



**maestros  
del web**

# CURSO **ANDROID**

Desarrollo de  
aplicaciones móviles

CATEGORÍA   
**PROGRAMACIÓN**



NIVEL  
**INTERMEDIO**



SOBRE LA GUÍA

# Curso Android: Desarrollo de aplicaciones móviles

Versión 1 / junio 2011

Nivel: Básico / Intermedio

El Curso Android se encuentra en línea en:

<http://www.maestrosdelweb.com/editorial/curso-android>

Un proyecto de Maestros del Web

- ▶ Edición: Stephanie Falla Aroche
- ▶ Diseño y diagramación: Iván E. Mendoza
- ▶ Autor: Adrián Catalán

Este trabajo se encuentra bajo una licencia Creative Commons

Atribución-NoComercial-CompartirIgual 3.0 Unported (CC BY-NC-SA 3.0)

El logotipo de Android es compartido por Google bajo licencia Creative Commons por Atribución.

## Contacto

<http://www.maestrosdelweb.com/sitio/correo/>

## Redes sociales

Facebook: <http://www.facebook.com/maestrosdelweb>

Twitter: <http://www.twitter.com/maestros>

## SOBRE EL AUTOR

# Adrián Catalán



Guatemalteco, Ingeniero en Sistemas con una Maestría en Tecnologías de Información con especialidad en Seguridad informática y Redes de Computadoras de la Universidad Galileo.

Es catedrático de cursos sobre arquitectura de computadoras, máquinas y lenguajes, compiladores, bases de datos, ingeniería de software, desarrollo web, sistemas operativos, redes, voz sobre IP, seguridad informática y métodos forenses en IT.

Desarrollador de software, apasionado por aplicaciones web (Ruby On Rails) y móviles (Android), fundador del grupo de tecnologías de Google en Guatemala(GTUG) y co fundador de Elemental Geeks es un firme creyente del infinito talento hispanoamericano, consciente de la necesidad de fomentarlo y darlo a conocer.

Puedes comunicarte a través de redes sociales:

Email: [adriancatalan@gmail.com](mailto:adriancatalan@gmail.com)

Facebook: [adrian.catalan](https://www.facebook.com/adrian.catalan)

Twitter: [@ykro](https://twitter.com/@ykro)



## ÍNDICE

1   Sobre la guía .....	2
2   Sobre el autor .....	3
3   Construir un lector de feeds simple .....	5
4   Aumentar la funcionalidad de un lector de feeds .....	23
5   Trabajando con imágenes (cámara y galería) .....	44
6   Grabación y reproducción de vídeo.....	55
7   Geolocalización y utilización de mapas de Google .....	64
8   Trabajar con el acelerómetro .....	73
9   Reproducción de sonido en un ciclo infinito .....	83
10   Envío de email utilizando Android.....	90
11   Trabajando con APIs (Facebook y Twitter).....	99
12   Conectándonos con Apis de Google.....	117
13   Comunidades android .....	134
14   Otras guías de maestros del web .....	135

1

capítulo



# Construir un lector de feeds simple

## CAPÍTULO 1

## Construir un lector de feeds simple

Bienvenidos al primer capítulo del curso sobre Android daremos inicio con el funcionamiento de la arquitectura de esta plataforma móvil y los bloques básicos de una aplicación.

### Kernel de Linux

En la base tenemos el kernel 2.6 de Linux, Android lo utiliza por su robustez demostrada y por la implementación de funciones básicas para cualquier sistema operativo, por ejemplo: seguridad, administración de memoria y procesos, implementación de conectividad de red (network stack) y varios interpretes (drivers) para comunicación con los dispositivos físicos(hardware).



Android utiliza como base el kernel de Linux pero los dos sistemas no son lo mismo, Android no cuenta con un sistema nativo de ventanas de Linux ni tiene soporte para glibc (librería estándar de C) ni tampoco es posible utilizar la mayoría de aplicaciones de GNU de Linux.

Además de todo lo ya implementado en el kernel de Linux, Android agrega algunas cosas específicas para plataformas móviles como la comunicación entre procesos (lograda a través del binder), la forma de manejar la memoria compartida (ashmem) y la administración de energía (con wakelocks). De las características únicas del kernel utilizado por Android encuentran más información en [Android Kernel Features](#)<sup>1</sup>.

### Librerías y ejecución

Sobre el kernel, tenemos un conjunto de librerías de C y C++ utilizadas por el sistema para varios fines

<sup>1</sup> [http://elinux.org/Android\\_Kernel\\_Features](http://elinux.org/Android_Kernel_Features)

como el manejo de la pantalla (surface manager), mapas de bits y tipos de letra (Free Type), gráficas en 2D y 3D (SGL y OpenGL), manejo de multimedia (Media Framework), almacenamiento de datos (SQLite) y un motor para las vistas web y el navegador (WebKit).

