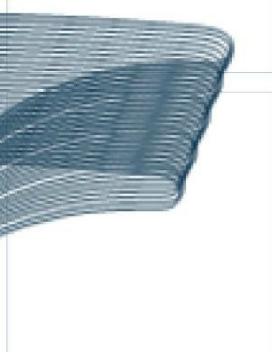
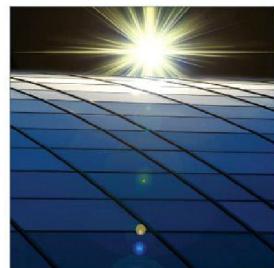
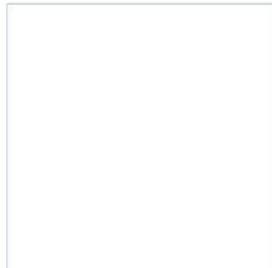
- 4.1.1. Arquitectura de los Servicios Web
- 4.1.2. Estándar de Servicios Web
- 4.1.3. REST
- 4.1.4. SOAP
- 4.1.5. Crear un Web Service en Java
- 4.1.6. Crear un Web Service en .net y c#

4.2. Cliente Android-Servidor PHP-MySql

- 4.2.1. Interfaz de usuario
- 4.2.2. Servidor PHP
- 4.2.3. Base de Datos
- 4.2.4. Código de servidor

01

ANDROID



ÍNDICE

◆ ÍNDICE	1
◆ OBJETIVOS	3
◆ INTRODUCCIÓN.....	4
1.1. ¿Qué es Android?	5
1.1.1. Historia	5
1.1.1.1. Versiones de Android.....	6
1.1.1.2. La plataforma Android.....	10
1.2. Instalación del entorno.....	11
1.2.1. Requisitos del Sistema.....	11
1.2.1.1. Sistemas operativos compatibles.....	11
1.2.2. Instalación de Java Development Kit	11
1.2.3. Instalación del SDK de Android.....	14
1.2.4. Añadir Variables Path.....	19
1.2.5. Instalación de Eclipse.....	22
1.2.6. Instalación Android Development Tools (ADT)	23
1.3. Mi Primer programa	26
1.3.1. Crear un Emulador AVD.....	26
1.3.2. Instalación de Dispositivo Android	29
1.3.3. Selección del Emulador.....	30
1.3.4. DDMS	33
1.3.5. Hola Mundo en Android.....	36
1.4. Explorando la SDK de Android	40
1.4.1. Iniciar el desarrollo de Android. Componentes de una aplicación.....	40
1.4.1.1. El archivo AndroidManifest.xml.....	40
1.4.1.2. Activity.....	42
1.4.1.3. Intent	45
1.4.1.4. Services	47
1.4.1.5. BroadcastReceiver	52
1.4.1.6. ContentProvider	55
1.4.2. La Interfaz de Usuario	59
1.4.2.1. FrameLayout.....	59
1.4.2.2. LinearLayout	61
1.4.2.3. TableLayout	63
1.4.2.4. RelativeLayout	67
◆ RESUMEN.....	71

OBJETIVOS

- Conoceremos cómo surge Android, cómo ha evolucionado y las diferentes etapas por las que ha pasado hasta su situación actual.
- Aprenderemos todas las herramientas necesarias para poder realizar un desarrollo en Android. Haremos un recorrido por todas ellas viendo todos los recursos disponibles y gratuitos que existen para ello.
- Tendremos un primer contacto con la programación Android, antes de meternos en profundidad con todos los conceptos básicos, y así conocer un poco las herramientas de desarrollo.
- Veremos todos los conceptos necesarios para poder entender cómo funciona la programación en Android, con ejemplos prácticos que el alumno podrá ir desarrollando y ejecutando tanto en un emulador como en un dispositivo real, si se dispone de él.

INTRODUCCIÓN

Esta unidad nos va a proporcionar un acercamiento al mundo de Android, donde veremos cómo surgió, los motivos de su aparición y cómo se ha impuesto en el mundo de las tecnologías.

Se comprueba el potencial que nos proporciona Android, dada la posibilidad de aprovechar todo el potencial que nos ofrecen los terminales móviles actuales, que cada vez traen mejores procesadores y mayor cantidad de memoria para realizar procesos cada vez más complicados.

En esta unidad veremos toda la sintaxis básica de Android y conoceremos los elementos más importantes para entender su arquitectura y funcionamiento.

1.1. ¿QUÉ ES ANDROID?

Android es la plataforma de software de Google y la Open Handset Alliance (Alianza Comercial que se encarga de desarrollar estándares abiertos para dispositivos móviles) de código fuente, para aplicaciones móviles, con la posibilidad de adaptarse a diferentes mercados y dispositivos móviles.

En la actualidad, Android dispone de una comunidad de desarrolladores implementando aplicaciones en los dispositivos, que hace que sea de las plataformas móviles más vendidas en el mundo. Dispone de una tienda para aplicaciones, el Android Market, con más de 200000 aplicaciones clasificadas por categorías, y con un buscador propio para facilitar su búsqueda por dichas categorías. A su favor hay que decir, que más del 65% de ellas son gratuitas, y que muchas de ellas son totalmente funcionales, conteniendo un pequeño *banner* de publicidad para ayudar al programador en el desarrollo de la misma.

Empezaremos viendo todos los hitos importantes por los que ha pasado a lo largo de su corta vida.

1.1.1. HISTORIA

Mirando al pasado, en julio de 2005, Google compra Android Inc., una pequeña empresa situada en Palo Alto, California, y a partir de entonces empiezan a correr rumores acerca de que Google planeaba construir su propio dispositivo móvil libre y gratis, enfocando las ganancias en la publicidad de las búsquedas que los usuarios realizarían desde el propio dispositivo móvil. Obviamente, esos rumores de un móvil gratis fueron falsos, pero al final Android resultó ser algo mucho más interesante y revolucionario: un sistema operativo móvil open source (software de código abierto, es decir, que ha sido desarrollado y distribuido libremente. <http://opensource.org/>) propulsado nada más y nada menos que por Google, así que repasemos la historia de Android por muy breve que sea, mirando todas las versiones por las que ha pasado.

1.1.1.1. VERSIONES DE ANDROID

El lanzamiento inicial del Android Software Development Kit (SDK) fue en noviembre de 2007, posteriormente, a mediados de agosto de 2008, se lanzó el Android 0.9 SDK en su versión beta. Un mes después, a finales de septiembre 2008, lanzaron Android 1.0 SDK (Release 1). Seis meses después, a principios de marzo 2009, Google presentó la versión 1.1 de Android para el “dev phone” o teléfono para desarrolladores y la actualización incluía algunos pequeños cambios estéticos, además de tener soporte para “búsquedas por voz”, las primeras aplicaciones de pago en el Android Market, arreglos en la alarma del reloj interno, algunas pequeñas mejoras en Gmail y otros cambios menores.

A mediados de mayo 2009, Google lanzó la versión 1.5 de Android **Cupcake** con su respectivo SDK que dispone de nuevas características como la grabación en video, soporte para dispositivos Bluetooth estéreo (A2DP - Advanced Audio Distribution Profile. Es el perfil estándar de transmisión de audio, mono o estéreo, entre dispositivos bluetooth), que añadía la posibilidad de personalizar el teclado en pantalla y daba compatibilidad con otros teclados de software, implementó el reconocimiento de voz y el AppWidget framework (entorno de desarrollo de aplicaciones con widget), que permitió a los desarrolladores crear sus propios widgets para la página principal.



Figura 1.1. Android 1.5 Cupcake.

Luego apareció Android 1.6 **Donut**, en septiembre de 2009, con mejoras en las búsquedas, compatibilidad con pantallas de alta y baja densidad, indicador de uso de batería y posibilidad de realizar VPN (Red Privada Virtual, que consiste en extender una red local sobre la red pública para realizar conexiones privadas).



Figura 1.2. Android 1.6 Donut.

Unos meses después, el Motorola Milestone fue el primer dispositivo en incluir Android 2.0 **Eclair**, que incluía varias nuevas características y aplicaciones precargadas que requerían un hardware mucho más rápido que la generación anterior de móviles con Android, compatibilidad con la funcionalidad multitáctil, teclas virtuales, gestión de cuentas centralizada (un mes más tarde salió 2.0.1, una pequeña actualización que arreglaba pequeños bugs).

ANDROID 2.0



Figura 1.3. Android 2.0 Eclair.

Poco después, el **Google Nexus One** (el cuál marcó un antes y un después, ya que Google trató de venderlo él mismo y libre, además de a través de algunas operadoras) llegó con Android 2.1 (el cual algunos llamaron “Flan”, pero Google siguió considerándolo parte de “Eclair”) con nuevas capacidades 3D, incorporación de live wallpapers (fondos de pantalla animados) y lo que significó la gran mejora de la plataforma desde 1.6 Donut.



Figura 1.4. Google Nexus One.

En el evento que Google celebró en Mayo de 2010, se presentó la versión 2.2 **“Froyo”** de Android, y una cosa queda clara en este evento: el posicionamiento de Google como competencia directa de Apple, y Android 2.2, como el nuevo referente en dispositivos móviles.

Un ejemplo de esta competencia, y como anécdota del congreso, crearon un punto de acceso WiFi, a partir de la conexión 3G (tethering), y la utilizaron para conectar a internet un iPad.



Figura 1.5. Android 2.2 Froyo.

Esta versión 2.2 (Froyo), trajo muchísimas novedades y mejoras del sistema operativo, aunque ninguna de ellas podía tener tanta importancia como la posibilidad de reproducir contenidos flash. Aparte de esta gran mejora, se implementaron otras como fueron el nuevo mercado de aplicaciones (Market), posibilidad de instalar las aplicaciones en la tarjeta de memoria, el anteriormente comentado Tethering (compartir mediante una conexión WiFi tu red 3G del móvil), Portable Hotspot, un nuevo browser (navegador) y Adobe Air.

Pero no es ni mucho menos la mejora más destacada de Froyo. En el centro del sistema operativo ahora ejecuta un compilador Just-in-time (JIT), que optimiza la ejecución de código mediante precompilación y cacheo de las instrucciones compiladas. El resultado es impresionante: un incremento brutal de la velocidad de ejecución, que ahora es de dos a cinco veces más rápida que la versión anterior de Android 2.1.

Otra de las novedades de Froyo, es que te permite instalar aplicaciones desde nuestro PC, a través del navegador web, y que se instalen vía OTA (Over The Air que es la instalación de las actualizaciones o aplicaciones a través de la red móvil o conexión WiFi del dispositivo móvil).



Figura 1.6. Vista del Android Market en el navegador web y la instalación desde él de las aplicaciones.

A día de hoy, está en la calle ya la versión 2.3 “**Gingerbread**”, con su buque insignia el **Nexus S**, para poco a poco irse instalando en los terminales afortunados que lo vayan a recibir.



Figura 1.7. Nexus S con Gingerbread.

Esta versión de Android, ha actualizado su diseño de la interfaz de usuario, ofreciendo soporte a pantallas “extra grandes”.

Soporta telefonía VoIP (Voice Over IP, es decir, transmisión de la voz a través del protocolo IP).

Se han realizado diversas mejoras multimedia, tales como decodificación de audio ACC, efectos de audio, ecualización del mismo y refuerzo de graves.

Incorpora NFC, Near Field Communication, que es una tecnología inalámbrica de corto alcance, que permite el intercambio de datos entre dispositivos, que nos permitirá, por ejemplo, realizar pagos simplemente acercando el dispositivo móvil al terminal de pago, y otras aplicaciones que incorporen esta tecnología.

Se puede consultar en la plataforma un ejemplo de implementación de esta tecnología.

Otra de las mejoras que incorpora esta versión, es el soporte para múltiples cámaras, es decir, la cámara frontal para poder realizar video llamadas. La última versión de Google Talk, la aplicación de Google de mensajería instantánea, incorpora video llamada entre los dispositivos que poseen dicha cámara y la última versión de esta aplicación.

1.1.1.2. LA PLATAFORMA ANDROID

Android es un entorno de software creado para dispositivos móviles. No es una plataforma de hardware. Incluye un sistema operativo basado en Linux, una completa interfaz de usuario, aplicaciones, bibliotecas de código, estructuras para aplicaciones, compatibilidad multimedia, etc. Aunque los componentes del SO se escriban en C o C++, las aplicaciones para Android de diseñan en Java. Tenemos que pensar que Java es también Open Source, con lo que esto conlleva, poder desarrollar todo sin costes asociados. Todos los ejemplos de este manual se han creado en Java con el SDK de Android.

A lo largo del manual, siempre que haya que probar o ejecutar un código en un dispositivo, se utiliza un emulador basado en software. En próximos capítulos se encontrará más información de cómo configurar y utilizar el emulador de Android.

1.2. INSTALACIÓN DEL ENTORNO

Para poder desarrollar aplicaciones para Android, necesitaremos tener instalados una serie de programas gratuitos. Necesitaremos tener instalado y en el orden que se indica, para que posteriormente no nos surjan problemas con el desarrollo de las aplicaciones. Primero debemos instalar el *Java Development Kit* (JDK), de la página de *Oracle-Sun Microsystem*, el último SDK de Android, y el entorno de desarrollo eclipse. Todas ellas son herramientas totalmente gratuitas y de fácil instalación.

Vamos a ver su instalación paso a paso.

1.2.1. REQUISITOS DEL SISTEMA

En esta sección se describen los requisitos del sistema y software para el desarrollo de aplicaciones para Android con el SDK de Android.

1.2.1.1. SISTEMAS OPERATIVOS COMPATIBLES

- Windows XP (32 bits), Vista (32 o 64 bits) o Windows 7 (32 o 64 bits)
- Mac OS X 10.5.8 o posterior (solo x86).
- Linux (probado en Ubuntu Linux, Lucid Lynx).
 - ✓ Biblioteca GNU C (glibc) 2.7 o posterior.
 - ✓ En Ubuntu Linux, versión 8.04 o posterior.
 - ✓ Distribuciones de 64 bits debe ser capaz de ejecutar aplicaciones de 32 bits.

1.2.2. INSTALACIÓN DE JAVA DEVELOPMENT KIT

Para descargarnos el JDK de Java debemos acceder a la siguiente dirección, disponible también en la plataforma de estudio:

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

Una vez hayamos accedido, deberemos buscar la última versión de *Java Platform, Standard Edition*, que en nuestro caso es “Java SE 6” y descargarnos el [Download JDK](#). Nos redireccionará a una página donde debemos seleccionar la plataforma o sistema operativo que vamos a utilizar y el lenguaje deseado. Una vez descargado procedemos a su instalación.

Pulsamos “Next” en la pantalla de bienvenida:

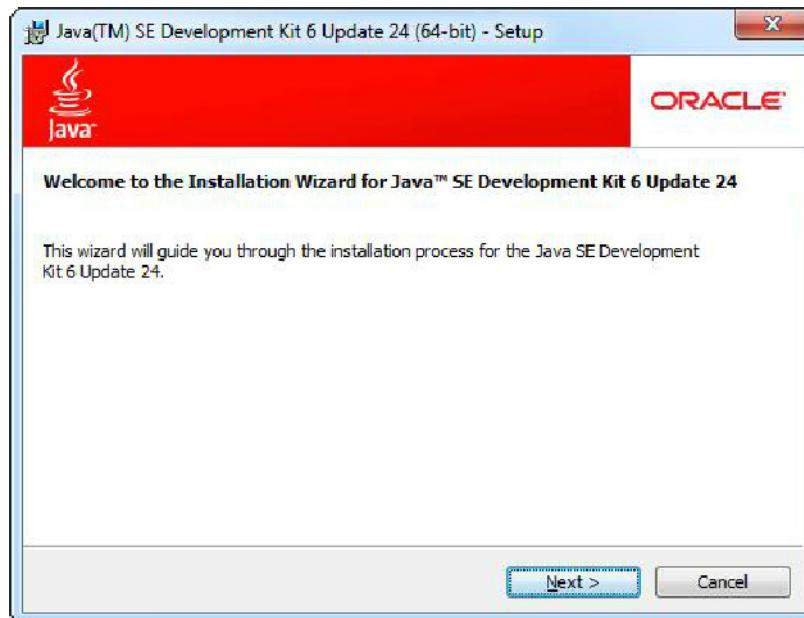


Figura 1.8. Ventana inicial instalación Java SDK.

Lo dejamos todo seleccionado para instalar en el disco duro:

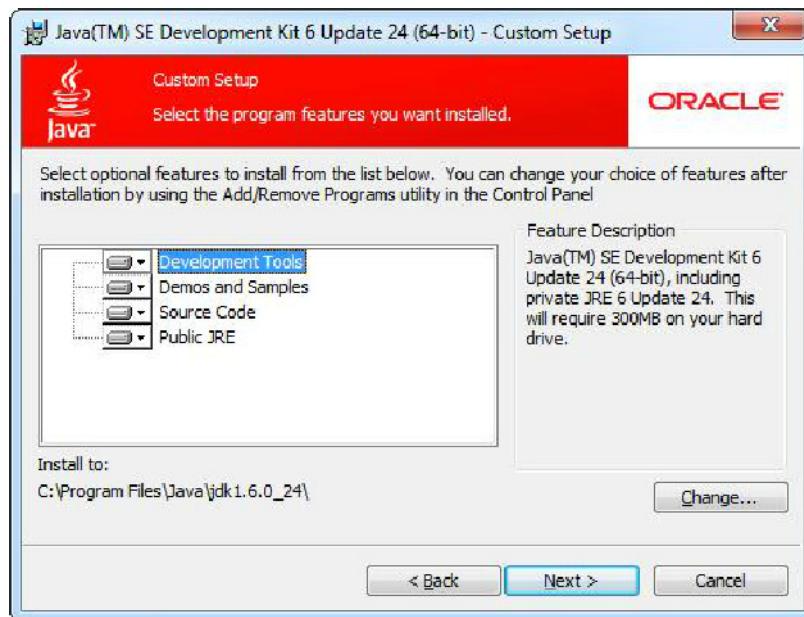


Figura 1.9. Ventana selección instalación.

Dejamos el directorio de instalación que nos muestra por defecto:

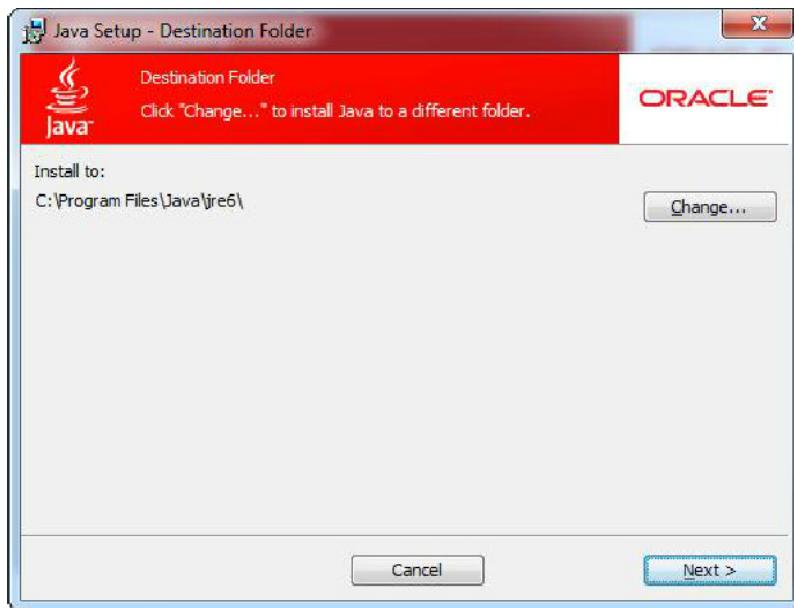


Figura 1.10. Ventana de elección de directorio de instalación.

Esperamos a que acabe de realizar la instalación.

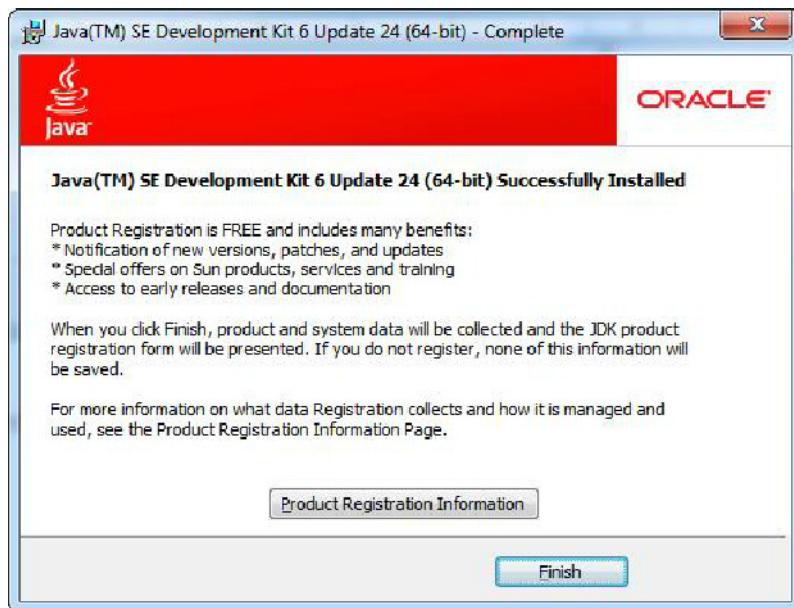


Figura 1.11. Ventana de finalización de la instalación de Java SDK.

1.2.3. INSTALACIÓN DEL SDK DE ANDROID

Para instalar el ultimo SDK debemos acceder a la página de *Android Developer* <http://developer.android.com/sdk/index.html> y descargamos el archivo “*installer_r10-windows.exe*”. (El anterior enlace está disponible en la plataforma de estudio).

Platform	Package	Size	MD5 Checksum
Windows	android-sdk_r11-windows.zip	32837554 bytes	0a2c52b8f8d97a4871ce8b3eb38e3072
	installer_r11-windows.exe (recommended)	32883649 bytes	3dc8a29ae5afed97b40910ef153caa2b
Mac OS X (intel)	android-sdk_r11-mac_x86.zip	28844968 bytes	85bed5ed25aea51f6a447a674d637d1e
Linux (i386)	android-sdk_r11-linux_x86.tgz	26904929 bytes	026c67f02627a3a70efb197ca3360d0a

Figura 1.12. Archivo que debemos descargar para su posterior instalación.

Una vez descargado procedemos a su instalación:



Figura 1.13. Ventana de inicio de instalación SDK Tools Android.

En algunas ocasiones, por motivos desconocidos, al pasar al siguiente punto, salta una pantalla de aviso comentándonos que no ha encontrado instalado el *Java SE Development Kit*:

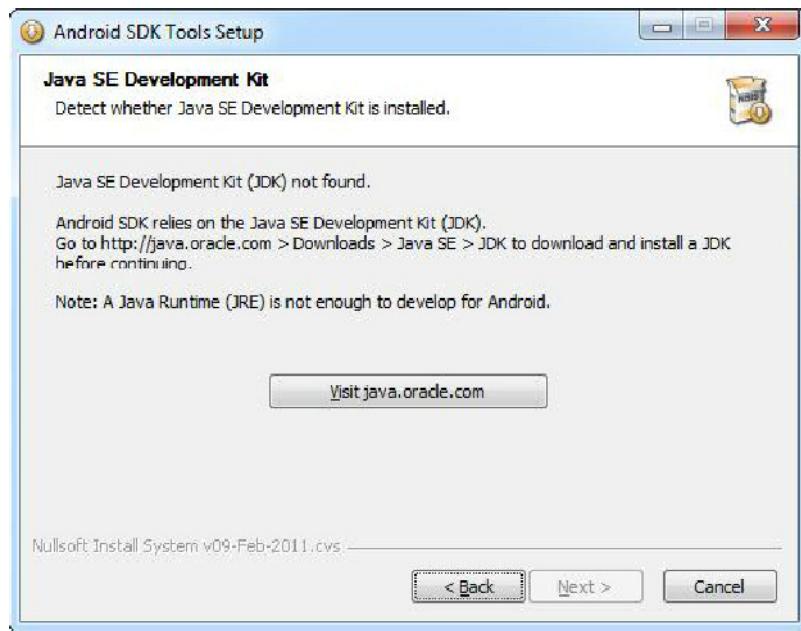


Figura 1.14. Ventana aviso fallo en la instalación.

Si ocurre esto, y realmente se ha instalado como describimos en el punto anterior, pulsamos sobre “Back” y volvemos a pulsar en “Next”. Probablemente se solucione nuestro problema y podamos continuar instalando el SDK de Android.

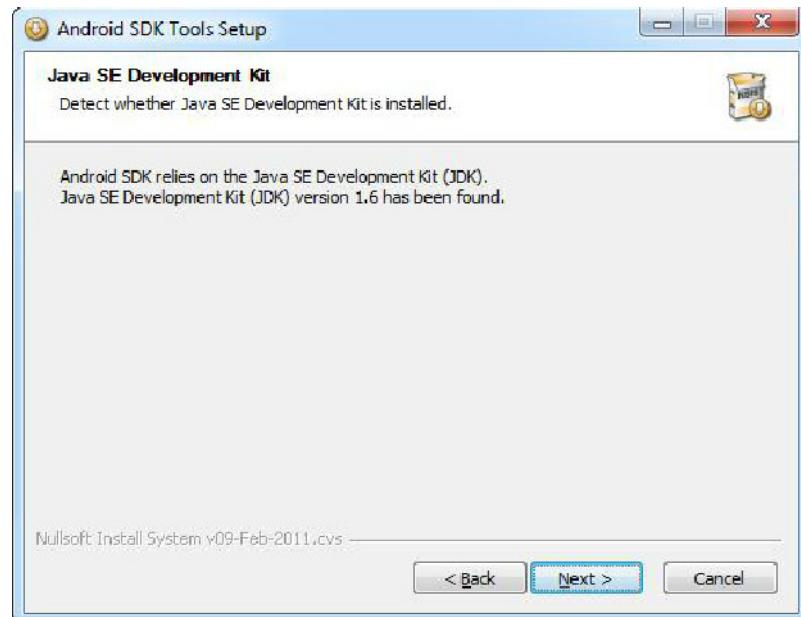


Figura 1.15. Ventana avisando que Java está correctamente instalado.

Volvemos a pulsar en “Next” y dejamos el directorio que viene por defecto:

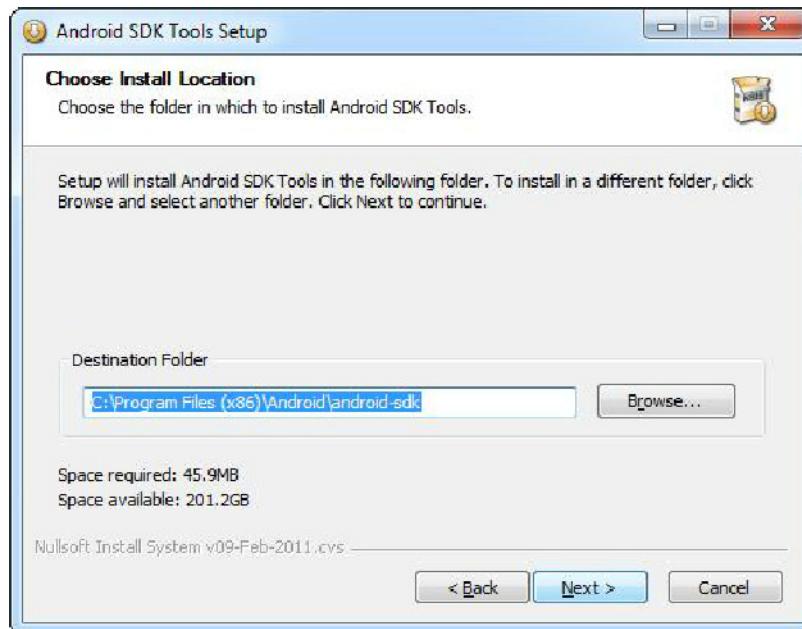


Figura 1.16. Ventana de selección directorio de instalación.

Dejamos las opciones que muestra por defecto y pulsamos sobre “Install”:

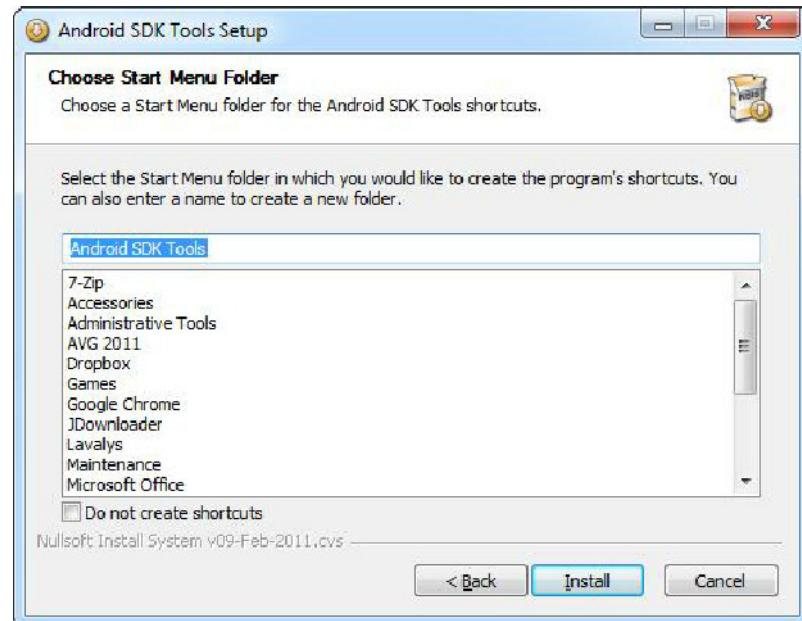


Figura 1.17. Ventana selección carpeta del Menú Inicio.

Una vez que se ha completado el proceso pulsamos en "Next":

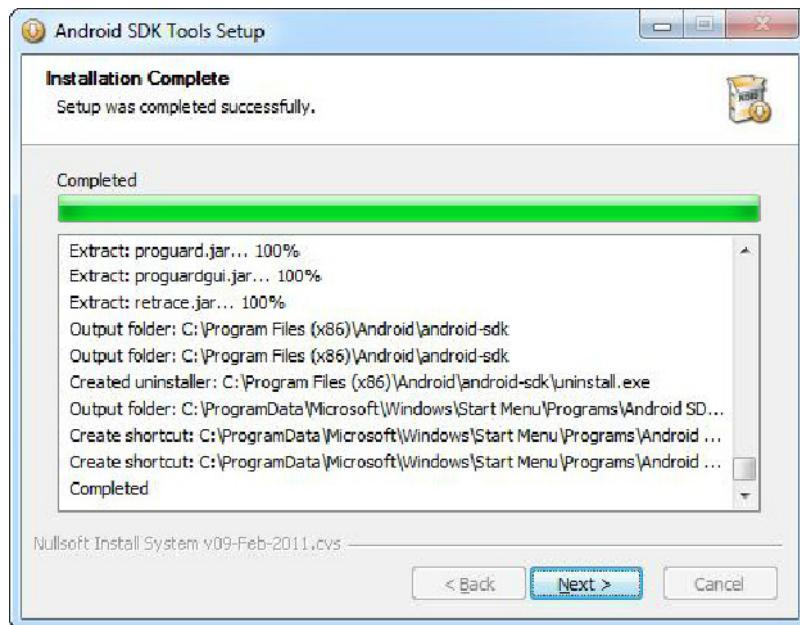


Figura 1.18. Ventana de proceso de instalación de las SDK Tools.

Y acabamos pulsando sobre "Finish" para continuar con la descarga de los elementos necesarios:

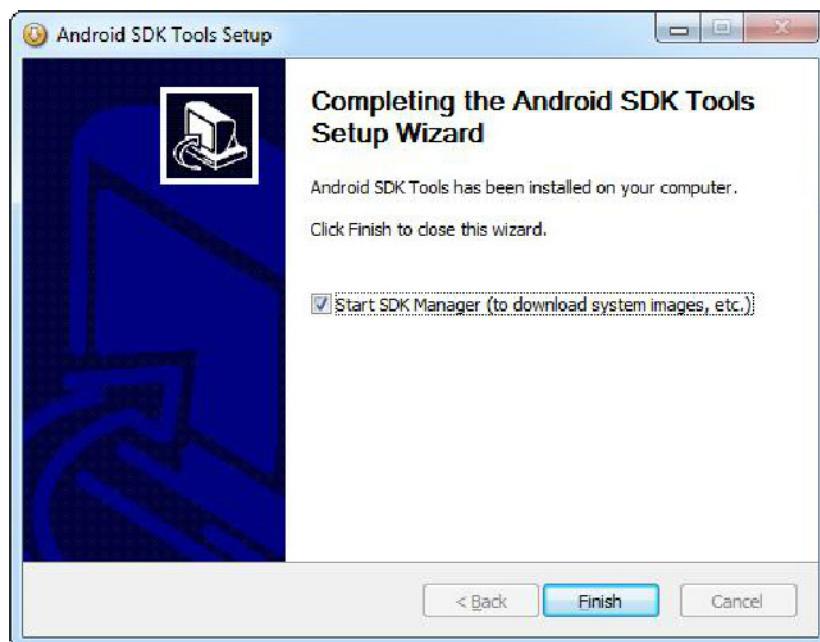


Figura 1.19. Ventana finalización proceso de instalación.

Una vez pulsado, se nos abre una ventana que se quedará en segundo plano en ejecución (*SDK Manager*), y otra ventana que nos deja elegir los paquetes que tenemos que descargarnos para poder desarrollar. Por defecto están seleccionados los paquetes propios de Google, pero recomendamos instalar todos.

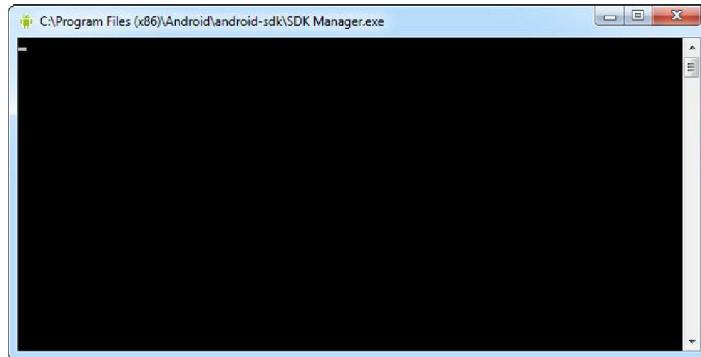


Figura 1.20. Ventana que se muestra en segundo plano.

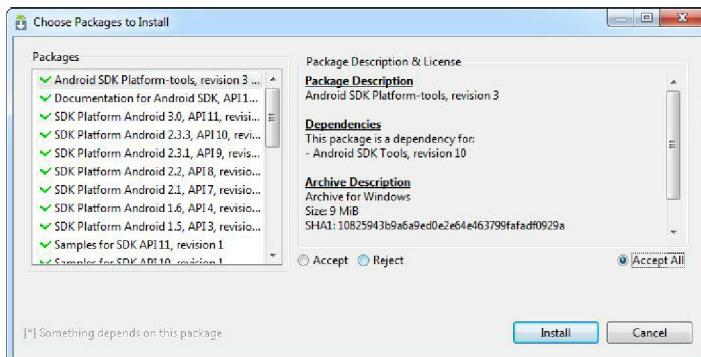


Figura 1.21. Ventana de selección de paquetes a instalar.

Le damos a “Install” y esperamos a que todo se instale. Una vez finalizada la instalación, se quedará una ventana que ya se puede cerrar y continuar con los siguientes pasos.

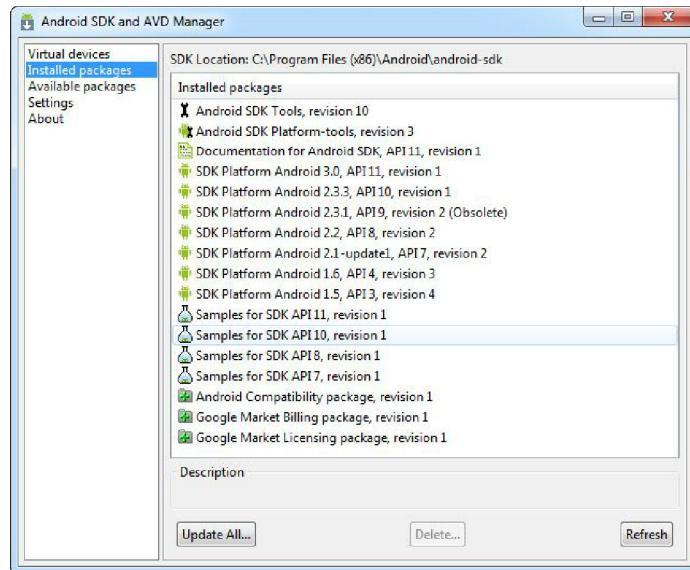


Figura 1.22. Ventana con la vista de los paquetes instalados.

1.2.4. AÑADIR VARIABLES PATH

En este punto debemos añadir el directorio `/tools` del SDK a la variable `Path` de Windows. Si hemos seguido todo el proceso correctamente, el SDK se ha tenido que instalar en la siguiente ruta: `C:\Program Files (x86)\Android\android-sdk`.

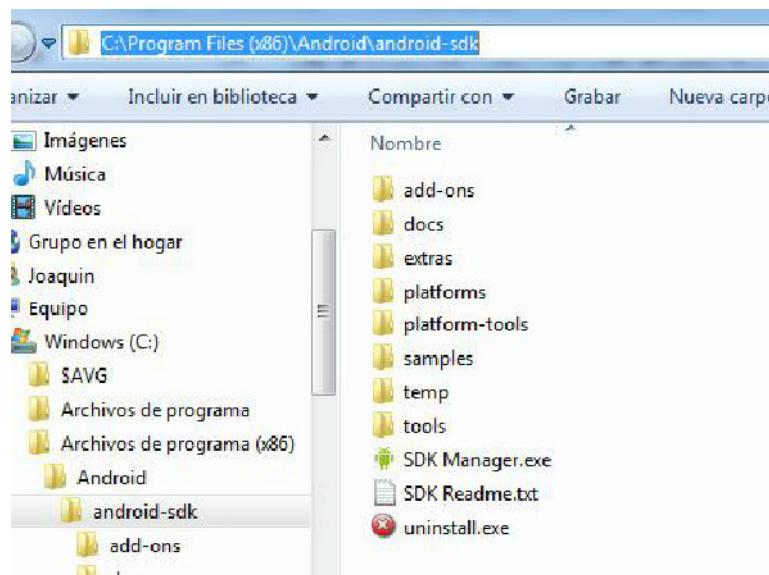


Figura 1.23. Ruta de instalación del SDK de Android.

En el caso de estar usando un sistema operativo x86 (32 bits) sería sin los paréntesis (x86). Para poder confirmar la ruta podemos hacerlo desde el explorador de archivos.

Ahora debemos ir a “*Propiedades del sistema*”. Dependiendo del sistema operativo, se llega a él de diferentes maneras:

- En Windows® Vista y Windows® 7, pulsando la tecla inicio  y escribiendo directamente “*Sistema*”. Pulsamos sobre



- En Windows® XP y 2003 Server, pulsamos con el botón derecho sobre “*Mi PC*” y seleccionamos “*Propiedades*”.

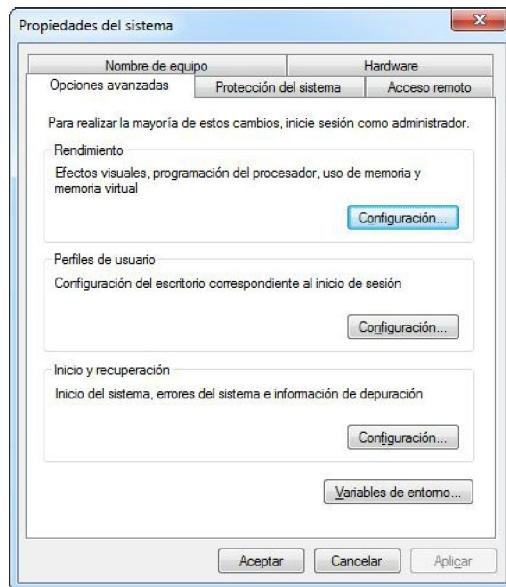


Figura 1.24. Propiedades del sistema.

Una vez aquí, en la pestaña de “*Opciones avanzadas*” pulsamos en “*Variables de entorno...*”. Y en el box inferior buscamos la línea “*Path*”, la seleccionamos y pulsamos en “*Editar*”:

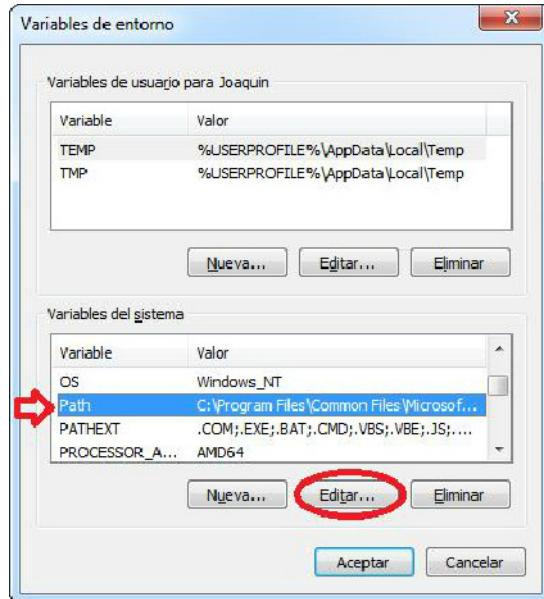


Figura 1.25. Ventana variables de entorno.

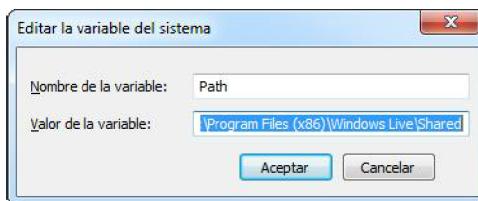


Figura 1.26. Ventana de edición de variables.

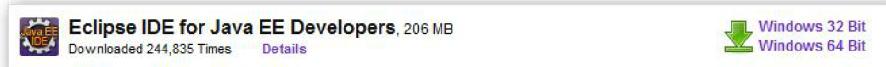
En la ventana que aparece, sin borrar nada, sitúate al final de la línea, e inserta lo siguiente:

;C:\Program Files (x86)\Android\android-sdk\tools

Y pulsamos sobre “Aceptar” hasta que cerremos todas las ventanas.

1.2.5. INSTALACIÓN DE ECLIPSE

Para descargar Eclipse, debemos hacerlo desde su web de descargas: <http://www.eclipse.org/downloads/>. Si lo deseas, puedes acceder al enlace desde la plataforma de estudio. Una vez allí, debemos seleccionar el archivo de 32 o 64 bits dependiendo de nuestro sistema instalado:



Una vez descargado, debemos ir a la carpeta donde se ha descargado el archivo y lo descomprimimos. Una vez descomprimido, nos queda una carpeta que se llama “eclipse”. Esta carpeta se puede dejar en la localización que se desee, puesto que el IDE de eclipse no es necesario instalarlo, únicamente ejecutarlo.



Figura 1.27. Ventana de inicio de Eclipse.

La primera vez que arranque, nos pedirá que le indiquemos un directorio donde colocar nuestro espacio de trabajo (workspace). Seleccionamos la ruta que queramos, que es el lugar donde se guardarán nuestros proyectos. Marcamos la opción “*Use this as the default...*” para que no nos vuelva a preguntar más y pulsamos OK.

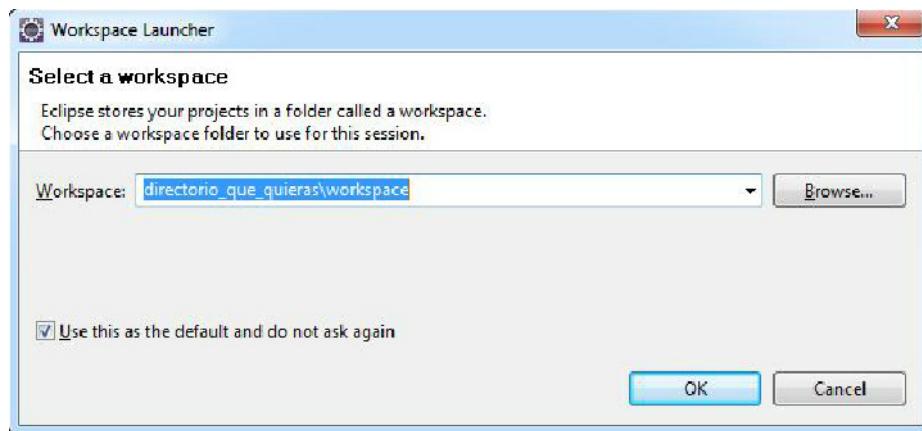


Figura 1.28. Ventana de selección del workspace.

1.2.6. INSTALACIÓN ANDROID DEVELOPMENT TOOLS (ADT)

Dentro de Eclipse, seleccionamos “*Install New Software...*” en el menú “*Help*”. Aparecerá una ventana de addons.

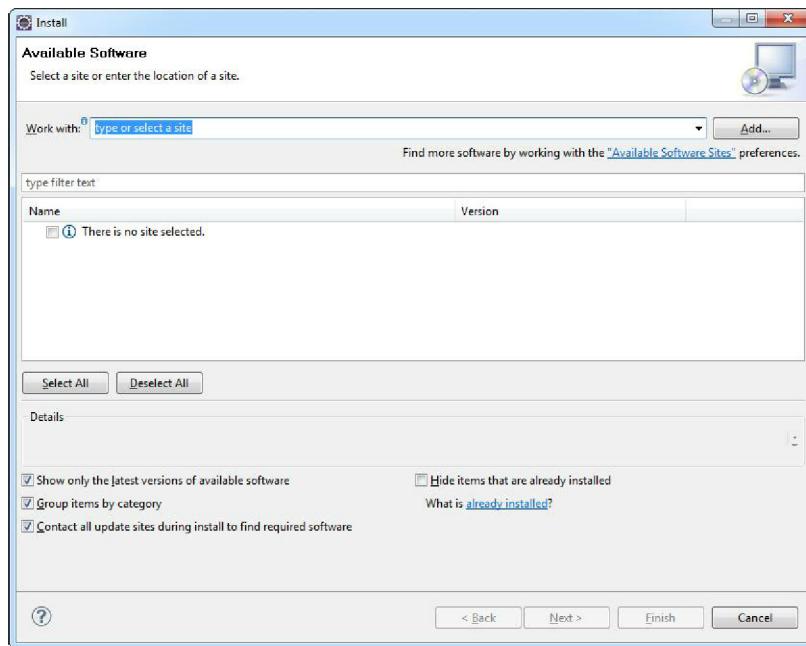


Figura 1.29. Ventana de inicio de instalación del ADT.

Hacemos click en “*Add...*” en la parte superior y escribimos lo siguiente:

Name: Android
Location: https://dl-ssl.google.com/android/eclipse/

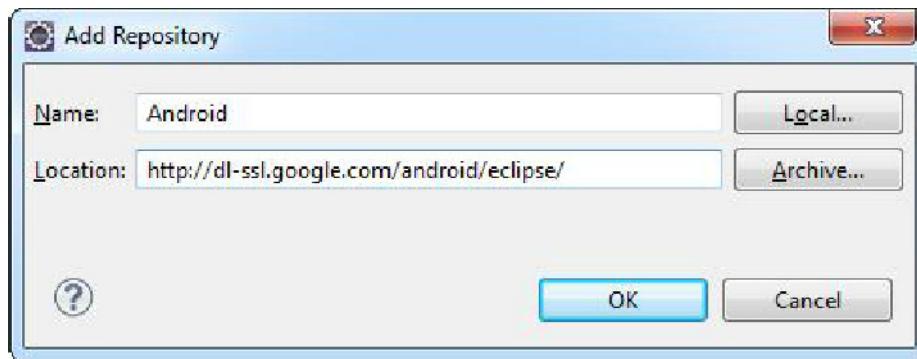


Figura 1.30. Ventana de añadir repositorio.

Pulsamos en “OK” y seleccionamos en el box “Developer Tools”

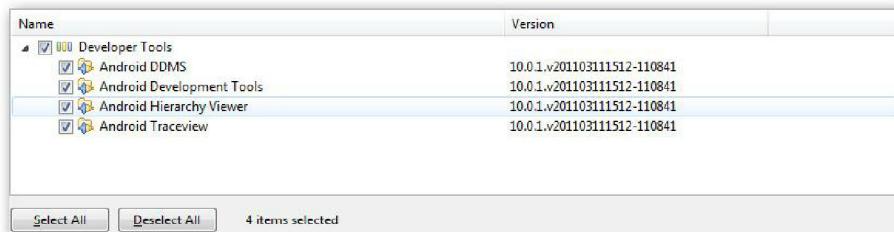


Figura 1.31. Seleccionamos todos las “Developer Tools”.

Seleccionamos todos y pulsamos en “Next”. El programa calcula los requerimientos y espacio necesario. Volvemos a pulsar en “Next” y aceptamos los términos de licencia, y pulsamos “Finish” para finalizar la instalación:

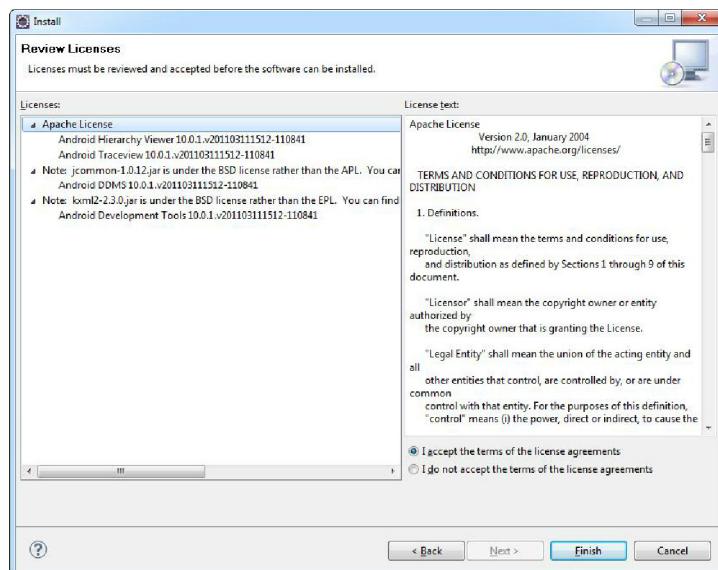


Figura 1.32. Aceptamos todos los términos de licencia.

El sistema nos preguntará si queremos seguir instalando software que no está firmado digitalmente. Pulsamos en “OK” y continuamos con la instalación:

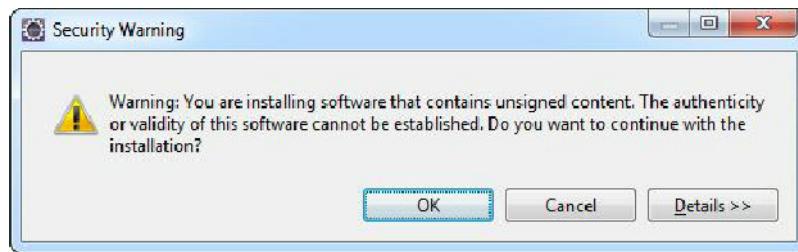


Figura 1.33. Ventana de aviso de Windows.

Una vez finalizada la instalación, reiniciamos Eclipse. Una vez reiniciado, debemos configurar Eclipse para que acceda al SDK de Android ya instalado. Abimos el menú “Window” y seleccionamos “Preferences”. En el apartado Android, en el cuadro de texto “SDK Location” debemos introducir la ruta donde hemos descomprimido el SDK de Android (C:\Program Files (x86)\Android\android-sdk) y pulsamos sobre “Apply”. Nos aparecerán todos los SDK anteriormente instalados.

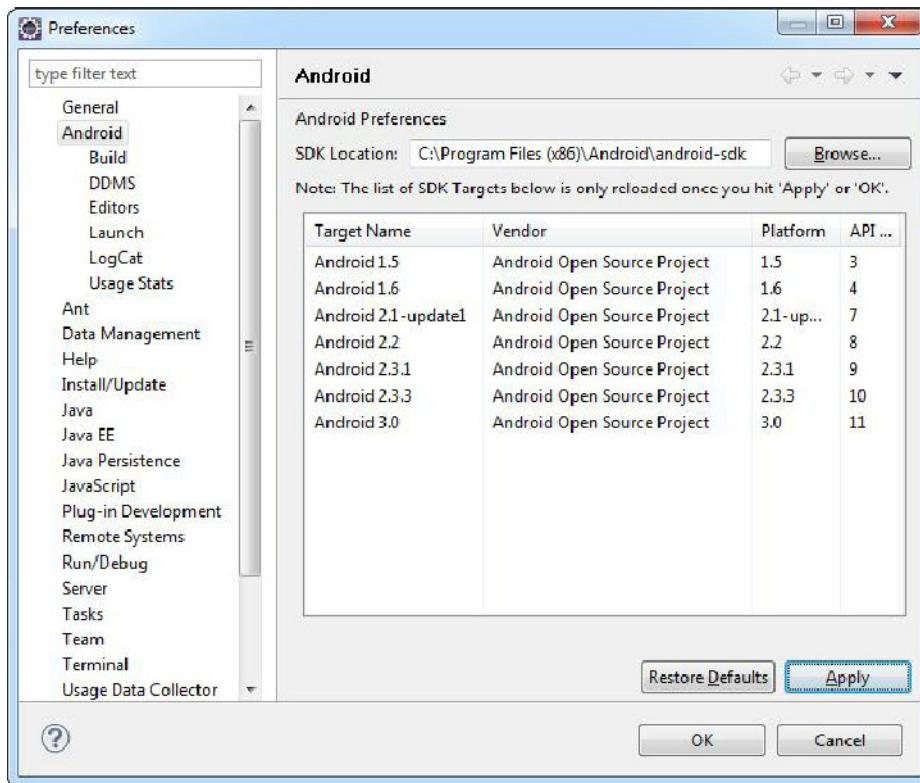


Figura 1.34. Ventana preferencias de SDK de Android en Eclipse.

Presionamos “Ok” para cerrar la ventana de preferencias. Con esto ya tenemos el SDK de Android integrado en Eclipse para empezar a programar.

1.3. MI PRIMER PROGRAMA

En este capítulo vamos a realizar una primera toma de contacto con una aplicación Android, antes de pasar a mayores contenidos de la programación. Lo primero de todo será crear el emulador de Android, donde veremos los resultados de nuestras aplicaciones.

1.3.1. CREAR UN EMULADOR AVD

El Dispositivo Virtual de Android (AVD), donde se realizan todas las pruebas y se visualiza el resultado de la aplicación en ejecución, es un emulador de un dispositivo real con Android.

Una vez iniciado Eclipse, debemos pulsar sobre el botón de “*Android SDK y AVD Manager*” . Este nos abrirá una pantalla como la siguiente:

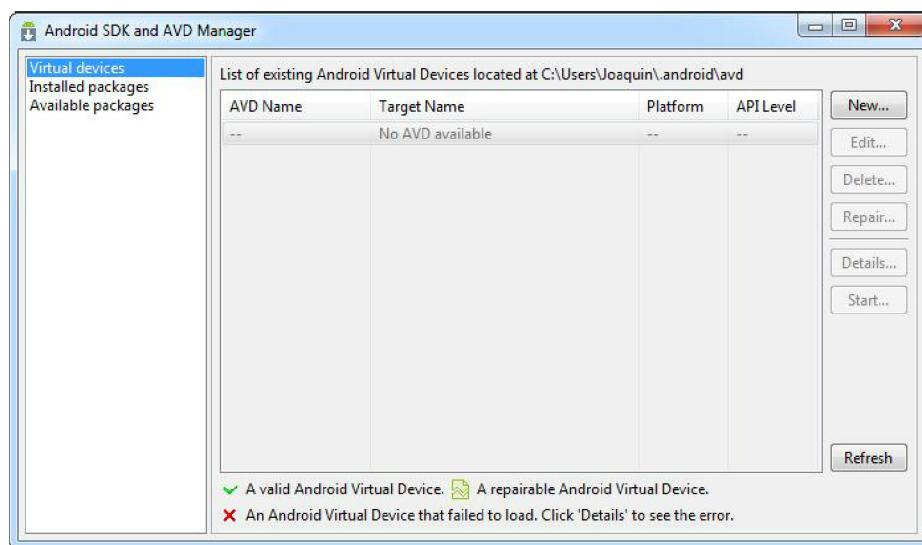


Figura 1.35. Pantalla de Android SDK and AVD Manager.

Pulsaremos sobre “*New*” y allí rellenaremos los siguientes campos con los siguientes datos:

Name: *avd_seas* (nombre del emulador)

Target: *Android 2.1-update1 – API Level 7* (mínimo dispositivo necesario para ejecutar la aplicación)

Size: *64 mb* (tamaño de la tarjeta SD que se emulará para el dispositivo)

Built-in: *QVGA* (resolución de la pantalla)

Hardware: dejar lo que sale por defecto, ya que depende de la resolución de pantalla.

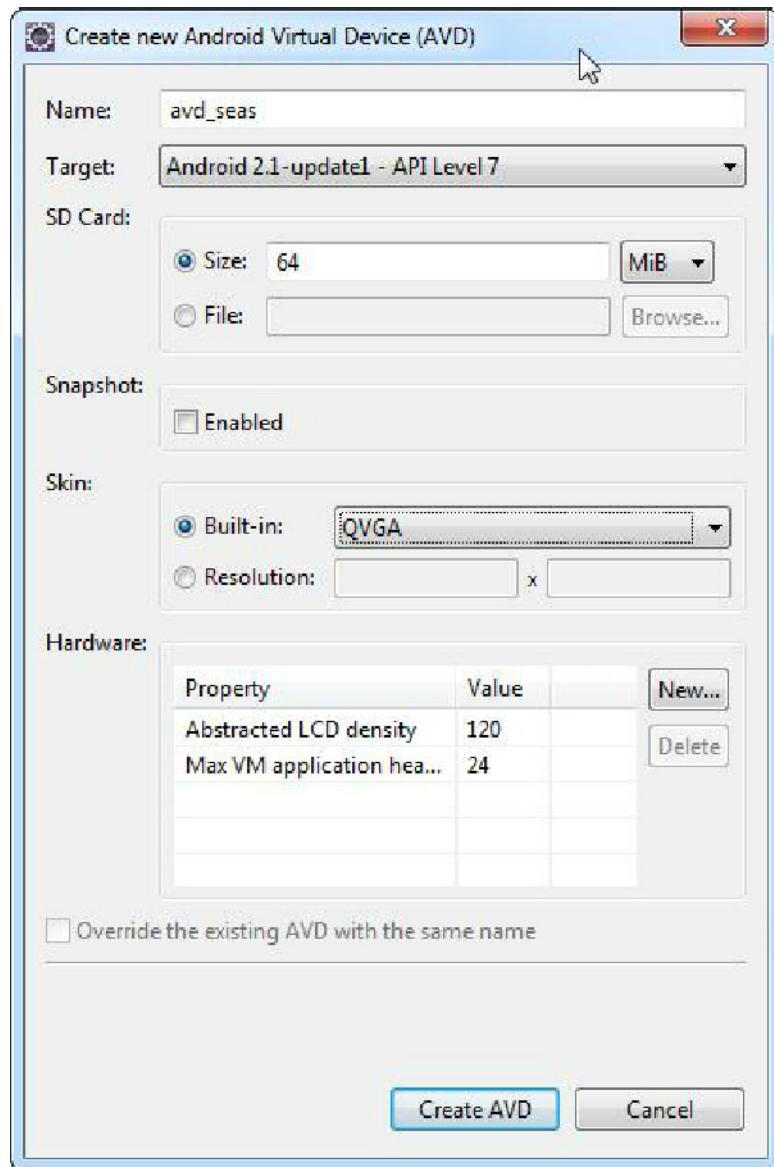


Figura 1.36. Pantalla de creación de AVD.

