ANDROID: NIVEL III

CONTENIDO: ANDROID III

PERMISOS EN ANDROID

El sistema operativo android tiene numerosas aplicaciones disponibles, y casi todas requieren de permisos especiales para su correcto funcionamiento. Si se quiere instalar una aplicación de estas características siempre será obligatorio aceptar los permisos impuestos, y si no los queremos aceptar no podremos instalarla. Estos permisos se especifican en el archivo AndroidManifest.xml se genera automáticamente al crear un proyecto y en él se declaran todas las especificaciones de nuestra aplicación. Cuando hablamos de especificaciones hacemos mención a las Activities utilizadas, los Intents, bibliotecas, el nombre de la aplicación, el hardware que se necesitará, los permisos de la aplicación, etcétera.

<uses-permissions>: Mediante este Tag especificamos los permisos que va a necesitar nuestra aplicación para poder ejecutarse, además son los que deberá aceptar el usuario antes de instalarla. Por ejemplo, si se desea utilizar funcionalidades con Internet o el vibrador del teléfono, hay que indicar que nuestra aplicación requiere esos permisos.

<uses-sdk>: En este tag determinamos las distintas versiones Android que va a utilizar nuestra aplicación, tanto sobre qué versiones va a correr como qué versión fue utilizada para realizar nuestras pruebas. Mediante el atributo android:minSdkVersion establecemos a partir de qué versión de Android nuestra aplicación podrá correr.

Tener la información del Manifest actualizada, correcta y ordenada ayudará mucho a la hora de mantener nuestra aplicación y su correcto funcionamiento, pero por sobre todas las cosas ayudará a que sea visible en el Android Market el cual, muestra a los teléfonos las aplicaciones sólo si estos cumplen con los requisitos de hardware y software especificados por la aplicación en su manifest. Es decir, si tenemos mal la información del manifest nuestra aplicación en el mejor de los casos podrá ser descargada por dispositivos que no la pueden ejecutar correctamente dando como resultado puntuaciones y comentarios negativos o directamente no aparecerá para ser descargada.

TRABAJANDO EN ANDROID CON:

- **Cámara**

Diseño de la interfaz

Nuestro diseño tendrá un botón para adquirir la imagen un RadioGroup que contendrá a 3 botones y un ImageView. Los 3 botones serán seleccionar de donde proviene la imagen ya sea de la cámara (como vista previa o como imagen completa) o de la galería.

Código

Definimos 3 constantes, con dos de ellas vamos a identificar la acción realizada (tomar una fotografía o bien seleccionarla de la galería) y con la otra estableceremos un nombre para el archivo donde escribiremos la fotografía de tamaño completo al tomarla.

```
private static int TAKE_PICTURE = 1;
private static int SELECT_PICTURE = 2;
private String name = “”;
```

La forma más sencilla de tomar fotografías es utilizar un intent con ACTION_IMAGE_CAPTURE, acción que pertenece al Media Store1 y luego sobrecargar el método onActivityResult para realizar algo con el archivo recibido de la cámara. Dentro del método onCreate asignaremos a la variable de instancia name y luego vamos a trabajar sobre la acción al click del botón.

Este nombre, inicializado con una llamada a getExternalStorageDirectory() guardará un archivo en la tarjeta SD del teléfono y el archivo se llamará test.jpg cada vez que grabemos una fotografía de tamaño completo se sobre escribe.

```
name = Environment.getExternalStorageDirectory() +
“/test.jpg”;
Button btnAction = (Button)findViewById(R.id.btnPic);
btnAction.setOnClickListener(new OnClickListener() {
      @Override
      public void onClick(View v) {
      ...
      }
}
```

Primero obtenemos los botones de imagen completa y de galería para revisar su estatus más adelante. Luego construimos un intent que es necesario si accesamos la cámara con la acción ACTION_IMAGE_CAPTURE, si accesamos la galería con la acción ACTION_PICK. En el caso de la vista previa (thumbnail) no se necesita más que el intent, el código e iniciar la Activity correspondiente. Por eso inicializamos las variables intent y code con los valores necesarios para el caso del thumbnail así de ser el botón seleccionado no validamos nada en un if.

```
RadioButton rbtnFull = (RadioButton)findViewById(R.id.radbtnFull);

RadioButton                    rbtnGallery                    =
(RadioButton)findViewById(R.id.radbtnGall);

     Intent intent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
```

Asignamos el código a tomar fotografía, este código junto al intent se utilizarán adelante para iniciar la Activity.

```
int code = TAKE_PICTURE;
```

Si el chequeado es el botón de vista previa no necesitamos agregar nada más. Si el chequeado es el botón de imagen completa, además del intent y código agregamos un URI para guardar allí el resultado. Si el chequeado es el de la galería necesitamos un intent y código distintos que asignamos en la consecuencia del if.

```
     if (rbtnFull.isChecked()) {
          Uri output = Uri.fromFile(new File(name));
          intent.putExtra(MediaStore.EXTRA_OUTPUT, output);
     } else if (rbtnGallery.isChecked()){
          intent = new Intent(Intent.ACTION_PICK,
          android.provider.MediaStore.Images.Media.INTERNAL_CONTEN
          T_URI);
          code = SELECT_PICTURE;
     }
```

Luego, con todo preparado iniciamos la Activity correspondiente.

```
          startActivityForResult(intent, code);
```

Además, es necesario sobrecargar la función onActivityResult para indicar que queremos hacer con la imagen recibida (ya sea de la cámara o de la galería) una vez ha sido seleccionada. Es

necesario revisar si la imagen viene de la cámara TAKE_PICTURE o de la galería SELECT_PICTURE.

```
protected void onActivityResult(int requestCode, int resultCode,
Intent data) {
      if (requestCode == TAKE_PICTURE) {
      …
      } else if (requestCode == SELECT_PICTURE){
      …
      }
}
```

Si viene de la cámara, verificamos si es una vista previa o una foto completa:

```
      if (data != null) {…}
       else {…}
```

En el caso de una vista previa, obtenemos el extra "data" del intent y lo mostramos en el ImageView:

```
      if (data != null) {…}
       else {…}
```

En el caso de una fotografía completa, a partir del nombre del archivo ya definido lo buscamos y creamos el bitmap para el ImageView:

```
      ImageView iv = (ImageView)findViewById(R.id.imgView);
      iv.setImageBitmap(BitmapFactory.decodeFile(name));
```

Si quisiéramos incluir esa imagen en nuestra galería, utilizamos un MediaScannerConnectionClient

```
new MediaScannerConnectionClient() {
      private MediaScannerConnection msc = null; {
      msc                          =                          new
      MediaScannerConnection(getApplicationContext(), this);
      msc.connect();
      }
      public void onMediaScannerConnected() {
      msc.scanFile(fileName, null);
      }
      public void onScanCompleted(String path, Uri uri) {
      msc.disconnect();
      }
};
```