Junto a estas librerías, encontramos lo necesario para la ejecución de las aplicaciones a través de la máquina virtual Dalvik. Cada aplicación utiliza una instancia de la máquina virtual ejecutando un archivo DEX (Dalvik Executable) y el sistema está optimizado para que se ejecuten múltiples instancias de la máquina virtual. Se desarrolla en Java pero no se utiliza una máquina virtual de Sun para su ejecución ni tampoco archivos CLASS.



## Estructura de aplicaciones

Sobre las librerías encontramos una estructura que nos brinda un contexto para desarrollar, este framework permite a los desarrolladores aprovechar un sistema de vistas ya construido, administrar notificaciones y accesar datos a través de proveedores de contenido entre otras cosas.



## Aplicaciones

Las aplicaciones centrales que incluye el sistema por defecto son: teléfono, navegador, manejo de contactos, etc. En esta capa de la arquitectura es donde trabajaremos desarrollando aplicaciones.

## Bloques básicos de una aplicación

Una vez vista la arquitectura, empezaremos con lo fundamental para desarrollar una aplicación. Los



componentes básicos de una aplicación son: *activities, intents, views, services, content providers* y *broadcast receivers*.

Si dominan bien estos términos pueden saltar directo al código.



- ▶ **Activities<sup>1</sup>**: son componentes de la interfaz que corresponde a una pantalla. Podemos visualizarlo como un mazo de cartas en el que tenemos varias cartas pero solamente una está hasta arriba. Una aplicación para una lista de cosas por hacer (remember the milk) puede tener una actividad para ingresar las cosas por hacer y otra actividad para mostrar el listado, en conjunto estas actividades conforman la aplicación.
- ▶ **Intents<sup>2</sup>**: son mensajes que provocan notificaciones o cambios de estatus, que al ser recibidos por actividades o servicios pueden levantar procesos. De esta forma se unen componentes dentro de la misma aplicación o de diferentes aplicaciones.
- ▶ **Views<sup>3</sup>**: son los componentes de la interfaz de usuario, diferentes vistas pueden agruparse a través de grupos logrando una jerarquía, esto se logra a través de la disposición de los componentes a través de un archivo XML.
- ▶ **Services<sup>4</sup>**: son componentes que ejecutan operaciones en segundo plano y no tienen una interfaz

1 <http://developer.android.com/guide/topics/fundamentals/activities.html>

2 <http://developer.android.com/guide/topics/intents/intents-filters.html>

3 <http://developer.android.com/guide/topics/ui/index.html>

4 <http://developer.android.com/guide/topics/fundamentals/services.html>



de usuario. Por ejemplo, al escuchar música, hay un servicio encargado de la reproducción que se ejecuta de fondo y la aplicación que manipulamos le manda mensajes a este servicio diciéndole que se detenga, pause o reproduzca la siguiente canción.

- ▶ **Content Providers<sup>1</sup>**: representan la abstracción para almacenar y obtener datos permanentes y aplicaciones diferentes. El sistema incluye algunos proveedores de contenido útiles (audio, video, etc) y además pueden desarrollarse nuevos.
- ▶ **Manifest<sup>2</sup>**: El archivo AndroidManifest.xml es donde se configura la aplicación, se agregan actividades, se asignan permisos, etc.
- ▶ **Broadcast Receivers**: son componentes que responden a avisos y anuncios de difusión (broadcast). Estos avisos provienen del sistema (batería baja, una llamada entrante, etc) y de aplicaciones (pasando avisos de una aplicación a otra). Aun que no muestran una interfaz de usuario algunas veces utilizan barras de progreso para mostrar avances. Estos se activan a través de mensajes asíncronicos llamados intents (mencionados arriba).

## Ejemplo: Lector de feeds

La aplicación que realizaremos es un lector para el feed de Maestros del Web, queremos que al finalizar se vea de la siguiente forma:

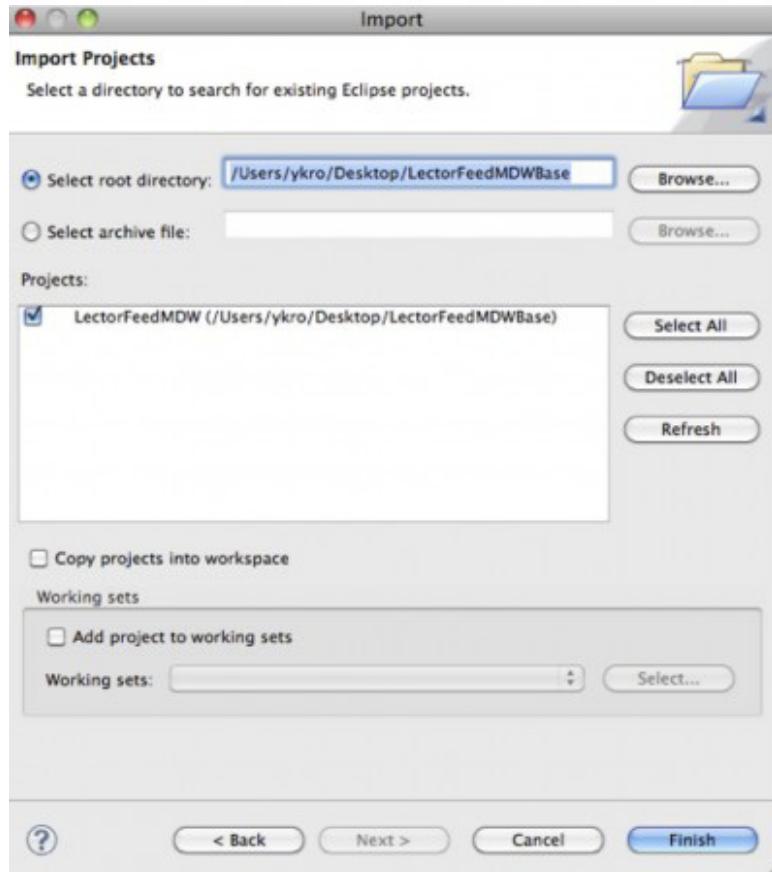


1 <http://developer.android.com/guide/topics/providers/content-providers.html>

2 <http://developer.android.com/guide/topics/manifest/manifest-intro.html>

## Disposición inicial

Arrancamos **descargando el código base**<sup>1</sup> que debe ser importado hacia un proyecto nuevo. En mi caso, el código descargado lo tengo en el folder “/Users/ykro/Desktop/LectorFeedMDWBase”:



En este código encontrarás algunas cosas de ayuda para el desarrollo del proyecto. Si quisieran iniciar desde cero, con un proyecto nuevo, lo mínimo que deberían hacer para tenerlo listo es:

- » Darle permiso a la aplicación para que pueda accesar a internet

Para hacer esto, vamos al archivo *AndroidManifest.xml* y justo antes de la etiqueta que cierra *manifest* colocamos:

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

<sup>1</sup> <https://github.com/androidMDW/guia1base>

- » Tener un LinearLayout con orientación vertical en el archivo de diseño principal.

Para lograr esto, en el archivo `/res/layout/main.xml` es necesario tener:

```
<LinearLayout android:id="@+id/LL01" android:layout_width="fill_parent"
    android:layout_height="fill_parent" android:orientation="vertical"
    xmlns:android="http://schemas.android.com/apk/res/android">
</LinearLayout>
```

- » Tener una forma de reconocer XML.

En el código de demo se provee la clase `src/com/android/mdw/demo/XMLParser.java`. No vamos a repasar el código de cómo se hace el reconocimiento de XML, este parser tiene un funcionamiento general sin embargo para los fines de esta aplicación nos interesa que devuelva una lista simplemente encadenada (`LinkedList` de Java) y en cada posición un diccionario (`HashMap` de Java) porque de esa forma vamos a almacenar los datos.

Esto se logra a través de un método llamado `parse()` y adicional a esto, en el constructor el `XMLParser` debe recibir la dirección del feed que reconocerá.

## Diseño

Trabajaremos con los siguientes views para lograr el objetivo.

- » **LinearLayout<sup>1</sup>**: agrupa los elementos en un solo sentido (vertical u horizontal).
- » **ListView<sup>2</sup>**: muestra los elementos en un listado vertical con scroll.
- » **TextView<sup>3</sup>**: una etiqueta para colocar texto.
- » **Button<sup>4</sup>**: un botón para presiona.

Vamos a colocar primero un `LinearLayout` con arreglo vertical y dentro de él dos cosas:

1 <http://developer.android.com/guide/tutorials/views/index.html>

2 <http://www.maestrosdelweb.com/editorial/curso-android-construir-lector-de-feeds/>

3 <http://developer.android.com/reference/android/widget/TextView.html>

4 <http://developer.android.com/reference/android/widget/Button.html>

- 1 | LinearLayout con arreglo horizontal (para la etiqueta y el botón).
- 2 | ListView para los elementos del feed.

Nuestro archivo `/res/layout/main.xml` quedaría entonces de la siguiente forma:

Código fuente



```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout android:id="@+id/LL01"
    android:layout_width="fill_parent" android:layout_height="fill_parent"
    android:orientation="vertical" xmlns:android="http://schemas.android.com
    /apk/res/android">
    <LinearLayout android:id="@+id/LL02"
        android:orientation="horizontal" android:layout_width="wrap_content"
        android:layout_height="wrap_content">
        <TextView android:id="@+id/tvLoadTip"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" android:text="Presione
            el botón para cargar datos">
        </TextView>
        <Button android:id="@+id/btnLoad"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Enviar">
        </Button>
    </LinearLayout>
    <ListView android:id="@+id/lstData"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:padding="5dip">
    </ListView>
</LinearLayout>
```

Es importante notar los identificadores que colocamos, ya que los utilizaremos más adelante.