Una vez rellenados todos los campos, pulsamos sobre “Create AVD”. Ya tendremos creado nuestro emulador o dispositivo virtual. Si queremos verlo en funcionamiento únicamente debemos seleccionarlo y pulsar sobre “Start”:

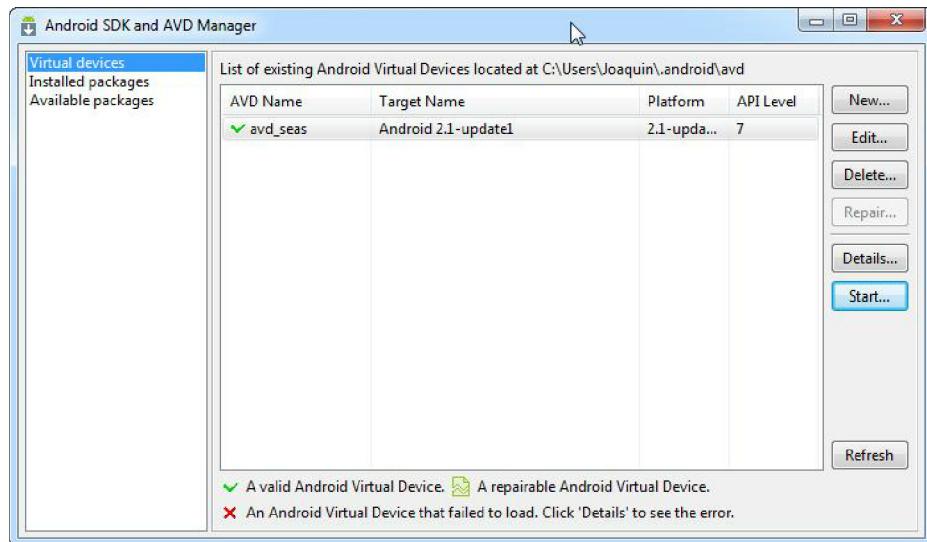


Figura 1.37. Ventana de selección de AVD.

Pulsamos en “Launch” en la nueva ventana que nos aparece y esperamos a que arranque por primera vez:

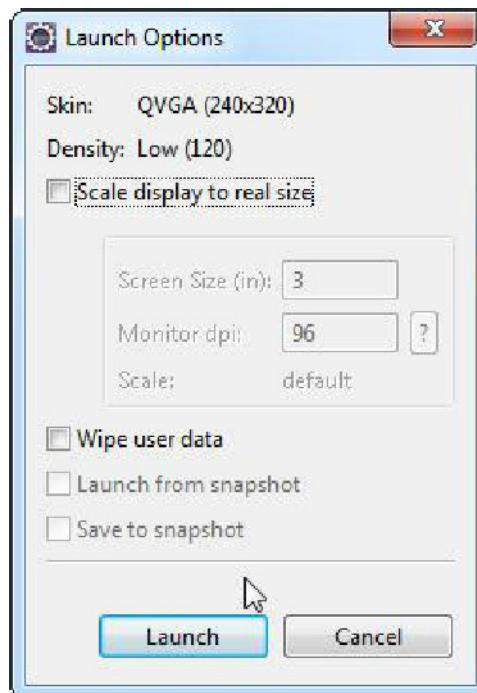


Figura 1.38. Opciones del AVD.

La primera vez que ejecutamos el emulador virtual puede tardar bastante tiempo en arrancar, dependiendo del equipo del que dispongamos. Una vez que ya haya cargado el emulador completamente, nos aparecerá una ventana similar a ésta:



Figura 1.39. Avd_seas - Emulador de Android.

1.3.2. INSTALACIÓN DE DISPOSITIVO ANDROID

Otra manera de poder probar las aplicaciones es hacerlo mediante un dispositivo móvil con SO Android. Para poder realizar la conexión de dicho dispositivo al ordenador, debemos de activar el modo “Depuración USB” en el menú del móvil.

Dependiendo del móvil y del fabricante, el menú puede estar en un sitio u otro. Consultamos el manual del fabricante o la página web del mismo para encontrar la ubicación exacta de este menú.

Por defecto se encuentra en el menú *Ajustes* → *Aplicaciones* → *Desarrollo*. Una vez allí, debemos activar el modo “Depuración USB”. Pero como cada terminal es diferente y puede que necesitemos drivers adicionales, deberemos buscar información relativa a nuestro dispositivo en la página del fabricante.

1.3.3. SELECCIÓN DEL EMULADOR

Si disponemos de un dispositivo con SO Android, podemos seleccionar en Eclipse en qué sistema queremos probar nuestra aplicación. Por defecto, éste se encuentra configurado para que se ejecute directamente en el emulador que anteriormente hemos creado, pero puede ser que queramos ver cómo queda realmente en nuestro propio móvil.

Para ello debemos acceder al menú *Run → Run Configurations...*. Una vez que has accedido a esa ventana, en la parte izquierda de la ventana seleccionamos “Android Application” y allí el proyecto que estemos desarrollando. Después, en la parte central seleccionamos la pestaña de “Target” y seleccionamos “Manual”.

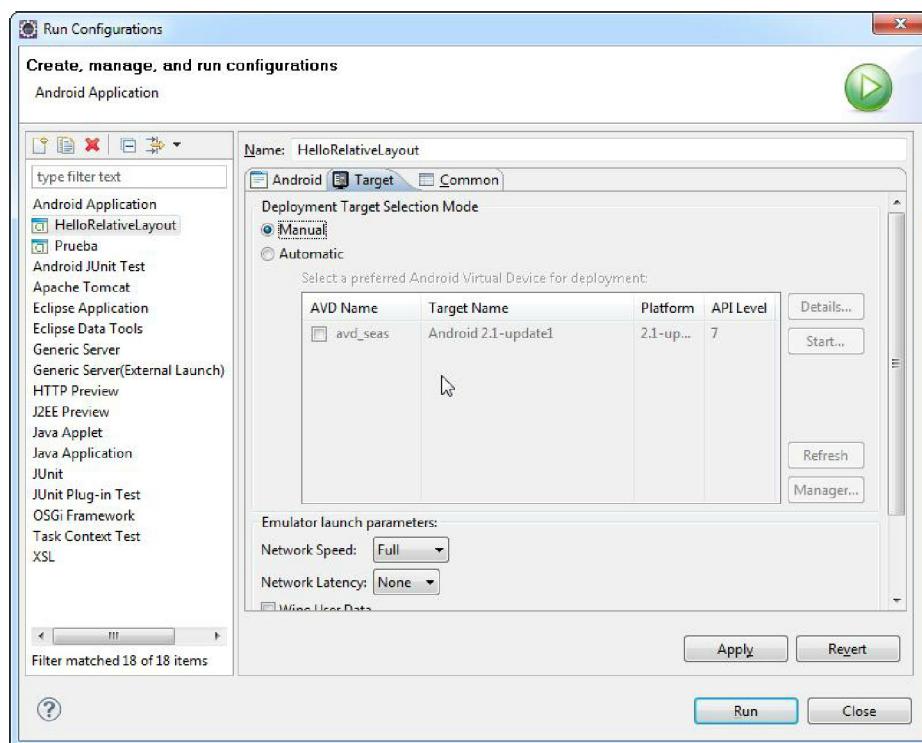


Figura 1.40. Menú de Configuración de Ejecución de aplicaciones Android

Una vez que tenemos configurado este modo de ejecución, la próxima vez que vayamos a compilar nuestra aplicación, el sistema nos preguntará en qué dispositivo queremos realizar la ejecución.

Para ver en funcionamiento el emulador, vamos a cargar un ejemplo del repositorio de Android. Para esto debemos ir a *File → New → Android Project*. Una vez nos aparezca la pantalla de creación, seleccionamos el “Build Target” Android 2.2, posteriormente seleccionamos “Create Project from existing simple” y en el desplegable seleccionamos “Spinner”:

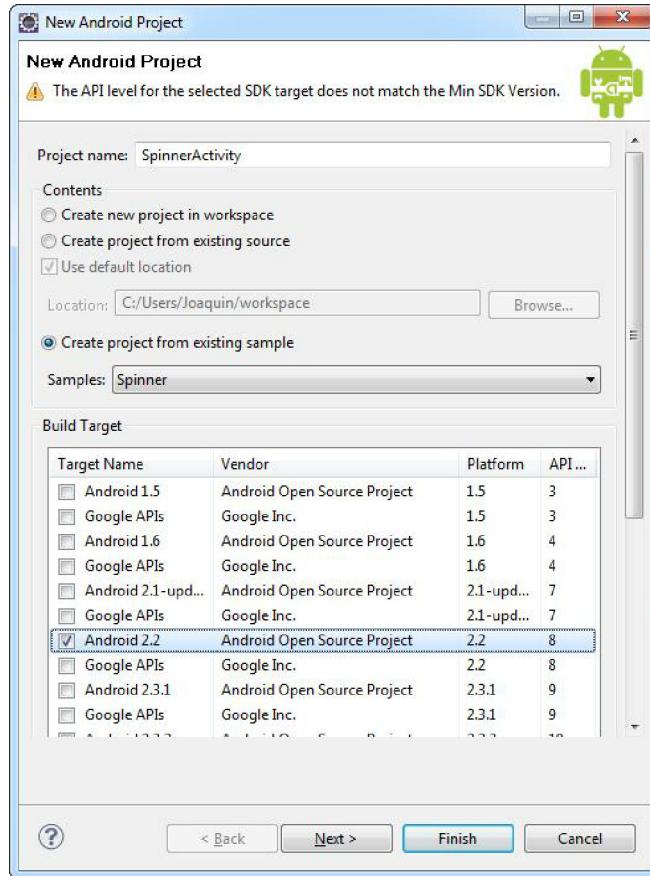


Figura 1.41. Ventana de nuevo proyecto de Android.

Pulsamos sobre “Finish” y se habrá creado el proyecto en nuestro *workspace*. Para ejecutarlo nos situamos encima del nuevo proyecto, *botón derecho* → *Run as* → *Android Application*. En ese momento nos aparece la ventana de selección del dispositivo donde seleccionamos dónde ejecutar la aplicación. En nuestro caso vamos a seleccionar el emulador de Android que acabamos de configurar y pulsamos en “Ok”.

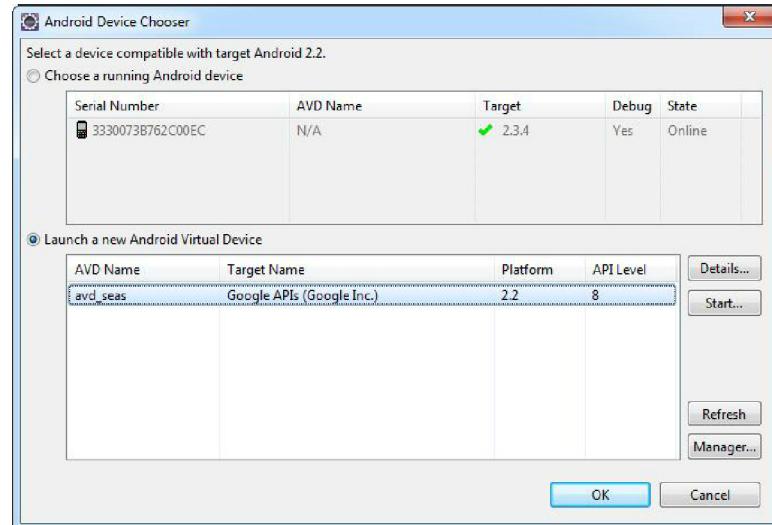


Figura 1.42. Pantalla de selección del dispositivo donde ejecutar la aplicación.

En la parte superior nos aparecerá el dispositivo móvil que tengamos conectado al ordenador. Recordemos que tenemos que tener activada la “Depuración USB” en nuestro dispositivo para poder realizar las pruebas.

Una vez que arranca el emulador, veremos en él la aplicación de prueba que hemos cargado:



Figura 1.43. Vista del emulador ejecutando la aplicación.



Es importante, aunque no indispensable, disponer de un terminal con Android para poder realizar pruebas de desarrollo sobre el terminal, ya que el emulador no puede realmente determinar la ubicación real o realizar llamadas telefónicas. Además, el emulador es un dispositivo genérico y no puede simular cualquier característica de un teléfono específico. El hecho de que la aplicación se ejecute bien en el emulador no garantiza que funcione correctamente en el dispositivo.

1.3.4. DDMS

DDMS (*Dalvik Debug Monitor Server*) es el monitor de depuración de la máquina virtual Dalvik, es decir, la máquina virtual de procesos de Android. Con esta aplicación podemos ver los servicios de redireccionamiento de los puertos, realizar capturas de pantalla, captura y listado de información en el dispositivo móvil, hacer LogCats (registro de procesos de Android), ver procesos e información del estado de la radio (señal de móvil), emular llamadas entrantes o SMS e incluso emular una posición determinada de una ubicación. Todo esto hace que el DDMS sea una herramienta muy útil para cualquier desarrollador.

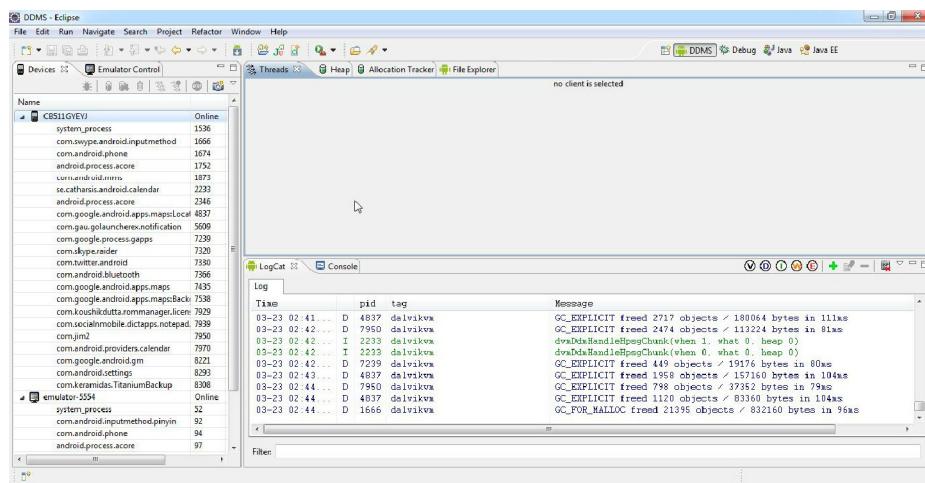
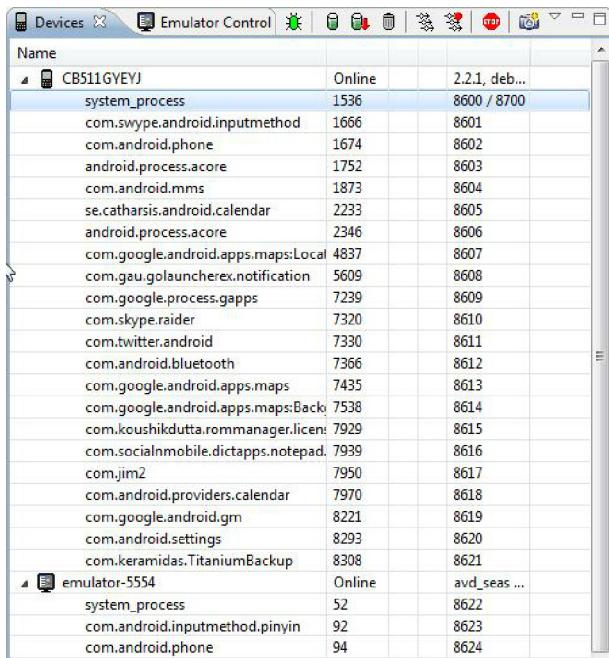


Figura 1.44. Visión general de la perspectiva DDMS.

DDMS trabajará con un dispositivo conectado al ordenador o con el emulador del SDK (`avd_seas`), si ambos están conectados y en funcionamiento al mismo tiempo, por defecto usa el emulador.

En la parte izquierda podemos encontrar cada emulador o dispositivo conectado, con el listado de las máquinas virtuales (MV) en ejecución dentro de cada dispositivo.

Éstas se identifican por el nombre del paquete de la aplicación que aloja, y de esta manera podemos encontrar y conectar con la MV de la actividad que deseamos depurar.



The screenshot shows the DDMS (Android Debug Bridge) interface. The top bar has tabs for 'Devices' (selected), 'Emulator Control', and other tools. Below is a table with columns: Name, Status, and two numerical columns. The table lists several devices and their processes:

Name	Status		
CB511GYEYJ	Online	2.2.1, deb...	
system_process	1536	8600 / 8700	
com.swype.android.inputmethod	1666	8601	
com.android.phone	1674	8602	
android.process.acore	1752	8603	
com.android.mms	1873	8604	
se.catharsis.android.calendar	2233	8605	
android.process.acore	2346	8606	
com.google.android.apps.maps:Local	4837	8607	
com.gau.golauncherex.notification	5609	8608	
com.google.process.gapps	7239	8609	
com.skype.raider	7320	8610	
com.twitter.android	7330	8611	
com.android.bluetooth	7366	8612	
com.google.android.apps.maps	7435	8613	
com.google.android.apps.maps:Back	7538	8614	
com.koushikdutta.rommanager.licen	7929	8615	
com.socialinmobile.dictapps.notepad.	7939	8616	
com.jim2	7950	8617	
com.android.providers.calendar	7970	8618	
com.google.android.gm	8221	8619	
com.android.settings	8293	8620	
com.keramidas.TitaniumBackup	8308	8621	
emulator-5554	Online	avd_seas ...	
system_process	52	8622	
com.android.inputmethod.pinyin	92	8623	
com.android.phone	94	8624	

Figura 1.45. Panel izquierdo del DDMS

En la columna más cercana al nombre del paquete nos encontramos con el identificador de proceso de Linux, y en la columna más alejada al nombre del paquete se ven los puertos a los que cada MV está conectada. Cada proceso tiene su identificación o PID, y como Android está basado en el Kernel de Linux, se nos muestran los identificadores aquí.

Los botones que se ven en la imagen superior tienen las siguientes funciones:

-  Activa el muestreo de las actualizaciones de la memoria.
-  Realiza un volcado de memoria en un archivo HPROF.
-  Hace un *Garbage Collection* instantáneo al pulsarlo, es decir, borra toda la información recogida.
-  Activa las actualizaciones de información de los hilos en su pestaña correspondiente.
-  Activa el método de perfiles.



Detiene la MV seleccionada en el dispositivo.



Realiza una captura de pantalla de la aplicación en ejecución en el dispositivo.

Otra parte importante del DDMS es el Emulador Control, donde podemos simular el envío de mensajes o la realización de llamadas al Emulador AVD.

Podemos incluso simular una localización de una ubicación para poder realizar pruebas de aplicaciones que necesiten de este tipo de datos.

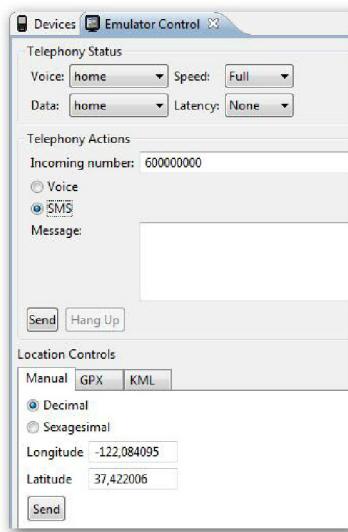


Figura 1.46. Emulator Control.

En el panel derecho encontramos multitud de pestañas en las que no vamos a entrar muy detenidamente, ya que mostramos una visión general de la programación en Android.

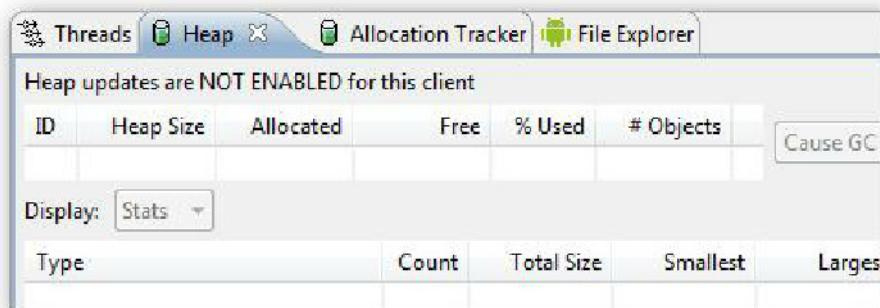


Figura 1.47. Panel derecho de la perspectiva DDMS.

1.3.5. HOLA MUNDO EN ANDROID

Vamos a realizar nuestra primera aplicación para dispositivos Android. Para ello debemos ir al menú *File* → *New* y seleccionar “*Android Project*”. Allí indicamos su nombre, el “*target*” deseado, el nombre de la aplicación, el paquete java por defecto para nuestras clases y el nombre de la clase (*Activity*) principal.

En nuestro caso, un ejemplo de cómo hacerlo sería así:

Project Name: *SeasHolaMundo*

Application Name: *Seas HolaMundo*

Package Name: *com.seas.ejemplo.HolaMundo*

Create Activity: *SeasHolaMundo*

Min SDK Version: 3

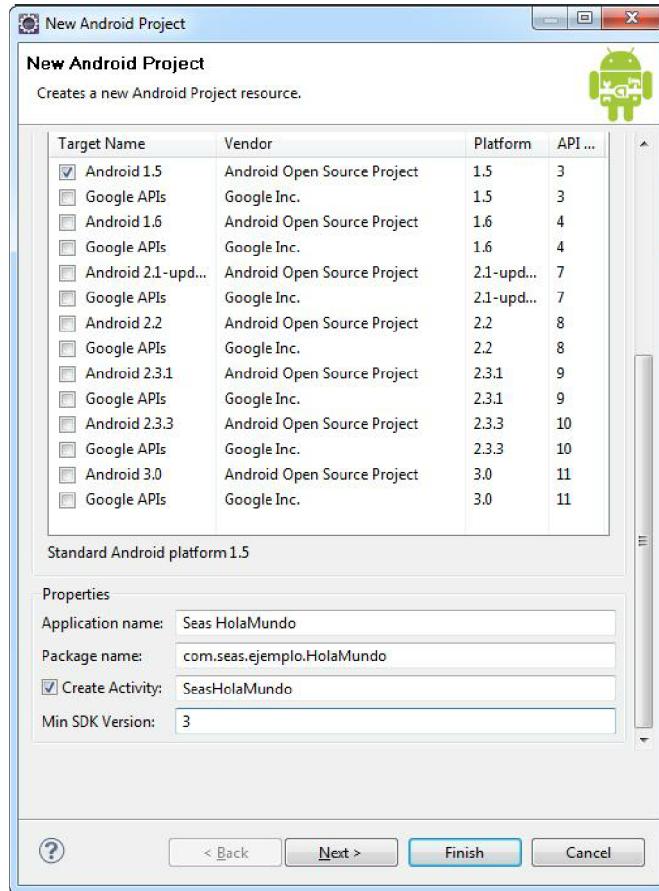


Figura 1.48. Ventana de creación de un nuevo proyecto Android.

Con esto creamos toda la estructura de carpetas necesaria para compilar un proyecto para Android (hablaremos de ella más adelante).

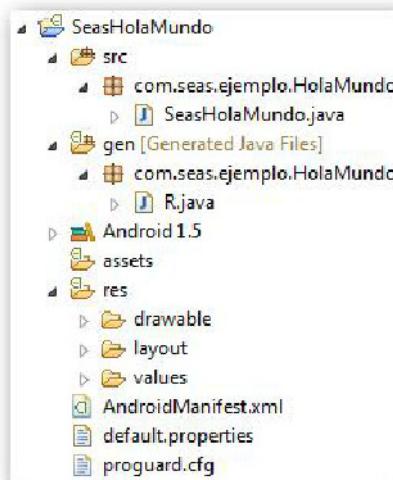


Figura 1.49. Estructura de carpetas de una aplicación Android.

Para continuar con el ejemplo, debe desplegarse en el explorador de paquetes la carpeta denominada “src”. En esta carpeta existirán dos ficheros: “SeasHolaMundo.java” y “R.java”. Ambos constituyen hasta el momento los dos únicos ficheros fuente del proyecto.

El objetivo es crear una simple interfaz de usuario que permita mostrar el mensaje “Mi primera aplicación, Hola Seas”. Para ello, el código ha de ser el siguiente:

```
package com.seas.ejemplo.HolaMundo;

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

public class SeasHolaMundo extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle icicle) {
        super.onCreate(icicle);
        TextView tv = new TextView(this);
        tv.setText("Mi primera aplicación, Hola Seas");
        setContentView(tv);
    }
}
```

En Android, la interfaz de usuario está compuesta de una jerarquía de clases llamadas `Views`. Una `View` es simplemente un objeto que puede ser pintado en la pantalla, tal como una animación o un botón. El nombre específico para la subclase de `View` que manipula texto es `TextView` cuya sintaxis para crear objetos de este tipo es:

```
TextView tv = new TextView(this);
```

Al constructor del `TextView` le pasamos como argumento una instancia de `Context`, que simplemente es un enlace hacia el sistema y es el que da acceso a ciertos recursos, como pueden ser bases de datos o a la configuración de algunas características de la aplicación.

La clase `Activity` hereda de `Context` y debido a que nuestra aplicación es un subclase de `Activity`, entonces también lo es de `Context`, de ese modo, la clase `HolaMundo.java` también es un contexto válido en Android y podemos referenciarla usando `this` como argumento del constructor `TextView`.

Una vez creada la etiqueta de texto, para asignarle un valor simplemente se llama al método `setText()` con la cadena “[Mi primera aplicación, Hola Seas](#)” como argumento:

```
tv.setText("Mi primera aplicación, Hola Seas");
```

Como último paso, se tiene que vincular los elementos visuales que se hayan creado (las “vistas”, como `TextView`) con la pantalla que la aplicación mostrará al usuario. Este paso se realiza mediante el método `setContentView()` de la clase `Activity`, que indica al sistema que `View` debe ser asociada con la IU de la `Activity`. Si una `Activity` no llama a este método, ninguna IU será presentada y el sistema simplemente mostrará una pantalla en blanco:

```
setContentView(tv);
```

Para ejecutar el proyecto tal cual, debemos hacerlo accediendo al menu *Run → Run As y seleccionando “Android Application”.*

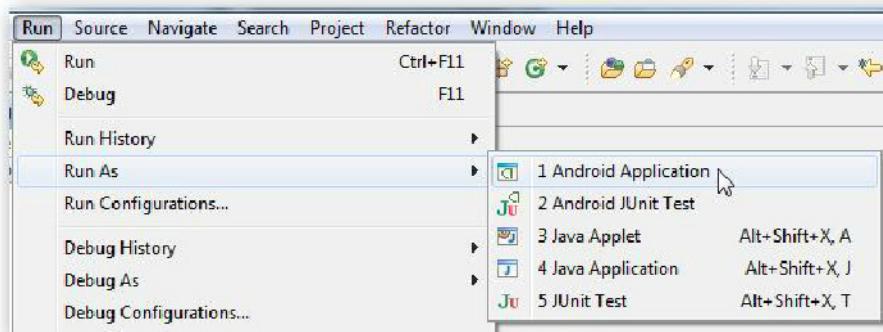


Figura 1.50. Ejecutar la aplicación Android

Al ejecutar el proyecto, se abrirá el diálogo de selección de dispositivo donde ejecutar la aplicación. Seleccionaremos el que configuramos anteriormente o, si tenemos conectado el dispositivo en modo depuración USB, seleccionamos el móvil y se cargará automáticamente nuestra aplicación.



Figura 1.51. Emulador de Android con aplicación Seas HolaMundo.

1.4. EXPLORANDO LA SDK DE ANDROID

Las SDK de Android son un conjunto de herramientas para desarrolladores que permiten poner en marcha proyectos de desarrollo. En este apartado vamos a conocer su funcionamiento y los conceptos básicos que debemos tener claros.

1.4.1. INICIAR EL DESARROLLO DE ANDROID. COMPONENTES DE UNA APLICACIÓN

Nos vamos a adentrar en el desarrollo de Android y para ello debemos tener muy claros unos cuantos conceptos en la programación Android, que serán la piedra angular de todo desarrollador.

1.4.1.1. EL ARCHIVO ANDROIDMANIFEST.XML

Este archivo está presente en todas las aplicaciones. En él se especifican los componentes de la aplicación, así como su configuración global. En el `AndroidManifest.xml`:

- Se describen los nombres de los paquetes java que tiene la aplicación. Este nombre es como un identificador único para la solicitud.
- Se describen los componentes de la aplicación, es decir, las actividades, los servicios, los receptores de radiodifusión, y los proveedores de contenido de los que la aplicación se compone. También se describen los nombres de las clases que implementan cada uno de los componentes y se declaran sus capacidades (por ejemplo, los mensajes `Intent` que pueden manejar). Se define el `Activity` que será la pantalla principal cuando arranque la aplicación.
- Se declaran qué permisos son necesarios para poder acceder a otras partes protegidas de la API e interactuar con otras aplicaciones.
- Se declaran los permisos que los demás están obligados a tener para poder interactuar con los componentes de nuestra aplicación.
- Se enumeran las clases `Instrumentation`, que proporcionan los perfiles y otra información mientras la aplicación está en ejecución. Estas declaraciones están presentes en el manifiesto únicamente mientras la aplicación está siendo desarrollada y probada, una vez la aplicación ha sido publicada, deben ser borrados.
- Se declara el nivel mínimo requerido de la API de Android.
- Se enumeran las librerías que la aplicación necesita tener vinculadas.

Estructura del archivo AndroidManifest.xml

En el siguiente diagrama se ve la estructura general del archivo `AndroidManifest.xml` y todos los elementos que puede contener:

```
<?xml version= "1.0" encoding= "utf-8"?>

<manifest>

    <uses-permission />
    <permission />
    <permission-tree />
    <permission-group />
    <instrumentation />
    <uses-sdk />
    <uses-configuration />
    <uses-feature />
    <supports-screens />
    <compatible-screens />
    <supports-gl-texture />
    <application>
        <activity>
            <intent-filter>
                <action />
                <category />
                <data />
            </intent-filter>
            <meta-data />
        </activity>
        <activity-alias>
            <intent-filter> . . . </intent-filter>
            <meta-data />
        </activity-alias>
        <service>
            <intent-filter> . . . </intent-filter>
            <meta-data/>
        </service>
        <receiver>
            <intent-filter> . . . </intent-filter>
            <meta-data />
        </receiver>
        <provider>
            <grant-uri-permission />
            <meta-data />
        </provider>
        <uses-library />
    </application>

</manifest>
```

En una aplicación habitual, dentro de este archivo habrá un elemento <application>, y dentro del cual habrá uno o varios elementos <activity>. Cada uno de estos elementos realizará una interacción con los usuarios (habitualmente una ventana), y se corresponderá con una clase que hereda de la clase `Activity`.

Todos estos conceptos se terminarán de entender en temas posteriores.

1.4.1.2. ACTIVITY

Un `Activity` es una parte del código, con un objetivo determinado, que interactúa con el usuario.

Activity



Un `Activity` es el componente de Android que proporciona una ventana en una aplicación, con la que los usuarios pueden interactuar para realizar una acción, como buscar un contacto, marcar un teléfono, realizar una foto, enviar un email o ver un mapa.

Cada una de las actividades, conlleva una pantalla o ventana en la que se dibuja su interfaz de usuario. Dicha pantalla o ventana puede llenar toda la pantalla del dispositivo, pero también puede ser de menor tamaño o incluso flotar en la parte superior de otras ventanas.

Esta clase crea una ventana que muestra una interfaz de usuario, la cual está definida a su vez en una instancia de otra clase, la clase `View`.

Cuando ejecutamos una aplicación Android, lo primero que se va a mostrar al usuario es la ventana definida por la actividad que esté marcada en el `AndroidManifest.xml` como principal. Las actividades se gestionan como una pila, así que desde una actividad se puede llamar a otra, y cuando ésta finaliza se retorna a la actividad inicial.

Una actividad puede estar ejecutándose, en pausa o detenida. Es decir, cuando está en ejecución es visible e interacciona con el usuario; está en pausa cuando es visible pero otra ventana, ya sea transparente o que no ocupe toda la pantalla, tiene el foco; y está detenida cuando no es visible. Pero en todos estos casos, la clase mantiene su información.

El siguiente gráfico nos muestra el ciclo de vida de una aplicación Android. En él podemos observar que `onCreate()` y `onDestroy()` abarcan todo el ciclo de vida, cada uno de estos métodos representan el principio y el fin de la actividad.

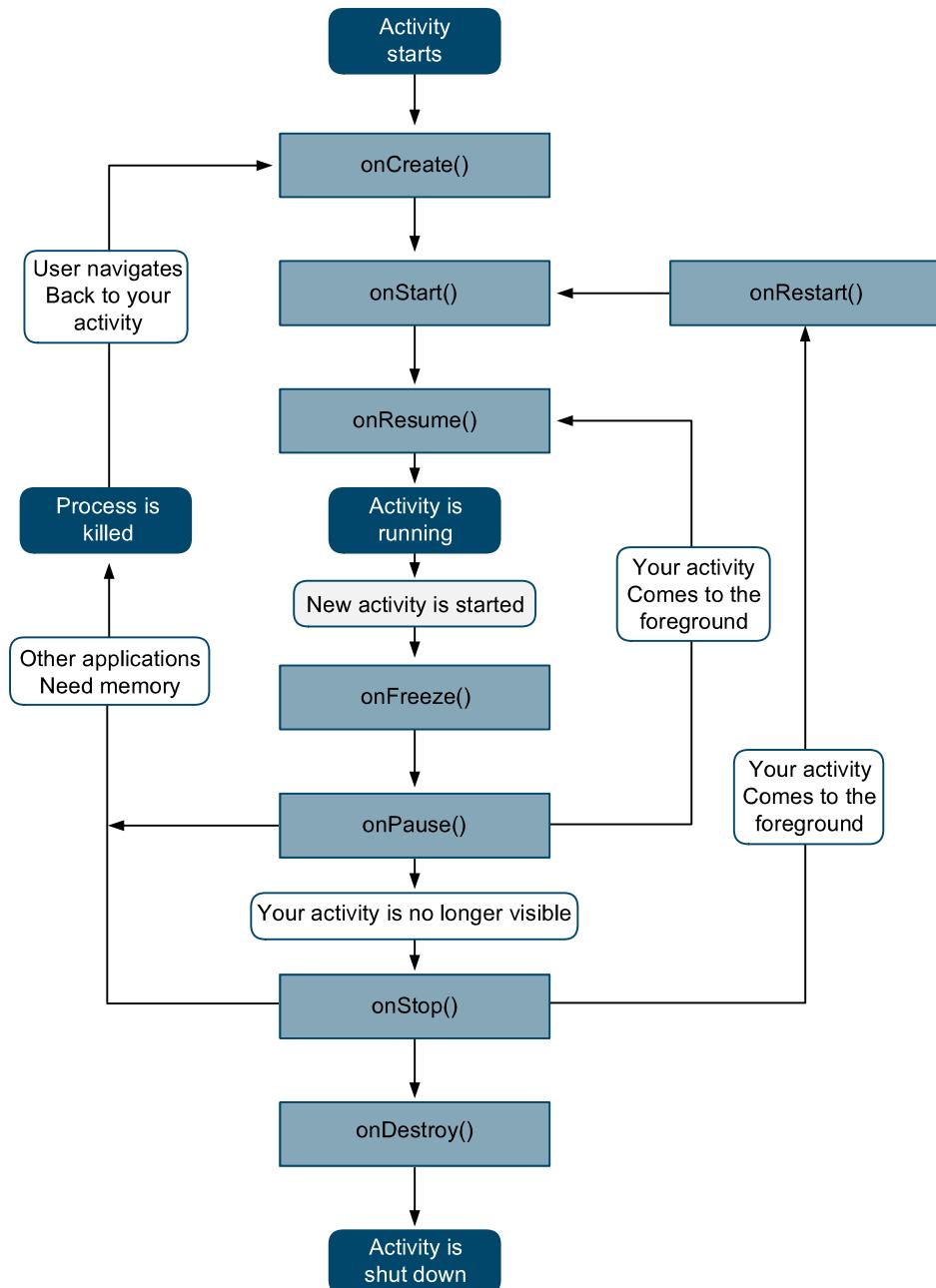


Figura 1.52. Ciclo de vida de una aplicación Android.

Vemos también que `onStart()` y `onStop()` representan la parte visible del ciclo de vida, es decir, que desde `onStart()` hasta `onStop()`, la actividad será visible para el usuario, aunque seguramente no tenga el foco de acción por existir otras actividades superpuestas con las que el usuario está interactuando. Éstas pueden ser llamadas varias veces.

Debemos darnos cuenta que `onResume()` y `onPause()` son las que delimitan la parte útil del ciclo de vida. Desde `onResume()` hasta `onPause()`, la actividad no solo es visible, sino que además, va a tener el foco de la acción y el usuario va a poder interactuar con ella.

Y por último, tenemos que saber que el proceso que mantiene a esta Activity puede ser eliminado únicamente cuando se encuentra en `onPause()` o en `onStop()`, es decir, cuando no tiene el foco de la aplicación. Android nunca eliminará procesos con los que el usuario está interactuando en ese momento. Si se ha eliminado el proceso, el usuario no tiene por qué saber de dicha situación e incluso podría volver hacia atrás y querer usarlo de nuevo. Entonces el proceso se restaura gracias a una copia y vuelve a estar activo como si no hubiera sido eliminado. Además, la Activity puede haber estado en segundo plano, invisible, y entonces es despertada pasando por el estado `onRestart()`.



La idea importante con la que quedarse es que una Activity que esté pausada o detenida (tras `onPause()` u `onStop()`) puede ser destruida por el sistema sin previo aviso, por lo que deberemos encargarnos de guardar antes la información necesaria (durante `onFreeze()` y `onPause()`).

Pero, ¿qué ocurre en realidad cuando no existen recursos suficientes? Está claro que los recursos son siempre limitados y dependen, además, de cada dispositivo. En el momento en el que Android detecta que no hay los recursos necesarios, éste analizará los procesos existentes en ese momento y eliminará los que sean menos prioritarios para poder liberar sus recursos.

Para los procesos en segundo plano, existe una lista llamada LRU (*Least Recently Used*). En esta lista se marca el proceso que va a ser eliminado, dependiendo del tiempo que está sin utilizarse. Los primeros que se eliminan, son aquellos que llevan más tiempo sin usarse. Así el sistema se asegura de mantener siempre vivos los últimos procesos que han sido usados.

Si el usuario regresa a una actividad que estaba dormida, el sistema simplemente la despierta. En este caso, no es necesario recuperar el estado guardado porque el proceso todavía existe y mantiene el mismo estado. Sin embargo, cuando el usuario quiere regresar a una aplicación cuyo proceso ya no existe porque ha sido eliminado por Android, ya que se necesitaba liberar sus recursos, Android lo crea de nuevo y utiliza el estado previamente guardado para poder restaurar una copia fresca del mismo. Como ya se ha explicado anteriormente, el usuario no percibe esta situación ni conoce si el proceso ha sido eliminado o está dormido.

1.4.1.3. INTENT

La fuerza de Android radica en la forma que tiene de traducir la Web en aplicaciones móviles. Es la base de su funcionamiento y de la forma de interactuar el usuario con el dispositivo móvil. Hablando a grandes rasgos, el poder de internet radica en que “*todo se encuentra a un clic de distancia*”.

Estos click son los URL y los URI, que son los Localizadores e Identificadores Uniformes de Recursos, respectivamente. La relación de los URI con Intent es la forma de navegar de un usuario, y es lo fundamental para su éxito comercial. Los menús sencillos, un número bajo de pulsaciones, en definitiva, una facilidad de uso. En este momento es donde presentamos el concepto de Intent e IntentFilters, que es la forma de navegación y ejecución de Android.

Intent



Intent es un mecanismo para describir una acción específica, una declaración de necesidades. Son fragmentos de información que describen la acción o servicio deseado. Dicho de otro modo, es una clase que permite especificar una Activity a ejecutar, llamando a uno de los métodos de la clase Activity con ese Intent de parámetro.

IntentFilter



IntentFilter es una declaración de capacidad e interés, soluciones o asistencia al que la necesita. Puede ser genérico o específico, respecto al elemento Intent al que presta servicio.

En android casi todo funciona por medio de intenciones, como “*sacar una foto*”, “*llamar a Paco*”...

Existen dos formas de realizar las llamadas a un Activity, de una forma explícita o implícita.

La forma explícita es la más fácil de entender. Primero creamos un Intent con el nombre de la clase de la actividad y el paquete, después llamamos a startActivity (o startSubActivity si, además, queremos que se nos notifique cuándo finaliza la actividad) y listo. Esto lo veremos más adelante. El sistema lo que hará, será buscar la clase y crear la instancia, pasándole los datos necesarios que hemos añadido al Intent en el objeto Bundle del método onCreate() de la nueva instancia.

La clase `Bundle` contiene tipos y objetos de otras clases. Con esta clase podemos pasar datos entre distintas `activities`. Más adelante veremos un ejemplo de cómo realizar este intercambio de información entre `activities`.

Para verlo mejor pongamos un ejemplo. `ClassActivity` es la clase de la actividad que queremos iniciar. `this` nos indica el `Context` actual, para saber en qué package tendrá que buscar esta clase:

```
Intent i = new Intent(this, ClassActivity.class);
```

A partir de aquí, esta información la recuperamos en el objeto `Bundle` de `onCreate()`:

```
i.putExtra("NombreParametro", valorParametro);  
startActivity(i);
```

La llamada implícita de una actividad la realizamos también con la clase `Intent`. Pero es implícita porque no indicamos el nombre de la clase correspondiente a la actividad a llamar, sino que establecemos una serie de criterios, y dejamos que el sistema sea el que elija la actividad que cumpla con esos criterios.

A un `Intent` se le pueden asociar acciones, datos y categorías. La `Activity` declarará el tipo de acción que puede llevar a cabo y el tipo de datos que va a gestionar. Las acciones serán cadenas de texto estándar que describirán lo que la `Activity` puede hacer. Por poner un ejemplo:

`android.intent.action.EDIT`

Es una acción que indica que la `activity` puede editar datos. Esta acción viene de la clase `Intent`, pero es posible definir nuevas acciones para nuestras `Activities`. La misma actividad puede declarar, que el tipo de datos del que se ocupa es, por ejemplo, "com.android.cursor.dir/option". También puede declarar una categoría, que nos indicará desde donde se lanzará la `Activity`, que posiblemente sea desde el lanzador de aplicaciones, el menú de otra aplicación o desde otra `Activity`. En el `AndroidManifest.xml` de la aplicación lo definiríamos de la siguiente manera:

```
<activity android:name= ".HolaSeas" android:label= "@string/app_name">  
    <intent-filter>  
        <action android:name= "android.intent.action.VIEW" />  
        <category android:name= "android.intent.category.DEFAULT" />  
        <data android:mimeType= "vnd.android.cursor.dir/option" />  
    </intent-filter>  
</activity>
```

Para realizar una llamada implícita a una Activity a través de un Intent, en vez de asignar el nombre de la clase, debemos asignar alguna de las acciones que ésta puede llevar a cabo, contando con el tipo de datos adecuado.

Activity e Intent son los dos grandes pilares sobre los que se basa la arquitectura de las aplicaciones Android.

1.4.1.4. SERVICES

Llamamos Service en Android a las tareas que se están ejecutando en segundo plano, sin que el usuario esté interactuando con él. Un ejemplo claro sería el reproductor de música. La reproducción la puede iniciar una actividad, y lo normal es que esa reproducción siga aun cuando el usuario abra otro programa diferente. Por lo tanto, el código que controla la reproducción debe estar en un Service. Otro ejemplo sería una utilidad de sincronización de datos de fondo que se esté ejecutando de forma continuada, también debería implementarse como Service.

Veamos un pequeño ejemplo de un servicio que arranca un contador y cada segundo va sumando una unidad. Lo insertaremos dentro de un servicio, para que cuando salgamos de la aplicación, siga contando en segundo plano. Y de nuevo al arrancar la aplicación el contador siga en marcha. Lo primero creamos un nuevo proyecto de Android, tal y como hemos descrito anteriormente. Y en el `AndroidManifest.xml` insertamos el siguiente código:

```
<?xml version= "1.0" encoding= "utf-8"?>
<manifest xmlns:android=
"http://schemas.android.com/apk/res/android"
    package= "com.seas.ejemploservice"
    android:versionCode= "1"
    android:versionName= "1.0">
<uses-sdk android:minSdkVersion= "3" />

    <application android:icon= "@drawable/icon" android:label=
"@string/app_name">
        <activity android:name= ".SeassService"
                  android:label= "@string/app_name">
            <intent-filter>
                <action android:name=
"android.intent.action.MAIN" />
                <category android:name=
"android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <service android:name= "MiServicio"/>
    </application>
</manifest>
```

Creamos la clase MiServicio.java, y añadimos el siguiente código:

```
package com.seas.ejemploservice;

import java.util.Timer;
import java.util.TimerTask;

import android.app.Service;
import android.content.Intent;
import android.os.Handler;
import android.os.IBinder;
import android.os.Message;

public class MiServicio extends Service {

    private Timer timer = new Timer();
    private static final long UPDATE_INTERVAL = 1000;
    public static ActualizarServicioIUListener
UI_UPDATE_LISTENER;

    private int count = 0;

    public static void
setUpdateListener(ActualizarServicioIUListener l) {
        UI_UPDATE_LISTENER = l;
    }

    @Override
    public IBinder onBind(Intent arg0) {
        return null;
    }

    @Override
    public void onCreate() {
        super.onCreate();
        _startServicio();
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        _shutdownServicio();
    }

    private void _startServicio() {
        timer.scheduleAtFixedRate(
            new TimerTask() {
                public void run() {
                    count++;
                    handler.sendMessage(0);
                }
            },
            0,
```

```
        UPDATE_INTERVAL);
    }

    private void _shutdownServicio() {
        if (timer != null) timer.cancel();
    }

    private Handler handler = new Handler() {
        @Override
        public void handleMessage(Message msg) {
            UI_UPDATE_LISTENER.actualizar(count);
        }
    };
}

}
```

Ahora debemos crear una nueva clase de java con el nombre SeassService.java. En ella insertamos el siguiente código:

```
package com.seas.ejemploservice;

import java.util.Timer;
import java.util.TimerTask;

import android.app.Service;
import android.content.Intent;
import android.os.Handler;
import android.os.IBinder;
import android.os.Message;

public class MiServicio extends Service {

    private Timer timer = new Timer();
    private static final long UPDATE_INTERVAL = 1000;
    public static ActualizarServicioIUListener
    UI_UPDATE_LISTENER;

    private int count = 0;

    public static void
    setUpdateListener(ActualizarServicioIUListener l) {
        UI_UPDATE_LISTENER = l;
    }

    @Override
    public IBinder onBind(Intent arg0) {
        return null;
    }

    @Override
    public void onCreate() {
```

```
        super.onCreate();
        _startServicio();
    }

@Override
public void onDestroy() {
    super.onDestroy();
    _shutdownServicio();
}

private void _startServicio() {
    timer.scheduleAtFixedRate(
        new TimerTask() {
            public void run() {
                count++;
                handler.sendMessage(0);
            }
        },
        0,
        UPDATE_INTERVAL);
}

private void _shutdownServicio() {
    if (timer != null) timer.cancel();
}

private Handler handler = new Handler() {
    @Override
    public void handleMessage(Message msg) {
        UI_UPDATE_LISTENER.actualizar(count);
    }
};

}
```

Hasta ahora hemos visto como crear un servicio y como lo conectamos a los elementos de la IU del proyecto. Ahora creamos la clase ActualizarServicioIUListener.java, e insertamos el siguiente código:

```
package com.seas.ejemploservice;

public interface ActualizarServicioIUListener {
    public void actualizar(int count);
}
```

En el directorio res/layout abrimos el archivo main.xml, el archivo en el que se ve el diseño de la página principal... (más adelante veremos un capítulo entero a su modelado) e insertamos lo siguiente:

```
<?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= "Contador:"
        android:layout_marginTop= "20px"
        android:layout_marginBottom= "20px"
        android:id= "@+id/count"
        />
    <Button android:text= "Empezar servicio" android:id=
"@+id/start" android:layout_width= "wrap_content"
    android:layout_height= "wrap_content"></Button>
    <Button android:text= "Acabar servicio" android:id=
"@+id/stop" android:layout_width= "wrap_content"
    android:layout_height= "wrap_content"></Button>
</LinearLayout>
```

Una vez insertado este código, vemos que la pantalla principal quedaría de esta forma:

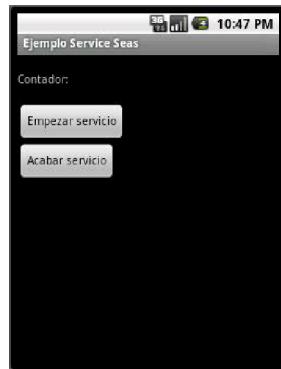


Figura 1.53. Pantalla principal.

Para probar la aplicación, la ejecutamos en el emulador o en el dispositivo móvil. Pulsamos sobre el botón “*Empezar servicio*”. Salimos al menú general con la tecla atrás, y desde el menú de aplicaciones pulsamos en el ícono llamado “*Ejemplo Service Seas*”, y al volver a arrancarla, vemos que el contador sigue en marcha y sumando.



El ejemplo completo “**SeasService**” se puede descargar desde la plataforma de estudio.

1.4.1.5. BROADCASTRECEIVER

Un BroadcastReceiver es el componente que está destinado a recibir y responder ante eventos globales generados por el sistema, como por ejemplo: “*Batería baja*”, “*Llamada Entrante*”, “*SMS recibido*”, “*Tarjeta SD insertada*” o por otras aplicaciones. Para registrar un BroadcastReceiver existen dos técnicas:

- Implementar un elemento <receiver> en el archivo AndroidManifest.xml, donde describimos el nombre de clase BroadcastReceiver y enumeramos sus IntentFilter.
- Realizarlo en tiempo de ejecución mediante el método registerReceiver de la clase Context.

Igual que ocurre con Service, BroadcastReceiver no tiene IU. Y algo muy importante a tener en cuenta, es que el código ejecutado en el onReceive() del BroadcastReceiver no debería asumir operaciones de persistencia o de ejecución prolongada.

Veamos un pequeño ejemplo de cómo funciona esta clase. Lo que vamos a hacer es interceptar eventos en Android, y en este caso haremos que el sistema capture el texto de un SMS y el emisor de dicho mensaje SMS. Antes de comenzar a programar debemos configurar nuestra aplicación para que pueda suscribirse a este tipo de eventos. Así que lo primero será crear un nuevo proyecto, que en nuestro caso lo llamaremos SeasBroadcastReceiver. Una vez creado debemos modificar el AndroidManifest.xml:

```
<?xml version= "1.0" encoding= "utf-8"?>
<manifest xmlns:android=
"http://schemas.android.com/apk/res/android"
    package= "com.seas.ejemplo.broadcastReaceiver"
    android:versionCode= "1"
    android:versionName= "1.0">
    <uses-sdk android:minSdkVersion= "3" />

    <application android:icon= "@drawable/icon"
        android:label= "@string/app_name">
        <activity android:name= ".BroadcastReceiver"
            android:label= "@string/app_name">
            <intent-filter>
                <action android:name=
"android.intent.action.MAIN" />
```

```

        <category android:name=
"android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
<receiver android:name= "RecibirSMSEntrante">
    <intent-filter>
        <action android:name=
"android.provider.Telephony.SMS_RECEIVED" />
    </intent-filter>
</receiver>

</application>

<uses-permission android:name=
"android.permission.RECEIVE_SMS" />

</manifest>
```

Lo que hemos hecho hasta ahora es dar permisos a la aplicación para que pueda interceptar los mensajes mediante el `uses-permission`, e implementar la clase `RecibirSMSEntrante` que va a heredar de `BroadcastReceiver`. Ahora creamos la clase y le insertamos el siguiente código:

```

package com.seas.ejemplo.BroadcastReceiver;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.telephony.gsm.SmsMessage;
import android.util.Log;
public class RecibirSMSEntrante extends BroadcastReceiver {
    private final static String LOGTAG = "RecibirSMSEntrante";
    @Override
    public void onReceive(Context context, Intent intent) {
        Log.d(LOGTAG, "SMS recibido");
        Bundle bundle = intent.getExtras();
        if (bundle != null) {
            Object[] pdus = (Object[]) bundle.get("pdus");
            final SmsMessage[] messages = new SmsMessage[pdus.length];
            for (int i = 0; i < pdus.length; i++) {
                messages[i] = SmsMessage.createFromPdu((byte[]) pdus[i]);

                String idMensaje = messages[i].getOriginatingAddress();
                String textoMensaje = messages[i].getMessageBody();

                Log.d(LOGTAG, "Mensaje recibido: id="+idMensaje+
texto+"+textoMensaje);
            }
        }
    }
}
```

Para ejecutar esta aplicación lo vamos a realizar de una manera diferente. Para poder simular el envío de un SMS debemos usar el perfil DDMS de nuestro eclipse. Para ello debemos añadir el perfil DDMS.

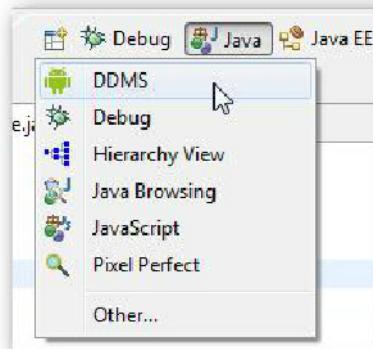


Figura 1.54. Añadir perfil DDMS.

Una vez que lo tenemos añadido, pulsamos sobre él y en la columna de la izquierda tenemos la opción de emular un SMS en "Telephony Actions":

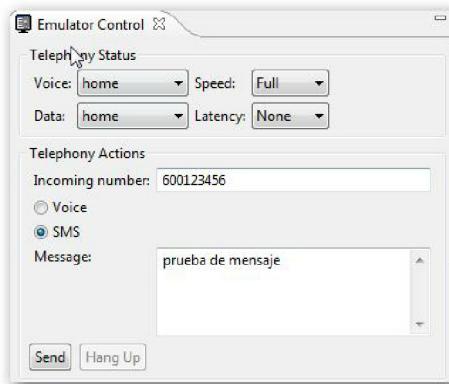


Figura 1.55. Emulador de envío de SMS o de Llamada recibida.

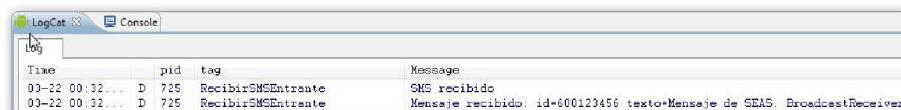


Figura 1.56. LogCat del DDMS.

Podemos observar en la imagen superior, como el BroadcastReceiver ha recogido la ID y el texto del mensaje. Como anteriormente hemos comentado, al igual que Service, no dispone de IU, por lo que si lo intentamos ejecutar como una aplicación en el emulador pasa lo siguiente:



Figura 1.57. Error al intentar ejecutar el BroadcastReceiver.



El ejemplo completo “**SeasBroadcastReceiver**” se puede descargar desde la plataforma de estudio.

1.4.1.6. CONTENT PROVIDER

Un content provider es la forma de compartir datos entre aplicaciones que se han definido en Android. De esta manera, es posible compartir determinados datos de nuestra aplicación sin tener que mostrar detalles sobre su almacenamiento interno, su estructura o su implementación. De la misma forma, nuestra aplicación podrá tener acceso a los datos de otras aplicaciones usando los content provider que se hayan definido.

Normalmente, ContentProvider se usa con una base de datos (*SQLite*), pero también puede usarse con otro medio para compartir variables, un archivo o datos almacenados en el dispositivo.

Android ya incluye una serie de componentes ContentProvider que dejan publicar cualquier tipo de dato básico que resulta útil entre aplicaciones, como la información de los contactos, la fotografías almacenadas en la SD, vídeos, los mensajes de texto (SMS), archivos de audio, etc. Todos estos ContentProvider ya definidos y listos para utilizar se pueden encontrar en el paquete android.provider. Si se quiere consultar todas las clases de datos que Android provee, se puede consultar la guía que Android nos facilita en:

<http://developer.android.com/reference/android/provider/package-summary.html>



Debemos tener claras dos cosas, que si se almacenan los datos en una base de datos, dichos datos solo serán accesibles desde la aplicación que los creó, por lo tanto, no podrán ser consultados por otras aplicaciones. Así que, si se tiene que compartir datos, se deberá usar el modelo de ContentProvider que recomienda Android.

Los almacenes de datos que Android proporciona, y que más podemos usar son los siguientes:

- **Browser** (Navegador). Favoritos, historial, etc.
- **CallLog** (Registro llamadas). Llamadas perdidas, recibidas, detalles.
- **Contacts** (Contactos). Detalle de los contactos.
- **MediaStore** (Multimedia). Archivos de audio, video e imágenes.
- **Settings** (Ajustes). Ajustes del dispositivo y preferencias.

Para realizar una consulta a un ContentProvider se realiza mediante un string en forma de URI que tendrá la sintaxis siguiente:

<standard_prefix>://<authority>/<data_path>/<id>

Por ejemplo:

content://contacts/people. Recuperaría todos los contactos guardados en la aplicación “Contactos”.

content://contacts/people/12. Así recuperaríamos un contacto determinado (id=12).

content://browser/bookmarks. Con esta se recuperan los marcadores almacenados en la aplicación Browser.

Veamos un pequeño ejemplo de cómo implementar la clase `ContentProvider` en una simple aplicación que nos muestra las llamadas realizadas, recibidas y perdidas en el *LogCat* de la vista DDMS.