Si viene de la galería recibimos el URI de la imagen y construimos un Bitmap a partir de un stream de bytes:

```
Uri selectedImage = data.getData();
InputStream is;
      try {
      is                                                    =
      getContentResolver().openInputStream(selectedImage);
      BufferedInputStream bis = new BufferedInputStream(is);
      Bitmap bitmap = BitmapFactory.decodeStream(bis);
      ImageView iv = (ImageView)findViewById(R.id.imgView);
      iv.setImageBitmap(bitmap);
} catch (FileNotFoundException e) {}
```

Utilización de intents para utilizar la cámara y accesar la galería: se utilizaron intents para realizar acciones específicas, la existencia de intents para acciones comunes predefinidas nos facilitan muchas tareas, en este caso no fue necesario accesar directamente el hardware de la cámara si no pudimos tomar una fotografía de una manera más sencilla.

Algo similar sucede con el acceso a la galería de fotos.

- Video

Para ello realizaremos lo siguiente en el proyecto sobre el que vayamos a trabajar:

Configuración de la rotación de pantalla:

Para evitarnos problemas porque cada vez que se mueve el teléfono y rota la pantalla se destruye y re-crea la activity, en el manifiesto le hemos configurado que no pueda rotarse con android:screenOrientation=”portrait” dentro de la etiqueta de la activity principal.

Permisos en el manifiesto

Por la utilización de la cámara para grabar audio y vídeo al SD requerimos de estos cuatro permisos:

En la clase de la activity principal, implementaremos la interfaz SurfaceHolder.Callback para el manejo del SurfaceView que se agregará más adelante.

Esta interfaz requiere la implementación de los siguientes métodos:

```
	@Override
	public void surfaceChanged(SurfaceHolder arg0, int arg1,
	int arg2, int
	arg3) {
	}
	@Override
	public void surfaceCreated(SurfaceHolder arg0) {
	}
	@Override
	public void surfaceDestroyed(SurfaceHolder arg0) {
}
```

SurfaceView

Nos permite tener un espacio dedicado a dibujar los frames del vídeo para la vista y la reproducción.

Nuestro diseño tendrá tres botones: "Grabar", "Detener" y "Reproducir" además del ya mencionado SurfaceView El código completo del layout es:

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

Código

Vamos a definir 4 variables globales, las primeras dos son para gestionar la grabación y reproducción:

```
	private MediaRecorder mediaRecorder = null;
	private MediaPlayer mediaPlayer = null;
```

También tendremos una tercera variable de instancia para el nombre del archivo en el que se va a escribir:

```
	private String fileName = null;
```

La cuarta variable de instancia para controlar cuando se está grabando:

```
private boolean recording = false;
```

De los métodos heredados por la interface nos interesan 2: surfaceDestroyed donde vamos a liberar los recursos y surfaceCreated donde vamos a inicializar.

```
public void surfaceDestroyed(SurfaceHolder holder) {
mediaRecorder.release();
mediaPlayer.release();
}
```

Verificamos si las variables son nulas (para ejecutar este código sólo una vez) y luego de inicializarlas se coloca el SurfaceHolder como display para la vista previa de la grabación y para la vista de la reproducción:

```
public void surfaceCreated(SurfaceHolder holder) {
if (mediaRecorder == null) {
mediaRecorder = new MediaRecorder();
mediaRecorder.setPreviewDisplay(holder.getSurface());
}
if (mediaPlayer == null) {
mediaPlayer = new MediaPlayer();
mediaPlayer.setDisplay(holder);
}}
```

Adicionalmente, se agrega un método para preparar la grabación configurando los atributos de la fuente para audio y vídeo, el formado y el codificador.

```
public void prepareRecorder(){
mediaRecorder.setAudioSource(MediaRecorder.AudioSource.M
IC);
mediaRecorder.setVideoSource(MediaRecorder.VideoSource.C
AMERA);
mediaRecorder.setOutputFormat(MediaRecorder.OutputFormat
.MPEG_4);
mediaRecorder.setAudioEncoder(MediaRecorder.AudioEncoder
.DEFAULT);
mediaRecorder.setVideoEncoder(MediaRecorder.VideoEncoder
.MPEG_4_SP);
}
```

Dentro del método onCreate realizamos algunas inicializaciones, primero para la variable del nombre del archivo:

```
fileName = Environment.getExternalStorageDirectory() + "/test.mp4";
```

También para el SurfaceView donde se reproducirá el vídeo:

```
SurfaceView surface = (SurfaceView)findViewById(R.id.surface);
SurfaceHolder holder = surface.getHolder();
holder.addCallback(this);
holder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
```

Definimos los botones sobre los que vamos a trabajar en su evento de click:

```
final Button btnRec = (Button)findViewById(R.id.btnRec);
final Button btnStop = (Button)findViewById(R.id.btnStop);
final Button btnPlay = (Button)findViewById(R.id.btnPlay);
```

Botón de grabación

Al iniciar deshabilitamos los botones de grabar y reproducir luego habilitamos el de detener. Llamamos el método que configura el MediaRecorder y le decimos el archivo de salida. Una vez configurado todo llamamos al método prepare que deja todo listo para iniciar la grabación e iniciamos la grabación y actualizamos el estatus de la variable recording.

```
btnRec.setOnClickListener(new OnClickListener() {
public void onClick(View v) {
btnRec.setEnabled(false);
btnStop.setEnabled(true);
btnPlay.setEnabled(false);
prepareRecorder();
mediaRecorder.setOutputFile(fileName);
try {
mediaRecorder.prepare();
} catch (IllegalStateException e) {
} catch (IOException e) {
} mediaRecorder.start(); recording = true;
}
```

Botón de reproducir

Deshabilitamos los botones de grabar, reproducir y habilitamos el de detener. Si concluye la reproducción (porque el video se acabó no porque el usuario haya presionado detener) habilitamos los botones de grabar, reproducir y deshabilitamos el de detener).

Luego configuramos el archivo a partir del cual se reproducirá, preparamos el Media Player e iniciamos la reproducción.

```
      btnPlay.setOnClickListener(new OnClickListener() {
      @Override
      public void onClick(View v) {
      btnRec.setEnabled(false);
      btnStop.setEnabled(true);
      btnPlay.setEnabled(false);
      mediaPlayer.setOnCompletionListener(new
      OnCompletionListener() {
      @Override
      public void onCompletion(MediaPlayer mp) {
      btnRec.setEnabled(true);
      btnStop.setEnabled(false);
      btnPlay.setEnabled(true);
      }
      });
      try {
      mediaPlayer.setDataSource(fileName);
      mediaPlayer.prepare();
      } catch (IllegalStateException e) {
      } catch (IOException e) {
      }
      mediaPlayer.start();
      }
});
```

- **SurfaceView**

Utilizamos este tipo de view que permite el manejo de gráficas y provee una forma de dibujar, en nuestro caso las imágenes necesarias para la vista previa del video y luego para su reproducción.