Mostrar datos siguiendo el diseño

Luego de terminado el diseño podemos enfocarnos en el código que nos permitirá poblar el ListView con los elementos del feed que vamos a leer.

Trabajemos ahora en el archivo `src/com/android/mdw/demo/Main.java`.

Inicialmente encontramos 2 constantes que utilizaremos para guardar la información en un HashMap.

```
static final String DATA_TITLE = "T";
static final String DATA_LINK = "L";
```

Agregamos una estructura de datos estática para guardar la información del feed una vez la fuimos a traer y reconocimos.

```
static LinkedList<HashMap<String, String>> data;
```

En una aplicación real esto no debería hacerse nunca, todo campo estático (variable de clase) **debería ser final<sup>1</sup>** y en este caso no lo es porque lo asignaremos luego de que el parser devuelva la data. La forma correcta de realizar este almacenamiento volátil es utilizando clases de aplicación, algo que veremos en próximos capítulos y si se quisiera almacenamiento no volátil se tiene acceso a base de datos a través de SQLite. Haremos también una función auxiliar que recibe una lista de mapas, y utilizando esta data crea un adaptador para poblar al ListView del diseño.

Código fuente



```
private void setData(LinkedList<HashMap<String, String>> data) {
    SimpleAdapter sAdapter = new SimpleAdapter(getApplicationContext(), data,
        android.R.layout.two_line_list_item,
        new String[] { DATA_TITLE, DATA_LINK },
        new int[] { android.R.id.text1, android.R.id.text2 });
    ListView lv = (ListView) findViewById(R.id.lstData);
    lv.setAdapter(sAdapter);
```

1 [http://en.wikipedia.org/wiki/Final\\_%28Java%29](http://en.wikipedia.org/wiki/Final_%28Java%29)



{}

Nuestro objetivo final es tomar una vista (ListView) creada en el diseño y poblarla de datos a través de un adaptador . La información que representaremos por cada fila del listado es el título del artículo y el link del mismo. Para nuestro caso vamos a utilizar el adaptador simple (SimpleAdapter) que recibe 5 parámetros.

- 1 | Un contexto sobre el cual puede trabajar, lo obtenemos llamando a `getApplicationContext()`
- 2 | Un Listado de mapas con la data a mostrar, lo recibimos como parámetro en la función.
- 3 | Un Layout para cada fila, en este caso usaremos uno de los predeterminados de Android llamado `android.R.layout.two_line_list_item` este nos permite tener elementos con información en 2 líneas.
- 4 | Un arreglo de `String` con las llaves del diccionario que representarán los datos obtenidos de la lista especificada anteriormente.
- 5 | Un arreglo de `int` con los identificadores de los elementos de cada línea.

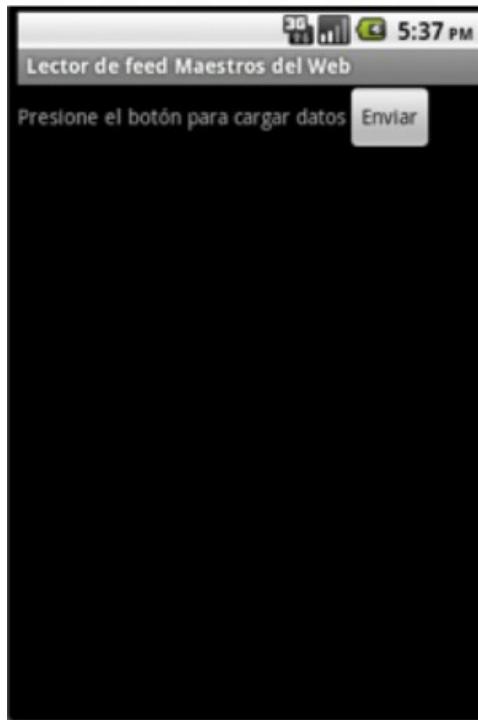
Con esta configuración, el adaptador nos dará elementos de 2 líneas mostrando en la primera el título del artículo y en la segunda el link del artículo. Esta configuración resulta muy útil porque todo está hecho, sin embargo no es personalizable, si queremos cambiar algunas cosas (fondo, color, letra, etc) de cada elemento sería necesario agregar otro layout para cada fila y si deseamos mostrar más información o no utilizar una lista de mapas entonces sería necesario hacer nuestro propio adaptador (esto lo haremos en próximos capítulos).

## Carga de datos

La carga de datos nos tomará cierto tiempo, vamos a mostrarle un aviso de que los datos se están cargando. Queremos que el aviso se muestre y como tarea de fondo cargue los datos (esto NO es un servicio) vamos a necesitar de un **hilo de ejecución**<sup>1</sup> que haga esta tarea.