Para ello creamos un nuevo “*Android Project*”, le damos el nombre que deseemos y cuyo *Package Name* le damos el siguiente: com.seas.ejemplo.CPEejemplo y la Activity la llamamos: CPEejemploActivity. Una vez creado, editamos la clase CPEejemploActivity.java e insertamos el siguiente código:

```
package com.seas.ejemplo.CPEejemplo;

import android.app.Activity;
import android.database.Cursor;
import android.net.Uri;
import android.os.Bundle;
import android.provider.CallLog.Calls;
import android.util.Log;

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

        Uri allCalls =
Uri.parse("content://call_log/calls");
        Cursor c = managedQuery(allCalls, null, null, null,
null);
        if (c.moveToFirst()) {
            do {
                String callType = "";
                switch (Integer.parseInt(c.getString(c.getColumnIndex(Calls.TYPE)))) {
                    case 1:
                        callType = "Incoming";
                        break;
                    case 2:
                        callType = "Outgoing";
                        break;
                    case 3:
                        callType = "Missed";
                }
                Log.v("Content Providers",
c.getString(c.getColumnIndex(Calls._ID)) + ", "
+
c.getString(c.getColumnIndex(Calls.NUMBER))
+ ", " + callType);
            } while (c.moveToNext());
        }
    }
}
```

}

De esta manera conseguimos tener acceso al registro de llamadas que ha realizado. Para que la aplicación funcione, debemos obtener el permiso necesario añadiendo en el `AndroidManifest.xml` el permiso `READ_CONTACTS`:

```
<?xml version= "1.0" encoding= "utf-8"?>
<manifest xmlns:android=
"http://schemas.android.com/apk/res/android"
    package= "com.seas.ejemplo.CPEejemplo"
    android:versionCode= "1"
        android:versionName= "1.0.0">
    <application android:icon= "@drawable/icon"
        android:label= "@string/app_name">
        <activity android:name= ".CPEejemploActivity"
        android:label= "@string/app_name">
            <intent-filter>
                <action android:name=
"android.intent.action.MAIN" />
                <category android:name=
"android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
    <uses-permission android:name=
"android.permission.READ_CONTACTS">
    </uses-permission>
</manifest>
```

Para realizar la prueba, desde la perspectiva DDMS, realizaremos llamadas al emulador, probando con números de teléfono ficticios, y desde el emulador, contestamos alguna. Posteriormente, ejecutamos la aplicación y visualizamos los resultados en el `LogCat`:

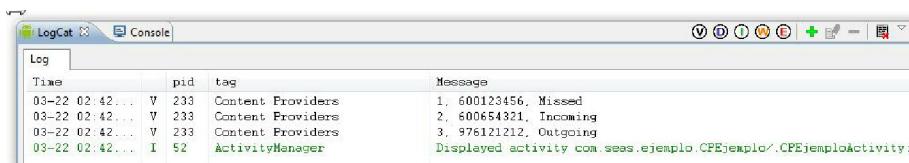


Figura 1.58. LogCat del emulador de Android.

1.4.2. LA INTERFAZ DE USUARIO

Hasta ahora hemos visto que las Activity son como las pantallas que puede tener una aplicación. Por sí solo puede realizar muchas funcionalidades pero no tiene presencia en pantalla. Para diseñar su interfaz de usuario habrá que trabajar con los VIEWS y los grupos de VIEWGROUPS, que son las unidades básicas de la IU en Android.

Aquí es donde entra el concepto de layouts. Los layouts son elementos “no visuales” orientados a controlar la distribución, posición y dimensiones de los controles que se inserten en ellos. Estos extienden a la clase base ViewGroup, y son capaces de contener a otros controles. Veamos dichos controles, que se encuentran en la ruta de nuestro proyecto res/layout/main.xml

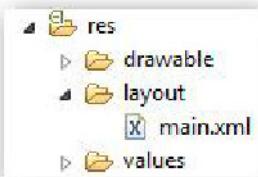


Figura 1.59. Ruta de los layouts.

1.4.2.1. FRAMELAYOUT

Éste es el más simple de todos. Lo que hace es colocar todos sus controles hijos alineados con su esquina superior izquierda, de manera que cada control queda oculto por el control siguiente (salvo que éste último tenga transparencia). Éste suele utilizarse para mostrar un único control en su interior, a modo de contenedor sencillo para un solo elemento como, por ejemplo, una imagen.

Las propiedades de este control son android:layout_width y android:layout_height, y podrán tener dos valores “fill_parent” (el control hijo toma la dimensión de su layout contenedor) o “wrap_content” (el control hijo toma la dimensión de su contenido).

Creamos un nuevo proyecto y en el res/layout/main.xml insertamos el siguiente código:

```
<?xml version= "1.0" encoding= "utf-8"?>
<FrameLayout
    xmlns:android=
    "http://schemas.android.com/apk/res/android"
    android:layout_width= "fill_parent"
    android:layout_height= "fill_parent">
```

```
<EditText android:id= "@+id/TxtNombre"  
    android:layout_width= "fill_parent"  
    android:layout_height= "fill_parent" />  
  
</FrameLayout>
```

Al ejecutar, nos aparece un cuadro de texto que ocupa toda la pantalla en el cual podemos escribir.

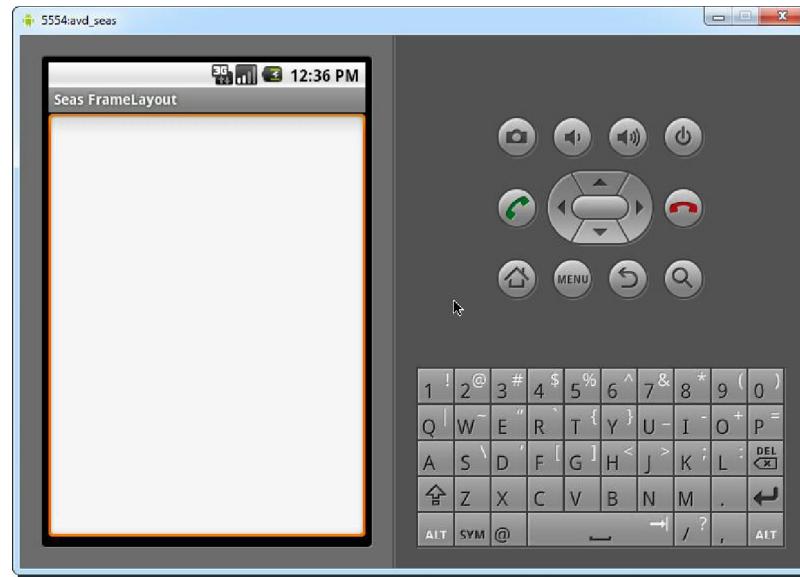


Figura 1.60. Avd_seas – FrameLayout.



El ejemplo completo “**SeasFrameLayout**” se puede descargar desde la plataforma de estudio.

1.4.2.2. LINEARLAYOUT

Este layout apila uno detrás de otro sus elementos hijos de manera horizontal o vertical según se indique en su propiedad android:orientation.

Las propiedades de este layout son android:layout_width, android:layout_height y otro parámetro más para determinar sus dimensiones, la propiedad android:layout_weight, con el que podemos determinar su ancho. Si insertamos el siguiente código en el res/layout/main.xml:

```
<?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">

    <LinearLayout android:layout_width= "fill_parent"
        android:layout_weight= "1" android:layout_height=
"fill_parent"
            android:orientation= "horizontal">
                <TextView android:text= "horiz.1" android:gravity=
"center_horizontal"
                    android:layout_weight= "1"
                android:layout_width= "wrap_content"
                    android:layout_height= "fill_parent" />
                <TextView android:text= "horiz.2" android:gravity=
"center_horizontal"
                    android:layout_width= "wrap_content"
                android:layout_weight= "1"
                    android:layout_height= "fill_parent" />
                <TextView android:layout_height= "fill_parent"
                android:id= "@+id/textView2"
                    android:gravity= "center_horizontal"
                android:layout_weight= "1"
                    android:text= "horiz.3" android:layout_width=
"wrap_content">
                    </TextView>

    </LinearLayout>

    <LinearLayout android:orientation= "vertical"
        android:layout_width= "fill_parent"
    android:layout_height= "fill_parent"
        android:layout_weight= "1">
        <TextView android:layout_height= "wrap_content"
    android:id= "@+id/textView1"
        android:layout_width= "fill_parent"
    android:textSize= "10pt"
        android:layout_weight= "1" android:text= "fila
1"></TextView>
```

```

<TextView android:text= "fila 2" android:textSize=
"10pt"
        android:layout_width= "fill_parent"
        android:layout_height= "wrap_content"
        android:layout_weight= "1" />
<TextView android:text= "fila 3" android:textSize=
10pt"
        android:layout_width= "fill_parent"
        android:layout_height= "wrap_content"
        android:layout_weight= "1" />
</LinearLayout>
```

El resultado será el siguiente:



Figura 1.61. Avd_seas – LinearLayout.

Observamos que con el

```
<LinearLayout android:orientation= "horizontal"
```

Hacemos que los TextView se añadan horizontalmente a la pantalla, y en el segundo LinearLayout que declaramos con su propiedad

```
<LinearLayout android:orientation= "vertical"
```

Las TextView se posicionan verticalmente.



El ejemplo completo “**SeasLinearLayout**” se puede descargar desde la plataforma de estudio.

1.4.2.3. TABLELAYOUT

Un TableLayout permite definir las filas, las columnas y la posición de sus elementos hijos dentro de una tabla.

La estructura es similar a como se realiza en HTML, es decir, indicando las filas que compondrán la tabla y las columnas necesarias, con la diferencia de que en Android no existe ningún objeto para definir una columna, lo que se hace es insertar los controles necesarios dentro del TableRow y cada uno de ellos (ya sea un control sencillo u otro ViewGroup) corresponderá a una columna de la tabla. El número final de filas de la tabla vendrá definido por el número de elementos TableRow insertados, y el número de columnas será el número de componentes de la fila que más componentes tenga.

Lo normal es que el ancho de cada columna se corresponda con el mayor ancho del componente de esa columna, pero para modificar esto existen una serie de propiedades:

`android:stretchColumns`: indica las columnas que se pueden expandir para coger el espacio libre dejado por el resto de columnas a la derecha de la pantalla.

`android:shrinkColumns`: indica las columnas que se pueden contraer para dejar espacio al resto de columnas que puede que se salgan por la derecha de la pantalla.

`android:collapseColumns`: indica las columnas que se quieren ocultar totalmente.

Veamos un par de ejemplos de tabla. En el primero simularemos un menú de opciones desplegables. Para ello creamos un nuevo proyecto y modificamos el `res/layout/main.xml`:

```
<?xml version= "1.0" encoding= "utf-8"?>
<TableLayout xmlns:android=
"http://schemas.android.com/apk/res/android"
    android:layout_width= "fill_parent"
    android:layout_height = "fill_parent"
    android:stretchColumns= "1">

    <TableRow>
```

```
<TextView android:layout_column= "1" android:text=
"Abrir..." android:padding= "3dip" />
<TextView android:text= "Ctrl-A" android:gravity=
"right" android:padding= "3dip" />
</TableRow>

<TableRow>
<TextView android:layout_column= "1" android:text=
"Guardar..." android:padding= "3dip" />
<TextView android:text= "Ctrl-G" android:gravity=
"right" android:padding= "3dip" />
</TableRow>

<TableRow>
<TextView android:layout_column= "1" android:text=
"Guardar como..." android:padding= "3dip" />
<TextView android:text= "Ctrl-Shift-G" android:gravity=
"right" android:padding= "3dip" />
</TableRow>

<View android:layout_height= "2dip" android:background=
"#FF909090" />

<TableRow>
<TextView android:text= "X" android:padding= "3dip"
/>
<TextView android:text = "Importar de..." android:padding= "3dip" />
</TableRow>

<TableRow>
<TextView android:text= "X" android:padding= "3dip"
/>
<TextView android:text= "Exportar a..." android:padding= "3dip" />
<TextView android:text= "Ctrl-E" android:gravity=
"right" android:padding= "3dip" />
</TableRow>

<View android:layout_height= "1dip" android:background=
"#FF909090" />

<TableRow>
<TextView android:layout_column= "1" android:text=
"Salir" android:padding= "3dip" />
```

```
</TableRow>
</TableLayout>
```

Observemos el resultado en el emulador.



Figura 1.62. Avd_seas – TableLayout 1.



El ejemplo completo “**SeasTableLayout**” se puede descargar desde la plataforma de estudio.

Veamos otro ejemplo para entender mejor el funcionamiento de las tablas en Android. Modificamos el `res/layout/main.xml` e insertamos el siguiente código:

```
<?xml version= "1.0" encoding= "utf-8"?>
<TableLayout xmlns:android=
"http://schemas.android.com/apk/res/android"
    android:layout_width= "fill_parent"
    android:layout_height= "fill_parent"
    android:stretchColumns= "1,2,3">

    <TableRow>
        <TextView android:text= "Celda 1.1" />
        <TextView android:text= "Celda 1.2" />
        <TextView android:text= "Celda 1.3" />
    </TableRow>
```

```
<TableRow>
    <TextView android:text= "Celda 2.1" />
    <TextView android:text= "Celda 2.2"
android:layout_span= "2" />
</TableRow>

<TableRow>
    <TextView android:text= "Celda 3.1 y 3.2"
android:layout_span="2" />
    <TextView android:text= "Celda 3.3" />
</TableRow>

</TableLayout>
```

Observamos como podemos cambiar la distribucion de las columnas con la propiedad android:stretchColumns.



Figura 1.63. Avd_seas – TableLayout2.



El ejemplo completo “**SeasTableLayout2**” se puede descargar desde la plataforma de estudio.

1.4.2.4. RELATIVELAYOUT

Con este layout especificamos la posición de cada elemento de una manera relativa a su elemento padre o a cualquier otro incluido en el propio layout. De esta forma, al incluir un nuevo elemento podremos elegir dónde queremos colocarlo, debajo de qué elemento y si lo queremos alineado o no del layout padre. Veámoslo ejecutando un pequeño ejemplo. Agregamos el siguiente código en el res/layout/main.xml:

```
<?xml version= "1.0" encoding= "utf-8"?>
<RelativeLayout xmlns:android=
"http://schemas.android.com/apk/res/android"
    android:layout_width= "fill_parent"
    android:layout_height= "wrap_content"
    android:padding= "10px">
    <TextView android:id= "@+id/label"
        android:layout_width= "fill_parent"
        android:layout_height= "wrap_content"
        android:text= "Ejemplo seas - RelativeLayout" />
    <EditText android:text= ""
        android:layout_height= "wrap_content"
        android:id= "@+id/entry"
        android:background=
"@drawable/editbox_background"
        android:layout_width= "fill_parent"
        android:layout_below= "@+id/label"
        android:layout_alignLeft=
"@+id/label"></EditText>
    <Button android:id= "@+id/uno"
        android:layout_width= "wrap_content"
        android:layout_height= "wrap_content"
        android:text= "1"
        android:layout_below= "@+id/entry"
        android:layout_alignParentRight= "true"/>
    <Button android:id= "@+id/dos"
        android:layout_width= "wrap_content"
        android:layout_height= "wrap_content"
        android:text= "2"
        android:layout_toLeftOf= "@+id/uno"
        android:layout_alignTop= "@+id/uno"/>
    <Button android:id= "@+id/tres"
        android:layout_width= "wrap_content"
        android:layout_height= "wrap_content"
        android:text= "3"
        android:layout_below= "@+id/uno"
        android:layout_alignParentLeft= "true"/>
    <Button android:id= "@+id/cuatro"
        android:layout_width= "wrap_content"
        android:layout_height= "wrap_content"
        android:text= "4"
        android:layout_toRightOf= "@+id/tres"
        android:layout_alignTop= "@+id/tres"/>
```

```

<Button android:id= "@+id/cinco"
        android:layout_width= "wrap_content"
        android:layout_height= "wrap_content"
        android:text= "cinco"
        android:layout_below= "@+id/cuatro"
        android:layout_alignParentRight= "true"/>
<Button android:id= "@+id/seis"
        android:layout_width= "wrap_content"
        android:layout_height= "wrap_content"
        android:text= "6"
        android:layout_toLeftOf= "@+id/cinco"
        android:layout_alignTop= "@+id/cinco"/>
</RelativeLayout>

```

Podemos observar que con las propiedades:

```

android:layout_width= "wrap_content"
android:layout_height= "wrap_content"

```

Definimos el tamaño del alto y el ancho dependiendo del contenido. Observa, por ejemplo, el botón "*cinco*".

Con las propiedades definidas en el botón 4

`android:layout_toRightOf= "@+id/tres"` estamos alineando a derecha del botón `"@+id/tres"`

`android:layout_alignTop= "@+id/tres"` dejamos el botón a la altura de `"@+id/tres"`



Figura 1.64. Avd_seas – RelativeLayout.



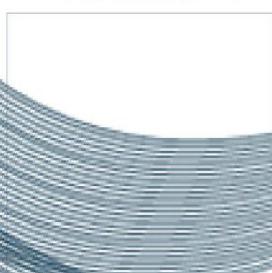
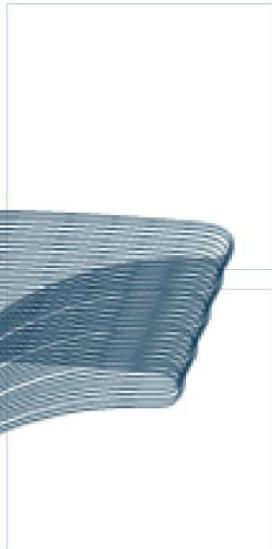
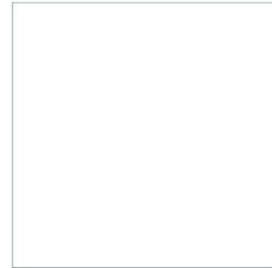
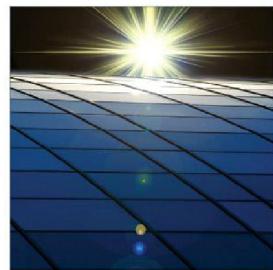
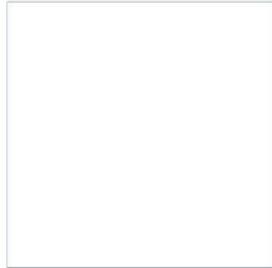
El ejemplo completo “**SeasRelativeLayout**” se puede descargar desde la plataforma de estudio.

RESUMEN

- Hemos conocido como surge Android, como ha ido evolucionando y como se ha ido mejorando a sí mismo según han ido surgiendo las nuevas versiones. Hemos visto que cada versión ha traído toda una serie de herramientas de desarrollo para potenciar el nacimiento de aplicaciones cada vez más completas.
- Hemos aprendido e instalado todo el entorno de desarrollo necesario para poder realizar nuestras aplicaciones en Android.
- Hemos creado un emulador para poder probar el resultado que vayamos desarrollando, hemos instalado el dispositivo físico para usarlo como emulador, hemos visto el entorno de desarrollo y hemos realizado nuestra primera aplicación en Android.
- Hemos visto las partes más importantes de la estructura de una aplicación en Android y cómo funciona cada una de ellas, viendo ejemplos de código para ilustrarlo mejor:
 - La importancia del archivo `AndroidManifest.xml`, donde hemos especificado todos los componentes de la aplicación, así como su configuración global.
 - Hemos visto cómo funcionan las Activities, siendo las que nos proporcionan las ventanas en una aplicación, con las que los usuarios posteriormente interactúan para realizar una acción.
 - Hemos aprendido la importancia de los Intent, que son los mecanismos que usa Android para describir una acción específica, dicho de otra manera, son las clases que permiten especificar una Actividad a ejecutar.
 - Hemos visto que los Services son tareas que se ejecutan en el dispositivo sin la necesidad que el usuario interactúe con ellos.
 - Ya sabemos que los BroadcastReceiver son los componentes que están destinados a recibir y responder ante los eventos globales generados por el sistema, como por ejemplo batería baja, mensaje de texto recibido y la forma de capturarlos mediante un ejemplo.
 - Hemos aprendido que los ContentProvider son los que facilitan que podamos compartir información entre aplicaciones.
- Hemos aprendido a usar en esta primera parte de la Interfaz de Usuario de Android, cómo realizar el diseño de las pantallas y la situación de los diferentes elementos posicionándolos a nuestro gusto. Ya sea de una forma lineal, relativa o mediante tablas.

02

ANDROID



AVANZADO
EN ANDROID

ÍNDICE

◆ ÍNDICE	1
◆ OBJETIVOS	3
◆ INTRODUCCIÓN.....	4
2.1. Gráficos 2D	5
2.1.1. Conceptos básicos	5
2.1.1.1. Color	5
2.1.1.2. Paint.....	6
2.1.1.3. Canvas (lienzo).....	6
2.1.1.4. Path.....	7
2.1.2. Aplicando conceptos	8
2.1.2.1. Pintar una imagen.....	8
2.1.2.2. Pintar imagen 2D	11
2.1.2.3. Posicionando la imagen.....	16
2.1.2.4. Pintando varias imágenes.....	18
2.1.2.5. Moviendo las imágenes	21
2.2. Gráficos 3D	24
2.2.1. Introducción a OpenGL	24
2.2.2. Implementando OpenGL	24
2.3. Multimedia	37
2.3.1. Reproducción de audio	37
2.3.1.1. Formatos compatibles.....	40
2.3.2. Reproducción de video.....	41
2.3.2.1. Formatos compatibles.....	43
2.4. Bases de Datos SQLite	44
2.4.1. Aplicación con SQLite	46
2.4.2. Vinculación de datos	51
2.4.3. Utilizar un ContentProvider	56
◆ RESUMEN.....	65

OBJETIVOS

- Descubriremos el mundo de los gráficos 2D y cómo “dibujar” imágenes en Android. Realizaremos varios ejemplos de cómo usar estas librerías de diseño.
- Nos adentraremos en el mundo del 3D y las librerías de OpenGL.
- Descubriremos, en los momentos que estamos, la importancia de los archivos multimedia, el audio y el video. Veremos ejemplos para reproducir un video o una canción de mp3.
- Por último, aprenderemos lo básico para utilizar bases de datos en nuestros teléfonos Android. Realizaremos una aplicación para demostrar los conocimientos adquiridos.

INTRODUCCIÓN

Hasta ahora hemos visto los conceptos básicos de Android, también hemos visto cómo crear una interfaz de usuario simple, compuesta de algún cuadro de diálogo y de unos botones. Ahora vamos a ver los gráficos, que van a incluir un poco de entretenimiento y mejora estética a nuestras aplicaciones.

Android proporciona una de las bibliotecas gráficas más potentes que existen para dispositivos móviles. Realmente son dos las bibliotecas, una para gráficos bidimensionales y la otra para los tridimensionales.

Para los gráficos 2D usaremos la biblioteca nativa de gráficos `android.graphics` y sus clases más genéricas (`color` y `canvas`) para poder comenzar a dibujar en poco tiempo.

Para los 3D usaremos la biblioteca del estándar OpenGL ES.

2.1. GRÁFICOS 2D

Android nos proporciona una librería nativa de gráficos 2D contenida en su paquete `android.graphics`. Intentaremos realizar una comprensión básica de las clases `Color` y `Canvas`. Si entendemos ambas clases, será suficiente para empezar a dibujar en poco tiempo.

2.1.1. CONCEPTOS BÁSICOS

Para entender cómo funcionan los gráficos en 2D en Android, debemos tener claros unos pocos conceptos básicos que veremos a continuación.

- `Color`.
- `Paint`.
- `Canvas` (lienzo).
- `Path`.

2.1.1.1. COLOR

En Android, los colores están representados con cuatro números, cada uno de ellos para *ARGB*, *Alfa*, *Red*, *Green and Blue*. Cada componente puede tener entre [0-255] posibles valores (8 bits), de esta manera un color podrá estar empaquetado en un entero de 32 bits. Por eficacia, se emplea un entero en lugar de una instancia de la clase.

Los colores rojo, verde y azul no necesitan explicación alguna, pero con el color alfa vamos a dar una explicación más explícita. Alfa realmente no es un color, es una medida de transparencia. Sus valores están entre 0, que sería totalmente transparente, en este caso, el resto de los valores de los colores no importan, y 255, que nos indica que el color es totalmente opaco. Los valores comprendidos entre éstos, servirán para los colores translúcidos o semitransparentes. Éstos permitirán ver algo del contenido que esté en la parte posterior del objeto.

Un color se puede crear de dos formas diferentes. Una es utilizando una de las constantes de la clase `Color`:

```
// constante predifinida
int color = Color.RED;
// manualmente
int color2 = Color.argb(100, 255, 0, 0);
```

Y la otra manera es definirlo en un archivo de recurso XML. Este archivo se encuentra en la ruta `/res/values/strings.xml`. De esta manera, si queremos cambiarlos posteriormente solo tendremos que hacerlo en esta ubicación y estará modificado en todos los sitios en lo que lo hayamos instanciado, permitiendo configurarlo sin tener que entrar al código:

```
<resources>
    <color name="micolor">#FF0000</color>
</resources>
```

Para usarlo en otras partes de la aplicación simplemente debemos instanciarlo:

```
int color = getResources().getColor(R.color.micolor);
```

El método `getResources()` nos devuelve la clase `ResourceManager` de la actividad actual y con `getColor()` buscamos el color dado al ID de recurso.

2.1.1.2. PAINT

La clase `Paint` es otra de las bibliotecas nativas de Android más importantes. Con esta clase abarcaremos el estilo, el color y otra información necesaria para poder dibujar un gráfico como un mapa de bits, una forma geométrica o un texto.

Lo normal es que cuando queramos dibujar algo en pantalla, lo hagamos con colores sólidos. Estos colores los establecemos con el método `Paint.setColor()`.

2.1.1.3. CANVAS (LIENZO)

Esta clase representa la superficie en la cual se dibuja. Con sus métodos se pueden dibujar líneas, rectángulos, círculos y cualquier otro gráfico.

Recordemos que en Android, la pantalla está lanzada por una actividad (`Activity`), la cual contiene una vista (`View`), que es la encargada de almacenar un lienzo (`Canvas`). Se puede dibujar sobre dicho lienzo modificando el método `View.onDraw()`. A `onDraw()` habrá que pasarle como parámetro el lienzo sobre el que dibujar. Veamos un ejemplo de una `Activity` llamada `Grafico`, que contiene una vista llamada `Panel`:

```
package com.seas.ejemplo.Grafico;

import android.app.Activity;
import android.content.Context;
import android.graphics.Canvas;
import android.os.Bundle;
import android.view.View;
```

```

public class Grafico extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(new Panel(this));
    }

    class Panel extends View {
        public Panel(Context context) {
            super(context);
        }

        @Override
        public void onDraw(Canvas canvas) {
            // Aquí introduciremos los comandos de dibujo
        }
    }
}

```

En los próximos capítulos veremos algunos de los comandos a incluir en el método `onDraw()`.

2.1.1.4. PATH

Esta clase contiene una serie de comandos para realizar trazados vectoriales, como líneas, rectángulos y curvas. Veamos cómo sería un ejemplo de un trazado circular:

```

circle = new Path();
circle.addCircle(150, 150, 100, Direction.CW);
// con Direction definimos la dirección que llevará: clockwise
or counter-clockwise
// dirección de las manetas del reloj o en contra

```

Con esto hemos definido un círculo, que estará en la posición $x=150$ e $y=150$, con un radio $r=100px$. Una vez definido vamos a utilizarlo dentro del método `onDraw()` para dibujarlo:

```

public void onDraw(Canvas canvas) {
    canvas.drawPath(circle, cPaint);
    // le pasamos la clase paint (cPaint) previamente instanciada
}

```

2.1.2. APLICANDO CONCEPTOS

Vamos a ver unos pequeños ejemplos de código para ver cómo implementar estos conceptos básicos vistos hasta ahora.

- Pintar una imagen.
- Pintar imagen 2D.
- Posicionando la imagen.
- Pintando varias imágenes.
- Movimiento de imágenes.

2.1.2.1. PINTAR UNA IMAGEN

Con este ejemplo vamos a ver cómo mostrar una imagen en una vista normal.

Para ello vamos a crear un nuevo proyecto que llamaremos “Imagen” y vamos a llenar los campos de la siguiente manera:

- **Project Name (nombre del proyecto):** SeasMostrarImagen
- **Build Target (objetivo de compilación):** Android 1.6
- **Application name (nombre de la aplicación):** Seas, Imagen 2D
- **Package name (nombre del paquete):** com.seas.ejemplo.Imagen
- **Create Activity (crear actividad):** Imagen
- **Min SDK Version:** 5

```
package com.seas.ejemplo.Imagen;

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

Ahora debemos añadir nuestra vista personalizada. Vamos a crear una nueva clase denominada “Panel” que se extiende desde `View` y reemplazaremos el método `OnDraw(Canvas)` porque lo que queremos es dibujar un mapa de bits por defecto.

```
public class Panel extends View {

    public Panel(Context context) {
        super(context);
    }

    @Override
    public void onDraw(Canvas canvas) {
    }

}
```

Lo primero que tenemos que hacer es conseguir la imagen, que la insertaremos en la carpeta `/res/drawable`, definida en una variable miembro.

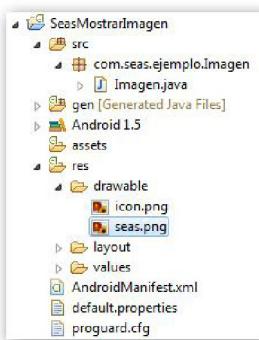


Figura 2.1. Imagen Seas en la carpeta drawable.

Para hacer esto modificamos el constructor.

```
private Bitmap mBitmap;

public Panel(Context context) {
    super(context);
    mBitmap = BitmapFactory.decodeResource(getResources(),
        R.drawable.seas);
}
```

Queremos un fondo negro, así que sacamos el color negro utilizando `drawColor()`. Después de que sacamos nuestro mapa de bits utilizando `drawBitmap()`.

```
@Override
public void onDraw(Canvas canvas) {
    // definimos el fondo negro
    canvas.drawColor(Color.BLACK);
    // imagen
    canvas.drawBitmap(mBitmap, 20, 20, null);
}
```

Ahora tenemos que usar nuestra vista personalizada para que se muestren los objetos definidos, por lo que le debemos cambiar el `setContentView()`. Además, queremos tener una ventana sin título, por lo que añadiremos lo siguiente:

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    //mostramos nuestra ventana sin mostrar ningun titulo
    requestWindowFeature(Window.FEATURE_NO_TITLE);
    setContentView(new Panel(this));
}
```

Iniciamos el emulador y veremos lo siguiente:



Figura 2.2. Emulador con aplicación de SEAS.

Adjuntamos como quedaría la clase completa:

```
package com.seas.ejemplo.Imagen;

import android.app.Activity;
import android.content.Context;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.graphics.Canvas;
import android.graphics.Color;
import android.os.Bundle;
import android.view.View;
import android.view.Window;
```

```

public class Imagen extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //mostramos nuestra ventana sin mostrar ningun titulo
        requestWindowFeature(Window.FEATURE_NO_TITLE);
        setContentView(new Panel(this));
    }

    public class Panel extends View {

        private Bitmap mBitmap;

        public Panel(Context context) {
            super(context);
            mBitmap =
                BitmapFactory.decodeResource(getResources(),
                    R.drawable.seas);
        }

        @Override
        public void onDraw(Canvas canvas) {
            canvas.drawColor(Color.BLACK);
            canvas.drawBitmap(mBitmap, 20, 20, null);
        }
    }
}

```

2.1.2.2. PINTAR IMAGEN 2D

Ahora vamos a usar la clase `SurfaceView`. La ventaja de usar esta clase en vez de `View`, es la facilidad de sacar todo lo que quieras en la pantalla sin tener que trabajar con los diseños y los archivos XML. Es también la mejor manera de hacer animaciones personalizadas y, por supuesto, juegos.

Veamos sus diferencias en la siguiente tabla:

SURFACEVIEW	VIEW
SurfaceView es el tipo de vista caracterizado por contener una superficie (un objeto Surface) sobre la que dibujar	Estructura de datos cuyas propiedades contienen los datos de la capa, la información del área de la pantalla y que permite establecer el layout.
Proporciona una superficie en la que un subproceso se puede ejecutar en la pantalla. Para usarlo necesitamos definir un hilo (Thread)	Maneja la distribución de la pantalla y la interacción con el usuario.
	Contiene los layout, drawing, focus change, scrolling, etc.

Vamos a ver esto realizando un nuevo proyecto. Lo creamos e introducimos los siguientes parámetros:

- **Project Name (nombre del proyecto):** SeasGrafico2D
- **Build Target (objetivo de compilación):** Android 2.1
- **Application name (nombre de la aplicación):** Seas, Imagen 2D
- **Package name (nombre del paquete):** com.seas.ejemplo.seasgrafico2D
- **Create Activity (crear actividad):** SeasGrafico2D
- **Min SDK Version:** 7

Lo primero que debemos hacer es cambiar la clase

```
class Panel extends SurfaceView {  
}
```

Si en este momento compilamos y ejecutamos, lo que veríamos sería... nada, ya que el método `onDraw()` no ha sido llamado todavía. Para poder tener el `SurfaceView` actualizado periódicamente, tenemos que implementar un hilo de la `View` que será el que maneje las llamadas a nuestro método de dibujo, y de esta manera tener más control sobre ellos. Creamos la clase `Panel.java`

Vamos a comenzar con lo básico de nuestro `SurfaceView`. Lo primero será implementar la interfaz `SurfaceHolder.Callback` que nos dará tres métodos que utilizaremos más adelante:

```
public class Panel extends SurfaceView implements  
SurfaceHolder.Callback {  
    @Override  
    public void surfaceChanged(SurfaceHolder holder, int  
format, int width, int height) {  
        // TODO Auto-generated method stub  
    }  
  
    @Override  
    public void surfaceCreated(SurfaceHolder holder) {  
        // TODO Auto-generated method stub  
    }  
  
    @Override  
    public void surfaceDestroyed(SurfaceHolder holder) {  
        // TODO Auto-generated method stub  
    }  
}
```

Para registrar la clase como devolución de llamada, debemos añadir la siguiente línea al constructor de la clase `Panel`.

```
public Panel(Context context) {
```

```

super(context);
mBitmap = BitmapFactory.decodeResource(getResources(),
R.drawable.seas);
getHolder().addCallback(this);
}

```

Nuestro siguiente paso que debemos realizar es crear el hilo que va a controlar el dibujo de nuestra clase Panel.

```

public class ViewThread extends Thread {
    private Panel mPanel;
    private SurfaceHolder mHolder;
    private boolean mRun = false;

    public ViewThread(Panel panel) {
        mPanel = panel;
        mHolder = mPanel.getHolder();
    }

    public void setRunning(boolean run) {
        mRun = run;
    }

    @Override
    public void run() {
        Canvas canvas = null;
        while (mRun) {
            canvas = mHolder.lockCanvas();
            if (canvas != null) {
                mPanel.doDraw(canvas);
                mHolder.unlockCanvasAndPost(canvas);
            }
        }
    }
}

```



Un hilo

Es la parte de código encargada de realizar alguna acción a la misma vez que se está ejecutando otra.

Crear un hilo es muy fácil. Tenemos al constructor que tiene nuestra clase Panel como un parámetro. La variable mHolder se utiliza para evitar la repetición de las llamadas del método getHolder() (de esta manera mejoraremos el rendimiento). La variable lógica mRun nos posibilita poner fin a un bucle infinito que controla el dibujo.

El método run() es muy importante. Tenemos un bucle while con el que bloqueamos el Canvas.



El método `run()` básicamente trabaja como la sincronización vertical, mientras no tiene todos los datos a pintar, no lo actualiza.

Habrás podido observar que hemos cambiando el nombre del método a `doDraw()` en vez de `OnDraw()`. Y esto es muy importante, ya que ahora sólo podemos controlar el dibujo. Y, si alguna vez Android quiere invalidar nuestro panel, esto llamará al `doDraw()` que no hace nada.

Ahora tenemos que manejar los hilos de nuestra clase `Panel`. Para esto, los nuevos métodos de la interfaz son perfectos. El método `surfaceCreated()` es llamado cuando el `SurfaceView` está listo para ser utilizado. En este punto es donde comienza nuestro hilo. También es donde tenemos que comprobar, si el hilo sigue vivo, como cuando se va a la pantalla de inicio y vuelve de nuevo.

```
private ViewThread mThread;

public Panel(Context context) {
    super(context);
    mBitmap = BitmapFactory.decodeResource(getResources(),
R.drawable.icon);
    getHolder().addCallback(this);
    mThread = new ViewThread(this);
}

// se ejecuta cuando SurfaceView está listo para ser utilizado
public void surfaceCreated(SurfaceHolder holder) {
    // si no está vivo
    if (!mThread.isAlive()) {
        mThread = new ViewThread(this);
        mThread.setRunning(true);
        // lanzo el hilo
        mThread.start();
    }
}

public void surfaceDestroyed(SurfaceHolder holder) {
    if (mThread.isAlive()) {
        mThread.setRunning(false);
    }
}
```

En primer lugar, tenemos que crear una variable hilo que sostenga nuestro hilo. Dentro del constructor instanciamos el hilo. En el `surfaceCreated()` comprobamos si el hilo está vivo. Si está vivo, no hacemos nada. Si no lo está, creamos una nueva instancia e iniciamos de nuevo el hilo. No podemos volver a utilizar el antiguo objeto hilo, porque en este punto no sabemos a ciencia cierta si el hilo se está ejecutando aún y una instancia de hilo no se puede iniciar más de una vez.

El método `surfaceDestroyed()` es llamado cuando se destruye la vista. Una razón puede ser que la actividad esté corriendo en un segundo plano, ya sea porque haya comenzado otra Activity o simplemente se haya presionado el botón "Atrás" o el de "*Home*". Hay que asegurarse de que el hilo está terminando para detener el bucle. Para ello debemos establecer la variable `mRun` con el método `setRunning()` a falso.

```
public void surfaceDestroyed(SurfaceHolder holder) {  
    if (mThread.isAlive()) {  
        mThread.setRunning(false);  
    }  
}
```

La pantalla muestra el mismo resultado que lo realizado en capítulo anterior, pero ahora realmente hemos usado gráficos 2D.



Figura 2.3. Resultado mostrado en el emulador.



La aplicación completa “**SeasGrafico2D**” se puede descargar desde la plataforma de estudio.

2.1.2.3. POSICIONANDO LA IMAGEN

Vamos a avanzar un poquito, y vamos a ver en este capítulo cómo posicionar la imagen anterior en la pantalla, dependiendo de la zona pulsada en la pantalla.

Siguiendo con el código del ejemplo anterior, en primer lugar, necesitamos definir dos variables miembro para almacenar las posiciones cuando realizamos el toque en la pantalla. Las llamaremos mX y mY que serán de tipo entero.

```
private int mX;  
private int mY;
```

Para poder ver la imagen en estas coordenadas, cambiaremos la llamada al `drawBitmap()` que se encuentra en el método `doDraw()` y reemplazaremos el 20, 20 con nuestras variables (mX y mY).

```
public void doDraw(Canvas canvas) {  
    canvas.drawColor(Color.BLACK);  
    canvas.drawBitmap(mBitmap, mX, mY, null);  
}
```

Ahora debemos establecer las variables con los valores correctos. A la información sobre el evento táctil se puede acceder con el método `onTouchEvent()`, que será llamado cada vez que se toque sobre la pantalla. Cada vista tiene este método y como hereda de la `SurfaceView`, simplemente debemos anularlo. Así podemos reaccionar con él de la manera que queramos.

Como en nuestro caso queremos obtener la información de los puntos de la pantalla que hemos pulsado, usaremos los métodos `getX()` y `getY()` para capturar los eventos y almacenarlos en nuestras variables anteriormente definidas. Debemos convertirlos explícitamente a `int`, porque devuelven valor `float`, y en el momento en el que los convertimos podríamos perder algo de precisión, es necesario realizar el Casting de forma explícita.

Finalmente, añadimos este método en nuestra clase `Panel`:

```
@Override  
public boolean onTouchEvent(MotionEvent event) {  
    mX = (int) event.getX();  
    mY = (int) event.getY();  
    return super.onTouchEvent(event);  
}
```

Ya podemos ejecutar nuestra aplicación. Podemos observar que cuando pulsas en un punto de la pantalla, la imagen se coloca allí desde la esquina superior izquierda de la imagen, y eso ocurre por el sistema de coordenadas del dispositivo, ya que el punto 0,0 se encuentra en la parte superior izquierda.

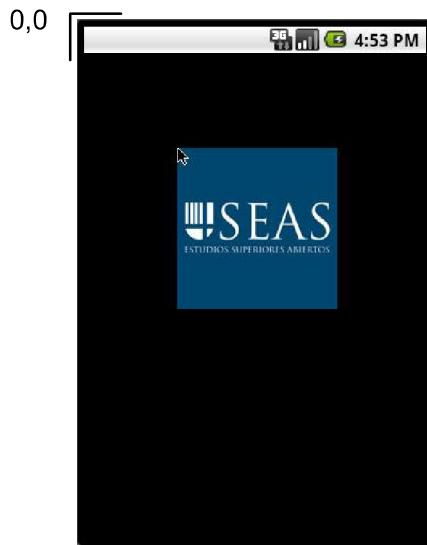


Figura 2.4. Emulador con indicador de coordenadas de pantalla.

Si queremos que el centro de la imagen este en el punto que toquemos sobre la pantalla, deberemos restar la mitad de la anchura de la imagen de la coordenada X y la mitad de la altura de la coordenada Y. Debemos añadir lo siguiente en el método `onTouchEvent()` :

```
mX = (int) event.getX() - mBitmap.getWidth() / 2;  
mY = (int) event.getY() - mBitmap.getHeight() / 2;
```

Al ejecutarlo de nuevo, la imagen aparece centrada sobre la posición pulsada en la pantalla:



Figura 2.5. Resultado de la ejecución.

2.1.2.4. PINTANDO VARIAS IMÁGENES

En este capítulo vamos a aprender a agregar más imágenes en la vista.

Lo primero que debemos hacer es encapsular la imagen en una clase separada. Vamos a nombrar a esta clase `Element`. Esta clase contendrá las coordenadas del dibujo, la imagen y su propio método de dibujo.

```
package com.seas.ejemplo.seasgrafico2D;

import android.content.res.Resources;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.graphics.Canvas;

public class Element {
    private int mX;
    private int mY;

    private Bitmap mBitmap;

    public Element(Resources res, int x, int y) {
        mBitmap = BitmapFactory.decodeResource(res,
R.drawable.icon);
        mX = x - mBitmap.getWidth() / 2;
        mY = y - mBitmap.getHeight() / 2;
    }

    public void doDraw(Canvas canvas) {
        canvas.drawBitmap(mBitmap, mX, mY, null);
    }
}
```

Una vez creada deberemos modificar la clase `Panel`. Tenemos que crear un `ArrayList` que contendrá todos los elementos añadidos.



ArrayList

Es un objeto lista que permite contener y ordenar objetos, incluso objetos duplicados. Tiene un tamaño dinámico que crecerá según se vayan insertando más elementos. Recordar que el índice de un `ArrayList` empieza en 0.

También necesitamos recorrer ese `ArrayList` para sacar todos sus elementos, y por supuesto, tenemos que añadir un nuevo elemento en el método `onTouchEvent()`.

```
private ArrayList<Element> mElements = new
ArrayList<Element>();

public Panel(Context context) {
    super(context);
```

```

        getHolder().addCallback(this);
        mThread = new ViewThread(this);
    }

    public void doDraw(Canvas canvas) {
        canvas.drawColor(Color.BLACK);
        for (Element element : mElements) {
            element.doDraw(canvas);
        }
    }

    @Override
    public boolean onTouchEvent(MotionEvent event) {
        mElements.add(new Element(getResources(), (int)
event.getX(), (int) event.getY()));
        return super.onTouchEvent(event);
    }
}

```

Si en este punto ejecutamos la aplicación, podemos observar que funciona, pero que se va a bloquear y dar un cierre forzado (FC) al azar. Esto va a ser debido a una `ConcurrentModificationException`, que se produce en el método `doDraw()`. La razón tiene una sencilla explicación: mientras la clase `ViewThread` llama al método `doDraw()` y se ejecuta en el bucle, es posible cambiar la lista con la que itera, añadiendo un nuevo elemento a la lista. Para evitarlo, podemos usar la función `synchronized()`, que impide una sincronización de escritura y el acceso de lectura al objeto dado. Así que la llamada `add()` espera hasta que el dibujo está acabado y el objeto `mElements` está libre para poder modificarlo.

Añadiremos el siguiente código en la clase `Panel`:

```

public void doDraw(Canvas canvas) {
    canvas.drawColor(Color.BLACK);
    synchronized (mElements) {
        for (Element element : mElements) {
            element.doDraw(canvas);
        }
    }
}

@Override
public boolean onTouchEvent(MotionEvent event) {
    synchronized (mElements) {
        mElements.add(new Element(getResources(), (int)
event.getX(), (int) event.getY()));
    }
    return super.onTouchEvent(event);
}

```



Synchronized lleva entre paréntesis la referencia a un objeto. Cada vez que un thread quiere acceder a un bloque sincronizado, le pregunta si no hay otro thread ejecutando algún bloque sincronizado con ese objeto. Si es así, el thread actual se suspende y se pone en espera hasta que el objeto bloqueado se libere. Si está libre, el thread actual bloquea el objeto y entra a ejecutar el bloque. Cuando el próximo thread intente ejecutar un bloque sincronizado con ese objeto, se pondrá en espera. El Thread se libera cuando el que lo tiene tomado sale del bloque por cualquier razón. En ese momento termina la ejecución del bloque, ejecuta un return o lanza una excepción.

Una vez añadido el anterior código, ya podemos ejecutar y probar tantas pulsaciones en la pantalla como deseemos.



Figura 2.6. Emulador mostrando resultado de la aplicación.



La aplicación completa “**SeasGrafico2D**” se puede descargar desde la plataforma de estudio.

2.1.2.5. MOVIENDO LAS IMÁGENES

En este capítulo vamos a “animar” las imágenes en la pantalla. Lo que haremos será tocar la pantalla y veremos las imágenes en movimiento por ella. De momento la dirección y la velocidad las dejaremos al azar. Partiremos del proyecto anterior. Y realizaremos las siguientes modificaciones.

En nuestra clase `Element`, que era la que dibujaba cada imagen en la pantalla, debemos declarar dos nuevas variables que serán las que definan la velocidad y la dirección con las que se moverá por la pantalla nuestra imagen. Asimismo, deberemos modificar el constructor, llamar a la clase de java `Random()`, que será la que dará las posiciones aleatorias.

```
private int mSpeedX;
private int mSpeedY;

public Element(Resources res, int x, int y) {
    Random rand = new Random();
    mBitmap = BitmapFactory.decodeResource(res,
R.drawable.seas);
    mX = x - mBitmap.getWidth() / 2;
    mY = y - mBitmap.getHeight() / 2;
    mSpeedX = rand.nextInt(7) - 3;
    mSpeedY = rand.nextInt(7) - 3;
}
```

Ahora creamos el método `animate()` para animar la imagen. En nuestro caso solo cambiaremos la posición de la imagen, que viene dada por la velocidad y el tiempo transcurrido desde la última vez que se le llamó.

```
public void animate(long elapsedTime) {
    mX += mSpeedX * (elapsedTime / 20f);
    mY += mSpeedY * (elapsedTime / 20f);
    checkBorders();
}
```

Y el método `checkBorders()` que comprueba si el elemento se encuentra dentro de la pantalla. Si no es el caso, se cambia la velocidad en este eje y se ajusta la posición hacia el punto más lejano todavía visible en el borde. Esto evitará que las imágenes se muevan fuera de la vista de la pantalla.

```
private void checkBorders() {
    if (mX <= 0) {
        mSpeedX = -mSpeedX;
        mX = 0;
    } else if (mX + mBitmap.getWidth() >= Panel.mWidth) {
        mSpeedX = -mSpeedX;
        mX = Panel.mWidth - mBitmap.getWidth();
    }
    if (mY <= 0) {
        mY = 0;
        mSpeedY = -mSpeedY;
    }
}
```

```

        if (mY + mBitmap.getHeight() >= Panel.mHeight) {
            mSpeedY = -mSpeedY;
            mY = Panel.mHeight - mBitmap.getHeight();
        }
    }
}

```

Ahora en la clase Panel, debemos realizar unos ligeros cambios en el método `surfaceChanged()`.

```

public static float mWidth;
public static float mHeight;

public void surfaceChanged(SurfaceHolder holder, int format, int
width, int height) {
    mWidth = width;
    mHeight = height;
}

```

Debemos añadir las variables `mWidth` y `mHeight`, que ya utilizamos en el método `checkBorders()`, y serán las que definan las dimensiones de nuestra `surfaceView`. También debemos modificar el método `doDraw()`:

```

private Paint mPaint = new Paint();

public Panel(Context context) {
    super(context);
    getHolder().addCallback(this);
    mThread = new ViewThread(this);
    mPaint.setColor(Color.WHITE);
}

public void doDraw(Canvas canvas) {
    canvas.drawColor(Color.BLACK);
    synchronized (mElements) {
        for (Element element : mElements) {
            element.doDraw(canvas);
        }
    }
    canvas.drawText("FPS: " + Math.round(1000f / elapsed) + "
Elements: " + mElementNumber, 10, 10, mPaint);
}

```

Ahora se necesitan dos parámetros en lugar de solo el `Canvas`. El primer parámetro será el tiempo transcurrido, y el segundo, el `Canvas` utilizado para dibujar. El tiempo transcurrido se utiliza para calcular el FPS (*Frames por Segundo*).

El `onTouchEvent()` también lo vamos a modificar, porque ahora actualizamos la variable `mElementNumber`, de modo que no tiene que ser llamado cada vez que se genera en el marco del método `doDraw()`

```

private int mElementNumber = 0;

@Override
public boolean onTouchEvent(MotionEvent event) {

```

```

    synchronized (mElements) {
        mElements.add(new Element(getResources(), (int)
            event.getX(), (int) event.getY()));
        mElementNumber = mElements.size();
    }
    return super.onTouchEvent(event);
}

```

Insertamos el método `animate()` en la clase `Panel`, al que llamaremos en cada bucle del hilo, realiza la llamada a cada elemento de la clase `Element`.

```

public void animate(long elapsedTime) {
    synchronized (mElements) {
        for (Element element : mElements) {
            element.animate(elapsedTime);
        }
    }
}

```

Y por último, en la clase `ViewThread`, en el método `run()` calcularemos el tiempo transcurrido y llamaremos a los métodos `animate()` y `doDraw()` de la clase `Panel`.

```

@Override
public void run() {
    Canvas canvas = null;
    mStartTime = System.currentTimeMillis();
    while (mRun) {
        canvas = mHolder.lockCanvas();
        if (canvas != null) {
            mPanel.animate(mElapsed);
            mPanel.doDraw(mElapsed, canvas);
            mElapsed = System.currentTimeMillis() -
mStartTime;
            mHolder.unlockCanvasAndPost(canvas);
        }
        mStartTime = System.currentTimeMillis();
    }
}

```

Si ejecutamos la aplicación, cada vez que pulsemos sobre la pantalla aparecerá un nuevo ícono que se empezará a mover por la pantalla de una forma aleatoria.



La aplicación completa “**SeasMovimiento2D**” se puede descargar desde la plataforma de estudio.

2.2. GRÁFICOS 3D

Los gráficos en 2D son ideales para la mayoría de programas, pero hay ocasiones que vamos a necesitar de un nivel adicional de realismo que no es posible realizar con las dos dimensiones. Para estos casos, Android nos ofrece una biblioteca de gráficos tridimensionales basada en el **estándar OpenGL ES**.

En los siguientes capítulos, exploraremos los conceptos sobre las 3D y realizaremos un ejemplo que implementa OpenGL.

2.2.1. INTRODUCCIÓN A OPENGL

OpenGL fue desarrollado por *Silicon Graphics* en 1992. Proporciona a los desarrolladores un interfaz para que puedan aprovechar el hardware de cualquier dispositivo.

En su núcleo implementa conceptos familiares, e intenta ocultar la capa de hardware al desarrollador.

Realmente, OpenGL fue desarrollado para estaciones de trabajo, por lo que es demasiado grande como para ser utilizado en un dispositivo móvil, así que se implementa una versión *Embedded Systems*, o para sistemas incrustados. Este estándar fue creado por el *Kronos Group*, del cual formaban parte empresas como AMD, Intel, Nvidia, Nokia, Samsung y Sony.

Android implementa esta biblioteca vinculada al lenguaje Java, de manera que puede consultar libros acerca de JSR 239 (*Java Specification Request*) y OpenGL ES, para obtener información adicional y completa de sus clases y métodos.

2.2.2. IMPLEMENTANDO OPENGL

Comenzaremos creando un nuevo proyecto y rellenando el cuadro de diálogo, introduciendo los siguientes parámetros:

- **Project Name (nombre del proyecto):** SeasOpenGL
- **Build Target (objetivo de compilación):** Android 2.2
- **Application name (nombre de la aplicación):** Seas, OpenGL
- **Package name (nombre del paquete):** com.seas.ejemplo.opengl
- **Create Activity (crear actividad):** OpenGL
- **Min SDK Version:** 8

Una vez creada la Activity principal, lo editamos y modificamos para que haga referencia a una vista personalizada que llamaremos GLView:

```
package com.seas.ejemplo.opengl;

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

public class OpenGL extends Activity {
    GLView view;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        view = new GLView(this);
        // esta línea nos dará error porque crearemos la clase más
        adelante
        setContentView(view);
    }

    @Override
    protected void onPause() {
        super.onPause();
        view.onPause();
    }

    @Override
    protected void onResume() {
        super.onResume();
        view.onResume();
    }
}
```

Modificamos también los métodos onPause() y onResume() para que puedan llamar a métodos con el mismo nombre dentro de la vista. Y definimos nuestra clase de vista personalizada GLView añadiendo el siguiente código:

```
package com.seas.ejemplo.opengl;

import android.content.Context;
import android.opengl.GLSurfaceView;

class GLView extends GLSurfaceView {
    private final GLRenderер renderer;

    GLView(Context context) {
        super(context);

        // si queremos activar la comprobación de errores y el
        // registro debemos descomentar la línea siguiente
        // setDebugFlags(DEBUG_CHECK_GL_ERROR |
        DEBUG_LOG_GL_CALLS);

        renderer = new GLRenderер(context);
        setRenderер(renderer);
    }
}
```

Aquí aparece la clase `GLSurfaceView`. Es la que se ocupa de seleccionar el formato de pixel del framebuffer y gestionar un hilo de renderizado independiente que permitirá mostrar una animación suavizada.



GLSurfaceView

La clase `GLSurfaceView`, que se introdujo en Android 1.5, que simplificaba bastante el uso de OpenGL, es el pegamento para conectar OpenGL ES con el sistema de vista y el ciclo de vida de una actividad.

Necesitamos que `GLView` extienda a `GLSurfaceView` y definir un renderizador para la vista.

Ahora vamos a llenar la pantalla con un color sólido. Recordando conceptos anteriormente definidos, en 2D llamábamos al método `onDraw()` de la vista cuando necesitábamos redibujar una sección de la pantalla. OpenGL funciona de una manera un poco diferente.

En OpenGL, el dibujado está separado en una clase de renderizado que va a ser el responsable de inicializar y dibujar la pantalla. Para ello debemos definir la clase `GLRenderer`:

```
package com.seas.ejemplo.opengl;

import javax.microedition.khronos.egl.EGLConfig;
import javax.microedition.khronos.opengles.GL10;

import android.content.Context;
import android.opengl.GLSurfaceView;
import android.opengl.GLU;
import android.util.Log;

class GLRenderer implements GLSurfaceView.Renderer {
    private static final String TAG = "GLRenderer";
    private final Context context;

    public void onSurfaceCreated(GL10 gl, EGLConfig config) {
        // ...
    }
    public void onSurfaceChanged(GL10 gl, int width, int height) {
        // ...
    }
    public void onDrawFrame(GL10 gl) {
        // ...
    }
}
```

GLRender er implementa la interfaz GLSurfaceView.Renderer, la cual tiene tres métodos: onSurfaceCreated(), onSurfaceChanged() y onDrawFrame().

El primero de ellos, onSurfaceCreated, se llama cuando la Surface (parecido al Canvas del 2D) OpenGL se crea y vuelve a crear:

```
public void onSurfaceCreated(GL10 gl, EGLConfig config) {  
  
    // Definimos todas las opciones OpenGL que necesitamos  
    gl.glEnable(GL10.GL_DEPTH_TEST);  
    gl.glDepthFunc(GL10.GL_EQUAL);  
    gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);  
  
    // Optional: desactivamos dither para mejora rendimiento  
    // gl.glDisable(GL10.GL_DITHER);  
}
```

En este método hemos definido de momento un par de opciones OpenGL. Esto lo realizamos con glEnable() y glDisable(). Las opciones más utilizadas son las siguientes:

- **GL_BLEND**: mezcla valores del color entrante con los que ya están en el buffer de color.
- **GL_CULL_FACE**: ignora polígonos en función de su ondulación (que puede ser en el sentido de las manetas del reloj o en sentido contrario) en coordenadas de ventana. Es una manera fácil de eliminar caras traseras.
- **GL_DEPTH_TEST**: realiza comparaciones y actualiza el buffer de profundidad. Se ignoran píxeles que están más alejados de los ya dibujados.
- **GL_LIGHTi**: incluye el número de la luz 'i' cuando se va a interpretar el color y el brillo de un objeto.
- **GL_LIGHTING**: activa cálculos de iluminación y material.
- **GL_LINE_SMOOTH**: dibuja líneas suavizadas.
- **GL_MULTISAMPLE**: es el que realiza un muestreo múltiple para suavizar líneas y dar otros efectos.
- **GL_POINT_SMOOTH**: dibuja puntos suavizados.
- **GL_TEXTURE_2D**: dibuja superficies utilizando texturas.

Por defecto, todas estas opciones están desactivadas, excepto GL_DITCHER y GL_MULTISAMPLE. Debemos tener presente, que todo lo que se active, posteriormente repercutirá en el rendimiento.

Vamos a seguir con nuestro código, y rellenaremos el método `onSurfaceChanged()`, que será llamado después de que la Surface sea creada y siempre que el tamaño de ésta cambie:

```
public void onSurfaceChanged(GL10 gl, int width, int height) {  
    // ...  
  
    // Definimos la vista frustum  
    gl.glViewport(0, 0, width, height);  
    gl.glMatrixMode(GL10.GL_PROJECTION);  
    gl.glLoadIdentity();  
    float ratio = (float) width / height;  
    GLU.gluPerspective(gl, 45.0f, ratio, 1, 100f);  
  
}
```

Con esto hemos configurado nuestra *view frustum* y definimos más opciones de OpenGL. Para entender mejor que es el *frustum* veamos la siguiente imagen.

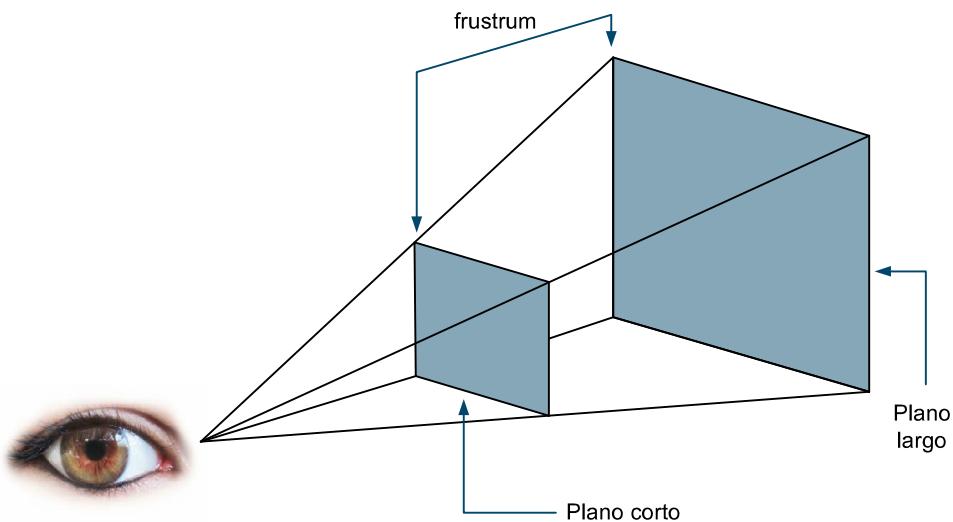


Figura 2.7. View frustum.



Frustum

Es el espacio que contiene en su interior todos los objetos potencialmente visibles en la pantalla.

Si observamos la llamada a la función `GLU.gluPerspective()`, los dos argumentos últimos son la distancia que hay entre el ojo y los planos de recorte.