- **Manejo y configuración de Media Recorder**

Para la captura de video, aprendimos a inicializar, configurar y utilizar el Media Recorder.

- **Manejo y configuración de Media Player**

Una vez grabado un video, el Media Player nos sirvió para reproducirlo.

- **Utilización de la cámara de vídeo**

Hicimos uso de la cámara de nuevo pero ahora para la grabación de video, de una manera similar que en el caso de las fotos (guía anterior) no manejamos directamente el hardware si no que a través de las herramientas que nos provee Android utilizamos el componente sin complicaciones.

- **Audio**

Reproducción de audio (archivo contenido en la aplicación)

Para ello nuestra aplicación deberá contener una carpeta RAW con los audios a reproducir.

Formatos soportados

Los formatos soportados por Android son: Ogg, Wav, mp3

Clase MediaPlayer

Es la que nos permitirá rescatar el audio almacenado en nuestra aplicación.

```
MediaPlayer mp=MediaPlayer.create(this,R.raw.gato);
```

Seguidamente llamamos al método start() que ejecutará la reproducción del audio:

```
mp.start();
```

Reproducción, pausa, continuación y detención de un archivo de audio

En ocasiones será necesario iniciar un archivo mp3, detener, continuar, detener en forma definitiva y activación o no de la reproducción en forma circular.

Para ello definimos tres atributos uno de la clase MediaPlayer para administrar el archivo mp3, un entero donde se almacena la posición actual de reproducción en milisegundos (para poder continuarla en el futuro) y la referencia de un objeto de la clase Button:

```
MediaPlayer mp;

Button b1;

int  posicion=0;
```

Destruir

El método destruir verifica con un if si el objeto de la clase MediaPlayer está creado procede a liberar recursos del mismo llamando al método release:

```
public void destruir() {

    if(mp!=null)

        mp.release();

}
```

Iniciar y reproducción circular

El método iniciar que se ejecuta al presionar el botón "Iniciar" primero llama al método destruir (para el caso que el mp3 este en ejecución actualmente) seguidamente creamos un objeto de la clase MediaPlayer llamando al método create (en este hacemos referencia al archivo que copiamos a la carpeta raw) Llamamos al método start.

Por último verificamos si la reproducción debe ejecutarse en forma circular (en forma indefinida una y otra vez):

```
public void iniciar(View v) {

    destruir();

    mp = MediaPlayer.create(this,R.raw.numeros);

    mp.start();

    String op=b1.getText().toString();

    if (op.equals("No reproducir en forma circular"))

        mp.setLooping(false);

    else

        mp.setLooping(true);

}
```

Pausar

El método pausar verifica que el objeto de la clase MediaPlayer este creado y en ejecución, en caso afirmativo recuperamos la posición actual de reproducción y llamamos seguidamente al método pause:

```
public void pausar(View v) {

    if(mp != null && mp.isPlaying()) {

        posicion = mp.getCurrentPosition();

        mp.pause();

    }

}
```

Continuar

El método continuar verifica que el objeto de la clase MediaPlayer este creado y la propiedad isPlaying retorne false para proceder a posicionar en que milisegundo continuar la reproducción:

```
    public void continuar(View v) {
        if(mp != null && mp.isPlaying()==false) {
            mp.seekTo(posicion);
            mp.start();
        }
    }
```

Detener

El método detener interrumpe la ejecución del mp3 e inicializa el atributo posición con cero:

```
    public void detener(View v) {
        if(mp != null) {
            mp.stop();
            posicion = 0;
        }
    }
```

Cuando se presiona el botón que cambia si la reproducción se efectúa en forma circular o no procedemos a extraer su texto y según dicho valor almacenamos el valor opuesto:

```
    public void circular(View v) {
        detener(null);
        String op=b1.getText().toString();
        if (op.equals("No reproducir en forma circular"))
            b1.setText("reproducir en forma circular");
        else
            b1.setText("No reproducir en forma circular");
    }
```

Reproducción de audio contenido en una tarjeta SD

Para ello creamos un objeto de la clase Uri llamando al método parse donde indicamos el path y nombre del archivo a recuperar:

```
Uri datos = Uri.parse(Environment.getExternalStorageDirectory().getPath() +"/gato.mp3");
```

Creamos el objeto de la clase MediaPlayer pasando ahora la referencia del objeto de la clase Uri:

```
MediaPlayer mp=MediaPlayer.create(this, datos);
```

Iniciamos la reproducción del mp3:

```
mp.start();
```

Reproducción de audio localizado en internet

El primer paso es modificar el archivo AndroidManifest.xml donde autorizamos a la aplicación a acceder a recursos localizados en internet:

```
android.permission.INTERNET
```

Para recuperar un archivo mp3 de internet procedemos de la siguiente manera, primero creamos un objeto de la clase MediaPlayer:

```
MediaPlayer mp=new MediaPlayer();
```

Luego llamamos al método setDataSource indicando la dirección de internet donde se almacena el archivo mp3:

```
mp.setDataSource("http://wwwX.XXXXX /recursos/gato.mp3");
```

Llamamos al método prepare y seguidamente llamamos a start:

```
mp.prepare();

mp.start();
```

Todo esto lo hacemos en un bloque try/catch para capturar excepciones de tipo IOException.

Esta primera aproximación para ejecutar un mp3 localizado en internet bloquea la aplicación hasta que se carga por completo el archivo, es decir queda ejecutándose el método mp.prepare() hasta que finaliza la recuperación en forma completa.