---

<sup>1</sup> <http://download.oracle.com/javase/1.5.0/docs/api/java/lang/Thread.html>



Android nos presenta la restricciones que no podemos alterar, los elementos de interfaz gráfica en un hilo de ejecución que no sea el principal por lo que es necesario utilizar un manejador para enviar un mensaje de un hilo a otro cuando la carga de datos haya terminado. Agregaremos entonces una variable para el diálogo de progreso es necesaria una variable global porque iniciamos el diálogo en una función y lo ocultamos en otra.

```
private ProgressDialog progressDialog;
```

Otra para el manejador de mensajes entre hilos de ejecución:

Código fuente



```
private final Handler progressHandler = new Handler() {  
    @SuppressWarnings("unchecked")  
    public void handleMessage(Message msg) {  
        if (msg.obj != null) {  
            data = (LinkedList<HashMap<String, String>>)msg.obj;  
            setData(data);  
        }  
    }  
}
```

```
progressDialog.dismiss();  
}  
};
```

Para el manejador, haremos una **clase interna y anónima**<sup>1</sup> este tipo de clases las verán seguido para agregar funcionalidades que requieren de clases pero su tarea es tan pequeña que no vale la pena darle un nombre (y luego terminar con muchos nombres que se utilizan una sola vez).

Dentro de la clase *Handler* hemos agregado la anotación `@SuppressWarnings ("unchecked")` para evitar una advertencia por la conversión de tipo (*type casting*) de objeto hacia lista realizada adelante. Es necesario implementar el método *handleMessage (Message msg)* y es aquí donde revisamos si viene un mensaje, lo convertimos a lista, llamamos al método *setData* previamente definido y cerramos el diálogo de progreso.

Por último, vamos a hacer una función auxiliar que inicia la carga de datos, muestra al usuario un diálogo de que se están cargando los datos y levanta un thread para lograr la carga.

Código fuente



```
private void loadData() {  
    ProgressDialog progressDialog = ProgressDialog.show(  
        Main.this,  
        "",  
        "Por favor espere mientras se cargan los datos...",  
        true);  
    new Thread(new Runnable() {  
        @Override  
        public void run() {  
            XMLParser parser = new XMLParser(feedUrl);  
            Message msg = progressHandler.obtainMessage();  
            msg.obj = parser.parse();  
            progressHandler.sendMessage(msg);  
        }  
    }).start();
```

<sup>1</sup> [http://www.google.com/url?q=http%3A%2F%2Fdownload.oracle.com%2Fjavase%2Ftutorial%2Fjava%2FjavaOO%2FInherclasses.html&sa=D&sntz=1&usg=AFQjCNEbFcVxidDxasA0Rg3ED01\\_7EDHfA](http://www.google.com/url?q=http%3A%2F%2Fdownload.oracle.com%2Fjavase%2Ftutorial%2Fjava%2FjavaOO%2FInherclasses.html&sa=D&sntz=1&usg=AFQjCNEbFcVxidDxasA0Rg3ED01_7EDHfA)

{}

Para el diálogo de progreso, le enviamos 4 argumentos:

- 1 | Contexto, dado por la aplicación, como esta función la vamos a llamar desde una clase interna, el acceso es con el campo this de la clase Main.
- 2 | Título, no nos interesa que tenga título el aviso.
- 3 | Mensaje, le pedimos al usuario que espere.
- 4 | Indeterminado, no se puede cuantificar cuanto tiempo falta o cuanto progreso se ha logrado entonces el aviso será de duración indeterminada.

Luego tenemos un Thread a partir de otra clase interna y anónima que es una instancia de Runnable y definimos el método run donde se especifica lo que hará el hilo de ejecución y llamamos al método start para iniciarla. En el método run instanciamos un parser nuevo de XML construimos un mensaje a partir del manejador y se lo enviamos con la data obtenida del parser.

### Llamando todo desde la función onCreate

Trabajemos dentro de la función onCreate la cual se llama cuando la actividad es creada, primero le ponemos un bonito título a la ventana.

```
setTitle("Lector de feed Maestros del Web");
```

El siguiente paso es asociarle una acción al botón para que cargue los datos, primero obtenemos el botón haciendo uso de la función findViewById con el identificador colocado en el diseño para posteriormente colocarle las acciones a realizar cuando se presione a través del método setOnClickListener. Este método recibe una instancia de la clase OnClickListener, de nuevo vamos a hacer una clase anónima y sobrecargaremos el método onClick.

Código fuente



```
Button btn = (Button) findViewById(R.id.btnLoad);  
btn.setOnClickListener(new OnClickListener() {  
    @Override  
    public void onClick(View v) {
```

```
}
```

```
});
```

Dentro de la función `onClick` validaremos si el `ListView` ya tiene datos, si no los tiene los cargamos con una llamada a `loadData` pero si ya los tiene vamos a mostrarle un diálogo de alerta al usuario preguntando si está seguro que desea hacerlo.