Vamos a dibujar algo. Realizamos llamadas una y otra vez al método `onDrawFrame()` en el hilo de renderizado creado por la clase `GLSurfaceView`.

```
public void onDrawFrame(GL10 gl) {
    // ...

    // limpia la pantalla y la pone en negro
    gl.glClear(GL10.GL_COLOR_BUFFER_BIT |
    GL10.GL_DEPTH_BUFFER_BIT);

    // Posiciona el modelo para que lo podamos ver
    gl.glMatrixMode(GL10.GL_MODELVIEW);
    gl.glLoadIdentity();
    gl.glTranslatef(0, 0, -3.0f);

    // otros comandos de dibujo irán aquí...
}
```

Con esto coloreamos la pantalla en negro, limpiamos el buffer de profundidad y de color. Tendremos que realizar esto si no queremos obtener resultados extraños procedentes de la información de profundidad de un fotograma anterior.

También hemos definido la posición inicial para el resto de comandos de dibujo, que iremos completando más adelante. Si ejecutamos en este momento la aplicación, lo único que se verá es la pantalla en negro una y otra vez en bucle.

Ahora sí que vamos a realizar algo más interesante. Lo primero vamos a definir el modelo. Para ello debemos tener en cuenta, que dependiendo de la complejidad de los objetos que se quieran dibujar, lo recomendable es crearlo utilizando alguna herramienta de diseño gráfico e importarlo. Para este ejemplo lo haremos definiendo un modelo sencillo: **un cubo**. Para ello debemos crear la clase `GLCube` e insertar el siguiente código:

```
package com.seas.ejemplo.opengl;

import java.nio.ByteBuffer;
import java.nio.ByteOrder;
import java.nio.IntBuffer;

import javax.microedition.khronos.opengles.GL10;

import android.content.Context;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.opengl.GLUtis;

class GLCube {
    private final IntBuffer mVertexBuffer;
```

```
    private final IntBuffer mTextureBuffer;
```

```
public GLCube() {

    int one = 65536;
    int half = one / 2;
    int vertices[] = {
        // FRONT
        -half, -half, half, half, -half, half,
        -half, half, half, half, half, half,
        // BACK
        -half, -half, -half, -half, half, -half,
        half, -half, -half, half, half, -half,
        // LEFT
        -half, -half, half, -half, half, half,
        -half, -half, -half, -half, half, -half,
        // RIGHT
        half, -half, -half, half, half, -half,
        half, -half, half, half, half, half,
        // TOP
        -half, half, half, half, half, half,
        -half, half, -half, half, half, -half,
        // BOTTOM
        -half, -half, half, -half, -half, -half,
        half, -half, half, half, -half, -half, };

    int texCoords[] = {
        // FRONT
        0, one, one, one, 0, 0, one, 0,
        // BACK
        one, one, one, 0, 0, one, 0, 0,
        // LEFT
        one, one, one, 0, 0, one, 0, 0,
        // RIGHT
        one, one, one, 0, 0, one, 0, 0,
        // TOP
        one, 0, 0, 0, one, one, 0, one,
        // BOTTOM
        0, 0, 0, one, one, 0, one, one, };

    // Buffers que pasaran a funciones gl_Pointer() deben ser
    // directas. Deben colocarse en el conjunto nativo
    // donde el colector de basura no pueda moverlos
    //
    // Buffers con tipos de datos de varios bits(short, int,
    // float) deberán tener su orden de bits ya establecido como
    // orden nativo
    ByteBuffer vbb = ByteBuffer.allocateDirect(vertices.length *
4);
    vbb.order(ByteOrder.nativeOrder());
    mVertexBuffer = vbb.asIntBuffer();
    mVertexBuffer.put(vertices);
    mVertexBuffer.position(0);

    // ...
}
```

```

        ByteBuffer tbb = ByteBuffer.allocateDirect(texCoords.length *
4);
        tbb.order(ByteOrder.nativeOrder());
        mTextureBuffer = tbb.asIntBuffer();
        mTextureBuffer.put(texCoords);
        mTextureBuffer.position(0);

    }

    public void draw(GL10 gl) {
        gl.glVertexPointer(3, GL10.GL_FIXED, 0, mVertexBuffer);

        gl.glEnable(GL10.GL_TEXTURE_2D); // workaround bug 3623
        gl.glTexCoordPointer(2, GL10.GL_FIXED, 0, mTextureBuffer);

        gl.glColor4f(1, 1, 1, 1);
        gl.glNormal3f(0, 0, 1);
        gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, 0, 4);
        gl.glNormal3f(0, 0, -1);
        gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, 4, 4);

        gl.glColor4f(1, 1, 1, 1);
        gl.glNormal3f(-1, 0, 0);
        gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, 8, 4);
        gl.glNormal3f(1, 0, 0);
        gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, 12, 4);

        gl.glColor4f(1, 1, 1, 1);
        gl.glNormal3f(0, 1, 0);
        gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, 16, 4);
        gl.glNormal3f(0, -1, 0);
        gl.glDrawArrays(GL10.GL_TRIANGLE_STRIP, 20, 4);
    }

    static void loadTexture(GL10 gl, Context context, int resource) {
        Bitmap bmp = BitmapFactory.decodeResource(
            context.getResources(), resource);
        GLUtils.texImage2D(GL10.GL_TEXTURE_2D, 0, bmp, 0);
        gl.glTexParameterx(GL10.GL_TEXTURE_2D,
            GL10.GL_TEXTURE_MIN_FILTER, GL10.GL_LINEAR);
        gl.glTexParameterx(GL10.GL_TEXTURE_2D,
            GL10.GL_TEXTURE_MAG_FILTER, GL10.GL_LINEAR);
        bmp.recycle();
    }
}

```

La matriz vértices (`int vertices[]`) define las esquinas del cubo en coordenadas de modelo de punto fijo. Cada cara del cubo es un cuadrado que está formado por dos triángulos. Hemos utilizado el modo habitual que usa OpenGL para dibujar, llamado “*triangle strips*” (triángulos adyacentes). En este modo, lo que hacemos es especificar dos puntos iniciales y, a continuación, cada punto subsecuente define un triángulo con los dos puntos anteriores. Hay que tener en cuenta, que cada punto va a tener tres coordenadas (x, y, z).

Los ejes “x-“ e “y-“ apuntan hacia la derecha y hacia arriba, y el eje “z-“ apuntará hacia la pantalla en línea con el punto de vista.

En el método `draw()`, estamos utilizando el buffer de vértice que hemos creado en el constructor y dibujamos seis rondas distintas de triángulos (una para cada cara del cubo).

Veamos ahora la nueva clase en `GLRenderer`:

```
private final GLCube cube = new GLCube();
public void onDrawFrame(GL10 gl) {

    // ...
    // Aquí dibujamos el modelo
    cube.draw(gl);
}
```

Si ejecutamos nuestra aplicación, veremos una pantalla en negro con un cuadrado en el centro.

Para continuar con nuestra aplicación debemos detenernos un poco, y hablar de la iluminación. OpenGL nos permite definir hasta ocho fuentes de luz en su escena. En la iluminación tenemos dos partes, la luz y el objeto que queremos iluminar. Comencemos hablando de la luz. Las bibliotecas son compatibles con tres tipos:

- **Difusa:** es una iluminación suave, parecida a la que da la luz fluorescente.
- **Ambiental:** es un resplandor general de luz tenue que ilumina la escena completa.
- **Especular (o de espejo):** es una luz reluciente, procedente de fuentes brillantes direccionales.

Estos valores se incluyen en la ecuación que se define en el método `GLRenderer.onSurfaceCreated()`:

```
public void onSurfaceCreated(GL10 gl, EGLConfig config) {

    // ...
    //Definimos la iluminación

    float lightAmbient[] = new float[] { 0.2f, 0.2f, 0.2f, 1 };
    float lightDiffuse[] = new float[] { 1, 1, 1, 1 };
    float[] lightPos = new float[] { 1, 1, 1, 1 };
    gl.glEnable(GL10.GL_LIGHTING);
    gl.glEnable(GL10.GL_LIGHT0);
    gl.glLightfv(GL10.GL_LIGHT0, GL10.GL_AMBIENT, lightAmbient,
0);
    gl.glLightfv(GL10.GL_LIGHT0, GL10.GL_DIFFUSE, lightDiffuse,
0);
    gl.glLightfv(GL10.GL_LIGHT0, GL10.GL_POSITION, lightPos, 0);
```

Hemos definido la fuente de iluminación en la posición (1,1,1). Será una luz blanca con una única dirección, con un componente difuso brillante y uno más ambiental tenue. En nuestro ejemplo dejamos de usar la iluminación especular.

A continuación, vamos a añadir los materiales con los que estará fabricado nuestro cubo. Añadimos este código al método `onSurfaceCreated()` así definimos la reacción del material con los tres tipos de iluminación:

```
// Definimos materiales con los que esta echo

float matAmbient[] = new float[] { 1, 1, 1, 1 };
float matDiffuse[] = new float[] { 1, 1, 1, 1 };
gl.glMaterialfv(GL10.GL_FRONT_AND_BACK, GL10.GL_AMBIENT,
matAmbient, 0);
gl.glMaterialfv(GL10.GL_FRONT_AND_BACK, GL10.GL_DIFFUSE,
matDiffuse, 0);
```

Hasta este punto hemos realizado ya la iluminación de la escena y los materiales con los que hemos construido el cubo. Ahora vamos a animarlo un poco. Para ello debemos modificar los métodos `onSurfaceCreated()` y `onDrawFrame()` dentro de la clase `GLRender`.

```
private long startTime;
private long fpsStartTime;
private long numFrames;
public void onSurfaceCreated(GL10 gl, EGLConfig config) {
    // ...
    startTime = System.currentTimeMillis();
    fpsStartTime = startTime;
    numFrames = 0;
}
public void onDrawFrame(GL10 gl) {

    // Establecemos el angulo de rotacion en funcion del tiempo
    long elapsed = System.currentTimeMillis() - startTime;
    gl.glRotatef(elapsed * (30f / 1000f), 0, 1, 0);
    gl.glRotatef(elapsed * (15f / 1000f), 1, 0, 0);

    // Aqui dibujamos el modelo
    cube.draw(gl);
}
```

Así estamos girando el cubo cada vez que recorremos el bucle principal. Exactamente, 30 grados por segundo alrededor de “x” y 15 grados alrededor de “y”.

Ahora vamos a aplicar una textura a nuestro cubo. El procedimiento para hacerlo es bastante extenso. No hay que preocuparse si no se entiende directamente. En la clase `GLCube` añadimos el siguiente código:

```
private final IntBuffer mTextureBuffer;

public GLCube() {
    int texCoords[] = {
        // FRONT
```

```

        0, one, one, one, 0, 0, one, 0,
        // BACK
        one, one, one, 0, 0, one, 0, 0,
        // LEFT
        one, one, one, 0, 0, one, 0, 0,
        // RIGHT
        one, one, one, 0, 0, one, 0, 0,
        // TOP
        one, 0, 0, 0, one, one, 0, one,
        // BOTTOM
        0, 0, 0, one, one, 0, one, one, };
    // ...
    ByteBuffer tbb =
    ByteBuffer.allocateDirect(texCoords.length * 4);
    tbb.order(ByteOrder.nativeOrder());
    mTextureBuffer = tbb.asIntBuffer();
    mTextureBuffer.put(texCoords);
    mTextureBuffer.position(0);

}

static void loadTexture(GL10 gl, Context context, int
resource) {
    Bitmap bmp =
    BitmapFactory.decodeResource(context.getResources(),
        resource);
    GLUtils.texImage2D(GL10.GL_TEXTURE_2D, 0, bmp, 0);
    gl.glTexParameterx(GL10.GL_TEXTURE_2D,
    GL10.GL_TEXTURE_MIN_FILTER,
        GL10.GL_LINEAR);
    gl.glTexParameterx(GL10.GL_TEXTURE_2D,
    GL10.GL_TEXTURE_MAG_FILTER,
        GL10.GL_LINEAR);
    bmp.recycle();
}

}

```

Una vez realizado esto, tenemos que indicarle a OpenGL las coordenadas de textura, añadiendo el siguiente código al comienzo del método draw():

```

gl.glEnable(GL10.GL_TEXTURE_2D); // solucion de bugs 3623
gl.glTexCoordPointer(2, GL10.GL_FIXED, 0, mTextureBuffer);

```

y, tenemos que llamar por último al método loadTexture() en GLRenderer, añadiendo las siguientes líneas al final del método onSurfaceCreated():

```

// Activamos las texturas
gl.glEnableClientState(GL10.GL_TEXTURE_COORD_ARRAY);
gl.glEnable(GL10.GL_TEXTURE_2D);

// cargamos la textura del cubo a partir de nuestra imagen
GLCube.loadTexture(gl, context, R.drawable.seas);

```

Con esto hemos activado primero las texturas y las coordenadas, para después llamar al método loadTexture(), pasándole el contexto Activity y el ID del recurso, para que pueda cargar la imagen de textura.

R.drawable.*seas* es un archivo PNG de 128 píxeles que se copia al directorio res/drawable-nodpi/. Nosotros hemos usado esa imagen corporativa, pero se puede usar cualquier archivo PNG del tamaño que desee.

Llegados a este punto y con el código ya casi terminado, vamos a hacer que el cubo se vuelva parcialmente transparente. Añadiremos el siguiente código a GLRenderer.onResume():

```
public void onSurfaceCreated(GL10 gl, EGLConfig config) {  
    // ...  
  
    boolean SEE_THRU = true;  
  
    // ...  
    if (SEE_THRU) {  
        gl.glDisable(GL_DEPTH_TEST);  
        gl.glEnable(GL_BLEND);  
        gl.glBlendFunc(GL_SRC_ALPHA, GL_ONE);  
    }  
}
```

Estas opciones de OpenGL, como por ejemplo *GL_DEPTH_TEST*, nos sirven para desactivar las comparaciones de profundidad, así los veremos poco nítidos al igual que los que están en primer plano.

Con *GL_BLEND* permitimos que la opacidad de los objetos esté basada en su canal alfa. Conseguiremos que las caras traseras del cubo aparezcan a través de las caras frontales.



Figura 2.8. Versión final del cubo a través de la que podemos ver.

Aunque ya hemos acabado nuestro pequeño ejemplo de aplicación en 3D, debemos hablar de un tema de suavidad y delicadeza a la hora de representar movimientos gráficos, que viene definido por la velocidad con la que se pueden actualizar la visualización en pantalla.

Esta velocidad suele medirse contando el número de pantallas o fotogramas por segundo. Lo recomendable sería conseguir unas tasas consistentes de 60FPS. Esto supone un reto bastante importante, ya que tan solo tenemos 1/60 de un segundo (16,67 milisegundos) entre las llamadas a `onDrawFrame()` para poder realizar todo lo que necesita hacer, incluyendo animaciones, cálculos físicos, más el tiempo que tarda en dibujar la escena propiamente dicha para el fotograma actual.

Vamos a realizar una medición de dichos FPS. Para ello vamos a añadir al final del método `onDrawFrame()` el siguiente código:

```
// Llevar un registro del numero de FPS

numFrames++;
long fpsElapsed = System.currentTimeMillis() - fpsStartTime;
if (fpsElapsed > 5 * 1000) { // cada 5 segundos
    float fps = (numFrames * 1000.0F) / fpsElapsed;
    Log.d(TAG, "Frames per second: " + fps + " (" +
        numFrames + " frames in " + fpsElapsed + " "
        + ms));
    fpsStartTime = System.currentTimeMillis();
    numFrames = 0;
}
```

Vamos a registrar cada 5 segundos el número de FPS medio en el registro del sistema, que podremos ver en el *LogCat* de la perspectiva DDMS. Si realizamos la prueba en el emulador y en un dispositivo real, podremos observar que en este último las tasas son más cercanas a los 60FPS, lo que nos enseña que para realizar pruebas de rendimiento no se puede confiar en el emulador.



En este capítulo hemos aprendido a utilizar la biblioteca de gráficos 3D de Android. Existe mucha información disponible sobre OpenGL ES. En concreto, le recomendamos el javadoc sobre JSR239 (<http://download.oracle.com/javame/config/cldc/opt-pkgs/api/jb/jsr239/index.html>).



El ejemplo completo “**SeasOpenGL**” se puede descargar desde la plataforma de estudio.

2.3. MULTIMEDIA

En este capítulo vamos a mostrarle cómo añadir multimedia a nuestras aplicaciones Android.

2.3.1. REPRODUCCIÓN DE AUDIO

Android es compatible con salidas de sonido y música a través de la clase `MediaPlayer` del paquete `android.media`. Vamos a realizar un sencillo ejemplo que va a reproducir sonidos cuando se pulse una tecla del teclado.

Vamos a comenzar creando un nuevo proyecto de Android utilizando los siguientes parámetros en el cuadro de diálogo:

- **Project Name (nombre del proyecto):** SeasAudio
- **Build Target (objetivo de compilación):** Android 2.2
- **Application name (nombre de la aplicación):** Seas, Audio
- **Package name (nombre del paquete):** com.seas.ejemplo.audio
- **Create Activity (crear actividad):** Audio
- **Min SDK Version:** 8

Una vez creado el nuevo proyecto, debemos conseguir unos cuantos sonidos en mp3 y debemos copiarlos en la carpeta `res/raw` de nuestro proyecto. Esto nos generará en la clase R un símbolo Java. Las carpetas quedarán de la siguiente manera.

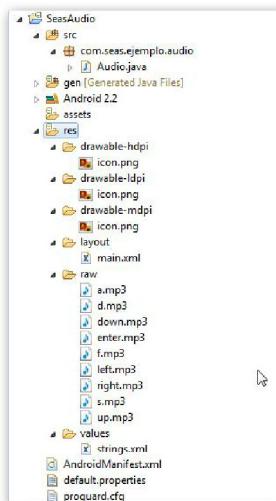


Figura 2.9. Vista de cómo quedarán los directorios de nuestro proyecto.

Pasemos al código. Debemos llenar la actividad Audio, declarando un campo llamado mp que almacenará una instancia de la clase MediaPlayer. En nuestra aplicación tendremos activa una MediaPlayer:

```
package com.seas.ejemplo.audio;

import android.app.Activity;
import android.media.AudioManager;
import android.media.MediaPlayer;
import android.os.Bundle;

public class Audio extends Activity {
    private MediaPlayer mp;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        setVolumeControlStream(AudioManager.STREAM_MUSIC);
    }
}
```

Con el método `setVolumeControlStream()` le vamos a indicar a Android, que mientras nuestra aplicación se esté ejecutando, cuando el usuario pulse la tecla de subir o bajar el volumen, se ajuste el volumen multimedia u otros medios en vez de hacerlo para el timbre del teléfono.

Ahora debemos de interceptar cada pulsación de tecla y reproducir el sonido elegido para ello. Debemos modificar el método `Activity.onKeyDown()` añadiendo el siguiente código en nuestra clase Audio:

```
@Override
public boolean onKeyDown(int keyCode, KeyEvent event) {
    int resId;
    switch (keyCode) {
        case KeyEvent.KEYCODE_DPAD_UP:
            resId = R.raw.up;
            break;
        case KeyEvent.KEYCODE_DPAD_DOWN:
            resId = R.raw.down;
            break;
        case KeyEvent.KEYCODE_DPAD_LEFT:
            resId = R.raw.left;
            break;
        case KeyEvent.KEYCODE_DPAD_RIGHT:
            resId = R.raw.right;
            break;
        case KeyEvent.KEYCODE_DPAD_CENTER:
        case KeyEvent.KEYCODE_ENTER:
            resId = R.raw.enter;
            break;
        case KeyEvent.KEYCODE_A:
            resId = R.raw.a;
            break;
        case KeyEvent.KEYCODE_S:
```

```
    resId = R.raw.s;
    break;
case KeyEvent.KEYCODE_D:
    resId = R.raw.d;
    break;
case KeyEvent.KEYCODE_F:
    resId = R.raw.f;
    break;
default:
    return super.onKeyDown(keyCode, event);
}

// liberamos cualquier recurso anterior de MediaPlayer
if (mp != null) {
    mp.release();
}

// Creamos un nuevo MediaPlayer para reproducir este sonido
mp = MediaPlayer.create(this, resId);
mp.start();

// Indicamos que esta tecla ya fue controlada
return true;
}
```

Con este método primero seleccionamos un recurso (mp3) en base a la tecla que fue pulsada. Desde ese momento, llamamos al método `release()` para detener cualquier sonido que se estuviese reproduciendo y liberamos cualquier recurso que estuviese asociado al `MediaPlayer`. Si no hiciésemos esto, el programa fallaría, ya que a continuación, cuando utilizamos el método `create()` para crear un nuevo `MediaPlayer`, utilizando el recurso seleccionado y llamamos al método `start()` (este método es asíncrono, por lo tanto, devuelve inmediatamente con independencia de lo que tarde el sonido en terminar) tendría todavía el recurso anterior y daría fallo al llamarlo.

Para dejar un poco más bonita la aplicación, en la pantalla principal vamos a poner unas pequeñas instrucciones de cómo interactuar con la aplicación. Debemos modificar el `resource` y añadir la siguiente string al archivo `main.xml`:

```
<?xml version= "1.0" encoding= "utf-8"?>
<resources>
    <string name= "hello"> Hello World, Audio! </string>
    <string name= "app_name"> Seas, Audio </string>
    <string name= "directions">
        Presiona una tecla para escuchar un sonido:
        Arriba, Izquierda, Abajo, Derecha, Enter,
        A, S, D, o F.
    </string>
</resources>
```

Y el layout.xml modificarlo para que muestre el recurso anteriormente definido:

```
<?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/directions"
    />
</LinearLayout>
```

Si ejecutamos la aplicación y pulsamos sobre una de las teclas, deberíamos oír un sonido. Si no disponemos de teclado físico en el teléfono, mantenemos pulsada la tecla “Menú” del teléfono hasta que salga el teclado en pantalla.



La aplicación completa “**SeasAudio**” se puede descargar desde la plataforma de estudio.

2.3.1.1. FORMATOS COMPATIBLES

Habrá que diferenciar entre compatibilidad teórica, compatibilidad en el emulador y compatibilidad en dispositivos reales. En teoría, Android es compatible con los siguientes tipos de archivos:

- WMA.
- MIDI.
- AAC.
- OGG.
- AMR.
- WAV.
- MP3.

Realizando diferentes pruebas, hemos podido comprobar que los formatos que realmente funcionan en el emulador son OGG, WAV y MP3, así que son los que te recomendamos usar a la hora de realizar aplicaciones.

2.3.2. REPRODUCCIÓN DE VIDEO

Para poder reproducir video en nuestras aplicaciones usaremos la misma clase que para el audio, `MediaPlayer`, y funciona de la misma forma que lo hace con el audio. Lo que deberemos crear una `Surface` para que el reproductor utilice las imágenes, haciendo uso de los métodos `onStart()` y `onStop()` para controlar la reproducción.

Pero en vez de realizar otro proyecto ejemplo de la clase `MediaPlayer`, lo que haremos será usar la clase `VideoView`.

Vamos a comenzar creando un nuevo proyecto de Android utilizando los siguientes parámetros en el cuadro de diálogo:

- **Project Name (nombre del proyecto):** SeasVideo
- **Build Target (objetivo de compilación):** Android 2.2
- **Application name (nombre de la aplicación):** Seas, Video
- **Package name (nombre del paquete):** com.seas.ejemplo.video
- **Create Activity (crear actividad):** Video
- **Min SDK Version:** 8

Una vez creado el nuevo proyecto cambiamos el archivo `res/layout/main.xml` por éste:

```
<?xml version= "1.0" encoding= "utf-8"?>
<FrameLayout
    xmlns:android= "http://schemas.android.com/apk/res/android"
    android:layout_width= "fill_parent"
    android:layout_height= "fill_parent">
    <VideoView
        android:id= "@+id/video"
        android:layout_height= "wrap_content"
        android:layout_width= "wrap_content"
        android:layout_gravity= "center" />
</FrameLayout>
```

Con esto hemos creado un nuevo componente `VideoView` que nos va a ocupar toda la pantalla, y será donde visualizaremos el video.

Ahora en la clase `Video.java` cambiamos el método `onCreate()` por lo siguiente:

```
package com.seas.ejemplo.video;

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

public class Video extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        // cargamos la vista desde el recurso
        setContentView(R.layout.main);
        VideoView video = (VideoView) findViewById(R.id.video);

        // cargamos y comenzamos a reproducir la película
        video.setVideoPath("/sdcard/video/nombreVideo.extension");
        video.start();
    }
}
```

Debemos insertar en la carpeta `sdcard/video` el archivo de video que deseemos reproducir. Valdrá cualquier video que hayamos grabado con la videocámara del móvil y con poner el nombre del archivo en la línea de nuestro código.

Con el método `setVideoPath()` definimos la ubicación del archivo. Y con `video.start()` abrimos el archivo, lo redimensionamos al tamaño del contenedor para poder mantener la relación de aspecto y comenzamos a reproducirlo.

Pero aún nos faltaría darle un poco de presencia, para que el video ocupe la pantalla completa y desaparezca la barra de título y la barra de estado. Para ello debemos modificar el `AndroidManifest.xml`:

```
<?xml version= "1.0" encoding= "utf-8"?>
<manifest xmlns:android=
"http://schemas.android.com/apk/res/android"
    package= "com.seas.ejemplo.video"
    android:versionCode= "1"
    android:versionName= "1.0">
    <uses-sdk android:minSdkVersion= "8" />

    <application android:icon= "@drawable/icon" android:label=
"@string/app_name">
        <activity android:name= ".Video"
            android:label= "@string/app_name"
            android:theme=
"@android:style/Theme.NoTitleBar.Fullscreen">
            <intent-filter>
                <action android:name= "android.intent.action.MAIN"
/>
                <category android:name=
"android.intent.category.LAUNCHER" />
```

```
</intent-filter>
</activity>

</application>

</manifest>
```

Si a mitad de reproducción gira la pantalla, observaremos que el video se reinicia desde el principio. Y eso es debido a que Android da por hecho que su programa no sabe nada acerca de las rotaciones de pantalla. Lo que hace Android es destruir y volver a crear la actividad desde cero, con lo que se vuelve a llamar al `onCreate()`, lo que implica la reapertura del video.

Existen dos formas para poder optimizar la transición. La manera fácil es implementar `onRetainNonConfigurationInstance()` en su actividad y guardar alguno de sus datos que se mantendrán en las llamadas a `onDestroy()` y `onCreate()`. Al volver, utilizamos `getLastNonConfigurationInstance()` en la nueva instancia de su actividad para traer dicha información. De esta manera, mantendremos las referencias a su intención actual e hilos de ejecución.

2.3.2.1. FORMATOS COMPATIBLES

Al igual que hicimos con el audio, te damos un listado de los formatos oficialmente compatibles con Android:

- MP4.
- 3GP (H.263).
- AVC (H.264).

Y el formato recomendado va a ser H.263 puesto que todas las plataformas de hardware son compatibles con él y bastante eficaz hablando de rendimiento. Es compatible con otros dispositivos como el iPhone, y se puede reproducir en el ordenador con programas de licencia gratuita.

Hay que intentar siempre utilizar las menores resoluciones y tasas de transferencia de bits para que podamos ahorrar espacio sin que eso afecte a la calidad del video.



Se puede descargar el código completo del programa **"SeasVideo"** desde la plataforma de estudio.

2.4. BASES DE DATOS SQLITE

SQLite es un potente motor de base de datos, creado en el año 2000 por el Dr. Richard Hipp. Podríamos decir, que se trata del motor más usado en el mundo, ya que se encuentra presente tanto en el *iPhone*, como en los teléfonos *Symbian*, en el *Mozilla Firefox*, en *PHP*, en *Skype*, en *Mac OS X*, en *Adobe AIR*, en *Solaris* y en muchas otras plataformas y sistemas. Puedes ampliar información visitando su página web <http://sqlite.org>.

Los motivos por los que se ha hecho tan popular son:

Pequeño: la versión actual ocupa aproximadamente los 150KB.

Gratis: esta para su descarga en el dominio público y no cobran por su uso.

No es necesaria su configuración o administración, ya que no dispone de servidor, archivo de configuración ni necesita de un administrador.

Tenemos que tener en cuenta que una BBDD de SQLite únicamente es un archivo. Ese archivo lo podemos mover e incluso copiar a otro sistema (como a nuestro disco duro del ordenador) y funcionará perfectamente. Ese archivo está almacenado en el siguiente directorio

`data/data/nombreDelPaquete/databases:`

Name	Size	Date	Time	Permissions	Info
data		2011-03-30	17:01	drwxrwx--x	
app		2011-04-04	16:03	drwxrwx--x	
app-private		2011-03-30	16:59	drwxrwx--x	
backup		2011-03-30	17:01	drwx-----	
dalvik-cache		2011-04-04	16:03	drwxrwx--x	
data		2011-04-01	11:49	drwxrwx--x	
android.tts		2011-03-30	17:00	drwxr-x--x	
com.android.alarmclock		2011-03-30	17:02	drwxr-x--x	
databases		2011-03-31	14:49	drwxrwx--x	
alarms.db	4096	2011-03-31	14:49	-rw-rw----	

Figura 2.10. Archivo que SQLite emplea para la BBDD.

Sería importante saber, o por lo menos, poseer ciertos conocimientos de SQL. Si este no es el caso, vamos a realizar una breve introducción a ello, pero sería necesario que se realizara algún curso o tutorial al respecto.

Las BBDD funcionan de la siguiente manera. Nosotros enviamos instrucciones y obtenemos resultados. Existen tres tipos: DDL, de consulta y de modificación.

DDL

Hay que tener claro que un archivo de BBDD contiene un número determinado de tablas, cada una de ellas está compuesta de filas y éstas a su vez contienen una serie de columnas. Cada columna posee un nombre y un tipo de datos (int, varchar, txt,...). Para definir esto usamos los DDL (*Data Definitions Language*). Por poner un ejemplo, el siguiente código crea una tabla con cuatro columnas:

```
create table miTabla {

    _id integer primary key autoincrement,
    Nombre text, Apellidos text, Telefono "txt");
```

Observamos que una columna la creamos como PRIMARY_KEY, que será el número que identificará de forma única a la fila. Con AUTOINCREMENT nos aseguramos que la BBDD añadirá una a la clave de cada registro. Por convención, llamaremos `_id` a la primera columna, y aunque no es necesaria en SQLite, más adelante, cuando usemos ContentProvider la necesitaremos.

Instrucciones de modificación

Con este tipo de instrucciones podremos insertar, actualizar y borrar registros en la BBDD. Por ejemplo, en la tabla que antes hemos creado, vamos a insertar unos valores ficticios:

```
insert into miTabla values (null, 'Pepito', 'Perez Perez',
'600123456');

insert into miTabla values (null, 'Antonio', 'Martin Martin',
'600456321');

insert into miTabla values (null, 'Miguel', 'Gomez Gomez',
'600654123');
```

Fíjate como los valores los introducimos siempre en el mismo orden que se realizó la instrucción de CREATE_TABLE, y vemos también como especificamos el valor NULL para el `_id`, ya que es un valor que se introducirá automáticamente.

Instrucciones de consulta

Éstas son las instrucciones que se ejecutarán contra la BBDD, una vez que ya tengamos insertados todos los datos necesarios. Utilizaremos la instrucción SELECT para recuperar los datos. Si, por ejemplo, queremos obtener la línea 2, escribiríamos:

```
select * from miTabla where (_id=2);
```

Hay que tener claro que SQL no distingue entre mayúsculas y minúsculas, ni en nombres de columna, ni en las cadenas de búsqueda.



Es recomendable escribir el nombre de las tablas en mayúsculas y el nombre de la columna en minúsculas.

Aunque ha sido una breve introducción a SQL, si no se es muy docto con el tema, se recomienda buscar información acerca de gestión de BBDD. Vamos a ver un pequeño ejemplo de cómo implementar una BBDD en una aplicación.

2.4.1. APPLICACIÓN CON SQLITE

Vamos a crear una pequeña aplicación llamada `Events` en la que guardaremos registros para mostrarlos posteriormente. Abrimos un nuevo proyecto utilizando los siguientes parámetros en el cuadro de diálogo:

- **Project Name (nombre del proyecto):** SeasAccess1
- **Build Target (objetivo de compilación):** Android 2.2
- **Application name (nombre de la aplicación):** Seas, Accesos
- **Package name (nombre del paquete):** com.seas.ejemplo.access
- **Create Activity (crear actividad):** Access
- **Min SDK Version:** 8

Lo primero que necesitamos es crear un sitio para mantener las constantes que describan la BBDD, así que vamos a crear una interfaz `Constants`:

```
package com.seas.ejemplo.access;

import android.provider.BaseColumns;

public interface Constants extends BaseColumns {
    public static final String TABLE_NAME = "access";

    // Columnas de la BBDD Events
    public static final String TIME = "tiempo";
    public static final String TITLE = "titulo";
}
```

Con esto vamos a almacenar cada evento como una fila en la tabla `access`. Tendremos tres columnas en cada fila con los nombres `_id`, `tiempo` y `titulo`.

Ahora vamos a crear `AccessData`, que será una clase de ayuda para representar la BBDD en sí. Ésta extiende de `SQLiteOpenHelper`, que es la clase de Android que realiza la creación y control de las versiones de las BBDD. Debemos proporcionarle el constructor y modificar ambos métodos.

```
package com.seas.ejemplo.access;

import static android.provider.BaseColumns._ID;
import static com.seas.ejemplo.access.Constants.TABLE_NAME;
import static com.seas.ejemplo.access.Constants.TIME;
import static com.seas.ejemplo.access.Constants.TITLE;
import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;

public class AccessData extends SQLiteOpenHelper {
    private static final String DATABASE_NAME = "access.db";
    private static final int DATABASE_VERSION = 1;

    // creamos un objeto de ayuda para la BBDD events
    public EventsData(Context ctx) {

        super(ctx, DATABASE_NAME, null, DATABASE_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL("CREATE TABLE " + TABLE_NAME + " (" + _ID
                + " INTEGER PRIMARY KEY AUTOINCREMENT, " + TIME
                + " INTEGER," + TITLE + " TEXT NOT NULL);");
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion,
            int newVersion) {
        db.execSQL("DROP TABLE IF EXISTS " + TABLE_NAME);
        onCreate(db);
    }
}
```

En el constructor `AccessData`, `DATABASE_NAME` será el nombre de la BBDD (`access.db`) y `DATABASE_VERSION` será el número de versión que deberemos cambiar cada vez que hagamos algún cambio significativo en la BBDD. En nuestro caso no faremos cambios, puesto que se trata de un ejemplo.

`SQLiteOpenHelper` cuando intente acceder por primera vez a la BBDD, verá que no existe y llamará al método `onCreate()` para hacerlo.

Si Android observa que estamos haciendo referencia a una base de datos antigua basándose en el número de versión, llamará al método `onUpgrade()`. Lo que hemos hecho nosotros en este ejemplo con este método es borrar la BBDD existente y crearla de nuevo.

Definamos el programa principal. Vamos a almacenar los eventos y los vamos a mostrar en un TextView. Esto lo realizamos en el archivo res/layout/main.xml:

```
<?xml version= "1.0" encoding= "utf-8"?>
<ScrollView
    xmlns:android= "http://schemas.android.com/apk/res/android"
    android:layout_width= "fill_parent"
    android:layout_height= "fill_parent">
    <TextView
        android:id= "@+id/text"
        android:layout_width= "fill_parent"
        android:layout_height= "wrap_content" />
</ScrollView>
```

Vamos a ir creando todos los métodos de la Activity Access y explicándolos. Comenzemos con onCreate():

```
package com.seas.ejemplo.access;

import static android.provider.BaseColumns._ID;
import static com.seas.ejemplo.access.Constants.TABLE_NAME;
import static com.seas.ejemplo.access.Constants.TIME;
import static com.seas.ejemplo.access.Constants.TITLE;
import android.app.Activity;
import android.content.ContentValues;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.os.Bundle;
import android.widget.TextView;

public class Access extends Activity {
    private AccessData access;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        access = new AccessData(this);
        try {
            addAccess("Hola, Seas!"); /*1*
            Cursor cursor = getAccess(); ///*2*
            showAccess(cursor); /*3*
        } finally {
            access.close();
        }
    }
}
```

Hemos creado una instancia de la clase AccessData y mediante un try llamamos al método addAccess() para añadir un nuevo acceso cada vez que se ejecute el programa. Posteriormente llamamos al método getAccess() para obtener los accesos y al showAccess() para mostrárselos al usuario.

Vamos a ir definiendo cada uno de los métodos que hemos nombrado. Empezamos por el de añadir accesos en la BBDD (*1*).

```
private void addAccess(String string) {
    // Insertamos un nuevo registro dentro de la BBDD access
    // Habría que realizar algo parecido para actualizar y borrar.
    SQLiteDatabase db = access.getWritableDatabase();
    ContentValues values = new ContentValues();
    values.put(TIME, System.currentTimeMillis());
    values.put(TITLE, string);
    db.insertOrThrow(TABLE_NAME, null, values);
}
```

Hemos llamado a `getWritableDatabase` para modificar los datos, que es un manejador de lectura/escritura (en este caso escritura), que va a ser almacenado en la cache, pudiéndolo llamar tantas veces como se necesite. Rellenamos el objeto `ContentValues` con la hora y el título del acceso, y se lo pasamos al método `insertOrThrow()` (insertar o tirar).

Continuamos definiendo el método `getAccess()`, que será el que realice la consulta a la BBDD:

```
private static String[] FROM = { _ID, TIME, TITLE, };
private static String ORDER_BY = TIME + " DESC";
private Cursor getAccess() {
    // Consulta controlada: la actividad manejará el cierre
    // y nuevas consultas cuando sea necesario
    SQLiteDatabase db = access.getReadableDatabase();
    Cursor cursor = db.query(TABLE_NAME, FROM, null, null, null,
    null, ORDER_BY);
    startManagingCursor(cursor);
    return cursor;
}
```

En este método llamaremos a `getReadableDatabase()`, ya que no tenemos que modificar, sino únicamente obtener datos de la BBDD. Este será un manejador de solo lectura. Posteriormente, llamaremos a `query()` que será el que ejecutará la Select. Hemos definido `FROM` como una matriz de las columnas que nos devolverá la consulta y `ORDER_BY` será la que indique la forma de ordenación del resultado por tiempo y en orden descendente.

Un `Cursor` es como un `Iterator` de Java o un `ResultSet` de JDBC. Llamaremos a métodos incluidos en él para obtener información de las filas y pasar a la siguiente fila. Veremos cómo funciona cuando definamos el método de mostrar los resultados.

Por último, hemos llamado al método `startManagingCursor()`, que es el que gestiona el ciclo de vida del cursor en base al ciclo de vida de la `Activity`. Con esto queremos decir, que si la `Activity` está en pausa, el cursor se va a desactivar automáticamente, y volverá a realizar la consulta, cuando se reanude la `Activity`. Si la finalizamos, se cierran todos los cursos.

Vamos a definir el método `showAccess()` (*3*) que será el que muestre los resultados al usuario que estén contenidos en la BBDD.

```
private void showAccess(Cursor cursor) {
    // guardamos todos los datos en una unica cadena
    StringBuilder builder = new StringBuilder("Accesos
guardados:\n");
    while (cursor.moveToNext()) {
        long id = cursor.getLong(0);
        long time = cursor.getLong(1);
        String title = cursor.getString(2);
        builder.append(id).append(": ");
        builder.append(time).append(": ");
        builder.append(title).append("\n");
    }
    // los mostramos en la pantalla
    TextView text = (TextView) findViewById(R.id.text);
    text.setText(builder);
}
```

StringBuilder

Son objetos de cadena de texto, que pueden ser modificados. Internamente, estos objetos son tratados como matrices de longitud variable que contienen una secuencia de caracteres. En cualquier momento, el tamaño y el contenido de la secuencia de caracteres se pueden cambiar a través de llamadas a métodos.



Los principales son los `append` e `insert`. Ambos están sobrecargados con el fin de aceptar datos de cualquier tipo. Cada uno de ellos, convierte un dato dado a cadena de texto y luego añade o inserta los caracteres de esa cadena al constructor de cadena. El método `append` siempre añade estos caracteres al final del constructor, y en cambio, el método `insert` los agrega en un punto especificado.

En esta versión únicamente se va a guardar en una cadena de texto cada vez que se ejecute el programa. Con el método `Cursor.moveToNext()` avanzamos a la siguiente fila del conjunto de datos. Cuando su valor es `false`, nos indica que ya no quedan más filas. Con los métodos `getLong()` y `getString()`, obtendremos los datos de las columnas que nos interesan para posteriormente añadirlos a la cadena.

Si ejecutamos dicho ejemplo, veremos algo similar a lo de la imagen siguiente. En nuestro caso vemos que hemos accedido al programa cinco veces.



Figura 2.11. Resultado de nuestra primera versión del programa.

Nuestra tabla únicamente contiene cinco registros, pero si contuviese millones de ellos, sería lentísimo y podríamos quedarnos sin memoria intentando construir una cadena con todos los datos. Además, si quisiéramos realizar algún evento sobre alguno de ellos, si están en una sola cadena, no es posible. Para solucionar todo esto Android nos da la posibilidad de usar la vinculación de datos.



El ejemplo completo “**SeasAccess1**” se puede descargar desde la plataforma de estudio.

2.4.2. VINCULACIÓN DE DATOS

Nos permitirá conectar un modelo de datos con su vista en tan solo unas pocas líneas de código. Partiremos del ejemplo anterior *SeasAccess1*, modificándolo para que use un *ListView* vinculada a una consulta a la BBDD. Para ello copiamos la carpeta del proyecto y cuando nos de la opción de cambiar el nombre la llamamos *SeasAccess2*. Y el proyecto *SeasAccess1* lo podemos cerrar para guardarlo como tal.

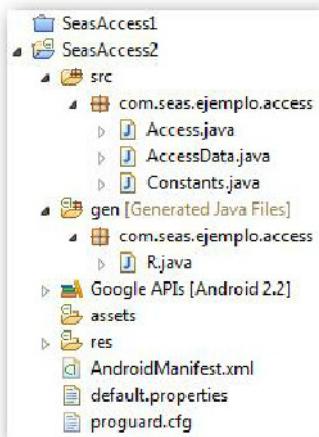


Figura 2.12. Vista con los dos proyectos del workspace.

Una vez hecho esto, necesitamos que la clase `Access` extienda a `ListActivity`:

```
import android.app.ListActivity;
//...

public class Access extends ListActivity {
    //...
}
```

Además, tendremos que cambiar la manera de mostrar nuestros eventos en el método `showAccess()`:

```
import android.widget.SimpleCursorAdapter;
private static String[] FROM = { _ID, TIME, TITLE, };
private static int[] TO = { R.id.rowid, R.id.time, R.id.title, };
private void showAccess(Cursor cursor) {
    // Vinculacion de datos
    SimpleCursorAdapter adapter = new SimpleCursorAdapter(this,
        R.layout.item, cursor, FROM, TO);
    setListAdapter(adapter);
}
```

Veamos como el código se ha reducido notablemente. Únicamente hemos creado un `SimpleCursorAdapter` para el `Cursor` y hemos indicado a `ListActivity` que lo utilice. El adaptador es el que conecta la vista con el origen de sus datos.

El `SimpleCursorAdapter` toma cinco parámetros:

- **context:** referencia a la actividad actual.
- **layout:** define las vistas.
- **cursor:** cursor del conjunto de datos.

- **from:** nombres de columna de donde provienen los datos.
- **to:** lista de vistas hacia donde dirigimos los datos.

Debemos definir el elemento de la lista en `layout/item.xml`. Para ello debemos crear un nuevo archivo `xml`. A continuación vemos como llenar los campos que se necesitan:

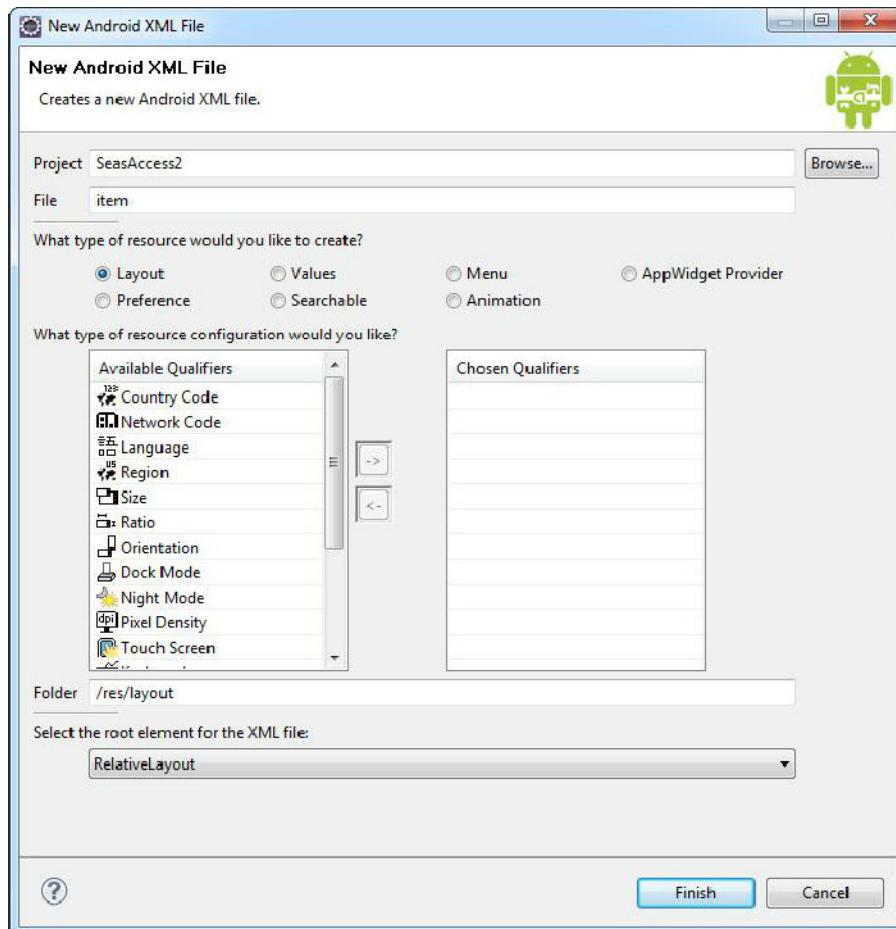


Figura 2.13. Cómo insertar el archivo `item.xml`.

Una vez creado insertamos el siguiente código:

```
<?xml version= "1.0" encoding= "utf-8"?>
<RelativeLayout
    xmlns:android= "http://schemas.android.com/apk/res/android"
    android:layout_width= "fill_parent"
    android:layout_height= "fill_parent"
    android:orientation= "horizontal"
    android:padding= "10sp">
    <TextView
        android:id= "@+id/rowid"
        android:layout_width= "wrap_content"
```

```
        android:layout_height= "wrap_content" />
<TextView
    android:id= "@+id/rowidcolon"
    android:layout_width= "wrap_content"
    android:layout_height= "wrap_content"
    android:text= ":" "
    android:layout_toRightOf= "@+id/rowid" />
<TextView
    android:id= "@+id/time"
    android:layout_width= "wrap_content"
    android:layout_height= "wrap_content"
    android:layout_toRightOf= "@+id/rowidcolon" />
<TextView
    android:id= "@+id/timecolon"
    android:layout_width= "wrap_content"
    android:layout_height= "wrap_content"
    android:text= ":" "
    android:layout_toRightOf= "@+id/time" />
<TextView
    android:id= "@+id/title"
    android:layout_width= "fill_parent"
    android:layout_height= "wrap_content"
    android:ellipsize= "end"
    android:singleLine= "true"
    android:textStyle= "italic"
    android:layout_toRightOf= "@+id/timecolon" />
</RelativeLayout>
```

Con esto ya tenemos colocado el ID, la hora y el título en una línea con dos puntos entre ellos, un pequeño relleno y un poco de formato para que quede un poco más bonito.

Por último, debemos cambiar en `layout/main.xml` e insertamos lo siguiente:

```
<?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">
    <!-- observa los id incluidos para 'list' y 'empty' -->
    <ListView
        android:id= "@+id/list"
        android:layout_width= "wrap_content"
        android:layout_height= "wrap_content"/>
    <TextView
        android:id= "@+id/empty"
        android:layout_width= "wrap_content"
        android:layout_height= "wrap_content"
        android:text= "@string/empty" />
</LinearLayout>
```

Como la actividad extiende de `ListActivity`, Android buscará los dos ID especiales. Si la lista tiene elementos que mostrar, lo hará `android:id/list`; si sucede lo contrario, se mostrará la vista `android:id/empty`. Si no hay elementos, el usuario verá el mensaje “No hay accesos!” en vez de la pantalla negra. Para ello debemos insertar en el `strings.xml` el recurso:

```
<?xml version= "1.0" encoding= "utf-8"?>
<resources>
    <string name= "app_name">Seas, Accesos</string>
    <string name= "empty">No hay accesos!</string>
</resources>
```

Si ejecutamos nuestra aplicación, vemos nuestros registros de la *BBDD* con los que ya podemos interactuar.

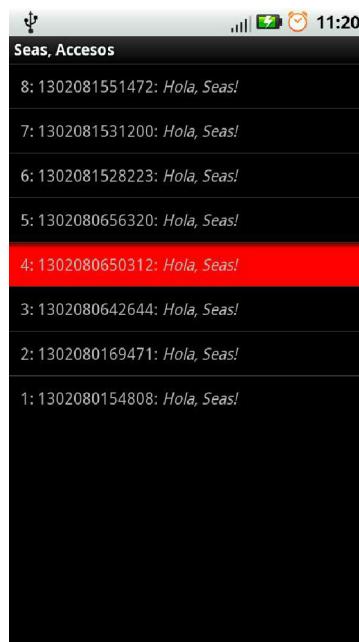


Figura 2.14. Versión de SQLite con vinculación de datos.

Pero todavía tenemos un problema, ninguna otra aplicación podrá añadir registros a la BBDD de accesos ni acceder a ellos. Para poder realizar esto hay que utilizar un `ContentProvider` de Android.

2.4.3. UTILIZAR UN CONTENTPROVIDER

En la unidad 1 de este temario ya se habló de los ContentProvider, y recordamos ahora que los archivos escritos en una aplicación no podrán ser accedidos por ninguna otra aplicación, tal y como se define en el modelo de seguridad Android. Recordemos también, que cada aplicación o programa tiene un ID de usuario Linux propio y directorio de datos (`data/data/nombrePaquete`), al igual que su espacio protegido de memoria. Hay dos formas de comunicarse entre programas de Android:

- **IPC (Inter Process Communication)** que es la comunicación entre procesos. Esta técnica se utiliza para llamadas entre procedimientos remotos a hilos de ejecución `Service` que se ejecutan en segundo plano. No hablaremos de estos métodos, pero si se quiere más información al respecto se puede visitar:
 - ✓ <http://d.android.com/guide/developing/tools/aidl.html>
 - ✓ <http://d.android.com/reference/android/app/Service.html>
 - ✓ <http://d.android.com/reference/android/os/IBinder.html>
- **ContentProvider:** los procesos se van a registrar por sí mismos en el sistema como proveedores de ciertos tipos de datos. Cuando se necesita cierta información, Android los llama mediante una API fija para poder acceder a ese contenido, ya sea para solo consultarlo o modificarlo. Esto lo vamos a utilizar en nuestro ejemplo de accesos.

Como ya vimos en la unidad 1, los ContentProvider gestionan la información mediante un direccionamiento a través de una URI con la forma siguiente:

`content://authority/path/id`

Dónde:

- `content://` es el prefijo estándar.
- `Authority` es el nombre del proveedor. Usar siempre el nombre completo del paquete para evitar problemas con los nombres y posibles colisiones.
- `Path` es el directorio virtual incluido en el proveedor, con el que identificamos el tipo de datos que queremos de él.
- `Id` es la clave primaria del registro específico que necesitamos

Un ejemplo de ellos sería:

- `content://browser`
- `content://contactContract`
- `content://media`
- `content://settings`



Android tiene muchos proveedores. Puede consultarlos todos en la siguiente dirección:

<http://d.android.com/reference/android/provider/package-summary.html>

Enlace disponible en la plataforma de estudio.

Para ver cómo usar un ContentProvider usado con SQLite, vamos a convertir nuestra aplicación de “Accesos” para que utilice uno. Para realizar nuestro proveedor usaremos las siguientes URI válidas:

```
content://com.seas.ejemplo.access/access/7 -un solo acceso con _id=7
content://com.seas.ejemplo.access/access - todos los accesos
```

Lo primero necesitamos añadir dos nuevas constantes en nuestra clase Constants:

```
import android.net.Uri;
// ...
public static final String AUTHORITY = "com.seas.ejemplo.access";
public static final Uri CONTENT_URI = Uri.parse("content://" +
    AUTHORITY + "/" + TABLE_NAME);
```

Los archivos main.xml e item.xml no será necesario que los editemos. Ahora vamos a realizar algunos cambios en la clase Access. El método onCreate() va a ser más sencillo, puesto que ya no habrá que controlar ningún objeto de la BBDD:

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    addAccess("Hola, Seas!");
    Cursor cursor = getAccess();
    showAccess(cursor);
}
```

Ahora hemos eliminado el bloque try/finally y eliminamos las referencias a EventData.

Al método addAccess() le modificamos dos líneas:

```
import static com.seas.ejemplo.access.Constants.CONTENT_URI;

private void addAccess(String string) {
    // Insertamos un nuevo registro dentro de la BBDD Events
    // Habria que realizar algo parecido para actualizar y borrar.
    ContentValues values = new ContentValues();
    values.put(TIME, System.currentTimeMillis());
    values.put(TITLE, string);
    getContentResolver().insert(CONTENT_URI, values);
}
```

La llamada a getWritableDatabase() la hemos eliminado y getContentResolver() sustituye a la llamada insertOrThrow() que se realizaba anteriormente. Lo que estamos haciendo es cambiar un manejador de BBDD por una URI de contenido.

El método getAccess() se simplifica:

```
private Cursor getAccess() {
    // Consulta controlada: la actividad manejará el cierre
    // y nuevas consultas cuando sea necesario
    return managedQuery(CONTENT_URI, FROM, null, null, ORDER_BY);
}
```

Hemos utilizado el método managedQuery(), pasándole como atributos la URI de contenido, las columnas y el orden en el que aparecerán.

Hasta aquí hemos disociado el cliente Access del proveedor de datos Access, eliminando todas las referencias a la BBDD. Ahora es cuando vamos a implementar el ContentProvider.

Lo primero que haremos será declararlo en el sistema, ya que se trata de un objeto de alto nivel como una Activity. Lo añadiremos al AndroidManifest.xml dentro de la etiqueta <application> y antes de la de <activity>:

```
<?xml version= "1.0" encoding= "utf-8"?>
<manifest xmlns:android=
"http://schemas.android.com/apk/res/android"
    android:versionCode= "1" android:versionName= "1.0" package=
"com.seas.ejemplo.access">
    <uses-sdk android:minSdkVersion= "8" />

    <application android:icon= "@drawable/icon" android:label=
"@string/app_name">
        <provider android:name= ".AccessProvider"
        android:authorities= "com.seas.ejemplo.access" />
        <activity android:name= "Access" android:label=
"@string/app_name">
            <intent-filter>
```

```

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

</application>

</manifest>
```

Con android:name asignamos el nombre de la clase, y con android:authorities la cadena que utilizamos en la URI de contenido.

Creamos la clase AccessProvider como extensión de ContentProvider. Insertamos el código siguiente:

```

package com.seas.ejemplo.access;

import static android.provider.BaseColumns._ID;
import static com.seas.ejemplo.access.Constants.AUTHORITY;
import static com.seas.ejemplo.access.Constants.CONTENT_URI;
import static com.seas.ejemplo.access.Constants.TABLE_NAME;

import com.seas.ejemplo.access.AccessData;

import android.content.ContentProvider;
import android.content.ContentUris;
import android.content.ContentValues;
import android.content.UriMatcher;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.net.Uri;
import android.text.TextUtils;

public class AccessProvider extends ContentProvider {

    private static final int ACCESS = 1;
    private static final int ACCESS_ID = 2;

    // El tipo MIME de un directorio de eventos
    private static final String CONTENT_TYPE =
        "vnd.android.cursor.dir/vnd.seas.ejemplo.access";

    // El tipo MIME de un solo evento
    private static final String CONTENT_ITEM_TYPE =
        "vnd.android.cursor.item/vnd.seas.ejemplo.access";

    private AccessData access;
    private UriMatcher uriMatcher;
    // ...