Para poder capturar el evento que el archivo se terminó de recuperar debemos implementar la interface OnPreparedListener:

```
public class AudioActivity extends Activity implements OnPreparedListener {
```

Con esto decimos que nuestra clase implementará el método onPrepared donde iniciamos la ejecución del mp3:

```
    public void onPrepared(MediaPlayer mp) {mp.start();}
```

En el evento click del botón creamos el objeto de la clase MediaPlayer, le pasamos al método setOnPreparedListener la dirección del objeto que capturará el evento de que el recurso está completo. Luego llamamos a los métodos setDataSource y prepareAsync para inicializar la carga del mp3. Finalmente mostramos un mensaje para informar al usuario que el archivo se está descargando:

```
    public void ejecutar(View v) {
        mp=new MediaPlayer();
        mp.setOnPreparedListener(this);
         try {
mp.setDataSource("http://www.XXX.com/recursos/gato.mp3");
             mp.prepareAsync();
         }catch(IOException e) {
         }
         Toast t=Toast.makeText(this,"Espere un momento mientras se carga el mp3", Toast.LENGTH_SHORT);
         t.show();
    }
```

MAPAS EN ANDROID (GOOGLE MAPS ANDROID API V2)

La segunda versión de su API de Google Maps para Android. Esta nueva versión presenta muchas novedades interesantes, de las que cabe destacar las siguientes:

- Integración con los Servicios de Google Play (*Google Play Services*) y la Consola de APIs.
- Utilización a través de un nuevo tipo específico de fragment (MapFragment), una mejora muy esperada por muchos.
- Utilización de *mapas vectoriales*, lo que repercute en una mayor velocidad de carga y una mayor eficiencia en cuanto a uso de ancho de banda.
- Mejoras en el sistema de caché, lo que reducirá en gran medida las famosas áreas en blanco que tardan en cargar.
- Los mapas son ahora 3D, es decir, podremos mover nuestro punto de vista de forma que lo veamos en perspectiva.

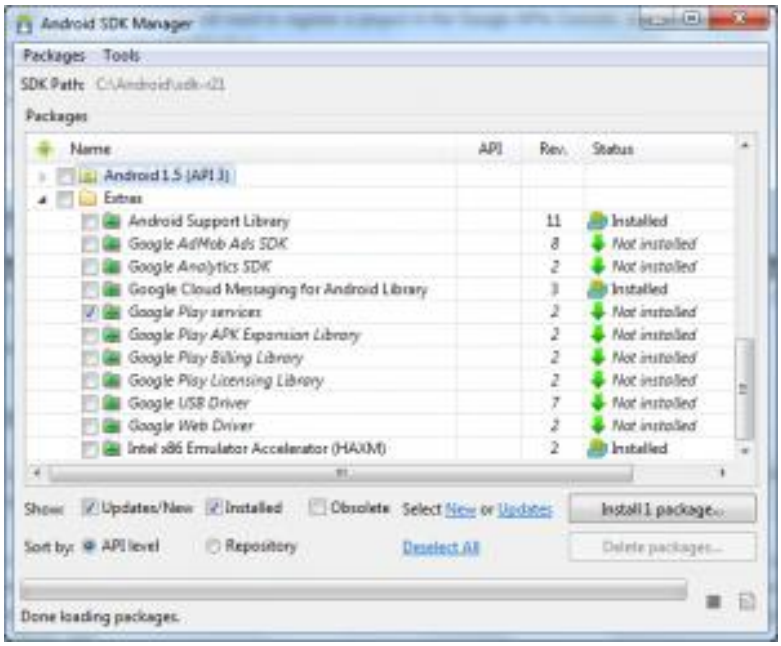
La principal diferencia con respecto a la version de Google maps anterior es el componente que utilizaremos para la inclusión de mapas en nuestra aplicación. En la anterior versión de la API, para incluir un mapa en la aplicación debíamos utilizar un control de tipo MapView, que además requería que su actividad contenedora fuera del tipo MapActivity. Con la nueva API nos olvidaremos de estos dos componentes y pasaremos a tener sólo uno, un nuevo tipo específico de *fragment* llamadoMapFragment. Esto nos permitirá entre otras cosas añadir uno [o varios, esto también es una novedad] mapas a cualquier actividad, sea del tipo que sea, y contando por supuesto con todas las ventajas del uso de fragments.

Nota importante: dado que el nuevo control de mapas se basa en fragments, si queremos mantener la compatibilidad con versiones de Android anteriores a la 3.0 tendremos que utilizar la librería de soporte *android-support*.

Además de ello la integración de la API con los *Google Play Services* y la *Consola de APIs* de Google, harán que los preparativos del entorno, las librerías utilizadas, y el proceso de obtención de la *API Key* de Google Maps sean un poco distintos a lo que eran en la version anterior.

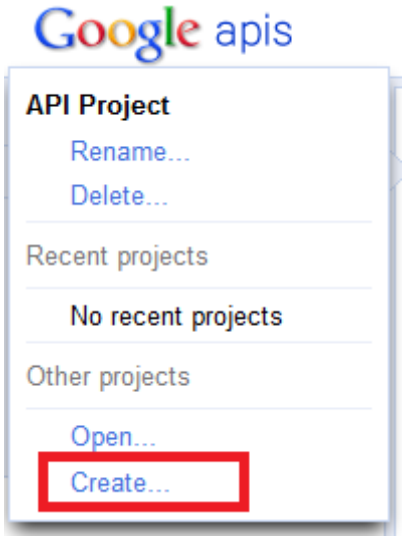
Antes de empezar a trabajar con Google Maps en nuestras aplicaciones hay que realizar unos pasos previos antes.

En primer lugar, dado que la API v2 se proporciona como parte del SDK de *Google Play Services*, será necesario incorporar previamente a nuestro entorno de desarrollo dicho paquete. Haremos esto accediendo desde Eclipse al Android SDK Manager y descargando del apartado de extras el paquete llamado "Google Play Services".

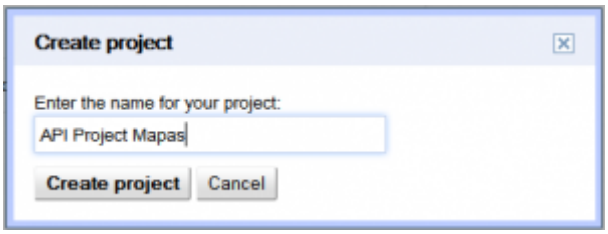


Tras pulsar el botón de *Install* y aceptar la licencia correspondiente el paquete quedará instalado en nuestro sistema, concretamente en la ruta: <carpeta-sdk-android>/extras/google/google_play_services/. Recordemos esto porque nos hará falta más adelante.

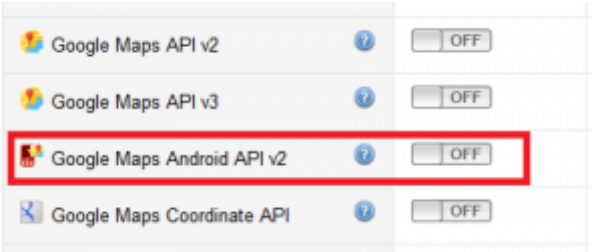
El siguiente paso será obtener una *API Key* para poder utilizar el servicio de mapas de Google en nuestra aplicación. Este paso ya lo comentamos en los artículos sobre la API v1, pero en este caso el procedimiento será algo distinto. La nueva API de mapas de Android se ha integrado por fin en la Consola de APIs de Google (una herramienta de la que ya hablamos por ejemplo en los artículos dedicados a *Google Cloud Messaging*), por lo que el primer paso será acceder a ella. Una vez hemos accedido, tendremos que crear un nuevo proyecto desplegando el menú superior izquierdo y seleccionando la opción "Create…".