Código fuente



```
ListView lv = (ListView) findViewById(R.id.lstData);  
if (lv.getAdapter() != null) {  
    AlertDialog.Builder builder = new AlertDialog.Builder(Main.this);  
    builder.setMessage("ya ha cargado datos, ¿Está seguro de hacerlo de  
nuevo?")  
        .setCancelable(false)  
        .setPositiveButton("Si", new DialogInterface.OnClickListener()  
        {  
            public void onClick(DialogInterface dialog, int  
id) {  
                loadData();  
            }  
        })  
        .setNegativeButton("No", new DialogInterface.OnClickListener()  
        {  
            public void onClick(DialogInterface dialog, int  
id) {  
                dialog.cancel();  
            }  
        })  
        .create()  
        .show();  
} else {  
    loadData();  
}
```



Para crear el diálogo de alerta, encadenamos llamadas a partir de un constructor `AlertDialog.Builder` que recibe un contexto y de nuevo utilizamos `Main.this`.

Estas llamadas encadenadas nos permiten:

- ▶ Establecer un mensaje (`setMessage`)
- ▶ Obligar al usuario a responder y no solo cerrar (`setCancelable`)
- ▶ Colocar una acción cuando la respuesta es positiva (`setPositiveButton`)
- ▶ Colocar una acción cuando la respuesta es negativa (`setNegativeButton`)
- ▶ Crear el diálogo (`create`)
- ▶ Mostrar el diálogo (`show`)

Las más complicadas son las llamadas para colocar acciones ante una respuesta positiva o negativa. Ambas requieren de un Listener que se disparara cuando el usuario presione el botón. Dentro del método `onClick` de otra clase anónima más, cuando es positivo llamamos a `loadData` y cuando es negativo ocultamos el diálogo.

El código completo queda de esta forma:

#### Código fuente



```
Button btn = (Button) findViewById(R.id.btnLoad);  
btn.setOnClickListener(new OnClickListener() {  
    @Override  
    public void onClick(View v) {  
        ListView lv = (ListView) findViewById(R.id.lstData);  
        if (lv.getAdapter() != null) {  
            AlertDialog.Builder builder = new AlertDialog.Builder(Main.this);  
            builder.setMessage("ya ha cargado datos, ¿Está seguro de hacerlo de  
nuevo?")  
                .setCancelable(false)  
                .setPositiveButton("Si", new  
                    DialogInterface.OnClickListener() {  
                        public void onClick(DialogInterface dialog, int id) {  
                            loadData();  
                        }  
                    })  
                .setNegativeButton("No", new  
                    DialogInterface.OnClickListener() {  
                        public void onClick(DialogInterface dialog, int id) {  
                            dialog.cancel();  
                        }  
                    })  
            builder.show();  
        }  
    }  
});
```



```
loadData();  
}  
})  
.setNegativeButton("No", new  
DialogInterface.OnClickListener() {  
public void onClick(DialogInterface  
dialog, int id) {  
dialog.cancel();  
}  
})  
.create()  
.show();  
} else {  
loadData();  
}  
}  
});
```

Para terminar, queremos aprovechar que contamos con el link de cada artículo y que cuando el usuario presione algún elemento en nuestra aplicación pueda visitarse el artículo vía el navegador.

Código fuente



```
ListView lv = (ListView) findViewById(R.id.lstData);  
lv.setOnItemClickListener(new OnItemClickListener() {  
@Override  
public void onItemClick(AdapterView<?> av, View v, int position, long id)  
{  
HashMap<String, String> entry = data.get(position);  
Intent browserAction = new Intent(Intent.ACTION_VIEW,  
Uri.parse(entry.get(DATA_LINK)));  
startActivity(browserAction);  
}  
});
```

Esto lo conseguimos modificando el evento de click sobre los ítems de la lista (setOnItemClickListener) primero obteniendo el elemento que ha sido presionado y luego con un intent de **ACTION\_VIEW**<sup>1</sup> este intent predefinido realiza la acción más obvia para mostrar datos al usuario, en nuestro caso se le envía un **URI**<sup>2</sup> lo que permite visualizarlo con el navegador.

1 [http://developer.android.com/reference/android/content/Intent.html#ACTION\\_VIEW](http://developer.android.com/reference/android/content/Intent.html#ACTION_VIEW)

2 <http://developer.android.com/reference/android/net/Uri.html>



¿Dudas?