    @Override
    public boolean onCreate() {
        uriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
        uriMatcher.addURI(AUTHORITY, "access", ACCESS);
        uriMatcher.addURI(AUTHORITY, "access/#", ACCESS_ID);
        access = new AccessData(getApplicationContext());
    }
}
```

```
        return true;
    }

    @Override
    public Cursor query(Uri uri, String[] projection, String
selection,
                        String[] selectionArgs, String orderBy) {
        if (uriMatcher.match(uri) == ACCESS_ID) {
            long id =
Long.parseLong(uri.getPathSegments().get(1));
            selection = appendRowId(selection, id);
        }

        // traemos la BBDD y ejecutamos la consulta
        SQLiteDatabase db = access.getReadableDatabase();
        Cursor cursor = db.query(TABLE_NAME, projection,
selection,
                                selectionArgs, null, null, orderBy);

        // Le decimos al cursor cual es la URI a buscar,
        // que detecta cuando cambia la fuente de sus datos

        cursor.setNotificationUri(getContext().getContentResolver(),
uri);
        return cursor;
    }

    @Override
    public String getType(Uri uri) {
        switch (uriMatcher.match(uri)) {
        case ACCESS:
            return CONTENT_TYPE;
        case ACCESS_ID:
            return CONTENT_ITEM_TYPE;
        default:
            throw new IllegalArgumentException("URI
desconocida " + uri);
        }
    }

    @Override
    public Uri insert(Uri uri, ContentValues values) {
        SQLiteDatabase db = access.getWritableDatabase();

        // Validamos la URI requerida
        if (uriMatcher.match(uri) != ACCESS) {
            throw new IllegalArgumentException("URI
desconocida " + uri);
        }

        // Insertamos en la BBDD
        long id = db.insertOrThrow(TABLE_NAME, null, values);

        // Notificamos cualquier cambio
        Uri newUri = ContentUris.withAppendedId(CONTENT_URI,
id);
        getContext().getContentResolver().notifyChange(newUri,
null);
        return newUri;
    }
}
```

```
    }

    @Override
    public int delete(Uri uri, String selection, String[]
selectionArgs) {
        SQLiteDatabase db = access.getWritableDatabase();
        int count;
        switch (uriMatcher.match(uri)) {
        case ACCESS:
            count = db.delete(TABLE_NAME, selection,
selectionArgs);
            break;
        case ACCESS_ID:
            long id =
Long.parseLong(uri.getPathSegments().get(1));
            count = db.delete(TABLE_NAME,
appendRowId(selection, id),
                selectionArgs);
            break;
        default:
            throw new IllegalArgumentException("URI
desconocida " + uri);
        }

        // notificamos los cambios
        getContext().getContentResolver().notifyChange(uri,
null);
        return count;
    }

    @Override
    public int update(Uri uri, ContentValues values, String
selection,
        String[] selectionArgs) {
        SQLiteDatabase db = access.getWritableDatabase();
        int count;
        switch (uriMatcher.match(uri)) {
        case ACCESS:
            count = db.update(TABLE_NAME, values, selection,
selectionArgs);
            break;
        case ACCESS_ID:
            long id =
Long.parseLong(uri.getPathSegments().get(1));
            count = db.update(TABLE_NAME, values,
appendRowId(selection, id),
                selectionArgs);
            break;
        default:
            throw new IllegalArgumentException("URI
desconocida " + uri);
        }

        // notificamos los cambios
        getContext().getContentResolver().notifyChange(uri,
null);
        return count;
    }
```

```
// definimos la clausula where
private String appendRowId(String selection, long id) {
    return _ID
        + "="
        + id
        + (!TextUtils.isEmpty(selection) ? " AND (" +
+ selection + ')'
        : ""));
}
```

Vamos a utilizar por convención, `vnd.seas.ejemplo.access` en lugar de `com.seas.ejemplo.access` en el tipo MIME.

MIME



Es un estándar de Internet con el que describimos el tipo de cualquier clase de contenido. Son unas especificaciones dirigidas al intercambio de todo tipo de archivos a través de Internet, mejorando las posibilidades de intercambio de texto en distintos idiomas y alfabetos.

Con `AccessProvider` manejaremos dos tipos de datos:

- ACCESS (MIME tipo `CONTENT_TYPE`): Un directorio o lista de accesos.
- ACCESS_ID (MIME tipo `CONTENT_ITEM_TYPE`): Un solo acceso.

Con el primer tipo no especificamos un ID, en cambio con el segundo sí. Con la clase `UriMatcher` de Android analizamos la URI y volvemos a usar la clase `AccessData` para gestionar la BBDD real dentro del proveedor.

Si ejecutamos nuestra aplicación, veremos lo mismo que hemos visto en las anteriores. Aunque en la estructura interna ya disponemos de un almacén de accesos que podría ser utilizado por otras aplicaciones del sistema o aplicaciones de otros programadores.

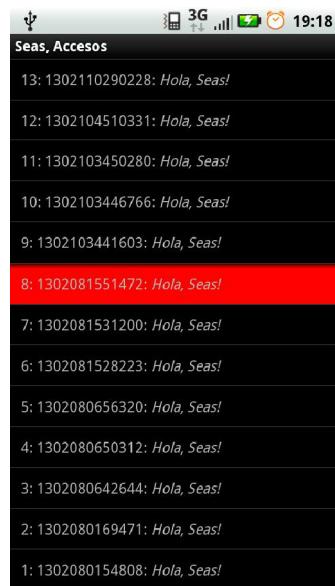


Figura 2.15. Vista de la aplicación, exactamente igual pero muy distinta internamente.



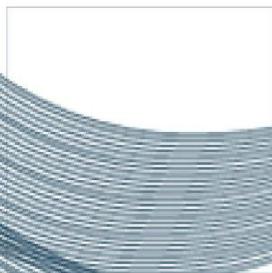
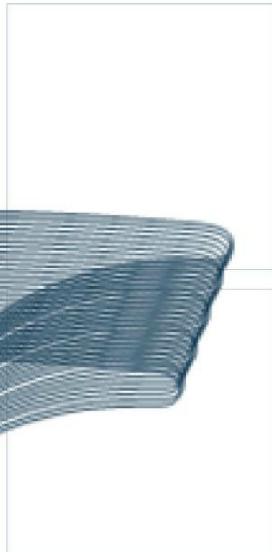
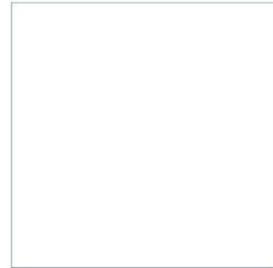
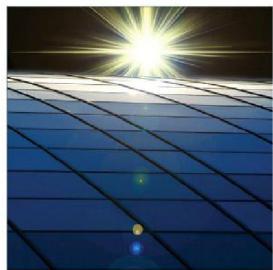
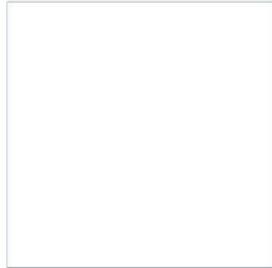
La aplicación completa “**SeasAccesos3**” se puede descargar desde la plataforma de estudio.

RESUMEN

- En esta unidad hemos visto los gráficos en 2D, implementando la clase `Color`, donde hemos descubierto que existe un cuarto parámetro en el RGB, que es alfa y es el nivel de transparencia.
- Hemos visto la clase `Paint`, con la que vamos a definir los estilos y colores.
- Hemos visto la clase `Canvas`, con la que definimos la superficie donde dibujamos, y que con sus métodos se pueden dibujar líneas, rectángulos, círculos y cualquier otro tipo de gráfico.
- También vimos la clase `Path`, con la que realizamos los trazados vectoriales.
- Hemos utilizado las bibliotecas de gráficos 3D, `OpenGL`, con la que se pueden realizar infinidad de configuraciones para dar una perspectiva más realista a nuestras imágenes.
- Hemos visto como reproducir una canción o un video mediante la utilización de la clase `MediaPlayer`, que implementa todos los métodos necesarios para esto.
- Hemos visto cómo utilizar las Bases de Datos de `SQLite` implementándolas en una aplicación que registraba cada acceso a dicha aplicación en una BBDD.

03

ANDROID



PRÓXIMAS
GENERACIONES

 SEAS
ESTUDIOS SUPERIORES ABIERTOS

ÍNDICE

◆ ÍNDICE	1
◆ OBJETIVOS	3
◆ INTRODUCCIÓN.....	4
3.1. Sistemas de Geolocalización.....	5
3.1.1. Geolocalizándonos	5
3.1.2. Configurar el Api Key	11
3.1.3. Mostrando Google Maps	13
3.1.4. Aplicación Localización GPS.....	29
3.2. Sensores	37
3.3. Bluetooth	41
3.4. MultiTouch	54
3.5. Reconocimiento de voz.....	63
◆ RESUMEN.....	71

OBJETIVOS

- Conoceremos los sistemas de geolocalización con Android y como geoposicionarnos.
- Veremos los sensores de los que dispone y que están presentes en cada uno de los dispositivos móviles que implementan este sistema operativo.
- Veremos los diferentes usos del acelerómetro que implementan los dispositivos y un ejemplo de cómo usarlo.
- Veremos los diferentes usos del Bluetooth.
- Analizaremos el sistema multitouch de los dispositivos que dispongan de él y lo veremos en un ejemplo.
- Veremos cómo funciona el motor de búsqueda por voz de Google y lo implementaremos a través de una sencilla aplicación.

INTRODUCCIÓN

En esta unidad se pretende aprovechar los sensores de los que dispone Android para el desarrollo de aplicaciones más atractivas. Recorremos lo que son los sensores y el hardware disponible de los dispositivos, que por suerte, Android nos permite manejar y obtener datos de ellos.

Veremos cómo usar el GPS, cómo funcionan los demás sensores del dispositivo, tanto el acelerómetro, como los sensores de gravedad, rotación, el multitouch de las pantallas, que se implementarán desde Android 2.0. y veremos cómo usar el dispositivo Bluetooth implementándolo en una pequeña aplicación.

3.1. SISTEMAS DE GEOLOCALIZACIÓN

La geolocalización es un término que se está usando desde mediados del 2009 y con el que nos referimos a nuestra ubicación geográfica. Ésta se recoge de manera automática y hay varias maneras de realizarlo, y los dispositivos móviles son los que más fácilmente realizan dicha lectura de nuestra posición, por la portabilidad de los dispositivos.

Desde hace tiempo, la mayoría de los dispositivos traen integrados receptores GPS.

El GPS o Global Positioning System



Es el sistema de posicionamiento global que fue creado por el Dpto. de Defensa de Estados Unidos y que suministra información sobre la posición y la velocidad en todo el mundo. Mediante el acceso a una red de 24 satélites que rodean al planeta, podemos ubicarnos en cualquier sitio que nos encontremos.

Android utiliza diversas tecnologías para realizar dicha geoposición, una de ellas es la de los satélites GPS descrita anteriormente, y la otra la realiza calculando la posición por la información que suministran las torres de telefonía móvil, o las conexiones a redes Wi-Fi. Estas posiciones no son tan exactas como las que nos ofrecen los satélites GPS, pero pueden servir para dar una posición aproximada de nuestra situación.

3.1.1. GEOLOCALIZÁNDONOS

Para ver los servicios de posicionamiento de Android, vamos a realizar una pequeña aplicación que vaya mostrando nuestra situación actual y la renueve en pantalla cada vez que nos desplacemos.

Para ello vamos a crear un nuevo proyecto y vamos a llenar los campos de la siguiente manera:

- **Project Name (nombre del proyecto):** SeasLocationTest
- **Build Target (objetivo de compilación):** Android 2.2
- **Application name (nombre de la aplicación):** Seas, Localizacion
- **Package name (nombre del paquete):** com.seas.ejemplo.locationtest
- **Create Activity (crear actividad):** LocationTest
- **Min SDK Version:** 8

Una vez realizado esto, tenemos que modificar el archivo `AndroidManifest.xml`, otorgar los permisos necesarios y tener acceso a la información de ubicación:

```
<?xml version= "1.0" encoding= "utf-8"?>
<manifest xmlns:android=
"http://schemas.android.com/apk/res/android"
    package= "com.seas.ejemplo.locationtest" android:versionCode=
"1"
        android:versionName= "1.0">
    <uses-sdk android:minSdkVersion= "8" />
    <application android:icon= "@drawable/icon" android:label=
"@string/app_name">
        <!-- Permisos localizacion redes moviles y WIFI -->
        <uses-permission android:name=
"android.permission.ACCESS_COARSE_LOCATION" />
        <!-- Permisos localizacion receptor GPS -->
        <uses-permission android:name=
"android.permission.ACCESS_FINE_LOCATION" />
        <!-- Permisos acceso INTERNET -->
        <uses-permission android:name=
"android.permission.INTERNET" />
        <activity android:name= ".LocationTest" android:label=
"@string/app_name">
            <intent-filter>
                <action android:name=
"android.intent.action.MAIN" />
                <category android:name=
"android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Hemos dado permiso tanto a los proveedores de ubicación por GPS, como a los de posición por triangulación de antenas móviles. Toda la información que nos dé la mostraremos con un `TextView` desplazable, que lo definimos en el `res/layout/main.xml`:

```
<?xml version= "1.0" encoding= "utf-8"?>
<ScrollView
    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/output"
        android:layout_width= "fill_parent"
        android:layout_height= "wrap_content" />
</ScrollView>
```

Vamos a seguir con nuestra clase `LocationTest` y el código que debemos introducir en ella, en el método `onCreate()`. De momento ignoramos la referencia a `LocationListener` que implementamos en la `Activity`:

```
package com.seas.ejemplo.locationtest;

import java.util.List;

import android.app.Activity;
import android.location.Criteria;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.location.LocationProvider;
import android.os.Bundle;
import android.widget.TextView;

public class LocationTest extends Activity implements

    LocationListener {

    private LocationManager mgr;
    private TextView output;
    private String best;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        mgr = (LocationManager) getSystemService(LOCATION_SERVICE);
        output = (TextView) findViewById(R.id.output);

        log("Location providers:");
        dumpProviders();

        Criteria criteria = new Criteria();
        best = mgr.getBestProvider(criteria, true);
        log("\nBest provider is: " + best);

        log("\nLocations (starting with last known):");
        Location location = mgr.getLastKnownLocation(best);
        dumpLocation(location);
    }
}
```

La llamada al método `getSystemService()` nos devuelve una clase `LocationManager` que almacenamos para su posterior uso. También hemos llamado a `dumpProviders()`, con el que imprimimos a todos los proveedores de posición. De esa lista seleccionaremos uno. Tenemos la opción de seleccionar el primero disponible o, como va a ser nuestro caso, usar el mejor proveedor, `getBestProvider()`, dependiendo del `Criteria` que le pasemos. Estos pueden ser restricciones de energía, coste, precisión, etc. Nosotros no le vamos a poner ninguna restricción.

Dependiendo del proveedor, nuestro dispositivo puede tardar un tiempo más o menos largo en interpretar su posición actual, pero tenemos que tener en cuenta que Android almacena la última posición conocida, por lo que podemos usar el método `getLastKnownLocation()` para consultarla e imprimirla directamente. Lo más seguro será que dicha posición difiera bastante de la posición actual, pero tenemos un punto de partida para la aplicación.

Ahora vamos a hacer que se muestren los cambios de posición, llamando al método `requestLocationUpdates()` en el objeto `LocationManager`. Para poder ahorrar batería, algo muy importante en los dispositivos móviles, lo que vamos a hacer es que se actualice únicamente cuando esté en segundo plano la aplicación. Por lo que modificaremos los métodos `onPause()` y `onResume()`.

```
@Override  
protected void onResume() {  
    super.onResume();  
    // comienza las actualizaciones (retraso recomendado >= 60000  
    ms)  
    mgr.requestLocationUpdates(best, 15000, 1, this);  
}  
  
@Override  
protected void onPause() {  
    super.onPause();  
    // Detiene las actualizaciones para ahorrar bateria  
    // mientras la aplicacion este en pausa  
    mgr.removeUpdates(this);  
}
```

El método `requestLocationUpdates()` será llamado cuando se reanude la aplicación. Este método coge cuatro parámetros, el proveedor, el retraso de las actualizaciones, la distancia mínima y un objeto `LocationListener`.

Si la aplicación está en pausa, se llamará al método `removeUpdates()` y así detendremos la obtención de la actualización de la posición.

Este es el motivo por el que `LocationListener` es implementado por `LocationTest`, así pasamos una referencia a la actividad en vez de crear un nuevo objeto.

A continuación definimos los cuatro métodos requeridos por la interfaz:

```
public void onLocationChanged(Location location) {  
    dumpLocation(location);  
}  
  
public void onProviderDisabled(String provider) {  
    log("\nProvider disabled: " + provider);  
}  
  
public void onProviderEnabled(String provider) {  
    log("\nProvider enabled: " + provider);  
}
```

```
public void onStatusChanged(String provider, int status,
    Bundle extras) {
    log("\nProvider status changed: " + provider + ", status="
        + S[status] + ", extras=" + extras);
}
```

El método `onLocationChanged()` es llamado cada vez que cambia la posición del dispositivo. Los métodos `onStatusChanged()`, `onProviderDisabled()` y `onProviderEnabled()`, se pueden utilizar si el proveedor que hayamos elegido no estuviese disponible. A continuación introducimos los demás métodos:

```
// definimos los nombres legibles
private static final String[] A = { "invalid", "n/a", "fine",
"coarse" };
private static final String[] P = { "invalid", "n/a", "low",
"medium",
"high" };
private static final String[] S = { "out of service",
"temporarily unavailable", "available" };

/** Escribe un string en la ventana de salida */
private void log(String string) {
    output.append(string + "\n");
}

/** Escribe informacion de todos los proveedores de posición */
private void dumpProviders() {
    List<String> providers = mgr.getAllProviders();
    for (String provider : providers) {
        dumpProvider(provider);
    }
}

/** Escribe información de un solo proveedor de posición */
private void dumpProvider(String provider) {
    LocationProvider info = mgr.getProvider(provider);
    StringBuilder builder = new StringBuilder();
    builder.append("LocationProvider[")
        .append("name=")
        .append(info.getName())
        .append(",enabled=")
        .append(mgr.isProviderEnabled(provider))
        .append(",getAccuracy=")
        .append(A[info.getAccuracy() + 1])
        .append(",getPowerRequirement=")
        .append(P[info.getPowerRequirement() + 1])
        .append(",hasMonetaryCost=")
        .append(info.hasMonetaryCost())
        .append(",requiresCell=")
        .append(info.requiresCell())
        .append(",requiresNetwork=")
        .append(info.requiresNetwork())
        .append(",requiresSatellite=")
        .append(info.requiresSatellite())
        .append(",supportsAltitude=")
```

```

.append(info.supportsAltitude())
.append(", supportsBearing=")
.append(info.supportsBearing())
.append(", supportsSpeed=")
.append(info.supportsSpeed())
.append("]");
log(builder.toString());
}

/** Describe la posición dada, que podría ser ninguna */
private void dumpLocation(Location location) {
    if (location == null)
        log("\nLocation[unknown]");
    else
        log("\n" + location.toString());
}

```

Aquí vemos el resultado de nuestra aplicación, cómo muestra todos los proveedores y cómo realmente coge el que mejor posicionamiento le va a dar.

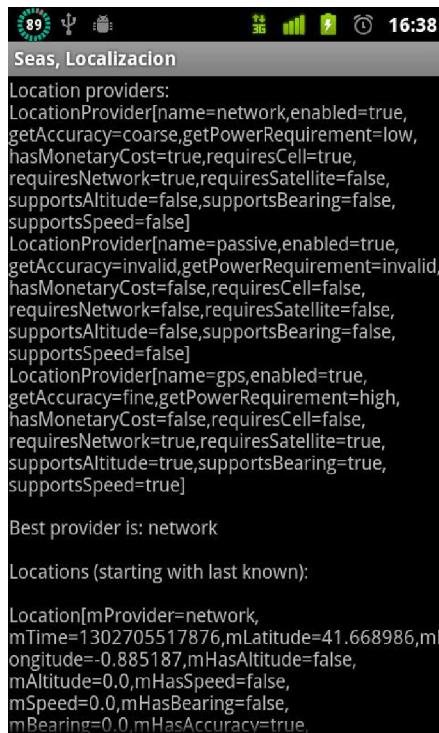


Figura 3.1. Aplicación Seas LocationTest en ejecución.

Vamos a ver otro ejemplo de aplicación en la que vamos a usar los mapas de google.



La aplicación completa “**SeasLocationTest**” se puede descargar desde la plataforma de estudio.

3.1.2. CONFIGURAR EL API KEY

Antes de poder usar un mapa, necesitaremos una clave de API de *Google Maps* y declararla en el archivo de diseño. Para ello debemos solicitar una en la siguiente dirección disponible también en la plataforma de estudio:

<http://code.google.com/intl/es-ES/android/maps-api-signup.html>



Figura 3.2. Página de Google donde solicitar el API Key.

Veremos una página como la anterior en la que nos pide lo primero que insertemos nuestro certificado MD5. Para obtenerlo debemos actuar de la siguiente manera. Lo primero tenemos que ir al explorador de archivos y realizar una búsqueda en nuestro equipo del archivo “debug.keystore”.

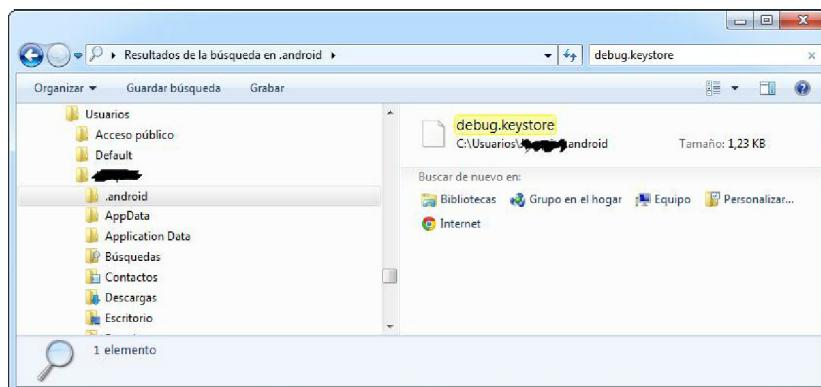


Figura 3.3. Búsqueda en nuestro equipo del archivo “debug.keystore”.

Deberíamos encontrarlo en el directorio de instalación por defecto de las SDK Tools: "C:\Documents and Settings\<username>\Local Settings\Application Data\Android". Aunque en cada equipo y sistema operativo puede instalarse en una ruta diferente, recomendamos encarecidamente se realice la búsqueda en cada ordenador.

Una vez encontrado dicho archivo hacemos una copia de él, en el directorio "C:\Android\", si no existe el directorio, lo creamos para tal efecto.

Una vez realizada la copia del archivo, abrimos una consola de sistema (mediante "ejecutar" y "cmd") y accedemos hasta el directorio en el que instalamos la máquina virtual java. Una vez allí, vamos a escribir lo siguiente:

```
keytool.exe -list -alias androiddebugkey -keystore  
"C:\android\debug.keystore" -storepass android -keypass android
```

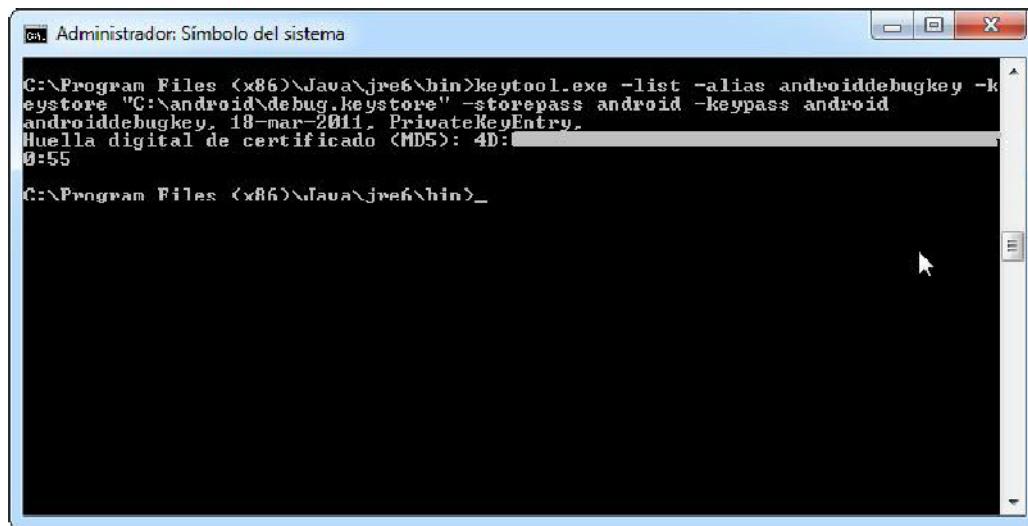


Figura 3.4. Obtención de la huella de certificado MD5.

Una vez obtenido, lo copiamos y lo insertamos en la página que anteriormente teníamos abierta:



Figura 3.5. Escribimos en el cuadro la obtenida anteriormente.

Pulsamos sobre “Generate API Key” y nos aparece la siguiente pantalla dándonos la conformidad de que el proceso se ha generado correctamente:

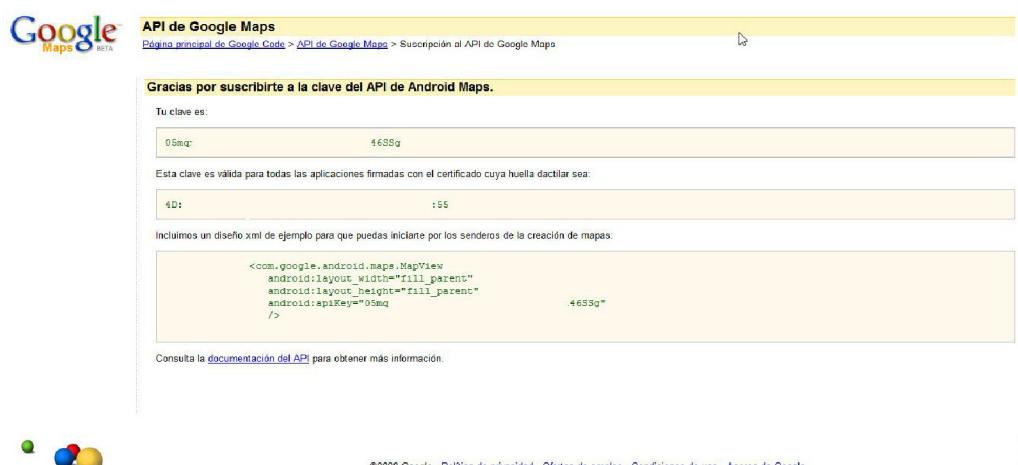


Figura 3.6. Pantalla de confirmacion del proceso.

Debemos guardar bien dicha información, ya que no será necesario generar una nueva clave cada vez que queramos insertarla en una aplicación diferente.

3.1.3. MOSTRANDO GOOGLE MAPS

Vamos a crear una aplicación que nos va a mostrar los mapas de *Google Maps* en pantalla. Para ello creamos un nuevo proyecto de Android y rellenamos con lo siguiente:

- **Project Name (nombre del proyecto):** SeasGoogleMaps
- **Build Target (objetivo de compilación):** Android 2.1 (Google APIs)
- **Application name (nombre de la aplicación):** Seas, GoogleMaps
- **Package name (nombre del paquete):** com.seas.ejemplo.GoogleMaps
- **Create Activity (crear actividad):** MapsActivity
- **Min SDK Version:** 8

Lo primero, para poder usar el API de *Google Maps* en nuestra aplicación debemos modificar el `AndroidManifest.xml` y añadir las librerías necesarias y los permisos. De momento empezaremos dando los permisos de acceso a Internet

```
<?xml version= "1.0" encoding= "utf-8"?>
<manifest xmlns:android=
"http://schemas.android.com/apk/res/android"
```

```

        package= "com.seas.ejemplo.GoogleMaps" android:versionCode=
"1"
        android:versionName= "1.0">
<uses-sdk android:minSdkVersion= "8" />

<application android:icon= "@drawable/icon" android:label=
"@string/app_name">
    <uses-library android:name= "com.google.android.maps" />
    <activity android:name= ".MapsActivity" android:label=
"@string/app_name">
        <intent-filter>
            <action android:name=
"android.intent.action.MAIN" />
            <category android:name=
"android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
</application>

<uses-permission android:name= "android.permission.INTERNET"
/>
</manifest>
```

Para poder ver los mapas debemos modificar el main.xml, que se encuentra en res/layout/ y añadir en él el API Key obtenido en el punto anterior:

```

<?xml version= "1.0" encoding= "utf-8"?>
<RelativeLayout xmlns:android=
"http://schemas.android.com/apk/res/android"
    android:layout_width= "fill_parent"
    android:layout_height= "fill_parent">

    <com.google.android.maps.MapView
        android:id= "@+id/mapView"
        android:layout_width= "fill_parent"
        android:layout_height= "fill_parent"
        android:enabled= "true"
        android:clickable= "true"
        android:apiKey= "Mi_ApiKey" <!-- introducimos la apiKey
obtenida en internet -->
    />
</RelativeLayout>
```

En el atributo apiKey, debemos insertar el que hemos obtenido anteriormente a través de la página del proceso de registro. Si no realizamos este paso, no se visualizarán los mapas.

En el archivo MapsActivity.java, modificaremos la clase para que extienda de la clase MapActivity, en vez de que lo haga de la clase Activity:

```

package com.seas.ejemplo.GoogleMaps;

import android.os.Bundle;

import com.google.android.maps.MapActivity;
```

```
public class MapsActivity extends MapActivity {  
    /** se crea cuando se llama a la activity. */  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.main);  
    }  
  
    @Override  
    protected boolean isRouteDisplayed() {  
        return false;  
    }  
}
```

Hay que observar que si extendemos la clase de `MapActivity`, es necesario sobrescribir el método `isRouteDisplayed()`. Lo más sencillo es establecer el método para que devuelva falso. Si ejecutamos nuestra aplicación, vemos como nos muestra el mapa.



Figura 3.7. Visión de Seas Google Maps.

Ahora le vamos a ir añadiendo funcionalidades a nuestra aplicación. Lo primero será el zoom. Para ello debemos añadir primero un `LinearLayout` al `main.xml`:

```
<LinearLayout android:id= "@+id/zoom"
    android:layout_width= "wrap_content"
    android:layout_height= "wrap_content"
    android:layout_alignParentBottom= "true"
    android:layout_centerHorizontal= "true"
    />
```

Y también debemos modificar nuestra clase y añadir los siguientes imports:

```
import com.google.android.maps.MapView.LayoutParams;
import android.view.View;

import android.widget.LinearLayout;
```

Y añadir en el método `onCreate()` lo siguiente:

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    mapView = (MapView) findViewById(R.id.mapView);
    LinearLayout zoomLayout =
    (LinearLayout)findViewById(R.id.zoom);
    View zoomView = mapView.getZoomControls();

    zoomLayout.addView(zoomView,
        new LinearLayout.LayoutParams(
            LayoutParams.WRAP_CONTENT,
            LayoutParams.WRAP_CONTENT));
    mapView.displayZoomControls(true);
}
```

Ésta es la forma antigua de hacerlo, y podrá verlo insertado en alguna aplicación, y aunque todavía sigue siendo aceptada y funcional, hay una forma más rápida de realizar este paso.

Únicamente debemos reemplazar el código anterior por la siguiente línea en el método `onCreate()`:

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    mapView = (MapView) findViewById(R.id.mapView);

    mapView.setBuiltInZoomControls(true);
}
```

Si ejecutamos ahora la aplicación, veremos los botones de zoom en la parte inferior de la pantalla.



Figura 3.8. Mapas con zoom.

Otra forma de añadir las funciones de zoom es mediante los métodos `zoomIn()` y `zoomOut()` con la clase `MapController()`. Se le pueden asignar a los eventos cuando pulsamos una tecla del teclado:

```
public boolean onKeyDown(int keyCode, KeyEvent event)
{
    MapController mc = mapView.getController();
    switch (keyCode)
    {
        case KeyEvent.KEYCODE_1:
            mc.zoomOut();
            break;
        case KeyEvent.KEYCODE_3:
            mc.zoomIn();
            break;
        case KeyEvent.KEYCODE_5:
            mapView.setStreetView(true);
            break;
        case KeyEvent.KEYCODE_6:
            mapView.setStreetView(false);
            break;
        case KeyEvent.KEYCODE_7:
            mapView.setSatellite(true);
            break;
        case KeyEvent.KEYCODE_8:
            mapView.setSatellite(false);
            break;
    }
    return super.onKeyDown(keyCode, event);
}
```

De paso hemos añadido a las teclas “5”, “6”, “7” y “8” las funciones de “activar” y “desactivar” las vistas de StreetView y Satelite que proporciona Maps.



Figura 3.9. Vista Satelite en Google Maps.

Seguimos implementando funciones. Vamos a intentar hacer que por defecto se abra una localización insertada manualmente.

Existen varias formas de obtener las coordenadas de un punto determinado con *Google Maps*. Una de ellas es situarse sobre el punto que queremos conocer las coordenadas y pulsar sobre él con el botón derecho y elegir la opción de “¿Qué hay aquí?”:



Figura 3.10. Seleccionamos la opción de ¿Qué hay aquí?.

Una vez elegida, veremos que sobre la barra de Google aparecen las coordenadas que necesitamos localizar:

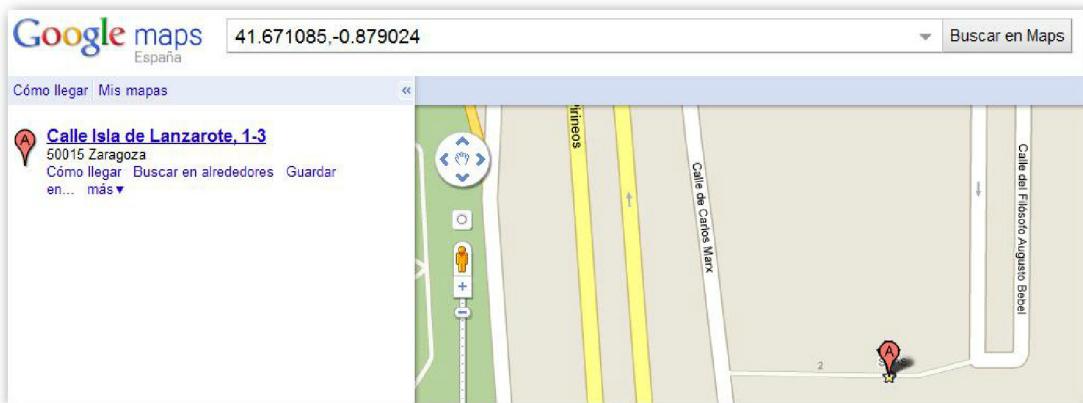


Figura 3.11. Imagen con las coordenadas del punto buscado.

La otra forma de encontrar las coordenadas es mediante una función javascript. Para ello, debemos situarnos sobre el punto que queremos obtener las coordenadas y con el botón derecho pulsamos sobre “Centrar el mapa aquí”:



Figura 3.12. Imagen con la opción a seleccionar.

Una vez realizado esto, sobre la barra del navegador debemos insertar la siguiente instrucción:

```
javascript:void(prompt(' ',gApplication.getMap().getCenter()));
```

Una vez insertada, el navegador nos devuelve la respuesta del servidor mediante un pop up con las coordenadas:

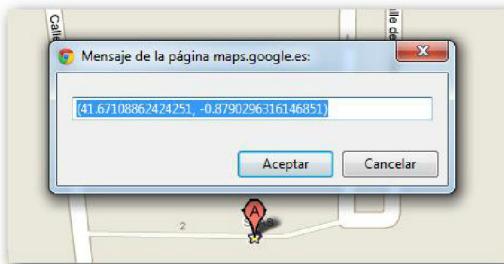


Figura 3.13. Respuesta javascript con las coordenadas del mapa.

Una vez tenemos las coordenadas, las insertaremos en el método `animateTo()` de la clase `MapController`:

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    mapView = (MapView) findViewById(R.id.mapView);
    mapView.setBuiltInZoomControls(true);

    mc = mapView.getController();
    String coordinates[] = {"41.67109", "-0.87867"};
    double lat = Double.parseDouble(coordinates[0]);
    double lng = Double.parseDouble(coordinates[1]);

    p = new GeoPoint(
        (int) (lat * 1E6),
        (int) (lng * 1E6));

    mc.animateTo(p);
    mc.setZoom(17);
    mapView.invalidate();
}
```

Lo primero, hemos obtenido un controlador de la instancia de `MapView` y se lo hemos asignado a un objeto `MapController` (`mc`). Utilizamos un objeto `GeoPoint` para representar un lugar determinado. Debemos tener en cuenta que para esta clase la latitud y la longitud de una ubicación están representadas en grados micro. Esto significa que se almacenan como valores enteros. Por lo tanto, deberemos multiplicarlos por 1E6. El método `setZoom()` nos permite especificar el nivel de zoom en el que se mostrará el mapa.

Insertando estas coordenadas nos abre por defecto la ubicación de Seas en Zaragoza.



Figura 3.14. Posición predefinida cuando abrimos la aplicación.

Ahora vamos a insertar una chincheta para que pueda agregar marcadores que indiquen los puntos de interés. Para ello debemos crear una imagen de una chincheta en el directorio res/drawable

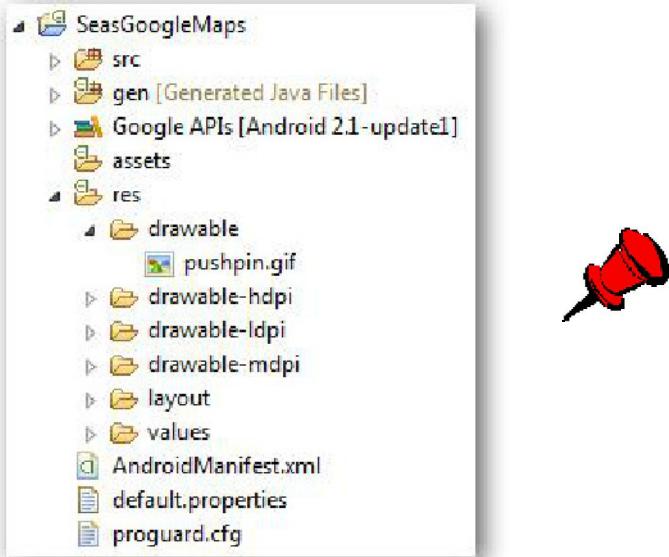


Figura 3.15. Imagen de la chincheta en el directorio drawable.

Para un mejor efecto, debe hacer que el fondo de la imagen sea transparente, de manera que no bloquee las partes del mapa cuando la imagen se añada a él.

Para realizar esto podemos buscar la imagen en internet, y una vez encontrada, pulsamos con el botón derecho sobre ella y seleccionamos “*Guardar imagen como...*”

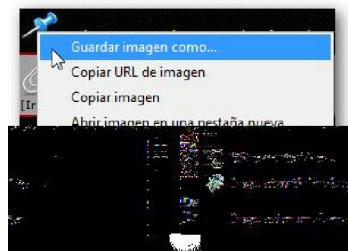


Figura 3.16. Seleccionamos la opción Guardar imagen como...

En ese momento se abrirá un diálogo para poder nombrar la imagen y guardarla donde deseemos:

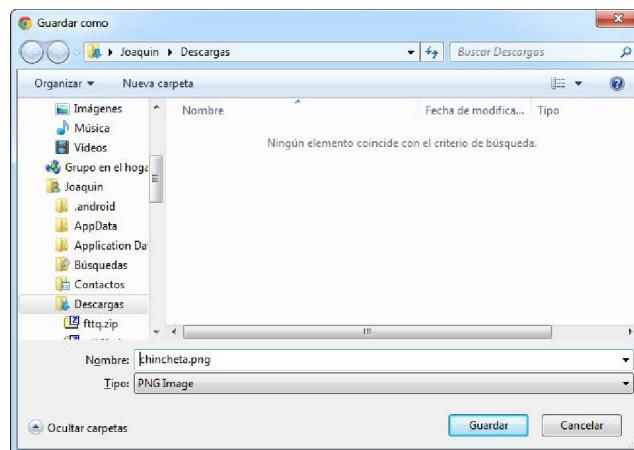


Figura 3.17. Ventana diálogo de guardar como...

Una vez guardada la imagen continuamos con el código para insertar la imagen. Para agregarlo, primero debemos definir una clase que extienda de la clase Overlay.

```
class MapOverlay extends com.google.android.maps.Overlay
{
    @Override
    public boolean draw(Canvas canvas, MapView mapView,
        boolean shadow, long when)
    {
        super.draw(canvas, mapView, shadow);

        // convertimos el GeoPoint en pixels
        Point screenPts = new Point();
        mapView.getProjection().toPixels(p, screenPts);
```

```

    // añadimos la chincheta
    Bitmap bmp = BitmapFactory.decodeResource(
        getResources(), R.drawable.pushpin);
    canvas.drawBitmap(bmp, screenPts.x, screenPts.y-50,
null);
    return true;
}
}
}

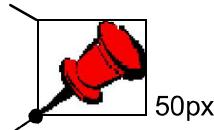
```

En la clase `MapOverlay` que acabamos de definir, sobrescribimos el método `draw()` para que pueda dibujar la imagen chincheta en el mapa. Hay que darse cuenta que tenemos que traducir la posición geográfica (representada por un objeto `GeoPoint`, `p`) en coordenadas de pantalla.

Como lo que queremos es, que el extremo puntiagudo del alfiler sea el que nos indique la posición de la ubicación, deberemos restar la altura de la imagen (50 píxeles en nuestro caso) de la coordenada del punto, y que señale a la imagen en ese lugar.

Punto para dibujar la imagen

`screenPts.x, screenPts.y-50`



`screenPts.x, screenPts.y`

Localización del punto

Para agregar el marcador, debemos crear una instancia de la clase `MapOverlay` y agregarlo a la lista de plantillas disponibles en el objeto `MapView`:

```

// Añadir la chincheta
MapOverlay mapOverlay = new MapOverlay();
List<Overlay> listOfOverlays = mapView.getOverlays();
// limpiar puntos
listOfOverlays.clear();

listOfOverlays.add(mapOverlay);

```

La clase quedaría de la siguiente manera con todo lo insertado hasta este momento:

```

package com.seas.ejemplo.GoogleMaps;

import java.util.List;

import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.graphics.Canvas;
import android.graphics.Point;
import android.os.Bundle;
import android.view.KeyEvent;

```

```
import com.google.android.maps.GeoPoint;
import com.google.android.maps.MapActivity;
import com.google.android.maps.MapController;
import com.google.android.maps.MapView;
import com.google.android.maps.Overlay;

public class MapsActivity extends MapActivity {
    MapView mapView;
    MapController mc;
    GeoPoint p;

    class MapOverlay extends com.google.android.maps.Overlay
    {
        @Override
        public boolean draw(Canvas canvas, MapView mapView,
                            boolean shadow, long when)
        {
            super.draw(canvas, mapView, shadow);

            // convertimos el GeoPoint en pixels
            Point screenPts = new Point();
            mapView.getProjection().toPixels(p, screenPts);

            // añadimos la chincheta
            Bitmap bmp = BitmapFactory.decodeResource(
                getResources(), R.drawable.pushpin);
            canvas.drawBitmap(bmp, screenPts.x, screenPts.y-50,
null);
            return true;
        }
    }
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        mapView = (MapView) findViewById(R.id.mapView);
        mapView.setBuiltInZoomControls(true);

        mc = mapView.getController();
        String coordinates[] = {"41.67109", "-0.87867"};
        double lat = Double.parseDouble(coordinates[0]);
        double lng = Double.parseDouble(coordinates[1]);

        p = new GeoPoint(
            (int) (lat * 1E6),
            (int) (lng * 1E6));

        mc.animateTo(p);
        mc.setZoom(17);

        // Añadimos la chincheta a la posicion
        MapOverlay mapOverlay = new MapOverlay();
        List<Overlay> listOfOverlays = mapView.getOverlays();
        listOfOverlays.clear();
        listOfOverlays.add(mapOverlay);

        mapView.invalidate();
    }
}
```

```

public boolean onKeyDown(int keyCode, KeyEvent event)
{
    MapController mc = mapView.getController();
    switch (keyCode)
    {
        case KeyEvent.KEYCODE_1:
            mc.zoomOut();
            break;
        case KeyEvent.KEYCODE_3:
            mc.zoomIn();
            break;
        case KeyEvent.KEYCODE_5:
            mapView.setStreetView(true);
            break;
        case KeyEvent.KEYCODE_6:
            mapView.setStreetView(false);
            break;
        case KeyEvent.KEYCODE_7:
            mapView.setSatellite(true);
            break;
        case KeyEvent.KEYCODE_8:
            mapView.setSatellite(false);
            break;
    }
    return super.onKeyDown(keyCode, event);
}

@Override
protected boolean isRouteDisplayed() {
    return false;
}
}

```

Ahora vamos a hacer que nos devuelva una posición (latitud y longitud) cuando pulsemos sobre el mapa.

Como hemos añadido una superposición en el mapa, puede sobrescribir el método `onTouchEvent()` de la clase `Overlay`. Este se activará cada vez que toquemos el mapa.

Este método tiene dos parámetros, `MotionEvent` y `MapView`. Usando el parámetro `MotionEvent`, se puede saber si el usuario ha levantado el dedo de la pantalla utilizando el método `getAction()`. Con el siguiente código, sabremos si el usuario ha tocado y luego ha levantado el dedo, y se mostrará la latitud y la longitud de la ubicación tocada:

```

import android.view.MotionEvent;
import android.widget.Toast;

class MapOverlay extends com.google.android.maps.Overlay
{
    @Override
    public boolean onTouchEvent(MotionEvent event, MapView
mapView)
    {
        // cuando el usuario levante el dedo

```

```
    if (event.getAction() == 1) {  
        // coordenadas x,y del punto seleccionado en la  
        // pantalla  
        GeoPoint p = mapView.getProjection().fromPixels(  
            (int) event.getX(),  
            (int) event.getY());  
        Toast.makeText(getApplicationContext(),  
            p.getLatitudeE6() / 1E6 + "," +  
            p.getLongitudeE6() /1E6 ,  
            Toast.LENGTH_SHORT).show();  
    }  
    return false;  
}  
}
```

Si ejecutamos nuestra aplicación podremos ver lo siguiente:



Figura 3.18. Visualización de la latitud y la longitud de un punto tocado en el mapa.

Si conocemos la latitud y la longitud de un lugar, se puede averiguar su dirección mediante un proceso conocido como geocodificación. *Google Maps* en Android soporta esta vía mediante la clase Geocoder. El código siguiente muestra cómo se puede averiguar la dirección en vez de la longitud y latitud, de un lugar que acabamos de tocar con el método `getFromLocation()`:

```
import java.io.IOException;
class MapOverlay extends com.google.android.maps.Overlay
{
    @Override
    public boolean onTouchEvent(MotionEvent event, MapView
mapView)
    {
        if (event.getAction() == 1) {
            GeoPoint p = mapView.getProjection().fromPixels(
                (int) event.getX(),
                (int) event.getY());

            Geocoder geoCoder = new Geocoder(
                getBaseContext(), Locale.getDefault());
            try {
                List<Address> addresses =
geoCoder.getFromLocation(
                    p.getLatitudeE6() / 1E6,
                    p.getLongitudeE6() / 1E6, 1);

                String add = "";
                if (addresses.size() > 0)
                {
                    for (int i=0;
i<addresses.get(0).getMaxAddressLineIndex();
                        i++)
                        add += addresses.get(0).getAddressLine(i)
+ "\n";
                }

                Toast.makeText(getApplicationContext(), add,
Toast.LENGTH_SHORT).show();
            }
            catch (IOException e) {
                e.printStackTrace();
            }
            return true;
        }
        else
            return false;
    }
}
```

El código para mostrar la longitud y la latitud ha sido comentado para poder mostrar la nueva funcionalidad.



Figura 3.19. Mostrando geocodificación en Google Maps.

Si conocemos la dirección de una ubicación, pero queremos saber su latitud y longitud, podemos hacerlo a través de geocodificación inversa. Una vez más, se puede utilizar la clase `Geocoder` para este propósito. El código siguiente muestra cómo se puede encontrar la ubicación exacta de la Plaza del Pilar de Zaragoza, mediante el método `getFromLocationName()`. Para ello debemos cambiar las siguientes líneas de código en el método `onTouchEvent()`:

```
// con este try, cuando levante el dedo mostrara la posicion
// en pantalla
try {
    List<Address> addresses = geoCoder.getFromLocationName(
        "Plaza del Pilar, Zaragoza", 5);
    String add = "";
    if (addresses.size() > 0) {
        p = new GeoPoint(
            (int) (addresses.get(0).getLatitude() * 1E6),
            (int) (addresses.get(0).getLongitude() * 1E6));
        mc.animateTo(p);
        mapView.invalidate();
    }
}
```

Si ejecutamos la aplicación, nos mostrará primero la posición que tenemos predefinida de inicio, y en cuanto toquemos la pantalla y levantemos el dedo, nos buscará la Plaza del Pilar, de Zaragoza.



Figura 3.20. Posición de la Plaza del Pilar.



La aplicación completa “**SeasGoogleMaps**” se puede descargar desde la plataforma de estudio.

3.1.4. APPLICACIÓN LOCALIZACIÓN GPS

Con esta aplicación vamos a capturar la posición GPS de donde nos encontramos, usando la antena GPS del dispositivo.

Debemos crear un nuevo proyecto Android e insertar los siguientes campos:

- **Project Name (nombre del proyecto):** SeasLocationGPS
- **Build Target (objetivo de compilación):** Android 2.1 (Google Apis)
- **Application name (nombre de la aplicación):** Seas, LocalizacionGPS
- **Package name (nombre del paquete):** com.seas.ejemplo. LocationGPS
- **Create Activity (crear actividad):** LocationGPS
- **Min SDK Version:** 7

El programa lanzará un Thread que buscará la posición GPS. Y usamos un hilo para poder mostrar mientras que está buscando, con un ProgressDialog.

Lo primero que haremos para poder usar la señal de GPS es añadir al `AndroidManifest.xml` el permiso `"ACCESS_FINE_LOCATION"`:

```
<?xml version= "1.0" encoding= "utf-8"?>
<manifest xmlns:android=
"http://schemas.android.com/apk/res/android"
    package= "com.seas.ejemplo.LocationGPS"
    android:versionCode= "1"
    android:versionName= "1.0">
    <uses-sdk android:minSdkVersion= "7" />

    <uses-permission android:name=
"android.permission.ACCESS_FINE_LOCATION" />
    <application android:icon= "@drawable/icon" android:label=
"@string/app_name">
        <activity android:name= ".LocationGPS"
            android:label= "@string/app_name">
            <intent-filter>
                <action android:name= "android.intent.action.MAIN"
/>
                <category android:name=
"android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Vamos a definir un botón que será el encargado de realizar la búsqueda de la señal en el momento que lo pulsemos. Debemos modificar el archivo `main.xml` del directorio `res/layout`:

```
<?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:paddingTop= "10px"
        android:paddingBottom= "10px"
        android:paddingLeft= "10px"
        android:layout_width= "fill_parent"
        android:layout_height= "wrap_content"
        android:text= "@string/hello"
    />
    <Button android:id= "@+id/btsearch"
        android:text= "@string/btsearch"
        android:layout_width= "wrap_content"
        android:layout_height= "wrap_content" />
    <TextView android:id= "@+id/outlat"
        android:paddingLeft= "10px"
        android:paddingTop= "10px"
        android:layout_width= "fill_parent"
```

```

        android:layout_height= "wrap_content"
    />
<TextView  android:id= "@+id/outlong"
    android:paddingLeft= "10px"
    android:paddingTop= "10px"
    android:layout_width= "fill_parent"
    android:layout_height= "wrap_content"
/>

</LinearLayout>

```

Definiremos unas cuantas variables strings en el archivo strings.xml que serán llamadas posteriormente dependiendo del estado en el que se encuentre nuestro dispositivo:

```

<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name= "hello">Localizacion GPS! </string>
    <string name= "app_name">Seas LocalizacionGPS </string>
    <string name= "gps_signal_not_found">Señal de GPS no encontrada. Compruebe el estado del
    GPS </string>
    <string name= "gps_signal_found">Señal de GPS encontrada
    correctamente </string>
    <string name= "search">Buscando... </string>
    <string name= "search_signal_gps">Buscando señal GPS </string>
    <string name= "btsearch">Buscar posicion GPS </string>

</resources>

```

Una vez realizado esto, en nuestra clase LocationGPS.java necesitamos un objeto LocationManager, al que le asignaremos un escuchador (LocationListener) para que nos informe cada vez que cambia el estado del GPS mediante el método onLocationChanged(). Cada vez que registremos el escuchador, asignaremos el tiempo y la distancia mínima para que refresque el estado de la señal GPS con el método requestLocationUpdates():

```

package com.seas.ejemplo.LocationGPS;

import android.app.Activity;
import android.app.ProgressDialog;
import android.content.Context;
import android.content.DialogInterface;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.os.Bundle;
import android.os.Handler;
import android.os.Looper;
import android.os.Message;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;

public class LocationGPS extends Activity implements Runnable {

```

```
private ProgressDialog pd;

LocationManager mLocationManager;
Location mLocation;
MyLocationListener mLocationListener;

Location currentLocation = null;

TextView outlat;
TextView outlong;

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    outlat = (TextView) findViewById(R.id.outlat);
    outlong = (TextView) findViewById(R.id.outlong);

    Button btsearch = (Button) findViewById(R.id.btsearch);
    btsearch.setOnClickListener(new View.OnClickListener() {

        public void onClick(View view) {
            writeSignalGPS();
        }
    });
}

private void setCurrentLocation(Location location) {
    currentLocation = location;
}

private void writeSignalGPS() {
    DialogInterface.OnCancelListener dialogCancel = new
    DialogInterface.OnCancelListener() {

        public void onCancel(DialogInterface dialog) {
            Toast.makeText(
                getBaseContext(),
                getResources().getString(R.string.gps_signal_not_found),
                Toast.LENGTH_LONG).show();
            handler.sendEmptyMessage(0);
        }
    };
    pd = ProgressDialog.show(this,
        this.getResources().getString(R.string.search), this
        .getResources().getString(R.string.search_signal_gps),
        true, true, dialogCancel);
    Thread thread = new Thread(this);
}
```

```
        thread.start();

    }

    public void run() {

        mLocationManager = (LocationManager)
getSystemService(Context.LOCATION_SERVICE);

        if
(mLocationManager.isProviderEnabled(LocationManager.GPS_PROVIDER)) {

            // ejecutamos el bucle de mensajes
Looper.prepare();

            mLocationListener = new MyLocationListener();

            // definimos el tiempo de refresco minimo y la
distancia minima
            mLocationManager.requestLocationUpdates(
                LocationManager.GPS_PROVIDER, 0, 0,
mLocationListener);
            Looper.loop();
            Looper.myLooper().quit();

        } else {

            // cuadro de texto emergente si no encuentra señal
de GPS
            Toast.makeText(getApplicationContext(),
getResources().getString(R.string.gps_signal_not_found),
Toast.LENGTH_LONG).show();

        }
    }

    private Handler handler = new Handler() {
    @Override
    public void handleMessage(Message msg) {
        pd.dismiss();
        mLocationManager.removeUpdates(mLocationListener);
        if (currentLocation != null) {
            outlat.setText("Latitud: " +
currentLocation.getLatitude());
            outlong.setText("Longitud: " +
currentLocation.getLongitude());
        }
    }
};

    private class MyLocationListener implements LocationListener {
        public void onLocationChanged(Location location) {
            if (location != null) {
                Toast.makeText(getApplicationContext(),
getResources().getString(R.string.gps_signal_found),
Toast.LENGTH_LONG).show();
                setCurrentLocation(location);
            }
        }
    }
}
```

```
        handler.sendMessage(0);
    }

    public void onProviderDisabled(String provider) {
        // TODO Auto-generated method stub
    }

    public void onProviderEnabled(String provider) {
        // TODO Auto-generated method stub
    }

    public void onStatusChanged(String provider, int status,
Bundle extras) {
        // TODO Auto-generated method stub
    }
}
```

En el método `run()` usaremos el método `requestLocationUpdates()` del objeto `LocationManager` para registrar el objeto `LocationListener`.

Debemos recordar que en Android podemos manejar las localizaciones a partir de Providers. `GPS_PROVIDER` es nuestro GPS y en esta aplicación estamos pidiéndole a él que nos diga la posición. Además, para trabajar con Threads es necesario llamar al principio del método `run()`, al método `looper.prepare()` y a `looper.loop()` al final del mismo. Esta clase se utiliza para ejecutar un bucle de mensajes en un Thread.

Invocaremos al objeto `Handler` cuando ya tengamos la localización y será el que cierre la ventana de proceso y establecerá los valores de la longitud y la latitud en los `TextView`. Esto último es importante, ya que no podemos establecer los valores de los `TextView` desde el método `onLocationChanged()` del `LocationListener`, ya que para establecer los valores debe hacerse desde un Thread principal.

Si ejecutamos la aplicación sobre el emulador, debemos pasarle la posición a través del visor del DDMS. Lo primero, ejecutamos la aplicación y pulsamos en el botón “*Buscar posición GPS*”.



Figura 3.21. Emulador ejecutando la aplicación Seas LocalizacionGPS.

En ese momento, en la ventana del DDMS, en la pestaña “Emulator Control” introducimos las coordenadas y pulsamos sobre “Send”:

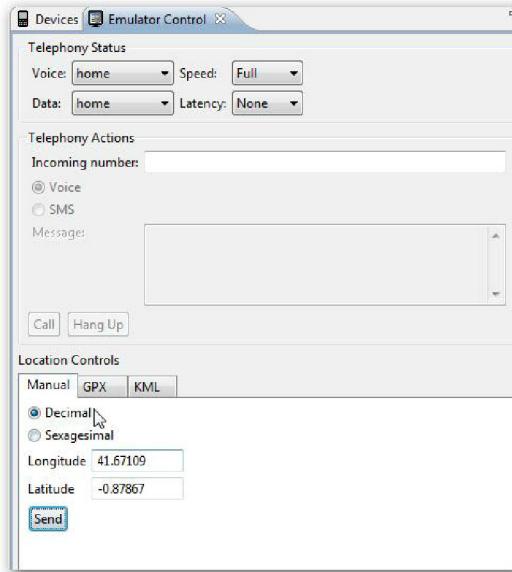


Figura 3.22. Enviamos las coordenadas al emulador.

Si observamos en ese momento de nuevo la ventana del emulador, veremos la siguiente ventana:

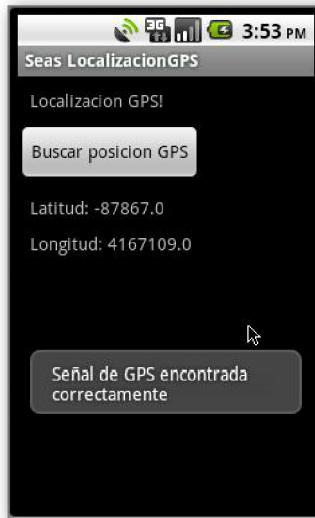


Figura 3.23. Ventana del emulador cuando se le envían las coordenadas desde el DDMS.



El ejemplo completo “**SeasLocationGPS**” se puede descargar desde la plataforma de estudio.

3.2. SENSORES

Todos los dispositivos móviles actuales disponen de diferentes sensores que no dan una información determinada. Android permite acceder a esos sensores internos del dispositivo, utilizando las clases del paquete android.hardware, Sensor, SensorEvent y SensorManager, y de la interfaz SensorEventListener. La clase Sensor acepta distintos tipos de sensores, y estos varían en función del aparato utilizado, pero los que implementa la clase son los siguientes:

- **TYPE GRAVITY**: mide la gravedad.
- **TYPE ROTATION VECTOR**: sensor de rotación del dispositivo.
- **TYPE LIGHT**: indica la luminosidad de alrededor.
- **TYPE PROXIMITY**: mide la distancia entre el sensor y un objeto.
- **TYPE MAGNETIC FIELD**: devuelve la atracción magnética.
- **TYPE ORIENTATION**: mide el balanceo. Se recomienda usar `sensorManager.getOrientation()`.
- **TYPE ACCELEROMETER**: mide la aceleración en los ejes
- **TYPE PRESSURE**: mide la presión atmosférica.
- **TYPE TEMPERATURE**: mide la temperatura.

Vamos a visualizar un pequeño ejemplo donde se implementarán las clases anteriormente nombradas y si disponemos de un dispositivo Android, podremos observar los sensores disponibles en nuestro dispositivo.

Para ello vamos a crear un nuevo proyecto Android e insertamos lo siguiente:

- **Project Name (nombre del proyecto)**: SeasSensorTest
- **Build Target (objetivo de compilación)**: Android 2.1 (Google APIs)
- **Application name (nombre de la aplicación)**: Seas, Test Sensores
- **Package name (nombre del paquete)**: com.seas.ejemplo.SensorTest
- **Create Activity (crear actividad)**: SensorTest
- **Min SDK Version: 7**

Tenemos que tener en cuenta que la clase SensorManager nos va a devolver actualizaciones con muchísima frecuencia, incluso llegando a cientos por segundo.

