Desarrollo de Aplicaciones para Android

Índice

1 Introducción a Android........ 5
1.1 Android........ 5
1.2 Desarrollo de aplicaciones........ 10
1.3 Emulador........ 14
1.4 AndroidManifest.xml........ 15
1.5 Externalizar recursos........ 17
1.6 Plug-in para Eclipse........ 18
1.7 Hola, mundo........ 26
2 Introducción a Android - Ejercicios........ 33
2.1 Instalación........ 33
2.2 Android Virtual Device........ 33
2.3 Emulador........ 33
2.4 Depuración del Hola Mundo........ 33
2.5 Mejorar el Hola Mundo Mundo........ 33
3 Interfaz de usuario........ 35
3.1 Views........ 35
3.2 Layouts........ 40
3.3 Eventos........ 40
3.4 Activities e Intents........ 41
3.5 Menús y preferencias........ 44
3.6 Visor de Google Maps........ 45
4 Intefaz de usuario - Ejercicios........ 47
4.1 LinearLayout........ 47
4.2 Ciudades........ 47
4.3 Calculadora sencilla........ 48

Copyright © 2014 Depto. CCIA All rights reserved.

4.4 Abrir actividades desde un menú (*)........ 49
4.5 Fragments (*)........ 50
5 Gráficos avanzados........ 51
5.1 Elementos drawables........ 51
5.2 Componentes propios........ 57
5.3 Gráficos 3D........ 66
6 Gráficos avanzados - Ejercicios........ 73
6.1 Personalización del aspecto........ 73
6.2 Personalización de botones........ 73
6.3 Animaciones por fotogramas........ 73
6.4 Niveles........ 74
6.5 Creación de componentes propios........ 74
6.6 Texto y métricas........ 75
6.7 Gráficos 3D........ 75
7 Sensores y eventos........ 77
7.1 Pantalla táctil........ 78
7.2 Orientación y aceleración........ 83
7.3 Geolocalización........ 87
7.4 Reconocimiento del habla........ 91
8 Sensores y eventos - Ejercicios........ 93
8.1 Pantalla táctil........ 93
8.2 Gestos........ 93
8.3 Acelerómetro........ 94
8.4 Geolocalización........ 94
8.5 Reconocimiento del habla........ 95
9 Multimedia........ 96
9.1 Reproducción de audio........ 96
9.2 Reproducción de vídeo usando el control VideoView........ 98
9.3 Reproducción de vídeo basada en MediaPlayer........ 100
9.4 Toma de fotografías........ 102
9.5 Agregar ficheros multimedia en el Media Store........ 103
9.6 Sintetizador de voz de Android........ 104

Copyright © 2014 Depto. CCIA All rights reserved.

10 Multimedia - Ejercicios.......... 108
10.1 Reproducción de un clip de audio.......... 108
10.2 Reproducción de un clip de vídeo por medio del control Video View.......... 109
10.3 Síntesis de voz con Text to Speech.......... 109
11 Ficheros y acceso a datos.......... 111
11.1 Ficheros tradicionales.......... 111
11.2 Preferencias.......... 111
11.3 Base de datos SQLite.......... 114
11.4 Proveedores de contenidos.......... 117
12 Ficheros y acceso a datos - Ejercicios.......... 124
12.1 Escribir en un archivo de texto.......... 124
12.2 Crear y utilizar un DataHelper para SQLite.......... 124
12.3 Proveedor de contenidos propio.......... 126
12.4 ¿Por qué conviene crear proveedores de contenidos?.......... 127
12.5 Proveedores nativos.......... 127
13 Servicios de red.......... 129
13.1 Conexiones HTTP.......... 129
13.2 Parsing de XML.......... 129
13.3 Cargar imágenes de red.......... 131
13.4 Estado de la red.......... 131
13.5 Operaciones lentas.......... 132
14 Servicios de red - Ejercicios.......... 135
14.1 Cargar y trocear XML.......... 135
14.2 Operaciones largas de caraga de datos.......... 136
14.3 Lector de RSS - versión gráfica.......... 137
15 Servicios Avanzados.......... 139
15.1 Servicios en segundo plano.......... 139
15.2 Notificaciones.......... 140
15.3 AppWidgets.......... 142
15.4 Publicación de software.......... 145
16 Servicios avanzados - Ejercicios.......... 147
16.1 Servicio reproductor de música.......... 147

Copyright © 2014 Depto. CCIA All rights reserved.

16.2 Servicio con proceso en background. Contador..148
16.3 Servicio con notificaciones. Números primos...148
16.4 IP AppWidget..150

Copyright © 2014 Depto. CCIA All rights reserved.

1. Introducción a Android

1.1. Android

Android es un sitema operativo de código abierto para dispositivos móviles, se programa principalmente en Java, y su núcleo está basado en Linux.

1.1.1. Historia

Antiguamente los dispositivos empotrados sólo se podían programar a bajo nivel y los programadores necesitaban entender completamente el hardware para el que estaban programando.

En la actualidad los sistemas operativos abstraen al programador del hardware. Un ejemplo clásico es Symbian. Pero este tipo de plataformas todavía requieren que el programador escriba código C/C++ complicado, haciendo uso de bibliotecas (libraries) propietarias. Especiales complicaciones pueden surgir cuando se trabaja con hardware específico, como GPS, trackballs o touchscreens, etc.

Java ME abstrae completamente al programador del hardware, pero su limitación de máquina virtual le recorta mucho la libertad para acceder al hardware del dispositivo.

Esta situación motivó la aparición de Android, cuya primera version oficial (la 1.1) se publició en febrero de 2009. Esto coincidió con la proliferación de smartphones con pantallas táctiles.

Desde entonces han ido apareciendo versiones nuevas del sistema operativo, desde la 1.5 llamada Cupcake que se basaba en el núcleo de Linux 2.6.27 hasta la versión 4.0.x que está orientada tablets y a teléfonos móviles. Cada versión del sistema operativo tiene un nombre inspirado en la repostería, que cumple un orden alfabético con respecto al resto de versiones de Android (Cupcake, Donut, Eclair, Froyo, Gingerbread, Honeycomb, Ice Cream Sandwich, etc).

1.1.2. Open source

Android - tanto el sitema operativo, como la plataforma de desarrollo - están liberados bajo la licencia de Apache. Esta licencia permite a los fabricantes añadir sus propias extensiones propietarias, sin tener que ponerlas en manos de la comunidad de software libre.

Al ser de open source, Android hace posible:

- Una comunidad de desarrollo, gracias a sus completas APIs y documentación ofrecida.

Copyright © 2014 Depto. CCIA All rights reserved.

- Desarrollo desde cualquier plataforma (Linux, Mac, Windows, etc).
- Un sistema operativo para cualquier tipo de dispositivo móvil, al no estar diseñado para un sólo tipo de móvil.
- Posibilidad para cualquier fabricante de diseñar un dispositivo que trabaje con Android, y la posibilidad de abrir el sistema operativo y adaptarlo o extenderlo para su dispositivo.
- Valor añadido para los fabricantes de dispositivos: las empresas se ahorran el coste de desarrollar un sistema operativo completo para sus dispositivos.
- Valor añadido para los desarrolladores: los desarrolladores se ahorran tener que programar APIs, entornos gráficos, aprender acceso a dispositivos hardware particulares, etc.

¿De qué está hecho Android?

- Núcleo basado en el de Linux para el manejo de memoria, procesos y hardware. (Se trata de un branch, de manera que las mejoras introducidas no se incorporan en el desarrollo del núcleo de GNU/Linux).
- Bibliotecas open source para el desarrollo de aplicaciones, incluyendo SQLite, WebKit, OpenGL y manejador de medios.
- Entorno de ejecución para las aplicaciones Android. La máquina virtual Dalvik y las bibliotecas específicas dan a las aplicaciones funcionalidades específicas de Android.
- Un framework de desarrollo que pone a disposición de las aplicaciones los servicios del sistema como el manejador de ventanas, de localización, proveedores de contenidos, sensores y telefonía.
- SDK (kit de desarrollo de software) que incluye herramientas, plug-in para Eclipse, emulador, ejemplos y documentación.
- Interfaz de usuario útil para pantallas táctiles y otros tipos de dispositivos de entrada, como por ejemplo, teclado y trackball.
- Aplicaciones preinstaladas que hacen que el sistema operativo sea útil para el usuario desde el primer momento. Cabe destacar que cuenta con las últimas versions de Flash Player.
- Muy importante es la exitencia del Android Market, y más todavía la presencia de una comunidad de desarrolladores que suben ahí aplicaciones, tanto de pago como gratuitas. De cara al usuario, el verdadero valor del sistema operativo está en las aplicaciones que se puede instalar.

¿Quién desarrola Android?

La Open Handset Alliance. Consorcio de varias compañías que tratan de definir y establecer una serie de estándares abiertos para dispositivos móviles. El consorcio cuenta con decenas de miembros que se pueden clasificar en varios tipos de empresas:

- Operadores de telefonía móvil
- Fabricantes de dispositivos
- Fabricantes de procesadores y microelectrónica
- Compañías de software

Copyright © 2014 Depto. CCIA All rights reserved.

- Compañías de comercialización

Android no es "de Google" como se suele decir, aunque Google es una de las empresas con mayor participación en el proyecto.

1.1.2.1. Cuestiones éticas

Uno de los aspectos más positivos de Android es su carácter de código abierto. Gracias a él, tanto fabricantes como usuarios se ven beneficiados y el progreso en la programación de dispositivos móviles y en su fabricación, se abarata y se acelera. Todos salen ganando.

Otra consecuencia de que sea de código abierto es la mantenibilidad. Los fabricantes que venden dispositivos con Android tienen el compromiso de que sus aparatos funcionen. Si apareciera algún problema debido al sistema operativo (no nos referimos a que el usuario lo estropee, por supuesto) el fabricante, en última instancia, podría abrir el código fuente, descubrir el problema y solucionarlo. Esto es una garantía de éxito muy importante.

Por otro la seguridad informática también se ve beneficiada por el código abierto, como ha demostrado la experiencia con otros sistemas operativos abiertos frente a los propietarios.

Hoy en día los dispositivos móviles cuentan con hardware que recoge información de nuestro entorno: cámara, GPS, brújula y acelerómetros. Además cuentan con constante conexión a Internet, a través de la cuál diariamente circulan nuestros datos más personales. El carácter abierto del sistema operativo nos ofrece una transparencia con respecto al uso que se hace de esa información. Por ejemplo, si hubiera la más mínima sospecha de que el sistema operativo captura fotos sin preguntarnos y las envía, a los pocos días ya sería noticia.

Esto no concierne las aplicaciones que nos instalamos. Éstas requieren una serie de permisos antes de su instalación. Si los aceptamos, nos hacemos responsables de lo que la aplicación haga. Esto no es un problema de seguridad, los problemas de seguridad ocurren si se hace algo sin el consentimiento ni conocimiento del usuario.

No todo son aspectos éticos positivos, también abundan los preocupantes.

Los teléfonos con Android conectan nuestro teléfono con nuesto ID de google. Hay una transmisión periódica de datos entre Google y nuestro terminal: correo electrónico, calendario, el tiempo, actualizaciones del Android Market, etc. En este sentido, el usuario depende de Google (ya no dependemos sólo de nuestro proveedor de telefonía móvil). Además, la inmensa mayoría de los servicios que Google nos ofrece no son de código abierto. Son gratuitas, pero el usuario desconoce la suerte de sus datos personales dentro de dichas aplicaciones.

El usuario puede deshabilitar la localización geográfica en su dispositivo, y también puede indicar que no desea que ésta se envíe a Google. Aún así, seguimos conectados con Google por http, dándoles información de nuestra IP en cada momento. De manera

Copyright © 2014 Depto. CCIA All rights reserved.

indirecta, a través del uso de la red, ofrecemos información de nuestra actividad diaria. Si bien el usuario acepta estas condiciones de uso, la mayoría de los usuarios ni se paran a pensar en ello porque desconocen el funcionamiento del sistema operativo, de los servicios de Google, y de Internet en general.

Lógicamente, nos fiamos de que Google no hará nada malo con nuestros datos, ya que no ha habido ningún precedente.

1.1.3. Dispositivos

Los dispositivos con Android son numerosos. Aunque los hay de tipos muy diferentes, los principales son los tablets y los teléfonos móviles. La configuración típica es la de pantalla táctil y algunos botones para el menú, salir, home, apagar, volumen. Muchos dispositivos también cuentan con un teclado físico, y algunos incluso cuentan con varias pantallas.

Teléfono de desarrollador de Android

Motorla Charm

Copyright © 2014 Depto. CCIA All rights reserved.

Tablet Dawa

HTC Evo 4G

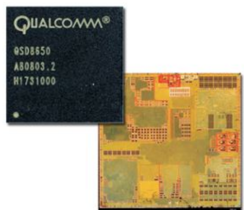
Sony Ericsson Xperia Mini

La mayoría de los dispositivos diseñados para Android utilizan procesadores con arquitectura ARM. ARM significa Advanced RISC Machine y, como su nombre indica, se trata de arquitecturas RISC (Reduced Instruction Set Computer) y son procesadores de

Copyright © 2014 Depto. CCIA All rights reserved.

32 bits. Aproximadamente el 98% de los teléfonos móviles usan al menos un procesador basado en arquitectura ARM.

Qualcomm es el fabricante del procesador Snapdragon, procesador basado en ARM que se está utilizando en los últimos dispositivos móviles del mercado. Realmente es una plataforma que en algunos de sus modelos incluye dos CPUs de 1.5 GHz, HSPA+, GPS, Bluetooth, grabación y reproducción de video full definition, Wi-Fi y tecnologías de televisión móvil. Lo están utilizando no sólo los fabricantes de teléfonos, sino también los de tablets y netbooks.



Procesador Snapdragon

1.2. Desarrollo de aplicaciones

Aunque el desarrollo de librerías de bajo nivel es posible con el Android Native Development Toolkit, vamos a centrarnos en el desarrollo de aplicaciones con el Android Software Development Toolkit.

1.2.1. Android SDK

El SKD de android incluye numerosas y completas API's para facilitar el desarrollo. Algunas de las características más relevantes son:

- Licencias, distribución y desarrollo gratuitos, tampoco hay procesos de aprobación del software.
- Acceso al hardware de WiFi, GPS, Bluetooth y telefonía, permitiendo realizar y recibir llamadas y SMS.
- Control completo de multimedia, incluyendo la cámara y el micrófono.
- APIs para los sensores: acelerómetros y brújula.
- Mensajes entre procesos (IPC).
- Almacenes de datos compartidos, SQLite, acceso a SD Card.
- Aplicaciones y procesos en segundo plano.
- Widgets para la pantalla de inicio (escritorio).
- Integración de los resultados de búsqueda de la aplicación con los del sistema.
- Uso de mapas y sus controles desde las aplicaciones.
- Aceleración gráfica por hardware, incluyendo OpenGL ES 2.0 para los 3D.

Muchas de estas características ya están, de una manera o de otra, para los SDK de otras

Copyright © 2014 Depto. CCIA All rights reserved.

plataformas de desarrollo móvil. Las que diferencian a Android del resto son:

- Controles de Google Maps en nuestras aplicaciones
- Procesos y servicios en segundo plano
- Proveedores de contenidos compartidos y comunicación entre procesos
- No diferencia entre aplicaciones nativas y de terceros, todas se crean igual, con el mismo aspecto, y con las mismas posibilidades de usar el harware y las APIs.
- Widgets de escritorio

1.2.2. Capas

El núcleo de Linux es la capa encargada de los controladores (drivers) del hardware, los procesos, la memoria, seguridad, red, y gestión de energía. Es la capa que abstrae el resto de las capas del hardware.

El Android run time es lo que hace a Android diferente de una distribución de Linux embebido. Está compuesto por las librerías "core" (núcleo) y por Dalvik, la máquina virtual de Java, basada en registros que cuenta con el núcleo de Linux para la gestión de hilos y para el manejo de memoria a bajo nivel.

El framework de aplicaciones está compuesto por las clases que se utilizan para crear aplicaciones Android: actividades, servicios, views, proveedores de contenidos, etc.

Finalmente la capa de aplicaciones está compuesta por las aplicaciones nativas y por las de terceros, así como las de desarrollo.

Copyright © 2014 Depto. CCIA All rights reserved.

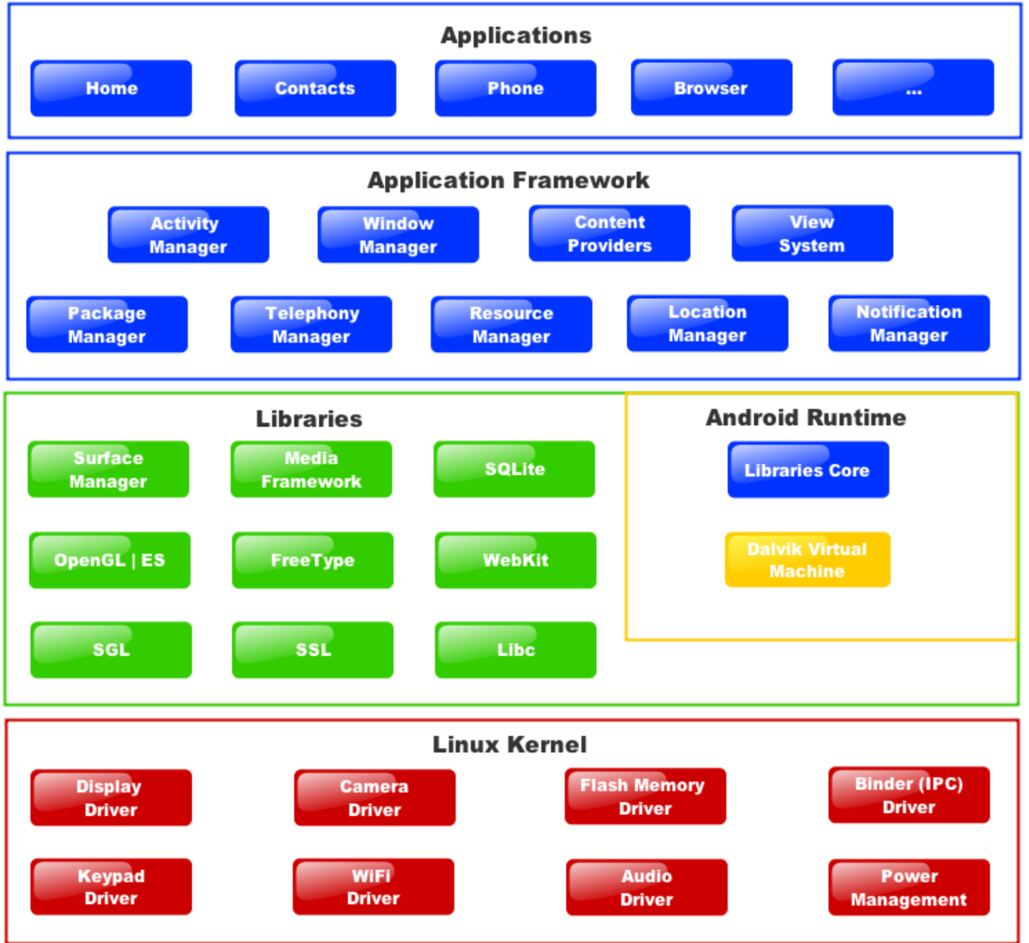


Diagrama de Android

Las clases más importantes para el desarrollo de aplicaciones en Android son las siguientes:

- `ActivityManager`: Controla el ciclo de vida de las actividades.
- `View`: Se usan para construir interfaces en las actividades.
- `NotificationManager`: Mecanismo no intrusivo para mostrar avisos al usuario.
- `ContentProvider`: Permiten intercambiar datos de una manera estandarizada.
- Resource Manager: permite usar en la aplicación recursos que no forman parte del código, como XML, strings, recursos gráficos, audio, vídeo, etc.

1.2.3. Tipos de aplicaciones

Hay tres tipos de aplicaciones para Android: las de primer plano (foreground), las de segundo plano (background) y los widget (o AppWidget).

Copyright © 2014 Depto. CCIA All rights reserved.

Las aplicaciones de primer plano constan de actividades que muestran una interfaz de usuario. No tienen mucho control sobre su ciclo de vida, ya que en cualquier momento se les puede robar el foco, o pueden pasar a segundo plano. Si faltan recursos, tanto de memoria, como de procesador, pueden ser incluso terminadas. El objetivo del sistema operativo es que la aplicación que en este momento esté en primer plano ofrezca respuestas fluidas. Si para ello hay que terminar otras aplicaciones en segundo plano, Android lo hará.

Las aplicaciones de segundo plano se denominan servicios. Las hay de dos tipos: servicios puros, o servicios combinados con actividades que permiten configurarlos. El ejemplo típico es el de un reproductor multimedia. En cuanto hemos seleccionado qué cancion reproducir, cerramos la actividad y un servicio mantiene el audio en reproducción.

Otro tipo de aplicaciones son los widget. Se trata de apliaciones de pequeña interfaz gráfica que se colocan sobre el escritorio (home) de Android y que se refrescan cada cierto intervalo de tiempo. Normalmente detrás de ellas corre un servicio de actualización. Un ejemplo es el del reloj, o el calendario.

1.2.4. Consideraciones para el desarrollo

El desarrollo para dispositivos móviles y embebidos requiere que el programador tenga especial cuidado en determinados aspectos. El sistema operativo no puede encargarse de controlar todo porque limitaría demasiado al programador, así que determinados aspectos como ahorrar CPU y memoria son responsabilidad del programador en la mayoría de los casos. La limitaciones de hardware que el dispositivo impone suelen ser:

- Pequeña capacidad de procesamiento
- Memoria RAM limitada
- Memoria permanente de poca capacidad
- Pantallas pequeñas de poca resolución
- Transferencias de datos costosa (en términos de energía y económicos) y lenta
- Inestabilidad de las conexiones de datos
- Batería muy limitada
- Necesidad de terminar la aplicación en cualquier momento

A partir de estas limitaciones, las consideraciones que el desarrollador debe tener son: ser eficiente en cuanto a bucles y cálculos; asumir poca memoria y optimizar el almacenamiento al máximo, eliminando siempre los datos que ya no van a ser necesarios; diseñar para pantallas de distintos tamaños pero sobretodo tener en cuenta las más pequeñas; ahorrar comunicaciones y asumir que serán lentas y que fallarán, lo cuál no debe limitar la capacidad de respuesta de la aplicación.

También es muy importante respetar al usuario. No hay que robar el foco a otras aplicaciones, hay que usar avisos lo mínimo posible y de la manera menos intrusiva, hay que mantener un estilo de interfaz consistente con el sistema y que responda bien.

Copyright © 2014 Depto. CCIA All rights reserved.

También hay que intentar que la aplicación se recupere rápidamente al rearrancarla y que se mantenga en el último estado en el que estuvo, para que el usuario no note la diferencia cuando la aplicación ha sido terminada o cuando ha continuado su ejecución tras estar en segundo plano. También, no hace falta ni decir, que el usuario no quiere gastar más batería de lo necesario y eso conciene el procesamiento y sobretodo la comunicación por red.

Si desarrollamos nuestras aplicaciones con responsabilidad y respetamos los consejos de desarrollo de Android (Guía de desarrollo de Android http://developer.android.com/guide/index.html, sección Best Practices) contribuiremos a que el sistema operativo Android ofrezca mejores experiencias a sus usuarios. Al fin y al cabo los usuarios juzgan a un sistema operativo por sus aplicaciones, aunque éstas sean de terceros. A los desarrolladores nos interesa que la plataforma sobre la que programamos sea utilizada por el máximo número de usuarios posible para que nuestras aplicaciones lleguen a más gente.

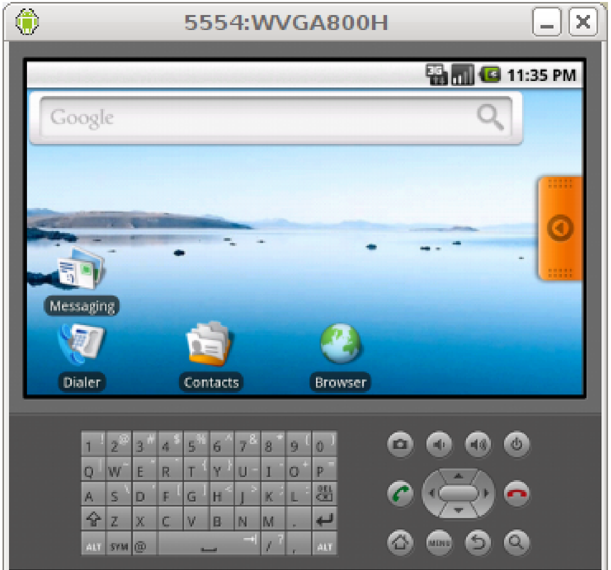
1.3. Emulador

Android SDK viene con un emulador en el que podemos probar la mayoría de nuestras aplicaciones. Desde Eclipse podemos ejecutar nuestras aplicaciones directamente en un emulador arrancado, que corre sobre un puerto. También podemos tener varios emuladores arrancados para que se comuniquen entre ellos si la aplicación lo requiere. Dentro del emulador contamos con una distribución de Android instalada con sus aplicaciones nativas y la mayoría de las funcionalidades.

Podemos simular GPS, llamadas entrantes, salientes, SMS, entradas por la pantalla y el teclado, reproducción de audio y vídeo, y comunicaciones por red. Todos los tipos de aplicaciones están soportadas: widgets, servicios y actividades. Se pueden instalar y desinstalar como si de un móvil real se tratara.

Todavía no se pueden simular conexiones por USB, captura por la cámara, auriculares o manos libres conectados al móvil, conexión o desconexión de la corriente eléctrica ni el estado de la batería, bluetooth, tampoco la inserción y extracción de la tarjeta de memoria SD, sensores, ni tampoco el multitouch.

Copyright © 2014 Depto. CCIA All rights reserved.



Emulador de Android

El emulador incluye una consola a la que se puede conectar por telnet:

```
telnet localhost <puerto>
```

El puerto suele ser el 5554, como en el ejemplo de la imagen. Se indica en la barra de la ventana del emulador. Este puerto también se utiliza como número de teléfono para simular llamadas y SMS.

El emulador de Android necesita información de configuración del dispositivo virtual, denominado AVD, Android Virtual Device. Debemos crear un dispositivo AVD para poder arrancar el emulador con determinada configuración. En esta configuración debemos especificar el nivel de API que tiene el dispositivo (por ejemplo, para el caso de Android 2.2 el nivel de API es el 8), el tipo de pantalla y el resto del hardware: tarjeta SD, presencia de teclado físico, etc. Esta configuración se puede crear con el AVD Manager del plugin de Eclipse.

Toda la información sobre el emulador de Android se puede encontrar en el sitio oficial http://developer.android.com/guide/developing/tools/emulator.html. Ahí están enumerados todos los comandos de control y de configuración del emulador.

1.4. AndroidManifest.xml

Todos los proyectos de Android contienen un fichero `AndroidManifest.xml`. Su finalidad es declarar una serie de metadatos de la aplicación que el dispositivo debe conocer antes de instalarla. En él se indican: el nombre del paquete, el nombre de la aplicación, las actividades, servicios, receptores broadcast, proveedores de contenidos,

Copyright © 2014 Depto. CCIA All rights reserved.

cuál es la actividad principal. En él también se indica el nivel mínimo de API que la aplicación requiere, y los permisos que necesita para trabajar. Los permisos serán concedidos de manera explícita por el usuario si da su consentimiento para instalar la aplicación. También se puede indicar aspectos más avanzados, como los permisos que otras aplicaciones necesitarían para poder interactuar con la nuestra. Las librerías utilizadas también se declaran en este archivo.

La estructura del `AndroidManifest.xml` es la siguiente:

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

<manifest>

    <uses-permission />
    <permission />
    <permission-tree />
    <permission-group />
    <instrumentation />
    <uses-sdk />
    <uses-configuration />
    <uses-feature />
    <supports-screens />

    <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>
```

Nosotros abriremos este archivo básicamente cuando creemos una nueva actividad,

Copyright © 2014 Depto. CCIA All rights reserved.

cuando veamos que necesitamos permisos (para acceder a Internet, para acceder a los contactos, etc), cuando declaremos un servicio o un widget, cuando creemos un proveedor de contenidos, o cuando utilicemos una librería, como por ejemplo es la de Google Maps. No nos hará falta tocar nada para la actividad principal, ya que ésta nos la crea el asistente de Eclipse y nos añade la información necesaria en el manifest.

1.5. Externalizar recursos

La externalización de recursos hace la aplicación más mantenible y fácilmente personalizable y adaptable a otros idiomas. Android ofrece facilidades para la externalización de recursos. Todos los recursos de una aplicación se almacenan en la carpeta `res` del proyecto, de manera jerárquica. El asistente de Eclipse nos genera una estructura de recursos que contiene las carpetas `values`, `drawable-ldpi`, `drawable-mdpi`, `drawable-hdpi` y `layout`. En esta última se guardan los layouts para la interfaz gráfica. En la primera, `values`, se guardan strings, colores y otros recursos que contienen pares de identificador y valor. Si añadimos algún menú o preferencias, los añade en una nueva carpeta, `xml`.

Todos los recursos se guardan en archivos con el siguiente formato:

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

Cada tipo de recurso se especifica con determinado tag de XML. Algunos ejemplos de recursos son los siguientes.

- Strings. Se guardan en `res/values/strings.xml` en formato

```
<string name="saludo">¡Hola!</string>
```

- Colores. Se guardan en `res/values/colors.xml` en formato hexadecimal #RGB, #RRGGBB, #ARGB ó #AARRGGBB, como en el ejemplo:

```
<color name="verde_transparente">#7700FF00</color>
```

- Dimensiones. Se guardan en `res/values/dimensions.xml` en formato `px` (píxeles de pantalla), `in` (pulgadas), `pt` (puntos físicos), `mm` (milímetros físicos), `dp` (píxeles relativos a pantalla de 160-dpi, independientes de la densidad), `sp` (píxeles independientes a la escala), como en el siguiente ejemplo:

```
<dimen name="altura_mifuente">12sp</dimen>
```

- Arrays

Copyright © 2014 Depto. CCIA All rights reserved.

```
<array name="ciudades">
        <item>Alicante</item>
        <item>Elche</item>
        <item>San Vicente</item>
</array>
```

- Estilos y temas. Permiten modificar el aspecto de los `View` pasándoles como parámetro el nombre del estilo especificado en los recursos. Por ejemplo, para especificar tamaño y color de la fuente para el estilo "EstiloTexto1":

```
<style name="EstiloTexto1">
        <item name="android:textSize">18sp</item>
        <item name="android:textColor">#00F</item>
</style>
```

Para usar recursos desde el código se puede acceder a sus identificadores con `R.id.TextView01` para los views; con `R.layout.main` para los layouts, donde `main` se correspondería con el layout `res/layout/main.xml`; con `R.string.saludo`, etc. Por ejemplo:

```
        TextView tv = (TextView)findViewById(R.id.TextView01);
        tv.setText(R.string.saludo);
```

Para usar los String desde los layouts (en XML) hay que referenciarlos como