Pregunta en el Foro Android  
de Foros del Web



Descarga el código  
completo en

```
if (true) {  
    Intent intent = new Intent(Intent.ACTION_VIEW);  
    AlertDialod.Builder builder = new  
    AlertDialod.Builder(Main.this);  
    builder.setMessage("ya ha cargado datos. ¿Está seguro de  
    hacerlo de nuevo?");  
    builder.setCancelable(false);  
    builder.setPositiveButton("SI",  
        new DialogInterface.OnClickListener(){  
            public void onClick(DialogInterface dialog, int id) {  
                //...  
            }  
        });  
    builder.show();  
}
```

**github**  
SOCIAL CODING





# Conclusión

En esta guía hemos visto varias cosas:

- **Arquitectura de Android:** conocimos la forma en que está construída la arquitectura.
- **Bloques básicos de una aplicación:** los elementos principales de una aplicación de Android.
- **Manifest:** aprendimos la forma y la ubicación del manifest, archivo para configuración de la aplicación.
- **LinearLayout, TextView y Button:** tuvimos un primer acercamiento a los elementos de la UI a través de estos 3 controles, una forma de manejar el diseño (layout) y dos controles para la interfaz de la aplicación.
- **Eventos OnClick:** aprendimos a manejar los eventos con listeners y asociar acciones para cuando el usuario interactúa con los elementos de la UI.
- **Dialogs:** iniciamos a trabajar con diálogos para mostrar avisos al usuario, en este caso únicamente de confirmación previo a la carga de datos.
- **Threads y Handlers:** el proceso de la carga y parsing de datos fue realizado con un thread extra y para la comunicación con el thread principal utilizamos un Message Handler.
- **Iniciar otras activities:** aprendimos el mecanismo necesario para iniciar otras activities aun que no fueran hechas por nosotros.

# 2

capítulo



## Aumentar la funcionalidad de un lector de feeds



## CAPÍTULO 2

## Aumentar la funcionalidad de un lector de feeds

El lector de feed que desarrollaremos en este capítulo se diferencia del anterior porque permitiremos al usuario decidir si al presionar sobre un elemento quiere verlo en una vista previa dentro de la aplicación o en el navegador, para esta elección agregaremos un menú y personalizaremos el View de cada fila del resultado.

El ejemplo se verá de la siguiente manera:



## Disposición inicial

Empecemos **descargando el código<sup>1</sup>** que debe ser importado hacia un nuevo proyecto es muy similar al del capítulo anterior, pero revisemos las diferencias:

- El XMLParser ahora obtiene no solo el link y título de cada artículo, también su autor, fecha, descrip-

<sup>1</sup> <https://github.com/androidMDW/guia2base>



ción y una imagen.

- Para ser más ordenados, la información se guardará utilizando una clase de Java que represente a cada artículo llamada Element (src/com/android/mdw/demo/Element.java)
- La *Activity* principal (Main) hereda de *ListActivity* y no de *Activity* facilitando y permite caminos cortos para cosas específicas a una Activity que muestra un listado de cosas. Por lo mismo, en el layout es necesario un ListView con el identificador @*android:id/list*

Además, tiene las especificaciones de la guía anterior.

- Es necesario darle permiso a la aplicación para que pueda accesar a Internet, tener un LinearLayout con orientación vertical en el archivo de diseño principal por trabajar con una ListActivity también necesitamos el ListView con id @*android:id/list* y tener una forma de reconocer XML.

## Diseño

Trabajamos con Views del capítulo anterior como: LinearLayout, ListView, TextView, Button y agregamos nuevas Views.

- **ImageView**<sup>1</sup> : muestra una imagen que puede cargarse de diferentes fuentes para el ejemplo vamos a obtenerla de una dirección de Internet (etiqueta img de HTML dentro del artículo) revisar ([2d Graphics](#)<sup>2</sup>).
- **Menu**<sup>3</sup> y **MenuItem**<sup>4</sup> : en este capítulo aprenderás cómo hacer un menú de opciones en XML, revisar ([Menus](#)<sup>5</sup> ).

Para el layout principal, repetimos el procedimiento del capítulo anterior de nuevo recordando que el identificador (*android:id*) del ListView debe ser @*android:id/list*.

Además, necesitaremos 3 Views adicionales:

---

1 <http://developer.android.com/reference/android/widget/ImageView.html>

2 <http://developer.android.com/guide/topics/graphics/2d-graphics.html>

3 <http://developer.android.com/reference/android/view/Menu.html>

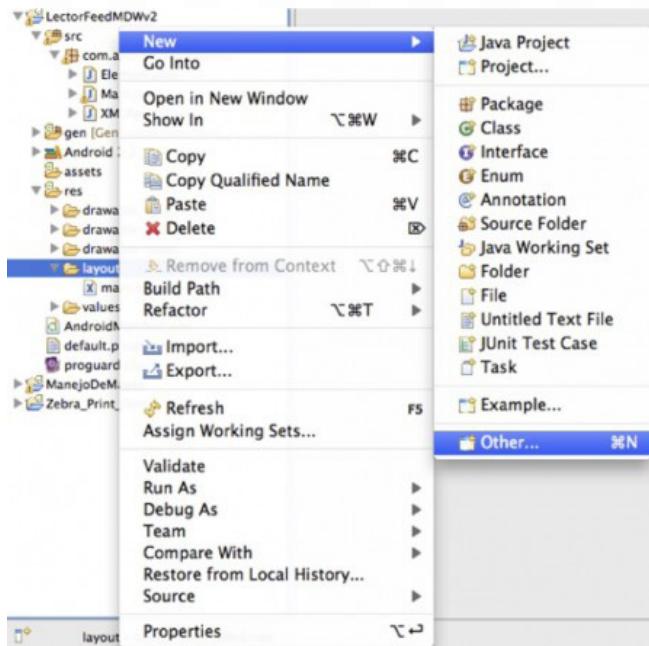
4 <http://developer.android.com/reference/android/view/MenuItem.html>