Aparecerá entonces una ventana que nos solicitará el nombre del proyecto. Introducimos algún nombre descriptivo y aceptamos sin más.



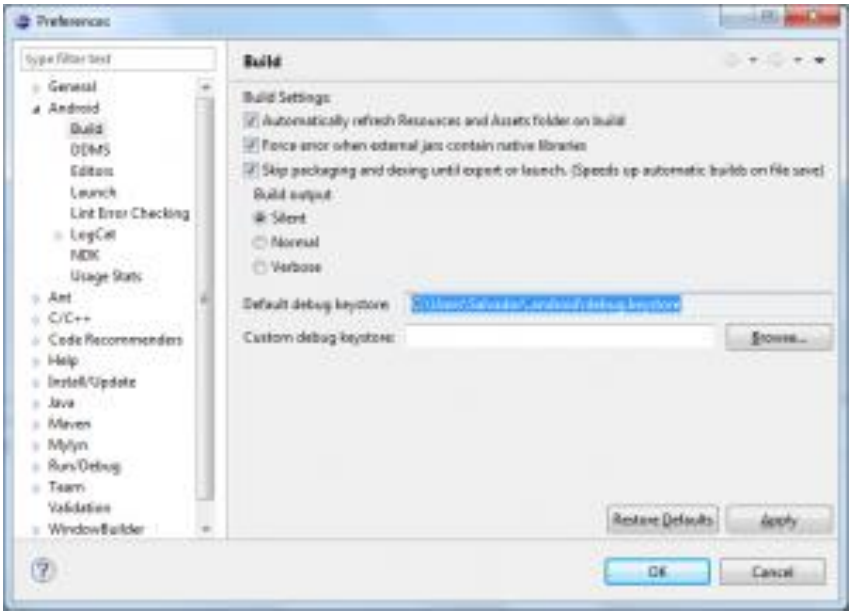
Una vez creado el proyecto, accederemos a la opción "Services" del menú izquierdo. Desde esta ventana podemos activar o desactivar cada uno de los servicios de Google que queremos utilizar. En este caso sólo activaremos el servicio llamado "Google Maps Android API v2" pulsando sobre el botón ON/OFF situado justo a su derecha.



Una vez activado aparecerá una nueva opción en el menú izquierdo llamada "API Access". Accediendo a dicha opción tendremos la posibilidad de obtener nuestra nueva API Key que nos permita utilizar el servicio de mapas desde nuestra aplicación particular.

Para ello, pulsaremos el botón "Create new Android key…". Esto nos llevará a un cuadro de diálogo donde tendremos que introducir algunos datos identificativos de nuestra aplicación. En concreto necesitaremos dos: la huella digital (SHA1) del certificado con el que firmamos la aplicación, y el paquete java utilizado. El segundo no tiene misterio, pero el primero requiere alguna explicación. Toda aplicación Android debe ir firmada para poder ejecutarse en un dispositivo, tanto físico como emulado. Este proceso de firma es uno de los pasos que tenemos que hacer siempre antes de distribuir públicamente una aplicación.

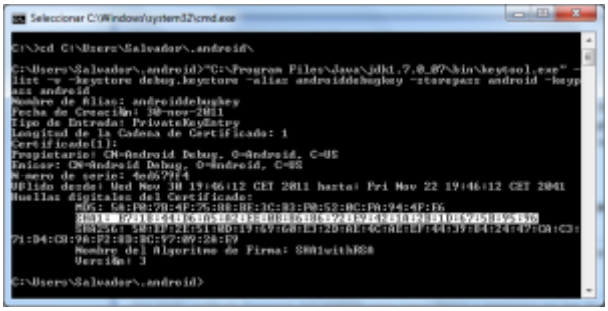
Adicionalmentes, durante el desarrollo de la misma, para realizar pruebas y la depuración del código, aunque no seamos conscientes de ello también estamos firmado la aplicación con un "certificado de pruebas". Podemos saber en qué carpeta de nuestro sistema está almacenado este certificado accediendo desde Eclipse al menú Window /Preferences y accediendo a la sección Android / Build.



Como se puede observar, en mi caso el certificado de pruebas está en la ruta "C:\Users\Salvador\.android\debug.keystore". Pues bien, para obtener nuestra huella digital SHA1 deberemos acceder a dicha ruta desde la consola de comando de Windows y ejecutar los siguientes comandos:

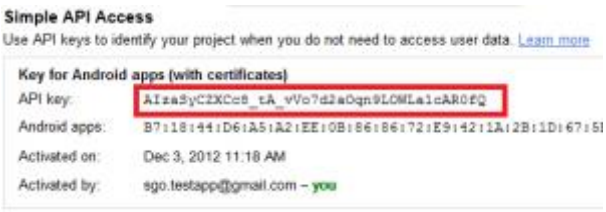
```
C:\>cd C:\Users\Salvador\.android\
C:\Users\Salvador\.android>"C:\Program
Files\Java\jdk1.7.0_07\bin\keytool.exe"    -list    -v    -keystore
debug.keystore -alias androiddebugkey -storepass android -keypass
android
```

Suponiendo que tu instalación de Java está en la ruta "C:\Program Files\Java\jdk1.7.0_07". Si no es así sólo debes sustituir ésta por la correcta. Esto nos deberá devolver varios datos del certificado, entre ellos la huella SHA1.



Ahora nos copiamos este dato y lo añadimos a la ventana de obtención de la API Key donde nos habíamos quedado antes, y a continuación separado por un punto y coma añadimos el paquete java que vayamos a utilizar en nuestra aplicación, que en mi caso será"net.sgoliver.android.mapasapi2".

Pulsamos el botón "Create" y ya deberíamos tener nuestra API Key generada, podremos verla en la pantalla siguiente dentro del apartado "Key for Android Apps (with certificates)". Apuntaremos también este dato para utilizarlo más tarde.



Con esto ya habríamos concluido los preparativos iniciales necesarios para utilizar el servicio de mapas de Android en nuestras propias aplicaciones, por lo que empecemos a crear un proyecto de ejemplo en Eclipse.

Abriremos Eclipse y crearemos un nuevo proyecto estandar de Android, en mi caso lo he llamado "android-mapas-api2". Recordemos utilizar para el proyecto el mismo paquete java que hemos indicado durante la obtención de la API key.