```
        @string/nombrestring
```

De la misma forma que los layout quedan indexados entre los recursos y se pueden referenciar, también se pueden utilizar recursos de imagen, audio, vídeo. Hay que usar sólo minúsculas en los nombres.

1.6. Plug-in para Eclipse

1.6.1. Instalación

Una vez descargado y descomprimido el Android SDK, hay que instalar el plug-in para Eclipse que incluye una serie de interfaces, herramientas y asistentes, así como editores visuales de los archivos XML.

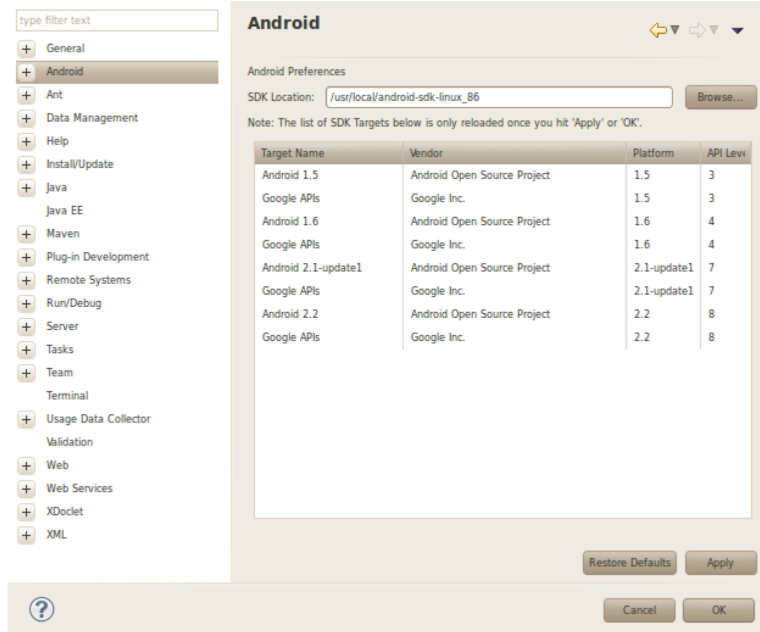
La información detallada sobre la instalación del Android ADT Plugin para Eclipse está en http://developer.android.com/sdk/eclipse-adt.html. Para instalarlo hay que ir a los menús de Eclipse: Help > Install New Software, y en Available Software pulsar Add, y añadir el sitio:

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


Copyright © 2014 Depto. CCIA All rights reserved.

Pulsar Ok, seleccionar el nuevo software a instalar, Next, y Finish. Hará falta reiniciar Eclipse.

Antes de empezar a usar el plugin hay que configurarlo. En Windos > Preferences, seleccionamos Android, y para la SDK Location pulsamos Browse para indicar ahí la ruta donde tenemos descomprimido el Android SDK.



Preferencias Android en Eclipse y SDK location.

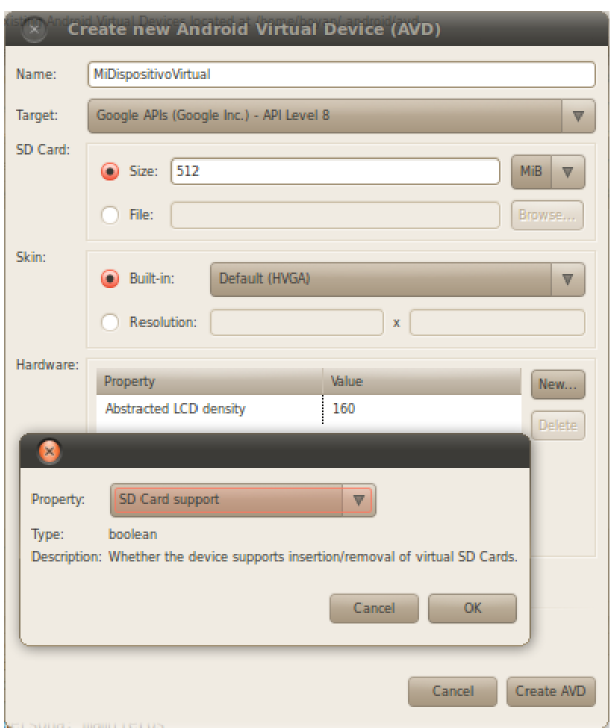
Ya está listo para usar. Sólo falta crear un dispositivo AVD para que el Emulador lo pueda arrancar cuando queramos ejecutar nuestras aplicaciones Android.

1.6.2. Herramientas

Vamos a enumerar algunas de las herramientas que el plug-in proporciona.

AVD Manager. Accesible desde el menú Window de Eclipse o bien desde la barra de herramientas de Eclipse. Permite crear nuevos dispositivos virtuales (AVDs) y especificar su hardware:

Copyright © 2014 Depto. CCIA All rights reserved.

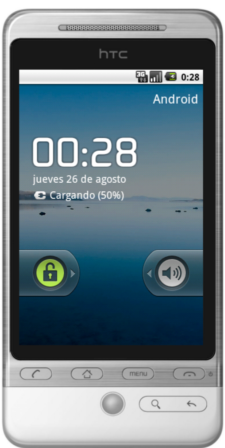


AVD Manager: asistente para crear un nuevo AVD; añadiendo nuevo hardware

Emulador. Emula los AVDs que hemos creado. Según el AVD y según el "skin" puede tener un aspecto diferentes. Por defecto viene un único skin:

Emulador con el skin por defecto y teclado físico

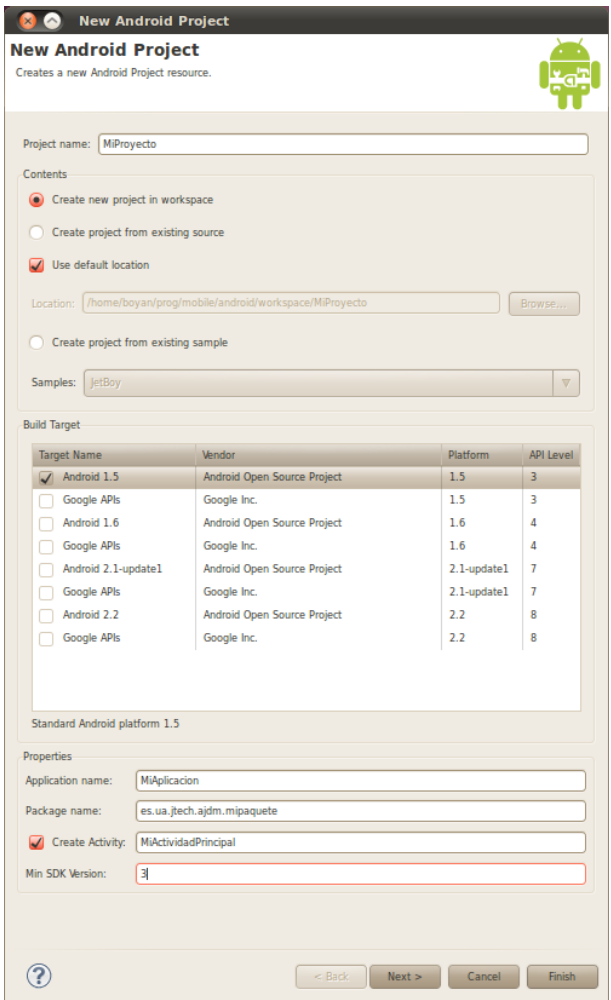
Copyright © 2014 Depto. CCIA All rights reserved.



Emulador con el skin de HTC Hero

Asistente para la creación de proyectos. Nos genera la estructura de directorios básica y el AndroidManifest.xml, con una actividad principal cuyo nombre indicamos:

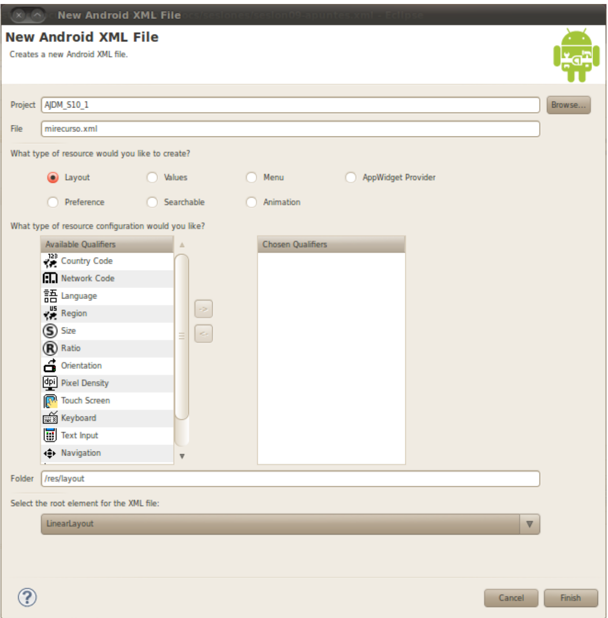
Copyright © 2014 Depto. CCIA All rights reserved.



Asistente para la creación de proyectos Android.

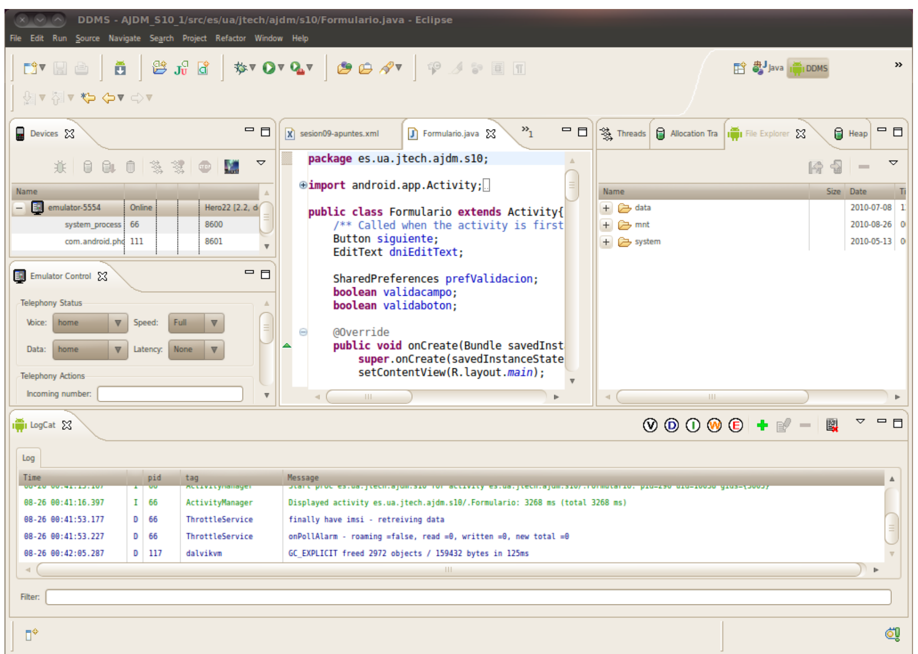
Asistente para la creación de recursos XML. Nos ayuda a crear layouts, valores, menús, AppWidgets, preferencias y otros:

Copyright © 2014 Depto. CCIA All rights reserved.



Asistente para la creación de recursos XML de Android.

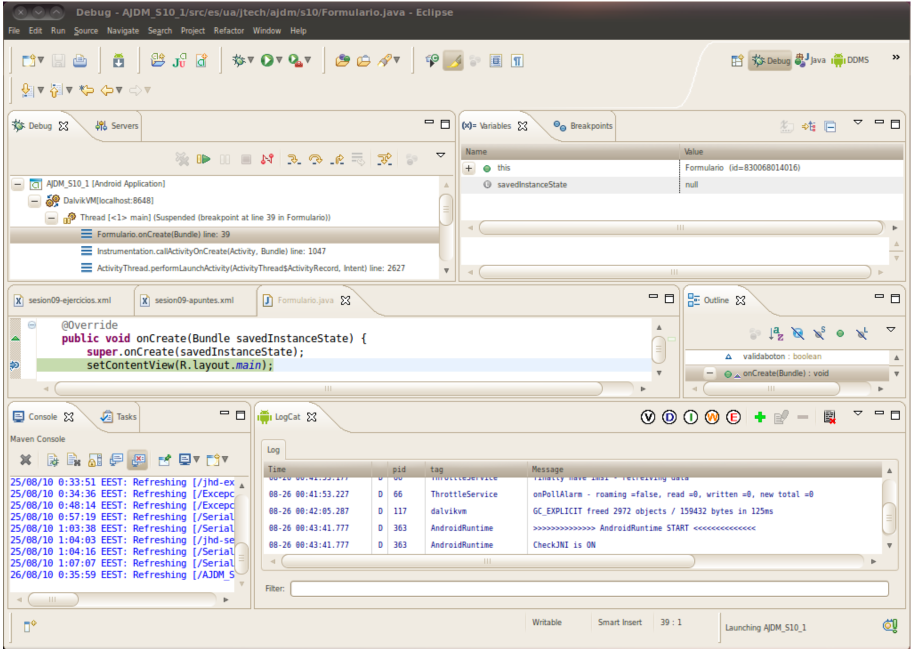
Vista DDMS. Nos permite monitorizar y controlar el emulador durante la ejecución de los programas. Podemos ver el LogCat, los procesos del dispositivo emulado, los hilos, la reserva de memoria, su sistema de ficheros, y mandar algunas configuraciones al emulador, así como simular llamadas o SMS. Se accede a través de los botones de vistas de Eclipse, en la esquina superior derecha:



Vista DDMS

Copyright © 2014 Depto. CCIA All rights reserved.

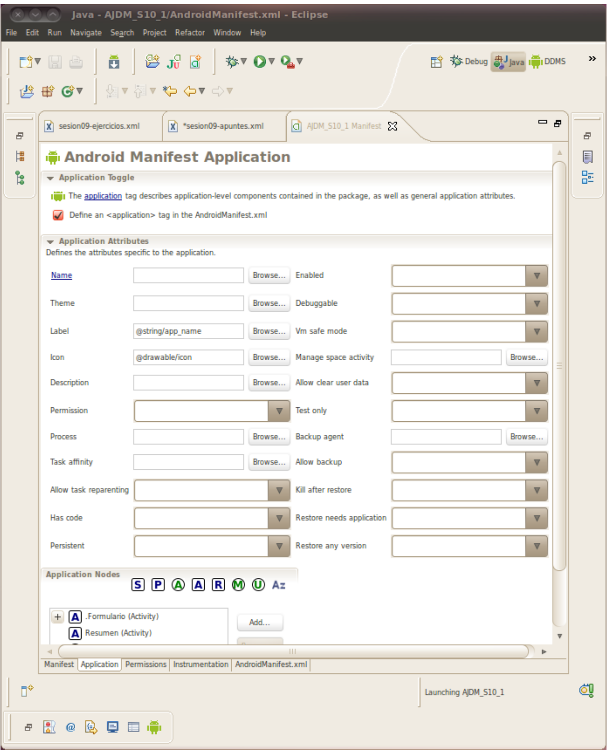
Debugging. Es otra vista de Eclipse y nos permite depurar programas emulados exactamente igual que en los programas Java tradicionales. Para que un programa sea publicable, el debugging se debe deshabilitar declarándolo en el AndroidManifest.xml, lo cuál no permitiría depurarlo desde Eclipse.



Vista Debug

Editor gráfico del AndroidManifest.xml. Cuando editamos este archivo, en la parte inferior izquierda tenemos una serie de pestañas. La última de ellas nos permite verlo en formato XML, el resto son interfaces gráficos para su edición:

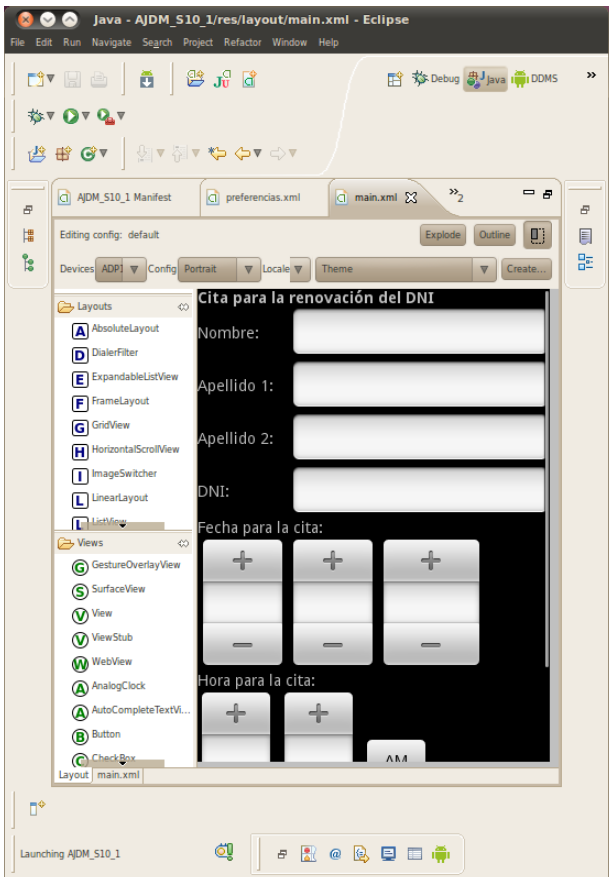
Copyright © 2014 Depto. CCIA All rights reserved.



Editor del AndroidManifest.xml

Editor visual de layouts. Nos permite añadir view y layouts y los previsualiza. Para pasar a modo XML, seleccionamos la pestaña correspondiente, en la parte inferior izquierda del editor:

Copyright © 2014 Depto. CCIA All rights reserved.



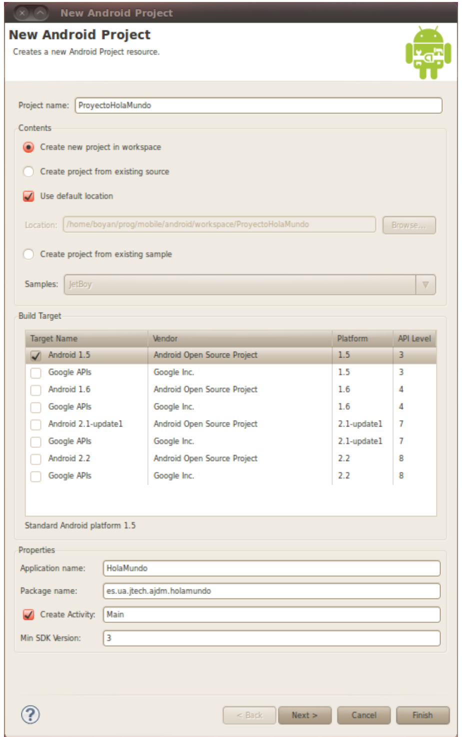
Editor gráfico de Layouts.

1.7. Hola, mundo

1.7.1. Proyecto nuevo y recursos

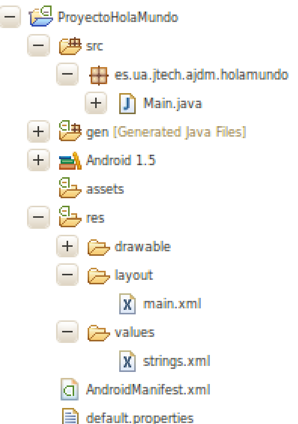
Creamos un New > Android project y lo titulamos `ProyectoHolaMundo`. A través del asistente damos un nombre a la aplicación, a la actividad principal y al paquete, como se ilustra en la imagen. También tenemos que seleccionar la version mínima de API necesaria, si vamos a usar Android 1.5, ésta puede ser la 3.

Copyright © 2014 Depto. CCIA All rights reserved.



Asistente de creación del proyecto.

Pulsamos finalizar y se genera la siguiente estructura de directorios:



Estructura de ficheros generada.

Abrimos el `AndroidManifest.xml` y observamos el código:

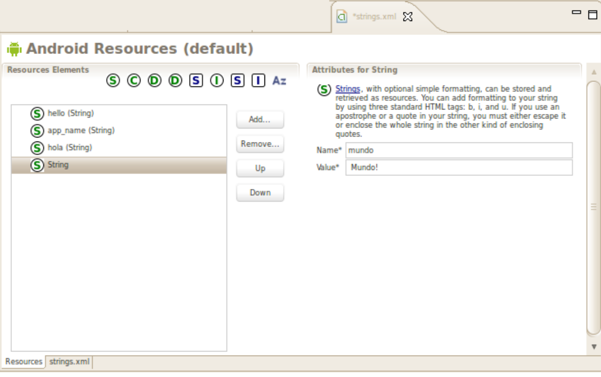
```
        <?xml version="1.0" encoding="utf-8"?>
        <manifest
xmlns:android="http://schemas.android.com/apk/res/android"
              package="es.ua.jtech.ajdm.holamundo"
              android:versionCode="1"
```


Copyright © 2014 Depto. CCIA All rights reserved.

```
              android:versionName="1.0">
            <application android:icon="@drawable/icon"
android:label="@string/app_name">
                <activity android:name=".Main"
                          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-sdk android:minSdkVersion="3" />

        </manifest>
```

El manifest nos indica, en primer lugar, el nombre del paquete y la versión. Después indica que tenemos una aplicación con un nombre (cuyo valor está en los recursos) y un icono. En esta aplicación tenemos una actividad principal Main. Finalmente indica que se requiere como mínimo la versión 3 de la API. Si un móvil tuviera menor versión, esta aplicación no se instalaría en él.

Vamos a abrir el archivo `res/values/strings.xml` y a comprobar que el nombre de la aplicación está ahí.



Editor de recursos de tipo string.

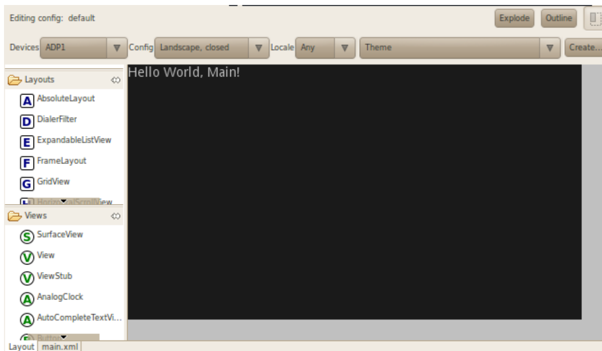
También hay un string "hello", que usa el layout principal, como veremos a continuación. De momento vamos a añadir los strings que vamos a utilizar en nuestra aplicación: `"Hola, "`, `" Mundo!"` y `"Hola ¿qué?"`, que se llamarán "`hola`", "`mundo`" y "`que`" , respectivamente. El XML del archivo debe quedar así:

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


Copyright © 2014 Depto. CCIA All rights reserved.

1.7.2. Layout

Ahora podemos editar el layout `main.xml`. En vista de diseño aparece así:



Layout main.xml original.

Eliminamos la etiqueta que ya viene y en su lugar dejamos caer una nueva etiqueta `TextView`y un botón `Button`. Si ahora miramos el layout en modo XML, veremos algo así:

```
<?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:text="@+id/TextView01" android:id="@+id/TextView01"
        android:layout_width="wrap_content"
android:layout_height="wrap_content" />

<Button android:text="@+id/Button01" android:id="@+id/Button01"
        android:layout_width="wrap_content"
android:layout_height="wrap_content" />

</LinearLayout>
```

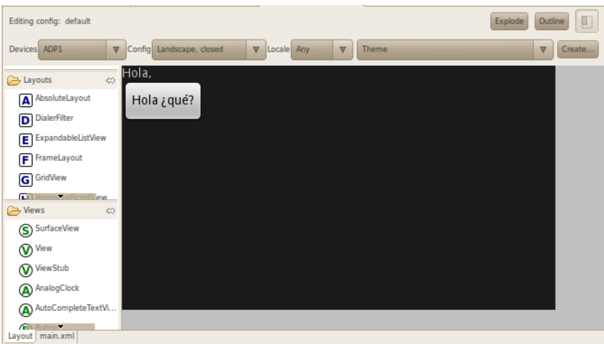
Vamos a cambiar los atributos `android:text` del `TextView` y del `Button`. En el primero pondremos el string "hola" que ya hemos definido en los strings, y en el segundo pondremos "que", también definido. El código quedará así:

```
<?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:text="@string/hola" android:id="@+id/TextView01"
        android:layout_width="wrap_content"
```


Copyright © 2014 Depto. CCIA All rights reserved.

```
android:layout_height="wrap_content" />

<Button android:text="@string/que" android:id="@+id/Button01"
        android:layout_width="wrap_content"
android:layout_height="wrap_content" />

</LinearLayout>
```

Y en vista de diseño se verá así:



Layout main.xml tras realizar los cambios.

Ahora lo podemos ejecutar y ver el resultado en el emulador. Hay que ejecutar, o bien el proyecto, o bien el `Main.java` como Android application. En el emulador tendremos un aspecto similar a la previsualización de la imagen anterior, y el botón será pulsable, sin realizar ninguna acción.

> **Nota:**
> Es posible que el emulador se inicie con la pantalla bloqueada. La desbloquearemos como si de un móvil real se tratara. Además el emulador es lento arrancando, a veces hay que tener paciencia.

1.7.3. Actividad y eventos

Vamos a añadir acción al botón. Para ello vamos a editar el `Main.java` que se encuentra en la carpeta `src`, dentro del paquete que hemos especificado. Se trata de la actividad que hemos llamado `Main` (podía haber tenido otro nombre cualquiera) y hereda de `Activity`, por eso es una actividad. Tiene sobrecargado el método `onCreate(Bundle) y en él, aparte de ejecutar el mismo método pero de la clase padre, lo que hace en la siguiente instrucción:`

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

es poner en pantalla el layout especificado en el recurso `res/layout/main.xml`. En este layout tenemos un botón y una etiqueta. Para poder acceder a ellos desde la actividad vamos a declararnos dos variables de sus respectivos tipos como campos de la actividad,

Copyright © 2014 Depto. CCIA All rights reserved.

```
TextView textView;
Button   button;
```

y vamos a asignarles valores en el método `onCreate`, tras la instrucción `setContentView`:

```
textView = (TextView)findViewById(R.id.TextView01);
button = (Button)findViewById(R.id.Button01);
```

Con esto ya podemos acceder desde el código a ambos View. Empezamos asignando un `onClickListener` al botón, para que escuche los eventos de click sobre este botón en particular:

```
button.setOnClickListener(new OnClickListener() {

        @Override
        public void onClick(View v) {

        }

    });
```

Nota:

En Eclipse, cuando tecleemos "`button.setOnClickListener(new` ", la autocompleción (que se activa con Ctrl+spacebar) nos sugerirá el `OnClickListener` y si pulsamos Enter, nos completará la clase con el método `onClick` sobrecargado, puesto que es obligatorio. Es posible que nos marque el `OnClickListener` como error por no encontrarlo, porque no ha añadido automáticamente el import necesario. Podemos añadir el import haciendo caso a la sugerencia para solucionar el error, o bien con "Fix imports", que se consigue con Mayús+Ctrl+o. Finalmente, sólo nos faltará poner el punto y coma ";" al final de todas las llaves y paréntesis.

Dentro del método `OnClickListener.onClick(View)` ejecutamos una acción, en este caso la acción será añadir el string "mundo" al `textView` (que hasta el momento sólo contiene "hola". Así, cuando se pulse el botón, aparte de "hola" aparecerá "mundo":

```
textView.append(getString(R.string.mundo));
```

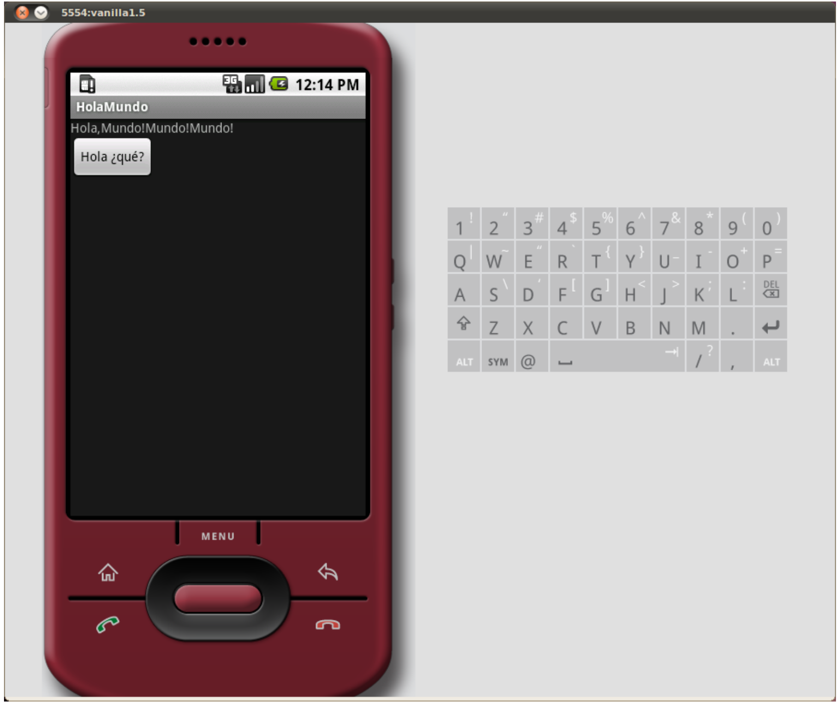
El código completo quedaría así:

```
package es.ua.jtech.ajdm.holamundo;

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

public class Main extends Activity {
```


Copyright © 2014 Depto. CCIA All rights reserved.

```
    /** Called when the activity is first created. */
        TextView textView;
        Button   button;

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

        textView = (TextView)findViewById(R.id.TextView01);
        button = (Button)findViewById(R.id.Button01);

        button.setOnClickListener(new OnClickListener() {

                        @Override
                        public void onClick(View v) {
                                textView.append(getString(R.string.mundo));
                        }

                });
    }
}
```

Ahora podemos volver a ejecutar y comprobar que cada vez que pulsamos el botón se añade el texto correspondiente. En la imagen se ve el resultado tras pulsar el botón tres veces:



Hola Mundo tras pulsar el botón tres veces.

Copyright © 2014 Depto. CCIA All rights reserved.

2. Introducción a Android - Ejercicios

2.1. Instalación

Comprueba que el plugin de Android para Eclipse tenga el path correcto. Si no está configurado, configúralo.

2.2. Android Virtual Device

Vamos a crear dos nuevos AVDs (Android Virtual Device): uno con Android 2.3.x y otro con Android 4.0.x. (Este último podría tardar mucho en arrancar e incluso colgarse). Poned tarjetas de memoria de 512MB, pantalla táctil, sin teclado físico, y con el resto de hardware que consideréis oportuno.

2.3. Emulador

Arrancad los dos AVDs que habéis creado y probad sus sistemas operativos. Probad programas, y probad añadir algún contacto nuevo. Realizad una llamada simulada desde uno al otro, y enviad un SMS entre ellos.

2.4. Depuración del Hola Mundo

Implementa el Hola Mundo visto en clase y pruébalo.

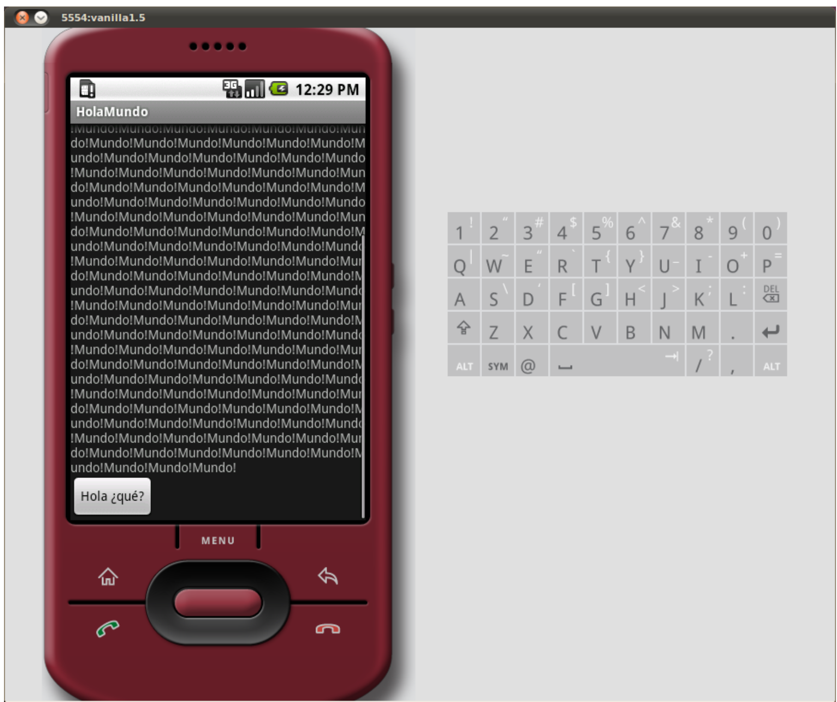
Pon un breakpoint en la primera instrucción del método `onCreate()` y depura. En la vista de Debug, avanza instrucción a instrucción hasta el final del método. Al final tendrás que pulsar a "Continuar", pero antes de ello, coloca otro breakpoint de tal manera que se pare el depurador cuando se pulse el botón.

Haz que aparezca en el LogCat todo el contenido del `TextView` cada vez que se pulse el botón.

2.5. Mejorar el Hola Mundo Mundo

Si pulsamos repetidas veces el botón, el campo de texto llega al final de la línea y pasa a la siguiente, desplazando hacia abajo el botón. Conforme lo desplazan, al final el botón desaparece de la pantalla. Esto sería un problema para los usuarios más exigentes. Lo podemos solucionar con un `ScrollView`. Edita en modo XML el layout y haz que un `ScrollView` envuelva el `LinearLayout`. Sugerencia: coloca al `ScrollView` los mismos atributos que tiene el `LinearLayout`, eso no debería dar problemas.

Copyright © 2014 Depto. CCIA All rights reserved.



Hola Mundo Mundo con Scrollbar

Copyright © 2014 Depto. CCIA All rights reserved.

3. Interfaz de usuario

En esta sesión vamos a introducir el diseño y programación de interfaces de usuario básicas para Android. La API de Android proporciona el acceso a una serie de componentes de alto nivel que nos ahorran tener que programarlos desde cero. Por otro lado su uso nos permitirá dar a nuestras aplicaciones el mismo aspecto que el resto de aplicaciones del sistema.

3.1. Views

Todos los componentes visuales de la interfaz de usuario, tales como botones, campos de texto, selectores, etc, se denominan `Views` en Android. Los views se pueden agrupar en `ViewGroups` que sirven para reutilizar componentes que siempre vayan a utilizarse juntos.

Los views se distribuyen sobre `Layouts`. Hay distintos tipos de layout, según la distribución de componentes que queramos tener en la pantalla. El layout que más se utiliza es el `LinearLayout` que puede disponer los componentes uno después del otro, o bien horizontalmente, o bien verticalmente. Para hacer combinaciones se pueden incluir layouts más pequeños dentro de otros.

Cualquier view o layout puede ocupar, o bien el tamaño completo que su contenedor le permita: `fill_parent`, o bien el tamaño mínimo que necesite para dar cabida a los componentes y contenidos que haya en él: `wrap_content`. Estos dos valores pueden ser aplicados tanto en horizontal como en vertical, mediante los atributos `layout_width` y `layout_height`.

Aunque cualquier interfaz gráfico se podría crear programáticamente, sin hacer uso de ningún recurso XML, lo normal es diseñar nuestros layouts en formato XML y con la ayuda del diseñador de interfaces disponible en el plugin de Android para Eclipse. Así, podemos introducir un componente `TextView` en un layout llamado `main.xml` y mostrarlo en nuestra actividad principal:

```
public class Interfaces extends Activity {
    /** Called when the activity is first created. */

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        ((TextView)findViewByID(R.id.TextView01)).setText("Hola Android");
        }
}
```

donde el XML de `main.xml` sería:

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


Copyright © 2014 Depto. CCIA All rights reserved.

```
<LinearLayout
android:orientation="vertical"
android:layout_width="fill_parent"
android:layout_height="wrap_content">
<TextView
        android:text="Hola Android"
        android:id="@+id/TextView01"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        />
</LinearLayout>
```

O bien podemos prescindir de recursos XML y añadir los views desde el código:

```
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        TextView miTextView = new TextView(this);
        setContentView(miTextView);
        miTextView.setText("Hola Android");
    }
```

Por supuesto, es preferible trabajar con los recursos XML cuando sea posible, ya que facilitan la mantenibilidad del programa, así como su diseño, que viene apoyado por la herramienta diseñadora del plugin para Eclipse.

Algunos views estándar que Android proporciona son los siguientes:

- `TextView`, etiqueta de texto.
- `EditText`, campo de texto.
- `Button`, botón pulsable con etiqueta de texto.
- `ListView`, grupo de views que los visualiza en forma de lista vertical.
- `Spinner`, lista desplegable, internamente es una composición de TextView y de List View.
- `CheckBox`, casilla marcable de dos estados.
- `RadioButton`, casilla seleccionable de dos estados, donde un grupo de RadioButtons sólo permitiría seleccionar uno de ellos al mismo tiempo.
- `ViewFlipper`, un grupo de Views que nos permite seleccionar qué view visualizar en este momento.
- `ScrollView`, permite usar barras de desplazamiento. Sólo puede contener un elemento, que puede ser un Layout (con otros muchos elementos dentro).
- `DatePicker`, permite escoger una fecha.
- `TimePicker`, permite escoger una hora.
- Otros más avanzados como `MapView` (vista de Google Maps) y `WebView` (vista de navegador web), etc.

Una buena práctica de programación es extender el comportamiento de los componentes por medio de herencia. Así crearemos nuestros propios componentes personalizados. Más información sobre Views se puede obtener en el tutorial oficial:

Copyright © 2014 Depto. CCIA All rights reserved.

http://developer.android.com/resources/tutorials/views/index.html.

3.1.1. Algunas clases útiles

En la API para interfaces gráficos hay otras clases útiles para la interacción con el usuario. Veamos algunas de ellas.

3.1.1.1. Toast

Los `Toast` sirven para mostrar al usuario algún tipo de información de la manera menos intrusiva posible, sin robar el foco a la actividad y sin pedir ningún tipo de interacción, desapareciendo automáticamente. El tiempo que permanecerá en pantalla puede ser, o bien `Toast.LENGTH_SHORT`, o bien `Toast.LENGTH_LONG`. Se crean y muestran así:

```
Toast.makeText(MiActividad.this,
                "Preferencia de validación actualizada",
                Toast.LENGTH_SHORT).show();
```

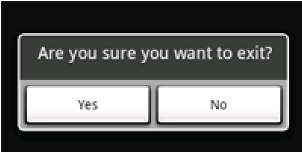
No hay que abusar de ellos pues, si se acumulan varios, irán apareciendo uno después de otro, esperando a que acabe el anterior y quizás a destiempo. Son útiles para confirmar algún tipo de información al usuario, que le dará seguridad de que está haciendo lo correcto. No son útiles para mostrar información importante, ni información extensa. Por último, si se van a utilizar como mensajes de Debug, aunque son útiles es mucho mejor utilizar la instrucción `LOG.d("TAG","Mensaje a mostrar")` y seguir el LogCat en el nivel de debugging.

3.1.1.2. AlertDialog

Los `AlertDialog` son útiles para pedir confirmaciones, o bien formular preguntas que requieran pulsar "aceptar" o "cancelar". A continuación se muestra un ejemplo de la documentación oficial de Android:

```
AlertDialog.Builder builder = new AlertDialog.Builder(this);
builder.setMessage("Are you sure you want to exit?")
       .setCancelable(false)
       .setPositiveButton("Yes", new DialogInterface.OnClickListener() {
           public void onClick(DialogInterface dialog, int id) {
                MyActivity.this.finish();
           }
       })
       .setNegativeButton("No", new DialogInterface.OnClickListener() {
           public void onClick(DialogInterface dialog, int id) {
                dialog.cancel();
           }
       });
AlertDialog alert = builder.create();
```

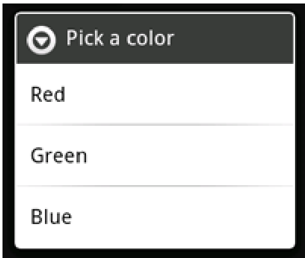

Copyright © 2014 Depto. CCIA All rights reserved.



Alert Dialog (de developer.android.com)

Si queremos una lista de la que seleccionar, podemos conseguirlo de la siguiente manera:

```
final CharSequence[] items = {"Red", "Green", "Blue"};

AlertDialog.Builder builder = new AlertDialog.Builder(this);
builder.setTitle("Pick a color");
builder.setItems(items, new DialogInterface.OnClickListener() {
    public void onClick(DialogInterface dialog, int item) {
        Toast.makeText(getApplicationContext(), items[item],
Toast.LENGTH_SHORT).show();
    }
});
AlertDialog alert = builder.create();
```



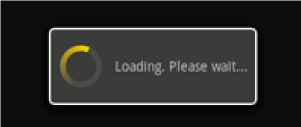
AlertDialog con lista (de developer.android.com)

3.1.1.3. ProgressDialog

Los `ProgressDialog` sirven para indicar progreso. Por ejemplo,

```
ProgressDialog dialog = ProgressDialog.show(MyActivity.this, "",
                "Loading. Please wait...", true);
```

genera un diálogo con indicador de progreso indefinido:



Diálogo de progreso indefinido (de developer.android.com)

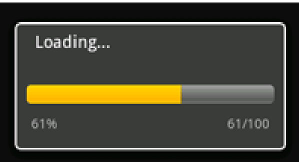
mientras que

```
ProgressDialog progressDialog;
progressDialog = new ProgressDialog(mContext);
progressDialog.setProgressStyle(ProgressDialog.STYLE_HORIZONTAL);
```


Copyright © 2014 Depto. CCIA All rights reserved.

```
progressDialog.setMessage("Loading...");
progressDialog.setCancelable(false);
```

genera un diálogo con una barra de progreso horizontal:



Diálogo con barra de progreso (de developer.android.com)

En este caso, para indicar el progreso se utiliza el método `progressDialog.setProgress(int)` indicando el porcentaje total de progreso, y lo típico sería que esta llamada se hiciera desde otro hilo con un `Handler`, o bien desde el método `AsyncTask.onProgressUpdate(String)`.

3.1.1.4. InputFilter

Cuando se introduce texto en un `EditText`, el contenido permitido se puede limitar y/o corregir usando un `InputFilter` o una colección de ellos. Hay dos `InputFilter` ya creados, uno es para obligar a que todo sean mayúsculas y el otro es para limitar la longitud del campo. Además se pueden crear filtros personalizados. En el siguiente ejemplo se asignan tres filtros (uno de cada tipo) a un campo de texto. Los filtros se aplican por el orden en el que estén en el vector.

```
EditText editText = (EditText)findViewById(R.id.EditText01);
InputFilter[] filters = new InputFilter[3];
filters[0] = new InputFilter.LengthFilter(9);
filters[1] = new InputFilter.AllCaps();
filters[2] = new InputFilter() {
    public CharSequence filter(CharSequence source, int start, int end,
            Spanned dest, int dstart, int dend) {
    if (end > start) {
        String destTxt = dest.toString();
        String resultingTxt = destTxt.substring(0, dstart) +
                source.subSequence(start, end) + destTxt.substring(dend);
        if (!resultingTxt.matches("^[A-F0-9]*$")) {
         if (source instanceof Spanned) {
             SpannableString sp = new SpannableString("");
             return sp;
         } else {
             return "";
         }
        }
    }
    return null;
    }
};
dniEditText.setFilters(filters);
```

El último filtro comprueba que se cumpla la expresión regular `^[A-F0-9]*$` (caracteres de número hexadecimal).

Copyright © 2014 Depto. CCIA All rights reserved.

3.2. Layouts

Los Layouts son una extensión de la clase `ViewGroup` y se utilizan para posicionar controles (Views) en la interfaz de usuario. Se pueden anidar unos dentro de otros.

Los layout se pueden definir en formato XML en la carpeta `res/layout`. Por ejemplo, el siguiente layout lineal dispondrá sus elementos (TextView y Button) uno debajo del otro:

```
<LinearLayout
android:orientation="vertical"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:gravity="right">

        <TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/hello"
    />

        <Button android:text="Siguiente"
        android:id="@+id/Button01"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        />

</LinearLayout>
```

Para disponerlos uno al lado del otro se utiliza `orientation="horizontal"`. Por otro lado, el atributo `gravity` indica hacia qué lado se van a alinear los componentes.

Algunos de los layouts más utilizados son:

- `LinearLayout`, dispone los elementos uno después del otro.
- `FrameLayout`, dispone cualquier elemento en la esquina superior izquierda.
- `RelativeLayout`, dispone los elementos en posiciones relativas con respecto a otros, y con respecto a las fronteras del layout.
- `TableLayout`, dispone los elementos en forma de filas y columnas.
- `Gallery`, dispone los elementos en una única fila desplazable.

3.3. Eventos

Para que los views sean usables, hay que asignar manejadores a los eventos que nos interesen. Por ejemplo, para un `Button` podemos asociar un comportamiento asignándole un `onClickListener`:

```
        ImageButton imageButton =
(ImageButton)findViewById(R.id.ImageButton01);
        imageButton.setOnClickListener(new OnClickListener() {

                        @Override
                        public void onClick(View v) {
```


Copyright © 2014 Depto. CCIA All rights reserved.