Vamos a insertar un TextView que lo llamaremos “output” en el res/layout/main.xml que será donde sacaremos los resultados de los sensores.

```
<?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/output"
        android:layout_width= "fill_parent"
        android:layout_height= "wrap_content" />
</LinearLayout>
```

Lo primero, llamamos al método `getSystemService()`, y definimos el TextView para sacar la información por pantalla, y un `sensorList` de `List<Sensor>`.

```
public class SensorTest extends Activity implements
SensorEventListener {

    private SensorManager mgr;
    private TextView output;
    private List<Sensor> sensorList;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        // ...
        mgr = (SensorManager) getSystemService(SENSOR_SERVICE);
        output = (TextView) findViewById(R.id.output);
    }

    @Override
    protected void onResume() {
        super.onResume();

        // Cada sensor se registra por separado
        sensorList =
            mgr.getSensorList(Sensor.TYPE_ACCELEROMETER);
        Sensor acelerometerSensor = sensorList.get(0);
        mgr.registerListener(this, acelerometerSensor,
SensorManager.SENSOR_DELAY_NORMAL);
        /*
        sensorList =
            mgr.getSensorList(Sensor.TYPE_MAGNETIC_FIELD);
        Sensor magneticSensor = sensorList.get(0);
        mgr.registerListener(this, magneticSensor,
SensorManager.SENSOR_DELAY_NORMAL);

        sensorList = mgr.getSensorList(Sensor.TYPE_ORIENTATION);
        Sensor orientationSensor = sensorList.get(0);
        mgr.registerListener(this, orientationSensor,
SensorManager.SENSOR_DELAY_NORMAL);
```

```
        */  
    }  
  
    @Override  
    protected void onPause() {  
        super.onPause();  
        // paramos la actualizacion para ahorrar bateria  
        mgr.unregisterListener(this);  
    }  
  
    public void onAccuracyChanged(Sensor sensor, int accuracy) {  
        // TODO Auto-generated method stub  
    }  
  
    public void onSensorChanged(SensorEvent event) {  
        StringBuilder builder = new StringBuilder();  
        builder.append("--- SENSOR ---");  
        builder.append("\nName: ");  
        Sensor sensor = event.sensor;  
        builder.append(sensor.getName());  
        builder.append("\nType: ");  
        builder.append(sensor.getType());  
        builder.append("\nVendor: ");  
        builder.append(sensor.getVendor());  
        builder.append("\nVersion: ");  
        builder.append(sensor.getVersion());  
        builder.append("\nMaximum Range: ");  
        builder.append(sensor.getMaximumRange());  
        builder.append("\nPower: ");  
        builder.append(sensor.getPower());  
        builder.append("\nResolution: ");  
        builder.append(sensor.getResolution());  
  
        builder.append("\n\n--- EVENT ---");  
        builder.append("\nAccuracy: ");  
        builder.append(event.accuracy);  
        builder.append("\nTimestamp: ");  
        builder.append(event.timestamp);  
        builder.append("\nValues:\n");  
        for (int i = 0; i < event.values.length; i++) {  
            // ...  
            builder.append("    [");  
            builder.append(i);  
            builder.append("] = ");  
            builder.append(event.values[i]);  
            builder.append("\n");  
        }  
        output.setText(builder);  
    }  
}
```

Tenemos que registrar cada tipo de sensor por separado para obtener información de todos ellos. El método `registerListener()` tiene cuatro argumentos, el primero es la instancia de la clase que implementa el `SensorEventListener`, y con el tercero indicaremos al sistema con qué frecuencia nos gustaría recibir actualizaciones del sensor. Esto sirve para que el sistema calcule cuánta atención necesitan los sensores, pero no nos asegura una frecuencia concreta.

Con el método `onSensorChanged()` mostramos en pantalla cada uno de los atributos que tiene cada sensor, cada vez que se desata el evento. Si queremos ver otros sensores, simplemente descomentemos el sensor que queramos obtener de nuestro dispositivo.

Si ejecutamos la aplicación en un dispositivo veremos el siguiente resultado.

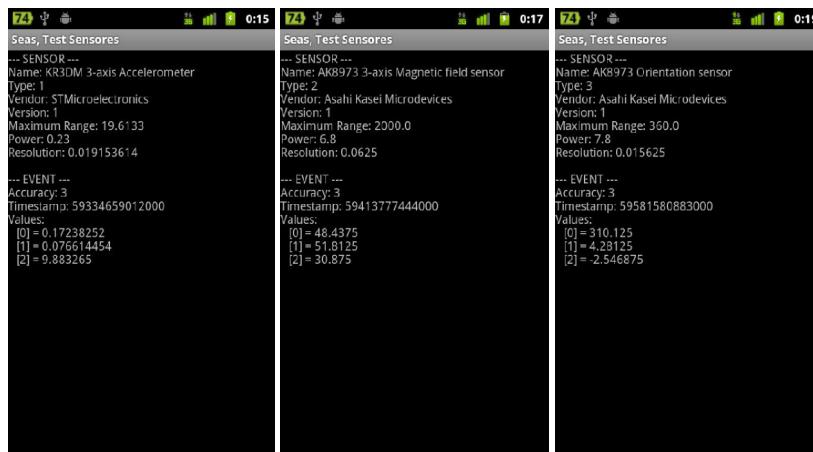


Figura 3.24. Visión de cada uno de los sensores.

Tenemos que tener en cuenta que todos los sensores devuelven una matriz de valores, y su tamaño depende del sensor. Por ejemplo, `TYPE_TEMPERATURE` devuelve un único valor, la temperatura en Celsius. Hay que recordar que no todos los dispositivos tienen todos los sensores, y alguno de ello podría dar error al probar.



La aplicación completa “**SeasSensorTest**” se puede descargar desde la plataforma de estudio.

3.3. BLUETOOTH

La tecnología bluetooth en estos momentos es un estándar y una forma de conexión sin cables entre dispositivos electrónicos. Esta tecnología fue desarrollada por *Ericsson* y después fue implementada por *IBM*, *Toshiba*, *Intel*, *Ericsson* y *Nokia*. Mientras tanto, el grupo de trabajo del Bluetooth ha sido enriquecido con muchos representantes de empresas como, por ejemplo, *Motorola*, *Lucent*, *VLSI*, *Qualcomm*, *Psion Dacom*, *Compaq*, *Xircom*, *Dell*, *3Com Palm* y *One2One*.

El Bluetooth funciona de una forma similar a la de las redes de telecomunicaciones, pero las frecuencias que ocupa son libres (la banda de los 2,45 GHz) y no están sujetas a licenciamiento. La velocidad de transmisión está entre los 720 kbps y 1 Mbps por segundo.

Simplemente como curiosidad de dónde surge el nombre: Harald Bluetooth (Harald "Dientes Azules") fue un gran rey Vikingo que unió Dinamarca y Noruega en el siglo X. Su nombre quedará ligado para siempre a la tecnología revolucionaria que permite a los aparatos electrónicos trabajar en conjunto sin necesidad de cables. Si nos fijamos un poco, el logotipo juega con la H y la B, iniciales del rey Vikingo.

Para ver el uso de esta tecnología en Android, vamos a realizar una aplicación que envía vía bluetooth al *HyperTerminal* del pc, un carácter determinado dependiendo del botón que se pulse sobre el dispositivo.

Para ello debemos realizar la sincronización entre el pc y el dispositivo por bluetooth, y mediante algún programa gratuito configurar el dispositivo como un puerto serie por bluetooth en nuestro equipo.

La idea es que una vez configurado y lanzado el programa de *HyperTerminal*, se quede a la escucha de que se active la conexión entre el dispositivo y el pc. Una vez realizada dicha configuración, al lanzar nuestra aplicación en el dispositivo con Android, se realizará la conexión entre ambos, y ya se podrán mandar los comandos mediante bluetooth del dispositivo al pc, mostrándose el resultado por la ventana del *HyperTerminal*.

Para poder configurar su dispositivo bluetooth como puerto serie COM, existen muchas formas y tutoriales disponibles en internet. Aquí lo vamos a realizar de la siguiente manera.

Lo primero que debemos hacer es ir al ícono del bluetooth y pulsamos sobre él con el botón derecho del ratón y seleccionamos: “Abrir la configuración de Bluetooth”

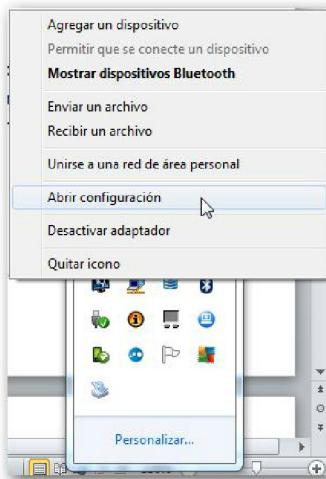


Figura 3.25. Opciones de Bluetooth en el icono del pc.

En la pestaña de “Puertos COM”, pulsamos sobre “Agregar...” y esperamos a que el sistema acabe de agregar el puerto. Una vez realizado esto, podemos “Aceptar” para cerrar la ventana. Sobre todo, debemos recordar el puerto que nos ha configurado, ya que más adelante necesitaremos introducirlo en el *HyperTerminal* (o *putty* en su defecto).

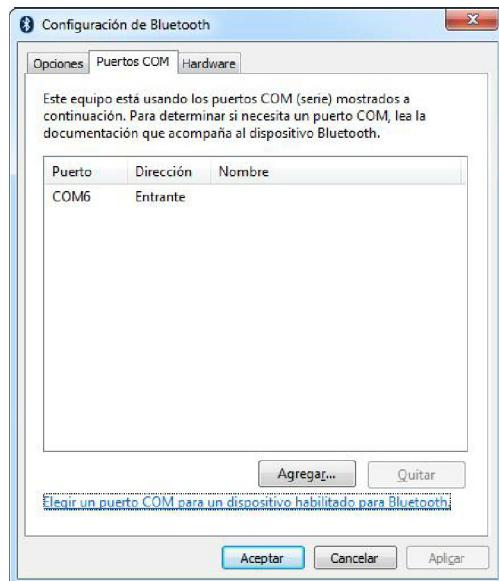


Figura 3.26. Ventana de configuración de dispositivo bluetooth.

Primero comentar, que Windows incluía en su sistema operativo una versión básica del programa de *HyperTerminal*. A partir de versiones posteriores de Windows XP, no viene instalado con el sistema y para realizar las mismas funciones de escucha, podemos descargarnos el programa gratuito “*putty*” de la página <http://www.putty.org/> que más adelante procederemos a configurarlo para que se quede a la espera de la conexión del dispositivo.

Ahora vamos a realizar nuestra aplicación. Para ello vamos a crear un nuevo proyecto de Android e insertaremos los siguientes datos:

- **Project Name (nombre del proyecto):** SeasBluetoothSend
- **Build Target (objetivo de compilación):** Android 2.2
- **Application name (nombre de la aplicación):** Seas, Boton Bluetooth
- **Package name (nombre del paquete):** com.seas.ejemplo.BluetoothSend
- **Create Activity (crear actividad):** BluetoothSend
- **Min SDK Version:** 8

Una vez creado el proyecto, vamos a modificar el archivo `main.xml` de la carpeta `res/layout` para insertar cuatro botones en la ventana principal:

```
<?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/text" android:layout_width=
"fill_parent"
        android:layout_height= "wrap_content" android:text=
"Bluetooth" />
    <LinearLayout xmlns:android=
"http://schemas.android.com/apk/res/android"
        android:orientation= "horizontal" android:layout_width=
"fill_parent"
        android:layout_height= "fill_parent">
        <Button android:text= "Letra A" android:id=
"@+id/Button01"
            android:layout_width= "wrap_content"
        android:layout_height= "wrap_content">
        </Button>
        <Button android:text= "Letra B" android:id=
"@+id/Button02"
            android:layout_width= "wrap_content"
        android:layout_height= "wrap_content">
        </Button>
        <Button android:text= "Letra C" android:id=
"@+id/Button03"
            android:layout_width= "wrap_content"
        android:layout_height= "wrap_content">
        </Button>
```

```
<Button android:text= "Letra D" android:id= "@+id/Button04"
        android:layout_width= "wrap_content"
        android:layout_height= "wrap_content">
    </Button>
</LinearLayout>
```

De esta manera veremos algo parecido a esto en nuestra pantalla del terminal:

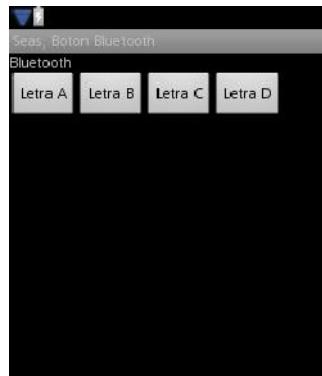


Figura 3.27. Diseño de la ventana principal de la aplicación.

Insertamos el siguiente código en la clase java:

```
package com.seas.ejemplo.BluetoothSend;

import java.io.IOException;
import java.io.OutputStream;
import java.util.UUID;

import android.app.Activity;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothSocket;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.Toast;

public class BluetoothSend extends Activity {
    public Button b1;
    public Button b2;
    public Button b3;
    public Button b4;

    private static final String TAG = "THINBTCLIENT";
    private static final boolean D = true;
    private BluetoothAdapter mBluetoothAdapter = null;
    private BluetoothSocket btSocket = null;
```

```

private OutputStream outStream = null;

    // Serial Port Profile UUID (RFCOMM 1 (por defecto) si no esta
en uso);
    private static final UUID MY_UUID = UUID
        .fromString("00001101-0000-1000-8000-
00805F9B34FB");

    // ==> escribimos aqui la mac address del servidor, es decir,
    // del dispositivo bluetooth del PC <==
    private static String address = "E8:39:DF:37:3C:2F";

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

        if (D)
            Log.e(TAG, "+++ ON CREATE +++");

        mBluetoothAdapter =
        BluetoothAdapter.getDefaultAdapter();
        if (mBluetoothAdapter == null) {
            Toast.makeText(this, "Bluetooth no disponible.",
Toast.LENGTH_LONG).show();
            finish();
            return;
        }

        if (!mBluetoothAdapter.isEnabled()) {
            Toast.makeText(this, "Por favor, conecte el BT del
dispositivo y vuelva a ejecutar la aplicación.",
Toast.LENGTH_LONG).show();
            finish();
            return;
        }

        if (D)
            Log.e(TAG, "+++ DONE IN ON CREATE, GOT LOCAL BT
ADAPTER +++");
    }

    b1 = (Button) findViewById(R.id.Button01);
    b2 = (Button) findViewById(R.id.Button02);
    b3 = (Button) findViewById(R.id.Button03);
    b4 = (Button) findViewById(R.id.Button04);

    misEventos misEv = new misEventos();
    b1.setOnClickListener(misEv);
    b2.setOnClickListener(misEv);
    b3.setOnClickListener(misEv);
    b4.setOnClickListener(misEv);
}

```

Hemos definido el BluetoothAdapter, que será el bluetooth del dispositivo, el BluetoothSocket, que será el que realice la conexión con el servidor y el OutputStream, que será el que se encargue de la salida de bytes de escritura.

Necesitamos insertar la UUID (Universally Unique Identifier) del bluetooth.

UUID



Es la abreviatura de Identificador Universal Único y es un identificador estándar que se usa para la construcción del software, y que ha sido estandarizado por la OSF (Open Software Foundation) como parte del DCE (Distributed Computing Environment) o Entorno de Computación Distribuida.

La UUID por defecto, dejaremos esa y en la string Address, debemos poner la Mac del dispositivo bluetooth del pc.

Para encontrar la dirección Mac del dispositivo móvil debemos actuar de la siguiente manera. Primero, activamos el bluetooth en nuestro móvil. Una vez activado, nos vamos al menú de “Ajustes” y seleccionamos la última opción “Acerca del teléfono”. Una vez hemos accedido, pulsamos sobre “Estado”, y al final de la página veremos la “Dirección de Bluetooth”:



Figura 3.28. Ventana de estado del teléfono con la dirección MAC Bluetooth.

Una vez arrancada la aplicación, comprobamos que está conectado el bluetooth del móvil.

```
class misEventos implements OnClickListener {

    public void onClick(View v) {
        // TODO Auto-generated method stub
        if (v == b1) {
            enviar("A");
        } else if (v == b2) {
            enviar("B");
        } else if (v == b3) {
            enviar("C");
        } else if (v == b4) {
            enviar("D");
        }
    }
}
```

Con esta clase definimos el evento que desencadena cada pulsación de un botón.

```
public void enviar(String texto) {

    byte[] msgBuffer = texto.getBytes();
    try {
        outStream.write(msgBuffer);
    } catch (IOException e) {
        Log.e(TAG, "ON RESUME: Exception during write.",
e);
    }
}
```

Con el método `enviar()` estamos enviando cada texto vía bluetooth al pc.

```
@Override
public void onStart() {
    super.onStart();
    if (D)
        Log.e(TAG, "++ ON START ++");
}

@Override
public void onResume() {
    super.onResume();

    if (D) {
        Log.e(TAG, "+ ON RESUME +");
        Log.e(TAG, "+ ABOUT TO ATTEMPT CLIENT CONNECT to "
+ address);
    }

    // cuando nos devuelve el device, conoceremos al
    servidor por su
    // direccion mac

    BluetoothDevice device =
mBluetoothAdapter.getRemoteDevice(address);
```

```
        // Necesitamos dos cosas antes de que se puede conectar
correctamente
        // (por problemas de autenticación): una dirección MAC,
que ya tenemos
        // definida,
        // y un canal RFCOMM
        //
        // Debido a que los canales RFCOMM (puertos) son
limitados en número,
        // Android no permite que se utilicen directamente, sino
que tendra que
        // solicitar
        // que se asigne el puerto RFCOMM sobre la base del
servicio ID.
        // En nuestro caso, vamos a utilizar SPP Service ID.
        // Esta ID está en formato UUID (GUID para
Microsofties).
        // Teniendo en cuenta el UUID, Android se encargará de
la asignación por
        // usted.
        // En general, se devolverá RFCOMM 1, pero no siempre,
sino que
        // dependera de lo
        // que otros servicios Bluetooth están en uso en su
dispositivo Android.

    try {
        btSocket =
device.createRfcommSocketToServiceRecord(MY_UUID);
    } catch (IOException e) {
        Log.e(TAG, "ON RESUME: Socket creation failed.",
e);
    }
    Log.d(TAG, "socket created");

    // podria ser que estuviese ejecutando 'búsqueda de
dispositivos'
    // de en la configuracion
    // Bluetooth de tu móvil, por lo que llamamos a
cancelDiscovery () .
    // ya que buscar dispositivos es un proceso muy pesado,
    // y no nos interesa ahora mismo, que intentamos
conectar

    mBluetoothAdapter.cancelDiscovery();

    try {
        btSocket.connect();
        Log.e(TAG,
                "ON RESUME: BT connection established,
data transfer link open.");
    } catch (IOException e) {
        Log.e(TAG, "ON RESUME: connect threw
IOException.");
    }
    try {
        btSocket.close();
    } catch (IOException e2) {
        Log.e(TAG,
```

```

        "ON RESUME: Unable to close
socket during connection failure",
e2);
}
return;
}

// creamos la conexion de datos para comunicarnos con el
servidor
if (D)
    Log.e(TAG, "+ ABOUT TO SAY SOMETHING TO SERVER
+");

try {
    outStream = btSocket.getOutputStream();
} catch (IOException e) {
    Log.e(TAG, "ON RESUME: Output stream creation
failed.", e);
}

String message = "Hola, mensaje del cliente al
servidor.";
byte[] msgBuffer = message.getBytes();
try {
    outStream.write(msgBuffer);
} catch (IOException e) {
    Log.e(TAG, "ON RESUME: Exception during write.",
e);
}
}

@Override
public void onPause() {
super.onPause();

if (D)
    Log.e(TAG, "- ON PAUSE -");

if (outStream != null) {
    try {
        outStream.flush();
    } catch (IOException e) {
        Log.e(TAG, "ON PAUSE: Couldn't flush output
stream.", e);
    }
}

try {
    btSocket.close();
} catch (IOException e2) {
    Log.e(TAG, "ON PAUSE: Unable to close socket.",
e2);
}
}

@Override
public void onStop() {
super.onStop();
if (D)

```

```

        Log.e(TAG, "-- ON STOP --");
    }

@Override
public void onDestroy() {
    super.onDestroy();
    if (D)
        Log.e(TAG, "--- ON DESTROY ---");
}

}

```

En el método `onResume()` le pasamos al bluetooth del móvil la MAC del dispositivo al que queremos conectar. Creamos la conexión con el puerto COM del ordenador y cancelamos cualquier búsqueda que el dispositivo móvil estuviese realizando con el bluetooth del mismo. Si dicha conexión, se ha realizado correctamente, el dispositivo envía un mensaje de bienvenida al pc. En los métodos `onStop()` y `onDestroy()` únicamente guardaremos el Log.

Una vez insertado todo el código vamos a probar nuestra aplicación. Pero antes debemos tener el bluetooth del ordenador encendido, configurado como puerto COM (en nuestro caso COM6) y debemos ejecutar el programa *HyperTerminal*.

Para configurarlo debemos crear una nueva conexión. Seleccionamos el primer ícono:

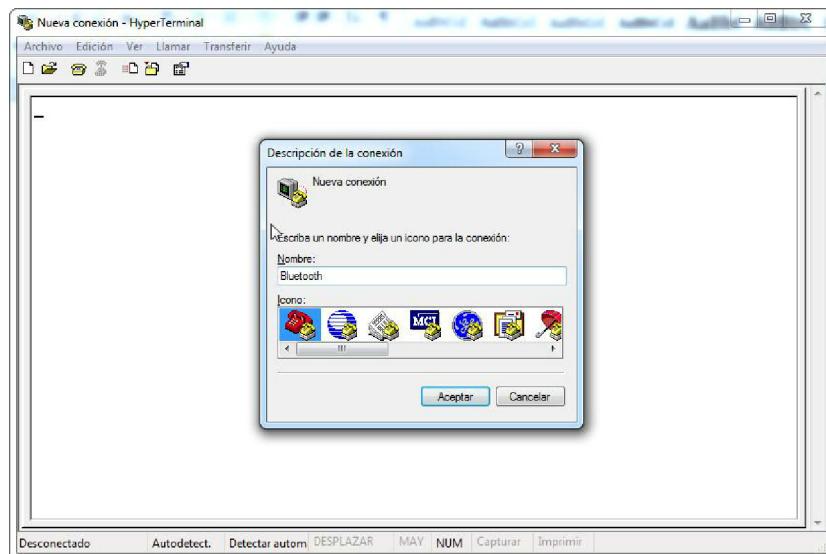


Figura 3.29. Ventana configuración nueva conexión del HyperTerminal.

En la siguiente ventana seleccionamos en “Conectar usando” el puerto COM que anteriormente habíamos asignado al bluetooth en el apartado anterior y lo seleccionamos:

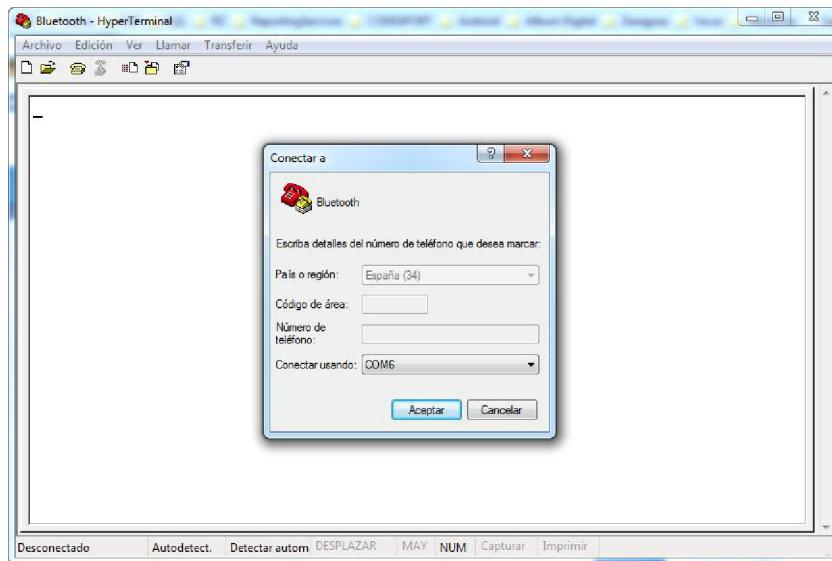


Figura 3.30. Ventana de selección de forma de marcación.

Una vez que pulsamos “Aceptar”, el programa se quedará a la espera de que conectemos. Ahora ya podemos ejecutar la aplicación en el dispositivo. Si la conexión ha sido correcta en la ventana del *HyperTerminal* del pc deberíamos ver el siguiente mensaje:

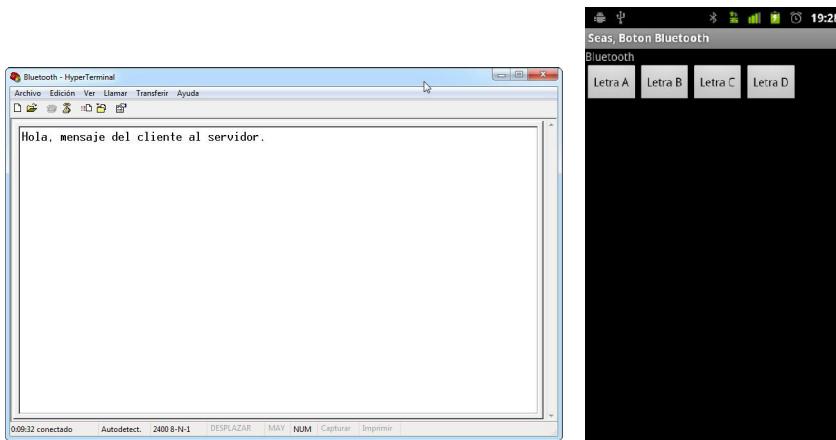


Figura 3.31. Mensaje de conexión correcta entre el móvil y el pc y ventana del dispositivo.

Y si, por ejemplo, pulsamos el botón de “Letra B”, deberíamos ver en la ventana cómo aparece el carácter “B”:

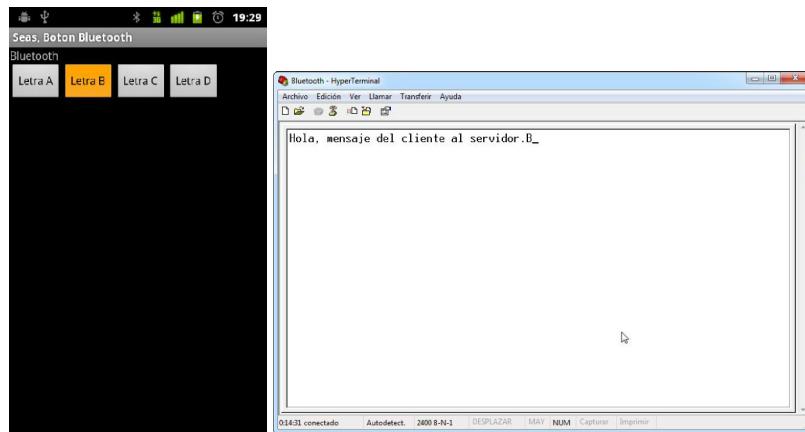


Figura 3.32. Pulsación de botón “Letra B” en el dispositivo móvil.

Disponemos de una versión más nueva de Windows XP, deberemos utilizar el programa “Putty”. Para que éste funcione de la misma manera que el *HyperTerminal* debe configurarlo de la siguiente manera. Arranque el programa y la pantalla principal debe quedar de la siguiente manera:

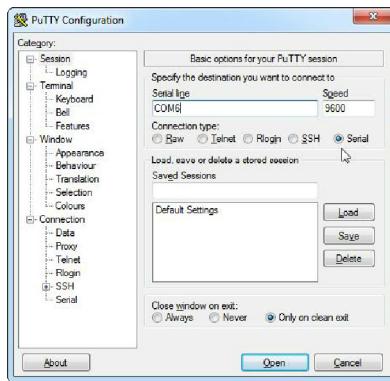


Figura 3.33. Ventana configuración del Putty.exe.

En la parte de “*Connection Type*” deberá seleccionar “*Serial*” y en el cuadro de texto de “*Serial Line*” escribir el puerto COM que anteriormente le ha asignado el sistema. Pulse sobre “*Open*” y el programa se quedará a la espera que ejecute la aplicación en el dispositivo móvil.

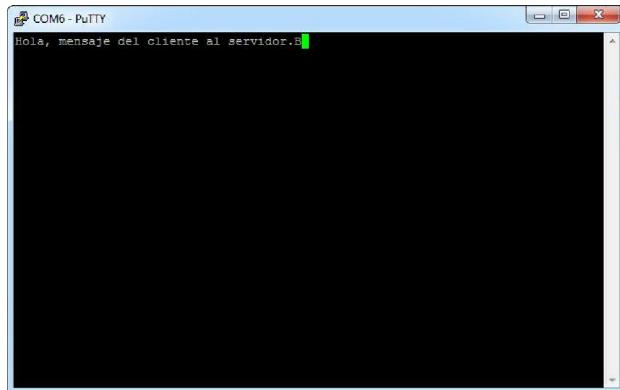


Figura 3.34. Ventana del putty con la conexión del dispositivo móvil realizada.



La aplicación completa “**SeasBluetoothSend**” se puede descargar desde la plataforma de estudio.

3.4. MULTITOUCH

Con cada versión de Android surgen nuevas funcionalidades. El multitouch apareció con Android 2.0. Para definirla de una manera, multitouch es una extensión de la interfaz táctil de usuario, por la que se pueden utilizar dos o más dedos sobre la pantalla. Desde ese momento, es posible “pellizcar” la pantalla para poder aplicar un zoom.

Actualmente existe una clase llamada `ScaleGestureDetector` que ya reconoce el gesto del pellizco. Como ésta ha surgido con Android 2.2 y dicha versión todavía no está más que en una minoría de los teléfonos, hemos decidido no usarla por temas de compatibilidad.

A continuación, vamos a desarrollar un ejemplo, donde vamos a realizar un visualizador de imágenes para poder aplicar zooms y desplazarnos por la imagen. No intentemos ejecutar este ejemplo en un dispositivo con Android 1.5 y 1.6, ya que fallará.

Comenzaremos creando un nuevo proyecto con los siguientes parámetros:

- **Project Name (nombre del proyecto):** SeasTouch
- **Build Target (objetivo de compilación):** Android 2.1
- **Application name (nombre de la aplicación):** Seas, Touch
- **Package name (nombre del paquete):** com.seas.ejemplo.touch
- **Create Activity (crear actividad):** Touch
- **Min SDK Version:** 7

Lo primero, empezaremos editando la actividad principal en `Touch.java`, colocando un escuchador táctil y añadiendo algunas importaciones que más adelante usaremos:

```
package com.seas.ejemplo.touch;

import android.app.Activity;
import android.graphics.Matrix;
import android.graphics.PointF;
import android.os.Bundle;
import android.util.FloatMath;
import android.util.Log;
import android.view.MotionEvent;
import android.view.View;
import android.view.View.OnTouchListener;
import android.widget.ImageView;

public class Touch extends Activity implements OnTouchListener {
    private static final String TAG = "Touch";
    @Override
```

```

public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    ImageView view = (ImageView) findViewById(R.id.imageView);
    view.setOnTouchListener(this);
}
@Override
public boolean onTouch(View v, MotionEvent event) {
    // Manejo de eventos tactiles...
    return false;
}
}

```

Antes de rellenar el método `onTouch()` vamos a modificar la composición del `main.xml`:

```

<?xml version= "1.0" encoding= "utf-8"?>
<FrameLayout
    xmlns:android= "http://schemas.android.com/apk/res/android"
    android:layout_width= "fill_parent"
    android:layout_height= "fill_parent">
    <ImageView android:id= "@+id/imageView"
        android:layout_width= "fill_parent"
        android:layout_height= "fill_parent"
        android:src= "@drawable/imagen"
        android:scaleType= "matrix">
    </ImageView>
</FrameLayout>

```

Con esto hemos definido un `ImageView` que va a ocupar toda la pantalla. Con la propiedad `android:src` definimos la imagen que vamos a utilizar. Podemos utilizar la imagen que deseemos en JPG o PNG, colocándola en el directorio `res/drawable`. Con el atributo `android:scaleType` indicaremos que usamos una matriz para la posición y la escala de la imagen.

El archivo `AndroidManifest.xml` lo rellenaremos con el siguiente código:

```

<?xml version= "1.0" encoding= "utf-8"?>
<manifest xmlns:android=
"http://schemas.android.com/apk/res/android"
    package= "com.seas.ejemplo.touch"
    android:versionCode= "1"
    android:versionName= "1.0">
    <uses-sdk android:minSdkVersion= "7" />

    <application android:icon= "@drawable/icon"
        android:theme= "@android:style/Theme.NoTitleBar.Fullscreen"
        android:label= "@string/app_name">
        <activity android:name= ".Touch"
            android:label= "@string/app_name">
            <intent-filter>
                <action android:name= "android.intent.action.MAIN"
/>
                <category android:name=
"android.intent.category.LAUNCHER" />
            </intent-filter>
        
```

```

        </activity>
    </application>
</manifest>
```

Con `android:theme="@android:style/Theme.NoTitleBar.Fullscreen"` le estamos diciendo a Android que use toda la pantalla sin que muestre ni la barra de título ni la barra de estado en la parte superior. Ahora añadimos una llamada al método `dumpEvent()` en el `onTouch()`:

```

public boolean onTouch(View v, MotionEvent event) {
    // Manejo de eventos tactiles...
    dumpEvent(event);
    return true; // indica que el evento tuvo lugar
}
```

Devolvemos `true` para indicar a Android que el evento tuvo lugar. Definimos el método `dumpEvent()`. Mas adelante explicaremos que hace este método:

```

/** Muestra un evento en la vista LogCat, para depuracion */
private void dumpEvent(MotionEvent event) {
    String names[] = { "DOWN", "UP", "MOVE", "CANCEL",
"OUTSIDE", "POINTER_DOWN", "POINTER_UP", "7?", "8?",
"9?" };
    StringBuilder sb = new StringBuilder();
    int action = event.getAction();
    int keyCode = action & MotionEvent.ACTION_MASK;
    sb.append("event ACTION_").append(names[keyCode]);
    if (keyCode == MotionEvent.ACTION_POINTER_DOWN
        || keyCode ==
MotionEvent.ACTION_POINTER_UP) {
        sb.append("(pid ").append(
            action >>
MotionEvent.ACTION_POINTER_ID_SHIFT);
        sb.append(")");
    }
    sb.append("[");
    for (int i = 0; i < event.getPointerCount(); i++) {
        sb.append("#").append(i);
        sb.append("(pid ").append(event.getPointerId(i));
        sb.append(")=").append((int) event.getX(i));
        sb.append(",").append((int) event.getY(i));
        if (i + 1 < event.getPointerCount())
            sb.append(";");
    }
    sb.append("]");
    Log.d(TAG, sb.toString());
}
```

Si ejecutamos de nuevo y cuando en pantalla movemos el dedo por ella cuando nos muestra la foto, podremos ver el LogCat que hemos generado con el método anterior. La única forma de realizar esta comprobación es mediante un terminal o dispositivo real, ya que el emulador no es compatible con el multitouch.

Vemos los resultados, a continuación, del LogCat:

```
event ACTION_DOWN[#0(pid 0)=136,140]
event ACTION_POINTER_DOWN(pid 1)[#0(pid 0)=136,140;#1(pid 1)=301,647]
event ACTION_MOVE[#0(pid 0)=136,144;#1(pid 1)=301,647]
event ACTION_MOVE[#0(pid 0)=136,144;#1(pid 1)=301,647]
event ACTION_MOVE[#0(pid 0)=135,144;#1(pid 1)=301,647]
event ACTION_MOVE[#0(pid 0)=135,145;#1(pid 1)=301,647]
event ACTION_MOVE[#0(pid 0)=135,146;#1(pid 1)=303,646]
event ACTION_MOVE[#0(pid 0)=135,146;#1(pid 1)=303,645]
event ACTION_MOVE[#0(pid 0)=135,146;#1(pid 1)=303,645]
event ACTION_MOVE[#0(pid 0)=135,146;#1(pid 1)=303,644]
event ACTION_MOVE[#0(pid 0)=135,147;#1(pid 1)=304,643]
event ACTION_MOVE[#0(pid 0)=135,147;#1(pid 1)=304,642]
event ACTION_MOVE[#0(pid 0)=135,147;#1(pid 1)=304,641]
event ACTION_MOVE[#0(pid 0)=134,148;#1(pid 1)=304,640]
event ACTION_MOVE[#0(pid 0)=134,148;#1(pid 1)=304,640]
event ACTION_MOVE[#0(pid 0)=134,148;#1(pid 1)=304,640]
event ACTION_POINTER_UP(pid 0)[#0(pid 0)=134,148;#1(pid 1)=304,640]
event ACTION_UP[#0(pid 1)=304,640]
```

Figura 3.35. Resultado del LogCat cuando movemos el dedo por la pantalla.

El evento `ACTION_DOWN` nos indica que el usuario pulsó con un solo dedo sobre la pantalla. Este dedo fue posicionado en las coordenadas $x=136$, $y=140$.

Con el siguiente evento `ACTION_POINTER_DOWN`, indica que el usuario pulso en la pantalla con otro dedo. `pid 1` indica que se pulso el puntero con ID 1, o sea, el dedo 1, ya que el dedo 0 ya estaba pulsado con el anterior evento.

Después, aparecen eventos `ACTION_MOVE`, donde salen los dos `pid` y si observamos detenidamente veremos que las coordenadas se van acercando.

Posteriormente, aparece un `ACTION_POINTER_UP` en el `pid 0` que nos indica que el dedo 0 se ha levantado de la pantalla, la posición donde se ha levantado y la posición donde el dedo 1 continua (`pid 1`).

Y por último, vemos un `ACTION_UP` que indica que hemos levantado el dedo 1 y la posición en la que estaba cuando se realizó la acción.

Volviendo al código del `dumpEvent()`, podemos ver que `getAction()` nos devuelve la acción que se lleva a cabo. Los 8 bits más bajos son el código de la acción y los 8 siguientes son el ID del puntero o del dedo, por lo que usamos un operador AND a nivel de bits y un desplazamiento a la derecha `>>` para separarlos.

Veremos cuantas posiciones de dedos están incluidas con el método `getPointerCount()`. Y con `getX()` y `getY()`, las coordenadas X e Y. por último con `getPointerId()` averiguaremos de qué dedo hablamos, si del primero que tocó la pantalla o del siguiente dedo. Con una buena interpretación de estos datos podemos pensar la forma de actuación posterior.

En teoría, la API de Android puede con 256 dedos al mismo tiempo. Parece de risa, pero pensando que está pensado para dispositivos que pueden ser Tablet, un juego de tablero en la pantalla, con varios jugadores a la vez, podemos pensar que múltiples dedos van a estar tocándola al mismo tiempo.

Vamos a seguir con nuestra aplicación, y ahora vamos a crear los métodos para realizar la transformación de la imagen. Para ello usaremos una pequeña función de la clase `ImageView` llamada transformación de matrices. Con la matriz representaremos cualquier tipo de rotación o torsión a la imagen. Esta función ya la hemos activado antes con el atributo `android:scaleType= "matrix"` en el archivo `res/layout/main.xml`.

Vamos a declarar dos matrices como campos, una servirá para guardar el valor actual y la otra para guardar el valor original antes de la transformación. Las usaremos en el `onTouch()`. Igualmente necesitamos de una variable `mode` para que nos diga si es un gesto de arrastre o de zoom, y las variables `start`, `oldDist` y `mid`, con las que controlaremos el proceso de zoom:

```
public class Touch extends Activity implements OnTouchListener {  
    private static final String TAG = "Touch";  
  
    // matrices usadas para mover y hacer zoom sobre la imagen  
    Matrix matrix = new Matrix();  
    Matrix savedMatrix = new Matrix();  
  
    // nos podemos encontrar en uno de estos 3 estados  
    static final int NONE = 0;  
    static final int DRAG = 1;  
    static final int ZOOM = 2;  
    int mode = NONE;  
  
    // recuerda algunos aspectos para el proceso de zoom  
    PointF start = new PointF();  
    PointF mid = new PointF();  
    float oldDist = 1f;  
  
    public boolean onTouch(View v, MotionEvent event) {  
        ImageView view = (ImageView) v;  
  
        // volcamos el evento tactil en el registro  
        dumpEvent(event);  
  
        // Manejo de eventos tactiles...  
        switch (event.getAction() & MotionEvent.ACTION_MASK) {  
            case MotionEvent.ACTION_DOWN:  
                savedMatrix.set(matrix);  
                start.set(event.getX(), event.getY());  
                mode = DRAG;  
                break;  
            case MotionEvent.ACTION_POINTER_DOWN:  
                oldDist = getFingerSpacing();  
                savedMatrix.set(matrix);  
                start.set(event.getX(0), event.getY(0));  
                mid.set(event.getX(1), event.getY(1));  
                mode = ZOOM;  
                break;  
            case MotionEvent.ACTION_MOVE:  
                if (mode == DRAG) {  
                    matrix.set(savedMatrix);  
                    matrix.postTranslate(event.getX() - start.x, event.getY() - start.y);  
                } else if (mode == ZOOM) {  
                    float newDist = getFingerSpacing();  
                    if (newDist > oldDist) {  
                        matrix.set(savedMatrix);  
                        matrix.postScale((newDist / oldDist) - 1, (newDist / oldDist) - 1, mid.x, mid.y);  
                    } else {  
                        matrix.set(savedMatrix);  
                        matrix.postScale((oldDist / newDist) - 1, (oldDist / newDist) - 1, mid.x, mid.y);  
                    }  
                }  
                break;  
        }  
        view.setImageMatrix(matrix);  
        return true; // indica que el evento tuvo lugar  
    }  
}
```

Ahora vamos a implementar el gesto de arrastre en `matrix`. Recordando el registro que nos devolvió anteriormente nuestra aplicación, un gesto de arrastre comienza con `ACTION_DOWN`, que es cuando el primer dedo toca la pantalla, y termina con `ACTION_UP` o `ACTION_POINTER_UP`, que es cuando ese dedo se levanta de la pantalla. Definimos esto en `switch`:

```
// Manejo de eventos tactiles...
switch (event.getAction() & MotionEvent.ACTION_MASK) {
    case MotionEvent.ACTION_DOWN:
        savedMatrix.set(matrix);
        start.set(event.getX(), event.getY());
        Log.d(TAG, "mode=DRAG");
        mode = DRAG;
        break;

    case MotionEvent.ACTION_UP:
    case MotionEvent.ACTION_POINTER_UP:
        mode = NONE;
        Log.d(TAG, "mode=NONE");
        break;
    case MotionEvent.ACTION_MOVE:
        if (mode == DRAG) {
            // ...
            matrix.set(savedMatrix);
            matrix.postTranslate(event.getX() - start.x,
event.getY()
                    - start.y);
        }
        break;
}
```

Cuando empezamos el gesto, recordamos la posición inicial del puntero y el valor actual de la matriz de transformación. Cada vez que movamos el dedo, iniciaremos de nuevo esta matriz en su valor original y llamaremos a `postTranslate()` con el que añadiremos un vector de traducción, que será la diferencia entre las posiciones actual e inicial.

Si ejecutamos el programa, podremos mover la imagen utilizando un dedo.



Figura 3.36. Imagen arrastrada por la pantalla.

Ahora implementaremos el gesto del pellizco para realizar el zoom. Para ello primero recordaremos que este gesto comienza con ACTION_POINTER_DOWN, que es cuando el segundo dedo pulsa la pantalla. Añadimos en el switch lo siguiente:

```
case MotionEvent.ACTION_POINTER_DOWN:  
    oldDist = spacing(event);  
    Log.d(TAG, "oldDist=" + oldDist);  
    if (oldDist > 10f) {  
        savedMatrix.set(matrix);  
        midPoint(mid, event);  
        mode = ZOOM;  
        Log.d(TAG, "mode=ZOOM");  
    }  
  
    break;  
  
case MotionEvent.ACTION_MOVE:  
    if (mode == DRAG) {  
        // ...  
    }  
    else if (mode == ZOOM) {  
        float newDist = spacing(event);  
        Log.d(TAG, "newDist=" + newDist);  
        if (newDist > 10f) {  
            matrix.set(savedMatrix);  
            float scale = newDist / oldDist;  
            matrix.postScale(scale, scale, mid.x, mid.y);  
        }  
    }  
  
    break;
```

En el momento en el que se pulsa con el segundo dedo la pantalla, calculamos y guardamos la distancia entre los dedos. Cuando nos llega un evento de movimiento dentro del modo zoom, se vuelve a calcular la distancia entre dedos. Si es muy pequeña, ignoramos el evento. Si no lo es, escalamos la imagen alrededor del punto medio y restablecemos la matriz de transformación.

La escala será la proporción entre la nueva distancia dividida por la distancia antigua. Si el resultado es mayor de 1, significará que los dedos se han separado y agrandaremos la imagen. Si los dedos se han acercado, esta escala será menor de 1, hará más pequeña la imagen. Y si son igual a 1, la imagen no será modificada.

Para realizar todo esto debemos definir los métodos `spacing()` y `midPoint()`:

```
/** determinamos la distancia entre los dedos */
private float spacing(MotionEvent event) {
    float x = event.getX(0) - event.getX(1);
    float y = event.getY(0) - event.getY(1);
    return FloatMath.sqrt(x * x + y * y);
}
```

Lo que hacemos es construir un vector (x, y) que es la diferencia entre los dos puntos, para después aplicar la fórmula de la distancia euclíadiana para calcular el espacio.

Y ahora definimos el método `midPoint()` para calcular el punto medio entre dos puntos:

```
/** Calculamos el punto medio entre los dedos */
private void midPoint(PointF point, MotionEvent event) {
    float x = event.getX(0) + event.getX(1);
    float y = event.getY(0) + event.getY(1);
    point.set(x / 2, y / 2);
}
```

Si ejecutamos la aplicación, deberíamos poder arrastrar la imagen con un solo dedo, y si apoyamos un segundo dedo sobre ella, poder realizar el gesto del pellizco o de separarlos para hacer zoom.



Figura 3.37. Zoom en la imagen mediante gesto “pellizco”.



La aplicación completa “**SeasTouch**” se puede descargar desde la plataforma de estudio.

3.5. RECONOCIMIENTO DE VOZ

El reconocimiento de voz de Android solo funciona con una conexión a internet. Los datos son recogidos y enviados a los centros de datos de Google, donde sus potentes servidores aplican modelos estadísticos para determinar lo que está diciendo. El proceso es súper rápido y se puede realizar desde cualquier lugar, y es increíblemente preciso.

Seguramente se estará pensando, y por qué no implementar ese sistema directamente en el dispositivo sin necesidad de tener que usar las redes móviles o las redes WIFI. Pues muy sencillo. Imaginemos el dispositivo que haría falta para poder realizar ese procesado de datos en tiempo real. Debería tener unas especificaciones que consumiría todos los recursos del dispositivo y no podríamos realizar ninguna otra operación mientras realizase dicha búsqueda.

Para ver el potencial de la búsqueda de Google, vamos a realizar una aplicación que implementa el motor de búsqueda. Para ello vamos a crear un nuevo proyecto de Android e insertamos lo siguiente:

- **Project Name (nombre del proyecto):** SeasVoiceSearch
- **Build Target (objetivo de compilación):** Android 2.1 (Google APIs)
- **Application name (nombre de la aplicación):** Seas, Busqueda Voz
- **Package name (nombre del paquete):** com.seas.ejemplo.VoiceSearch
- **Create Activity (crear actividad):** VoiceSearch
- **Min SDK Version:** 7

Definimos nuestro archivo `AndroidManifest.xml` de la siguiente manera:

```
<?xml version= "1.0" encoding= "utf-8"?>
<manifest xmlns:android=
"http://schemas.android.com/apk/res/android"
    package= "com.seas.ejemplo.VoiceSearch"
    android:versionCode= "1"
    android:versionName= "1.0">
<uses-sdk android:minSdkVersion= "7" />

    <application android:icon= "@drawable/icon" android:label=
"@string/app_name">
        <activity android:name= ".VoiceSearch"
            android:label= "@string/app_name">
            <intent-filter>
                <action android:name= "android.intent.action.MAIN"
/>
                <category android:name=
"android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
```

```
</application>  
</manifest>
```

No vamos a necesitar de ningún permiso especial para realizar el reconocimiento vocal. Ahora debemos modificar el archivo `main.xml` del directorio `res/layout`:

```
<?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">  
            </TextView>  
            <Button android:text= "@string/boton" android:id=  
        "@+id	btn_speak"  
                android:layout_width= "wrap_content"  
        android:layout_height= "wrap_content">  
            </Button>  
            <ListView android:id= "@+id/list" android:layout_width=  
        "wrap_content"  
                android:layout_height= "wrap_content">  
            </ListView>  
</LinearLayout>
```

Hemos creado un botón que va a ser el que inicie la escucha del dispositivo a la espera de que el usuario comience a dictar y un `ListView`, que será donde se presentarán los resultados que nos dé dicha búsqueda.

Debemos insertar en archivo `string.xml` del directorio `res/values` la definición de las etiquetas:

```
<?xml version= "1.0" encoding= "utf-8"?>  
<resources>  
    <string name= "hello"> VoiceSearch! </string>  
    <string name= "app_name"> Seas, Busqueda Voz </string>  
    <string name= "boton"> Comenzar reconocimiento </string>  
</resources>
```

Una vez definido todo, debemos implementar la clase `OnClickListener` para controlar los eventos de la vista:

```
package com.seas.ejemplo.VoiceSearch;  
  
import java.util.ArrayList;  
import java.util.List;  
  
import android.app.Activity;  
import android.content.Intent;  
import android.content.pm.PackageManager;  
import android.content.pm.ResolveInfo;  
import android.os.Bundle;  
import android.speech.RecognizerIntent;  
import android.view.View;
```

```
import android.view.View.OnClickListener;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.ListView;

public class VoiceSearch extends Activity implements OnClickListener {

    private static final int VOICE_RECOGNITION_REQUEST_CODE =
1234;

    private ListView mList;

    /**
     * Called with the activity is first created.
     */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setContentView(R.layout.main);

        // Obtenemos los elementos en pantalla para la
        interacción posterior
        Button speakButton = (Button)
findViewById(R.id.btn_speak);

        mList = (ListView) findViewById(R.id.list);

        // Comprueba si la actividad de reconocimiento está
        presente
        PackageManager pm = getPackageManager();
        List<ResolveInfo> activities =
pm.queryIntentActivities(new Intent(
                    RecognizerIntent.ACTION_RECOGNIZE_SPEECH),
0);
        if (activities.size() != 0) {
            speakButton.setOnClickListener(this);
        } else {
            speakButton.setEnabled(false);
            speakButton.setText("Reconocimiento no presente");
        }
    }

    /**
     * Manejamos el click de inicio del botón.
     */
    public void onClick(View v) {
        if (v.getId() == R.id.btn_speak) {
            startVoiceRecognitionActivity();
        }
    }

    /**
     * Intent de inicio de la actividad de reconocimiento de voz.
     */
    private void startVoiceRecognitionActivity() {
        Intent intent = new
Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEECH);
```

```
        intent.putExtra(RecognizerIntent.EXTRA_LANGUAGE_MODEL,
                        RecognizerIntent.LANGUAGE_MODEL_FREE_FORM);
        intent.putExtra(RecognizerIntent.EXTRA_PROMPT,
                        "Reconocimiento por voz");
        startActivityForResult(intent,
VOICE_RECOGNITION_REQUEST_CODE);
    }

    /**
     * manejo de los resultados de la actividad de reconocimiento.
     */
    @Override
    protected void onActivityResult(int requestCode, int
resultCode, Intent data) {
        if (requestCode == VOICE_RECOGNITION_REQUEST_CODE
            && resultCode == RESULT_OK) {
            // aquí recogemos las frases interpretadas
            ArrayList<String> matches = data

            .getStringArrayListExtra(RecognizerIntent.EXTRA_RESULTS);
            mList.setAdapter(new ArrayAdapter<String>(this,
                android.R.layout.simple_list_item_1,
matches));
        }
        super.onActivityResult(requestCode, resultCode, data);
    }

    @Override
    protected void onStop() {
        super.onStop();
    }

    public void onStart() {
        super.onStart();
    }

    @Override
    public void onResume() {
        super.onResume();
    }

    @Override
    public void onPause() {
        super.onPause();
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
    }
}
```

El método `onCreate()` realiza la inicialización cuando la actividad se crea por primera vez. Este método también consulta la `packagemanager` para comprobar si existen paquetes instalados que puedan manejar los intents de `ACTION_RECOGNIZE_SPEECH`. Esto lo hacemos para comprobar que tenemos un paquete instalado que pueda hacer la traducción, y si no lo tuviésemos, deshabilitaríamos el botón.

El método `speakButtonClicked()` se invoca cuando se pulsa el botón. El método `startVoiceRecognitionActivity()` invoca una actividad que puede manejar el reconocimiento de voz y ajusta el modo de lenguaje de forma libre.

El método `onActivityResult()` es el que devuelve la llamada desde la orden anterior, que primero comprueba que el código de solicitud coincide con la que se aprobó, y asegura que el resultado es correcto y no un error.

A continuación, los resultados son recuperados con el `intent` y los escribimos en el `ListView` para mostrarlos en la pantalla.

Si ejecutamos la aplicación, la ventana principal nos muestra un botón para comenzar con el reconocimiento de voz:

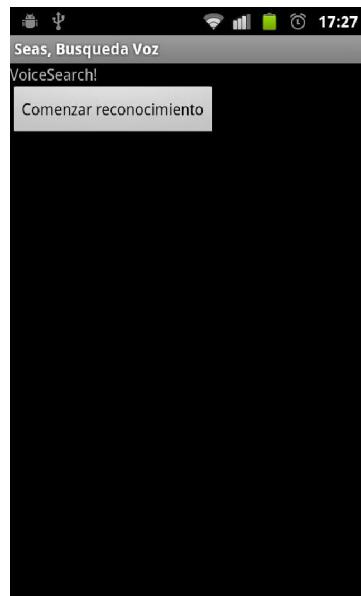


Figura 3.38. Pantalla de inicio de la aplicación.

Si pulsamos sobre el botón de “Comenzar Reconocimiento”, y sobre todo si tenemos activada la conexión a internet en el dispositivo, nos aparecerá un cuadro como el siguiente:



Figura 3.39. Cuadro de diálogo que aparece cuando pulsamos sobre el botón.

En ese momento comenzamos a hablar y en cuanto el dispositivo detecte un silencio largo, se pondrá a buscar:



Figura 3.40. Ventana de búsqueda de la aplicación.

Y en el momento que se obtengan los resultados, los inserta en el ListView.

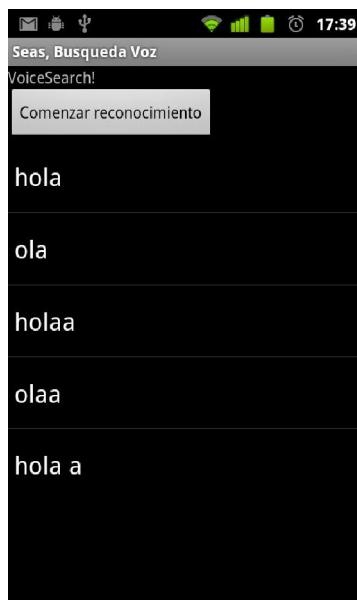


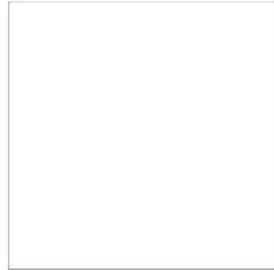
Figura 3.41. Ventana que muestra los resultados obtenidos.



La aplicación completa “**SeasVoiceSearch**” se puede descargar desde la plataforma de estudio.

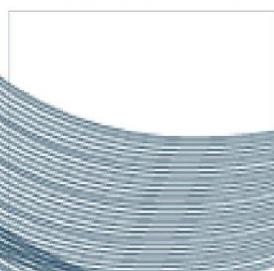
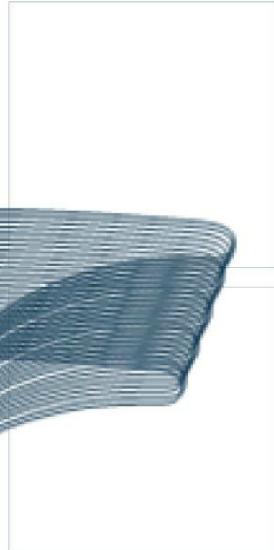
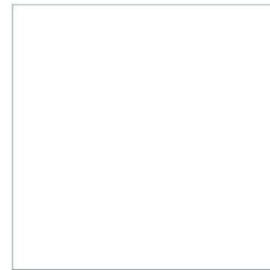
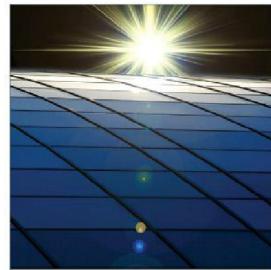
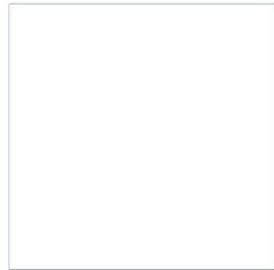
RESUMEN

- Hemos visto cómo funciona la geolocalización. La posibilidad que nos ofrecen los dispositivos Android de recoger de manera automática nuestra posición, ya sea a través de la localización por GPS, o por la que dan los proveedores de los servicios de internet.
 - Hemos aprendido a recoger la información que nos da el dispositivo.
 - Hemos configurado el Api Key para poder usar los mapas de Google Maps en nuestro dispositivo.
 - Hemos trabajado con los mapas de Google añadiendo funcionalidades en ellos.
- Hemos visto los sensores de los dispositivos que Android nos permite acceder para poder usarlos en nuestras aplicaciones, dándole un nuevo concepto de manejabilidad.
- Hemos operado con otro hardware de los dispositivos de conectividad que nos permite Android: el Bluetooth. Hemos podido realizar un envío de datos desde el dispositivo al pc con una aplicación cliente-servidor.
- Otra de las mejoras que Android ha desarrollado en sus últimas versiones, el multitouch. Hemos aprendido su funcionamiento e implementación de los gestos de “pellizco” y zoom.
- Debemos saber ya cómo funciona el reconocimiento vocal de Google y su implementación en una sencilla aplicación.



04

ANDROID



WEB MÓVIL
EN ACCIÓN

 SEAS
ESTUDIOS SUPERIORES ABIERTOS

ÍNDICE

◆ ÍNDICE	1
◆ OBJETIVOS	3
◆ INTRODUCCIÓN.....	4
4.1. Web Services	5
4.1.1. Arquitectura de los Servicios Web	6
4.1.2. Estándar de Servicios Web	7
4.1.3. REST	8
4.1.4. SOAP	18
4.1.5. Crear un Web Service en Java	25
4.1.6. Crear un Web Service en .net y c#	35
4.2. Cliente Android-Servidor PHP-MySql.....	37
4.2.1. Interfaz de usuario.....	37
4.2.2. Servidor PHP.....	60
4.2.3. Base de Datos	63
4.2.4. Código de servidor	69
◆ RESUMEN.....	73

OBJETIVOS

- En esta unidad veremos qué son los Servicios Web, su arquitectura, para qué sirven y qué nos ofrecen.
- Aprenderemos a desarrollar un Servicio Web con diferentes lenguajes. Desarrollaremos en Java con servidor Apache Tomcat y en Visual Studio con su propio servidor de aplicaciones.
- Veremos cómo funciona el protocolo de comunicaciones REST.
- Aprenderemos a usar SOAP con un ejemplo práctico, realizando consultas a un Web Service gratuito de internet. Realizaremos las conexiones necesarias y veremos las respuestas del servidor.
- En la segunda parte de esta unidad realizaremos una práctica que va a unir muchas de las partes que se han ido viendo por separado durante el desarrollo de este temario. Uniremos los servicios de posicionamiento que nos ofrece Google, con el uso de las bases de datos para realizar una aplicación que nos ofrezca información detallada de puntos determinados que habremos predefinido con anterioridad, y se mostrará en un mapa dependiendo de la posición en la que nos encontremos y de los parámetros que le pasemos.

INTRODUCCIÓN

En los temas anteriores hemos introducido conceptos y soluciones gráficas, como dibujar imágenes en 2D y 3D, también hemos desarrollado bases de datos y en esta unidad las vamos a intentar unir todas en una única aplicación.