Tras esto lo primero que haremos será añadir al fichero AndroidManifest.xml la API Key que acabamos de generar. Para ello añadiremos al fichero, dentro de la etiqueta <application>, un nuevo elemento <meta-data> con los siguientes datos:

```
...
<application>
...
   <meta-data android:name="com.google.android.maps.v2.API_KEY"
              android:value="api_key"/>
...
</application>
```

Como valor del parámetro android:value tendremos que poner nuestra API Key recien generada.

Siguiendo con el AndroidManifest, también tendremos que incluir una serie de permisos que nos permitan hacer uso de los mapas. En primer lugar tendremos que definir y utilizar un permiso llamado "tu.paquete.java.permission.MAPS_RECEIVE", en mi caso quedaría de la siguiente forma:

```
<permission
     android:name="net.sgoliver.android.mapasapi2.permission.MAPS_
RECEIVE"
     android:protectionLevel="signature"/>
<uses-permission
android:name="net.sgoliver.android.mapasapi2.permission.MAPS_RECEI
VE"/>
```

Además, tendremos que añadir permisos adicionales que nos permitan acceder a los servicios web de Google, a Internet, y al almacenamiento externo del dispositivo (utilizado para la caché de los mapas):

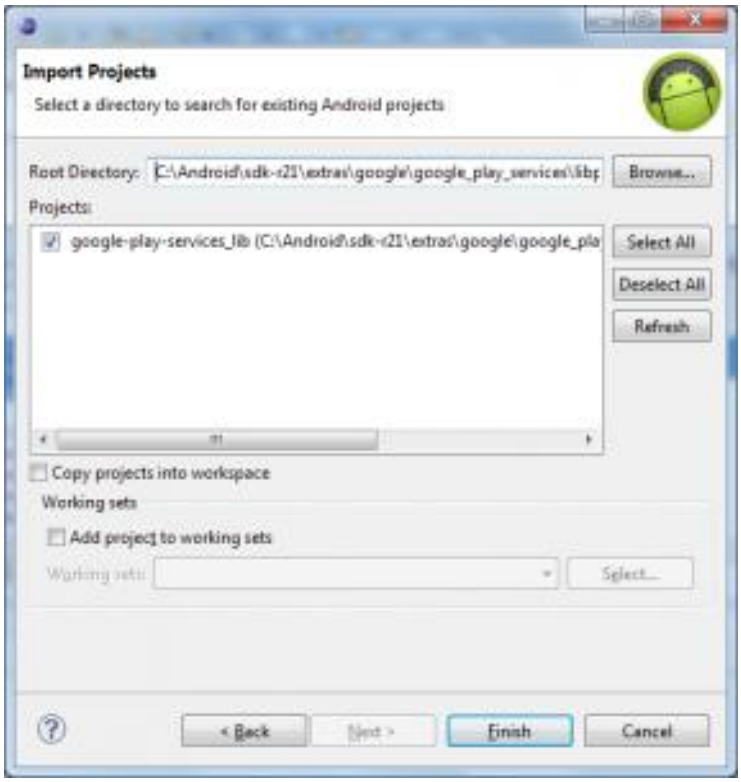
```
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission
android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission
android:name="com.google.android.providers.gsf.permission.READ_GSE
RVICES"/>
```

Por último, dado que la API v2 de Google Maps Android utiliza OpenGL ES versión 2, deberemos especificar también dicho requisito en nuestro AndroidManifest añadiendo un nuevo elemento <uses-feature>:

```
<uses-feature android:glEsVersion="0x00020000"
              android:required="true"/>
```

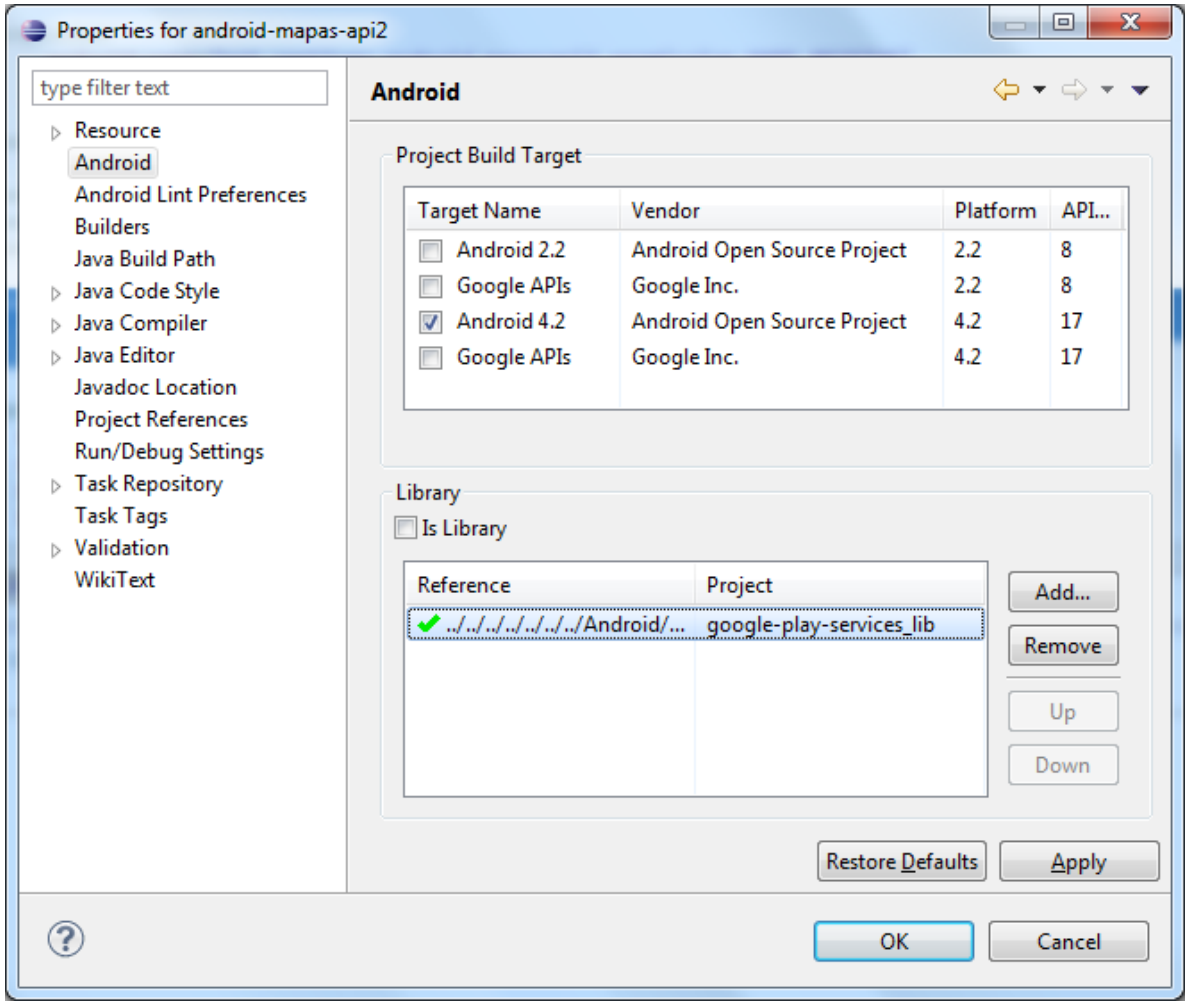
Una vez hemos configurado todo lo necesario en el AndroidManifest, y antes de escribir nuestro código, tenemos que seguir añadiendo elementos externos a nuestro proyecto. El primero de ellos será referenciar desde nuestro proyecto la librería con el SDK de Google Play Services que nos descargamos al principio de este tutorial. Para ello, desde Eclipse podemos importar la librería a nuestro conjunto de proyectos mediante la opción de menú “File / Import… / Existing Android Code Into Workspace”. Como ya dijimos este paquete se localiza en la ruta “<carpeta-sdk-