```
                              Toast.makeText(getApplicationContext(),
                               "Gracias por pulsar.",
Toast.LENGTH_SHORT).show();
                       }
               });
```

Se pueden escuchar los eventos de cualquier otro tipo de view, incluso de los `TextView`.

3.4. Activities e Intents

Ya estamos familiarizados con las actividades de android: se trata de tareas que muestran un interfaz gráfico al usuario, y sólo podemos ver en pantalla una Activity a la vez. Muchas aplicaciones tienen una actividad principal que puede llevarnos a otras actividades de la aplicación, o incluso a actividades de otras aplicaciones.

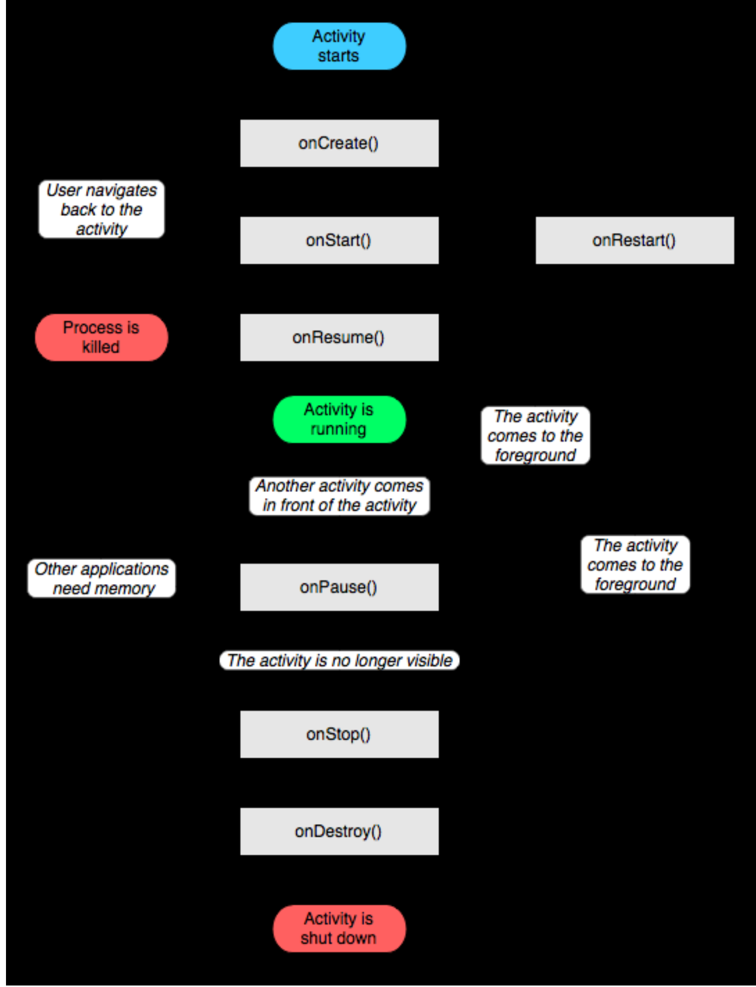


Diagrama de la documentación oficial de Android, que representa el ciclo de vida de las

Copyright © 2014 Depto. CCIA All rights reserved.

actividades.

Este ciclo de vida puede definirse por medio de los siguientes métodos:

```
public class Activity extends ApplicationContext {
    protected void onCreate(Bundle savedInstanceState);

    protected void onStart();

    protected void onRestart();

    protected void onResume();

    protected void onPause();

    protected void onStop();

    protected void onDestroy();
}
```

Hay tres subciclos típicos en la vida de una actividad:

- Todo el ciclo de vida ocurre desde la primera llamada a `onCreate(Bundle)` hasta la llamada (única) al método `onDestroy()`. Por tanto en esta última se deben liberar los recursos que queden por liberar.
- El tiempo de vida visible ocurre entre `onStart()` y `onStop()`. Durante este tiempo el usuario verá la actividad en pantalla, incluso aunque ésta no tenga el foco en este momento o esté en segundo plano (pero visible). Los métodos `onStart()` y `onStop()` podrán ser llamados múltiples veces, según la actividad se haga visible o se oculte.
- El tiempo de vida en primer plano ocurre entre los métodos `onResume` y `onPause`. La actividad puede pasar con frecuencia entre el estado pausado y primer plano, de manera que el código de estos métodos debe ser rápido.

Nota:

Cuando una actividad se pausa, ésta puede no volver nunca a primer plano sino ser matada debido a que el sistema operativo lo decida así, por falta de recursos de memoria. Por tanto tendremos que intentar guardar los estados de nuestras actividades de tal manera que si la actividad se retoma con `onResume()`, el usuario tenga la misma experiencia que si arranca la aplicación de nuevo. Para ello nos ayuda el parámetro de `onCreate(Bundle)` que guarda el estado de los formularios y los rellena de nuevo "automágicamente", sin tener que programarlo nosotros.

Para pasar de una actividad a otra se utilizan los `Intent`. Un Intent es una descripción abstracta de una operación a realizar. Se puede utilizar con el método `startActivity` para lanzar una actividad, con `broadcastIntent` para enviarse a cualquier componente receptor `BroadcastReceiver`, y con `startService` o `bindService` para comunicar con un servicio (`Service`) que corre en segundo plano.

Por ejemplo, para lanzar la actividad llamada `MiActividad`, lo haremos así:

Copyright © 2014 Depto. CCIA All rights reserved.

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

Para iniciar una llamada de teléfono también utilizaremos intents:

```
 Intent intent = new Intent(Intent.ACTION_DIAL,
Uri.parse("tel:965903400"));
 startActivity(intent);
```

Podemos pasar un código de petición a la actividad:

```
startActivityForResult(intent, CODIGO_INT);
```

Y podemos esperar un resultado de la actividad, generándolo así

```
Intent resultado = new Intent(null, Uri.parse("content://datos/1");
resultado.putExtra(1,"aaa");
resultado.putExtra(2,"bbb");

if(ok)
        setResult(RESULT_OK, resultado);
else
        setResult(RESULT_CANCELED, null);

finish();
```

y recogiendo el resultado con el método `onActivityResult()` sobrecargado:

```
@Override
public void onActivityResult(int reqCode, int resultCode, Intent data){
        switch(reqCode){
        case 1:
                break;
        case 2:
                break;
        default:
        }
}
```

Algunas acciones nativas da Android son:

- ACTION_ANSWER
- ACTION_CALL
- ACTION_DELETE
- ACTION_DIAL
- ACTION_EDIT
- ACTION_INSERT
- ACTION_PICK
- ACTION_SEARCH
- ACTION_SENDTO
- ACTION_VIEW
- ACTION_WEB_SEARCH

Copyright © 2014 Depto. CCIA All rights reserved.

3.4.1. Fragmentos

Android 3.0 y 4.0 está preparado para tablets. Uno de los objetivos que se persiguen es que las aplicaciones puedan estar preparadas tanto para pantallas grandes como para pantallas pequeñas. Los `Fragments` permiten al programador ejecutar varias actividades en una misma pantalla. De esta manera, si la pantalla es grande (o bien si la pantalla está en modo apaisado), se pueden mostrar varias actividades, que de otra manera se irían mostrando una a una. El típico ejemplo es el de una lista para cuyos items se muestra una pantalla con detalles. Podría, o bien mostrarse como una actividad nueva que se abre y sustituye a la anterior, o bien como otro Fragment al lado de la primera. Los Fragment se definen en XML en los layouts y en ellos se indica la ruta de la Activity que se corresponde con cada Fragment.

3.5. Menús y preferencias

Hay varios tipos de menús en Android:

- los "icon menu" que aparecen en la parte inferior de la pantalla cuando se pulsa el botón físico de menú
- menús expandidos que aparecen cuando el usuario pulsa el botón "más" del incon menu
- los submenús que aparecen en forma de ventanas flotantes, de las que se sale con la tecla física "atrás" (o "cerrar")
- los menús contextuales que se abren al pulsar prolongadamente sobre un componente

Un icon menú se puede definir en un archivo XML, por ejemplo, en `res/menu/menu.xml`:

```
<?xml version="1.0" encoding="utf-8"?>
<menu
  xmlns:android="http://schemas.android.com/apk/res/android">
<item android:title="Preferencias" android:id="@+id/item01"></item>
<item android:title="Acerca de..." android:id="@+id/item02"></item>
</menu>
```

Para que el menú se infle al pulsar el botón debemos sobrecargar la función `onCreateOptionsMenu(Menu m)` de nuestra actividad. Esta función deberá desplegar el menú con la función `getMenuInflater().inflate(R.menu.menu, menu);` y devolver `true`:

```
 @Override
public boolean onCreateOptionsMenu(Menu m) {
        getMenuInflater().inflate(R.menu.menu, m);
        return true;
}
```

Para programar las respuestas a las pulsaciones de cada opción del menú tenemos que

Copyright © 2014 Depto. CCIA All rights reserved.

sobrecargar el método `onOptionsItemSelected()`:

```
 @Override
public boolean onOptionsItemSelected(MenuItem item) {
        switch(item.getItemId()){
        case R.id.item01:
                break;
        case R.id.item02:
                break;
        }
        return true;
}
```

En cuanto a los menús contextuales, para crearlos habrá que sobrecargar la función `onCreateContextMenu(ContextMenu)` del componente correspondiente:

```
@Override
public void onCreateContextMenu(ContextMenu m){
        super.onCreateContextMenu(m);
        m.add("ContextMenuItem1");
}
```

Otra alternativa para los menús contextuales es registrar el view con el método `registerForContextMenu(view)`, para que así el menú contextual para este view sólo esté disponible en esta actividad, y no en cualquier actividad en la que se incluya el view.

3.6. Visor de Google Maps

Hay dos formas de utilizar el servicio de Google Maps. Una es lanzando un nuevo Intent para que se abra una actividad aparte de la aplicación. A pesar de abrir una nueva aplicación el efecto es igual que el de abrir otra pantalla de la aplicación. Al pulsar el botón "atrás", se cerrará la aplicación Google Maps y volverá a la última actividad, la de la aplicación que estaba en ejecución.

La otra forma es abrir una actividad que contenga dentro un visor de Google Maps. Para ello primero hay que añadir un visor de Google Maps al layout de la actividad, por ejemplo, `main.xml`. Para contar con ese componente necesitamos que la plataforma para la que se desarrolle nuestro proyecto sea "Google API", sea del nivel que sea.

Para usar el MapView desde nuestra aplicación necesitamos obtener una clave para la API de Google Maps. Ésta se genera a partir de la huella digital MD5 del certificado digital que usamos para firmar nuestras aplicaciones. Para el desarrollo será suficiente con que utilicemos el certificado de debug que se crea por el Android SDK para desarrollar. Sin embargo éste no nos servirá para producción. Para generar la clave para la API debemos seguir los siguientes pasos:

- Generar una huella digital MD5 para el certificado de debug. Encuéntralo en un fichero llamado debug.keystore, cuya ruta está indicada en Eclipse en las preferencias

Copyright © 2014 Depto. CCIA All rights reserved.

de Android Build. Desde el directorio indicado en la ruta, ejecutamos:

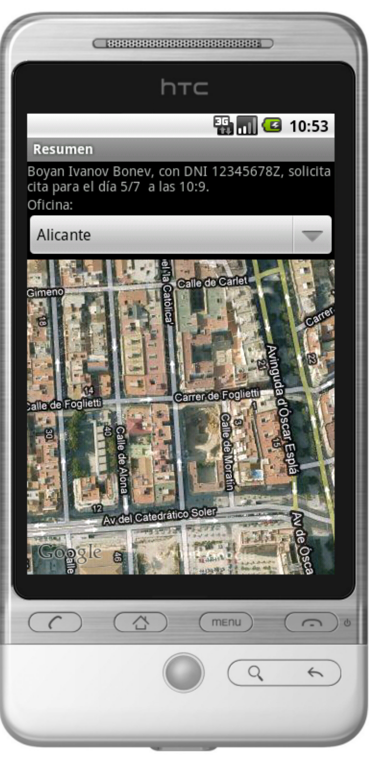
```
keytool -list -keystore debug.keystore
```

- Entramos en http://code.google.com/android/add-ons/google-apis/maps-api-signup.html y rellenamos el formulario. Para ello tenemos que autenticarnos con nuestra cuenta de Google. Es ahí donde pegaremos nuestra huella digital. Finalmente nos muestra la clave de la API, la copiamos y nos la guardamos.

Más información en http://code.google.com/android/add-ons/google-apis/mapkey.html

En la declaración del componente `MapView` del layor introduciremos la clave para la API que hemos obtenido. También se debe añadir al `AndroidManifest.xml` que la aplicación usa la librería de google maps, y que requiere permisos para Internet.

En el método `onCreate()` de la actividad se debe obtener la referencia al mapa y crear un controlador `MapController` a partir del `MapView` para poder moverlo. Desde ahí se puede controlar el mapa, por ejemplo, hacer zoom.



Visor de Google Maps en nuestra aplicación.

Copyright © 2014 Depto. CCIA All rights reserved.

4. Intefaz de usuario - Ejercicios

4.1. LinearLayout

Crea una aplicación llamada *LinearLayout*. La aplicación contendrá una única actividad, llamada *LinearLayout*, cuya interfaz gráfica estará contruida exclusivamente a partir de layouts de tipo `LinearLayout` y deberá ser lo más parecida posible a la mostrada en la siguiente imagen.

Interfaz gráfica de la aplicación LinearLayout

> **Nota:**
> Las líneas han sido creadas por medio de elementos `View` a los que se les ha asignado una altura de *1dip* mediante el atributo `android:layout_height` y un color de fondo *#FFFFFF* mediante el atributo `android:background`.

4.2. Ciudades

En este ejercicio practicaremos con los elementos de tipo `Spinner`. Importa el proyecto de las plantillas de la sesión. La aplicación *Ciudades* contendrá una única actividad. La interfaz de dicha actividad estará compuesta por un `TextView` con `android:id="@+id/texto"`y dos elementos de tipo `Spinner` con identificadores `android:id="@+id/paises"` y `android:id="@+id/ciudades"`. El primero de ellos permitirá escoger entre tres países cualquiera (inicialmente ningún país estará seleccionado). El segundo permitirá escoger una ciudad según el país seleccionado en el

Copyright © 2014 Depto. CCIA All rights reserved.

anterior. Cada vez que se seleccione un país en el primer `Spinner` deberán cambiar las opciones del segundo, mostrando dos ciudades del país seleccionado.

La ciudad seleccionada en el segundo `Spinner` aparecerá en el `TextView` de la parte superior.

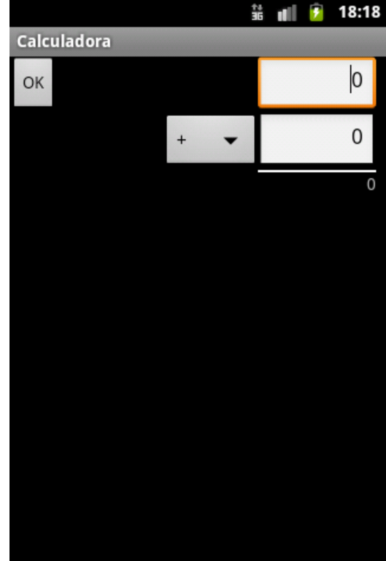
Para completar el ejercicio debes seguir los siguientes pasos:

- Añade el `TextView` y los dos `Spinner` al recurso layout de la aplicación, sin olvidar añadir a estos dos últimos su correspondiente atributo `android:prompt` (con los textos "Selecciona país" y "Selecciona ciudad" respectivamente).
- Las opciones para los `Spinner` ya están creadas en el archivo *arrays.xml* de los recursos. Ábrelo para ver los diferentes conjuntos que hay.
- El primer `Spinner` ya viene rellenado con sus correspondientes opciones.
- Rellena el segundo `Spinner` con las ciudades correspondintes al primer país. Esto debes hacerlo así porque siempre que inicies la actividad será el primer país el que se encuentre seleccionado.
- Asígnale al TextView como valor inicial el nombre de la primera ciudad, pues será la que se encontrará seleccionada al iniciar la actividad.
- El `Spinner` de paises ya tiene un manejador añadido para que cada vez que se seleccione una opción se muestren las opciones adecuadas en el `Spinner` de ciudades.
- Añade un manejador al `Spinner` de ciudades para que cada vez que se seleccione una opción se muestre en el `TextView`. Para obtener el texto correspondiente a la opción seleccionada en el `Spinner` puedes utilizar el método `getSelectedItem` del mismo. Una vez hecho esto puedes llamar al método `toString` para obtener la cadena correspondiente.

4.3. Calculadora sencilla

El objetivo de este ejercicio es implementar una calculadora sencilla. La aplicación *Calculadora* contendrá una única actividad de nombre *Principal*, vuya interfaz gráfica tendrá el siguiente aspecto:

Copyright © 2014 Depto. CCIA All rights reserved.



Interfaz gráfica de la aplicación Calculadora

Como se puede observar nuestra calculadora es bastante limitada. Tan solo acepta dos operandos (que se podrán introducir en los dos `EditText`) y cuatro operaciones seleccionables con el `Spinner`: +, -, * y /. En el `TextView` inferior deberá aparecer el resultado de la operación cuando se pulse el botón *Ok*.

A la hora de diseñar la interfaz se ha utilizado un `RelativeLayout`. Los atributos más importantes utilizados han sido: `layout_alignParentRight`, `layout_below`, `align_marginRight`, `android:inputType="number"` para los `EditText` y `android:gravity="right"` para el `TextView` y los `EditText`.

4.4. Abrir actividades desde un menú (*)

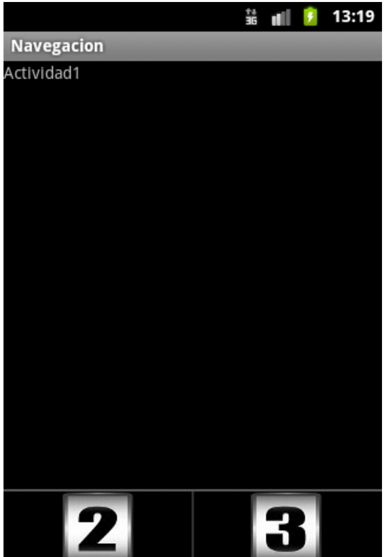
En este ejercicio vamos a importar las plantillas de una aplicación llamada *Activities* para navegar entre tres actividades. Las tres actividades tendrán de nombre *Actividad1*, *Actividad2* y *Actividad3*. La interfaz gráfica de cada una de ellas consistirá únicamente en un `TextView` mostrnado su nombre. Desde cada una de ellas se podrá acceder a las otras dos mediante un menú de iconos. Esto quiere decir que el menú de cada actividad será distinto y se compondrá de dos únicas opciones.

En las plantillas de la sesión se incluye un proyecto con una de las actividades ya creada. Se incluyen iconos para el menú, llamados *icono1.png*, *icono2.png* y *icono3.png*.

Recuerda que para crear una nueva actividad hay que añadirla también al `AndroidManifest.xml`, antes del cierre de `<application/>`:

```
<activity android:name=".Actividad2"
          android:label="@string/app_name">
</activity>
```


Copyright © 2014 Depto. CCIA All rights reserved.



Interfaz de la aplicación Navegacion

Nota:

Prueba a asignar al atributo `android:noHistory` de cada actividad en el *Manifest* de la aplicación el valor `true`. Çomprueba cuál es el nuevo comportamiento al pulsar el botón "Atrás" del dispositivo móvil.

4.5. Fragments (*)

Importa el proyecto `Fragments` de las plantillas. Ejecútalo en un emulador con Android 4.0.x y cambia la orientación del emulador con Ctrl-F11. Comprueba cómo funcionan los fragmentos en cada caso.

Copyright © 2014 Depto. CCIA All rights reserved.

5. Gráficos avanzados

Hasta este momento hemos visto como crear la interfaz de usuario de nuestra aplicación utilizando una serie de componentes predefinidos. Ahora vamos a ver cómo personalizar los componentes existentes o crear nuestros propios componentes.

Comenzaremos viendo cómo establecer el aspecto de los diferentes componentes de la interfaz de los que disponemos en Android, para así poder dar a nuestras aplicaciones un estilo gráfico propio. A continuación, veremos cómo crear componentes propios, en los que podamos tener un control absoluto sobre lo que se dibuja en pantalla. Por último, trataremos la forma de crear gráficos 3D y aplicaciones que necesiten una elevada tasa de refresco, como son los videojuegos.

5.1. Elementos drawables

Un *drawable* es un tipo de recurso que puede ser dibujado en pantalla. Podremos utilizarlos para especificar el aspecto que van a tener los diferentes componentes de la interfaz, o partes de éstos. Estos *drawables* podrán ser definidos en XML o de forma programática. Entre los diferentes tipos de *drawables* existentes encontramos:

- **Color**: Rellena el lienzo de un determinado color.
- **Gradiente**: Rellena el lienzo con un gradiente.
- **Forma** (*shape*): Se pueden definir una serie de primitivas geométricas básicas como *drawables*.
- **Imágen** (*bitmap*): Una imagen se comporta como *drawable*, ya que podrá ser dibujada y referenciada de la misma forma que el resto.
- **Nine-patch**: Tipo especial de imagen PNG que al ser escalada sólo se escala su parte central, pero no su marco.
- **Animación**: Define una animación por fotogramas, como veremos más adelante.
- **Capa** (*layer list*): Es un *drawable* que contiene otros *drawables*. Cada uno especificará la posición en la que se ubica dentro de la capa.
- **Estados** (*state list*): Este *drawable* puede mostrar diferentes contenidos (que a su vez son *drawables*) según el estado en el que se encuentre. Por ejemplo sirve para definir un botón, que se mostrará de forma distinta según si está normal, presionado, o inhabilitado.
- **Niveles** (*level list*): Similar al anterior, pero en este caso cada *item* tiene asignado un valor numérico (nivel). Al establecer el nivel del *drawable* se mostrará el *item* cuyo nivel sea mayor o igual que el indicado.
- **Transición** (*transition*): Nos permite mostrar una transición de un *drawable* a otro mediante un fundido.
- **Inserción** (*inset*): Ubica un *drawable* dentro de otro, en la posición especificada.
- **Recorte** (*clip*): Realiza un recorte de un *drawable*.
- **Escala** (*scale*): Cambia el tamaño de un *drawable*.

Copyright © 2014 Depto. CCIA All rights reserved.

Todos los *drawables* derivan de la clase `Drawable`. Esta nos permite que todos ellos puedan ser utilizados de la misma forma, independientemente del tipo del que se trate. Se puede consultar la lista completa de *drawables* y su especificación en la siguiente dirección:

http://developer.android.com/guide/topics/resources/drawable-resource.html

Por ejemplo, vamos a definir un *drawable* que muestre un rectángulo rojo con borde azul, creando un fichero XML de nombre `rectangulo.xml` en el directorio `/res/drawable/`. El fichero puede tener el siguiente contenido:

```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android=
    "http://schemas.android.com/apk/res/android"
    android:shape="rectangle">
  <solid android:color="#f00"/>
  <stroke android:width="2dp" android:color="#00f"
    android:dashWidth="10dp" android:dashGap="5dp"/>
</shape>
```

Podremos hacer referencia a este rectángulo desde el código mediante `R.drawable.rectangulo` y mostrarlo en la interfaz asignándolo a un componente de alto nivel como por ejemplo `ImageView`, o bien hacer referencia a él desde un atributo del XML mediante `@drawable/rectangulo`. Por ejemplo, podríamos especificar este *drawable* en el atributo `android:background` de la etiqueta `Button` dentro de nuestro *layout*, para que así el botón pase a tener como aspecto una forma rectangular de color rojo y con borde azul. De la misma forma podríamos darle al botón el aspecto de cualquier otro tipo de *drawable* de los vistos anteriormente. A continuación vamos a ver con más detalle los tipos de *drawables* más interesantes.

Nota

Podemos definir diferentes variantes del directorio de *drawables*, para así definir diferentes versiones de los *drawables* para los distintos tipos de dispositivos, por ejemplo según su densidad, tamaño, o forma de pantalla. Esto se aplica a todos los directorios de recursos. Por ejemplo se pueden diferenciar los recursos también según el idioma del dispositivo, o la orientación de la pantalla, lo cual se puede aprovechar para definir *layouts* diferentes. Para más información sobre los tipos de variantes disponibles consultar http://developer.android.com/guide/topics/resources/providing-resources.html#AlternativeResources

5.1.1. Imágenes

Las imágenes que introduzcamos en los directorios de recursos de tipo *drawable* (`/res/drawable/`) podrán ser tratadas igual que cualquier otro tipo de *drawable*. Por ejemplo, si introducimos en dicho directorio una imagen `titulo.png`, podremos hacer referencia a ella en los atributos de los XML mediante `@drawable/titulo` (no se pone la extensión), o bien desde el código mediante `R.drawable.titulo`.

Las imágenes se encapsulan en la clase `Bitmap`. Los *bitmaps* pueden ser mutables o

Copyright © 2014 Depto. CCIA All rights reserved.

inmutables, según si se nos permite modificar el valor de sus pixels o no respectivamente.

Si el *bitmap* se crea a partir de un *array* de pixels, de un recurso con la imagen, o de otro *bitmap*, tendremos un *bitmap* inmutable.

Si creamos el *bitmap* vacío, simplemente especificando su altura y su anchura, entonces será mutable (en este caso no tendría sentido que fuese inmutable ya que sería imposible darle contenido). También podemos conseguir un *bitmap* mutable haciendo una copia de un *bitmap* existente mediante el método `copy`, indicando que queremos que el *bitmap* resultante sea mutable.

Para crear un *bitmap* vacío, a partir de un *array* de pixels, o a partir de otro *bitmap*, tenemos una serie de métodos estáticos `createBitmap` dentro de la clase `Bitmap`.

Para crear un *bitmap* a partir de un fichero de imagen (GIF, JPEG, o PNG, siendo este último el formato recomendado) utilizaremos la clase `BitmapFactory`. Dentro de ella tenemos varios métodos con prefijo `decode` que nos permiten leer las imágenes de diferentes formas: de un *array* de bytes en memoria, de un flujo de entrada, de un fichero, de una URL, o de un recurso de la aplicación. Por ejemplo, si tenemos una imagen (`titulo.png`) en el directorio de *drawables* podemos leerla como `Bitmap` de la siguiente forma:

```
Bitmap imagen = BitmapFactory.decodeResource(getResources(),
    R.drawable.titulo);
```

Al crear un *bitmap* a partir de otro, podremos realizar diferentes transformaciones (escalado, rotación, etc).

Una vez no se vaya a utilizar más el *bitmap*, es recomendable liberar la memoria que ocupa. Podemos hacer esto llamando a su método `recycle`.

5.1.2. Imágenes nine-patch

Como hemos comentado anteriormente, utilizaremos los *drawables* para especificar el aspecto que queremos que tengan los componentes de la interfaz. La forma más flexible de definir este aspecto es especificar una imagen propia. Sin embargo, encontramos el problema de que los componentes normalmente no tendrán siempre el mismo tamaño, sino que Android los "estirará" según su contenido y según los parámetros de *layout* especificados (es decir, si deben ajustarse a su contenido o llenar todo el espacio disponible). Esto es un problema, ya que si siempre especificamos la misma imagen como aspecto para estos componentes, al estirarla veremos que ésta se deforma, dando un aspecto terrible a nuestra aplicación, como podemos ver a continuación:

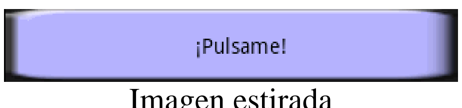
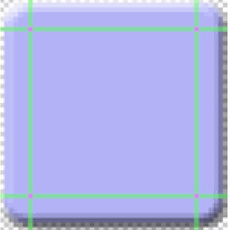


Imagen estirada

Sin embargo, tenemos un tipo especial de imágenes PNG llamadas *nine-patch* (llevan

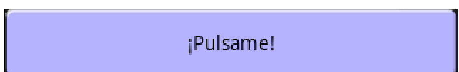
Copyright © 2014 Depto. CCIA All rights reserved.

extensión `.9.png`), que nos permitirán evitar este problema. Normalmente la parte central de nuestros componentes es homogénea, por lo que no pasa nada si se estira. Sin embargo, los bordes si que contienen un mayor número de detalles, que no deberían ser deformados, especialmente las esquinas. Las imágenes *nine-patch* se dividen en 9 regiones: la parte central, que puede ser escalada en cualquier dirección, las esquinas, que nunca pueden escaladas, y los bordes, que sólo pueden ser escalados en su misma dirección (horizontal o vertical). A continuación vemos un ejemplo de dicha división:



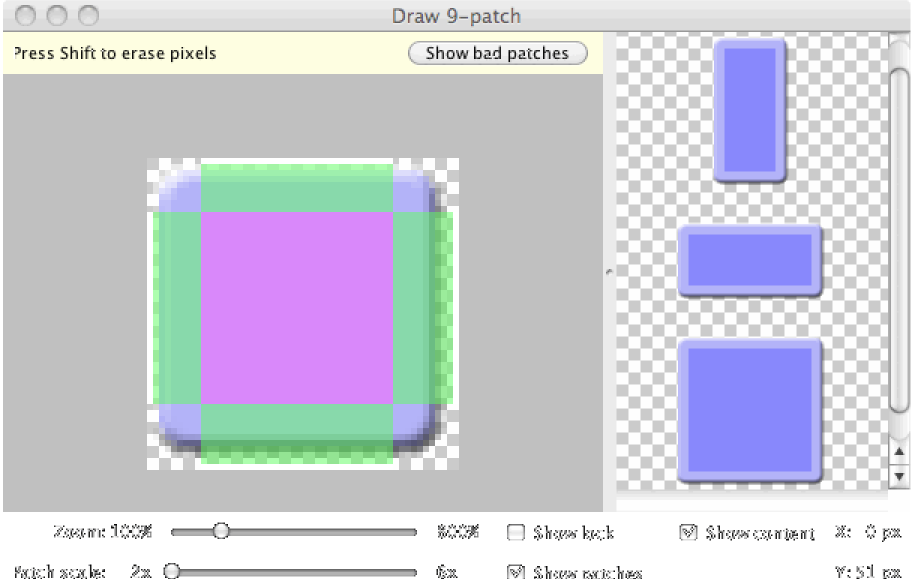
Parches de la imagen

Si ponemos una imagen de este tipo como *drawable* de fondo para un botón, veremos que siempre se mostrará con el aspecto correcto, independientemente de su contenido:



Aplicación de nine-patch a un botón

Podemos crear este tipo de imágenes con la herramienta `draw9patch` que podemos encontrar en el subdirectorio `tools` del SDK de Android. Lo único que necesitaremos es arrastrar el PNG que queramos tratar como *nine-patch*, y añadir una serie de píxeles en el marco de la imagen para marcar las regiones:



Copyright © 2014 Depto. CCIA All rights reserved.

Herramienta draw9patch

La fila de píxeles superior y la columna izquierda indican las zonas de la imagen que son flexibles y que se pueden ampliar si es necesario repitiendo su contenido. En el caso de la fila superior, indica que se pueden estirar en la horizontal, mientras que los del lateral izquierdo corresponden a la vertical.

Opcionalmente podemos especificar en la fila inferior y en la columna derecha la zona que utilizaremos como contenido. Por ejemplo, si utilizamos la imagen como marco de un botón, esta será la zona donde se ubicará el texto que pongamos en el botón. Marcando la casilla *Show content* veremos en el lateral derecho de la herramienta una previsualización de la zona de contenido.

5.1.3. Lista de estados

Siguiendo con el ejemplo del botón, encontramos ahora un nuevo problema. Los botones no deben tener siempre el mismo aspecto de fondo, normalmente cambiarán de aspecto cuando están pulsados o seleccionados, sin embargo sólo tenemos la posibilidad de especificar un único *drawable* como fondo. Para poder personalizar el aspecto de todos los estados en los que se encuentra el botón tenemos un tipo de *drawable* llamado *state list drawable*. Se define en XML, y nos permitirá especificar un *drawable* diferente para cada estado en el que se puedan encontrar los componentes de la interfaz, de forma que en cada momento el componente mostrará el aspecto correspondiente a su estado actual.

Por ejemplo, podemos especificar los estados de un botón (no seleccionado, seleccionado, y pulsado) de la siguiente forma:

```
<?xml version="1.0" encoding="utf-8"?>
<selector xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:state_pressed="true"
          android:drawable="@drawable/boton_pressed" /> <!-- presionado
-->
    <item android:state_focused="true"
          android:drawable="@drawable/boton_selected" /> <!-- seleccionado
-->
    <item android:drawable="@drawable/boton_normal" /> <!-- no
seleccionado -->
</selector>
```

Los *drawables* especificados para cada estado pueden ser de cualquier tipo (por ejemplo imágenes, *nine-patch*, o formas definidas en XML).

Un *drawable* similar es el de tipo *level list*, pero en este caso los diferentes posibles *drawables* a mostrar se especifican para un rango de valores numéricos. ¿Para qué tipos de componentes de la interfaz podría resultar esto de utilidad?

5.1.4. Animación por fotogramas

Este tipo de *drawable* nos permite definir una animación a partir de diferentes fotogramas, que deberemos especificar también como *drawables*, además del tiempo en

Copyright © 2014 Depto. CCIA All rights reserved.

milisegundos que durará el fotograma. Se definen en XML de la siguiente forma:

```
<animation-list
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:oneshot="false">
  <item android:drawable="@drawable/spr0" android:duration="50" />
  <item android:drawable="@drawable/spr1" android:duration="50" />
  <item android:drawable="@drawable/spr2" android:duration="50" />
</animation-list>
```

Además, la propiedad *one shot* nos indica si la animación se va a reproducir sólo una vez o en bucle infinito. Al ponerla como `false` especificamos que se reproduzca de forma continuada.

Desde el código, podremos obtener la animación de la siguiente forma, considerando que la hemos guardado en un fichero `animacion.xml`:

```
AnimationDrawable animFotogramas =
getResources().getDrawable(R.drawable.animacion);
```

De forma alternativa, podríamos haberla definido de forma programática de la siguiente forma:

```
BitmapDrawable f1 = (BitmapDrawable)getResources()
    .getDrawable(R.drawable.sprite0);
BitmapDrawable f2 = (BitmapDrawable)getResources()
    .getDrawable(R.drawable.sprite1);
BitmapDrawable f3 = (BitmapDrawable)getResources()
    .getDrawable(R.drawable.sprite2);
AnimationDrawable animFotogramas = new AnimationDrawable();

animFotogramas.addFrame(f1, 50);
animFotogramas.addFrame(f2, 50);
animFotogramas.addFrame(f3, 50);

animFotogramas.setOneShot(false);
```

Nota

La diferencia entre `Bitmap` y `BitmapDrawable` reside en que en el primer caso simplemente tenemos una imagen, mientras que en el segundo lo que tenemos es un *drawable* que encapsula una imagen, es decir, se le podrá proporcionar a cualquier componente que acepte *drawables* en general como entrada, y concretamente lo que dibujará será la imagen (`Bitmap`) que contiene.

Para que comience la reproducción deberemos llamar al método `start` de la animación:

```
animFotogramas.start();
```

De la misma forma, podemos detenerla con el método `stop`:

```
animFotogramas.stop();
```

Importante

El método `start` no puede ser llamado desde el método `onCreate` de nuestra actividad, ya que en ese momento el *drawable* todavía no está vinculado a la vista. Si lo que queremos es que se ponga en marcha nada más cargarse la actividad, el lugar idóneo para invocarlo es el evento

Copyright © 2014 Depto. CCIA All rights reserved.

> `onWindowFocusChanged`. Lo recomendable será llamar a `start` cuando obtengamos el foco, y a `stop` cuando lo perdamos.

5.1.5. Definición programática

Vamos a suponer que tenemos un `ImageView` con identificador `visor` y un drawable de nombre `rectangulo`. Normalmente especificaremos directamente en el XML el *drawable* que queremos mostrar en el `ImageView`. Para ello deberemos añadir el atributo `android:src = "@drawable/rectangulo"` en la definición del `ImageView`.

Podremos también obtener una referencia a dicha vista y mostrar en ella nuestro rectángulo especificando el identificador del *drawable* de la siguiente forma:

```
ImageView visor = (ImageView)findViewById(R.id.visor);
visor.setImageResource(R.drawable.rectangulo);
```

Otra alternativa para mostrarlo es obtener primero el objeto `Drawable` y posteriormente incluirlo en el `ImageView`:

```
Drawable rectangulo = this.getResources()
    .getDrawable(R.drawable.rectangulo);
visor.setImageDrawable(rectangulo);
```

Estas primitivas básicas también se pueden crear directamente de forma programática. En el paquete `android.graphics.drawable.shape` podemos encontrar clases que encapsulan diferentes formas geométricas. Podríamos crear el rectángulo de la siguiente forma:

```
RectShape r = new RectShape();
ShapeDrawable sd = new ShapeDrawable(r);
sd.getPaint().setColor(Color.RED);
sd.setIntrinsicWidth(100);
sd.setIntrinsicHeight(50);

visor.setImageDrawable(sd);
```

5.2. Componentes propios

Si no hay ningún componente predefinido que se adapte a nuestras necesidades, podemos crear un nuevo tipo de vista (`View`) en la que especificaremos exactamente qué es lo que queremos dibujar en la pantalla. El primer paso consistirá en crear una subclase de `View` en la que sobrescribiremos el método `onDraw`, que es el que define la forma en la que se dibuja el componente.

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

        @Override
        protected void onDraw(Canvas canvas) {
                // TODO Definir como dibujar el componente
```


Copyright © 2014 Depto. CCIA All rights reserved.

```
        }
}
```

5.2.1. Lienzo y pincel

El método `onDraw` recibe como parámetro el lienzo (`Canvas`) en el que deberemos dibujar. En este lienzo podremos dibujar diferentes tipos de elementos, como primitivas geométricas, texto e imágenes.

> **Importante**
>
> No confundir el `Canvas` de Android con el `Canvas` que existe en Java ME/SE. En Java ME/SE el `Canvas` es un componente de la interfaz, que equivaldría a `View` en Android, mientras que el `Canvas` de Android es más parecido al objeto `Graphics` de Java ME/SE, que encapsula el contexto gráfico (o lienzo) del área en la que vamos a dibujar.

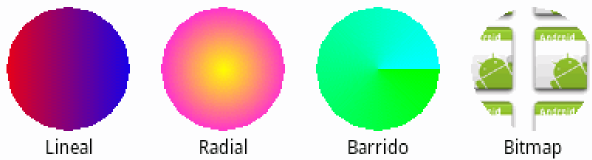
Además, para dibujar determinados tipos de elementos deberemos especificar también el tipo de pincel a utilizar (`Paint`), en el que especificaremos una serie de atributos como su color, grosor, etc.

Por ejemplo, para especificar un pincel que pinte en color rojo escribiremos lo siguiente:

```
Paint p = new Paint();
p.setColor(Color.RED);
```

Las propiedades que podemos establecer en el pincel son:

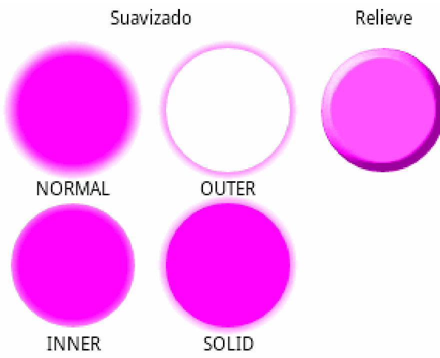
- **Color plano**: Con `setARGB` o `setColor` se puede especificar el código ARGB del color o bien utilizar constantes con colores predefinidos de la clase `Color`.
- **Gradientes y shaders**: Se pueden rellenar las figuras utilizando shaders de gradiente o de bitmap. Para utilizar un *shader* tenemos el método `setShader`, y tenemos varios *shaders* disponibles, como distintos *shaders* de gradiente (`LinearShader`, `RadialShader`, `SweepShader`), `BitmapShader` para rellenar utilizando un mapa de bits como patrón, y `ComposeShader` para combinar dos *shaders* distintos.