Nos adentraremos en los servicios que nos ofrece en la actualidad internet y vamos a aprender a obtener todos esos recursos, tratar los datos obtenidos y manejarlos en nuestra interfaz de usuario.

Configuraremos un ejemplo de Servicio Web en lenguaje java y .net. Veremos las formas de conectarnos a ellos mediante SOAP y REST.

Realizaremos la configuración de un servidor real en *php*, agregaremos una base de datos y posicionaremos en los mapas las localizaciones guardadas.

4.1. WEB SERVICES

El término de Servicio Web, dependiendo del origen de los datos y del destino de ellos, y con esto nos referimos al público al que irá destinado, tendrá un significado diferente.



Servicios Web

Son un medio para mostrar una API a través de un punto de red. Con ellos podemos invocar un método remoto para obtener el resultado deseado.

Los Servicios Web son muy usados, ya que son una plataforma neutral, son accesibles a través de vías estándar e interactivas, es decir, a través de la red y de aplicaciones móviles. Usan herramientas sencillas y disponibles en todas partes, son relativamente baratos de implementar y esto simplifica su integración en la empresa, es decir, podemos programar en Java e invocar un Web Service en .Net. Esto lo veremos más adelante.

Las propiedades más importantes de los Servicios Web son:

- **Interoperabilidad:** usa redes heterogéneas para realizar la conexión usando estándares web.
- **Económica:** recicla componentes, no es necesaria la instalación, ni una gran integración con el software.
- **Automática:** no es necesaria la intervención humana incluso para las operaciones de alta complejidad.
- **Accesible:** los activos heredados y las aplicaciones internas están expuestos y son accesibles en la Web.
- **Disponibilidad:** los servicios están disponibles en cualquier dispositivo, en cualquier lugar y en cualquier momento.
- **Escalable:** no hay límites en el alcance que tienen las aplicaciones y la cantidad de aplicaciones distintas existentes.

4.1.1. ARQUITECTURA DE LOS SERVICIOS WEB

Los Servicios Web tienen la siguiente arquitectura:

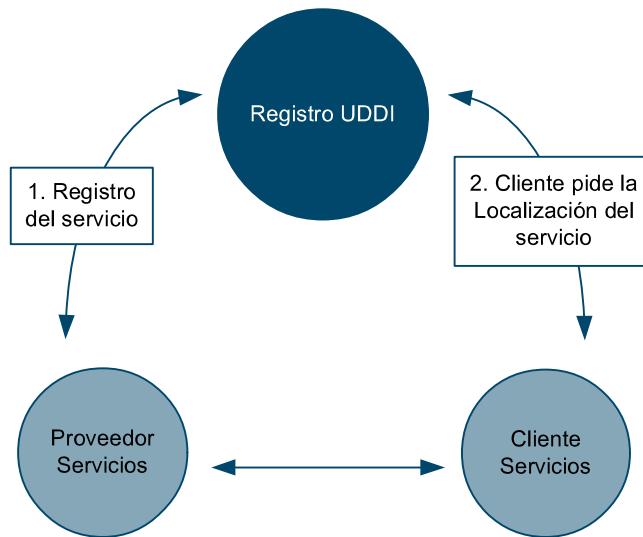


Figura 4.1. Arquitectura de los Servicios Web.

El proveedor de servicios

- Crea el Servicio Web por lo general como interfaz de servicios basados en SOAP.



SOAP

Es un protocolo ligero destinado a intercambiar información estructurada.

- Despliega el servicio y lo pone a disposición para la invocación a través de la red.
- Describe el Servicio Web como un servicio basado en WSDL.



WSDL

Es un lenguaje XML para describir Servicios Web como un conjunto de variables de comunicación entre puntos.

- Registra el Servicio WSDL con un corredor de servicio, que normalmente es un registro UDDI (descrito a continuación).

El registro UDDI

- Almacena la descripción del servicio como plantillas obligatorias y direcciones URL de WSDL situado en el entorno del proveedor de servicios.
- Lista de diferentes tipos de servicios, descripciones y ubicaciones de los servicios que ayudan a los solicitantes de dichos servicios para encontrar y suscribirse a los servicios requeridos.

El cliente de servicios

- Localiza los servicios requeridos para consultar el registro UDDI.
- Se obtiene la información de enlace y las direcciones URL para identificar el proveedor de servicios.
- Inicia el proveedor de servicios.
- Recupera la descripción del Servicio WSDL de los servicios registrados.
- Se comunica con el proveedor de servicios.
- Realiza el intercambio de datos o de mensajes invocando los servicios disponibles en el contenedor de servicios.

4.1.2. ESTÁNDAR DE SERVICIOS WEB

Los estándares de Servicios Web son los siguientes:

- XML (Extensible Markup Language).
- SOAP (Simple Object Access Protocol).
- WSDL (Web Services Description Language).
- UDDI (Universal Description and Discovery Integration).

Vamos a definir cada uno de ellos.

XML



Es un lenguaje “Marcado” basado en texto. Este marcado es el que tiene la información extra de la descripción y el formato de los datos. Debemos saber que es un lenguaje estándar para el intercambio y representación de los datos en Internet.



SOAP

Son las siglas de Protocolo Simple de Acceso a Objetos. Es un protocolo ligero destinado a intercambiar información estructurada. Utiliza la tecnología XML para definir un marco de mensajería extensible. Este marco ha sido diseñado para ser independiente de cualquier modelo de programación en particular y de la semántica de otra aplicación específica.



WSDL

Son las siglas de Lenguaje de Descripción de Servicios Web. Es un lenguaje XML para describir Servicios Web como un conjunto de variables de comunicación entre puntos (puertos). Estos se componen de dos partes: la definición abstracta de las operaciones y mensajes, y la unión para el formato del protocolo de red y de los mensajes.



UDDI

Son las siglas de Descripción, Descubrimiento e Integración Universal. Es el registro programado y descubrimiento de las entidades de negocio y sus Servicios Web. Tiene su base en HTML, SOAP y XML. El UDDI registra los siguientes datos:

- Los registros de negocio
 - La definición de los servicios tipo
-

Con todo lo visto hasta ahora, vamos a detenernos en los métodos REST y SOAP. Cada uno nos da unas directrices generales para acceder a los datos y exponer operaciones, y cada uno de éstos tiene unas ventajas y desventajas, algo que se magnifica en una plataforma móvil como Android. Va a ser imposible explicar todos los detalles de cada uno de ellos, pero veremos las principales diferencias y examinaremos su uso.

4.1.3. REST

REST es la abreviatura de *Representational State Transfer*. Es un estilo de arquitectura de software para sistemas hipermedia distribuidos, como la *World Wide Web*.

El término fue introducido por primera vez en la tesis doctoral de Roy Fielding en el año 2000, uno de los autores principales de la Especificación del Protocolo de transferencia de hipertexto (HTTP).

REST es el protocolo más detallado y se basa en el concepto de recursos para definir datos que después maneja con distintos métodos de http, dándole un enfoque similar a URL (parecido al sistema intent de Android).

REST se refiere estrictamente a un conjunto de principios de arquitectura de red que describen cómo se definen los recursos y a quién se ha dirigido. Los principales son:

- **GET.** Obtiene un recurso.
- **PUT.** Crea un recurso.
- **POST.** Actualiza un recurso.
- **DELETE.** Borra un recurso.
- **HEAD.** Obtiene los metadatos que definen el recurso.

El término se utiliza a menudo en un sentido más amplio para describir cualquier interfaz sencilla que transmite los datos específicos del dominio a través de HTTP sin una capa adicional de mensajería, tales como SOAP o una sesión de seguimiento a través de las cookies de HTTP.

Vamos a realizar un ejemplo práctico en el que consumimos un Servicio Web de Yahoo que nos devolverá los datos a través de una conexión REST. Yahoo pone a disposición de los desarrolladores una página para poder consumir los Servicios Web que ofrece. Podemos consultarla en la siguiente dirección:
<http://developer.yahoo.com/>

Debemos acceder a la sección de “Apis & Tools” y desde allí a la de “APIs and Web Services” y nos desplazamos hasta la opción de “Network Time”

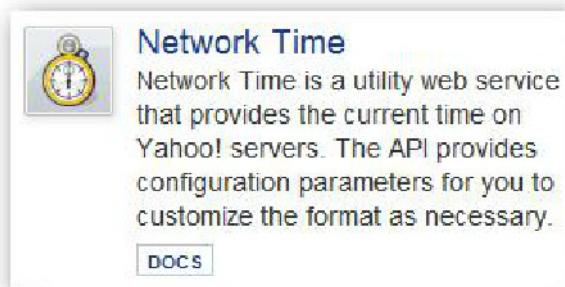


Figura 4.2. Web Service de Network Time.

Para comenzar con el desarrollo de la aplicación creamos un nuevo “Android Proyecto” en Eclipse e insertamos los siguientes datos:

- **Project Name (Nombre del proyecto):** SeasRestService
- **Build Target (Objetivo de compilación):** Android 1.6
- **Application name (Nombre de la aplicación):** Seas, RestService
- **Package name (Nombre del paquete):** com.seas.ejemplo.RestService
- **Create Activity (Crear actividad):** Start
- **Min SDK Version:** 4

Lo primero, necesitamos dar permisos de acceso a Internet a la aplicación. Insertamos el siguiente código en el `AndroidManifest.xml`:

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

Comenzamos configurando el archivo `main.xml` de la carpeta `res/layout` y definimos un botón que al pulsarlo lanzara la petición. Al devolverla nos escribirá la hora en un `TextView`:

```
<?xml version= "1.0" encoding= "utf-8"?>
<RelativeLayout android:id= "@+id/RelativeLayout"
    android:layout_width= "fill_parent"
    android:layout_height= "fill_parent"
    xmlns:android= "http://schemas.android.com/apk/res/android">
    <Button android:id= "@+id/btnCallWebService"
        android:layout_width= "wrap_content"
        android:layout_height= "wrap_content"
        android:text= "Llamada Servicio REST"
        android:height= "50dp"
        android:layout_centerVertical= "true"
        android:layout_centerHorizontal= "true">
    </Button>
    <TextView android:id= "@+id/txtTime"
        android:layout_width= "wrap_content"
        android:layout_height= "wrap_content"
        android:text= "Resultado"
        android:layout_above= "@+id/btnCallWebService"
        android:layout_alignLeft= "@+id/btnCallWebService">
    </TextView>
</RelativeLayout>
```

La Activity principal será `Start.java`. Vamos a definir el manejador del evento del click del botón en el método `onCreate()`:

```
package com.seas.ejemplo.RestService;

import org.json.JSONException;
import org.json.JSONObject;

import android.app.Activity;
import android.app.ProgressDialog;
```

```

import android.content.Context;
import android.os.AsyncTask;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.TextView;

public class Start extends Activity {

    private TextView txtTime;
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        txtTime = (TextView) findViewById(R.id.txtTime);

        // añadimos el manejador del evento click del botón

        final Button btnCallWebService = (Button)
        findViewById(R.id.btnCallWebService);

        btnCallWebService.setOnClickListener(new
        OnClickListener() {

            public void onClick(View v) {

                CallWebServiceTask task = new
                CallWebServiceTask();
                task.applicationContext = Start.this;
                task.execute();
            }
        });
    }
}

```

Creamos la clase CallWebServiceTask que extiende de AsyncTask y sobrescribimos sus métodos.

```

public class CallWebServiceTask extends AsyncTask <Void, Void,
String>{

    private ProgressDialog dialog;
    protected Context applicationContext;

    @Override
    protected void onPreExecute() {

        this.dialog =
        ProgressDialog.show(applicationContext, "Calling",
                            "Time Service...", true);
    }
}

```

```

@Override
protected String doInBackground(Void... params) {
    return Start.getTimeStampFromYahooService();
}

@Override
protected void onPostExecute(String result) {
    this.dialog.cancel();

    String timestamp =
Start.parseJSONResponse(result);

    timestamp =
Start.UnixTimeStampToDateTIme(timestamp);

    //Start.this.getText(R.id.txtTime);

    Start.this.txtTime.setText(timestamp);
}
}
}

```

En esta clase definimos tres métodos. En el método `OnPreExecute()` se realizan las acciones antes de comenzar con la parte principal del trabajo, que se realizará dentro del método `doInBackground()`. Aquí mostraremos el diálogo de progreso.

En el método `doInBackground()` llevaremos a cabo la parte principal del trabajo, y realizaremos la llamada a Servicio Web de Yahoo.

En el método `OnPostExecute()` realizamos las acciones después de hacer la principal. Cerraremos el diálogo de progreso, analizaremos los datos recibidos y mostraremos la fecha y hora en el cuadro de texto.

Insertamos el siguiente código entre del método `onCreate()` y la clase `CallWebServiceTask` que son los métodos de apoyo de la clase de la Activity `Start`:

```

// metodo que convierte la cadena de tiempo UnixTime a formato
public static String UnixTimeStampToDateTIme(String
unixTimeStamp) {

    long tiemstamp = Long.parseLong(unixTimeStamp);
    String dateStr = new
java.text.SimpleDateFormat("dd/MM/yyyy HH:mm:ss").format(new
java.util.Date (tiemstamp*1000));

    return dateStr;
}

public static String parseJSONResponse(String jsonResponse) {
    String timestamp = "";

```

```

        JSONObject json;
    try {
        json = new JSONObject(jsonResponse);
        JSONObject result = json.getJSONObject("Result");
        timestamp = result.getString("Timestamp");

    } catch (JSONException e) {

        e.printStackTrace();
    }

    return timestamp;
}

public static String getTimeStampFromYahooService() {

    String responseString = null;

    String baseurlString =
"http://developer.yahooapis.com/TimeService/V1/getTime";

    RestClient client = new RestClient(baseurlString);
    client.AddParam("appid", "YahooDemo");
    client.AddParam("output", "json");

    try {
        client.Execute(RequestMethod.GET);
    } catch (Exception e) {
        e.printStackTrace();
    }

    responseString = client.getResponse();

    return responseString;
}

```

Con el método `UnixTimeStampToDateTIme()` realizamos la conversión del formato de tiempo que nos envía Yahoo a formato estándar. El método `parseJSONResponse()` parsea la cadena JSON recibida desde Yahoo y el método `getTimeStampFromYahooService()` es el que realiza la conexión y recoge los datos del Web Service.

Necesitamos crear la clase `RestClient.java` que define el cliente REST. Implementaremos las librerías de `org.apache.http` incluidas en Android para realizar las conexiones al Servicio Web. Su código será el siguiente:

```

package com.seas.ejemplo.RestService;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.net.URLEncoder;
import java.util.ArrayList;

import org.apache.http.HttpEntity;
import org.apache.http.HttpResponse;

```

```
import org.apache.http.NameValuePair;
import org.apache.http.client.ClientProtocolException;
import org.apache.http.client.HttpClient;
import org.apache.http.client.entity.UrlEncodedFormEntity;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.client.methods.HttpUriRequest;
import org.apache.http.impl.client.DefaultHttpClient;
import org.apache.http.message.BasicNameValuePair;
import org.apache.http.protocol.HTTP;

public class RestClient {
    // definimos las variables para los parametros y cabeceras
    private ArrayList<NameValuePair> params;
    private ArrayList<NameValuePair> headers;

    // definimos variable URL que sera la direccion del servicio
    private String url;

    // definimos variables para el código de respuesta y el
    // mensaje
    private int responseCode;
    private String message;

    // definimos la variable respuesta
    private String response;

    public int getResponseCode() {
        return responseCode;
    }

    public String getErrorMessage() {
        return message;
    }

    public String getResponse() {
        return response;
    }

    public void AddHeader(String name, String value) {
        headers.add(new BasicNameValuePair(name, value));
    }

    public void AddParam(String name, String value) {
        params.add(new BasicNameValuePair(name, value));
    }

    public RestClient(String url) {
        this.url = url;
        params = new ArrayList<NameValuePair>();
        headers = new ArrayList<NameValuePair>();
    }
}
```

Una vez declaradas todas las variables, declaramos los métodos, empezando por el de `execute(requestMetod)`, definiendo el tipo de respuesta que nos dará el servidor, diferenciando entre una petición *GET* o *POST*:

```
public void Execute(RequestMethod method) throws Exception {
```

```

// definimos los métodos de respuesta GET o POST
switch (method) {
    case GET: {
        // add parameters
        String combinedParams = "";
        if (!params.isEmpty()) {
            combinedParams += "?";
            for (NameValuePair p : params) {
                String paramString = p.getName() + "="
                    +
                    URLEncoder.encode(p.getValue(), "UTF-8");
                if (combinedParams.length() > 1) {
                    combinedParams += "&" +
                    paramString;
                } else {
                    combinedParams += paramString;
                }
            }
        }
        HttpGet request = new HttpGet(url +
        combinedParams);

        // add headers
        for (NameValuePair h : headers) {
            request.addHeader(h.getName(),
            h.getValue());
        }

        executeRequest(request, url);
        break;
    }
    case POST: {
        HttpPost request = new HttpPost(url);

        // add headers
        for (NameValuePair h : headers) {
            request.addHeader(h.getName(),
            h.getValue());
        }

        if (!params.isEmpty()) {
            request.setEntity(new
            UrlEncodedFormEntity(params, HTTP.UTF_8));
        }

        executeRequest(request, url);
        break;
    }
}
}

```

Después, debemos declarar el método `executeRequest()` que nos devolverá los datos del servidor:

```

private void executeRequest(HttpUriRequest request, String
url) {
    // recogemos la Respuesta del servidor
    HttpClient client = new DefaultHttpClient();

```

```
    HttpResponse httpResponse;

    try {
        httpResponse = client.execute(request);
        responseCode =
httpResponse.getStatusLine().getStatusCode();
        message =
httpResponse.getStatusLine().getReasonPhrase();

        HttpEntity entity = httpResponse.getEntity();

        if (entity != null) {

            InputStream instream = entity.getContent();
            response = convertStreamToString(instream);

            // Closing the input stream will trigger
connection release
            instream.close();
        }

    } catch (ClientProtocolException e) {
        client.getConnectionManager().shutdown();
        e.printStackTrace();
    } catch (IOException e) {
        client.getConnectionManager().shutdown();
        e.printStackTrace();
    }
}
```

Y por último, debemos declarar el método `convertStreamToString()` que parsea el `InputStream` devuelto en cadena de texto:

```
private static String convertStreamToString(InputStream is) {

    BufferedReader reader = new BufferedReader(new
InputStreamReader(is));
    StringBuilder sb = new StringBuilder();

    String line = null;
    try {
        while ((line = reader.readLine()) != null) {
            sb.append(line + "\n");
        }
    } catch (IOException e) {
        e.printStackTrace();
    } finally {
        try {
            is.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    return sb.toString();
}
```

También debemos crear la clase RequestMethod.java donde definiremos los tipos enum de RequestMethod.

```
package com.seas.ejemplo.RestService;

public enum RequestMethod
{
    GET,
    POST
}
```

Si ejecutamos la aplicación, nos muestra la Activity principal Start :



Figura 4.3. Ventana principal de la aplicación RestService.

Cuando pulsamos sobre el botón, aparece el diálogo de proceso definido en el método onPreExecute () :



Figura 4.4. Ventana con el dialogo de proceso.

Cuando acaba con el proceso de la llamada, la parsea y la presenta en pantalla:



Figura 4.5. Ventana con el resultado devuelto del Servicio Web.



La aplicación completa “**Seas RestService**” se puede descargar desde la plataforma de estudio.

4.1.4. SOAP

SOAP impone reglas estrictas sobre los tipos de datos, mecanismos de transporte y seguridad.

El JDK de Java proporciona una herramienta llamada *wsimport* que genera las clases a partir de WSDL, y el usuario trabaja con ellas como con otras clases de Java. **Pero éstas no funcionan en Android...** Android SDK no proporciona clases para trabajar con los servicios Web. Por lo tanto, hay que realizarlo manualmente mediante SOAP.

Hay una buena biblioteca llamada kSOAP2 que es un servicio de cliente Web SOAP, que incorpora una biblioteca de cliente para entornos con limitaciones de Java, tales como *applets* o aplicaciones J2ME. Hay un puerto de esta biblioteca a la plataforma Android que ha sido publicado en el código de Google.

kSOAP2 es una librería ligera y eficiente, características imprescindibles para su utilización en entornos de aplicaciones móviles. Desde la página del proyecto <http://code.google.com/p/ksoap2-android/> podemos bajarnos la última versión de la librería para incluir en nuestro proyecto de Eclipse. Más adelante veremos cómo añadirla a nuestro proyecto.



Vamos a realizar una aplicación para Android, que usando las librerías anteriores realizará la petición de datos a un Web Service gratuito alojado en internet.

<http://fx.cloanto.com/webservices/CloantoCurrencyServer.asmx>

Puedes consultar el enlace en la plataforma de estudio.

Vemos una imagen de los servicios disponibles a los que podemos acceder en el Web Service.

The screenshot shows the 'CurrencyServer' service page. At the top, it says 'Currency Server - An exchange rate information and currency conversion Web service.' Below that, it states 'The following operations are supported. For a formal definition, please review the [Service Description](#).'. A bulleted list follows, detailing various operations such as AdminLoad, AdminMessage, AdminSave, AdminUpdateNow, ConvertToNum, ConvertToStr, CountryToCurrency, Currencies, CurrencyCustom, CurrencyExists, CurrencyHits, and CurrencyLastChangeTimeLocal.

Figura 4.6. Imagen del web service y los servicios que proporciona.

Podemos modificar los parámetros para realizar la conexión a cualquier Web Service, pero siempre teniendo en cuenta, que acepte soap1.1 o 1.2, es decir, que en su *xmlIn* contenga la siguiente línea:

```
Soap 1.1: xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/";
Soap 1.2: xmlns:soap12="http://www.w3.org/2003/05/soap-envelope"
```

SOAP 1.1

The following is a sample SOAP 1.1 request and response. The placeholders shown need to be replaced with actual values.

```

POST /webservices/CleanoCurrencyServer.asmx HTTP/1.1
Host: fx.ciano.com
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "http://webservices.ciano.com/currencyserver/AdminLoad"

<?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>
<AdminLoad xmlns="http://webservices.ciano.com/currencyserver/">
<licenseKey>string</licenseKey>
<fileString>string</fileString>
</AdminLoad>
</soap:Body>
</soap:Envelope>

```

Figura 4.7. Vista de las propiedades del Web Service.

El *Web Service* que hemos realizado en Java con Eclipse, no nos va a servir, puesto que no lo implementa. En cambio, el realizado con *.net* y *c#* sí que nos servirá para poder realizar pruebas en local. Al final del punto explicamos cómo realizar la conexión y qué parámetros debemos cambiar en ella.

Vamos a iniciar el desarrollo la aplicación. Para ello debemos crear un nuevo *Android Project*, e insertamos los siguientes parámetros:

- **Project Name (Nombre del proyecto):** SeasSoapWebService
- **Build Target (Objetivo de compilación):** Android 2.1 Update1
- **Application name (Nombre de la aplicación):** Seas, Soap
- **Package name (Nombre del paquete):** com.seas.ejemplo.seasSoap
- **Create Activity (Crear actividad):** soapWebService
- **Min SDK Version:** 7

Lo primero que vamos a realizar es insertar la librería KSOAP2 que nos hemos descargado anteriormente y la insertamos en nuestro proyecto. Para ello vamos al *Menú → Project → Properties*:

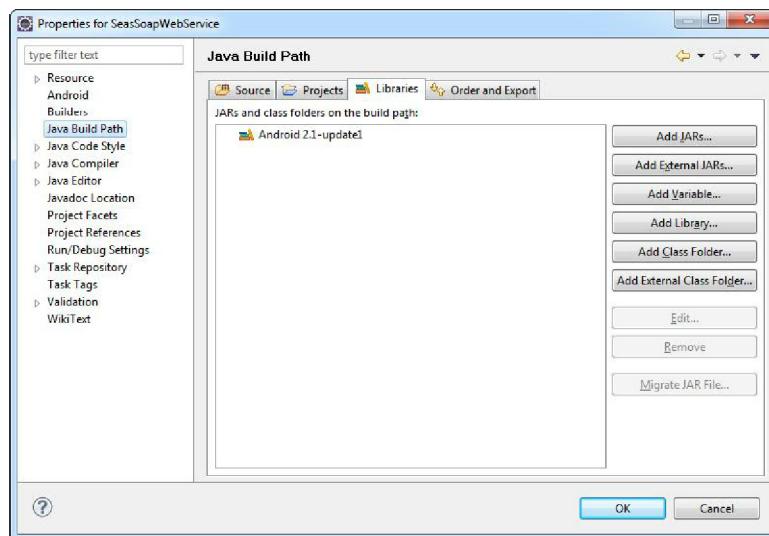


Figura 4.8. Ventana de Propiedades del Proyecto.

Y en el apartado de “Java Build Path” pulsamos sobre “Add External JARs...” Seleccionamos la librería que nos hemos descargado anteriormente y la añadimos.

Una vez importadas, nos metemos con nuestra clase. Lo primero vamos a dar permisos en el `AndroidManifest.xml` para tener acceso a internet. Añadimos la siguiente línea:

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

En la pantalla principal, únicamente vamos a insertar un botón que realizará la consulta cuando pulsemos sobre él. Definimos la pantalla en el archivo `main.xml` del directorio `res/layout`:

```
<?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">
    <Button
        android:layout_width= "wrap_content"
        android:layout_height= "wrap_content"
        android:id= "@+id/button1"
        android:text= "Button"> </Button>
</LinearLayout>
```

Pasamos a la clase que realizará la lógica. Lo primero vamos a definir cuatro variables que contienen los parámetros del *Web Service*:

```
package com.seas.ejemplo.seassSoap;

import org.ksoap2.SoapEnvelope;
import org.ksoap2.serialization.SoapObject;
import org.ksoap2.serialization.SoapSerializationEnvelope;
import org.ksoap2.transport.HttpTransportSE;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;

public class soapWebService extends Activity {
    // método en el web service
    private static final String SOAP_ACTION =
        "http://webservices.cloanto.com/currencyserver/Version";
    // método al que estamos llamando
    private static final String METHOD_NAME =
        "Version";
    private static final String NAMESPACE =
        "http://webservices.cloanto.com/currencyserver/";
    // dirección del web service
    private static final String URL =
```

```
"http://fx.cloanto.com/webservices/CloantoCurrencyServer.asmx?  
WSDL";
```

Hemos importado la librería de kSOAP2 para poder usarlas:

```
/** Called when the activity is first created. */  
@Override  
public void onCreate(Bundle icicle) {  
    super.onCreate(icicle);  
    setContentView(R.layout.main);  
  
    Button btnConsultar = (Button)  
findViewById(R.id.button1);  
    btnConsultar.setOnClickListener(new  
View.OnClickListener() {  
        public void onClick(View view) {  
            String sCadenaDevuelta = "no ha devuelto  
nada";  
            // definimos el objeto soap pasandole la  
            // direccion y el metodo  
            SoapObject request = new  
SoapObject(NAMESPACE, METHOD_NAME);  
  
            request.addProperty("licenseKey", "4");  
  
            SoapSerializationEnvelope envelope = new  
SoapSerializationEnvelope(  
                SoapEnvelope.VER11);  
  
            SoapObject result;  
            envelope.dotNet = true;  
            envelope.bodyOut = request;  
            envelope.setOutputSoapObject(request);  
  
            // realizamos la conexion  
            HttpTransportSE httpTransport = new  
HttpTransportSE(URL);  
  
            try {  
                {  
                    httpTransport.debug = true;  
                    httpTransport.call(SOAP_ACTION,  
envelope);  
                    result = (SoapObject)envelope.bodyIn;  
                    sCadenaDevuelta = (result.toString());  
                }  
                catch (Exception exception)  
                {  
                    sCadenaDevuelta =  
(exception.toString());  
                }  
                Toast.makeText(soapWebService.this,  
sCadenaDevuelta,  
                Toast.LENGTH_SHORT).show();  
            }  
        }  
    });  
}
```

```
        }
    });
}
}
```

En el método `onCreate()` hemos definido un `setOnClickListener()` para implementar en el método `onClick()` del botón las consultas al Web Service.

Necesitamos configurar el `layout` para que nos presente los datos recibidos del servidor. Debemos insertar el siguiente código en el archivo `main.xml` del directorio `res/layout`:

```
<?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">
    <Button
        android:layout_width= "wrap_content"
        android:layout_height= "wrap_content"
        android:id= "@+id/button1"
        android:text= "Button"> </Button>
</LinearLayout>
```

Hemos definido un botón que al ser pulsado será el que realice la conexión con el servidor, le envíe los parámetros y nos devuelva la información que tiene almacenada.

Si observamos nuestra aplicación en ejecución, veremos cómo nos devuelve el mismo resultado que si lo hiciésemos desde la página web:

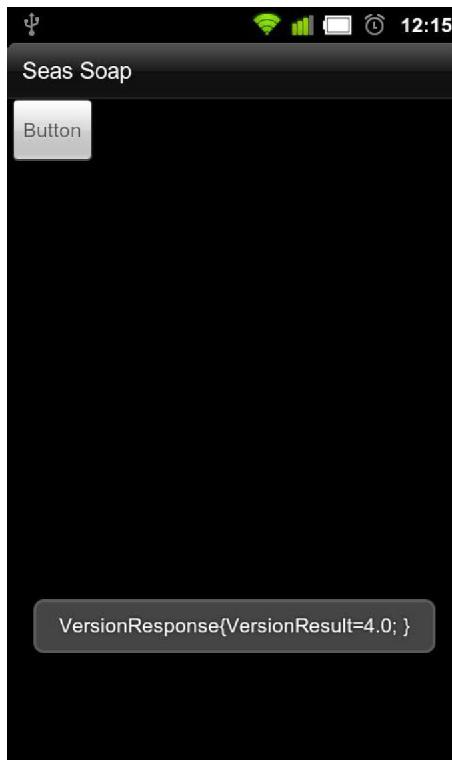


Figura 4.9. Resultado devuelto desde el terminal Android.

El resultado obtenido desde el navegador introduciendo la dirección del Servicio Web <http://fx.cloanto.com/webservices/CloantoCurrencyServer.asmx> sería el siguiente:

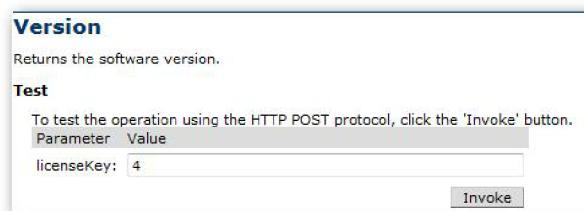


Figura 4.10. Invocación del Servicio Web.

<**stringstring**>

Figura 4.11. Respuesta del Servicio Web.



La aplicación completa “**SeasSoapWebService**” se puede descargar desde la plataforma de estudio.

4.1.5. CREAR UN WEB SERVICE EN JAVA

Para crear un Servicio Web, vamos a descargarnos el *Apache Tomcat 7.0*. Accedemos a la página web: <http://tomcat.apache.org/download-70.cgi> y nos descargamos el archivo:

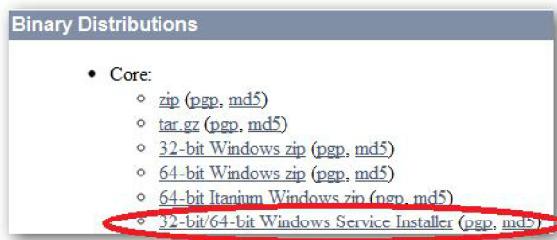


Figura 4.12. Archivos necesarios para la instalación de Tomcat.

Una vez descargado el archivo procedemos a su instalación.



Figura 4.13. Ventana inicial de instalación del Tomcat.

Aceptamos los términos de la licencia y continuamos:

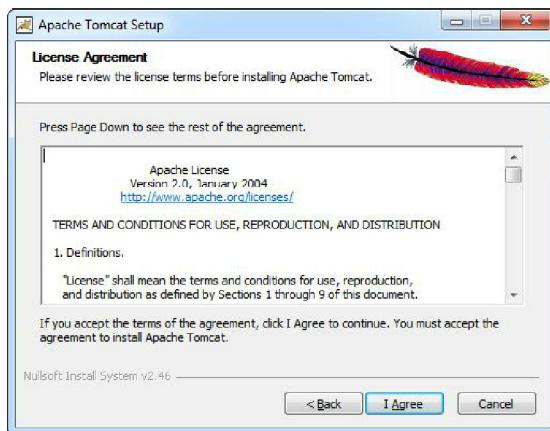


Figura 4.14. Ventana de términos de licencia.

Dejamos seleccionado por defecto lo que nos marca el tutorial de instalación:

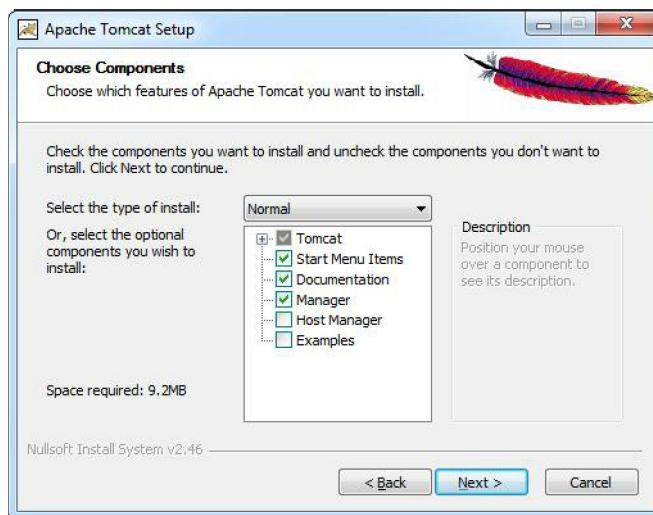


Figura 4.15. Ventana de selección de elementos a instalar.

El nombre de usuario y contraseña del servidor *Tomcat* lo dejamos en blanco:

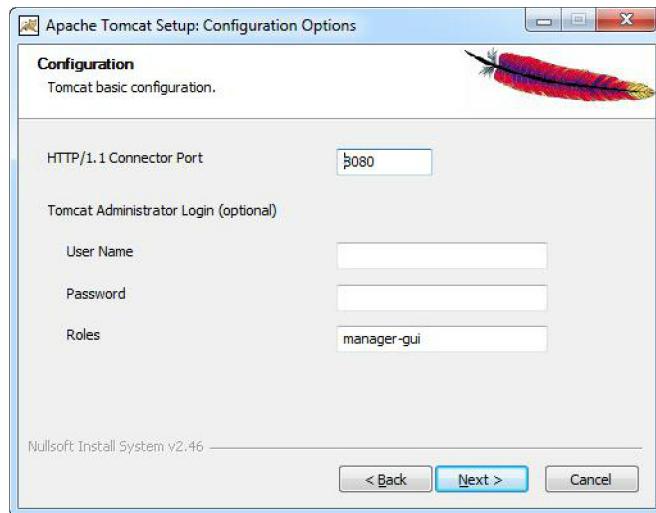


Figura 4.16. Ventana de introducción de usuario y contraseña del Tomcat.

En la ventana de selección de instalación de la máquina virtual java dejamos que nos la detecte y pulsamos sobre siguiente:

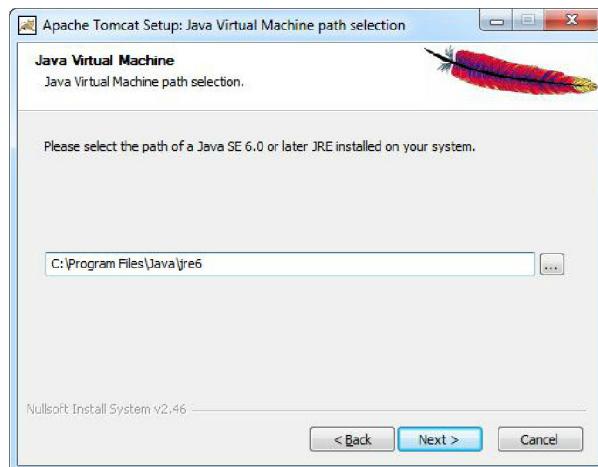


Figura 4.17. Ventana de selección de la máquina virtual Java.

Y en el siguiente diálogo seleccionamos el directorio de instalación de nuestro servidor:

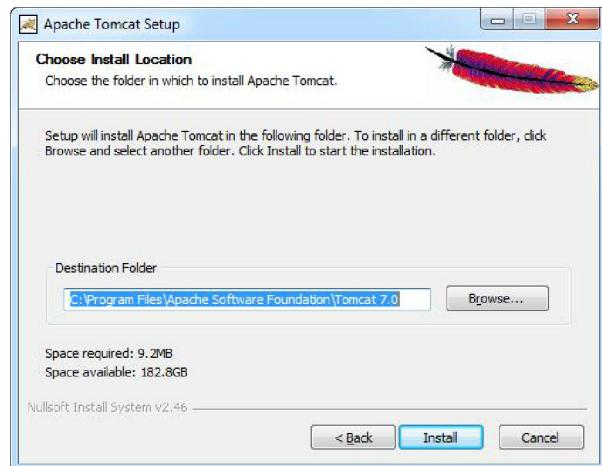


Figura 4.18. Ventana de selección de directorio de instalación.

Finalizamos la instalación del servidor *Tomcat*:

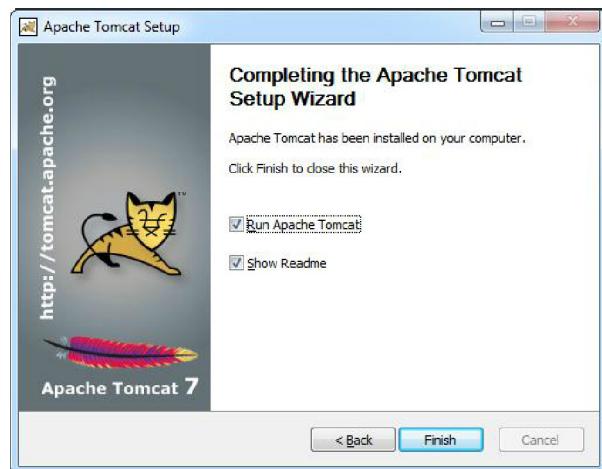


Figura 4.19. Pantalla de finalización de instalación.

Una vez instalado el *Apache Tomcat*, en el Eclipse vamos a añadirlo como servidor:

Vamos a *Menú* → *Window* → *Preferences* y en el apartado de server añadimos uno nuevo:

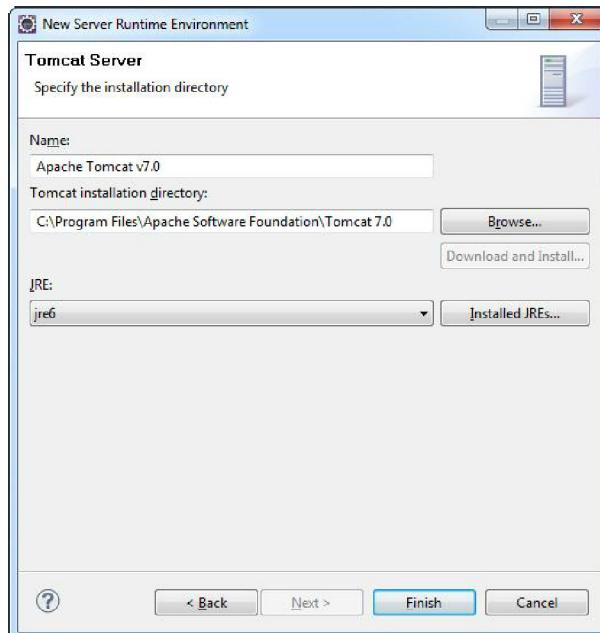


Figura 4.20. Diálogo de selección de nuevo Server.

Seleccionamos el directorio donde se nos ha instalado anteriormente y en la pestaña de *JRE* seleccionamos "jre6". Pulsamos sobre "Finish".

En nuestro workspace se nos ha tenido que añadir la siguiente carpeta:

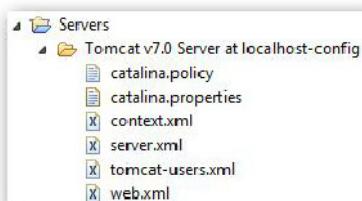


Figura 4.21. Carpeta que se añade tras la instalación correcta.

Ahora vamos a proceder a crear el *Web Service*. Para realizar esto vamos a *File → New → Project → Dynamic Web Project*:

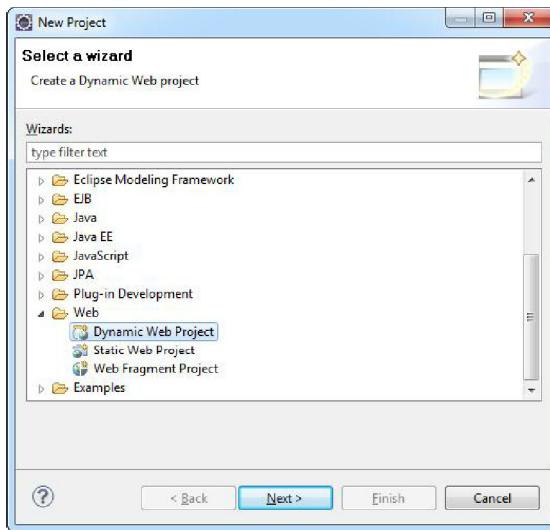


Figura 4.22. Ventana de creación de nuevo proyecto en Eclipse.

En la siguiente ventana pulsamos sobre “*Next*”:

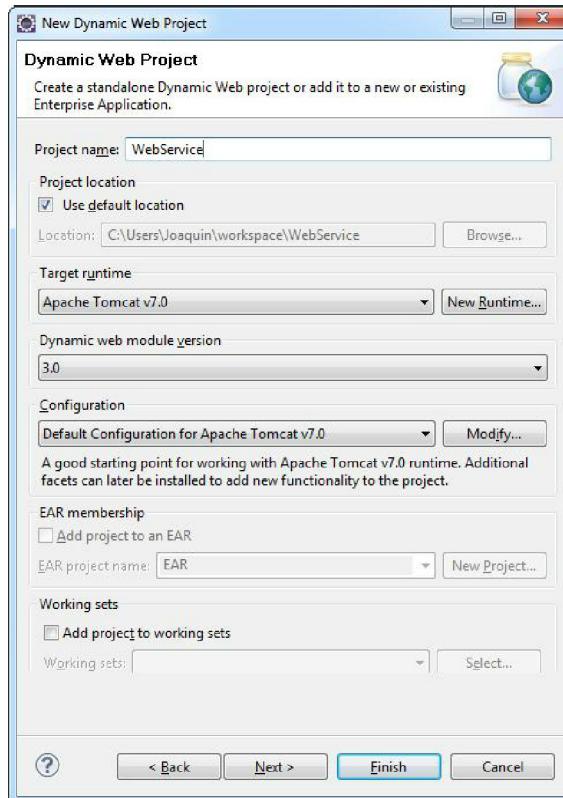


Figura 4.23. Diálogo de creación del Nuevo Proyecto Dinámico Web.

Una vez creado, añadimos un Nuevo paquete en el `src`, que lo llamaremos `com.seas.ws.servidor`.

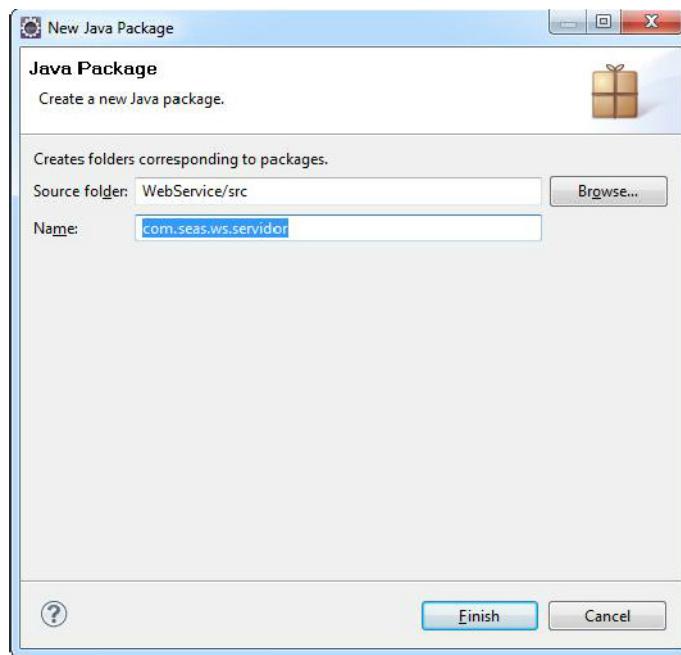


Figura 4.24. Diálogo Java Package.

Y dentro de este paquete añadiremos una nueva clase que llamaremos “Server”:

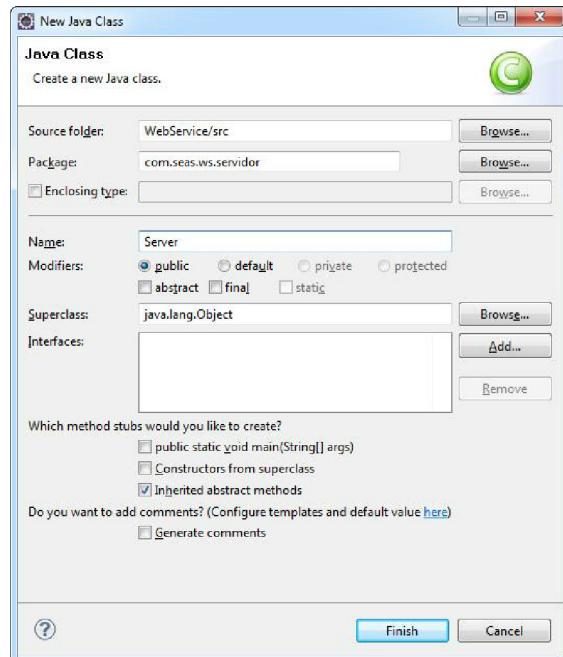


Figura 4.25. Diálogo de adición de nueva clase.

En esta clase insertaremos la lógica de nuestro servidor. Ahora creamos la vista de servidores:

```
package com.seas.ws.servidor;

public class Server {
    // implementa lógica de 'saludar'
    public String saludar(String nombre) {
        return "Bienvenido a Seas " + nombre;
    }

    // implementa lógica 'despedir'
    public String despedir(String nombre) {
        return "Adios " + nombre + ". Gracias por su visita a
Seas.";
    }
}
```

Creamos la vista de servidores. Para hacerlo accedemos a *Window* → *Show View* → *Others* → *Server*. Nos debería aparecer una nueva pestaña como la imagen siguiente:

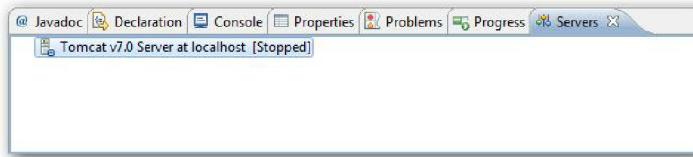


Figura 4.26. Pestaña Vista de Servidores.

Si nos aparece en blanco, debemos situarnos sobre la ventana, y con el botón derecho del ratón pulsamos sobre *New* → *Server*.

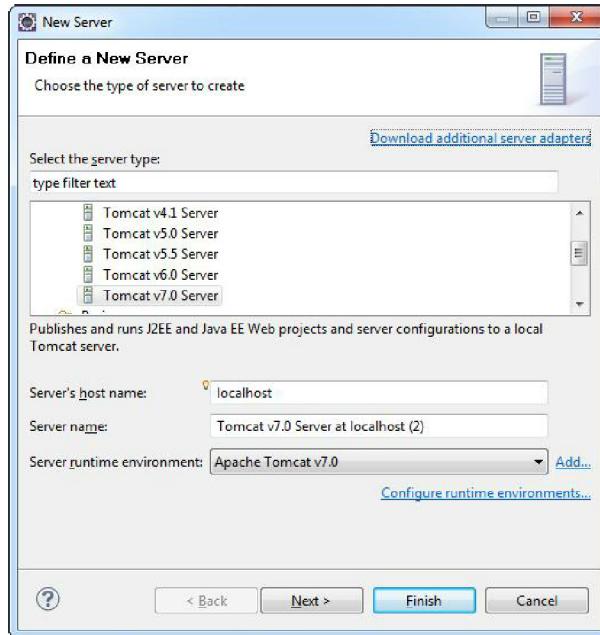


Figura 4.27. Diálogo para añadir un nuevo servidor.

Vamos a crear el Servicio Web, para ello, seleccionamos la clase creada y se pulsa el botón derecho del ratón. Seleccionamos *New → Others → Web Service* y seguimos el tutorial.

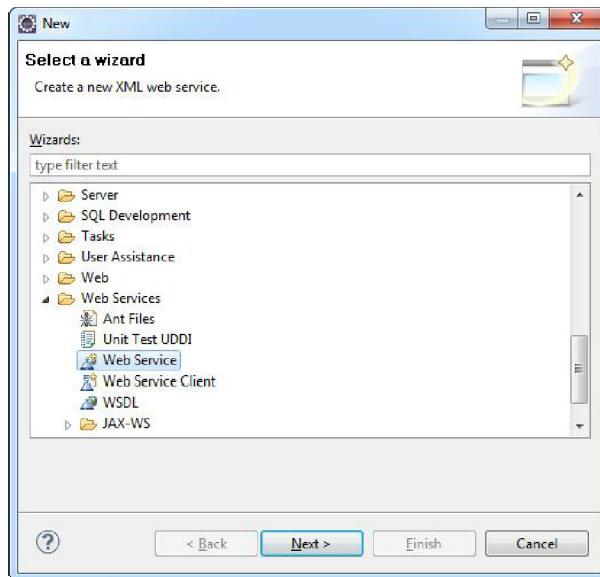


Figura 4.28. Diálogo de creación de un Web Service.

En la siguiente ventana seleccionamos “Next”:

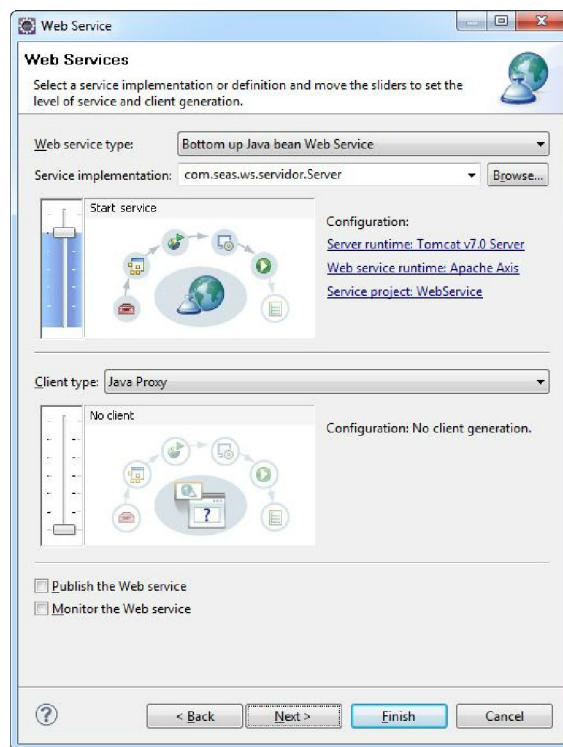


Figura 4.29. Diálogo de creación del Web Service.

Una vez pulsado “Finish”, vamos a probar que funciona correctamente. Para ello nos situamos sobre el archivo *Server.wsdl*, botón derecho del ratón, seleccionamos *WebService → Test With Web Services Explorer*. Veremos una pantalla parecida a la siguiente donde podemos probar los servicios definidos:

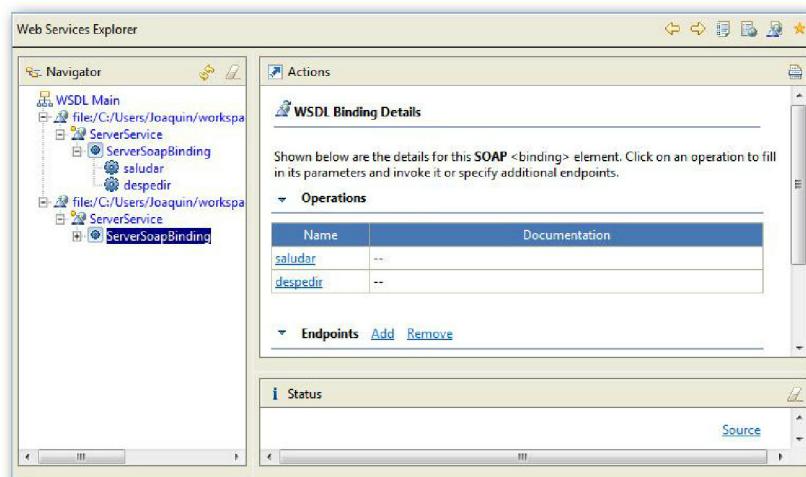


Figura 4.30. Vista del Web Services Explorer funcionando.

De esta manera hemos creado un Web Service en java con el que podemos realizar las pruebas necesarias.

4.1.6. CREAR UN WEB SERVICE EN .NET Y C#

En este punto vamos a crear un *Web Service* en *.Net* con *C#*, para poder realizar pruebas en local usando otras librerías diferentes que implementa *.Net*. Para realizarlo necesitaremos tener instalado *Visual Studio*.

Una vez abierto, nos vamos a *Menú → Nuevo → Proyecto*.

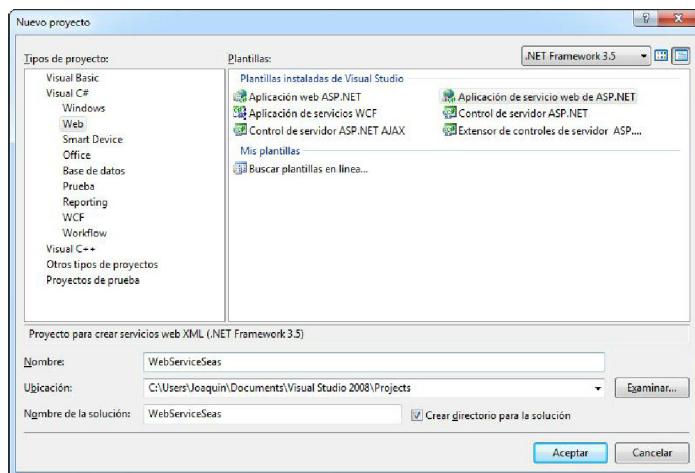


Figura 4.31. Ventana de selección de tipo de proyecto.

Allí seleccionamos “Aplicación de Servicios Web” y le damos un nombre. Pulsamos sobre “Aceptar”. Automáticamente nos crea el Servicio Web. Allí debemos añadir la lógica del servidor.

```
namespace WebServiceSeas
{
    /// <summary>
    /// Descripción breve de Service1
    /// </summary>
    [WebService(Namespace = "http://servidor.seas",
        Description = "Web Service en Localhost para realizar
Pruebas de acceso")]
    [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
    [ToolboxItem(false)]
    // Para permitir que se llame a este Servicio Web desde un
    script, usando ASP.NET AJAX,
    // quite la marca de comentario de la línea siguiente.
    // [System.Web.Services.ScriptService]
    public class MyService : System.Web.Services.WebService
    {
```

```

        [WebMethod (Description = "Metodo saludar. Le pasamos un
nombre y nos devuelve un string")]
        public string saludar (string Nombre)
        {
            return "Hola " + Nombre + ". Bienvenido a Seas";
        }

        [WebMethod (Description = "Metodo despedir. Le pasamos un
nombre y nos devuelve un string")]
        public string despedir(string Nombre)
        {
            return "Adios " + Nombre + ". Gracias por su visita";
        }
    }
}

```

Debemos añadir la etiqueta `WebMethod` para cada uno de ellos para que podamos acceder desde el *Web Service*.

En el explorador de soluciones, debemos situarnos sobre el archivo `MyService.asmx` y con el botón derecho del ratón pulsamos en “*Ver Marcado*”. Debemos tener algo parecido a esto. Asegurarse que la “*class*” coincide con el nombre de la que hemos definido nosotros.

```
<%@ WebService Language="C#" CodeBehind="MyService.asmx.cs"
Class="WebServiceSeas.MyService" %>
```

Si ejecutamos el proyecto, veremos en ejecución nuestro *Web Service*. Automáticamente se genera el archivo WSDL, que es el que contiene la descripción de nuestro servicio, con la documentación que se indicó en los atributos `WebService` y `WebMethod`. Podemos ver el fichero WSDL en la URL: <http://servidor.seas/WebServiceSeas/MyService.asmx?WSDL>.

Esta URL la usaremos más adelante para realizar las conexiones desde la aplicación del móvil.

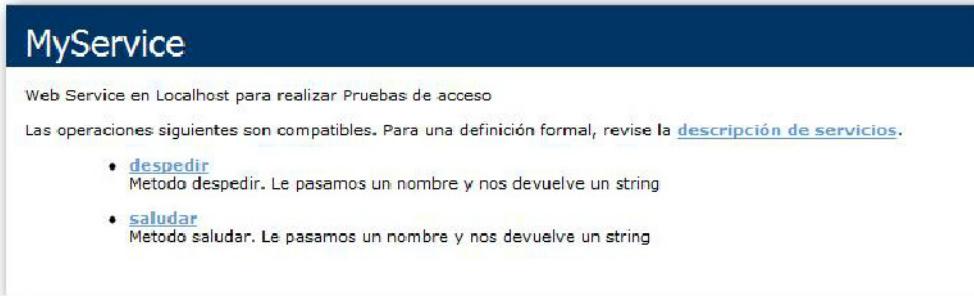


Figura 4.32. Ventana en el explorador al ejecutar el proyecto.

4.2. CLIENTE ANDROID-SERVIDOR PHP-MYSQL

En esta segunda parte de esta unidad vamos a realizar una aplicación con la que cargaremos en los *maps* de Google imágenes que estarán geoposicionadas. Dichas imágenes estarán almacenadas en una base de datos de *MySQL* y dispondremos de un servidor *php real*. Veremos cómo configurar dicho servidor. Cuando arrancemos la aplicación, nos cargará el mapa en la posición en la que nos encontramos, y nos mostrará las imágenes que nosotros hayamos configurado que se las descargarán directamente del servidor, a través de la red, que hemos configurado previamente en la web.

4.2.1. INTERFAZ DE USUARIO

En este punto vamos a realizar la aplicación, para ello vamos a crear un nuevo proyecto de Android e insertaremos los siguientes datos:

- **Project Name (Nombre del proyecto):** SeasTuristicLocationZaragoza
- **Build Target (Objetivo de compilación):** Android 2.2 (Google Apis)
- **Application name (Nombre de la aplicación):**
SeasTuristicLocationZaragoza
- **Package name (Nombre del paquete):**
com.seas.TuristicLocation.activityprincipal
- **Create Activity (Crear actividad):** SeasTuristicLocationZaragoza
- **Min SDK Version:** 8

Una vez que ya tenemos creado el nuevo proyecto, vamos a ir configurando cada una de las partes de la aplicación.

Lo primero, como ya hicimos en una aplicación anterior de mapas, debemos insertar nuestra *ApiKey* para que cuando ejecutemos la aplicación nos muestre los mapas. Para ello vamos a crear en el directorio *res/layout* el archivo *mapa.xml*, en el que insertamos el siguiente código:

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

```

        android:orientation= "horizontal">

    </LinearLayout>

    <LinearLayout android:id= "@+id/Layout01"
        android:layout_width= "wrap_content"
        android:layout_height= "wrap_content">
        <com.google.android.maps.MapView
            android:id= "@+id/mapView" android:layout_width=
            "fill_parent"
            android:layout_height= "fill_parent"
        android:enabled="true"
            android:clickable= "true" android:apiKey=
        "Inserta_aqui_tu_APIKEY" />
        </LinearLayout>

    </LinearLayout>

```

Como vista de inicio vamos a definir una lista de iconos sobre los que se realizará la pulsación para posteriormente mostrarnos el mapa con dichos recursos. Para ello vamos a crear dos *xml* donde definiremos una *IconListView*.