android>/extras/google/google_play_services/libproject/google-play-services_lib“.

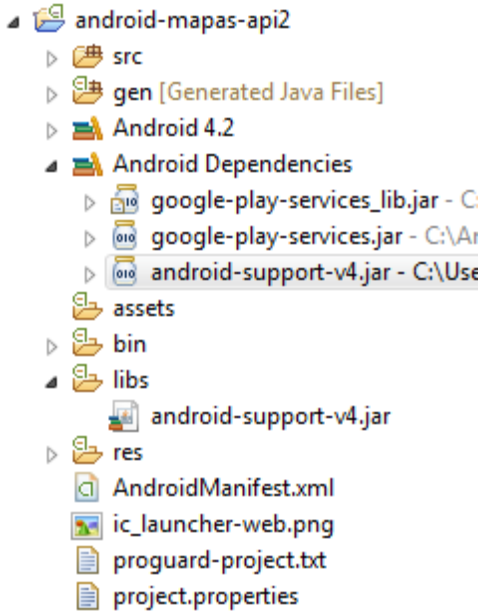


Tras seleccionar la ruta correcta dejaremos el resto de opciones con sus valores por defecto y pulsaremos Finish para que Eclipse importe esta librería a nuestro conjunto de proyectos.

El siguiente paso será referenciar esta librería desde nuestro proyecto de ejemplo. Para ello iremos a sus propiedades pulsando botón derecho / Properties sobre nuestro proyecto y accediendo a la sección Android de las preferencias. En dicha ventana podemos añadir una nueva librería en la sección inferior llamada Library. Cuando pulsamos el botón “Add…” nos aparecerá la librería recien importada y podremos seleccionarla directamente, añadiéndose a nuestra lista de librerías referenciadas por nuestro proyecto.



Como último paso de configuración de nuestro proyecto, si queremos que nuestra aplicación se pueda ejecutar desde versiones "antiguas" de Android (concretamente desde la versión de Android 2.2) deberemos asegurarnos de que nuestro proyecto incluye la librería android-support-v4.jar, que debería aparecer si desplegamos las sección "Android Dependencies" o la carpeta "lib" de nuestro proyecto.



Las versiones más recientes de ADT incluyen por defecto esta librería en nuestros proyectos, pero si no está incluida podéis hacerlo mediante la opción del menú contextual "Android Tools / Add Support Library…" sobre el proyecto, o bien de forma manual.

Y con esto hemos terminado de configurar todo lo necesario. Ya podemos escribir nuestro código. Y para este primer artículo sobre el tema nos vamos a limitar a mostrar un mapa en la pantalla principal de la aplicación. En artículos posteriores veremos como añadir otras opciones o elementos al mapa.

Para esto tendremos simplemente que añadir el control correspondiente al layout de nuestra actividad principal. En el caso de la nueva API v2 este "control" se añadirá en forma de*fragment* (de ahí que hayamos tenido que incluir la librería *android-support* para poder utilizarlos en versiones de Android anteriores a la 3.0) de un determinado tipo (concretamente de la nueva clase com.google.android.gms.maps.SupportMapFragment), quedando por ejemplo de la siguiente forma

```
<?xml version="1.0" encoding="utf-8"?>
<fragment
xmlns:android="http://schemas.android.com/apk/res/android"
       android:id="@+id/map"
       android:layout_width="match_parent"
       android:layout_height="match_parent"
       class="com.google.android.gms.maps.SupportMapFragment"/>
```

Por supuesto, dado que estamos utilizando fragments, la actividad principal también tendrá que extender a FragmentActivity (en vez de simplemente Activity como es lo "normal"). Usaremos también la versión de FragmentActivity incluida en la librería android-supportpara ser compatibles con la mayoría de las versiones Android actuales.

```
public class MainActivity extends
android.support.v4.app.FragmentActivity {
   @Override
   protected void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.activity_main);
   }
...
}
```

Con esto, ya podríamos ejecutar y probar nuestra aplicación. En mi caso las pruebas las he realizado sobre un dispositivo físico con Android 2.2 ya que por el momento parece haber algunos problemas para hacerlo sobre el emulador. Por tanto tendréis que conectar vuestro dispositivo al PC mediante el cable de datos e indicar a Eclipse que lo utilice para la ejecución de la aplicación.

Si ejecutamos el ejemplo deberíamos ver un mapa en la pantalla principal de la aplicación, sobre el que nos podremos mover y hacer zoom con los gestos habituales o utilizando los controles de zoom incluidos por defecto sobre el mapa.

LOCALIZACIÓN GEOGRÁFICA EN ANDROID

Lo primero que tenemos que hacer para poder usar la señal de GPS es añadir al Manifest.xml de nuestro proyecto el permiso "ACCESS_FINE_LOCATION". Para ello escribimos esta línea:

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

El programa lanza un Thread que será el que se encargará de buscar la posición GPS. Se usa un Thread para poder mostrar mientras que está buscando una barra de proceso (ProgressDialog).

Luego en el código JAVA necesitamos un objeto LocationManager, al cual asignaremos un escuchador (LocationListener) para que nos informe cada vez que cambia el estado del GPS mediante el método onLocationChanged. Cuando registremos el escuchador podremos asinarle el tiempo y la distancia mínima para que se refresque el estado de la señal GPS.