Tipos de gradiente

- **Máscaras**: Nos sirven para aplicar un suavizado a los gráficos (`BlurMaskFilter`) o dar efecto de relieve (`EmbossMaskFilter`). Se aplican con `setMaskFilter`.

Copyright © 2014 Depto. CCIA All rights reserved.

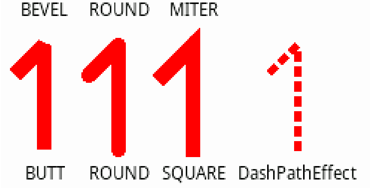


Mascaras de suavizado y relieve

- **Sombras**: Podemos crear efectos de sombra con `setShadowLayer`.
- **Filtros de color**: Aplica un filtro de color a los gráficos dibujados, alterando así su color original. Se aplica con `setColorFilter`.
- **Estilo de la figura**: Se puede especificar con `setStyle` que se dibuje sólo el trazo, sólo el relleno, o ambos.

Estilos de pincel

- **Estilo del trazo**: Podemos especificar el grosor del trazo (`setStrokeWidth`), el tipo de línea (`setPathEffect`), la forma de las uniones en las polilíneas (redondeada/`ROUND`, a inglete/`MITER`, o biselada/`BEVEL`, con `setStrokeJoin`), o la forma de las terminaciones (cuadrada/`SQUARE`, redonda/`ROUND` o recortada/`BUTT`, con `setStrokeCap`).



Tipos de trazo y límites

- **Antialiasing**: Podemos aplicar *antialiasing* con `setAntiAlias` a los gráficos para evitar el efecto *sierra*.
- **Dithering**: Si el dispositivo no puede mostrar los 16 millones de colores, en caso de haber un gradiente, para que el cambio de color no sea brusco, con esta opción (`setDither`) se mezclan pixels de diferentes colores para dar la sensación de que la transición entre colores es más suave.

Copyright © 2014 Depto. CCIA All rights reserved.

Efecto dithering

- **Modo de transferencia**: Con `setXferMode` podemos cambiar el modo de transferencia con el que se dibuja. Por ejemplo, podemos hacer que sólo se dibuje encima de pixels que tengan un determinado color.
- **Estilo del texto**: Podemos también especificar el tipo de fuente a utilizar y sus atributos. Lo veremos con más detalle más adelante.

Una vez establecido el tipo de pincel, podremos utilizarlo para dibujar diferentes elementos en el lienzo, utilizando métodos de la clase `Canvas`.

En el lienzo podremos también establecer algunas propiedades, como el área de recorte (`clipRect`), que en este caso no tiene porque ser rectangular (`clipPath`), o transformaciones geométricas (`translate`, `scale`, `rotate`, `skew`, o `setMatrix`). Si queremos cambiar temporalmente estas propiedades, y luego volver a dejar el lienzo como estaba originalmente, podemos utilizar los métodos `save` y `restore`.

Vamos a ver a continuación como utilizar los métodos del lienzo para dibujar distintos tipos de primitivas geométricas.

5.2.2. Primitivas geométricas

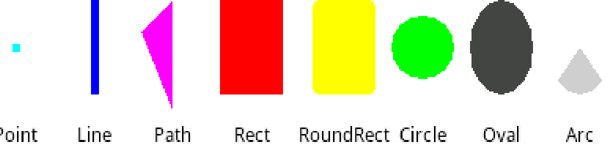
En la clase `Canvas` encontramos métodos para dibujar diferentes tipos de primitivas geométricas. Estos tipos son:

- **Puntos**: Con `drawPoint` podemos dibujar un punto en las coordenadas X, Y especificadas.
- **Líneas**: Con `drawLine` dibujamos una línea recta desde un punto de origen hasta un punto destino.
- **Polilíneas**: Podemos dibujar una polilínea mediante `drawPath`. La polilínea se especificará mediante un objeto de clase `Path`, en el que iremos añadiendo los segmentos de los que se compone. Este objeto `Path` representa un contorno, que podemos crear no sólo a partir de segmentos rectos, sino también de curvas cuadráticas y cúbicas.
- **Rectángulos**: Con `drawRect` podemos dibujar un rectángulo con los límites superior, inferior, izquierdo y derecho especificados.
- **Rectángulos con bordes redondeados**: Es también posible dibujar un rectángulo con esquinas redondeadas con `drawRoundRect`. En este caso deberemos especificar también el radio de las esquinas.
- **Círculos**: Con `drawCircle` podemos dibujar un círculo dando su centro y su radio.
- **Óvalos**: Los óvalos son un caso más general que el del círculo, y los crearemos con

Copyright © 2014 Depto. CCIA All rights reserved.

`drawOval` proporcionando el rectángulo que lo engloba.

- **Arcos**: También podemos dibujar arcos, que consisten en un segmento del contorno de un óvalo. Se crean con `drawArc`, proporcionando, además de los mismos datos que en el caso del óvalo, los ángulos que limitan el arco.
- **Todo el lienzo**: Podemos también especificar que todo el lienzo se rellene de un color determinado con `drawColor` o `drawARGB`. Esto resulta útil para limpiar el fondo antes de empezar a dibujar.



Tipos de primitivas geométricas

A continuación mostramos un ejemplo de cómo podríamos dibujar una polilínea y un rectángulo:

```
Paint paint = new Paint();
paint.setStyle(Style.FILL);
paint.setStrokeWidth(5);
paint.setColor(Color.BLUE);

Path path = new Path();
path.moveTo(50, 130);
path.lineTo(50, 60);
path.lineTo(30, 80);

canvas.drawPath(path, paint);

canvas.drawRect(new RectF(180, 20, 220, 80), paint);
```

5.2.3. Cadenas de texto

Para dibujar texto podemos utilizar el método `drawText`. De forma alternativa, se puede utilizar `drawPosText` para mostrar texto especificando una por una la posición de cada carácter, y `drawTextOnPath` para dibujar el texto a lo largo de un contorno (`Path`).

Para especificar el tipo de fuente y sus atributos, utilizaremos las propiedades del objeto `Paint`. Las propiedades que podemos especificar del texto son:

- **Fuente**: Con `setTypeface` podemos especificar la fuente, que puede ser alguna de las fuentes predefinidas (*Sans Serif*, *Serif*, *Monoespaciada*), o bien una fuente propia a partir de un fichero de fuente. También podemos especificar si el estilo de la fuente será normal, cursiva, negrita, o negrita cursiva.
- **Tamaño**: Podemos establecer el tamaño del texto con `setTextSize`.
- **Anchura**: Con `setTextScaleX` podemos modificar la anchura del texto sin alterar la altura.
- **Inclinación**: Con `setTextSkewX` podemos aplicar un efecto de desencajado al texto, pudiendo establecer la inclinación que tendrán los carácteres.

Copyright © 2014 Depto. CCIA All rights reserved.

- **Subrayado**: Con `setUnderlineText` podemos activar o desactivar el subrayado.
- **Tachado**: Con `setStrikeThruText` podemos activar o desactivar el efecto de tachado.
- **Negrita falsa**: Con `setFakeBoldText` podemos darle al texto un efecto de *negrita*, aunque la fuente no sea de este tipo.
- **Alineación**: Con `setTextAlign` podemos especificar si el texto de alinea al centro, a la derecha, o a la izquierda.
- **Subpixel**: Se renderiza a nivel de subpixel. El texto se genera a una resolución mayor que la de la pantalla donde lo vamos a mostrar, y para cada pixel real se habrán generado varios pixels. Si aplicamos *antialiasing*, a la hora de mostrar el pixel real, se determinará un nivel de gris dependiendo de cuantos pixels ficticios estén activados. Se consigue un aspecto de texto más suavizado.
- **Texto lineal**: Muestra el texto con sus dimensiones reales de forma lineal, sin ajustar los tamaños de los caracteres a la cuadrícula de pixels de la pantalla.
- **Contorno del texto**: Aunque esto no es una propiedad del texto, el objeto `Paint` también nos permite obtener el contorno (`Path`) de un texto dado, para así poder aplicar al texto los mismos efectos que a cualquier otro contorno que dibujemos.

Normal Subrayado
Normal lineal *Inclinado*
Negrita falsa Antialiasing
~~Tachado~~ Antialiasing subpixel

Efectos del texto

Con esto hemos visto como dibujar texto en pantalla, pero para poderlo ubicar de forma correcta es importante saber el tamaño en pixels del texto a mostrar. Vamos a ver ahora cómo obtener estas métricas.

Las métricas se obtendrán a partir del objeto `Paint` en el que hemos definido las propiedades de la fuente a utilizar. Mediante `getFontMetrics` podemos obtener una serie de métricas de la fuente actual, que nos dan las distancias recomendadas que debemos dejar entre diferentes líneas de texto:

- `ascent`: Distancia que asciende la fuente desde la línea de base. Para texto con espaciado sencillo es la distancia que se recomienda dejar por encima del texto. Se trata de un valor negativo.
- `descent`: Distancia que baja la fuente desde la línea de base. Para texto con espaciado sencillo es la distancia que se recomienda dejar por debajo del texto. Se trata de un valor positivo.
- `leading`: Distancia que se recomienda dejar entre dos líneas consecutivas de texto.
- `bottom`: Es la máxima distancia que puede bajar un símbolo desde la línea de base. Es un valor positivo.
- `top`: Es la máxima distancia que puede subir un símbolo desde la línea de base. Es un valor negativo.

Los anteriores valores son métricas generales de la fuente, pero muchas veces

Copyright © 2014 Depto. CCIA All rights reserved.

necesitaremos saber la anchura de una determinada cadena de texto, que ya no sólo depende de la fuente sino también del texto. Tenemos una serie de métodos con los que obtener este tipo de información:

- `measureText`: Nos da la anchura en pixels de una cadena de texto con la fuente actual.
- `breakText`: Método útil para cortar el texto de forma que no se salga de los márgenes de la pantalla. Se le proporciona la anchura máxima que puede tener la línea, y el método nos dice cuántos carácteres de la cadena proporcionada caben en dicha línea.
- `getTextWidths`: Nos da la anchura individual de cada carácter del texto proporcionado.
- `getTextBounds`: Nos devuelve un rectángulo con las dimensiones del texto, tanto anchura como altura.

5.2.4. Imágenes

Podemos también dibujar en nuestro lienzo imágenes que hayamos cargado como `Bitmap`. Esto se hará utilizando el método `drawBitmap`.

También podremos realizar transformaciones geométricas en la imagen al mostrarla en el lienzo lienzo con `drawBitmap`, e incluso podemos dibujar el *bitmap* sobre una malla poligonal con `drawBitmapMesh`

5.2.5. Drawables

También podemos dibujar objetos de tipo *drawable* en nuestro lienzo, esta vez mediante el método `draw` definido en la clase `Drawable`. Esto nos permitirá mostrar en nuestro componente cualquiera de los tipos disponibles de *drawables*, tanto definidos en XML como de forma programática.

5.2.6. Medición del componente

Al crear un nuevo componente, además de sobreescribir el método `onDraw`, es buena idea sobreescribir también el método `onMeasure`. Este método será invocado por el sistema cuando vaya a ubicarlo en el *layout*, para asignarle un tamaño. Para cada dimensión (altura y anchura), nos pasa dos parámetros:

- **Tamaño**: Tamaño en píxeles solicitado para la dimensión (altura o anchura).
- **Modo**: Puede ser `EXACTLY`, `AT_MOST`, o `UNSPECIFIED`. En el primer caso indica que el componente debe tener exactamente el tamaño solicitado, el segundo indica que como mucho puede tener ese tamaño, y el tercero nos da libertad para decidir el tamaño.

Antes de finalizar `onMeasure`, deberemos llamar obligatoriamente a `setMeasuredDimension(width, height)` proporcionando el tamaño que queramos que tenga nuestro componente. Una posible implementación sería la siguiente:

Copyright © 2014 Depto. CCIA All rights reserved.

```
@Override
protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec) {
  int widthMode = MeasureSpec.getMode(widthMeasureSpec);
  int widthSize = MeasureSpec.getSize(widthMeasureSpec);
  int heightMode = MeasureSpec.getMode(heightMeasureSpec);
  int heightSize = MeasureSpec.getSize(heightMeasureSpec);

  int width = DEFAULT_SIZE;
  int height = DEFAULT_SIZE;

  switch(widthMode) {
    case MeasureSpec.EXACTLY:
      width = widthSize;
      break;
    case MeasureSpec.AT_MOST:
      if(width > widthSize) {
        width = widthSize;
      }
      break;
  }

  switch(heightMode) {
    case MeasureSpec.EXACTLY:
      height = heightSize;
      break;
    case MeasureSpec.AT_MOST:
      if(height > heightSize) {
        height = heightSize;
      }
      break;
  }

  this.setMeasuredDimension(width, height);
}
```

Podemos ver que tenemos unas dimensiones preferidas por defecto para nuestro componente. Si nos piden unas dimensiones exactas, ponemos esas dimensiones, pero si nos piden unas dimensiones como máximo, nos quedamos con el mínimo entre nuestra dimensión preferida y la que se ha especificado como límite máximo que puede tener.

5.2.7. Atributos propios

Si creamos un nuevo tipo de vista, es muy probable que necesitemos parametrizarla de alguna forma. Por ejemplo, si queremos dibujar una gráfica que nos muestre un porcentaje, necesitaremos proporcionar un valor numérico como porcentaje a mostrar. Si vamos a crear la vista siempre de forma programática esto no es ningún problema, ya que basta con incluir en nuestra clase un método que establezca dicha propiedad.

Sin embargo, si queremos que nuestro componente se pueda añadir desde el XML, será necesario poder pasarle dicho valor como atributo. Para ello en primer lugar debemos declarar los atributos propios en un fichero `/res/values/attrs.xml`:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
        <declare-styleable name="Grafica">
                <attr name="percentage" format="integer"/>
        </declare-styleable>
</resources>
```


Copyright © 2014 Depto. CCIA All rights reserved.

En el XML donde definimos el *layout*, podemos especificar nuestro componente utilizando como nombre de la etiqueta el nombre completo (incluyendo el paquete) de la clase donde hemos definido la vista:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:app=
"http://schemas.android.com/apk/res/es.ua.jtech.grafica"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<es.ua.jtech.grafica.GraficaView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    app:percentage="60"
 />
</LinearLayout>
```

Podemos fijarnos en que para declarar el atributo propio hemos tenido que especificar el espacio de nombres en el que se encuentra. En dicho espacio de nombres deberemos especificar el paquete que hemos declarado en `AndroidManifest.xml` para la aplicación (en nuestro caso `es.ua.jtech.grafica`).

```
public GraficaView(Context context) {
  super(context);
}

public GraficaView(Context context, AttributeSet attrs, int defStyle) {
  super(context, attrs, defStyle);
  this.init(attrs);
}

public GraficaView(Context context, AttributeSet attrs) {
  super(context, attrs);
  this.init(attrs);
}

private void init(AttributeSet attrs) {
  TypedArray ta = this.getContext().obtainStyledAttributes(attrs,
                                            R.styleable.Grafica);
  this.percentage = ta.getInt(R.styleable.Grafica_percentage, 0);
}
```

En primer lugar podemos ver que debemos definir todos los posibles constructores de las vistas, ya que cuando se cree desde el XML se invocará uno de los que reciben la lista de atributos especificados. Una vez recibamos dicha lista de atributos, deberemos obtener el conjunto de atributos propios mediante `obtainStyledAttributes`, y posteriormente obtener los valores de cada atributo concreto dentro de dicho conjunto.

5.2.8. Actualización del contenido

Es posible que en un momento dado cambien los datos a mostrar y necesitemos actualizar el contenido que nuestro componente está dibujando en pantalla. Podemos forzar que se vuelva a dibujar llamando al método `invalidate` de nuestra vista (`View`).

Copyright © 2014 Depto. CCIA All rights reserved.

Esto podemos utilizarlo también para crear animaciones. De hecho, para crear una animación simplemente deberemos cambiar el contenido del lienzo conforme pasa el tiempo. Una forma de hacer esto es simplemente cambiar mediante un hilo o temporizadores propiedades de los objetos de la escena (como sus posiciones), y forzar a que se vuelva a redibujar el contenido del lienzo cada cierto tiempo.

Sin embargo, si necesitamos contar con una elevada tasa de refresco, como por ejemplo en el caso de un videojuego, será recomendable utilizar una vista de tipo `SurfaceView` como veremos a continuación.

5.3. Gráficos 3D

Para mostrar gráficos 3D en Android contamos con OpenGL ES, un subconjunto de la librería gráfica OpenGL destinado a dispositivos móviles.

Hasta ahora hemos visto que para mostrar gráficos propios podíamos usar un componente que heredase de `View`. Estos componentes funcionan bien si no necesitamos realizar repintados continuos o mostrar gráficos 3D.

Sin embargo, en el caso de tener una aplicación con una gran carga gráfica, como puede ser un videojuego o una aplicación que muestre gráficos 3D, en lugar de `View` deberemos utilizar `SurfaceView`. Esta última clase nos proporciona una superficie en la que podemos dibujar desde un hilo en segundo plano, lo cual libera al hilo principal de la aplicación de la carga gráfica.

Vamos a ver en primer lugar cómo utilizar `SurfaceView`, y las diferencias existentes con `View`.

Para crear una vista con `SurfaceView` tendremos que crear una nueva subclase de dicha clase (en lugar de `View`). Pero en este caso no bastará con definir el método `onDraw`, ahora deberemos crearnos un hilo independiente y proporcionarle la superficie en la que dibujar (`SurfaceHolder`). Además, en nuestra subclase de `SurfaceView` también implementaremos la interfaz `SurfaceHolder.Callback` que nos permitirá estar al tanto de cuando la superficie se crea, cambia, o se destruye.

Cuando la superficie sea creada pondremos en marcha nuestro hilo de dibujado, y lo pararemos cuando la superficie sea destruida. A continuación mostramos un ejemplo de dicha clase:

```
public class VistaSurface extends SurfaceView
              implements SurfaceHolder.Callback {
    HiloDibujo hilo = null;

    public VistaSurface(Context context) {
        super(context);

        SurfaceHolder holder = this.getHolder();
        holder.addCallback(this);
    }
```


Copyright © 2014 Depto. CCIA All rights reserved.

```
    public void surfaceChanged(SurfaceHolder holder, int format,
                               int width, int height) {
        // La superficie ha cambiado (formato o dimensiones)
    }

    public void surfaceCreated(SurfaceHolder holder) {
        hilo = new HiloDibujo(holder, this);
        hilo.start();
    }

    public void surfaceDestroyed(SurfaceHolder holder) {

        hilo.detener();
        try {
            hilo.join();
        } catch (InterruptedException e) { }
    }
}
```

Como vemos, la clase `SurfaceView` simplemente se encarga de obtener la superficie y poner en marcha o parar el hilo de dibujado. En este caso la acción estará realmente en el hilo, que es donde especificaremos la forma en la que se debe dibujar el componente. Vamos a ver a continuación cómo podríamos implementar dicho hilo:

```
class HiloDibujo extends Thread {
    SurfaceHolder holder;
    VistaSurface vista;
    boolean continuar = true;

    public HiloDibujo(SurfaceHolder holder, VistaSurface vista) {
        this.holder = holder;
        this.vista = vista;
        continuar = true;
    }

    public void detener() {
        continuar = false;
    }

    @Override
    public void run() {
        while (continuar) {
            Canvas c = null;
            try {
                c = holder.lockCanvas(null);
                synchronized (holder) {
                    // Dibujar aqui los graficos
                    c.drawColor(Color.BLUE);
                }
            } finally {
                if (c != null) {
                    holder.unlockCanvasAndPost(c);
                }
            }
        }
    }
}
```

Podemos ver que en el bucle principal de nuestro hilo obtenermos el lienzo (`Canvas`) a partir de la superficie (`SurfaceHolder`) mediante el método `lockCanvas`. Esto deja el lienzo bloqueado para nuestro uso, por ese motivo es importante asegurarnos de que

Copyright © 2014 Depto. CCIA All rights reserved.

siempre se desbloquee. Para tal fin hemos puesto `unlockCanvasAndPost` dentro del bloque `finally`. Además debemos siempre dibujar de forma sincronizada con el objeto `SurfaceHolder`, para así evitar problemas de concurrencia en el acceso a su lienzo.

Para aplicaciones como videojuegos 2D sencillo un código como el anterior puede ser suficiente (la clase `View` sería demasiado lenta para un videojuego). Sin embargo, lo realmente interesante es utilizar `SurfaceView` junto a OpenGL, para así poder mostrar gráficos 3D, o escalados, rotaciones y otras transformaciones sobre superficies 2D de forma eficiente.

El estudio de la librería OpenGL queda fuera del ámbito de este curso. A continuación veremos un ejemplo de cómo utilizar OpenGL (concretamente OpenGL ES) vinculado a nuestra `SurfaceView`.

Realmente la implementación de nuestra clase que hereda de `SurfaceView` no cambiará, simplemente modificaremos nuestro hilo, que es quien realmente realiza el dibujado. Toda la inicialización de OpenGL deberá realizarse dentro de nuestro hilo (en el método `run`), ya que sólo se puede acceder a las operaciones de dicha librería desde el mismo hilo en el que se inicializó. En caso de que intentásemos acceder desde otro hilo obtendríamos un error indicando que no existe ningún contexto activo de OpenGL.

En este caso nuestro hilo podría contener el siguiente código:

```
public void run() {
    initEGL();
    initGL();

    Triangulo3D triangulo = new Triangulo3D();
    float angulo = 0.0f;

    while(continuar) {
        gl.glClear(GL10.GL_COLOR_BUFFER_BIT |
                   GL10.GL_DEPTH_BUFFER_BIT);

        // Dibujar gráficos aquí
        gl.glMatrixMode(GL10.GL_MODELVIEW);
        gl.glLoadIdentity();
        gl.glTranslatef(0, 0, -5.0f);
        gl.glRotatef(angulo, 0, 1, 0);

        gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
        triangulo.dibujar(gl);

        egl.eglSwapBuffers(display, surface);
        angulo += 1.0f;
    }
}
```

En primer lugar debemos inicializar la interfaz EGL, que hace de vínculo entre la plataforma nativa y la librería OpenGL:

```
EGL10 egl;
GL10 gl;
EGLDisplay display;
EGLSurface surface;
EGLContext contexto;
```


Copyright © 2014 Depto. CCIA All rights reserved.

```
EGLConfig config;

private void initEGL() {
    egl = (EGL10)EGLContext.getEGL();
    display = egl.eglGetDisplay(EGL10.EGL_DEFAULT_DISPLAY);

    int [] version = new int[2];
    egl.eglInitialize(display, version);

    int [] atributos = new int[] {
        EGL10.EGL_RED_SIZE, 5,
        EGL10.EGL_GREEN_SIZE, 6,
        EGL10.EGL_BLUE_SIZE, 5,
        EGL10.EGL_DEPTH_SIZE, 16,
        EGL10.EGL_NONE
    };

    EGLConfig [] configs = new EGLConfig[1];
    int [] numConfigs = new int[1];
    egl.eglChooseConfig(display, atributos, configs,
                        1, numConfigs);

    config = configs[0];
    surface = egl.eglCreateWindowSurface(display,
                        config, holder, null);
    contexto = egl.eglCreateContext(display, config,
                        EGL10.EGL_NO_CONTEXT, null);
    egl.eglMakeCurrent(display, surface, surface, contexto);

    gl = (GL10)contexto.getGL();
}
```

A continuación debemos proceder a la inicialización de la interfaz de la librería OpenGL:

```
private void initGL() {
    int width = vista.getWidth();
    int height = vista.getHeight();
    gl.glViewport(0, 0, width, height);
    gl.glMatrixMode(GL10.GL_PROJECTION);
    gl.glLoadIdentity();

    float aspecto = (float)width/height;
    GLU.gluPerspective(gl, 45.0f, aspecto, 1.0f, 30.0f);
    gl.glClearColor(0.5f, 0.5f, 0.5f, 1);
}
```

Una vez hecho esto, ya sólo nos queda ver cómo dibujar una malla 3D. Vamos a ver como ejemplo el dibujo de un triángulo:

```
public class Triangulo3D {

    FloatBuffer buffer;

    float[] vertices = {
        -1f, -1f, 0f,
        1f, -1f, 0f,
        0f, 1f, 0f };


    public Triangulo3D() {
        ByteBuffer bufferTemporal = ByteBuffer
                   .allocateDirect(vertices.length*4);
        bufferTemporal.order(ByteOrder.nativeOrder());
        buffer = bufferTemporal.asFloatBuffer();
```


Copyright © 2014 Depto. CCIA All rights reserved.

```
        buffer.put(vertices);
        buffer.position(0);
    }

    public void dibujar(GL10 gl) {
        gl.glVertexPointer(3, GL10.GL_FLOAT, 0, buffer);
        gl.glDrawArrays(GL10.GL_TRIANGLES, 0, 3);
    }
}
```

Para finalizar, es importante que cuando la superficie se destruya se haga una limpieza de los recursos utilizados por OpenGL:

```
private void cleanupGL() {
    egl.eglMakeCurrent(display, EGL10.EGL_NO_SURFACE,
            EGL10.EGL_NO_SURFACE, EGL10.EGL_NO_CONTEXT);
    egl.eglDestroySurface(display, surface);
    egl.eglDestroyContext(display, contexto);
    egl.eglTerminate(display);
}
```

Podemos llamar a este método cuando el hilo se detenga (debemos asegurarnos que se haya detenido llamando a `join` previamente).

A partir de Android 1.5 se incluye la clase `GLSurfaceView`, que ya incluye la inicialización del contexto GL y nos evita tener que hacer esto manualmente. Esto simplificará bastante el uso de la librería. Vamos a ver a continuación un ejemplo de como trabajar con dicha clase.

En este caso ya no será necesario crear una subclase de `GLSurfaceView`, ya que la inicialización y gestión del hilo de OpenGL siempre es igual. Lo único que nos interesará cambiar es lo que se muestra en la escena. Para ello deberemos crear una subclase de `GLSurfaceViewRenderer` que nos obliga a definir los siguientes métodos:

```
public class MiRenderer implements GLSurfaceView.Renderer {

    Triangulo3D triangulo;
    float angulo;

    public MiRenderer() {
        triangulo = new Triangulo3D();
        angulo = 0;
    }

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

    public void onSurfaceChanged(GL10 gl, int w, int h) {
        // Al cambiar el tamaño cambia la proyección
        float aspecto = (float)w/h;
            gl.glViewport(0, 0, w, h);

        gl.glMatrixMode(GL10.GL_PROJECTION);
        gl.glLoadIdentity();
        GLU.gluPerspective(gl, 45.0f, aspecto, 1.0f, 30.0f);
    }

    public void onDrawFrame(GL10 gl) {
        gl.glClearColor(0.5f, 0.5f, 0.5f, 1.0f);
```


Copyright © 2014 Depto. CCIA All rights reserved.

```
        gl.glClear(GL10.GL_COLOR_BUFFER_BIT |
                   GL10.GL_DEPTH_BUFFER_BIT);

        // Dibujar gráficos aquí
        gl.glMatrixMode(GL10.GL_MODELVIEW);
        gl.glLoadIdentity();
        gl.glTranslatef(0, 0, -5.0f);
        gl.glRotatef(angulo, 0, 1, 0);

        gl.glEnableClientState(GL10.GL_VERTEX_ARRAY);
        triangulo.dibujar(gl);

        angulo += 1.0f;
    }
}
```

Podemos observar que será el método `onDrawFrame` en el que deberemos escribir el código para mostrar los gráficos. Con hacer esto será suficiente, y no tendremos que encargarnos de crear el hilo ni de inicializar ni destruir el contexto.

Para mostrar estos gráficos en la vista deberemos proporcionar nuestro *renderer* al objeto `GLSurfaceView`:

```
vista = new GLSurfaceView(this);
vista.setRenderer(new MiRenderer());
setContentView(vista);
```

Por último, será importante transmitir los eventos `onPause` y `onResume` de nuestra actividad a la vista de OpenGL, para así liberar a la aplicación de la carga gráfica cuando permanezca en segundo plano. El código completo de la actividad quedaría como se muestra a continuación:

```
public class MiActividad extends Activity {
    GLSurfaceView vista;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        vista = new GLSurfaceView(this);
        vista.setRenderer(new MiRenderer());
        setContentView(vista);
    }

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

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

Para utilizar esta vista deberemos tener al menos unos conocimientos básicos de OpenGL. Si lo que queremos es iniciarnos en el desarrollo de videojuegos para Android, contamos con diferentes librerías y *frameworks* que nos van a facilitar bastante el trabajo.

Copyright © 2014 Depto. CCIA All rights reserved.

Destacamos las siguientes:

- *libgdx*: Es la más rápida y flexible, aunque requiere tener algunos conocimientos de OpenGL. http://www.badlogicgames.com
- *AndEngine*: Es muy sencilla de utilizar. Proporciona gran cantidad de facilidades para crear videojuegos 2D. http://www.andengine.org/

Copyright © 2014 Depto. CCIA All rights reserved.

6. Gráficos avanzados - Ejercicios

6.1. Personalización del aspecto

Crea una nueva aplicación de Android llamada `Drawables` y haz que muestre un `TextView` que ocupe toda la pantalla y que tenga el siguiente aspecto:

- Un marco de color azul con esquinas redondeadas.
- El marco debe tener un grosor de 2 *density pixels*.
- Se dejará un margen de 10 *density pixels* entre el marco y el texto que contiene
- Se dejará un margen de 5 *density pixels* entre los bordes de la pantalla y el marco. ¿Encontramos algún atributo que haga esto en la definición del *drawable*? ¿Y en el *layout*?
- Un relleno de gradiente lineal para el marco, desde gris claro hasta blanco
- El texto será de color negro.

Ayúdate de la documentación oficial de Android para crear el recurso:

http://developer.android.com/guide/topics/resources/drawable-resource.html#Shape

6.2. Personalización de botones

Vamos a añadir un botón en la parte inferior de la pantalla para la aplicación anterior, que llenará todo el ancho. Se proporcionan en las plantillas de la sesión tres imágenes, una para cada uno de los estados del botón. Se pide:

a) Dar inicialmente al botón como aspecto la imagen correspondiente al estado sin pulsar.

b) Crear un *state list drawable* para que el botón cambie su aspecto para cada estado.

c) Convertir las imágenes a *nine-patch* y utilizar esta nueva versión en la aplicación.

Nos podemos ayudar de la documentación sobre *state list drawable*:

http://developer.android.com/guide/topics/resources/drawable-resource.html#StateList

6.3. Animaciones por fotogramas

Vamos a añadir a la pantalla de nuestra aplicación una vista de tipo `ProgressBar` que muestre que se está realizando un progreso de forma indeterminada. Se pide:

a) Ejecutar la aplicación y comprobar como aparece un círculo girando continuamente como indicativo de que se está realizando un progreso. Hacer que el indicador de progreso aparezca centrado en la horizontal (mirar el atributo `layout_gravity`).

Copyright © 2014 Depto. CCIA All rights reserved.

b) Vamos ahora a personalizar la animación que figura en el progreso. En las plantillas de la sesión tenemos cuatro fotogramas de un armadillo caminando. Crea un *animation-list drawable* a partir de dichos fotogramas, que se reproduzca de forma continuada cambiando de fotograma cada 200ms.

c) Reemplaza la animación por defecto del progreso por la animación que hemos definido nosotros. Pista: se debe especificar como *drawable*, y el progreso continuo indeterminado (sin dar un porcentaje concreto) que queremos mostrar, se conoce en la API de Android como *indeterminate progress*. Busca entre los atributos de `ProgressBar` el que consideres adecuado.

Nos podemos ayudar de la documentación sobre *animation list drawable*:

http://developer.android.com/guide/topics/resources/animation-resource.html#Frame

6.4. Niveles

En este ejercicio vamos a añadir una barra de de selección de nivel (`SeekBar`), que podría sernos útil por ejemplo como control de volúmen, o para desplazarnos en la reproducción de vídeo o audio. Se pide:

a) Añadir un `SeekBar` que ocupe todo el ancho de la pantalla a nuestra aplicación y comprobar que se muestra correctamente y que al pulsar sobre ella podemos mover el cursor. Por defecto, el nivel de la barra va de 0 a 10000.

b) Vamos a personalizarla para que el fondo de la barra cambie de color según el nivel seleccionado. Los colores que debe mostrar son:

- **Verde** (`#FF00FF00`): Niveles de 0 a 2500.
- **Amarillo** (`#FFFFFF00`): Niveles de 2501 a 5000.
- **Naranja** (`#FFFF8800`): Niveles de 5001 a 7500.
- **Rojo** (`#FFFF0000`): Niveles de 7501 a 10000.

Crea un *drawable* de tipo rectángulo con esquinas redondeadas para cada estado. Introdúcelos en un *level list drawable*, y establécelo como `progressDrawable` en la barra.

Nos podemos ayudar de la documentación sobre *level list drawable*:

http://developer.android.com/guide/topics/resources/drawable-resource.html#LevelList

6.5. Creación de componentes propios

Vamos a crear un nuevo componente que muestre un gráfico de tipo tarta para indicar un determinado porcentaje. Dentro del círculo, aparecerá en rojo el sector correspondiente al porcentaje indicado, y en azul el resto. Para ello crearemos una nueva aplicación Android de nombre `Grafica`. Se pide:

Copyright © 2014 Depto. CCIA All rights reserved.

a) Crear una vista con la gráfica que muestre por defecto un porcentaje del 25%. Mostrarla en la pantalla de forma programática.

b) Sobreescribir el método `onMeasure`, de forma que tenga un tamaño preferido de 50x50. Al ejecutarlo, ¿ha cambiado algo? ¿Por qué? (Pista: piensa en los parámetros de *layout* que se deben estar aplicando por defecto al introducir el componente de forma programática).

c) Declarar la vista en el XML, para incluirla en el *layout* creado por defecto para nuestra aplicación.

Atención

En este punto es necesario haber definido todos los constructores de la superclase `View`, ya que al inicializarlo desde XML podrá utilizar cualquiera de ellos según la información proporcionada.

d) Añadir un atributo `percentage` propio a la vista, e indicar dicho porcentaje en el XML.

e) Añade varias gráficas al *layout*, con diferentes porcentajes cada una de ellas.

f) Hacer que la gráfica se muestre con un gradiente radial, con un color más oscuro en el centro y más claro en los bordes.

Nota

El gradiente se establece con el método `setShader`, que toma como parámetro un objeto de tipo `Shader`. Busca entre sus subclases para encontrar el *shader* apropiado.

g) Añadir un borde a la figura del mismo color que el central y con grosor 2.

Ayuda

Primero deberemos dibujar el relleno, y luego el borde. Para ello utilizaremos `setStyle`.

6.6. Texto y métricas

Continuando con el código desarrollado en el ejercicio anterior, vamos ahora a mostrar también con texto el porcentaje que representa la gráfica. Este texto debe aparecer centrado horizontalmente y en la parte inferior, sin que se pueda llegar a cortar ninguna letra por el borde inferior de la vista. Añadir *antialiasing* al texto.

6.7. Gráficos 3D

En las plantillas de la sesión tenemos una aplicación `Graficos` en la que podemos ver un ejemplo completo de cómo utilizar `SurfaceView` tanto para gráficos 2D con el `Canvas`

Copyright © 2014 Depto. CCIA All rights reserved.

como para gráficos 3D con OpenGL, y también de cómo utilizar `GLSurfaceView`.

a) Si ejecutamos la aplicación veremos un triángulo rotando alrededor del eje Y. Observar el código fuente, y modificarlo para que el triángulo rote alrededor del eje X, en lugar de Y.

b) También podemos ver que hemos creado, además de la clase `Triangulo3D`, la clase `Cubo3D`. Modificar el código para que en lugar de mostrar el triángulo se muestre el cubo.

Copyright © 2014 Depto. CCIA All rights reserved.

7. Sensores y eventos

Una diferencia importante entre los dispositivos móviles y los ordenadores es la forma en la que el usuario interactua con las aplicaciones. Si bien en un ordenador la forma que tiene el usuario de introducir información es fundamentalmente con ratón y teclado, en un dispositivo móvil, debido a su reducido tamaño y al uso al que están enfocados, estos dispositivos de entrada no resultan adecuados.

Algunos dispositivos, en gran parte PDAs, incorporan un pequeño teclado y un puntero, para así presentar una interfaz de entrada similar al teclado y ratón de un ordenador. Sin embargo, este tipo de interfaz resulta poco usable para el uso común de estos dispositivos. Por un lado se debe a que el teclado resulta demasiado pequeño, lo cual no permite escribir cómodamente, y además ocupa un espacio importante del dispositivo, haciendo que éste sea más grande y dejando menos espacio para la pantalla. Por otro lado, el puntero nos obliga a tener que utilizar las dos manos para manejar el dispositivo, lo cual resulta también poco adecuado.

Dado que los dispositivos de entrada tradicionales no resultan apropiados para los dispositivos móviles, en estos dispositivos se han ido popularizando una serie de nuevos dispositivos de entrada que no encontramos en los ordenadores. En una gran cantidad de dispositivos encontramos sensores como:

- **Pantalla tácil:** En lugar de tener que utilizar un puntero, podemos manejar el dispositivo directamente con los dedos. La interfaz deberá crearse de forma adecuada a este tipo de entrada. Un dedo no tiene la precisión de un puntero o un ratón, por lo que los elementos de la pantalla deben ser lo suficientemente grandes para evitar que se pueda confundir el elemento que quería seleccionar el usuario.
- **Acelerómetro:** Mide la aceleración a la que se somete al dispositivo en diferentes ejes. Comprobando los ejes en los que se ejerce la acelaración de la fuerza gravitatoria podemos conocer la orientación del dispositivo.
- **Giroscopio:** Mide los cambios de orientación del dispositivo. Es capaz de reconocer movimientos que no reconoce el acelerómetro, como por ejemplo los giros en el eje Y (vertical).
- **Brújula:** Mide la orientación del dispositivo a partir del campo magnético. Normalmente para obtener la orientación del dispositivo de forma precisa se suelen combinar las lecturas de dos sensores, como la brújula o el giroscopio, y el acelerómetro.
- **GPS:** Obtiene la geolocalización del dispositivo (latitud y longitud) mediante la triangulación con los satélites disponibles. Si no disponemos de GPS o no tenemos visibilidad de ningún satélite, los dispositivos también pueden geolocalizarse mediante su red 3G o WiFi.
- **Micrófono:** Podemos también controlar el dispositivo mediante comandos de voz, o introducir texto mediante reconocimiento del habla.

Copyright © 2014 Depto. CCIA All rights reserved.

Vamos a continuación a estudiar cómo acceder con Android a los sensores más comunes.

7.1. Pantalla táctil

En la mayoría de aplicaciones principalmente la entrada se realizará mediante la pantalla táctil, bien utilizando algún puntero o directamente con los dedos. En ambos casos deberemos reconocer los eventos de pulsación sobre la pantalla.

Antes de comenzar a ver cómo realizar la gestión de estos eventos, debemos definir el concepto de **gesto**. Un gesto es un movimiento que hace el usuario en la pantalla. El gesto comienza cuando el dedo hace contacto con la pantalla, se prolonga mientras el dedo permanece en contacto con ella, pudiendo moverse a través de la misma, y finaliza cuando levantamos el dedo de la pantalla.

Muchos de los componentes nativos de la interfaz ya implementan toda la interacción con el usuario, incluyendo la interacción mediante la pantalla táctil, por lo que en esos casos no tendremos que preocuparnos de dicho tipo de eventos. Por ejemplo, si ponemos un `CheckBox`, este componente se encargará de que cuando pulsemos sobre él vaya cambiando su estado entre seleccionado y no seleccionado, y simplemente tendremos que consultar en qué estado se encuentra.

Sin embargo, si queremos crear un componente propio a bajo nivel, deberemos tratar los eventos de la pantalla tácil. Para hacer esto deberemos capturar el evento `OnTouch`, mediante un objeto que implemente la interfaz `OnTouchListener`.

De forma alternativa, si estamos creando un componente propio heredando de la clase `View`, podemos capturar el evento simplemente sobrescribiendo el método `onTouchEvent`:

```
public class MiComponente extends View
    ...
    @Override
    public boolean onTouchEvent(MotionEvent event) {
        // Procesar evento
        return true;
    }
    ...
}
```

El método devolverá `true` si ha consumido el evento, y por lo tanto no es necesario propagarlo a otros componentes, o `false` en caso contrario. En este último caso el evento pasará al padre del componente en el que nos encontramos, y será responsabilidad suya procesarlo. Ya no recibiremos el resto de eventos del gesto, todos ellos serán enviados directamente al componente padre que se haya hecho cargo.

Del evento de movimiento recibido (`MotionEvent`) destacamos la siguiente información:

- **Acción realizada:** Indica si el evento se ha debido a que el dedo se ha puesto en la pantalla (`ACTION_DOWN`), se ha movido (`ACTION_MOVE`), o se ha retirado de la pantalla (`ACTION_UP`). También existe la acción `ACTION_CANCEL` que se produce cuando se

Copyright © 2014 Depto. CCIA All rights reserved.

cancela el gesto que está haciendo el usuario. Se trata al igual que `ACTION_UP` de la finalización de un gesto, pero en este caso no deberemos ejecutar la acción asociada a la terminación correcta del gesto. Esto ocurre por ejemplo cuando un componente padre se apodera de la gestión de los eventos de la pantalla táctil.
- **Coordenadas:** Posición del componente en la que se ha tocado o a la que nos hemos desplazado. Podemos obtener esta información con los métodos `getX` y `getY`.

Con esta información podemos por ejemplo mover un objeto a la posición a la que desplacemos el dedo:

```
@Override
public boolean onTouchEvent(MotionEvent event) {
    if(event.getAction() == MotionEvent.ACTION_MOVE) {
        x = event.getX();
        y = event.getY();

        this.invalidate();
    }
    return true;
}
```

Nota

Después de cambiar la posición en la que se dibujará un gráfico es necesario llamar a `invalidate` para indicar que el contenido del componente ya no es válido y debe ser redibujado (llamando al método `onDraw`).

7.1.1. Dispositivos multitouch

Hemos visto como tratar los eventos de la pantalla táctil en el caso sencillo de que tengamos un dispositivo que soporte sólo una pulsación simultánea. Sin embargo, muchos dispositivos son *multitouch*, es decir, en un momento dado podemos tener varios puntos de contacto simultáneos, pudiendo realizar varios gestos en paralelo.

En este caso el tratamiento de este tipo de eventos es más complejo, y deberemos llevar cuidado cuando desarrollemos para este tipo de dispositivos, ya que si no tenemos en cuenta que puede haber más de una pulsación, podríamos tener fallos en la gestión de eventos de la pantalla tácil.

Vamos a ver en primer lugar cómo acceder a la información de los eventos *multitouch* a bajo nivel, con lo que podremos saber en cada momento las coordenadas de cada pulsación y las acciones que ocurren en ellas.

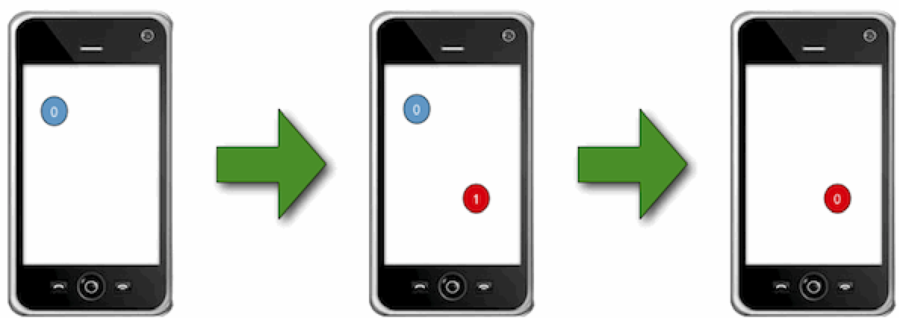
La capacidad *multitouch* se implementa en Android mediante la inclusión de múltiples punteros en la clase `MotionEvent`. Podemos saber cuántos punteros hay simultáneamente en pantalla llamando al método `getPointerCount` de dicha clase.

Cada puntero tiene un índice y un identificador. Si actualmente hay varios puntos de contacto en pantalla, el objeto `MotionEvent` contendrá una lista con varios punteros, y cada uno de ellos estará en un índice de esta lista. El índice irá desde 0 hasta el número de

Copyright © 2014 Depto. CCIA All rights reserved.

punteros menos 1. Para obtener por ejemplo la posición X de un puntero determinado, llamaremos a `getX(indice)`, indicando el índice del puntero que nos interesa. Si llamamos a `getX()` sin especificar ningún índice, como hacíamos en el caso anterior, obtendremos la posición X del primer puntero (índice 0).

Sin embargo, si uno de los punteros desaparece, debido a la finalización del gesto, es posible que los índices cambien. Por ejemplo, si el puntero con índice 0 se levanta de la pantalla, el puntero que tenía índice 1 pasará a tener índice 0. Esto nos complica el poder realizar el seguimiento de los eventos de un mismo gesto, ya que el índice asociado a su puntero puede cambiar en sucesivas llamadas a `onTouchEvent`. Por este motivo cada puntero tiene además asignado un identificador que permanecerá invariante y que nos permitirá realizar dicho seguimiento.



Eventos multitouch

El identificador nos permite hacer el seguimiento, pero para obtener la información del puntero necesitamos conocer el índice en el que está actualmente. Para ello tenemos el método `findPointerIndex(id)` que nos devuelve el índice en el que se encuentra un puntero dado su identificador. De la misma forma, para conocer por primera vez el identificador de un determinado puntero, podemos utilizar `getPointerId(indice)` que nos devuelve el identificador del puntero que se encuentra en el índice indicado.

Además se introducen dos nuevos tipos de acciones:

- `ACTION_POINTER_DOWN`: Se produce cuando entra un nuevo puntero en la pantalla, habiendo ya uno previamente pulsado. Cuando entra un puntero sin haber ninguno pulsado se produce la acción `ACTION_DOWN`.
- `ACTION_POINTER_UP`: Se produce cuando se levanta un puntero de la pantalla, pero sigue quedan alguno pulsado. Cuando se levante el último de ellos se producirá la acción `ACTION_UP`.

Además, a la acción se le adjunta el índice del puntero para el que se está ejecutando el evento. Para separar el código de la acción y el índice del puntero debemos utilizar las máscaras definidas como constantes de `MotionEvent`:

Código de la acción	`event.getAction & MotionEvent.ACTION_MASK`
Índice del puntero	`(event.getAction() &`

Copyright © 2014 Depto. CCIA All rights reserved.

	MotionEvent.ACTION_POINTER_INDEX_MASK) >> MotionEvent.ACTION_POINTER_INDEX_SHIFT

Para hacer el seguimiento del primer puntero que entre en un dispositivo *multitouch* podríamos utilizar el siguiente código:

```
private int idPunteroActivo = -1;

@Override
public boolean onTouchEvent(MotionEvent event) {
    final int accion = event.getAction() & MotionEvent.ACTION_MASK;
    final int indice = (event.getAction() &
                MotionEvent.ACTION_POINTER_INDEX_MASK)
                >> MotionEvent.ACTION_POINTER_INDEX_SHIFT;

    switch (accion) {
    case MotionEvent.ACTION_DOWN:
        // Guardamos como puntero activo el que se pulsa
        // sin haber previamente otro pulsado
        idPunteroActivo = event.getPointerId(0);

        x = event.getX();
        y = event.getY();
        ...
        break;

    case MotionEvent.ACTION_MOVE:
        // Obtenemos la posición del puntero activo
        indice = event.findPointerIndex(idPunteroActivo);

        x = event.getX(indice);
        y = event.getY(indice);
        ...
        break;

    case MotionEvent.ACTION_UP:
        // Ya no quedan más punteros en pantalla
        idPunteroActivo = -1;
        break;

    case MotionEvent.ACTION_CANCEL:
        // Se cancelan los eventos de la pantalla táctil
        // Eliminamos el puntero activo
        idPunteroActivo = -1;
        break;

    case MotionEvent.ACTION_POINTER_UP:
        // Comprobamos si el puntero que se ha levantado
        // era el puntero activo
        int idPuntero = event.getPointerId(indice);
        if (idPuntero == idPunteroActivo) {
            // Seleccionamos el siguiente puntero como activo
            // Si el índice del puntero desaparecido era el 0,
            // el nuevo puntero activo será el de índice 1.
            // Si no, tomaremos como activo el de índice 0.
            int indiceNuevoPunteroActivo = indice == 0 ? 1 : 0;

            x = event.getX(indiceNuevoPunteroActivo);
            y = event.getY(indiceNuevoPunteroActivo);

            idPunteroActivo = event
                       .getPointerId(indiceNuevoPunteroActivo);
        }
        break;
```


Copyright © 2014 Depto. CCIA All rights reserved.

```
    }

    return true;
}
```

Como vemos, el tratamiento de múltiples punteros simultáneos puede resultar bastante complejo. Por este motivo, se nos proporciona también una API de alto nivel para reconocimiento de gestos, que nos permitirá simplificar esta tarea.

7.1.2. Reconocimiento de gestos

Podemos utilizar una serie de filtros que consumen eventos de tipo `MotionEvent` y producen eventos de alto nivel notificándonos que se ha reconocido un determinado gesto, y liberándonos así de tener que programar a bajo nivel el reconocimiento de los mismos. Estos objetos se denominan detectores de gestos.

Aunque encontramos diferentes detectores de gestos ya predefinidos, éstos pueden servirnos como patrón para crear nuestros propios detectores. Todos los detectores ofrecen un método `onTouchEvent` como el visto en el anterior punto. Cuando recibamos un evento de la pantalla, se lo pasaremos al detector de gestos mediante este método para que así pueda procesar la información. Al llamar a dicho método nos devolverá `true` mientras esté reconociendo un gesto, y `false` en otro caso.

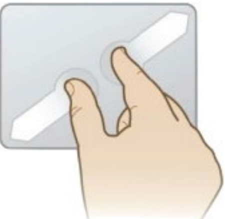
En el detector de gestos podremos registrar además una serie de listeners a los que nos avisará cuando detecte un determinado gesto, como por ejemplo el gesto de *pinza* con los dos dedos, para escalar imágenes.

Podemos encontrar tanto gestos de un sólo puntero como de múltiples. Con la clase `GestureDetector` podemos detectar varios gestos simples de un sólo puntero:

- `onSingleTapUp`: Se produce al dar un *toque* a la pantalla, es decir, pulsar y levantar el dedo. El evento se produce tras levantar el dedo.
- `onDoubleTap`: Se produce cuando se da un *doble toque* a la pantalla. Se pulsa y se suelta dos veces seguidas.
- `onSingleTapConfirmed`: Se produce después de dar un *toque* a la pantalla, y cuando se confirma que no le sucede un segundo toque.
- `onLongPress`: Se produce cuando se mantiene pulsado el dedo en la pantalla durante un tiempo largo.
- `onScroll`: Se produce cuando se arrastra el dedo para realizar *scroll*. Nos proporciona la distancia que hemos arrastrado en cada eje.
- `onFling`: Se produce cuando se produce un *lanzamiento*. Esto consiste en pulsar, arrastrar, y soltar. Nos proporciona la velocidad (en píxels) a la que se ha realizado en lanzamiento.

Además, a partir de Android 2.2 tenemos el detector `ScaleGestureDetector`, que reconoce el gesto *pinza* realizado con dos dedos, y nos proporciona la escala correspondiente al gesto.

Copyright © 2014 Depto. CCIA All rights reserved.



Gesto de pinza

A continuación mostramos como podemos utilizar el detector de gestos básico para reconocer el doble *tap*:

```
GestureDetector detectorGestos;

public ViewGestos(Context context) {
    super(context);

    ListenerGestos lg = new ListenerGestos();
    detectorGestos = new GestureDetector(lg);
    detectorGestos.setOnDoubleTapListener(lg);
}

@Override
public boolean onTouchEvent(MotionEvent event) {
    return detectorGestos.onTouchEvent(event);
}

class ListenerGestos extends
              GestureDetector.SimpleOnGestureListener {
    @Override
    public boolean onDown(MotionEvent e) {
        return true;
    }

    @Override
    public boolean onDoubleTap(MotionEvent e) {
        // Tratar el evento
        return true;
    }
}
```

Es importante definir el evento `onDown` en el detector, ya que si no devolvemos `true` en dicho método se cancelará el procesamiento del gesto. Esto podemos utilizarlo para que sólo detecte gestos que se inician en una determinada posición de la pantalla. Por ejemplo, que se arrastre una caja al pulsar sobre ella y hacer *scroll*.

7.2. Orientación y aceleración

Los sensores de orientación y movimiento se han popularizado mucho en los dispositivos móviles, ya que permiten implementar funcionalidades de gran utilidad para dichos dispositivos, como la detección de la orientación del dispositivo para visualizar correctamente documentos o imágenes, implementar aplicaciones de realidad aumentada combinando orientación y cámara, o aplicaciones de navegación con la brújula.

Para acceder a estos sensores utilizaremos la clase `SensorManager`. Para obtener un

Copyright © 2014 Depto. CCIA All rights reserved.

gestor de sensores utilizaremos el siguiente código:

```
String servicio = Context.SENSOR_SERVICE;
SensorManager sensorManager =
       (SensorManager)getSystemService(servicio);
```

Una vez tenemos nuestro `SensorManager` a través de él podemos obtener acceso a un determinado tipo de sensor. Los tipos de sensor que podemos solicitar son las siguientes constantes de la clase `Sensor`:

- `TYPE_ACCELEROMETER`: Acelerómetro de tres ejes, que nos proporciona la aceleración a la que se somete el dispositivo en los ejes x, y, z en m/s^2.
- `TYPE_GYROSCOPE`: Giroscopio que proporciona la orientación del dispositivo en los tres ejes a partir de los cambios de orientación que sufre el dispositivo.
- `TYPE_MAGNETIC_FIELD`: Sensor tipo brújula, que nos proporciona la orientación del campo magnético de los tres ejes en microteslas.
- `TYPE_ORIENTATION`: Su uso está desaconsejado. Se trata de un sensor virtual, que combina información de varios sensores para darnos la orientación del dispositivo. En lugar de este tipo de sensor, se recomienda obtener la orientación combinando manualmente la información del acelerómetro y de la brújula.
- `TYPE_LIGHT`: Detecta la iluminación ambiental, para así poder modificar de forma automática el brillo de la pantalla.
- `TYPE_PROXIMITY`: Detecta la proximidad del dispositivo a otros objetos, utilizado habitualmente para apagar la pantalla cuando situamos el móvil cerca de nuestra oreja para hablar.
- `TYPE_TEMPERATURE`: Termómetro que mide la temperatura en grados Celsius.
- `TYPE_PRESSURE`: Nos proporciona la presión a la que está sometido el dispositivo en kilopascales.

Vamos a centrarnos en estudiar los sensores que nos proporcionan la orientación y los movimientos que realiza el dispositivo.

Para solicitar acceso a un sensor de un determinado tipo utilizaremos el siguiente método:

```
Sensor sensor = sensorManager
    .getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
```

Una vez tenemos el sensor, deberemos definir un *listener* para recibir los cambios en las lecturas del sensor. Este *listener* será una clase que implemente la interfaz `SensorEventListener`, que nos obliga a definir los siguientes métodos:

```
class ListenerSensor implements SensorEventListener {
    public void onSensorChanged(SensorEvent sensorEvent) {
        // La lectura del sensor ha cambiado
    }

    public void onAccuracyChanged(Sensor sensor, int accuracy) {
        // La precisión del sensor ha cambiado
    }
}
```

Una vez hemos definido el *listener*, tendremos que registrarlo para que reciba los eventos

Copyright © 2014 Depto. CCIA All rights reserved.

del sensor solicitado. Para registrarlo, además de indicar el sensor y el *listener*, deberemos especificar la periodicidad de actualización de los datos del sensor. Cuanta mayor sea la frecuencia de actualización, más recursos consumirá nuestra aplicación. Podemos utilizar como frecuencia de actualización las siguientes constantes de la clase `SensorManager`, ordenadas de mayor o menor frecuencia:

- `SENSOR_DELAY_FASTER`: Los datos se actualizan tan rápido como pueda el dispositivo.
- `SENSOR_DELAY_GAME`: Los datos se actualizan a una velocidad suficiente para ser utilizados en videojuegos.
- `SENSOR_DELAY_NORMAL`: Esta es la tasa de actualización utilizada por defecto.
- `SENSOR_DELAY_UI`: Los datos se actualizan a una velocidad suficiente para mostrarlo en la interfaz de usuario.

Una vez tenemos el sensor que queremos utilizar, el *listener* al que queremos que le proporcione las lecturas, y la tasa de actualización de dichas lecturas, podemos registrar el *listener* para empezar a obtener lecturas de la siguiente forma:

```
ListenerSensor listener = new ListenerSensor();
sensorManager.registerListener(listener,
    sensor, SensorManager.SENSOR_DELAY_NORMAL);
```

Una vez hecho esto, comenzaremos a recibir actualizaciones de los datos del sensor mediante el método `onSensorChanged` del *listener* que hemos definido. Dentro de dicho método podemos obtener las lecturas a través del objeto `SensorEvent` que recibimos como parámetro. Este objeto contiene un *array* `values` que contiene las lecturas recogidas, que variarán según el tipo de sensor utilizado, pudiendo contar con entre 1 y 3 elementos. Por ejemplo, un sensor de temperatura nos dará un único valor con la temperatura, mientras que un sensor de orientación nos dará 3 valores, con la orientación del dispositivo en cada uno de los 3 ejes.

Una vez hayamos terminado de trabajar con el sensor, debemos desconectar nuestro *listener* para evitar que se malgasten recursos:

```
sensorManager.unregisterListener(listener);
```

Es recomendable quitar y volver a poner los *listeners* de los sensores cada vez que se pausa y se reanuda la aplicación, utilizando para ello los método `onPause` y `onResume` de nuestra actividad.

A continuación vamos a ver con más detalle las lecturas que se obtienen con los principales tipos de sensores de aceleración y orientación.

7.2.1. Aceleración

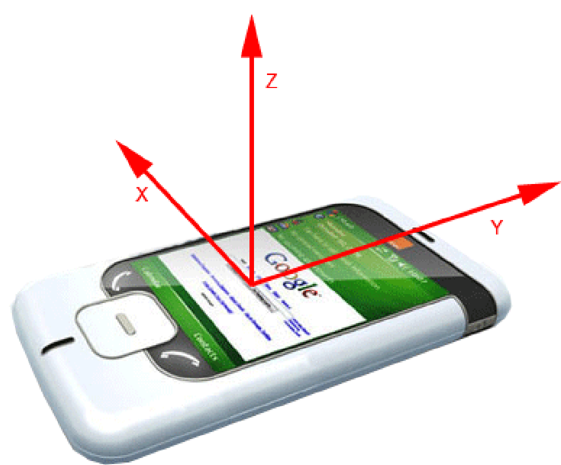
El acelerómetro (sensor de tipo `TYPE_ACCELEROMETER`) nos proporcionará la aceleración a la que sometemos al dispositivo en m/s^2 menos la fuerza de la gravedad. Obtendremos en `values` una tupla con tres valores:

- `values[0]`: Aceleración en el eje X. Dará un valor positivo si movemos el dispositivo

Copyright © 2014 Depto. CCIA All rights reserved.

hacia la derecha, y negativo hacia la izquierda.

- `values[1]`: Aceleración en el eje Y. Dará un valor positivo si movemos el dispositivo hacia arriba y negativo hacia abajo.
- `values[2]`: Aceleración en el eje Z. Dará un valor positivo si movemos el dispositivo hacia adelante (en la dirección en la que mira la pantalla), y negativo si lo movemos hacia atrás.



Ejes de aceleración

A todos estos valores deberemos restarles la fuerza ejercida por la gravedad, que dependerá de la orientación en la que se encuentre el móvil. Por ejemplo, si el móvil reposa sobre una mesa cara arriba la aceleración en el eje Z será de 9.81 m/s^2 (gravedad).

7.2.2. Orientación

En la mayoría de dispositivos podremos acceder a un sensor de tipo `TYPE_ORIENTATION`. Sin embargo, no se trata de un sensor físico, sino que es un sensor virtual cuyo resultado se obtiene combinando varios sensores (normalmente la brújula y el acelerómetro). Este sensor nos proporciona la orientación del dispositivo en los tres ejes, mediante una tupla de tres elementos en `values`:

- `values[0]`: *Azimuth*. Orientación del dispositivo (de `0` a `359` grados). Si el dispositivo reposa cara arriba sobre una mesa, este ángulo cambiará según lo giramos. El valor `0` corresponde a una orientación hacia el norte, `90` al este, `180` al sur, y `270 al oeste`.
- `values[1]`: *Pitch*. Inclinación del dispositivo (de `-180` a `180` grados). Si el dispositivo está reposando sobre una mesa boca arriba, este ángulo será `0`, si lo cogemos en vertical será `-90`, si lo cogemos al revés será `90`, y si lo ponemos boca abajo en la mesa será `180`.
- `values[2]`: *Roll*. Giro del dispositivo hacia los lados (de `-90` a `90` grados). Si el móvil reposa sobre la mesa será `0`, si lo ponemos en horizontal con la pantalla

Copyright © 2014 Depto. CCIA All rights reserved.

mirando hacia la izquierda será `-90`, y `90` si mira a la derecha.

Ejes de orientación

Sin embargo, el uso de este sensor se encuentra desaprobado, ya que no proporciona una buena precisión. En su lugar, se recomienda combinar manualmente los resultados obtenidos por el acelerómetro y la brújula.

La brújula nos permite medir la fuerza del campo magnético para los tres ejes en micro-Teslas. El *array* `values` nos devolverá una tupla de tres elementos con los valores del campo magnético en los ejes X, Y, y Z. Podemos guardar los valores de la aceleración y los valores del campo magnético obtenidos para posteriormente combinarlos y obtener la orientación con una preción mayor que la que nos proporciona el sensor `TYPE_ORIENTATION`, aunque con un coste computacional mayor. Vamos a ver a continuación cómo hacer esto. Considerando que hemos guardado las lecturas del acelerómetro en un campo `valuesAcelerometro` y las de la brújula en `valuesBrujula`, podemos obtener la rotación del dispositivo a partir de ellos de la siguiente forma:

```
float[] values = new float[3];
float[] R = new float[9];
SensorManager.getRotationMatrix(R, null,
    valuesAcelerometro, valuesBrujula);
SensorManager.getOrientation(R, values);
```

En `values` habremos obtenido los valores para el *azimuth*, *pitch*, y *roll*, aunque en este caso los tendremos en radianes. Si queremos convertirlos a grados podremos utilizar el método `Math.toDegrees`.

7.3. Geolocalización

Los dispositivos móviles son capaces de obtener su posición geográfica por diferentes medios. Muchos dispositivos cuentan un con GPS capaz de proporcionarnos nuestra posición con un error de unos pocos metros. El inconveniente del GPS es que sólo

Copyright © 2014 Depto. CCIA All rights reserved.

funciona en entornos abiertos. Cuando estamos en entornos de interior, o bien cuando nuestro dispositivo no cuenta con GPS, una forma alternativa de localizarnos es mediante la red 3G o WiFi. En este caso el error de localización es bastante mayor.

Para poder utilizar los servicios de geolocalización, debemos solicitar permiso en el *manifest* para acceder a estos servicios. Se solicita por separado permiso para el servicio de localización de forma precisa (*fine*) y para localizarnos de forma aproximada (*coarse*):

```
<uses-permission android:name=
    "android.permission.ACCESS_FINE_LOCATION"/>
<uses-permission android:name=
    "android.permission.ACCESS_COARSE_LOCATION"/>
```

Si se nos concede el permiso de localización precisa, tendremos automáticamente concedido el de localización aproximada. El dispositivo GPS necesita tener permiso para localizarnos de forma precisa, mientras que para la localización mediante la red es suficiente con tener permiso de localización aproximada.

Para acceder a los servicios de geolocalización en Android tenemos la clase `LocationManager`. Esta clase no se debe instanciar directamente, sino que obtendremos una instancia como un servicio del sistema de la siguiente forma:

```
LocationManager manager = (LocationManager)
    this.getSystemService(Context.LOCATION_SERVICE);
```

Para obtener una localización deberemos especificar el proveedor que queramos utilizar. Los principales proveedores disponibles en los dispositivos son el GPS (`LocationManager.GPS_PROVIDER`) y la red 3G o WiFi (`LocationManager.NETWORK_PROVIDER`). Podemos obtener información sobre estos proveedores con:

```
LocationProvider proveedor = manager
    .getProvider(LocationManager.GPS_PROVIDER);
```

La clase `LocationProvider` nos proporciona información sobre las características del proveedor, como su precisión, consumo, o datos que nos proporciona.

Es también posible obtener la lista de todos los proveedores disponibles en nuestro móvil con `getProviders`, u obtener un proveedor basándonos en ciertos criterios como la precisión que necesitamos, el consumo de energía, o si es capaz de obtener datos como la altitud a la que estamos o la velocidad a la que nos movemos. Estos criterios se especifican en un objeto de la clase `Criteria` que se le pasa como parámetro al método `getProviders`.

Para obtener la última localización registrada por un proveedor llamaremos al siguiente método:

```
Location posicion = manager
    .getLastKnownLocation(LocationManager.GPS_PROVIDER);
```

El objeto `Location` obtenido incluye toda la información sobre nuestra posición, entre la

Copyright © 2014 Depto. CCIA All rights reserved.

que se encuentra la latitud y longitud.

Con esta llamada obtenemos la última posición que se registró, pero no se actualiza dicha posición. A continuación veremos cómo solicitar que se realice una nueva lectura de la posición en la que estamos.

7.3.1. Actualización de la posición

Para poder recibir actualizaciones de nuestra posición deberemos definir un *listener* de clase `LocationListener`:

```
class ListenerPosicion implements LocationListener {
    public void onLocationChanged(Location location) {
        // Recibe nueva posición.
    }
    public void onProviderDisabled(String provider){
        // El proveedor ha sido desconectado.
    }
    public void onProviderEnabled(String provider){
        // El proveedor ha sido conectado.
    }
    public void onStatusChanged(String provider,
                    int status, Bundle extras){
        // Cambio en el estado del proveedor.
    }
};
```

Una vez definido el *listener*, podemos solicitud actualizaciones de la siguiente forma:

```
ListenerPosicion listener = new ListenerPosicion();
long tiempo = 5000; // 5 segundos
float distancia = 10; // 10 metros

manager.requestLocationUpdates(
    LocationManager.GPS_PROVIDER,
    tiempo, distancia, listenerPosicion);
```

Podemos observar que cuando pedimos las actualizaciones, además del proveedor y del *listener*, debemos especificar el intervalo mínimo de tiempo (en milisegundos) que debe transcurrir entre dos lecturas consecutivas, y el umbral de distancia mínima que debe variar nuestra posición para considerar que ha habido un cambio de posición y notificar la nueva lectura.

> **Nota**
>
> Debemos tener en cuenta que esta forma de obtener la posición puede tardar un tiempo en proporcionarnos un valor. Si necesitamos obtener un valor de posición de forma inmediata utilizaremos `getLastKnownLocation`, aunque puede darnos un valor sin actualizar.

Una vez hayamos terminado de utilizar el servicio de geolocalización, deberemos detener las actualizaciones para reducir el consumo de batería. Para ello eliminamos el *listener* de la siguiente forma:

```
manager.removeUpdates(listener);
```


Copyright © 2014 Depto. CCIA All rights reserved.

7.3.2. Alertas de proximidad

En Android podemos definir una serie de altertas que se disparan cuando nos acercamos a una determinada posición. Recibiremos los avisos de proximidad mediante *intents*. Por ello, primero debemos crearnos un `Intent` propio:

```
Intent intent = new Intent(codigo);
PendingIntent pi = PendingIntent.getBroadcast(this, -1, intent, 0);
```

Para programar las alertas de proximidad deberemos especificar la latitud y longitud, y el radio de la zona de proximidad en metros. Además podemos poner una caducidad a las alertas (si ponemos `-1` no habrá caducidad):

```
double latitud = 128.342353;
double longitud = 0.4887897;
float radio = 500f;
long caducidad = -1;

manager.addProximityAlert(latitud, longitud, radio,
                          caducidad, pi);
```

Necesitaremos también un receptor de *intents* de tipo *broadcast* para recibir los avisos:

```
public class ReceptorProximidad extends BroadcastReceiver {
  @Override
  public void onReceive(Context context, Intent intent) {
    // Comprobamos si estamos entrando o saliendo de la proximidad
    String key = LocationManager.KEY_PROXIMITY_ENTERING;
    Boolean entra = intent.getBooleanExtra(key, false);
    ...
  }
}
```

Finalmente, para recibir los *intents* debemos registrar el receptor que acabamos de crear de la siguiente forma:

```
IntentFilter filtro = new IntentFilter(codigo);
registerReceiver(new ReceptorProximidad(), filtro);
```

7.3.3. Geocoding

El *geocoder* nos permite realizar tranformaciones entre una dirección y las coordenadas en las que está. Podemos obtener el objeto `Geocoder` con el que realizar estas transformaciones de la siguiente forma:

```
Geocoder geocoder = new Geocoder(this, Locale.getDefault());
```

Podemos obtener la dirección a partir de unas coordenadas (latitud y longitud):

```
List<Address> direcciones = geocoder
    .getFromLocation(latitud, longitud, maxResults);
```

También podemos obtener las coordenadas correspondientes a una determinada dirección:

Copyright © 2014 Depto. CCIA All rights reserved.

```
List<Address> coordenadas = geocoder
    .getFromLocationName(direccion, maxResults);
```

7.4. Reconocimiento del habla

Otro sensor que podemos utilizar para introducir información en nuestras aplicaciones es el micrófono que incorpora el dispositivo. Tanto el micrófono como la cámara se pueden utilizar para capturar audio y video, lo cual será visto cuando estudiemos las capacidades multimedia. Sin embargo, una característica altamente interesante de los dispositivos Android es que nos permiten realizar reconocimiento del habla de forma sencilla para introducir texto en nuestras aplicaciones.

Para realizar este reconocimiento deberemos utilizar *intents*. Concretamente, crearemos un `Intent` mediante las constantes definidas en la clase `RecognizerIntent`, que es la clase principal que deberemos utilizar para utilizar esta característica.

Lo primer que deberemos hacer es crear un `Intent` para inicial el reconocimiento:

```
Intent intent = new Intent(
    RecognizerIntent.ACTION_RECOGNIZE_SPEECH);
```

Una vez creado, podemos añadir una serie de parámetros para especificar la forma en la que se realizará el reconocimiento. Estos parámetros se introducen llamando a:

```
intent.putExtra(parametro, valor);
```

Los parámetros se definen como constantes de la clase `RecognizerIntent`, todas ellas tienen el prefijo `EXTRA_`. Algunos de estos parámetros son:

Parámetro	Valor
`EXTRA_LANGUAGE_MODEL`	**Obligatorio**. Debemos especificar el tipo de lenguaje utilizado. Puede ser lenguaje orientado a realizar una búsqueda web (`LANGUAGE_MODEL_WEB_SEARCH`), o lenguaje de tipo general (`LANGUAGE_MODEL_FREE_FORM`).
`EXTRA_LANGUAGE`	**Opcional**. Se especifica para hacer el reconocimiento en un idioma diferente al idioma por defecto del dispositivo. Indicaremos el idioma mediante la etiqueta IETF correspondiente, como por ejemplo `"es-ES"` o `"en-US"`
`EXTRA_PROMPT`	**Opcional**. Nos permite indicar el texto a mostrar en la pantalla mientras se realiza el reconocimiento. Se especifica mediante una cadena de texto.
`EXTRA_MAX_RESULTS`	**Opcional**. Nos permite especificar el número máximo de posibles resultados que queremos

Copyright © 2014 Depto. CCIA All rights reserved.

	que nos devuelva. Se especifica mediante un número entero.

Una vez creado el *intent* y especificados los parámetros, podemos lanzar el reconocimiento llamando, desde nuestra actividad, a:

```
startActivityForResult(intent, codigo);
```

Como código deberemos especifica un entero que nos permita identificar la petición que estamos realizado. En la actividad deberemos definir el *callback* `onActivityResult`, que será llamado cuando el reconocimiento haya finalizado. Aquí deberemos comprobar en primer lugar que el código de petición al que corresponde el *callback* es el que pusimos al lanzar la actividad. Una vez comprobado esto, obtendremos una lista con los resultados obtenidos de la siguiente forma:

```
@Override
protected void onActivityResult(int requestCode,
                int resultCode, Intent data) {
    if (requestCode == codigo && resultCode == RESULT_OK) {

        ArrayList<String> resultados =
            data.getStringArrayListExtra(
                RecognizerIntent.EXTRA_RESULTS);

        // Utilizar los resultados obtenidos
        ...
    }
    super.onActivityResult(requestCode, resultCode, data);
}
```


Copyright © 2014 Depto. CCIA All rights reserved.

8. Sensores y eventos - Ejercicios

8.1. Pantalla táctil

Vamos a implementar una nueva aplicación `Eventos`, en la que mostraremos una caja en la pantalla (un rectángulo de 20x20) y la moveremos utilizando la pantalla táctil. Se pide:

a) Empezar haciendo que se mueva la caja al punto en el que el usuario pone el dedo y comprobar que funciona correctamente (sólo hace falta reconocer el evento `DOWN`).

b) Implementar ahora también el evento de movimiento (`MOVE`), para hacer que la caja se desplace conforme movemos el dedo.

c) Sólo querremos que la caja se mueva si cuando pusimos el dedo en la pantalla lo hicimos sobre la caja. En el evento `DOWN` ya no moveremos la caja, sino que simplemente comprobaremos si hemos pulsado encima de ella.

> **Ayuda**
> Esto último se puede conseguir de forma sencilla devolviendo `true` o `false` cuando se produzca el evento `DOWN`, según si queremos seguir recibiendo eventos para ese gesto o no. Si se pulsa fuera de la caja podemos devolver `false` para así no recibir ningún evento de movimiento correspondiente a ese gesto.

8.2. Gestos

Continuaremos trabajando con el proyecto anterior, en este caso para reconocer gestos. Se pide:

a) Modificar el ejercicio anterior para utilizar un detector de gestos para desplazar la caja. Utilizaremos el evento `onDown` para determinar si el gesto ha comenzado sobre la caja, y `onScroll` para desplazarla.

b) Reconocer el evento *tap* realizado sobre la caja. Cuando esto ocurra se deberá cambiar el color de la caja.

c) Reconocer el gesto *fling* ejercido sobre la caja. Cuando esto ocurra mostraremos un vector (línea) saliendo de la posición en la que terminó el gesto indicando la velocidad y dirección con la que se lanzó.

d) De forma optativa, se puede hacer que al realizar el gesto *fling* sobre la caja ésta se lance con cierta inercia. Para hacer esto necesitaremos un hilo o temporizador que vaya actualizando la posición de la caja según su velocidad, e irá dismunuyendo su velocidad debido al rozamiento.

Copyright © 2014 Depto. CCIA All rights reserved.

8.3. Acelerómetro

Implementar una aplicación `Acelerometro` que muestre en una tabla los valores de aceleración para las coordenadas X, Y, Z.

> **Nota**
> Sólo podremos hacer este ejercicio si contamos con un dispositivo real, ya que el emulador no soporta este tipo de sensor.

8.4. Geolocalización

Implementar una nueva aplicación `Geolocalizacion` que nos localice geográficamente utilizando GPS, y nos muestre tanto nuestras coordenadas como nuestra dirección en forma de texto.

Para poder probar esto en el emulador deberemos indicarle manualmente al emulador las coordenadas en las que queremos que se localize. Esto lo podemos hacer de dos formas: mediante línea de comando o mediante la aplicación DDMS. Vamos a verlas a continuación.

Para comunicar las coordenadas al emulador mediante línea de comando deberemos conectarnos a él mediante `telnet`. Por ejemplo, si nuestro emulador está funcionando en el puerto `5554`, haremos un `telnet` a `localhost` y a dicho puerto:

```
telnet localhost 5554
```

Una vez dentro de la línea de comando del emulador, invocaremos el comando `geo` para suministrarle las coordenadas. Por ejemplo, las siguientes coordenadas corresponden a la Universidad de Alicante:

```
geo fix -0.51515 38.3852333
```

Si no queremos tener que ir a línea de comando, podemos utilizar la aplicación DDMS a la que se puede acceder de forma indepdiente o desde dentro de Eclipse. Dado que estamos ejecutando el emulador desde Eclipse, deberemos lanzar DDMS también dentro de este entorno. Para ello deberemos mostrar la vista *Emulator Control*. En ella veremos unos cuadros de texto y un botón con los que enviar las coordenadas al emulador, siempre que esté en funcionamiento.

> **Advertencia**
> Debido a un *bug* del SDK de Android, el DDMS no envía correctamente las coordenadas al emulador si nuestro *locale* no está configurado con idioma inglés. Para solucionar esto de forma sencilla, podemos editar el fichero `eclipse.ini` y añadir dentro de él la opción `-Duser.language.en`. Si no hacemos esto, el emulador recibirá siempre las coordenadas `0, 0`.

Copyright © 2014 Depto. CCIA All rights reserved.

Para utilizar el *geocoder*, deberemos utilizar un emulador que incorpore las APIs de Google (para así poder acceder a la API de mapas). Además, dado que necesitará conectarse a Internet para obtener las direcciones, deberemos solicitar el permiso `INTERNET`.

> **Advertencia**
>
> En el emulador de la plataforma Android 2.2 no funciona correctamente el *geocoder*. Funcionará correctamente si utilizamos, por ejemplo, un emulador con Google APIs de nivel 3 (versión 1.5 de la plataforma).

8.5. Reconocimiento del habla

Implementar una nueva aplicación `Habla` con un campo de texto y un botón. Al pulsar sobre el botón se lanzará el módulo de reconocimiento del habla, y una vez finalizado mostraremos lo que se haya reconocido en el campo de texto.

> **Nota**
>
> Sólo podremos hacer este ejercicio si contamos con un dispositivo real, ya que el emulador no soporta el reconocimiento del habla.

Copyright © 2014 Depto. CCIA All rights reserved.

9. Multimedia

La capacidad de reproducir contenido multimedia es una característica presente en la práctica totalidad de las terminales telefónicas existentes en el mercado hoy en día. Muchos usuarios prefieren utilizar las capacidades multimedia de su teléfono, en lugar de tener que depender de otro dispositivo adicional para ello. Android incorpora la posibilidad de reproducir no sólo audio en diversos formatos, sino que también vídeo. Los formatos de audio soportados son los siguientes:

- AAC LC/LTP
- HE-AACv1 (AAC+)
- HE-AACv2 (Enhanced ACC+)
- AMR-NB
- AMR-WB
- FLAC
- MP3
- MIDI
- Ogg Vorbis
- PCM/Wave

Con respecto al vídeo, los formatos soportados son:

- H.263
- H.264 AVC
- MPEG-4 SP
- VP8

En esta sesión echaremos un vistazo a las herramientas necesarias para poder reproducir contenido multimedia (audio o vídeo) en una actividad. También veremos cómo añadir la capacidad a nuestra aplicación para la toma de fotografías, una característica perfectamente emulada por el emulador en las últimas versiones del Android SDK.

9.1. Reproducción de audio

La reproducción de contenido multimedia se lleva a cabo por medio de la clase `MediaPlayer`. Dicha clase nos permite la reproducción de archivos multimedia almacenados como recursos de la aplicación, en ficheros locales, en proveedores de contenido, o servidos por medio de streaming a partir de una URL. En todos los casos, como desarrolladores, la clase `MediaPlayer` nos permitirá abstraernos del formato así como del origen del fichero a reproducir.

Incluir un fichero de audio en los recursos de la aplicación para poder ser reproducido durante su ejecución es muy sencillo. Simplemente creamos una carpeta `raw` dentro de la carpeta `res`, y almacenamos en ella sin comprimir el fichero o ficheros que deseamos reproducir. A partir de ese momento el fichero se identificará dentro del código como

Copyright © 2014 Depto. CCIA All rights reserved.

`R.raw.nombre_fichero` (obsérvese que no es necesario especificar la extensión del fichero).

Para reproducir un fichero de audio tendremos que seguir una secuencia de pasos. En primer lugar deberemos crear una instancia de la clase `MediaPlayer`. El siguiente paso será indicar qué fichero será el que se reproducirá. Por último ya podremos llevar a cabo la reproducción en sí misma del contenido multimedia.

Veamos primero cómo inicializar la reproducción. Tenemos dos opciones. La primera de ellas consiste en crear una instancia de la clase `MediaPlayer` por medio del método `create()`. En este caso se deberá pasar como parámetro, además del contexto de la aplicación, el identificador del recurso, tal como se puede ver en el siguiente ejemplo:

```
Context appContext = getApplicationContext();

// Recurso de la aplicación
MediaPlayer resourcePlayer = MediaPlayer.create(appContext,
R.raw.my_audio);
// Fichero local (en la tarjeta de memoria)
MediaPlayer filePlayer = MediaPlayer.create(appContext,
        Uri.parse("file:///sdcard/localfile.mp3"));
// URL
MediaPlayer urlPlayer = MediaPlayer.create(appContext,
        Uri.parse("http://site.com/audio/audio.mp3"));
// Proveedor de contenidos
MediaPlayer contentPlayer = MediaPlayer.create(appContext,
        Settings.System.DEFAULT_RINGTONE_URI);
```

El otro modo de inicializar la reproducción multimedia es por medio del método `setDataSource()` para asignar una fuente multimedia a una instancia ya existente de la clase `MediaPlayer`. En este caso es muy importante recordar que se deberá llamar al método `prepare()` antes de poder reproducir el fichero de audio (recuerda que esto último no es necesario si la instancia de `MediaPlayer` se ha creado con el método `create()`).

Aviso:

La clase `MediaPlayer` funciona como una máquina de estados. Se deben seguir los diferentes pasos para la preparación y reproducción del contenido multimedia en el orden correcto. En caso contrario se lanzará una excepción. El diagrama de estados de dicha clase se puede consultar en http://developer.android.com/reference/android/media/MediaPlayer.html.