El otro archivo que necesitaremos crear lo llamaremos *lista.xml* y los añadiremos en la carpeta de *res/layout*. Su código será el siguiente:

```

<?xml version= "1.0" encoding= "utf-8"?>

<LinearLayout
    xmlns:android= "http://schemas.android.com/apk/res/android"
    android:id= "@+id/settings"
    android:orientation= "vertical"
    android:layout_width= "fill_parent"
    android:layout_height= "fill_parent">

    <ListView
        android:id= "@+id/android:list"
        android:layout_width= "wrap_content"
        android:layout_height= "wrap_content" />

</LinearLayout>

```

Un *ListView* solo nos permite añadir líneas de texto. Lo bueno que tienen los *ListView* es que si tenemos más líneas de las que caben en pantalla, él se encargará de hacer *Scroll*. Pero para poder añadir los iconos tenemos que crear otro *Layout* que será cada una de las líneas que tendrá nuestro *ListView*. Este archivo lo llamaremos *iconrow.xml*:

```

<RelativeLayout xmlns:android=
"http://schemas.android.com/apk/res/android"
    android:layout_width= "fill_parent"
    android:layout_height= "?android:attr/listPreferredItemHeight"
    android:padding= "6dip">
    <ImageView
        android:id= "@+id/icon"
        android:layout_width= "wrap_content"

```

```

        android:layout_height= "fill_parent"
        android:layout_alignParentTop= "true"
        android:layout_alignParentBottom= "true"
        android:layout_marginRight= "6dip"/>
<TextView
    android:id= "@+id/row_toptext"
    android:layout_width= "fill_parent"
    android:layout_height= "fill_parent"
    android:layout_toRightOf= "@+id/icon"
    android:layout_alignParentBottom= "true"
    android:layout_alignParentRight= "true"
    android:singleLine= "true"
    android:ellipsize= "marquee"/>

</RelativeLayout>

```



Figura 4.33. Vista de la pantalla principal de la aplicación.

Con esto hemos configurado nuestra página de inicio que nos cargará los mapas de Google en cuanto pulsemos sobre uno de nuestros iconos.

Para poder tener bien organizados los paquetes, vamos a crear uno nuevo donde insertaremos las clases que necesitaremos para la lógica de los mapas. A este paquete lo hemos llamado en nuestro caso:

com.seas.TuristicLocation.herramientasmapas

Una vez realizado este proceso, crearemos una nueva clase llamada `Local.java`, donde asignaremos a cada línea del `ListView` un elemento de esta clase:

```
package com.seas.TuristicLocation.herramientasmapas;

public class Local {

    private String localName;
    private int localImage;

    public String getLocalName() {
        return localName;
    }

    public void setLocalName(String localName) {
        this.localName = localName;
    }

    public int getLocalImage() {
        return localImage;
    }

    public void setLocalImage(int i) {
        this.localImage = i;
    }

}
```

Una vez gestionado el contenido, necesitamos un Adapter que se encargará de pasar la información de cada elemento de la clase al ListView. Para ello debemos crear la clase VistaListaIconos.java. Esta clase la insertaremos en el paquete activityprincipal:

```
package com.seas.TuristicLocation.activityprincipal;

import java.util.ArrayList;

import android.app.ListActivity;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.util.Log;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ArrayAdapter;
import android.widget.ImageView;
import android.widget.ListView;
import android.widget.TextView;

public class VistaListaIconos extends ListActivity {
    private ArrayList<Local> m_locals = null;
    private IconListViewAdapter m_adapter;
    final public static String MyKey = "mikey";

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.lista);
```

```

        // Al crear la clase se inicializa el ListView que
        muestra los menus
        m_locals = new ArrayList<Local>();
        this.m_adapter = new IconListViewAdapter(this,
R.layout.iconrow,
                m_locals);
        setListAdapter(this.m_adapter);

        inicializarSitios();

    }

@Override
protected void onListItemClick(ListView l, View v, int
position, long id) {
    Local local = (Local) l.getItemAtPosition(position);

    // Cargamos el Bundle con la colección de objetos que
    pasaremos
    // a la siguiente vista
    Bundle bundle = new Bundle();
    bundle.putString(MyKey, local.getLocalName());

    // Creamos la vista de Lista de objetos y le pasamos la
    // colección de objetos a mostrar
    Intent myIntent = new Intent(this, CargaMapa.class);
    myIntent.putExtras(bundle);

    startActivityForResult(myIntent, 0);

}

```

En el método `onListItemClick()` cargamos el `Bundle` con los objetos que le devolveremos a la clase `CargaMapa` que lo ha instanciado, haciendo esto en la variable `MyKey`.

```

private void inicializarSitios() {

    try {
        // creamos los sitios
        m_locals = new ArrayList<Local>();
        Local o1 = new Local();
        o1.setLocalName("Bancos");
        o1.setLocalImage(R.drawable.visa);
        Local o2 = new Local();
        o2.setLocalName("Gasolineras");
        o2.setLocalImage(R.drawable.gas);
        Local o3 = new Local();
        o3.setLocalName("Hospitales");
        o3.setLocalImage(R.drawable.hospital);
        Local o4 = new Local();
        o4.setLocalName("Restaurantes");
        o4.setLocalImage(R.drawable.restaurant);
        Local o5 = new Local();
        o5.setLocalName("Tiendas");
        o5.setLocalImage(R.drawable.shop);
        Local o6 = new Local();
    }
}

```

```
        o6.setLocalName("Seas");
        o6.setLocalImage(R.drawable.seas);
        Local o7 = new Local();
        o7.setLocalName("Todos");
        o7.setLocalImage(R.drawable.all);
        // añadimos los sitios al Array
        m_locals.add(o1);
        m_locals.add(o2);
        m_locals.add(o3);
        m_locals.add(o4);
        m_locals.add(o5);
        m_locals.add(o6);
        m_locals.add(o7);
        Log.i("Locales añadidos ", "" + m_locals.size());
    } catch (Exception e) {
        Log.e("BACKGROUND_PROC", e.getMessage());
    }

    if (m_locals != null && m_locals.size() > 0) {
        for (int i = 0; i < m_locals.size(); i++)
            m_adapter.add(m_locals.get(i));
    }

    m_adapter.notifyDataSetChanged();
}

public class IconListViewAdapter extends ArrayAdapter<Local> {

    private ArrayList<Local> items;

    public IconListViewAdapter(Context context, int textViewResourceId,
        ArrayList<Local> items) {
        super(context, textViewResourceId, items);
        this.items = items;
    }

    @Override
    public View getView(int position, View convertView,
        ViewGroup parent) {
        View v = convertView;
        if (v == null) {
            LayoutInflater vi = (LayoutInflater)
getSystemService(Context.LAYOUT_INFLATER_SERVICE);
            v = vi.inflate(R.layout.iconrow, null);
        }
        Local o = items.get(position);
        if (o != null) {

            // poblamos la lista de elementos

            TextView tt = (TextView)
v.findViewById(R.id.row_toptext);
            ImageView im = (ImageView)
v.findViewById(R.id.icon);

            if (im != null) {
```

```
        im.setImageResource(o.getLocalImage());
    }
    if (tt != null) {
        tt.setText(o.getLocalName());
    }
}
return v;
}
}
```

El método `IIconListViewAdapter` es el que se va a encargar de recoger los datos que le iremos pasando y ponerlos en el `ListView`. Con el método `findViewById` podemos acceder a los componentes de la fila y pasarle los datos que necesitemos.

Una vez hecho esto, vamos a configurar la clase java que se encargará de crear la interfaz para un cliente HTTP. Debemos comentar que los clientes HTTP encierran una mezcla heterogénea de objetos necesarios para ejecutar peticiones HTTP para poder manipular las *cookies*, la autenticación, la administración de la conexión, y otras características. La seguridad de los clientes HTTP dependerá de la implementación y configuración del cliente específico.

Ahora vamos a configurar el archivo `AndroidManifest.xml`, en el que vamos a definir dos `Activities`, marcando una de ellas como la principal y que será la que se mostrará cuando arranque la aplicación:

```
<?xml version= "1.0" encoding= "utf-8"?>
<manifest xmlns:android=
"http://schemas.android.com/apk/res/android"
    package= "seas.activity.principal"
    android:versionCode= "1"
    android:versionName= "1.0">
    <application android:icon= "@drawable/icon" android:label=
"@string/app_name">
        <uses-library android:name= "com.google.android.maps"/>
        <activity android:name=
"com.seas.TuristicLocation.activityprincipal.SeaTuristicLocationZar
agoza"
            android:label= "@string/app_name">
        </activity>
        <activity android:name=
"com.seas.TuristicLocation.activityprincipal.VistaListaIconos"
            android:label= "@string/app_name">
            <intent-filter>
                <action android:name= "android.intent.action.MAIN"
/>
            <category android:name=
"android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
    <uses-permission android:name= "android.permission.INTERNET" />
```

```
<uses-permission android:name=
"android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name=
"android.permission.ACCESS_COARSE_LOCATION" />
</manifest>
```

Podemos ver cómo hemos definido como actividad principal com.seas.TuristicLocation.activityprincipal.VistaListaIconos. Fijaros cómo hemos puesto el nombre del paquete para que sepa dónde encontrarlo. Si lo omitimos, no funcionará y dará error al arrancar. También hemos dado permisos para usar el GPS y el acceso a internet con android.permission, y el uso de la librería de google.maps.

Vamos a comenzar creando la Activity principal VistaListaIconos.java, que será la que nos muestre al arrancar la aplicación.

Nuestra aplicación al arrancar, va a mostrar un listado con los recursos insertados en la base de datos. Cuando seleccionemos uno de ellos, nos va a cargar el mapa pintándonos las imágenes en las coordenadas que hemos recuperado de la petición al servidor, y mostrándonos la localización en la que nos encontramos.

Para ello hemos creado un paquete com.seas.TuristicLocation.herramientasmapas donde insertaremos las clases para dibujar sobre los mapas. Una llamada MiItemizedOverlay.java que es la que nos va a pintar las imágenes, y la otra la llamamos MiOverlay.java con la que pintaremos nuestra posición en el mapa.

Definimos la clase MiItemizedOverlay.java

```
package com.seas.TuristicLocation.herramientasmapas;

import java.util.ArrayList;

import seas.activity.principal.R;
import android.app.AlertDialog;
import android.content.Context;
import android.graphics.drawable.Drawable;

import com.google.android.maps.ItemizedOverlay;
import com.google.android.maps.OverlayItem;
import com.seas.TuristicLocation.activityprincipal.SeasTuristicLocationZaragoza;

public class MiItemizedOverlay extends ItemizedOverlay<OverlayItem>
{
    ArrayList<OverlayItem> mOverlays = new
ArrayList<OverlayItem>();
    Context mContext;

    public MiItemizedOverlay(Context context) {
        super(boundCenterBottom(SeasTuristicLocationZaragoza.getInstance()
        .getInstan-
```

```

    .getResources().getDrawable(R.drawable.seas));
    mContext = context;
}

```

Hasta ahora hemos definido la clase MiItemizedOverlay que hereda de ItemizedOverlay, ya que necesitamos implementar esta clase, la cual puede manejar un conjunto de OverlayItem (donde cada objeto de esta clase es un marcador individual colocado sobre el mapa).

Declaramos el Array, donde pondremos cada uno de los objetos OverlayItem que deseemos en el mapa, el context e instanciamos la clase SeasTuristicLocationZaragoza. Definimos el constructor MiItemizedOverlay, donde definiremos el marcador por defecto para cada uno de los OverlayItems. Para que el Drawable realmente se dibuje, debemos tener sus límites definidos. Lo que queremos es que el punto central en la parte inferior de la imagen coincida con el punto de las coordenadas del mapa. Esto lo realizaremos con el método boundCenterBottom(). Con este método configuraremos la capacidad de manejar los eventos touch de los objeto Overlay. Primero necesitamos una referencia a la aplicación Context como miembro de esta clase. Por lo que añadimos Context mContext como miembro de la clase, para posteriormente inicializarlo con un nuevo constructor de la clase.

Con el método boundCenterBottomAux() ajustamos una imagen para que el punto 0,0 de ésta, esté en el centro:

```

// de la parte inferior
public Drawable boundCenterBottomAux(Drawable marker) {
    return boundCenterBottom(marker);
}

```

Para poder agregar nuevos OverlayItems al ArrayList, crearemos un nuevo método:

```

public void addOverlay(OverlayItem overlay) {
    // añadimos otra capa de imagen
    mOverlays.add(overlay);
    populate();
}

```

Cada vez que agreguemos un nuevo OverlayItem al ArrayList, debemos llamar al método populate() para el ItemizedOverlay, que leerá cada uno de los OverlayItems y los preparará para ser dibujados.

Cuando se ejecute el método `populate()`, llamaremos a `CreateItem(int)` en el `ItemizedOverlay` para poder recuperar cada `OverlayItem`. Tendremos que sobrescribir este método para leer correctamente desde el `ArrayList` y devolver el `OverlayItem` de la posición especificada por el entero dado. El método será el siguiente:

```
@Override  
protected OverlayItem createItem(int i) {  
    // creamos la imagen  
    return mOverlays.get(i);  
}
```

También debemos sobrescribir el método `size()` para devolver el número actual de elementos del `ArrayList`:

```
@Override  
public int size() {  
    // devolvemos el tamaño del ArrayList  
    return mOverlays.size();  
}
```

Y por último, sobrescribimos el método `OnTap(int)`, que será el responsable de controlar el evento cuando un elemento es pulsado por el usuario.

```
@Override  
protected boolean onTap(int index) {  
    // capturamos la imagen  
    OverlayItem item = mOverlays.get(index);  
    // definimos el cuadro al pulsar sobre la imagen  
    AlertDialog.Builder dialog = new  
    AlertDialog.Builder(mContext);  
    //asignamos los campos en el cuadro popup  
    dialog.setTitle(item.getTitle());  
    dialog.setMessage(item.getSnippet());  
    dialog.show();  
    return true;  
}
```

Vamos a crear la clase que será la que nos pinte en el mapa con nuestra posición geográfica. La insertaremos dentro del paquete

```
package com.seas.TuristicLocation.herramientasmapas;  
  
import android.graphics.Bitmap;  
import android.graphics.BitmapFactory;  
import android.graphics.Canvas;  
import android.graphics.Point;  
  
import com.google.android.maps.GeoPoint;  
import com.google.android.maps.MapView;  
import  
com.seas.TuristicLocation.activityprincipal.SeasTuristicLocationZara  
goza;
```

```

public class MiOverlay extends com.google.android.maps.Overlay {
    private GeoPoint punto;

    public MiOverlay(GeoPoint point) {
        super();
        this.punto = point;
    }

    @Override
    public boolean draw(Canvas canvas, MapView mapView, boolean
shadow,
                        long when) {
        super.draw(canvas, mapView, shadow);

        // se traduce el punto geo localizado a un punto en la
        pantalla
        Point scrnPoint = new Point();
        mapView.getProjection().toPixels(this.punto, scrnPoint);

        // se construye un bitmap a partir de la imagen
        Bitmap marker = BitmapFactory.decodeResource(
            SeasTuristicLocationZaragoza.getInstance().getResources(),
            seas.activity.principal.R.drawable.persona);

        // se dibuja la imagen del marker
        canvas.drawBitmap(marker, scrnPoint.x -
marker.getWidth() / 2,
                           scrnPoint.y - marker.getHeight() / 2, null);

        return true;
    }
}

```

Esta clase será llamada cada vez que se actualice la localización GPS, desde el LocationListener(). Lo veremos a continuación.

Vamos a crear ahora la clase que se encargará de realizar la geolocalización. La vamos a insertar dentro del paquete de herramientasmapas y la llamaremos MyLocationListener.java:

```

package com.seas.TuristicLocation.herramientasmapas;

import
com.seas.TuristicLocation.activityprincipal.SeasTuristicLocationZara
goza;

import android.content.Intent;
import android.location.Location;
import android.location.LocationListener;
import android.os.Bundle;
import android.widget.Toast;

public class MyLocationListener implements LocationListener {

    // CUANDO EL PROVEEDOR OBSERVA QUE HAY UN CAMBIO DE POSICIÓN

```

```
public void onLocationChanged(Location loc) {
    try {
        String longitud;
        String latitud;
        if (loc != null) {
            longitud =
                String.valueOf(loc.getLongitude());
            latitud = String.valueOf(loc.getLatitude());
        }
        SeasTuristicLocationZaragoza.getInstance()
            .anadePuntoAlMapaConOverlay(latitud, longitud);
        // overlayItemAnterior = overlay;
    } catch (Exception e) {
        Toast.makeText(
            SeasTuristicLocationZaragoza.getInstance().getBaseContext(),
            "Error al cargar la Geolocalización. ",
            Toast.LENGTH_SHORT)
            .show();
    }
}

///////////////////////////////////////////////////////////////////
// AYUDAN A MEJORAR LA LOCALIZACIÓN EN EL CASO DE QUE UN
PROVEEDOR FALLE.
// MENSAJES Y RECURRIR A OTROS PROVEEDORES
///////////////////////////////////////////////////////////////////
public void onProviderDisabled(String provider) {
    // TODO Auto-generated method stub
    Intent intent = new Intent(
        android.provider.Settings.ACTION_LOCATION_SOURCE_SETTINGS);
    SeasTuristicLocationZaragoza.getInstance().startActivity(intent);
}

public void onProviderEnabled(String provider) {
    // TODO Auto-generated method stub
}

public void onStatusChanged(String provider, int status,
Bundle extras) {
}
```

Hemos implementado la clase MyLocationListener que hereda de LocationListener, cuando el sistema encuentra la posición GPS. Cuando ese llame al método onLocationChanged(), nos indicará que la posición GPS ha cambiado y se actualizará en el mapa pintando la imagen en la nueva posición.

Continuamos con la creación de la clase que se encargará de recoger los datos almacenados en el servidor, darles formato y presentarlos en pantalla. La clase la llamaremos Post.java y realizará una petición POST al servidor:

```
package com.seas.TuristicLocation.conexion_http_post;

import java.io.BufferedReader;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.util.ArrayList;
import java.util.List;

import org.apache.http.HttpEntity;
import org.apache.http.HttpResponse;
import org.apache.http.NameValuePair;
import org.apache.http.client.HttpClient;
import org.apache.http.client.entity.UrlEncodedFormEntity;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.impl.client.DefaultHttpClient;
import org.apache.http.message.BasicNameValuePair;
import org.json.JSONArray;

import android.util.Log;

public class Post {
    private InputStream is = null;
    private String respuesta = "";

    private void conectaPost(ArrayList<String> parametros, String
URL)
        throws Exception {
        ArrayList<NameValuePair> nameValuePairs = null;
        HttpClient httpclient = null;
        HttpPost httppost = null;
        HttpResponse response = null;
        HttpEntity entity = null;
        try {
            httpclient = new DefaultHttpClient();
            httppost = new HttpPost(URL);
            nameValuePairs = new ArrayList<NameValuePair>();

            if (parametros != null) {
                for (int i = 0; i < parametros.size() - 1; i
+= 2) {
                    nameValuePairs.add(new
BasicNameValuePair(
                            parametros.get(i),
                            parametros.get(i + 1)));
                }
            }
            httppost.setEntity(new
UrlEncodedFormEntity(nameValuePairs));
        }
        response = httpclient.execute(httppost);
```

```
        entity = response.getEntity();
        is = entity.getContent();
    } catch (Exception e) {
        Log.e("log_tag", "Error in http connection " +
e.toString());
        throw new Exception("Error al conectar con el
servidor. ");
    } finally {
        if (entity != null) {
            entity = null;
        }
        ;
        if (response != null) {
            response = null;
        }
        ;
        if (httppost != null) {
            httppost = null;
        }
        ;
        if (httpClient != null) {
            httpClient = null;
        }
        ;
    }
}
```

En el método `conectaPost()` hemos definido una conexión `HttpPost` al servidor de `php` pasándole los parámetros desde la clase `CargaMapa`. Ahora necesitamos un método que obtenga la respuesta desde el servidor y la convierta en `string` para posteriormente devolverla al `SeasTuristicLocationZaragoza`:

```
private void getRespuestaPost() throws Exception {
    BufferedReader reader = null;
    try {
        reader = new BufferedReader(
            new InputStreamReader(is, "iso-8859-
1"), 8);
        StringBuilder sb = new StringBuilder();
        String line = null;
        while ((line = reader.readLine()) != null) {
            sb.append(line + "\n");
        }
        is.close();
        respuesta = sb.toString();
        Log.e("log_tag", "Cadena JSON " + respuesta);
    } catch (Exception e) {
        Log.e("log_tag", "Error converting result " +
e.toString());
        throw new Exception(
            "Error al recuperar las imágenes del
servidor. ");
    } finally {
        if (reader != null) {
            reader.close();
        }
    }
}
```

```

        }
    }
}

```

Recuperamos los datos del servidor con el método JSONArray getServerData():

```

public JSONArray getServerData(ArrayList<String> parametros,
String URL)
        throws Exception {
    JSONArray jsonArray = null;
    try {
        conectaPost(parametros, URL);
        getRespuestaPost();
        jsonArray = getJsonArray();
        return jsonArray;
    } catch (Exception e) {
        throw new Exception(e.getMessage());
    }
}
}

```



JSON

Es la abreviatura de *JavaScript Object Notation*, y es un formato ligero que se utiliza para el intercambio de datos. JSON no requiere el uso de XML.

Y por último, ese string devuelto lo debemos convertir en un JSON Array con el que poder trabajar:

```

@SuppressWarnings("finally")
private JSONArray getJsonArray() throws Exception {
    JSONArray jArray = null;
    try {
        jArray = new JSONArray(respuesta);
    } catch (Exception e) {
        throw new Exception("Error al convertir a
JSONArray.");
    } finally {
        return jArray;
    }
}

```

Necesitamos crear la clase ServiciosTuristicLocation.java que será la encargada de realizar la conexión al servidor *php* y recoger los datos contenidos en él para posteriormente pintarlos sobre el mapa. La crearemos dentro del paquete Su código será el siguiente:

```

package com.seas.TuristicLocation.pantallainicial.servicios;

import java.io.IOException;
import java.net.HttpURLConnection;
import java.net.URL;
import java.util.ArrayList;

```

```

import org.json.JSONArray;
import org.json.JSONObject;

import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.graphics.drawable.Drawable;
import android.widget.ImageView;
import android.widget.Toast;

import com.google.android.maps.GeoPoint;
import com.google.android.maps.OverlayItem;
import
com.seas.TuristicLocation.activityprincipal.SeasTuristicLocationZara
goza;
import com.seas.TuristicLocation.conexion_http_post.Post;

public class ServiciosTuristicLocation {
    private Bitmap downloadFile(String imageHttpAddress) {
        URL imageUrl = null;
        Bitmap loadedImage = null;
        HttpURLConnection conn = null;
        try {
            imageUrl = new URL(imageHttpAddress);
            conn = (HttpURLConnection)
imageUrl.openConnection();
            conn.connect();
            loadedImage =
BitmapFactory.decodeStream(conn.getInputStream());
        } catch (IOException e) {
            Toast.makeText(
SeasTuristicLocationZaragoza.getInstance()
                    .getApplicationContext(),
                    "Error cargando la imagen: " +
e.getMessage(),
                    Toast.LENGTH_LONG).show();
        } finally {
            if (conn != null) {
                conn.disconnect();
            }
        }
        return loadedImage;
    }
}

```

En el código anterior, declaramos el método `downloadFile()` que será el que se encargue de descargar la imagen del servidor cuando se le pase la URL de la imagen (`imageHttpAddress`). Dicha imagen la devolveremos para que pueda ser mostrada en el siguiente método:

```

private OverlayItem getOverlayItem(String nombre, String
descripcion,
                                    String latitud, String longitud, Drawable
drawable) {
    String coordinates[] = { latitud, longitud };
    double lat = Double.parseDouble(coordinates[0]);
    double lng = Double.parseDouble(coordinates[1]);

```

```
GeoPoint point = new GeoPoint((int) (lng * 1E6), (int)
(lat * 1E6));
OverlayItem overlayitem = new OverlayItem(point, nombre,
descripcion);

overlayitem.setMarker(SeasTuristicLocationZaragoza.getInstance()
()

.getItemizedOverlay().boundCenterBottomAux(drawable));
return overlayitem;
}

public void getSitiosTuristicos(String tipo) {
Post post = null;
try {
ImageView imagenView = new
ImageView(SeasTuristicLocationZaragoza
.getInstance().getBaseContext());
// INICIO Llamada a Servidor Web PHP
ArrayList<String> parametros = new
ArrayList<String>();
parametros.add("tipo");
parametros.add(tipo);
// Llamada a Servidor Web PHP
post = new Post();
JSONArray datos = post.getServerData(parametros,
"http://seas-
imagemaps.webatu.com/login.php");
if (datos != null && datos.length() > 0) {
for (int i = 0; i < datos.length(); i++) {
JSONObject json_data =
datos.getJSONObject(i);
int idSitio = json_data.getInt("ID");
String nombre =
json_data.getString("NOMBRE");
String descripcion =
json_data.getString("DESCRIPCION");
String longitud =
json_data.getString("LONGITUD");
String latitud =
json_data.getString("LATITUD");
String img =
json_data.getString("IMAGEN");
String urlImagen = "http://seas-
imagemaps.webatu.com/"
+ img;
Bitmap loadedImage =
downloadFile(urlImagen);

imagenView.setImageBitmap(loadedImage);
OverlayItem overlay =
getOverlayItem(nombre, descripcion,
latitud, longitud,
imagenView.getDrawable());
SeasTuristicLocationZaragoza.getInstance()
.anadePuntoAlMapaConItemizedOverlayEImagen(overlay);
}
}
```

```
        }

    } catch (Exception e) {
        Toast.makeText(
            SeasTuristicLocationZaragoza.getInstance().getBaseContext(),
            e.getMessage(),
            Toast.LENGTH_SHORT).show();
    }
} // FIN Llamada a Servidor Web PHP
}
```

Y ya por último, vamos a crear la clase `ServiciosTuristicLocation.java`. Esta clase será la que nos muestre el mapa con los recursos del servidor y con la posición de nuestra localización actual. Aunque es una clase bastante extensa, vamos a ir poco a poco insertando el código y comentándolo.

```
package com.seas.TuristicLocation.activityprincipal;

import java.util.List;

import seas.activity.principal.R;
import android.content.Context;
import android.location.Criteria;
import android.location.Location;
import android.location.LocationManager;
import android.os.Bundle;

import com.google.android.maps.GeoPoint;
import com.google.android.maps.MapActivity;
import com.google.android.maps.MapController;
import com.google.android.maps.MapView;
import com.google.android.maps.Overlay;
import com.google.android.maps.OverlayItem;
import
com.seas.TuristicLocation.herramientasmapas.MiItemizedOverlay;
import
com.seas.TuristicLocation.herramientasmapas.MyLocationListener;
import
com.seas.TuristicLocation.pantallainicial.servicios.ServiciosTuristi
cLocation;

public class SeasTuristicLocationZaragoza extends MapActivity {
    private LocationManager lm = null;
    private MyLocationListener locationListener = null;
    private String best = null;

    private MapView mapView = null;
    private MapController mc = null;
    private List<Overlay> mapOverlays = null;
    private MiItemizedOverlay itemizedoverlay = null;
    private ServiciosTuristicLocation serviciosTuristicLocation =
null;

    private static SeasTuristicLocationZaragoza
seasTuristicLocationZaragoza = null;
```

```

public static SeasTuristicLocationZaragoza getInstance() {
    if (seasTuristicLocationZaragoza == null) {
        seasTuristicLocationZaragoza = new
SeasTuristicLocationZaragoza();
    }
    return seasTuristicLocationZaragoza;
}

```

Hasta ahora hemos instanciado y definido todas las variables necesarias para realizar la presentación de las imágenes, la geoposición de las mismas.

```

@Override
public void onCreate(Bundle savedInstanceState) {
    /*
     * La clase Bundle sirve para contener tipos primitivos
y objetos de
     * otras clases. Con esta clase puedes pasar datos entre
distintas
     * activities. Implementa Parcelable Ejemplo de paso de
datos entre
     * activitys:
    */

    super.onCreate(savedInstanceState);
    // Asociamos la ventana mapa.xml al activity
    setContentView(R.layout.mapa);
    seasTuristicLocationZaragoza = this;
    serviciosTuristicLocation = new
ServiciosTuristicLocation();
    // Recuperamos el componente MapView que nos permite
mostrar el
    // mapa de google en la pantalla
    // Sólo se puede utilizar MapView en MapActivity
    // Librería: com.google.android.maps
    // Obtener ApiKey
    mapView = (MapView) findViewById(R.id.mapView);

    mc = mapView.getController();
    mapView.setBuiltInZoomControls(true);

    mapView.setTraffic(true);

    itemizedoverlay = new MiItemizedOverlay(this);

    // Recuperamos los lugares depende de la que se haya
pulsado
    Bundle extras = getIntent().getExtras();
    if (extras != null) {
        String parametro =
extras.getString(VistaListaIconos.MyKey);

        // utilizamos el parametro

        serviciosTuristicLocation.getSitiosTuristicos(parametro);
    }
    //GEOLOCALIZARME

    registraServiciosDeLocalizaciónGPS();
}

```

```
        mc.setZoom(12);
    }
```

En el método `onCreate()` hemos definido la vista del mapa.xml, recuperamos los lugares que deseamos mostrar en el mapa y realizamos la geolocalización con el método `registraServiciosDeLocalizaciónGPS()`, que definimos a continuación:

```
private void registraServiciosDeLocalizaciónGPS() {

    lm = (LocationManager)
getSystemService(Context.LOCATION_SERVICE); //
    Criteria criteria = new Criteria();
    // si el GPS = off --> best = Network
    criteria.setAccuracy(Criteria.ACCURACY_FINE);
    best = lm.getBestProvider(criteria, true);
    // a la última ubicación conocida.
    Location location = lm.getLastKnownLocation(best);
    if (location != null) {
        //
anadePuntoAlMapaConItemizedOverlay(String.valueOf(location.getLatitude()),String.valueOf(location.getLongitude()));
    }
    locationListener = new MyLocationListener();
    lm.requestLocationUpdates(best, 4000, 0,
locationListener);
    // Mostramos las coordenadas solo la primera vez que nos
localiza
    Toast.makeText(
        SeasTuristicLocationZaragoza.getInstance()
            .getBaseContext(),
        "Location changed : Lat: " +
loc.getLatitude()
            + " Lng: " +
loc.getLongitude(),
        Toast.LENGTH_SHORT).show();
    }
}
```

Criteria es una clase que indica los criterios para la selección del proveedor de ubicación. Estos proveedores se ordenan de acuerdo a los costos de precisión, el uso de la energía, la capacidad para informar de la altitud, la velocidad y el rumbo, y coste económico.

```
// añade imagen al mapa con Overlay
public synchronized void anadePuntoAlMapaConOverlay(String
latitud,
            String longitud) {
    String coordinates[] = { latitud, longitud };
    double lat = Double.parseDouble(coordinates[0]);
    double lng = Double.parseDouble(coordinates[1]);
    GeoPoint p;
    // Trabaja en microgrados para las coordenadas y por eso
hay que
            // multiplicar por un millón
```

```

        p = new GeoPoint((int) (lat * 1E6),
                          (int) (lng * 1E6));
        mapOverlays = mapView.getOverlays();
        mapOverlays.clear();
        com.seas.TuristicLocation.herramientasmapas.MiOverlay
marker = new com.seas.TuristicLocation.herramientasmapas.MiOverlay(
            p);
        mapOverlays.add(itemizedoverlay);
        mapOverlays.add(marker);
        mc.animateTo(p);
        mc.setZoom(14);
        mapView.invalidate();
    }
}

```

Con el método `anadePuntoAlMapaConOverlay()` dibujaremos nuestra posición en el mapa dependiendo de las coordenadas que nos dé, realizando la llamada a la clase `MiOverlay`. Y a continuación definimos otro método por si queremos realizarlo mediante el paso de objetos `ItemizedOverlay`.

```

public synchronized void
anadePuntoAlMapaConItemizedOverlayEIImagen(
        OverlayItem overlayItem) throws Exception {
    mapOverlays = mapView.getOverlays();
    itemizedoverlay.addOverlay(overlayItem);
    mapOverlays.add(itemizedoverlay);
    mc.animateTo(overlayItem.getPoint());
    mapView.invalidate();
}

```

Debemos sobrescribir el método `isRouteDisplayed()`, ya que los servidores de Google necesitan saber si debe mostrar o no algún tipo de información de las rutas, como puede ser un conjunto de instrucciones para la conducción.

```

@Override
protected boolean isRouteDisplayed() {
    return false;
}

```

A continuación definimos los estados del Activity:

```

/**
 * CONTROL DE ESTADOS DEL ACTIVITY
 */
@Override
protected void onDestroy() {
    // TODO Auto-generated method stub
    super.onDestroy();
}

@Override
protected void onPause() {
    // TODO Auto-generated method stub
    lm.removeUpdates(locationListener); // Detener las
actualizaciones
    super.onPause();
}

```

```
@Override  
protected void onResume() {  
    super.onResume();  
    // Empieza las actualizaciones.  
    lm.requestLocationUpdates(best, 4000, 10,  
locationListener);  
}
```

En el onPause() paramos las actualizaciones del posicionamiento y en el onResume() las empezamos de nuevo. Esto lo realizamos, por si pasamos la aplicación a un segundo plano. A continuación definimos los getter & setter de la clase:

```
/**  
 * GETTER/SETTER  
 *  
 * @return  
 */  
public MapView getMapView() {  
    return mapView;  
}  
  
public void setMapView(MapView mapView) {  
    this.mapView = mapView;  
}  
  
public MapController getMc() {  
    return mc;  
}  
  
public void setMc(MapController mc) {  
    this.mc = mc;  
}  
  
public MiItemizedOverlay getItemizedoverlay() {  
    return itemizedoverlay;  
}  
  
public List<Overlay> getMapOverlays() {  
    return mapOverlays;  
}  
}
```

Con esto nuestra aplicación quedaría definida y a la espera de la realización de los siguientes puntos para su total utilización. De todas formas le vamos a mostrar unas imágenes de su funcionamiento.

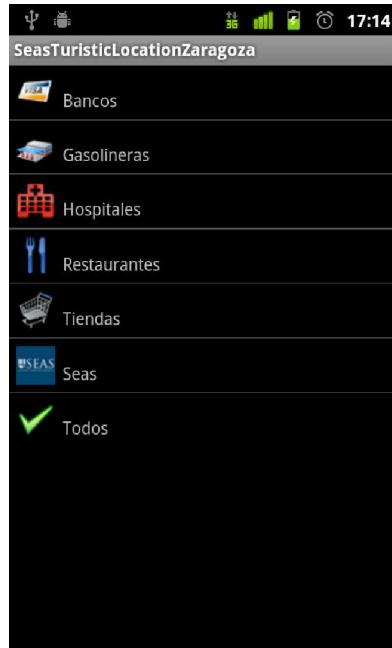


Figura 4.34. Pantalla de inicio de la aplicación.

En la imagen anterior observamos la pantalla de arranque de la aplicación, donde se muestran todos los tipos de recursos insertados. Si realizamos una pulsación sobre el elemento “*Todos*”, nos mostrará en pantalla todos los existentes en la base de datos. Si tenemos el GPS conectado, intentará realizar la localización por dicho dispositivo, sino lo hará por la red de 3G o en su defecto con la WiFi:



Figura 4.35. Búsqueda de señal GPS por la aplicación.

Una vez realizada dicha operación vemos todos los recursos geoposicionados sobre el mapa y un icono con nuestra posición sobre el mismo. Y por último, si pulsamos sobre cualquiera de ellos, nos dará la información almacenada en la base de datos de cada lugar:



Figura 4.36. Información de los recursos cuando pulsamos sobre alguna imagen descargada.

4.2.2. SERVIDOR PHP

En este apartado vamos a explicar cómo crear un servidor PHP gratuito válido para nuestra aplicación. Existen infinidad de páginas en internet que ofrece este servicio de “*Hosting*” gratuito.

Si usted ya es poseedor de un servidor propio, únicamente deberá insertar en la aplicación los datos de éste para poder funcionar con él y omitir los pasos de creación de uno gratuito.



El servidor elegido es el que se puede visitar en la siguiente página: <http://www.000webhost.com/?p=400>.

Allí encontramos un enlace para crear uno gratuito:

» Free Hosting	
Price	\$0.00
Disk Space	1500 MB
Data Transfer	100 GB / month
Add-on Domains	5
Sub-domains	5
E-mail Addresses	5
MySQL Databases	2
Free domain yourname.COM, .NET, .ORG, .INFO, .CO.UK	✗
Control Panel	Custom Panel
Reseller Hosting Feature	✗
Order Now	
Simple Site Builder	✓
Advanced Site Builder	✗
Support by Phone, Live Chat	✗
Assistance in Installing Scripts	✗
Help in Developing Your Website	✗
Backups	✓
Automated Weekly Backups	Limited
Uptime	99%
FTP Accounts	1

Figura 4.37. Selección de servidor gratuito.

Una vez pulsado en “Order Now”, nos pasa a la siguiente página donde debemos llenar los siguientes datos:

- **Domain:** este campo únicamente lo rellenaremos si disponemos ya de un dominio nuestro registrado, pero si no se dispone de él, rellenamos el campo siguiente.
- **Subdomain:** nosotros hemos elegido el de seas-imagemaps, igual que el nombre del proyecto.

Posteriormente nos pide nuestro nombre, una dirección de email y la contraseña.

En cuanto rellenemos esos datos, nos enviarán dos emails de confirmación como que se ha creado correctamente y la configuración del servidor para poder actuar con él.

The screenshot shows a web-based domain management interface. At the top, there is a search bar with the placeholder "Manage another domain" and a dropdown menu showing "seas-imagemaps.webatu.com". Below the search bar are two buttons: "Go" and "Create New". Underneath this, there is a section titled "List of your domains" featuring a house icon. A table below lists one domain entry:

» Domain	» Status	» Action
seas-imagemaps.webatu.com	Active	Go to CPanel

Figura 4.38. Información presentada por el servidor host de configuración.

Los datos del servidor no van a ser de dominio público, puesto que cada usuario deberá tener su propia cuenta de acceso al servidor, para que nadie más que él pueda modificarlo.

Más adelante veremos cómo configurar y añadir la base de datos. De momento vamos a ver cómo subir los archivos al servidor que vamos a necesitar para la que la aplicación los descargue cuando los necesite consultar.

Para ello debemos ir a la sección de “Files” para poder administrar los archivos que tenemos que subir al servidor.



Figura 4.39. Acceso al administrador de archivos del servidor.

Una vez que se accede, debemos subir las imágenes que vamos a usar posteriormente en nuestra aplicación y el archivo de configuración de “php.Login”, que anteriormente hemos configurado.



Figura 4.40. Archivos subidos al servidor de php.

Debemos subir todos los archivos al directorio “/public_html”, creado por el servidor y que será al que realmente tendremos acceso. Lo que se debe saber es que es un directorio de uso interno del servidor, por lo que no será necesario especificarlo en las rutas de acceso a nuestras imágenes.

4.2.3. BASE DE DATOS

Para crear nuestra base de datos, debemos realizarlo desde la misma página del servidor, el cual nos brinda una herramienta para la creación y configuración de la misma. Accedemos en la página del servidor, tal y como hemos descrito en el punto anterior. Bajamos en la página hasta que encontremos las herramientas siguientes:



Figura 4.41. Opciones del panel de control del servidor de Software/Services.



Pulsamos en la opción de para acceder al servicio de MySQL y proceder a crear la base de datos.

Nos volverá a aparecer una nueva página donde deberemos insertar la configuración que queramos darle a nuestra base de datos:

Manage another domain
seas-imagemaps.webatu.com ▾ Go Create New

Manage MySQL Databases

MySQL databases are required by many web applications including bulletin boards, content management systems, and others. To use MySQL, you need to create database and user, which will be automatically assigned to this database. Click for phpMyAdmin when database is created.

Important: MySQL Host for any database in this account is mysql12.000webhost.com , do not use localhost!

Create new database and user

MySQL database name: a7171134_
MySQL user name: a7171134_
Password for MySQL user: _____
Enter password again: _____

Figura 4.42. Pantalla de creación de la base de datos.

Los datos que nosotros hemos insertado para crear nuestra base de datos operativa con la aplicación desarrollada son los siguientes:

- **MySQL database name:** a7171134_seas
- **MySQL user name:** a7171134_seas
- **Password for MySQL user:** seas01
- **Enter password again:** seas01

Cada usuario deberá meter sus propios datos de configuración para poder acceder posteriormente desde la aplicación. Si el proceso ha sido correcto, el servidor nos presentará una pantalla similar a la siguiente:



Figura 4.43. Respuesta del servidor de creación de la base de datos.

Estos son los datos de configuración de nuestro servidor que posteriormente incorporaremos en nuestra aplicación para poder realizar la conexión y poder descargarnos los datos a ella, estemos en el lugar que estemos.

Si volvemos a acceder, veremos que la base de datos ya está creada. Y podemos pasar a modificarla para nuestras necesidades.

List of your current databases and users:			
» MySQL Database	» MySQL User	» MySQL Host	» Action
a7171134_seas	a7171134_seas	mysql12.000webhost.com	
[Go Back]			

Figura 4.44. Imagen de BBDD creadas en el servidor.

Ahora vamos a proceder a modificar nuestra base de datos. Para ello, debemos acceder desde el “CPanel” de la página principal del servidor a la sección de “phpMyAdmin”:



Figura 4.45. Sección de acceso para administrar la BBDD.

Una vez allí, pulsamos sobre el enlace de “Enter phpMyAdmin”:



Figura 4.46. Ventana de acceso a la BBDD.

Cuando accedemos la primera vez al panel de “phpMyAdmin”, nos encontramos una ventana como la siguiente. Nos está dando la información de nuestra base de datos y las herramientas para poder gestionarla:

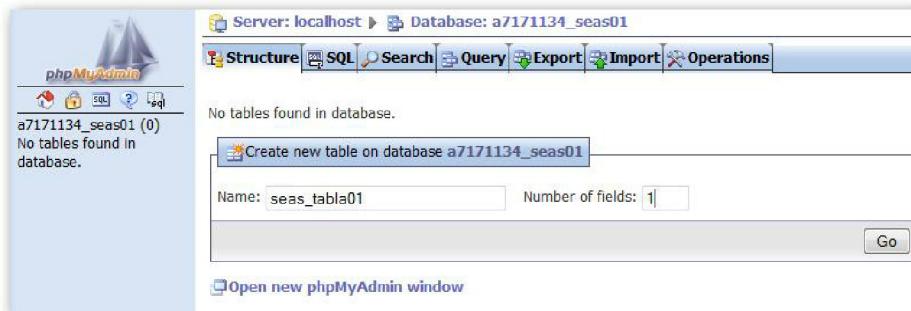


Figura 4.47. Ventana de inicio de phpMyAdmin.

Para crear una nueva tabla en la base de datos, rellenamos el campo “Name” con el nombre que le queremos asignar, y en “Fields”, el número de campos que tendrá nuestra tabla. Una vez llenados los campos, pulsamos sobre “Go” y pasamos a rellenar los datos de los campos de la tabla.



Figura 4.48. Pantalla de inserción de campos en nuestra tabla.

Los campos que necesitamos en nuestra base de datos van a ser "ID", "Nombre", "Descripcion", "Longitud", "Latitud" e "Imagen". Una vez insertados todos los campos, debemos ver que nuestra tabla debería quedar de la siguiente manera:

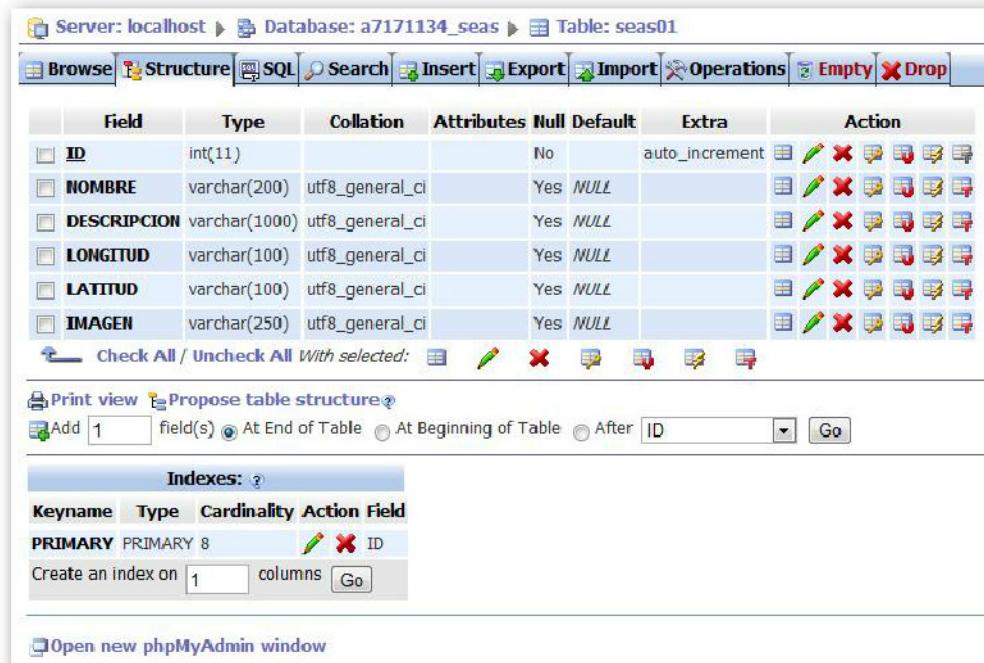


Figura 4.49. Campos de la tabla “seas01”.

Una vez que hemos insertado la descripción de todos los campos, debemos insertar en ellos unos pocos datos para poder realizar las pruebas necesarias y ver el funcionamiento real de la aplicación.

Para insertar dichos datos debemos irnos a la parte superior de la ventana a la opción de “Insert”. Una vez pulsado, nos aparecerá una ventana como la siguiente:

Field	Type	Function	Null	Value
ID	int(11)			
NOMBRE	varchar(200)		<input checked="" type="checkbox"/>	
DESCRIPCION	varchar(250)		<input checked="" type="checkbox"/>	
LONGITUD	varchar(100)		<input checked="" type="checkbox"/>	
LATITUD	varchar(100)		<input checked="" type="checkbox"/>	
IMAGEN	varchar(250)		<input checked="" type="checkbox"/>	

Figura 4.50. Tabla de registro de datos.

Para insertar debemos realizarlo sobre el campo de texto “Value”. Como el campo “ID” lo definimos anteriormente como “Autoincrement”, no será necesario que lo rellenemos, ya que se hará automáticamente.

Los campos que nosotros hemos creado son los que nos aparecerán definidos en la aplicación, cuando seleccionemos alguna de las categorías existentes:

	ID	NOMBRE	DESCRIPCION	LONGITUD	LATITUD	IMAGEN
<input type="checkbox"/>	1	Seas, Estudios Abiertos	El sistema de Formación Abierta de SEAS combina la...	41.67130099910545	-0.878981351852417	Seas.png
<input type="checkbox"/>	3	Hospital La Már	MA7 dispone de una amplia red de centros asistencia...	41.68586103098603	-0.8687353134155273	Hospitales.png
<input type="checkbox"/>	4	Seas	Seas Estudios Superiores Abiertos. Delegacion Cent...	41.64785539330573	-0.8823448419570923	Seas.png
<input type="checkbox"/>	5	Hospital Miguel Servet	Hospital Central de Zaragoza	41.634674480570886	-0.8996987342834473	Hospitales.png
<input type="checkbox"/>	6	Hospital General de la Defensa de Zaragoza	Hospital Militar de Zaragoza	41.63027962121499	-0.9045052528381348	Hospitales.png
<input type="checkbox"/>	7	Hospital Quiron Zaragoza	Clinica Quiron Zaragoza	41.63666261827313	-0.8937549591064453	Hospitales.png
<input type="checkbox"/>	8	CLINICA NTRA. SRA. DEL PILAR	La superación diaria y el esfuerzo en mejorar dia ...	41.63715975985198	-0.8889055252075195	Hospitales.png
<input type="checkbox"/>	9	Clinica Montpellier	En junio de 1.970 se da forma al edificio moderno ...	41.63626169485176	-0.9138822555541992	Hospitales.png
<input type="checkbox"/>	10	IberCaja	Sucursal de IberCeja	41.669152	-0.876903	Bancos.png
<input type="checkbox"/>	11	CAT	Sucursal CAT	41.669154	-0.876605	Bancos.png
<input type="checkbox"/>	12	Galerias Primero	Supermercado de Confianza, Galerias Primero	41.670817	-0.875522	Tiendas.png

Figura 4.51. Datos introducidos en la base de datos.

Hay que tener especial cuidado cuando insertemos el nombre de la imagen, puesto que deberá coincidir con el nombre de la imagen que anteriormente hemos subido al servidor *php*, y son las imágenes que se mostrarán posteriormente cuando se haga la solicitud al servidor.

Para facilitar el trabajo, adjuntamos el script de inserción de los datos en la base de datos, y de esta manera no hay que añadir campo por campo.

Para poder añadir dichos datos, deberemos realizarlo en la pestaña “SQL” del servidor:



Figura 4.52. Tabla de registro de datos.

Una vez allí copiamos y peguemos el siguiente código y cambiemos los datos que creamos conveniente para adaptarlo a nuestras necesidades:

```
INSERT INTO `seas01` VALUES(1, 'Seas, Estudios Abiertos', 'El sistema de Formacion Abierta de SEAS combina la metodologia a distancia con las nuevas tecnologias de comunicacion.', '41.67130099910645', '-0.878981351852417', 'Seas.png');

INSERT INTO `seas01` VALUES(3, 'Hospital La Maz', 'MAZ dispone de una amplia red de centros asistenciales y hospitalarios propios y concertados distribuidos por todo el estado, especificos para la prestacion de asistencia a nuestros trabajadores protegidos, y donde la prioridad es dar un servicio de ', '41.68586103098603', '-0.8687353134155273', 'Hospitales.png');

INSERT INTO `seas01` VALUES(4, 'Seas', 'Seas Estudios Superiores Abiertos. Delegacion Centro Zaragoza', '41.64785539330573', '-0.8823448419570923', 'Seas.png');

INSERT INTO `seas01` VALUES(5, 'Hospital Miguel Servet', 'Hospital Central de Zaragoza', '41.634874480570886', '-0.8996987342834473', 'Hospitales.png');

INSERT INTO `seas01` VALUES(6, 'Hospital General de la Defensa de Zaragoza', 'Hospital Militar de Zaragoza', '41.63027962121499', '-0.9045052528381348', 'Hospitales.png');
```

```

INSERT INTO `seas01` VALUES(7, 'Hospital Quiron Zaragoza',
'Clinica Quiron Zaragoza', '41.63666261827313', '-0.8937549591064453', 'Hospitales.png');

INSERT INTO `seas01` VALUES(8, 'CLINICA NTRA. SRA. DEL PILAR',
'La superacion diaria y el esfuerzo en mejorar dia a dia es la filosofia que orienta el trabajo en la Clinica Nuestra Senora del Pilar.', '41.63715975985198', '-0.8889055252075195',
'Hospitales.png');

INSERT INTO `seas01` VALUES(9, 'Clinica Montpellier', 'En junio de 1.970 se da forma al edificio moderno y funcional que alberga la Clinica Montpellier, que durante mas de 30 anos de servicio, se ha ganado a pulso un merecido prestigio dentro y fuera de la ciudad de Zaragoza.', '41.63626169485176', '-0.9138822555541992', 'Hospitales.png');

INSERT INTO `seas01` VALUES(10, 'IberCaja', 'Sucursal de IberCaja', '41.669152', '-0.876903', 'Bancos.png');

INSERT INTO `seas01` VALUES(11, 'CAI', 'Sucursal CAI', '41.669154', '-0.876605', 'Bancos.png');

INSERT INTO `seas01` VALUES(12, 'Galerias Primero', 'Supermercado de Confianza, Galerias Primero', '41.670817', '-0.875522', 'Tiendas.png');

INSERT INTO `seas01` VALUES(13, 'Gasolinera BP', 'Estacion de Servicio 24h', '41.666236', '-0.877179', 'Gasolineras.png');

```

4.2.4. CÓDIGO DE SERVIDOR

Como hemos visto anteriormente, la aplicación va a realizar la petición al servidor para que éste le devuelva un objeto JSON. **JSON es una representación de objetos en forma de cadena, que se usa sobre todo en javascript.**

La cadena que nos va a devolver nuestro servidor tras realizar la petición será de la siguiente manera:

```
{"ID":"1","NOMBRE":"Nombre","DESCRIPCION":"Descripcion","LONGITUD":"Longitud","LATITUD":"Latitud","IMAGEN":"Imagen.png"}
```

Podemos observar que es una representación muy sencilla donde los objetos devueltos van entre llaves.

Para crear el archivo usaremos código php. No es obligatorio tener grandes conocimientos de este lenguaje de programación. Para crear este archivo lo podemos realizar de dos maneras diferentes.

O bien creamos un archivo de texto plano con el block de notas, o lo podemos realizar directamente en el servidor con el editor que nos proporciona el mismo. Si lo hacemos de la primera forma, debemos guardar el archivo como “*login.php*”. Para hacerlo desde el mismo servidor, primero debemos acceder en la sección de “*Files*”, con la opción de “*File Manager*” al mismo:

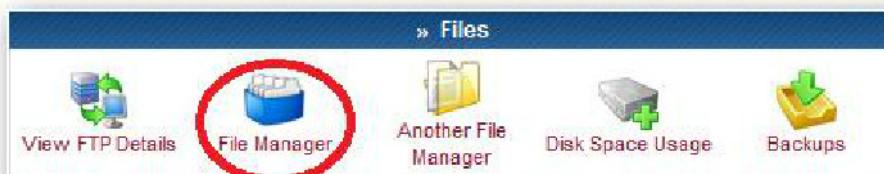


Figura 4.53. Acceso al administrador de archivos.

Una vez que accedemos, lo más seguro es que nos pida autenticarnos en el servidor con el nombre de usuario y la contraseña que nos enviaron al correo cuando lo creamos:

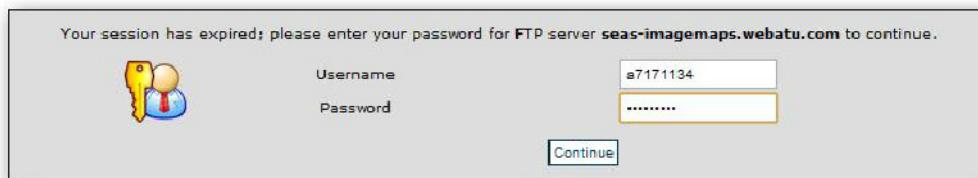


Figura 4.54. Autentificación para entrar al administrador de archivos del servidor.

Una vez en ese punto, nos desplazamos hasta la carpeta “*public/html*” y pulsamos sobre el botón “*New File*”:



Figura 4.55. Botón de creación de nuevos archivos en el servidor.

El código es muy sencillo:

```
<?php
$var = "";
if($_GET) {
$var = $_GET['tipo'] . ".png";
```

```
 } else if($_POST){  
 $var = $_POST['tipo'] . ".png";  
}  
  
$link = mysql_connect('mysql12.000webhost.com',  
 'a7171134_seas', 'seas01');  
  
if (!$link) {  
 die('No pudo conectar: ' . mysql_error());  
}  
  
  
mysql_select_db("a7171134_seas");  
$q = "";  
mysql_query("SET NAMES utf8");  
  
if($var != "Todos.png") {  
 $q=mysql_query("SELECT * FROM seas01 where imagen like  
 '%".$var ."%');  
}  
else {  
 $q=mysql_query("SELECT * FROM seas01");  
}  
  
while($e=mysql_fetch_assoc($q)) {  
 $output[]=$e;  
}  
  
print(json_encode($output));  
mysql_close($link);  
?>
```

Con este código, el servidor espera una petición GET o POST a la que se le va a pasar el parámetro tipo posteriormente usado como cláusula del ‘where’.

Posteriormente, hemos definido la cadena de conexión mediante el `mysql_connect`, donde se define el nombre del servidor (esto nos lo proporciona el servidor para que podamos realizar dicha conexión), el nombre de usuario de la base de datos y la contraseña de dicho usuario.

Después, definimos la base de datos a la cual realizamos la consulta con `mysql_select_db(nombre_BBDD)`.

Posteriormente, definimos la consulta a la base de datos, controlando que se le haya pasado algún valor a la variable ‘*tipo*’, para que cuando no se haga, se devuelvan todos los resultados de la base de datos.

Muy importante es que se coloque la línea `mysql_query("SET NAMES utf8")` que será la encargada de devolver los datos en UTF-8, es decir, que se muestren los caracteres como acentos, “Ñ” y específicos de dicha codificación.

Para probar si hemos hecho correctamente la configuración del archivo de conexión, podemos realizar las pruebas sobre el mismo navegador web. Para ello podemos escribir la siguiente cadena en la barra de direcciones (accesible desde la plataforma de estudio): <http://seas-imagemaps.webatu.com/login.php?tipo=Seas>, donde la respuesta del servidor es la siguiente:

```
[{"ID":"1","NOMBRE":"Seas, Estudios Abiertos","DESCRIPCION":"El sistema de Formacion Abierta de SEAS combina la metodologia a distancia con las nuevas tecnologias de comunicacion.","LONGITUD":"41.67130099910645","LATITUD": "-0.878981351852417","IMAGEN":"Seas.png"}, {"ID":"4","NOMBRE":"Seas","DESCRIPCION":"Seas Estudios Superiores Abiertos. Delegacion Centro Zaragoza","LONGITUD":"41.64785539330573","LATITUD": "-0.8823448419570923","IMAGEN":"Seas.png"}]
```

Con esto confirmaremos que el código insertado es correcto y el servidor responde como deseábamos. Esta configuración es únicamente válida para el servidor de seas. Si cada alumno se ha ido configurando su propio servidor, sus bases de datos y sus tablas, deberá cambiar todos esos parámetros en cada sitio para que funcione su propia configuración.

RESUMEN

- Hemos aprendido qué es un Servicio Web, cómo funciona y cómo obtener los recursos que nos proporcionan los que existen actualmente en internet.
- Hemos visto las partes que compone un Servicio Web, tanto el proveedor de servicios, el registro UDDI y el cliente de los servicios, y cómo interactúan entre ellos.
- Ahora ya conocemos los diferentes estándares existentes de los Servicios Web, como son XML, SOAP, WSDL y el UDDI.
- Hemos visto cómo funciona REST para recuperar los datos necesarios de una página web.
- Hemos aprendido a implementar el protocolo de comunicación SOAP y a recuperar sus recursos.
- Hemos realizado una aplicación que nos geoposiciona, indicándonos nuestra ubicación, y nos muestra los recursos desde un servidor, consultándolos de una base de datos *MySQL*, previamente configurada. También hemos configurado una conexión en *php*, que es la que realiza la conexión al servidor y nos devuelve los datos que tenemos en la base de datos.