El método run() que llama el Thread al ejecutar el método start() es el siguiente:

```
@Override
public void run() {
  mLocationManager =
    (LocationManager)getSystemService(Context.LOCATION_SERVICE);
  if
(mLocationManager.isProviderEnabled(LocationManager.GPS_PROVIDER))
{
    Looper.prepare();
    mLocationListener = new MyLocationListener();
    mLocationManager.requestLocationUpdates(
      LocationManager.GPS_PROVIDER, 0, 0, mLocationListener);
    Looper.loop();
    Looper.myLooper().quit();
  } else {
    Toast.makeText(getBaseContext(),
      getResources().getString(R.string.gps_signal_not_found),
      Toast.LENGTH_LONG).show();
  }
}
```

Aquí se puede ver como se usa el método requestLocationUpdates del objeto LocationManager para registrar el objeto LocationListener.

Varias cosas debemos de saber. En Android podemos manejar las localizaciones a patir de proveedores (Providers). El proveedor GPS_PROVIDER es nuestro GPS y en este código estamos pidiéndole a él que nos diga la posición. Además, para trabajar con Threads en Android es necesario llamar a Looper.prepare(); al principio del método run() y Looper.loop(); al final. Realmente no se para que funcionan, supongo que para controlar el estado del Thread, pero sin estas líneas el programa dará un error en tiempo de ejecución.

También tenemos un objeto Handler al cual llamaremos cuando tengamos la localización de nuestro GPS y que se encargará de

cerrar la ventana de proceso y de establecer los valores del longitud y latitud en los TextView. Esto último es importante, ya que no podemos establecer los valores de los TextView desde el método onLocationChanged del LocationListener ya que para establecer los valores del GUI no se puede hacer desde un Thread que no sea el principal.

Simular una señal de GPS en el emulador

Para poder probar la aplicación en el emulador debemos de mandar una señal de GPS a este.

Una alternativa es hacerlo mediante telnet. Para ello abrimos un terminal (tanto en Windows como en Linux) y escribimos:

```
$ telnet localhost 5554
```

Esto hará que nos podamos conectar al emulador (que tiene que estar ejecutándose, por supuesto). Una vez que estemos en la Android Conlose simplemente escribimos lo siguiente:

```
$ geo fix 6.46466 8.6456445
```

Donde el primero valor es la latitud y el segundo la longitud.

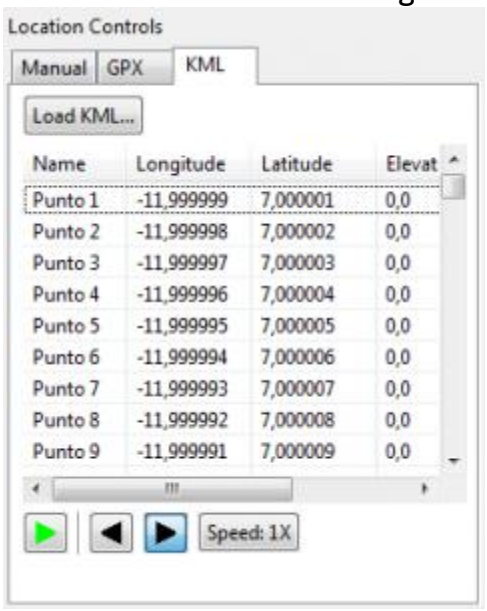
Para probarlo en el programa seguimos los siguientes pasos:

1. Abrimos el emulador y lanzamos el programa
2. Pulsamos en el botón "Buscar posición GPS" y lanzará el diálogo "Buscando..."
3. Ahora abrimos la consola y hacemos el telnet
4. Lanzamos la geoposición que queramos
5. El programa debería de decirlos la posición que le hemos enviado

También una forma menos manual de simular cambios frecuentes de posición para probar nuestras aplicaciones. Este método consiste en proporcionar, en vez de una sóla coordenada cada vez, una lista de coordenadas que se iran enviando automáticamente al emulador una tras otra a una determinada velocidad, de forma que podamos simular que el dispositivo se mueve constantemente y que nuestra aplicación responde de forma correcta y en el momento adecuado a esos cambios de posición. Y esta lista de coordenadas

se puede proporcionar de dos formas distintas, en formato GPX o como fichero KML. Ambos tipos de fichero son ampliamente utilizados por aplicaciones y dispositivos de localización, como GPS, aplicaciones de cartografía y mapas, etc. Los ficheros KML podemos generarlos por ejemplo a través de la aplicación Google Earth o manualmente con cualquier editor de texto.

Para utilizar este fichero como fuente de datos para simular cambios en la posición del dispositivo, accedemos nuevamente a los Location Controls y pulsamos sobre la pestaña GPX o KML, según el formato que hayamos elegido, que en nuestro caso será KML. Pulsamos el botón "Load KML…" para seleccionar nuestro fichero y veremos la lista de coordenadas como en la siguiente imagen:



Una vez cargado el fichero, tendremos disponibles los cuatro botones inferiores para (de izquierda a derecha):

- Avanzar automáticamente por la lista.
- Ir a la posición anterior de la lista de forma manual.
- Ir a la posición siguiente de la lista de forma manual.
- Establecer la velocidad de avance automático.

ANDROID TIPS

- Si se ejecuta un proyecto en un dispositivo virtual y luego se hacen modificaciones de código o se corre en modo debug,

no es necesario cerrar la instancia del AVD sino que simplemente hay que presionar el botón Run.

- Utilizar siempre Relative Layouts para tener GUI utilizables en todas las resoluciones.
- Aprovechar los recursos .xml disponibles, por ejemplo strings, colours y styles.
- Definir en el Manifest de la aplicación todas las características de la aplicación, de esta manera se evitará que se instale en dispositivos que no la soporten evadiendo los puntajes y comentarios negativos.
- Ubicar en las carpetas específicas para las distintas pantallas las imágenes diseñadas para las distintas resoluciones.
- El éxito de una aplicación para Android radica principalmente en el diseño de la interfaz gráfica y la interacción con los recursos del teléfono (por ejemplo notificaciones, widgets, luces, libreta de contacto,).
- Tener un buen ícono para la aplicación. Android Icons es un sitio web de donde pueden descargar íconos gratuitamente,
- Utilizar herramientas para hacer el prototipado de las UI. Droid Draw es un proyecto de Software Libre que aplica para este consejo.
- La mayores optimizaciones se pueden hacer en base al diseño de las UI.
- Utilizar “sp” (space independent pixels) o “dip” (distance independent pixels) en vez de “px” para definir medidas ya que de esta forma podremos tenes elementos escalables a las distintas densidades de pantallas.
- Desarrollar con la última versión de Android existente y hacer las pruebas sobre la versión más antigua soportada (por ejemplo Android 1.5).
- Hacer las pruebas sobre muchas configuraciones de dispositivos virtuales distintas.
- Tener cuidado con la Relative Layout en Android 1.5 y 1.6 ya que difieren las referencias a las posiciones de los elementos.