```
MediaPlayer mediaPlayer = new MediaPlayer();
mediaPlayer.setDataSource("/sdcard/test.mp3");
mediaPlayer.prepare();
```

Una vez que la instancia de la clase `MediaPlayer` ha sido inicializada, podemos comenzar la reproducción mediante el método `start()`. También es posible utilizar los métodos `stop()` y `pause()` para detener y pausar la reproducción. En este último caso la reproducción continuará tras hacer una llamada al método `play()`.

Otros métodos de la clase `MediaPlayer` que podríamos considerar interesante utilizar son los siguientes:

Copyright © 2014 Depto. CCIA All rights reserved.

- `setLooping()` nos permite especificar si el clip de audio deberá volver a reproducirse desde el principio una vez que éste llegue al final.

```
if (!mediaPlayer.isLooping())
        mediaPlayer.setLooping(true);
```

- `setScreenOnWhilePlaying()` nos permitirá conseguir que la pantalla se encuentre activada siempre durante la reproducción. Tiene más sentido en el caso de la reproducción de video, que será tratada en la siguiente sección.

```
mediaPlayer.setScreenOnWhilePlaying(true);
```

- `setVolume()` modifica el volumen. Recibe dos parámetros que deberán ser dos números reales entre 0 y 1, indicando el volumen del canal izquierdo y del canal derecho, respectivamente. El valor 0 indica silencio total mientras que el valor 1 indica máximo volumen.

```
mediaPlayer.setVolume(1f, 0.5f);
```

- `seekTo()` permite avanzar o retroceder a un determinado punto del archivo de audio. Podemos obtener la duración total del clip de audio con el método `getDuration()`, mientras que `getCurrentPosition()` nos dará la posición actual. En el siguiente código se puede ver un ejemplo de uso de estos tres últimos métodos.

```
mediaPlayer.start();

int pos = mediaPlayer.getCurrentPosition();
int duration = mediaPlayer.getDuration();

mediaPlayer.seekTo(pos + (duration-pos)/10);
```

Una acción muy importante que deberemos llevar a cabo una vez haya finalizado definitivamente la reproducción (porque se vaya a salir la actividad, por ejemplo) es destruir la instancia de la clase `MediaPlayer` y liberar su memoria. Para ello deberemos hacer uso del método `release()`.

```
mediaPlayer.release();
```

9.2. Reproducción de vídeo usando el control VideoView

La reproducción de vídeo es muy similar a la reproducción de audio, salvo dos particularidades. En primer lugar, no es posible reproducir un clip de vídeo almacenado como parte de los recursos de la aplicación. En este caso deberemos utilizar cualquiera de los otros tres medios (ficheros locales, streaming o proveedores de contenidos). Un poco más adelante veremos cómo añadir un clip de vídeo a la tarjeta de memoria de nuestro terminal emulado desde la propia interfaz de Eclipse. En segundo lugar, el vídeo necesitará de una superficie para poder reproducirse. Esta superficie se corresponderá con una vista dentro del layout de la actividad.

Existen varias alternativas para la reproducción de vídeo, teniendo en cuenta lo que acabamos de comentar. La más sencilla es hacer uso de un control de tipo *VideoView*, que encapsula tanto la creación de una superficie en la que reproducir el vídeo como el

Copyright © 2014 Depto. CCIA All rights reserved.

control del mismo mediante una instancia de la clase `MediaPlayer`. Este método será el que veamos en primer lugar.

El primer paso consistirá en añadir el control *VideoView* a la interfaz gráfica de la actividad en la que queramos que se reproduzca el vídeo. Podemos añadir algo como lo siguiente el fichero de layout correspondiente:

```
<VideoView android:id="@+id/superficie"
              android:layout_height="fill_parent"
              android:layout_width="fill_parent">
</VideoView>
```

Dentro del código Java podremos acceder a dicho elemento de la manera habitual, es decir, mediante el método `findViewById()`. Una vez hecho esto, asignaremos una fuente que se corresponderá con el contenido multimedia a reproducir. El control *VideoView* se encargará de la inicialización del objeto `MediaPlayer`. Para asignar un video a reproducir podemos utilizar cualquiera de estos dos métodos:

```
videoView1.setVideoUri("http://www.mysite.com/videos/myvideo.3gp");
videoView2.setVideoPath("/sdcard/test2.3gp");
```

Una vez inicializado el control se puede controlar la reproducción con los métodos `start()`, `stopPlayback()`, `pause()` y `seekTo()`. La clase `VideoView` también incorpora el método `setKeepScreenOn(boolean)`con la que se podrá controlar el comportamiento de la iluminación de la pantalla durante la reproducción del clip de vídeo. Si se pasa como parámetro el valor `true` ésta permanecerá constantemente iluminada.

El siguiente código muestra un ejemplo de asignación de un vídeo a un control *VideoView* y de su posterior reproducción. Dicho código puede ser utilizado a modo de esqueleto en nuestra propia actividad. También podemos ver un ejemplo de uso de `seekTo()`, en este caso para avanzar hasta la posición intermedia del video.

```
VideoView videoView = (VideoView)findViewById(R.id.superficie);
videoView.setKeepScreenOn(true);
videoView.setVideoPath("/sdcard/ejemplo.3gp");

if (videoView.canSeekForward())
        videoView.seekTo(videoView.getDuration()/2);

videoView.start();

// Hacer algo durante la reproducción

videoView.stopPlayback();
```

En esta sección veremos en último lugar, tal como se ha indicado anteriormente, la manera de añadir archivos a la tarjeta de memoria de nuestro dispositivo virtual, de tal forma que podamos almacenar clips de vídeo y resolver los ejercicios propuestos para la sesión. Se deben seguir los siguientes pasos:

- En primer lugar el emulador debe encontrarse en funcionamiento, y por supuesto, el dispositivo emulado debe hacer uso de una tarjeta SD.

Copyright © 2014 Depto. CCIA All rights reserved.

- En Eclipse debemos cambiar a la perspectiva *DDMS*. Para ello hacemos uso de la opción *Window->Open Perspective...*
- A continuación seleccionamos la pestaña *File Explorer*. El contenido de la tarjeta de memoria se halla en la carpeta */mnt/sdcard/*.
- En de dicha carpeta deberemos introducir nuestros archivos de vídeo, dentro del directorio *DCIM*. Al hacer esto ya podrán reproducirse desde la aplicación nativa de reproducción de vídeo y también desde nuestras propias aplicaciones. Podemos introducir un archivo de video con el ratón, arrastrando un fichero desde otra carpeta al interior de la carpeta *DCIM*, aunque también podemos hacer uso de los controles que aparecen en la parte superior derecha de la perspectiva *DDMS*, cuando la pestaña *File Explorer* está seleccionada. La función de estos botones es, respectivamente: guardar en nuestra máquina real algún archivo de la tarjeta de memoria virtual, guardar en la tarjeta de memoria virtual un archivo, y eliminar el archivo seleccionado.

Aviso:

A veces es necesario volver a arrancar el terminal emulado para poder acceder a los vídeos insertados siguiendo este método en la tarjeta de memoria desde la aplicación *Galería de Android*.

9.3. Reproducción de vídeo basada en MediaPlayer

La segunda alternativa para la reproducción de video consiste en la creación de una superficie sobre la que dicho vídeo se reproducirá y en el uso directo de la clase `MediaPlayer`. La superficie deberá ser asignada a la instancia correspondiente de dicha clase. En caso contrario el vídeo no se mostrará. Además, la clase `MediaPlayer` requiere que la superficie sea un objeto de tipo `SurfaceHolder`.

Para añadir una vista de tipo *SurfaceHolder* a la interfaz gráfica de la actividad debemos incluir una vista `SurfaceView` en el archivo de layout correspondiente

```
<SurfaceView
        android:id="@+id/superficie"
        android:layout_width="200px"
        android:layout_height="200px"
        android:layout_gravity="center">
</SurfaceView>
```

El siguiente paso será la inicialización el objeto `SurfaceView` y la asignación del mismo a la instancia de la clase `MediaPlayer` encargada de reproducir el vídeo. El siguiente código muestra cómo hacer esto. Obsérvese que es necesario que la actividad implemente la interfaz `SurfaceHolder.Callback`. Esto es así porque los objetos de la clase `SurfaceHolder` se crean de manera asíncrona, por lo que debemos añadir un mecanismo

Copyright © 2014 Depto. CCIA All rights reserved.

que permita esperar a que dicho objeto haya sido creado antes de poder empezar a reproducir el vídeo.

```
// La clase debe implementar la interfaz SurfaceHolder.Callback
public class MiActividad extends Activity implements
SurfaceHolder.Callback
{
        private MediaPlayer mediaPlayer;

        @Override
        public void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                setContentView(R.layout.main);
                mediaPlayer = new MediaPlayer();
                SurfaceView superficie =
(SurfaceView)findViewById(R.id.superficie);

                // Hacemos que la actividad maneje los eventos del
SurfaceHolder
                SurfaceHolder holder = superficie.getHolder();
                holder.addCallback(this);
                holder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
        }

        // No comenzamos la reproducción hasta que no se cree
        // la superficie
        public void surfaceCreated(SurfaceHolder holder) {
                try {
                        mediaPlayer.setDisplay(holder);
                        // Reproducir vídeo a continuación
                } catch (IllegalArgumentException e) {
                        Log.d("MEDIA_PLAYER", e.getMessage());
                } catch (IllegalStateException e) {
                        Log.d("MEDIA_PLAYER", e.getMessage());
                }
        }

        // Liberamos la memoria de la instancia de MediaPlayer
        // cuando se vaya a destruir la superficie
        public void surfaceDestroyed(SurfaceHolder holder) {
                mediaPlayer.release();
        }

        public void surfaceChanged(SurfaceHolder holder, int format, int
width,
        int height) { }
        }
}
```

Una vez que hemos asociado la superficie al objeto de la clase `MediaPlayer` debemos asignar a dicho objeto el identificador del clip de vídeo a reproducir. Ya que habremos creado la instancia del objeto `MediaPlayer` previamente, la única posibilidad que tendremos será utilizar el método `setDataSource()`, como se muestra en el siguiente ejemplo. Recuerda que cuando se utiliza dicho método es necesario llamar también al método `prepare()`.

```
public void surfaceCreated(SurfaceHolder holder) {
        try {
                mediaPlayer.setDisplay(holder);
                // Inicio de la reproducción una vez la superficie
                // ha sido asociada a la instancia de MediaPlayer
                mediaPlayer.setDataSource("/sdcard/prueba.3gp");
```


Copyright © 2014 Depto. CCIA All rights reserved.

```
                mediaPlayer.prepare();
                mediaPlayer.start();
        } catch (IllegalArgumentException e) {
                Log.d("MEDIA_PLAYER", e.getMessage());
        } catch (IllegalStateException e) {
                Log.d("MEDIA_PLAYER", e.getMessage());
        } catch (IOException e) {
                Log.d("MEDIA_PLAYER", e.getMessage());
        }
}
```

9.4. Toma de fotografías

De todas las alternativas existentes para la toma de fotografías desde Android veremos la más secilla. Dicha alternativa consiste en hacer uso de un `Intent` implícito cuyo parámetro sea la constante `ACTION_IMAGE_CAPTURE` definida en la clase `MediaStore` (de la que hablaremos más adelante). Al hacer uso de la siguiente línea de código

```
startActivityForResult(new Intent(MediaStore.ACTION_IMAGE_CAPTURE),
TAKE_PICTURE);
```

se ejecutará la aplicación nativa para la toma de fotografías (o cualquier otra actividad instalada que pueda hacerse cargo de dicha tarea), la cual también le permitirá al usuario modificar las opciones pertinentes. Como vemos estamos ante otro ejemplo de reutilización de componentes dentro del sistema Android que nos va a permitir como desarrolladores ahorrar bastante tiempo a la hora de crear nuestras propias aplicaciones. Debemos recordar que al hacer uso del método `startActivityForResult` la actividad actual quedará a la espera de que la fotografía sea tomada; una vez hecho esto la actividad volverá a estar en ejecución, pasándose el control al método `onActivityResult`.

> **Nota:**
> En versiones anteriores del SDK de Android la emulación de la cámara no estaba soportada. Hoy en día es posible simular la cámara del dispositivo virtual por medio de una webcam, así que ya no es necesario utilizar un dispositivo real para poder probar estos ejemplos.

Usando este método podremos optar por dos modos de funcionamiento:

- **Modo thumbnail**: este es el modo de funcionamiento por defecto. Se devolverá un thumbnail de tipo `Bitmap` en el parámetro extra de nombre `data` devuelto por el Intent en el método `onActivityResult()`.
- **Modo de imagen completa**: si se especifica una URI en el parámetro extra `MediaStore.EXTRA_OUTPUT` del `Intent`, se guardará la imagen tomada por la cámara, en su resolución real, en el destino indicado. En este caso no se devolverá un thumbnail como resultado del `Intent` y el campo `data` valdrá `null`.

En el siguiente ejemplo tenemos el esqueleto de una actividad en la que se utiliza un `Intent` para tomar una fotografía, ya sea en modo thumbnail o en modo de imagen completa. Según queramos una cosa o la otra deberemos llamar a los métodos `getThumbnailPicture()` o `saveFullImage()`, respectivamente. En

Copyright © 2014 Depto. CCIA All rights reserved.

`onActivityResult()` se determina el modo empleado examinando el valor del campo extra `data` devuelto por el `Intent`. Por último, una vez tomada la fotografía, se puede almacenar en el *Media Store* (hablamos de esto un poco más adelante) o procesarla dentro de nuestra aplicación antes de descartarla.

```
// La siguiente constante se usará para identificar el Intent lanzado
// para tomar una fotografía, de tal forma que podamos saber
// en onActivityResult que fue esa la actividad que acaba de terminar
private static int TAKE_PICTURE = 1;
private Uri ficheroSalidaUri;

private void getThumbailPicture() {
        // Preparamos y lanzamos el intent implícito para el modo
thumbnail
        Intent intent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
        startActivityForResult(intent, TAKE_PICTURE);
}

private void saveFullImage() {
        // Preparamos y lanzamos el intent implícito para el modo de
imagen completa
        // en este caso añadimos como parámetro extra la ruta donde
guardar el vídeo
        // generado
        Intent intent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
        File file = new File(Environment.getExternalStorageDirectory(),
"prueba.jpg");
        ficheroSalidaUri = Uri.fromFile(file);
        intent.putExtra(MediaStore.EXTRA_OUTPUT, ficheroSalidaUri);
        startActivityForResult(intent, TAKE_PICTURE);
}

@Override
protected void onActivityResult(int requestCode, int resultCode, Intent
data) {
        if (requestCode == TAKE_PICTURE) {
                Uri imagenUri = null;
                // Comprobamos si el Intent ha devuelto un thumbnail
                if (data != null) {
                        if (data.hasExtra("data")) {
                                Bitmap thumbnail =
data.getParcelableExtra("data");
                                // HACER algo con el thumbnail
                        }
                }
                else {
                         // HACER algo con la imagen almacenada en
ficheroSalidaUri
                }
        }
}
```

9.5. Agregar ficheros multimedia en el Media Store

El comportamiento por defecto en Android con respecto al acceso de contenido multimedia es que los ficheros multimedia generados u obtenidos por una aplicación no podrán ser accedidos por el resto. En el caso de que deseemos que un nuevo fichero multimedia sí pueda ser accedido desde el exterior de nuestra aplicación deberemos almacenarlo en el *Media Store*, un **proveedor de contenidos** que mantiene una base de

Copyright © 2014 Depto. CCIA All rights reserved.

datos de la metainformación de todos los ficheros almacenados tanto en dispositivos externos como internos del terminal telefónico.

> **Nota:**
> El Media Store es un proveedor de contenidos, y por lo tanto utilizaremos los mecanismos ya estudiados en sesiones anteriores (consultar el módulo de persistencia) para acceder a la información que contiene.

Existen varias formas de incluir un fichero multimedia en el *Media Store*. La más sencilla es hacer uso de la clase `MediaScannerConnection`, que permitirá determinar automáticamente de qué tipo de fichero se trata, de tal forma que se pueda añadir automáticamente sin necesidad de proporcionar ninguna información adicional.

La clase `MediaScannerConnection` proporciona un método `scanFile()` para realizar esta tarea. Sin embargo, antes de escanear un fichero se deberá llamar al método `connect()` y esperar una conexión al *Media Store*. La llamada a `connect()` es asíncrona, lo cual quiere decir que deberemos crear un objeto `MediaScannerConnectionClient` que nos notifique en el momento en el que se complete la conexión. Esta misma clase también puede ser utilizada para que se lleve a cabo una notificación en el momento en el que el escaneado se haya completado, de tal forma que ya podremos desconectarnos del *Media Store*.

En el siguiente ejemplo de código podemos ver un posible esqueleto para un objeto `MediaScannerConnectionClient`. En este código se hace uso de una instancia de la clase `MediaScannerConnection` para manejar la conexión y escanear el fichero. El método `onMediaScannerConected()` será llamado cuando la conexión ya se haya establecido, con lo que ya será posible escanear el fichero. Una vez se complete el escaneado se llamará al método `onScanCompleted()`, en el que lo más aconsejable es llevar a cabo la desconexión del *Media Store*.

```
MediaScannerConnectionClient mediaScannerClient = new
MediaScannerConnectionClient() {
        private MediaScannerConnection msc = null;
        {
                msc = new MediaScannerConnection(getApplicationContext(),
this);
                msc.connect();
        }

        public void onMediaScannerConnected() {
                msc.scanFile("/sdcard/test1.jpg", null);
        }

        public void onScanCompleted(String path, Uri uri) {

                // Realizar otras acciones adicionales

                msc.disconnect();
        }
};
```


Copyright © 2014 Depto. CCIA All rights reserved.

9.6. Sintetizador de voz de Android

Android incorpora desde la versión 1.6 un motor de síntesis de voz conocido como *Text To Speech*. Mediante su API podremos hacer que nuestros programas "lean" un texto al usuario. Es necesario tener en cuenta que por motivos de espacio en disco los paquetes de lenguaje pueden no estar instalados en el dispositivo. Por lo tanto, antes de que nuestra actividad utilice *Text To Speech* se podría considerar una buena práctica de programación el comprobar si dichos paquetes están instalados. Para ello podemos hacer uso de un `Intent` como el que se muestra a continuación:

```
Intent intent = new Intent(TextToSpeech.Engine.ACTION_CHECK_TTS_DATA);
startActivityForResult(intent, TTS_DATA_CHECK);
```

El método `onActivityResult()` recibirá un `CHECK_VOICE_DATA_PASS` si todo está correctamente instalado. En caso contrario deberemos iniciar una nueva actividad por medio de un nuevo `Intent` que haga uso de la acción `ACTION_INSTALL_TTS_DATA` del motor *Text To Speech*.

Una vez comprobemos que todo está instalado deberemos crear e inicializar una instancia de la clase `TextToSpeech`. Como no podemos utilizar dicha instancia hasta que esté inicializada, la mejor opción es pasar como parámetro al constructor un método `onInit()` de tal forma que en dicho método se especifiquen las tareas a llevar a cabo por el sintetizador de voz una vez esté inicializado.

```
boolean ttsIsInit = false;
TextToSpeech tts = null;

tts = new TextToSpeech(this, new OnInitListener() {
        public void onInit(int status) {
                if (status == TextToSpeech.SUCCESS) {
                        ttsIsInit = true;
                        // Hablar
                }
        }
});
```

Una vez que la instancia esté inicializada se puede utilizar el método `speak()` para sintetizar voz por medio del dispositivo de salida por defecto. El primer parámetro será el texto a sintetizar y el segundo podrá ser o bien **`QUEUE_ADD`**, que añade una nueva salida de voz a la cola, o bien **`QUEUE_FLUSH`**, que elimina todo lo que hubiera en la cola y lo sustituye por el nuevo texto.

```
tts.speak("Hello, Android", TextToSpeech.QUEUE_ADD, null);
```

Otros métodos de interés de la clase `TextToSpeech` son:

- `setPitch()` y `setSpeechRate()` permiten modificar el tono de voz y la velocidad. Ambos métodos aceptan un parámetro real.
- `setLanguage()` permite modificar la pronunciación. Se le debe pasar como parámetro una instancia de la clase `Locale` para indicar el país y la lengua a utilizar.
- El método `stop()` se debe utilizar al terminar de hablar; este método detiene la

Copyright © 2014 Depto. CCIA All rights reserved.

síntesis de voz.

- El método `shutdown()` permite liberar los recursos reservados por el motor de *Text To Speech*.

El siguiente código muestra un ejemplo en el que se comprueba si todo está correctamente instalado, se inicializa una nueva instancia de la clase `TextToSpeech`, y se utiliza dicha clase para decir una frase en español. Al llamar al método `initTextToSpeech()` se desencadenará todo el proceso.

```
private static int TTS_DATA_CHECK = 1;
private TextToSpeech tts = null;
private boolean ttsIsInit = false;

private void initTextToSpeech() {
        Intent intent = new Intent(Engine.ACTION_CHECK_TTS_DATA);
        startActivityForResult(intent, TTS_DATA_CHECK);
}

protected void onActivityResult(int requestCode, int resultCode, Intent
data) {
        if (requestCode == TTS_DATA_CHECK) {
                if (resultCode == Engine.CHECK_VOICE_DATA_PASS) {
                        tts = new TextToSpeech(this, new OnInitListener()
{
                                public void onInit(int status) {
                                        if (status ==
TextToSpeech.SUCCESS) {
                                                ttsIsInit = true;
                                                Locale loc = new
Locale("es","","");
                                                if
(tts.isLanguageAvailable(loc)
                                                    >=
TextToSpeech.LANG_AVAILABLE)
tts.setLanguage(loc);
                                                tts.setPitch(0.8f);
                                                tts.setSpeechRate(1.1f);
                                                speak();
                                        }
                                }
                        });
                } else {
                        Intent installVoice = new
Intent(Engine.ACTION_INSTALL_TTS_DATA);
                        startActivity(installIntent);
                }
        }
}

private void speak() {
        if (tts != null && ttsIsInit) {
                tts.speak("Hola Android", TextToSpeech.QUEUE_ADD, null);
        }
}

@Override
public void onDestroy() {
        if (tts != null) {
                tts.stop();
                tts.shutdown();
        }
```


Copyright © 2014 Depto. CCIA All rights reserved.

```
        super.onDestroy();
}
```

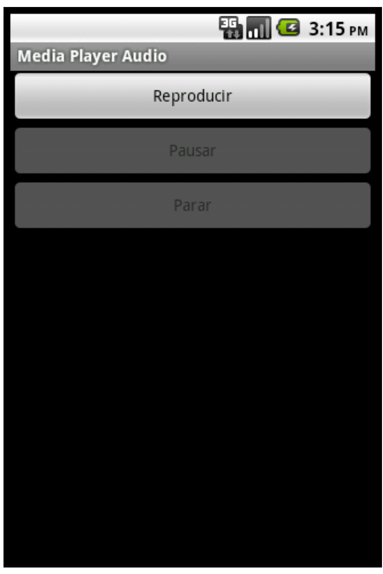

Copyright © 2014 Depto. CCIA All rights reserved.

10. Multimedia - Ejercicios

10.1. Reproducción de un clip de audio

En primer lugar vamos a crear una aplicación simple llamada `Audio` que permita controlar la reproducción de un fichero de audio (proporcionado en las plantillas de la sesión). Estos serán los pasos que deberás seguir:

- En primer lugar crea un directorio `raw` dentro de la carpeta `res` de la aplicación, en la que almacenarás el archivo mp3 proporcionado en las plantillas.
- A continuación modifica el layout de la aplicación principal para mostrar tres botones con las siguientes etiquetas: *Reproducir*, *Pausar* y *Detener*. Al comenzar la aplicación los dos últimos botones estarán desactivados.
- Al pulsar el botón *Reproducir* comenzará la reproducción del clip de audio desde el principio. Dicho botón pasará a estar desactivado, y mientras que los botones *Pausar* y *Detener* se activarán.
- Al pulsar el botón *Pausar* su etiqueta cambiará a *Reiniciar*. La reproducción quedará pausada hasta que se vuelva a pulsar el botón, momento en el que su etiqueta volverá a ser *Pausar*.
- Al pulsar el botón *Detener* se interrumpirá la reproducción del archivo de audio. El botón *Reproducir* quedará activado, mientras que los otros dos volverán a estar desactivados.
- Cuando se complete la reproducción del clip de audio se llevarán a cabo las mismas acciones que en el caso de que se hubiera pulsado el botón *Detener* (ver el punto anterior). Se puede utilizar el método `setOnCompletionListener` para crear un manejador de dicho evento.



Copyright © 2014 Depto. CCIA All rights reserved.

Aspecto de la aplicación Audio

10.2. Reproducción de un clip de vídeo por medio del control Video View

Vamos ahora a crear una aplicación llamada `EjemploVideoView` que permita reproducir un clip de vídeo (proporcionado en las plantillas de la sesión) mediante la clase `VideoView`. El objetivo del ejercicio será crear un reproductor que funcione de la misma manera que el reproductor de audio del ejercicio anterior. Es decir, deberemos crear unos botones *Reproducir*, *Pausar* y *Detener* cuyo funcionamiento sea el especificado anteriormente.

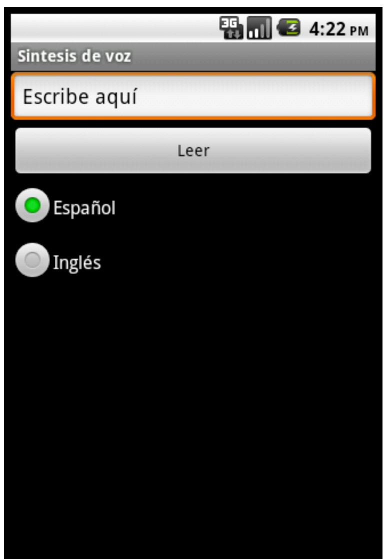
La reproducción del vídeo deberá ser realizada mediante un control de tipo `Video View` que deberá ser añadido al layout junto a los botones para controlar la reproducción.

Aspecto de la aplicación EjemploVideoView

10.3. Síntesis de voz con Text to Speech

La última aplicación que crearemos en esta sesión tendrá como nombre `SintesisVoz`. La aplicación mostrará un campo de edición de texto, un botón cuya etiqueta será *Leer* y dos botones de radio dentro del mismo grupo cuyas etiquetas serán *Español* e *Inglés*.

Copyright © 2014 Depto. CCIA All rights reserved.



Aspecto de la aplicación SintesisVoz

Al pulsar el botón *Leer* la aplicación leerá el texto que se encuentre en el campo de edición de texto. Los botones de radio permitirán escoger la pronunciación de la lectura (en español o en inglés).

Copyright © 2014 Depto. CCIA All rights reserved.

11. Ficheros y acceso a datos

11.1. Ficheros tradicionales

El uso de ficheros tradicionales está permitido para los programas de Android, aunque como veremos más adelante hay otras tecnologías mucho más avanzadas, como es el uso de base de datos y los `SharedPreferences`. En Android un fichero se puede guardar y leer así:

```
FileOutputStream fos = openFileOutput("fichero.txt",
Context.MODE_PRIVATE);
FileInputStream  fis = openFileInput("fichero.txt");
```

El modo `Context.MODE_PRIVATE` hace que el fichero sea privado a la aplicación. Se crearía el fichero si éste no existiera. Para añadir al archivo se utiliza el modo `Context.MODE_APPEND`.

Para escribir en la tarjeta de memoria (SD card) utilizaremos el método `Environment.getExternalStorageDirectory()` y un `FileWriter` de la siguiente manera:

```
try {
    File raiz = Environment.getExternalStorageDirectory();
    if (raiz.canWrite()){
        File file = new File(raiz, "fichero.txt");
        BufferedWriter out = new BufferedWriter(new FileWriter(file));
        out.write("Mi texto escrito desde Android");
        out.close();
    }
} catch (IOException e) {
    Log.e("FILE I/O", "Error en la escritura de fichero: " +
e.getMessage());
}
```

Con tal de poder probar este ejemplo vamos a necesitar un emulador que disponga de una tarjeta SD emulada. Podemos crear un fichero `.iso` que represente dicha tarjeta en la línea de comandos utilizando lo siguiente:

```
mksdcard 512M sdcard.iso
```

Una vez creada la tarjeta SD, y con el emulador en funcionamiento, podemos utilizar el siguiente comando para extraer un fichero de dicha tarjeta y guardarlo en la máquina local:

```
adb pull /sdcard/fichero.txt fichero.txt
```

La operación inversa (copiar un fichero desde la máquina local a la tarjeta SD emulada) se realiza mediante el comando `adb push`.

11.2. Preferencias

Copyright © 2014 Depto. CCIA All rights reserved.

Escribir en ficheros tradicionales puede llegar a ser muy tedioso y en el caso de programas que tengan que guardar preferencias, acaba en código poco mantenible. Para facilitar el almacenamiento de opciones, android nos proporciona un mecanismo conocido como `SharedPreferences`, el cual se utiliza para guardar datos mediante pares de clave y valor.

Este mecanismo podría ser utilizado, por ejemplo, en el manejador `onSaveInstanceState`, el cual es invocado para guardar el estado de la actividad cuando ésta deba ser terminada por parte del sistema operativo por falta de recursos.

11.2.1. Crear, guardar y leer preferencias

Para guardar preferencias comunes de la aplicación hay que crear un editor de la siguiente manera:

```
SharedPreferences pref;
pref =
PreferenceManager.getDefaultSharedPreferences(getApplicationContext());
SharedPreferences.Editor editor = pref.edit();
editor.putBoolean("esCierto",false);
editor.putString("Nombre","Pablo");
editor.commit();
```

Obsérvese como se hace uso de métodos de tipo `put` para asignar a una clave un valor determinado. Para obtener el valor asociado con estas claves desde otra parte de la actividad podríamos hacer uso del siguiente código:

```
boolean esCierto = pref.getBoolean("esCierto",false);
String  nombre   = pref.getString("Nombre", "sin nombre");
```

El segundo parámetro de los métodos `get` es un valor por defecto. Este valor es el devolverá el método en el caso en el que la clave indicada no exista.

Además puede ser interesante programar un Listener para que se ejecute cuando ocurra algún cambio en las preferencias:

```
prefValidacion.registerOnSharedPreferenceChangeListener(
        new OnSharedPreferenceChangeListener() {
                @Override
                public void onSharedPreferenceChanged(
                                SharedPreferences sharedPreferences,
                                String key)
                {
                        //Ha cambiado algo, emprender las acciones
necesarias.
                }
        });
```

11.2.2. Interfaz de usuario para las preferencias

Android permite de una manera sencilla la creación de actividades de preferencias. Cada campo de estas actividades se corresponderá con un par clave y valor de

Copyright © 2014 Depto. CCIA All rights reserved.

SharedPreferences. El primer paso para crear una actividad de modificación de preferencias es definirla mediante un archivo XML en la carpeta /res/xml/, dentro de los recursos de la aplicación. Un ejemplo de archivo de este tipo podría ser el siguiente, al que podríamos llamar preferencias.xml:

```
  <?xml version="1.0" encoding="utf-8"?>
<PreferenceScreen
  xmlns:android="http://schemas.android.com/apk/res/android">
<PreferenceCategory android:title="Validar DNI en:">
        <CheckBoxPreference
        android:title="en el campo"
        android:summary="Validará la introducción de números y una letra"
        android:key="validacampo"></CheckBoxPreference>
        <CheckBoxPreference
        android:title="al pulsar"
        android:summary="Comprobará también que la letra sea la correcta"
        android:key="validaboton"></CheckBoxPreference>
</PreferenceCategory>
<PreferenceCategory android:title="Otras preferencias:">
        <CheckBoxPreference android:enabled="false"
        android:title="Otra, deshabilitada"
        android:key="otra"></CheckBoxPreference>
</PreferenceCategory>
</PreferenceScreen>
```

Las claves android:key deben coinciden con las claves que después podremos leer mediante SharedPreferences. No es necesario crear ningún método que lea de los campos definidos en el XML, al mostrarse la actividad de preferencias, éstas se mapean automáticamente con las preferencias compartidas utilizando las claves.

Para que el XML se cargue en una actividad nueva hay que crear una clase que herede de PreferenceActivity:

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

Para mostrar esta actividad utilizamos, como siempre, un Intent:

```
startActivity(new Intent(this, Preferencias.class);
```

El resultado se muestra en la imagen siguiente:

Copyright © 2014 Depto. CCIA All rights reserved.

Menú de preferencias

11.3. Base de datos SQLite

SQLite es un gestor de bases de datos relacional y de código abierto, que cumple con los estándares, y además es extremadamente ligero. Otra de sus características es que guarda toda la base de datos en un único fichero. Es útil en aplicaciones pequeñas para que no requieran la instalación adicional de un gestor de bases de datos, así como para dispositivos embebidos con recursos limitados. Android incluye soporte a SQLite.

Una manera de separar el código que accede a la base de datos del resto del código es abstraerlo mediante un patrón adaptador que nos permita abrir la base de datos, leer, escribir, borrar, y otras operaciones que nuestro programa pueda requerir. Así, accedemos a métodos de este adaptador, y no tenemos que introducir código SQL en el resto del programa, haciendo además que el mantenimiento de nuestras aplicaciones sea más sencillo.

Para ilustrar el uso de SQLite en Android vamos a ver una clase adaptador de ejemplo. En esta clase (`DataHelper`) abriremos la base de datos de una manera estándar: mediante un patrón `SQLiteOpenHelper`. Esta clase abstracta nos obliga a implementar nuestro propio Helper de apertura de la base de datos, de una manera estándar. Básicamente sirve para que nuestro código esté obligado a saber qué hacer en caso de que la base de datos no exista (normalmente crearla), y cómo portar la base de datos en caso de detectarse un cambio de versión. La versión de la base de datos la indicamos nosotros. En este ejemplo el cambio de versión se implementa de una manera un poco drástica: borrar todos los contenidos y crear una nueva base de datos vacía, perdiendo todo lo anterior. Se trata de sólo un ejemplo, en cualquier aplicación real convendría portar los datos a las nuevas tablas.

Copyright © 2014 Depto. CCIA All rights reserved.

```
package es.ua.jtech.daa.db;

import android.content.Context;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import android.database.sqlite.SQLiteStatement;
import android.util.Log;
import java.util.ArrayList;
import java.util.List;

public class DataHelper {

   // Nombre del fichero en el que se guardará la base de datos
   private static final String DATABASE_NAME = "mibasededatos.db";
   // Versión de la base de datos, indicada por el programador. En el
   // caso de que introduzcamos algún cambio deberíamos modificar este
   // número de versión, con lo que el MiOpenHelper determinará qué hacer
   private static final int DATABASE_VERSION = 1;
   // Nuestra base de datos de ejemplo tiene una única tabla
   private static final String TABLE_NAME = "ciudades";
   // Y esta tabla tiene tres columnas, siendo la primera la clave
   // primaria
   private static final String[] COLUMNAS = {"_id","nombre","habitantes"};

   // Incluimos el código SQL como constantes
   private static final String INSERT = "insert into " + TABLE_NAME +
     "("+COLUMNAS[1]+","+COLUMNAS[2]+") values (?,?)";
   private static final String CREATE_DB = "CREATE TABLE " + TABLE_NAME +
         "("+COLUMNAS[0]+" INTEGER PRIMARY KEY, "
            +COLUMNAS[1]+" TEXT, "
            +COLUMNAS[2]+" NUMBER)";

   // El contexto de la aplicación
   private Context context;
   // La instancia de la base de datos que nos
   // proporcionará el Helper (ya sea abriendo una base de
   // datos ya existente, creándola si no existe, o actualizándola
   // en el caso de algún cambio de versión)
   private SQLiteDatabase db;
   // Este atributo se utilizará durante la inserción
   private SQLiteStatement insertStatement;

   public DataHelper(Context context) {
      this.context = context;
      // Obtenemos un puntero a una base de datos sobre la que poder
      // escribir mediante la clase MiOpenHelper, que es una clase
      // privada definida dentro de DataHelper
      MiOpenHelper openHelper = new MiOpenHelper(this.context);
      this.db = openHelper.getWritableDatabase();

      // La inserción se realizará mediante lo que se conoce mediante
      // una sentencia SQL compilada. Asociamos al objeto insertStatement
      // el código SQL definido en la constante INSERT. Obsérvese que
      // este código SQL se trata de una sentencia SQL genérica,
parametrizada
      // mediante el símbolo ?
      this.insertStatement = this.db.compileStatement(INSERT);
   }

   public long insert(String name, long number) {
      // Damos valor a los dos elementos genéricos (indicados por el
símbolo ?)
      // de la sentencia de inserción compilada mediante bind
      this.insertStatement.bindString(1, name);
```


Copyright © 2014 Depto. CCIA All rights reserved.

```
      this.insertStatement.bindLong(1, number);
      // Y llevamos a cabo la inserción
      return this.insertStatement.executeInsert();
   }

   public int deleteAll() {
             // En este caso hacemos uso de un método de la instancia de
la base
             // de datos para realizar el borrado. Existen también métodos
             // para hacer queries y otras operaciones con la base de
datos
      return db.delete(TABLE_NAME, null, null);
   }

   public List<String> selectAllNombres() {
      List<String> list = new ArrayList<String>();
      // La siguiente instrucción almacena en un cursor todos los valores
      // de las columnas indicadas en COLUMNAS de la tabla TABLE_NAME
      Cursor cursor = db.query(TABLE_NAME, COLUMNAS,
        null, null, null, null, null);
      // El cursor es un iterador que nos permite ir recorriendo
      // los resultados devueltos secuencialmente
      if (cursor.moveToFirst()) {
         do {
                 // Añadimos a la lista que devolveremos como salida
                 // del método el nombre de la ciudad en la posición
actual
            list.add(cursor.getString(1));
         // El método moveToNext devolverá false en el caso de que se
         // haya llegado al final
         } while (cursor.moveToNext());
      }
      if (cursor != null && !cursor.isClosed()) {
         cursor.close();
      }
      return list;
   }

   // Esta clase privada del DataHelper se encarga de proporcionar una
instancia
   // de base de datos a DataHelper sobre la que poder trabajar.
   private static class MiOpenHelper extends SQLiteOpenHelper {
      MiOpenHelper(Context context) {
         super(context, DATABASE_NAME, null, DATABASE_VERSION);
      }

      // Este método se utilizará en el caso en el que la base de datos no
existiera
      @Override
      public void onCreate(SQLiteDatabase db) {
         db.execSQL(CREATE_DB);
      }

      // Este método se ejecutará en el caso en el que se cambie el valor
      // de la constante DATABASE_VERSION. En este caso se borra la base
      // de datos anterior antes de crear una nueva, pero lo ideal sería
      // transferir los datos desde la versión anterior a la nueva
      @Override
      public void onUpgrade(SQLiteDatabase db, int oldVersion, int
newVersion) {
         Log.w("SQL","onUpgrade: eliminando tabla si existe y creándola de
nuevo");
         db.execSQL("DROP TABLE IF EXISTS " + TABLE_NAME);
         onCreate(db);
      }
```


Copyright © 2014 Depto. CCIA All rights reserved.

```
    }
}
```

Obsérvese también el uso del `Cursor` y cómo éste se recorre para obtener los resultados. En este caso sólo se han recogido los nombres de las ciudades. También se podían haber obtenido los valores de los habitantes, pero hubiera hecho falta una estructura de datos más elaborada que mantuviera pares de nombre y número de habitantes. En muchas ocasiones los adaptadores no devuelven una lista tras hacer un select, sino directamente el `Cursor`, así que es tarea del programador que hace uso de dicho adaptador recorrer los resultados apuntados por el cursor.

11.4. Proveedores de contenidos

Los proveedores de contenidos o `ContentProvider` proporcionan una interfaz para publicar y consumir datos, identificando la fuente de datos con una dirección URI que empieza por `content://`. Son una forma más estándar en Android que los adaptadores a una Base de Datos de desacoplar la capa de aplicación de la capa de datos.

11.4.1. Proveedores nativos

Hay una serie de proveedores de contenidos que Android nos proporciona de manera nativa. Entre estos proveedores de contenidos nativos nos encontramos el `Browser`, `CallLog`, `ContactsContract`, `MediaStore`, `Settings`, `UserDictionary`. Para hacer uso de ellos es necesario añadir los permisos correspondientes en el fichero `AndroidManifest.xml`. Por ejemplo, para acceder al listín telefónico, añadiríamos el siguiente permiso:

```
    ...
        <uses-sdk android:minSdkVersion="8" />
        <uses-permission android:name="android.permission.READ_CONTACTS"/>
</manifest>
```

Para acceder a la información proporcionada por cualquier proveedor de contenidos hacemos uso de la clase `ContentResolver`; en concreto, deberemos utilizar su método `query`, que devuelve un `Cursor` apuntando a los datos solicitados:

```
ContentResolver.query(
                  Uri uri,
                  String[] projection,
                  String selection,
                  String[] selectionArgs,
                  String sortOrder)
```

Por ejemplo, para acceder la lista de contactos utilizaríamos el método `query` de la siguiente forma:

```
ContentResolver cr = getContentResolver();
Cursor cursor = cr.query(ContactsContract.Contacts.CONTENT_URI,
                null, null, null, null);
```


Copyright © 2014 Depto. CCIA All rights reserved.

En el ejemplo anterior, la constante `CONTENT_URI` contiene la URI del proveedor de contenidos nativo correspondiente a la lista de contactos. Recuerda que se accede a los datos proporcionados por un proveedor de contenidos siempre a partir de una URI.

Nota:
En versiones anteriores a Android 2.0 las estructuras de datos de los contactos son diferentes y no se accede a esta URI.

Una ventaja de android es que es posible asociar directamente campos de un `Cursor` con componentes de la interfaz de la actividad. En ese caso podemos además hacer que cualquier cambio que ocurra en el cursor se refleje de manera automática (sin tener que programar nosotros el refresco) en el componente gráfico mediante el siguiente código:

```
cursor.setNotificationUri(cr, ContactsContract.Contacts.CONTENT_URI);
```

Por ejemplo, el siguiente código muestra como asignar los datos apuntados por un cursor a un elemento `ListView`. Para ello se le pasa como parámetro al método `setAdapter` del `ListView` una instancia de la clase `SimpleCursorAdapter`:

```
ListView lv = new (ListView)findViewById(R.id.ListView01);
SimpleCursorAdapter adapter = new SimpleCursorAdapter(
        getApplicationContext(),
        R.layout.textviewlayout,
        cursor,
        new String[]{
                ContactsContract.Contacts._ID,
                ContactsContract.Contacts.DISPLAY_NAME},
        new int[]{
                R.id.TextView1,
                R.id.TextView2});
        lv.setAdapter(adapter);
```

En este ejemplo los identificadores `R.id.TextView1` y `R.id.TextView2` se corresponden con vstas del layout que definen cada fila de la `ListView`, como se puede ver en el archivo de ejemplo de layout `textviewlayout.xml` que mostramos a continuación:

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

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
      android:orientation="horizontal"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content">
        <TextView android:id="@+id/TextView1"
          android:textStyle="bold"
          android:ems="2"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content">
        </TextView>
        <TextView android:id="@+id/TextView2"
          android:textStyle="bold"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content">
        </TextView>
</LinearLayout>
```


Copyright © 2014 Depto. CCIA All rights reserved.

La vista de tipo `ListView` (identificada por `R.id.ListView01` en el presente ejemplo) estaría incluida en otro fichero XML de layout en el que se indicaran los componentes de la interfaz de la actividad que contiene dicha lista.

> **Nota:**
> Para obtener los números de teléfono de cada persona habría que recorrer el cursor del ejemplo anterior y, por cada persona, crear un nuevo cursor que recorriera los teléfonos, ya que cabe la posibilidad de que una persona posea más de un número de teléfono.

11.4.2. Proveedores propios

Para acceder a nuestras fuentes de datos propias siguiendo el estándar de diseño Android nos interesa implementar nuestros propios `ContentProvider`. Para ello heredaremos de esta clase y sobrecargaremos una serie de métodos, como `onCreate()`, `delete(...)`, `insert(...)`, `query(...)`, `update(...)`, etc. También debemos sobrecargar el método `getType(Uri)` que nos deolverá el tipo MIME de los contenidos, dependiendo de la URI.

En esta sección vamos a seguir los pasos necesarios para crear un proveedor de contenidos propios que nos permita acceder a nuestra Base de Datos de ciuidades, a partir del adaptador mostrado en la sección de SQLite. Dicho proveedor de contenidos se implementará por medio de la clase `CiudadesProvider`.

La URI base la declararemos en una constante pública de nuestro proveedor y será de tipo `Uri`. Recuerda que la URI es el mecanismo mediante el cual indicamos el origen de los datos. Un ejemplo de URI base podría ser la siguiente:

```
public static final Uri CONTENT_URI =
          Uri.parse("content://es.ua.jtech.daa.proveedores/ciudades");
```

Sobre esta URI base se pueden construir otras URIs que nos permitan especificar ciertos aspectos en cuanto a qué datos se desea acceder. Por ejemplo, para diferenciar entre el acceso a una única fila de datos o a múltiples filas, se pueden emplear números, por ejemplo, "1" si se accede a todas las filas, y "2" si se busca una concreta. En este último caso, también habría que especificar el ID de la fila que se busca.

- Todas las filas: `content://es.ua.jtech.daa.proveedores/ciudades/1`
- Una fila: `content://es.ua.jtech.daa.proveedores/ciudades/2/23`

Para distinguir entre una URI y otra declaramos un `UriMatcher` constante que inicializamos de forma estática en nuestra clase proveedora de contenidos.

```
private static final UriMatcher uriMatcher;
static{
        uriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
        uriMatcher.addURI("es.ua.jtech.daa.proveedores", "ciudades",
ALLROWS);
        uriMatcher.addURI("es.ua.jtech.daa.proveedores", "ciudades/#",
SINGLE_ROW);
```


Copyright © 2014 Depto. CCIA All rights reserved.

```
}
```

Así, dentro de cada función que sobrecarguemos primero comprobaremos qué resultado debemos devolver, usando una estructura `switch`:

```
@Override
public String getType(Uri uri) {
        switch(uriMatcher.match(uri)){
                case ALLROWS: return
"vnd.ua.cursor.dir/ciudadesprovidercontent";
                case SINGLE_ROW: return
"vnd.ua.cursor.item/ciudadesprovidercontent";
                default: throw new IllegalArgumentException("URI no
soportada: "+uri);
        }
}
```

El código anterior muestra un ejemplo de sobrecarga de `getType(Uri)`, el cual devuelve dos tipos MIME diferentes según el caso. La primera parte de la cadena, antes de la barra, debe terminar en ".dir" si se trata de más de una fila y en .item si se trata de una sola. La primera parte de la cadena debe comenzar por "vnd." y a continuación debe ir el nombre base del dominio sin extensión, en el caso de "www.jtech.ua.es", sería "ua". Este identificador debe ir seguido de ".cursor", ya que los datos se devolverán en un cursor. Por último, después de la barra se suele indicar nombre de la clase seguido de "content", en este caso es "ciudadesprovidercontent" porque la clase se llama `CiudadesProvider`.

Para poder usar el proveedor de contenidos propios, éste se debe declarar en el `AndroidManifest.xml`, dentro del elemento `application`.

```
<provider
        android:name="CiudadesProvider"
        android:authorities="es.ua.jtech.daa.proveedores" />
```

Veamos a continuación el código completo de la clase `CiudadesProvider`. Observaremos muchas similitudes con el adaptador para la base de datos de la sección sobre SQLite, ya que al fin y al cabo se trata de otra interfaz diferente (pero más estándar) para acceder a los mismos datos. También volvemos a utilizar el mismo Helper para abrir la base de datos.

```
package es.ua.jtech.daa.proveedores;

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

public class CiudadesProvider extends ContentProvider {

        //Campos típicos de un ContentProvider:
        public static final Uri CONTENT_URI = Uri.parse(
```


Copyright © 2014 Depto. CCIA All rights reserved.

```
                "content://es.ua.jtech.daa.proveedores/ciudades");
        private static final int ALLROWS = 1;
        private static final int SINGLE_ROW = 2;
        private static final UriMatcher uriMatcher;
        static{
          uriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
          uriMatcher.addURI("es.ua.jtech.daa.proveedores", "ciudades",
ALLROWS);
          uriMatcher.addURI("es.ua.jtech.daa.proveedores", "ciudades/#",
SINGLE_ROW);
        }

        // Campos concretos de nuestro ContentProvider, haciendo
referencia a nuestra
        // base de datos
        public static final String DATABASE_NAME = "mibasededatos.db";
        public static final int DATABASE_VERSION = 1;
        public static final String TABLE_NAME = "ciudades";
        private static final String[] COLUMNAS =
{"_id","nombre","habitantes"};
        private static final String CREATE_DB = "CREATE TABLE " +
TABLE_NAME +
         "("+COLUMNAS[0]+" INTEGER PRIMARY KEY, "
            +COLUMNAS[1]+" TEXT, "
            +COLUMNAS[2]+" NUMBER)";

        // Contexto de la aplicación
        private Context context;
        // Instancia de la base de datos
        private SQLiteDatabase db;

        @Override
        public boolean onCreate() {
                this.context = getContext();
                MiOpenHelper openHelper = new MiOpenHelper(this.context);
                this.db = openHelper.getWritableDatabase();
                return true;
        }

        @Override
        public String getType(Uri uri) {
                switch(uriMatcher.match(uri)){
                case ALLROWS: return
"vnd.ua.cursor.dir/ciudadesprovidercontent";
                case SINGLE_ROW: return
"vnd.ua.cursor.item/ciudadesprovidercontent";
                default: throw new IllegalArgumentException("URI no
soportada: "+uri);
                }
        }

        @Override
        public int delete(Uri uri, String selection, String[]
selectionArgs) {
                int changes = 0;
                switch(uriMatcher.match(uri)){
                case ALLROWS:
                        changes = db.delete(TABLE_NAME, selection,
selectionArgs);
                        break;
                case SINGLE_ROW:
                        String id = uri.getPathSegments().get(1);
                        Log.i("SQL", "delete the id "+id);
                        changes =  db.delete(TABLE_NAME,
                                "_id = " + id +
```


Copyright © 2014 Depto. CCIA All rights reserved.

```
                                        (!TextUtils.isEmpty(selection) ?
                                                " AND (" + selection + ')'
:
                                                ""),
                                selectionArgs);
                        break;
                default: throw new IllegalArgumentException("URI no
soportada: "+uri);
                }
                context.getContentResolver().notifyChange(uri, null);
                return changes;
        }

        @Override
        public Uri insert(Uri uri, ContentValues values) {
                long id = db.insert(TABLE_NAME, COLUMNAS[1], values);
                if(id > 0 ){
                        Uri uriInsertado =
ContentUris.withAppendedId(CONTENT_URI, id);
context.getContentResolver().notifyChange(uriInsertado, null);
                        return uriInsertado;
                }
                throw new android.database.SQLException(
                        "No se ha podido insertar en "+uri);
        }

        @Override
        public Cursor query(Uri uri, String[] projection, String
selection,
                        String[] selectionArgs, String sortOrder) {
                return db.query(TABLE_NAME, COLUMNAS, selection,
selectionArgs,
                        null, null, null);
        }

        @Override
        public int update(Uri uri, ContentValues values, String selection,
                        String[] selectionArgs) {
                int changes = 0;
                switch(uriMatcher.match(uri)){
                case ALLROWS:
                        changes =db.update(TABLE_NAME, values, selection,
selectionArgs);
                        break;
                case SINGLE_ROW:
                        String id = uri.getPathSegments().get(1);
                        Log.i("SQL", "delete the id "+id);
                        changes =  db.update(TABLE_NAME, values,
                                "_id = " + id +
                                        (!TextUtils.isEmpty(selection) ?
                                                " AND (" + selection + ')'
                                                : ""),
                                selectionArgs);
                        break;
                default: throw new IllegalArgumentException("URI no
soportada: "+uri);
                }
                context.getContentResolver().notifyChange(uri, null);
                return changes;
        }

        private static class MiOpenHelper extends SQLiteOpenHelper {

                MiOpenHelper(Context context) {
                        super(context, DATABASE_NAME, null,
```


Copyright © 2014 Depto. CCIA All rights reserved.

```
DATABASE_VERSION);
                }

                @Override
                public void onCreate(SQLiteDatabase db) {
                        db.execSQL(CREATE_DB);
                }

                @Override
                public void onUpgrade(SQLiteDatabase db, int oldVersion,
int newVersion){
                        Log.w("SQL", "onUpgrade: eliminando tabla si ésta
existe,"+
                                " y creándola de nuevo");
                        db.execSQL("DROP TABLE IF EXISTS " + TABLE_NAME);
                        onCreate(db);
                }
        }

}
```

Préstese también atención a las líneas

```
context.getContentResolver().notifyChange(uri, null);
```

Es necesario notificar los cambios al `ContentResolver` para así poder actualizar los cursores que lo hayan pedido a través del método `Cursor.setNotificationUri(...)`, como en el anterior ejemplo del listín telefónico que se mostraba en un componente de la interfaz gráfica.

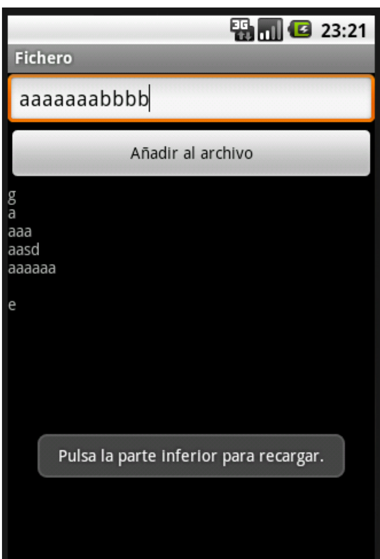
Copyright © 2014 Depto. CCIA All rights reserved.

12. Ficheros y acceso a datos - Ejercicios

12.1. Escribir en un archivo de texto

Vamos a leer y escribir en un archivo de texto. La escritura la haremos añadiendo al final del archivo, mientras que la lectura la haremos leyendo línea a línea.

- Crear un proyecto Fichero cuya única actividad, la principal, mostrará un `EditText` cuyo texto será recogido cada vez que se pulse un `Button` que habrá debajo. Dicho texto será añadido a un archivo de texto llamado `mytextfile.txt`. Para añadir texto a un archivo primero lo abriremos con `openFileOutput(FILENAME,Context.MODE_APPEND))` y después utilizaremos el método `append(...)` de `OutputStreamWriter`.
- ¿Cuándo abrimos el archivo? Recordemos que la actividad puede pasar a inactiva en cualquier momento y que puede no volver a recuperarse. ¿Cuándo nos conviene cerrar el archivo?
- Debajo del campo de texto y del botón, vamos a añadir un `TextEdit` que ocupe el resto de la pantalla. Lo haremos pulsable con el método `setClickable(true)` y cada vez que se haga click sobre él, leeremos el archivo línea a línea y lo mostraremos entero.



Programa que escribe y lee un fichero de texto

12.2. Crear y utilizar un DataHelper para SQLite

En las plantillas de la sesión tenemos el esqueleto de un DataHelper. Se trata de un patrón

Copyright © 2014 Depto. CCIA All rights reserved.

de diseño que nos ayuda a acceder a nuestros datos encapsulando todo el manejo de la base de datos en el `DataHelper`.

El `DataHelper` utiliza a su vez otro patrón de diseño, el `SQLiteOpenHelper`. Éste nos obliga a definir qué ocurre cuando la base de datos todavía no existe y debe ser creada, y qué ocurre si ya existe pero debe ser actualizada porque ha cambiado de versión. Así el `SQLiteOpenHelper` que implementemos, en este caso `MiOpenHelper`, nos devolverá siempre una base de datos separándonos de la lógica encargada de comprobar si la base de datos existe o no.

Se pide:

- Ejecutar la sentencia de creación de bases de datos (la tenemos declarada como constante de la clase) en el método `MiOpenHelper.onCreate(...)`.
- Implementar también el método onUpgrade. Idealmente éste debería portar las tablas de la versión antigua a la versión nueva, copiando todos los datos. Nosotros vamos a eliminar directamente la talba que tenemos con la sentencia SQL `"DROP TABLE IF EXISTS " + TABLE_NAME` y volveremos a crearla.
- En el constructor del `DataHelper` debemos obtener en el campo `db` la base de datos, utilizando `MiOpenHelper`.

También debemos completar código relacionado con las sentencias de inserción y de borrado.

- En el constructor del `DataHelper`, una vez obtenida la base de datos a través del helper, utilizarla para compilar la sentencia `INSERT` que tenemos como constante String.
- Completar la función `insert()` introduciendo el nombre en la sentencia compilada (con la función `bindString`) y ejecutándola. Devolverá el número de filas afectadas, que a su vez se devolverá con el `return`.
- Completar la función `deleteAll()` que también devolverá el número de filas afectadas.

Ahora podemos utilizar el helper para leer y escribir en la base de datos de forma transparente.

- En el `Main` debemos probar el borrado, inserción y listado de datos.
- Comprobemos por línea de comandos que la base de datos está creada. Serán útiles los siguientes comandos:

```
adb -e shell
#cd /data/data/es.ua.jtech.daa/databases
#sqlite3 misusuarios.db
sqlite> .schema
sqlite> .tables
sqlite> select * from usuarios;
```

- Ahora vamos a cambiar en el `DataHandler` el nombre de la segunda columna, en lugar de nombre, se va a llamar nombres. Ejecutamos la aplicación y comprobamos que sale con una excepción. Se pide comprobar en el Log cuál ha sido el error. ¿Cómo

Copyright © 2014 Depto. CCIA All rights reserved.

lo podemos solucionar?

> **Nota:**
> Pista: conforme hemos programado el `DataHandler` y siguiendo el patrón de diseño de `SQLiteOpenHelper`, podemos arreglar el problema tocando sólo una tecla.

Opcional:

- Añade al helper la función para eliminar por nombre de usuario.
- Añade al layout un spinner para que se pueda seleccionar qué usuario eliminar, y un botón para eliminarlo. Añade también un campo de texto para la introducción de nuevos, y un botón de inserción de nuevo usuario.

12.3. Proveedor de contenidos propio

Vamos a implementar otra forma de acceder a la misma base de datos, siguiento esta vez el patrón de diseño ContentProvider de Android. Seguiremos trabajando con el proyecto del ejercicio anterior.

- Creamos una nueva clase llamada `UsuariosProvider` que herede de `ContentProvider`. Nos obligará a sobrecargar una serie de métodos abstractos. Antes de implementar la `query` vamos a configurar el provider:
- Añadimos algunos campos típicos de los content provider:

```
public static final Uri CONTENT_URI =
        Uri.parse("content://es.ua.jtech.daa/usuarios");
private static final int ALLROWS = 1;
private static final int SINGLE_ROW = 2;
private static final UriMatcher uriMatcher;
static{
        uriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
        uriMatcher.addURI("es.ua.jtech.daa", "usuarios", ALLROWS);
        uriMatcher.addURI("es.ua.jtech.daa", "usuarios/#", SINGLE_ROW);
}
```

- Vamos a acceder a la misma base de datos que el ejercicio anterior, pero no vamos a hacerlo a través del helper que tuvimos que implementar, sino que vamos a copiar de él el código que nos haga falta. Copia los los campos que definen el nombre de la base de datos, de la tabla, de las columnas, la versión, así como la referencia al contexto y a la base de datos. La sentencia compilada del insert ya no va a hacer falta. Inicializa los valores que haga falta en el constructor. Para inicializar la referencia a la base de datos vamos a utilizar, una vez más, `MiOpenHelper`. Podemos copiarlo del helper del ejercicio anterior al `UsuariosProvider`.
- Implementa de forma apropiada el `getType` para devolver un tipo MIME diferente según si se trata de una URI de una fila o de todas las filas. Para ello ayúdate del `uriMatcher`.
- Implementa el método `query`. Simplemente se trata de devolver el cursor que obtenemos al hacer una query a la base de datos SQLite. Algunos de los parámetros que le pasaremos los recibimos como parámetros del método del provider. Los que no

Copyright © 2014 Depto. CCIA All rights reserved.

tengamos irán con valor `null`.

- Aunque no tenemos todos los métodos del `UsuariosProvider` implementados, podemos probarlo. Para ello debemos registarlo en el `AndroidManifest.xml`:

```
 ...
               <provider android:name="UsuariosProvider"
                       android:authorities="es.ua.jtech.daa"/>
    </application>
    <uses-sdk android:minSdkVersion="8" />
</manifest>
```

- En `Main` del ejercicio anterior se insertan una serie de valores con el helper y se muestran en el campo de texto. Manteniendo este código, vamos a añadir al campo de texto el resultado obtenido con la `query` del `UsuariosProvider` para comprobar que tanto el helper como el provider nos devuelven el mismo resultado.

12.4. ¿Por qué conviene crear proveedores de contenidos?

Porque es la forma estándar que establece Android de acceder a contenidos. Además, el content provider nos permitirá notificar al `ContentResolver` de los cambios ocurridos. Así componentes en la pantalla podrán refrescarse de forma automática.

- Descarga las plantillas y utiliza el proyecto `ProveedorContenidos`. Implementa la inserción en el proveedor de contenidos. Pruébala insertando algunos usuarios de ejemplo en el `Main`. Implementa también el `OnClickListener` del botón que inserta nuevos usuarios. El nombre del nuevo usuario irá indicado en el `EditText`.
- Comprueba que la inserción funciona y que, gracias a la línea

```
cursor.setNotificationUri(cr, UsuariosProvider.CONTENT_URI);
```

  y gracias a que usamos un `ContentProvider`, la lista se actualiza automáticamente cuando ocurre algún cambio, sin necesidad de pedir explícitamente la actualización al pulsar el botón.
- Implementa el método `delete` del proveedor de contenidos. Pruébalo en el `Main`. Termina de implementar el `onCreateContextMenuListener` que se ejecutará cada vez que se haga una pulsación larga sobre alguna entrada de la lista. Comprueba que funciona (eliminando el usuario correspondiente, y no otro).
- Opcional: Añade un campo más, por ejemplo "permisos" a la tabla y al proveedor de contenidos.

12.5. Proveedores nativos

Entre los proveedores de contenidos nativos nos encontramos el `Browser`, `CallLog`, `ContactsContract`, `MediaStore`, `Settings`, `UserDictionary`. En este ejercicio vamos a acceder a los contactos. Para reutilizar la tabla, vamos a copiar el proyecto anterior, `ProveedorContenidos` y lo vamos a pegar con nuevo nombre, `ProveedoresPropios`. Para cambiar el nombre de la aplicación tenemos que editar el recurso `strings.xml`.

- Necesitamos permisos para acceder a los contactos. Se añaden en el

Copyright © 2014 Depto. CCIA All rights reserved.

`AndroidManifest.xml`:

```
    ...
    <uses-sdk android:minSdkVersion="8" />
        <uses-permission android:name="android.permission.READ_CONTACTS"/>
</manifest>
```

- Elimina las acciones de los eventos de inserción y borrado.
- Cambia la query para que acceda a ContactsContract.Contacts.CONTENT_URI, así como la uri de notificación del cursor.
- En el `adapter` mapea los mismos campos de texto del anterior ejercicio (Id y Nombre) a las columnas `new String[]{ContactsContract.Contacts._ID, ContactsContract.Contacts.DISPLAY_NAME}`.
- Para comprobar que funciona correctamente deberás introducir algunos números de teléfono desde la agenda de Android.
- Opcional: Una vez que tengas el ID de un contacto, puedes acceder a sus números de teléfono que se encuentran en otra tabla, `ContactsContract.Data` y necesitarías otro cursor diferente para recorrerlos.

> **Nota:**
> Una forma mucho más directa de acceder desde una aplicación a los contactos es lanzando la actividad de los contactos, nativa de Android.

Copyright © 2014 Depto. CCIA All rights reserved.

13. Servicios de red

En esta sesión se exponen varios casos típicos de uso de las conexiones de red. Antes de pasar a implementar una aplicación real es muy importante ver la última sección titulada "Operaciones lentas", pues todas las operaciones de red son lentas.

Para todas las conexiones por internet necesitaremos declarar los permisos en el `AndroidManifest.xml`, fuera del `application` tag:

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

13.1. Conexiones HTTP

Las conexiones por HTTP son las más comunes en las comunicaciones de red. En Android podemos utilizar la clase `HttpURLConnnection` en combinación con `Url`. Podemos ver información de las cabeceras de HTTP como se muestra a continuación (la información se añade a un TextView).

```
        TextView textView = (TextView)findViewById(R.id.TextView01);
        textView.setText("Conexión http.\n\n");
        try {
                textView.setText("Cabeceras www.ua.es:\n");
                URL url = new URL("http://www.ua.es");
                HttpURLConnection http =
(HttpURLConnection)url.openConnection();
                textView.append(" longitud =
"+http.getContentLength()+"\n");
                textView.append(" encoding =
"+http.getContentEncoding()+"\n");
                textView.append(" tipo = "+http.getContentType()+"\n");
                textView.append(" response code =
"+http.getResponseCode()+"\n");
                textView.append(" response message =
"+http.getResponseMessage()+"\n");
                textView.append(" content = "+http.getContent()+"\n");
        } catch (MalformedURLException e) {
        } catch (IOException e) {
        }
```

13.2. Parsing de XML

En las comunicaciones por red es muy común transmitir información en formato XML, el ejemplo más conocido, depués del HTML, son las noticias RSS. En este último caso, al delimitar cada campo de la noticia por tags de XML se permite a los diferentes clientes lectores de RSS obtener sólo aquellos campos que les interese mostrar.

Android nos ofrece dos maneras de trocear o "parsear" XML. El `SAXParser` y el

Copyright © 2014 Depto. CCIA All rights reserved.

XmlPullParser. El parser SAX requiere la implementación de manejadores que reaccionan a eventos tales como encontrar la apertura o cierre de una etiqueta, o encontrar atributos. Menos implementación requiere el uso del parser Pull que consiste en iterar sobre el árbol de XML (sin tenerlo completo en memoria) conforme el código lo va requiriendo, indicándole al parser que tome la siguiente etiqueta (método `next()`) o texto (método nextText()).

A continuación mostramos un ejemplo sencillo de uso del `XmlPullParser`. Préstese atención a las sentencias y constantes resaltadas, para observar cómo se identifican los distintos tipos de etiqueta, y si son de apertura o cierre. También se puede ver cómo encontrar atributos y cómo obtener su valor.

```
        try {
                URL text = new URL("http://www.ua.es");

                XmlPullParserFactory parserCreator =
XmlPullParserFactory.newInstance();
                XmlPullParser parser = parserCreator.newPullParser();
                parser.setInput(text.openStream(), null);
                int parserEvent = parser.getEventType();
                while (parserEvent != XmlPullParser.END_DOCUMENT) {

                        switch (parserEvent) {
                        case XmlPullParser.START_DOCUMENT:
                                break;
                        case XmlPullParser.END_DOCUMENT:
                                break;
                        case XmlPullParser.START_TAG:
                                String tag = parser.getName();
                                if (tag.equalsIgnoreCase("title")) {
                                        Log.i("XML","El titulo es: "+
parser.nextText());
                                }else if(tag.equalsIgnoreCase("meta")){
                                        String name =
parser.getAttributeValue(
                                                null, "name");
if(name.equalsIgnoreCase("description")){
                                                Log.i("XML","La
descripción es:"+
parser.getAttributeValue(
null,"content"));
                                        }
                                }
                                break;
                        case XmlPullParser.END_TAG:
                                break;
                        }

                        parserEvent = parser.next();
                }
        } catch (Exception e) {
                Log.e("Net", "Error in network call", e);
        }
```

El ejemplo anterior serviría para imprimir en el LogCat el título del siguiente fragmento de página web, que en este caso sería "Universidad de Alicante", y para encontrar el `meta` cuyo atributo `name` sea "Description", para mostrar el valor de su atributo `content`:

Copyright © 2014 Depto. CCIA All rights reserved.

```
<html xmlns="http://www.w3.org/1999/xhtml" lang="es" xml:lang="es">
<head>
<title>Universidad de Alicante</title>
<meta name="Description" content="Informacion Universidad Alicante.
Estudios,
        masteres, diplomaturas, ingenierias, facultades, escuelas" />
<meta http-equiv="pragma" content="no-cache" />
<meta name="Author" content="Universidad de Alicante" />
<meta name="Copyright" content="&copy; Universidad de Alicante" />
<meta name="robots" content="index, follow" />
```

13.3. Cargar imágenes de red

Otro caso típico en el trabajo con HTTP es el de cargar imágenes para almacenarlas o bien mostrarlas. En el siguiente código descargamos una imagen a partir de su url, y la mostramos en un `ImageView`:

```
ImageView imageView =
(ImageView)convertView.findViewById(R.id.FilaImagen);
try{
        InputStream is= new
URL("http://www.ua.es/css/imagenes/logoua.gif").openStream();
        Drawable imagen = new
BitmapDrawable(BitmapFactory.decodeStream(is));
        imageView.setImageDrawable(imagen);
}catch(MalformedURLException e1){
}catch(IOException e2){
}
```

El `ImageView` se define, en el layout, así:

```
    <ImageView
        android:id="@+id/ImageView01"
        android:layout_width="wrap_content"
        android:layout_height="fill_parent"
        android:layout_marginRight="10dip"
        android:src="@drawable/icon" />
```

13.4. Estado de la red

En algunas aplicaciones puede convenir comprobar el estado de red. El estado de red no es garantía de que la conexión vaya a funcionar, pero sí que puede prevenirnos de intentar establecer una conexión que no vaya a funcionar. Por ejemplo, hay aplicaciones que requieren el uso de la WIFI para garantizar mayor velocidad.

A continuación se muestra cómo usar el `ConnectivityManager` para comprobar el estado de red.

```
        ConnectivityManager cm = (ConnectivityManager)
                        getSystemService(Context.CONNECTIVITY_SERVICE);
        NetworkInfo wifi =
cm.getNetworkInfo(ConnectivityManager.TYPE_WIFI);
```


Copyright © 2014 Depto. CCIA All rights reserved.

```
        NetworkInfo mobile =
cm.getNetworkInfo(ConnectivityManager.TYPE_MOBILE);
        boolean hayWifi = wifi.isAvailable();
        boolean hayMobile = mobile.isAvailable();
        boolean noHay = (!hayWifi && !hayMobile);
//¡¡Iiiinnteerneeeeeeer!!
```

El `ConnectivityManager` también puede utilizarse para controlar el estado de red, o bien estableciendo una preferencia pero permitiéndole usar el tipo de conectividad que realmente esté disponible,

```
        cm.setNetworkPreference(NetworkPreference.PREFER_WIFI);
```

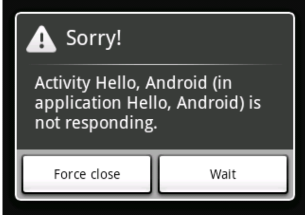
o bien pidiéndole explícitamente que se desconecte de la red móvil y se conecte a la red WiFi:

```
        cm.setRadio(NetworkType.MOBILE,false);
        cm.setRadio(NetworkType.WIFI,true);
```

13.5. Operaciones lentas

En Internet no se puede asumir que ninguna operación de red vaya a ser rápida o vaya a durar un tiempo limitado (el limite lo establece, en todo caso, el timeout de la conexión). En los dispositivos móviles, todavía menos, ya que continuamente pierden calidad de la señal o pueden cambiar de Wifi a 3G sin preguntarnos, y perder conexiones o demorarlas durante el proceso.

Si una aplicación realiza una operación de red en el mismo hilo de la interfaz gráfica, el lapso de tiempo que dure la conexión, la interfaz gráfica dejará de responder. Este efecto es indeseable ya que el usuario no lo va a comprender, ni aunque la operación dure sólo un segundo. Es más, si la congelación dura más de dos segundos, es muy probable que el sistema operativo muestre el diálogo ANR, "Application not responding", invitando al usuario a matar la aplicaicón:



Mensaje ANR

Para evitar esto hay que crear otro hilo (`Thread`) de ejecución. Crearlo puede ser tan sencillo como:

Copyright © 2014 Depto. CCIA All rights reserved.

```
ImageView imageView = (ImageView)findViewById(R.id.ImageView01);
new Thread(new Runnable() {
  public void run() {
    Drawable imagen = cargarLaImagen("http://...");
    //Desde aquí NO debo acceder a imageView
  }
}).start();
```

Pero hay un problema: tras cargar la imagen no puedo acceder a la interfaz gráfica porque la GUI de Android sigue un modelo de hilo único: sólo un hilo puede acceder a ella. Se puede solventar de varias maneras. Una es utilizar el método `View.post(Runnable)`.

```
ImageView imageView = (ImageView)findViewById(R.id.ImageView01);
new Thread(new Runnable() {
  public void run() {
    Drawable imagen = cargarLaImagen("http://...");
    imageView.post(new Runnable() {
      public void run() {
        imageView.setDrawable(imagen);
      }
    });
  }
}).start();
```

Otra manera es utilizar una `AsyncTask`. Es una clase creada para facilitar el trabajo con hilos y con interfaz gráfica, y es muy útil para ir mostrando el progreso de una tarea larga, durante el desarrollo de ésta. Nos facilita la separación entre tarea secundaria e interfaz gráfica permitiéndonos solicitar un refresco del progreso desde la tarea secundaria, pero realizarlo en el hilo principal.

```
TextView textView;
ImageView[] imageView;

public void bajarImagenes(){
        textView = (TextView)findViewById(R.id.TextView01);
        imageView[0] = (ImageView)findViewById(R.id.ImageView01);
        imageView[1] = (ImageView)findViewById(R.id.ImageView02);
        imageView[2] = (ImageView)findViewById(R.id.ImageView03);
        imageView[3] = (ImageView)findViewById(R.id.ImageView04);

        new BajarImagenesTask().execute(
                "http://a.com/1.png",
                "http://a.com/2.png",
                "http://a.com/3.png",
                "http://a.com/4.png");
}
private class BajarImagenesTask extends AsyncTask<String, Integer,
List<Drawable>> {
     @Override
     protected List<Drawable> doInBackground(String... urls) {
              ArrayList<Drawable> imagenes = new ArrayList<Drawable>();
              for(int i=1;i<urls.length; i++){
                cargarLaImagen(urls[0]);
                publishProgress(i);
         return imagenes;
     }
```


Copyright © 2014 Depto. CCIA All rights reserved.

```
    @Override
    protected void onPreExecute() {
      super.onPreExecute();
      textView.setText("Cargando imagenes...");
    }

    @Override
    protected void onProgressUpdate(String... values) {
            textView.setText(values[0] + " imagenes cargadas...");
    }

    @Override
    protected void onPostExecute(List<Drawable> result) {
      for(int i=0; i<result.length; i++){
        imageView[i].setDrawable(result.getItemAt(i));
      }
      textView.setText("Descarga finalizada");
    }

    @Override
    protected void onCancelled() {
            textView.setText("Cancelada la descarga");
    }
}
```

Nota:

La notación `(String ... values)` indica que hay un número indeterminado de parámetros, y se accede a ellos con `values[0], values[1]`, ..., etcétera. Forma parte de la sintaxis estándar de Java.

Lo único que se ejecuta en el segundo hilo de ejecución es el bucle del método `doInBackground(String...)`. El resto de métodos se ejecutan en el mismo hilo que la interfaz gráfica. La petición de publicación de progreso, `publishProgress(...)` está resaltada, así como la implementación de la publicación del progreso, `onProgressUpdate(...)`. Es importante entender que la ejecución de `onProgressUpdate(...)` no tiene por qué ocurrir inmediatamente después de la petición `publishProgress(...)`, o puede incluso no llegar a ocurrir.

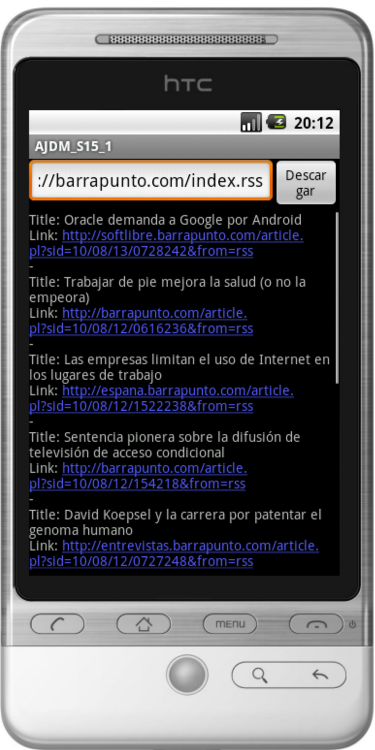
Copyright © 2014 Depto. CCIA All rights reserved.

14. Servicios de red - Ejercicios

14.1. Cargar y trocear XML

En este ejercicio vamos a hacer un sencillo lector de RSS. En las plantillas de la sesión tenemos el proyecto ParseXML que viene con la interfaz de usuario perparada. Se trata de un `EditText` donde se introducirá la URL, y cuando se pulse el botón o bien se pulse la tecla enter habrá que descargar el XML desde la URL indicada, trocearlo, y mostrar los resultados en el `TextView` que ocupa el resto de la pantalla.

- Añade al `AndroidManifest.xml` los permisos para utilizar Internet.
- Completa el método `accionDescargar()` que se encargará de recoger la URL del `EditText` y de introducir los resultados en el `TextView`.
- Necesitarás buscar tags `item`, y dentro de cada uno de ellos habrá tags `title` y `link` cuyo texto tendrás que añadir al resultado. Si no buscas las aperturas y cierres de `item`, también se mostraría el título de la propia página, que en realidad no es ninguna noticia que queramos mostrar.
- Opcional: Muestra también la fecha de cada noticia



Sencillo lector de RSS

Copyright © 2014 Depto. CCIA All rights reserved.

14.2. Operaciones largas de caraga de datos

Si la página web tardara en cargarse, el programa del ejercicio anterior se quedaría congelado durante el tiempo de la descarga. Si éste fuera superior a 2 segundos, podríamos obtener un mensaje de que la aplicación no responde y si deseamos cerrarla. Este tipo de experiencia de usuario es totalmente indeseable

El emulador nos permite simular conexiones lentas de la siguiente manera. Desde línea de comandos:

```
telnet localhost 5554
network delay gprs
OK
network speed gsm
OK
```

Vamos a intentar cargar los RSS de un sitio lento, o bien creernos los efectos indeseables de las cargas lentas.

La solución a las operaciones largas es la creacion de hilos o `Thread`. El problema de los hilos es que los componentes gráficos sólo deben ser modificados desde su propio hilo, y no desde otro. Por tanto hay que separar el código de la tarea que va en el segundo hilo, y de la modificación de los componentes, que debe ejecutarse en el hilo principal.

A partir de Android 1.5 se introdujo una nueva clase que nos facilita la programación de tareas en segundo plano: `AsyncTask`.

- Vamos a modificar el código del ejercicio anterior, introduciendo la carga y parseo de XML en el método `doInBackground(...)` de una nueva clase que extienda a `AsyncTask`.
- Tendremos que separar la actualización de componentes y pasarla a los métodos `onPostExecute(...)` y `onProgressUpdate(...)`. Una forma sería ir cargando el resultado en un campo String que declaramos en la clase, para finalmente actualizar el resultado en el campo de texto. También se puede ir actualizando conforme se va cargando.
- Añadir un campo de texto en la parte superior de la pantalla. En él se indicará el progreso de la descarga, por tanto será actualizado por el método `onProgressUpdate(...)`. Este método deberá ser llamado varias veces a lo largo del progreso de la carga, y lo tenemos que hacer explícitamente desde el método `doInBackground(...)`, con una llamada a `publishProgress("Cargando item número "+n)`.
- Finalmente para iniciar la tarea hay que crear un objeto de la nueva clase que hemos creado, y ejecutar su método `execute(...)`, pasándole como parámetro la dirección de la RSS. Si hemos hecho los anteriores puntos, veremos el progreso de la carga en el nuevo campo de texto que hemos añadido.

Copyright © 2014 Depto. CCIA All rights reserved.

14.3. Lector de RSS - versión gráfica

En este ejercicio vamos a ampliar el código del anterior. Vamos a organizar las noticias en una lista de objetos `Noticia`. Después vamos a crear un adaptador para que las noticias puedan ser cargadas en un `ListView`. Aprovecharemos para añadir una imagen si la hay.

Utilizad el proyecto LectorRSS que hay en las plantillas de la sesión.

Lector de RSS con imágenes

- Crea una clase `Noticia` con los siguientes campos:

```
        private String titulo;
        private String descripcion;
        private String link;
        private String fecha;
        private String linkImagen;
        private Drawable imagen;
```

  Genera getters y setters para todos ellos, y un constructor que los inicialice a "" y a null la imagen. Añade también un método `public void loadImagen(String url)`. Escribe en él el código que carga en el campo `imagen` un `BitmapDrawable` cargado desde la URL indicada. (Se puede utilizar un `InptuStream` que se obtenga a partir de un objeto de tipo `URL`).
- Copia en el método `doInBackground(...)` el código que parsea el XML y rellena la noticia con los campos de Título, Fecha, Descripción, Link, LinkImagen. Hay que

Copyright © 2014 Depto. CCIA All rights reserved.

tener en cuenta que puede haber diferentes formatos de RSS. Para simplificar el problema vamos a intentar que funcione al menos para los dos siguientes ejemplos:

- http://barrapunto.com/index.rss

```
<item rdf:about="http://softlibre.barrapunto.com/article.pl?sid=
                                  10/08/13/0728242&amp;from=rss">
<title>Oracle demanda a Google por Android</title>
<link>http://softlibre.barrapunto.com/article.pl?sid=
                             10/08/13/0728242&amp;from=rss</link>
<description>Oracle ha presentado una demanda contra Google
                                asegurando que ..."</description>
<dc:creator>mig21</dc:creator>
<dc:date>2010-08-13T07:30:00+00:00</dc:date>
<dc:subject>Oracle</dc:subject>
<slash:department>patent-IP-trolling</slash:department>
<slash:section>softlibre</slash:section>
<slash:comments>85</slash:comments>
<slash:hit_parade>85,83,41,29,11,7,3</slash:hit_parade>
</item>
```

- http://www.meneame.net/rss2.php

```
<item>
<meneame:link_id>1025318</meneame:link_id>
<meneame:user>Whitefox</meneame:user>
<meneame:votes>147</meneame:votes>
<meneame:negatives>0</meneame:negatives>
<meneame:karma>616</meneame:karma>
<meneame:comments>25</meneame:comments>
<meneame:url>http://techcrunch.com/2010/08/13/android-oracle-java-lawsuit/
                                                          </meneame:url>
<title>Google responde a la denuncia sobre Android presentada por Oracle
                                                           [ENG]</title>
<link>http://m.menea.me/lz52</link>
<comments>http://m.menea.me/lz52</comments>
<pubDate>Sat, 14 Aug 2010 09:40:02 +0000</pubDate>
<dc:creator>Whitefox</dc:creator>
<guid>http://m.menea.me/lz52</guid>
<description>&lt;![CDATA[ ... ]&gt;</description>
<media:thumbnail
  url="http://aws.mnmstatic.net/cache/a5/26/thumb-1025318.jpg"
  width='75' height='49' />
<wfw:commentRss>http://www.meneame.net/comments_rss2.php?id=1025318
</wfw:commentRss>
</item>
```

Recuerda que cada vez que se cierre un `item` en el XML, habrá que añadir el nuevo objeto `noticia` al ArrayList `noticias`.

- Asignar un `OnItemClickListener` al `ListView` para que al hacer click sobre una noticia, se muestre un `Toast` con la información de la noticia. Opcionalmente se puede lanzar un navegador del sistema que nos lleve a la URL de la noticia.

Copyright © 2014 Depto. CCIA All rights reserved.

15. Servicios Avanzados

15.1. Servicios en segundo plano

Los servicios en segundo plano, `Services` son similares a los demonios de los sistemas GNU/Linux. No necesitan una aplicación abierta para seguir ejecutándose. Sin embargo para el control de un servicio (iniciarlo, detenerlo, configurarlo) sí que es necesario programar aplicaciones con sus actividades con interfaz gráfica. En esta sección vamos a ver cómo crear nuestros propios servicios.

Los servicios heredan de la clase `Service` e implementan obligatoriamente el método `onBind(Intent)`. Este método sirve para comunicación entre servicios y necesita que se defina una interfaz AIDL (Android Interface Definition Language). Devolviendo `null` estamos indicando que no implementamos tal comunicación.

```
public class MiServicio extends Service {

        @Override
        public void onCreate() {
                super.onCreate();
                //Inicalizaciones necesarias
        }

        @Override
        public int onStartCommand(Intent intent, int flags, int startId) {
                //Comenzar la tarea de segundo plano
                return Service.START_STICKY;
        }


        @Override
        public void onDestroy() {
                super.onDestroy();
                //Terminar la tarea y liberar recursos
        }

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

}
```

Es muy importante declarar el servicio en `AndroidManifest.xml`:

```
        ...
        <service android:name=".MiServicio" />
    </application>
```

Si el servicio se encontrara declarado dentro de otra clase, el `android:name` contendría: `.MiOtraClase$MiServicio`.

Copyright © 2014 Depto. CCIA All rights reserved.

El ciclo de vida del servicio empieza con la ejecución del método `onCreate()`, después se invoca al método `onStartCommand(...)` y finalmente al detener el servicio se invoca al método `onDestroy()`.

> **Nota:**
> En versiones anteriores a Android 2.0 los servicios se arrancaban con el método `onStart()`. Utilizar el método `onStartCommand` y devolver la constante `Service.START_STICKY` es similar a usar el meodo `onStart()`. Esta constante se utiliza para servicios que se arrancan y detienen explícitamente cuando se necesitan.

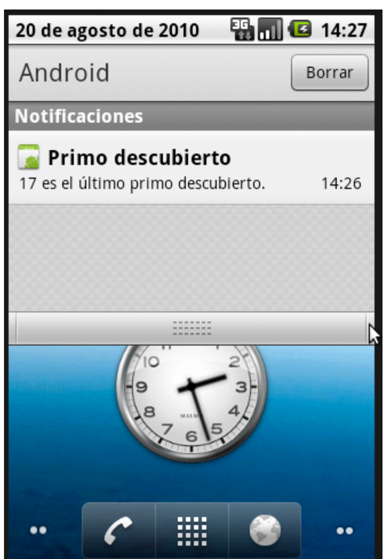
Para arrancar y detener el servicio desde una aplicación se utilizan los métodos

```
startService(new Intent(main, MiServicio.class));
stopService( new Intent(main, MiServicio.class));
```

El servicio también puede detenerse a sí mismo con el método `selfStop()`.

15.2. Notificaciones

El típico mecanismo de comunicación con el usuario que los Servicios utilizan son las `Notification`. Se trata de un mecanismo mínimamente intrusivo que no roba el foco a la aplicación actual y que permanece en una lista de notificaciones en la parte superior de la pantalla, que el usuario puede desplegar cuando le convenga.



Desplegando la barra de notificaciones

Para trabajar mostrar y ocultar notificaciones hay que obtener de los servicios del sistema el `NotificationManager`. Su método `notify(int, Notification)` muestra la

Copyright © 2014 Depto. CCIA All rights reserved.

notificación asociada a determinado identificador.

```
                Notification notification;
                NotificationManager notificationManager;
                notificationManager =
(NotificationManager)getSystemService(
                        Context.NOTIFICATION_SERVICE);
                notification = new Notification(R.drawable.icon,
                        "Mensaje evento", System.currentTimeMillis());
                notificationManager.notify(1, notification);
```

El identificador sirve para actualizar la notificación en un futuro (con un nuevo aviso de notificación al usuario). Si se necesita añadir una notificación más, manteniendo la anterior, hay que indicar un nuevo ID.

Para actualizar la información de un objeto `Notification` ya creado, se utiliza el método

```
                notification.setLatestEventInfo(getApplicationContext(),
                                "Texto", contentIntent);
```

donde `contentIntent` es un `Intent` para abrir la actividad a la cuál se desea acceder al pulsar la notificación. Es típico usar las notificaciones para abrir la actividad que nos permita reconfigurar o parar el servicio. También es típico que al pulsar sobre la notificación y abrirse una actividad, la notificación desaparezca. Este cierre de la notificación lo podemos implementar en el método `onResume()` de la actividad:

```
        @Override
        protected void onResume() {
                super.onResume();
                notificationManager.cancel(MiTarea.NOTIF_ID);
        }
```

A continuación se muestra un ejemplo completo de notificaciones usadas por una tarea `AsyncTask`, que sería fácilmente integrable con un `Service`. (Sólo haría falta crear una nueva `MiTarea` en `Service.onCreate()`, arrancarla con `miTarea.execute()` desde `Service.onStartCommand(...)` y detenerla con `miTarea.cancel()` desde `Service.onDestroy()`).

```
        private class MiTarea extends AsyncTask<String, String, String>{
                public static final int NOTIF1_ID = 1;
                Notification notification;
                NotificationManager notificationManager;

                @Override
                protected void onPreExecute() {
                        super.onPreExecute();
                        notificationManager =
(NotificationManager)getSystemService(
                                Context.NOTIFICATION_SERVICE);
                        notification = new Notification(R.drawable.icon,
                                "Mensaje evento",
```

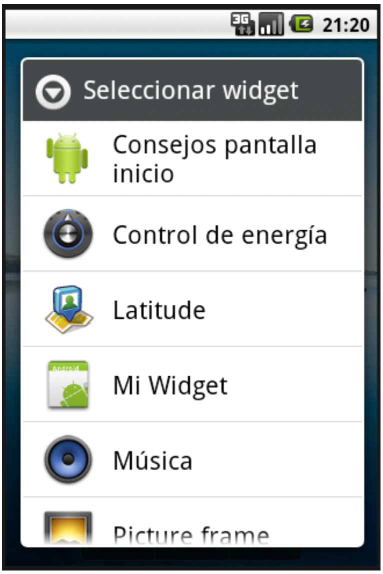

Copyright © 2014 Depto. CCIA All rights reserved.

```
System.currentTimeMillis());
                }

                @Override
                protected String doInBackground(String... params) {
                        while(condicionSeguirEjecutando){
                                if(condicionEvento){
                                        publishProgress("Información del
evento");
                                }
                        }
                        return null;
                }

                @Override
                protected void onProgressUpdate(String... values) {
                        Intent notificationIntent = new Intent(
                                getApplicationContext(),
MiActividadPrincipal.class);
                        PendingIntent contentIntent =
PendingIntent.getActivity(
                                getApplicationContext(), 0,
notificationIntent, 0);
notification.setLatestEventInfo(getApplicationContext(),
                                values[0], contentIntent);
                        notificationManager.notify(NOTIF_ID,
notification);
                }
        }
```

15.3. AppWidgets

Los widgets, que desde el punto de vista del programador son `AppWidgets`, son pequeños interfaces de programas Android que permanecen en el escritorio del dispositivo móvil. Para añadir alguno sobra con hacer una pulsación larga sobre un área vacía del escritorio y seleccionar la opción "widget", para que aparezca la lista de todos los que hay instalados y listos para añadir.

Copyright © 2014 Depto. CCIA All rights reserved.



Seleccionar un (App) Widget

Este es un ejemplo de widget, el del reloj, que viene con Android:

AppWidget reloj de Android

Los AppWidgets ocupan determinado tamaño y se refrescan con determinada frecuencia, datos que hay que declarar en el XML que define el widget. Se puede añadir como nuevo recurso XML de Android, y seleccionar el tipo Widget. Lo coloca en la carpeta `res/xml/`. Por ejemplo, este es el `res/xml/miwidget.xml`:

```
<?xml version="1.0" encoding="utf-8"? >
<appwidget-provider
xmlns:android="http://schemas.android.com/apk/res/android"
    android:minWidth="146dip"
    android:minHeight="72dip"
    android:updatePeriodMillis="600000"
    android:initialLayout="@layout/miwidget_layout"
    />
```

Este XML declara que el Layout del widget se encuentra en `res/layout/miwidget_layout.xml`.

Los AppWidgets no necesitan ninguna actividad, sino una clase que herede de `AppWidgetProvider`. Por tanto en el `AndroidManifest.xml` ya no necesitamos declarar una actividad principal. Lo que tenemos que declarar es el widget:

Copyright © 2014 Depto. CCIA All rights reserved.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="es.ua.jtech.ajdm.appwidget"
      android:versionCode="1"
      android:versionName="1.0">
    <application android:icon="@drawable/icon"
android:label="@string/app_name">

        <receiver android:name=".MiWidget" android:label="Mi Widget">
            <intent-filter>
                <action
android:name="android.appwidget.action.APPWIDGET_UPDATE" />
            </intent-filter>
            <meta-data android:name="android.appwidget.provider"
                    android:resource="@xml/miwidget" />
        </receiver>
        <service android:name=".MiWidget$UpdateService" />

    </application>
    <uses-sdk android:minSdkVersion="8" />
</manifest>
```

También puede ser necesario declarar, además de los permisos que se requieran, el uso del servicio que actualice el widget o que realice alguna tarea en background. En el anterior manifest se declara el servicio `UpdateService` dentro de la clase `MiWidget`.

A continuación se muestra un ejemplo de clase `MiWidget`que implementa un widget:

```
public class MiWidget extends AppWidgetProvider {

        @Override
        public void onUpdate(Context context, AppWidgetManager
appWidgetManager,
                        int[] appWidgetIds) {
                // Inicio de nuestro servicio de actualización:
                context.startService(new Intent(context,
UpdateService.class));
        }

        public static class UpdateService extends Service {
                @Override
                public int onStartCommand(Intent intent, int flags, int
startId) {
                        RemoteViews updateViews = new RemoteViews(
                                getPackageName(),
R.layout.miwidget_layout);

                        //Aqui se actualizarían todos los tipos de Views
que hubiera:
                        updateViews.setTextViewText(R.id.TextView01,
                                        "Valor con el que refrescamos");
                        // ...

                        //Además, ¿qué hacer si lo pulsamos? Lanzar alguna
actividad:
                        Intent defineIntent = new Intent(....);
                        PendingIntent pendingIntent =
PendingIntent.getActivity(
                                getApplicationContext(), 0, defineIntent,
0);
                        updateViews.setOnClickPendingIntent(R.id.miwidget,pendingIntent);
```


Copyright © 2014 Depto. CCIA All rights reserved.

```
                        //Y la actualización del widget con el updateViews
creado:
                        ComponentName thisWidget = new ComponentName(
                                        this, MiWidget.class);
AppWidgetManager.getInstance(this).updateAppWidget(
                                        thisWidget, updateViews);

                        return Service.START_STICKY;
                }

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

La actualización se realiza por medio de la clase `RemoteViews`, con métodos como `.setTextViewText(String)`, `.setImageViewBitmap(Bitmap)`, etc, con el propósito de actualizar los valores del layout indicado en la creación del `RemoteViews`, `R.layout.miwidget_layout` en este caso.

15.4. Publicación de software

Para publicar nuestras aplicaciones primero tenemos que empaquetarlas. Antes de empaquetar debemos preparar el código y comprobar que todo esté correcto:

- Nombre de la aplicación, icono y versión.
- Deshabilitar debugging en el AndroidManifest.xml (atributo `android:debuggable="false"` del tag de application).
- Eliminar cualquier mensaje de Log.
- Pedir sólo los permisos que de verdad la aplicación use, y no más de los necesarios.
- Por supuesto, haber probado la aplicación en terminales reales, a ser posible en más de uno.

Los paquetes de aplicaciones Android se pueden generar fácilmente con el plugin de Eclipse, pulsando sobre el proyecto con el botón derecho y seleccionando la opción Android Tools / Export Signed Application Package. Esto nos generaría el paquete con extensión apk (android package).

Para distribuir nuestra aplicación el paquete debe ir firmado (la firma nos la pide el asistente de Eclipse al empaquetar). Una firma digital puede ir certificada por una entidad certificadora conocida, o bien sin autoridad certificadora, "self-signed". Se puede usar la herramienta `keytool` para generar un certificado así.

```
keytool -genkey -v -keystore myandroidapplications.keystore
        -alias myandroidapplications -keyalg RSA -validity 10000
```

Para firmar un .apk ya generado se puede utilizar la herramienta `jarsigned`. Ambas herramientas vienen con el Android SDK.

Copyright © 2014 Depto. CCIA All rights reserved.

```
jarsigned -verbose -keystore myandroidapplications.keystore
          miaplicacion.apk myandroidapplications
jarsigned -verbose -certs -verify miaplicacion.apk
```

Una vez firmado el paquete, ya está listo para ser publicado.

> **Nota:**
> Si cambiamos de certificado en las versiones siguientes del programa, podemos tener el problema de que el terminal compruebe el certificado y no le coincida. En este caso hay que desinstalar por completo la aplicación y entonces instalar la nueva versión. Los updates de la aplicación asumen que el certificado no va a cambiar.

Para publicar en el Android Market, http://market.android.com/ debemos darnos de alta como desarrollador, pagando 25$ a través de Google Checkout, facilitando la información que nos pida el Google Checkout Merchant. Al subir la aplicación nos preguntará información sobre ésta en varios idiomas, para qué paises funciona, su tipo y categoría, el precio, la información de copyright y un contacto de soporte.

Otra opción es colgar la aplicación empaquetada en nuestro propio servidor. En este caso quien desee instalársela necesitará tener habilitada la instalación desde fuentes desconocidas en la configuración de su terminal Android.

Copyright © 2014 Depto. CCIA All rights reserved.

16. Servicios avanzados - Ejercicios

16.1. Servicio reproductor de música

Vamos a crear un servicio que inicie la reproducción de un recurso audio al arrancarse, y que detenga la reproducción al pararse.

- Descargad las plantillas de la sesión. En el proyecto `ServicioMusica` tenemos una actividad principal que muestra un botón Start y un botón Stop. En sus respectivos `OnClickListener`'s tendremos que iniciar y parar el servicio con los métodos `startService(...)` y `stopService(...)`, pasándoles en ambos casos un `new Intent(main, MiAudioServicio.class)` como parámetro. Pero para ello tendremos que crear antes la clase que define el servicio:
- Creamos una nueva clase Java que se llame `MiAudioServicio` y sobrecargamos los métodos `onStartCommand`, `onCreate`, `onDestroy` y `onBind`, ayudándonos de las herramientas que proporciona Eclipse.
- Declaramos un campo `private MediaPlayer mediaPlayer;` en la clase del servicio.
- Cuando iniciemos el servicio desde la actividad, primero se creará y se invocará al método `onCreate(...)`. En él crearemos el reproductor:

```
Toast.makeText(this,"Servicio creado ...", Toast.LENGTH_LONG).show();
mediaPlayer = MediaPlayer.create(getApplicationContext(), R.raw.ubuntu);
mediaPlayer.setLooping(true);
```

  mostrando un `Toast` para quedarnos tranquilos de que el servicio se ha iniciado. El recurso R.raw.ubuntu es un archivo .ogg que se incluye en la carpeta `res/raw` de las plantillas del proyecto. También podía haber sido un mp3.
- Una vez creado, se ejecutará el método `onStartCommand(...)`. En él iniciaremos la reproducción y devolveremos el valor `Service.START_STICKY`.

```
mediaPlayer.start();
return Service.START_STICKY;
```

- Finalmente, al destruir el servicio, detendremos la reproducción y mostraremos un `Toast`:

```
Toast.makeText(this,"onDestroy: Servicio destruido.",
Toast.LENGTH_LONG).show();
mediaPlayer.stop();
```

- En cuanto al método `onBind`, devolveremos `null`, que indica que el servicio no tiene definido un interfaz AIDL para comunicarse con otros.
- Para que el servicio funcione en la aplicación, habrá que declararlo en el `AndroidManifest.xml`, dentro de `application`:

Copyright © 2014 Depto. CCIA All rights reserved.

```
                ...
                <service android:enabled="true"
android:name=".MiAudioServicio"/>
    </application>
```

Si todo ha ido bien, y si hemos implementado los listeners de los botones que inician y detienen el servicio, debería funcionar. Probad iniciar el servicio y salir de la aplicación, entrar en otras, etc. El sonido seguirá reproduciéndose. Para detenerlo, volvemos a abrir la aplicación y lo detenemos.

16.2. Servicio con proceso en background. Contador

Los servicios se utilizan para ejecutar algún tipo de procesamiento en background. En el anterior ejercicio utilizamos el reproductor del sistema y simplemente le indicamos cuándo iniciarse y cuándo detenerse. En este ejercicio vamos a crear nuestro propio proceso que ejecute determinada tarea, en este caso, que vaya contando desde 1 hasta 100, deteniéndose 5 segundos antes de cada incremento. En cada incremento mostraremos un `Toast` que nos informe de la cuenta.

En las plantillas tenemos el proyecto `ServicioContador` que ya incluye la declaración del servicio en el manifest, la actividad que inicia y detiene el servicio, y el esqueleto del servicio `MiCuentaServicio`.

- En el esqueleto que se proporciona, viene definida una extensión de `AsyncTask` llamada `MiTarea`. Los métodos `onPreExecute`, `doInBackground`, `onProgressUpdate` y `onCancelled` están sobrecargados pero están vacíos. Se pide implementarlos, el primero de ellos inicializando el campo `i` que se utiliza para la cuenta, el segundo ejecutando un bucle desde 1 hasta 100, y en cada iteración pidiendo mostrar el progreso y durmiento después 5 segundos con `Thread.sleep(5000)`. El tercer método, `onProgressUpdate` mostrará el `Toast` con el progreso, y por último el método de cancelación pondrá el valor máximo de la cuenta para que se salga del bucle.
- En los métodos del servicio, `onCreate`, `onStartCommand` y `onDestroy`, introduciremos la creacion de la nueva `MiTarea`, su ejecución (método `execute()` de la tarea) y la cancelación de su ejecución (método `cancel()` de la tarea).

Una vez más, el servicio deberá seguir funcionando aunque se salga de la aplicación y podrá ser parado entrando de nuevo en la aplicación y pulsando Stop.

16.3. Servicio con notificaciones. Números primos

Este ejercicio es una extensión del anterior, pero vamos a utilizar un nuevo proyecto plantilla, el `ServicioNotificaciones`. En lugar de mostrar cualquier número de la cuenta, vamos a mostrarlos sólo si son primos. Además, en lugar de mostrar un `Toast`,

Copyright © 2014 Depto. CCIA All rights reserved.

vamos a mostrar una `Notification` que aparecerá en la barra de tareas y se actualizará con la llegada de cada nuevo número. Si salimos de la aplicación sin parar el servicio, seguirán apareciendo notificaciones, y si pulsamos sobre la notificación, volverá a lanzar la actividad, cerrándose la notificación que hemos pulsado.

- Dentro del servicio `MiNumerosPrimosServicio` se encuentra declarada la `AsyncTask` llamada `MiTarea`. En ella tenemos como campos de la clase una `Notification` y un `NotificationManager`. Hay que darles valores en el método `onPreExecute()`.
- El método `doInBackground(...)` ejecutará un bucle que irá incrementando `i` mientras su valor sea menor de `MAXCOUNT`. En cada iteración, si el número es primo (función incluida en la plantilla), pedirá que se muestre el progreso, pasándole como parámetro el nuevo primo encontrado.
- Implementar el método `onProgressUpdate(...)` para que muestre la notificación. Para ello habrá que actualizar la notificación con el método `setLatestEventInfo`, al cuál le pasaremos en un String la información del último primo descubierto y le pasaremos un `PendingIntent` para que al pulsar sobre la notificación, nos devuelva a la actividad de la aplicación, por si la hemos cerrado. Para crear el PendingIntent utilizaremos el método `PendingIntent.getActivity(...)` al cuál le tenemos que pasar un `new Intent(getApplicationContext(),Main.class)`.
- La aplicación debería funcionar en este punto, mostrando las notificaciones y relanzando la aplicación si son pulsadas, pero no cerrándolas al pulsarlas. Para ello simplemente tenemos que llamar al método `cancel(id)` del `notificationManager` y pasarle la constante NOTIF_ID para que la notificación no se muestre como una nueva, sino como actualización de la que ya habíamos puesto. Una manera de hacerlo es en un método estático del `MiNumerosPrimosServicio`, que podemos llamar `cerrarMiNotificacion(NotificationManager nm)`. Este método será invocado desde el `Main.onResume()`.

Copyright © 2014 Depto. CCIA All rights reserved.

Notificación del servicio de números primos

16.4. IP AppWidget

En programación de Android se denomina `Widget` a los componentes de alto nivel de la interfaz de usuario, y `AppWidgets` a los widgets que se pueden añadir al escritorio del sistema operativo, como el reloj, pequeños controles, etc.

Vamos crear un proyecto `AppWidget` para construir un AppWidget de Android, que nos muestre en todo momento la IP que el dispositivo está usando en este momento. No necesitaremos ninguna actividad, así que podemos desmarcar la casilla "Create activity", o bien eliminar la actividad después (no sólo la clase, sino también la declaración en el manifest).

En el proyecto pulsamos con el boton derecho y añadimos un nuevo Android XML File, de tipo AppWidget Provider, que se llame `miwidget.xml`. El editor nos permite pulsar sobre el AppWidget Provider y editar sus atributos. Ponemos los siguientes:

```
android:minWidth="146dip"
android:minHeight="72dip"
android:updatePeriodMillis="600000"
android:initialLayout="@layout/miwidget_layout"
```

El miwidget_layout lo tenemos que crear, o dará error. Así que creamos un nuevo Android XML File de tipo Layout llamado `miwidget_layout.xml` y le añadimos un campo de texto `TextView` con el texto vacío.

Copyright © 2014 Depto. CCIA All rights reserved.

Creamos una clase `MiWidget` que herede de `AppWidgetProvider`, en el paquete `es.ua.jtech.daa.appwidget`. Sobrecargamos su método `onUpdate(...)` y actualizamos en él el campo de texto, usando `RemoteViews` y pasándoselos al `AppWidgetManager`:

```
RemoteViews updateViews = new RemoteViews(context.getPackageName(),
                                          R.layout.miwidget_layout);
updateViews.setTextViewText(R.id.TextView01,"Hola");
ComponentName thisWidget = new ComponentName(context, MiWidget.class);
AppWidgetManager.getInstance(context).updateAppWidget(thisWidget,
updateViews);
```

Antes de probar el widget hay que declararlo en el `AndroidManifest.xml`, dentro de `application`:

```
        <receiver android:name=".MiWidget" android:label="Mi Widget">
            <intent-filter>
                <action
android:name="android.appwidget.action.APPWIDGET_UPDATE" />
            </intent-filter>
            <meta-data android:name="android.appwidget.provider"
                    android:resource="@xml/miwidget" />
        </receiver>
```

Ejecutamos el widget desde Eclipse, como aplicación android, y comprobamos que no ocurra ningún error en la consola de Eclipse. Ya se puede añadir el widget en el escritorio, efectuando una pulsación larga sobre una porción de área libre del escritorio, y seleccionando nuestro widget.

Copyright © 2014 Depto. CCIA All rights reserved.

Instalación del AppWidget en el emulador

Si todo funciona correctamente, vamos a implementar en el `MiWidget` un servicio `UpdateService` que realizará la actualización del widget, evitando así bloqueos debidos a la velocidad de la red. El servicio recogerá la información que le devuelve en texto plano la página http://www.whatismyip.org y la mostrará en el campo de texto del widget.

Instrucciones para programar el servicio que se pide:

- Creamos la clase `public static class UpdateService extends Service` dentro de la clase `MiWidget` y sobrecargamos los métodos `onBind` (que es obligatorio, pero devolverá `null`) y `onStartCommand` que devolverá `Service.START_STICKY`.
- Hay que declarar el servicio en el `AndroidManifest.xml`, dentro de `application`, con:

```
<service android:name=".MiWidget$UpdateService" />
```

- Del método `MiWidget.onUpdate(...)` podemos cortar todas las líneas y sustituirlas por la llamada al servicio:

```
        context.startService(new Intent(context, UpdateService.class));
```

- En el método `onStartCommand` del servicio, pegaremos las líneas que actualizan los RemoteViews, pero las modificaremos para que obtengan el contexto y el paquete del widget, quedando el método así:

```
@Override
public int onStartCommand(Intent intent, int flags, int startId) {
        RemoteViews updateViews = new RemoteViews(getPackageName(),
                        R.layout.miwidget_layout);
        updateViews.setTextViewText(R.id.TextView01,"Hola Serv");
        ComponentName thisWidget = new ComponentName(this,
MiWidget.class);
        AppWidgetManager.getInstance(this).updateAppWidget(thisWidget,
                        updateViews);
        return Service.START_STICKY;
}
```

Ahora podemos volver a probar el widget, ejecutándolo desde Eclipse. Si funciona, podemos pasar a sustituir la línea

```
                        updateViews.setTextViewText(R.id.TextView01,"Hola
Serv");
```

por el código que accede a la URL por HTTP, obteniendo un InputStream y convirtiendo los bytes a String para mostrarlo:

```
                    String ipstring = "Unknown IP";

            try {
```


Copyright © 2014 Depto. CCIA All rights reserved.

```
                            URL url = new URL("http://icanhazip.com/");
                            HttpURLConnection http =
(HttpURLConnection)url.openConnection();
                            InputStream is = http.getInputStream();
                            byte[] buffer = new byte[20];
                            is.read(buffer, 0, 20);
                            ipstring = "IP: "+new String(buffer);
            } catch (MalformedURLException e) {
                            e.printStackTrace();
                    } catch (IOException e) {
                            e.printStackTrace();
                    }

                    updateViews.setTextViewText(R.id.TextView01,ipstring);
```

Antes de probarlo hay que añadir el permiso de Internet en el `AndroidManifest.xml`, fuera de `application`:

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

Ejecutamos y observamos el resultado:

Widget que muestra la IP

Se puede añadir un comportamiento al pulsar sobre algún componente del widget. Por ejemplo, para que se abra un navegador con la web consultada, añadiríamos las siguientes líneas para actualizar el `updateViews`:

```
        Intent defineIntent = new Intent(Intent.ACTION_VIEW,
                Uri.parse("www.whatismyip.org"));
```


Copyright © 2014 Depto. CCIA All rights reserved.

```
         PendingIntent pendingIntent = PendingIntent.getActivity(
                getApplicationContext(), 0, defineIntent, 0);
        updateViews.setOnClickPendingIntent(R.id.miwidgetlayout,
pendingIntent);
```

Nota:

Para que la referencia al recurso `R.id.miwidgetlayout` funcione, se tiene que definir el atributo `android:id="@+id/miwidgetlayout"` del `LinearLayout` del widget, que se encuentra en el archivo `miwidget_layout.xml`.

Copyright © 2014 Depto. CCIA All rights reserved.

Copyright © 2014 Depto. CCIA All rights reserved.