5 <http://developer.android.com/guide/topics/ui/menus.html>

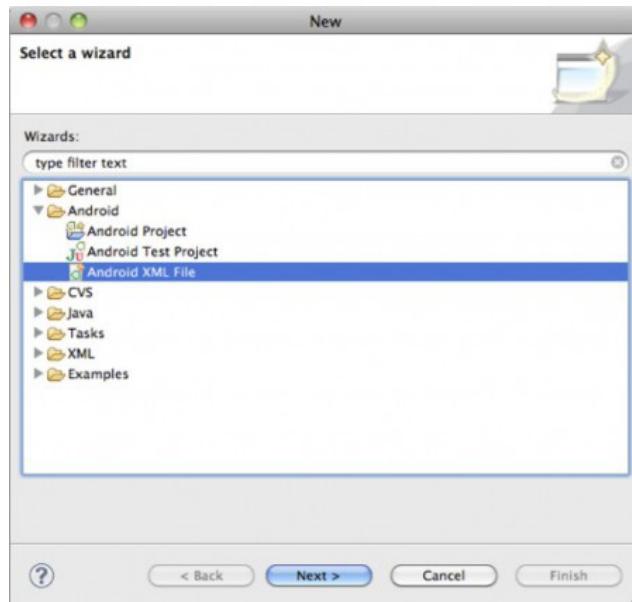


- Para el menú menu.xml
- Para la vista previa dentro de la aplicación showelement.xml
- Para cada fila del listado row.xml

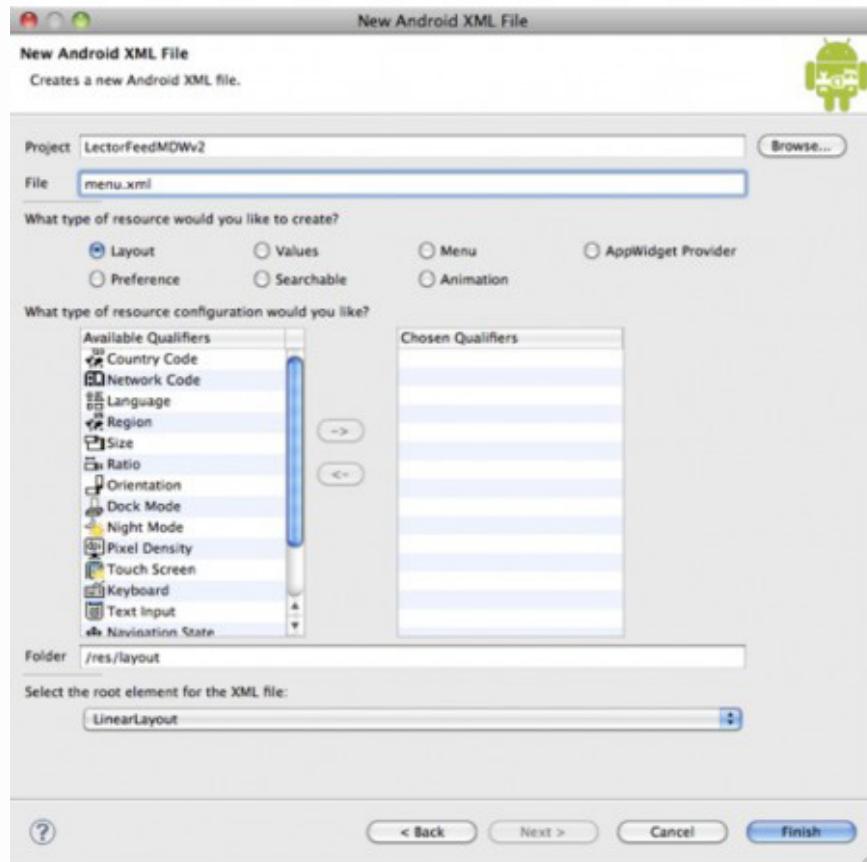
Para agregar estos Views, hacemos click derecho sobre la carpeta /res/layout luego New > (nuevo) y other... (otro):



En el diálogo que aparece seleccionamos archivo XML de Android y hacemos click en Next > (siguiente):



Escribimos el nombre del archivo (es importante incluir la extensión XML), en el ejemplo el nombre es menu.xml y hacemos click en el botón Finish (finalizar):



Este procedimiento es necesario cada vez que necesitemos un Layout nuevo, para esta aplicación serán necesarios 3: menu.xml, row.xml y showelement.xml describiremos a detalle cada uno de ellos.

## View para el menú: menu.xml

Utiliza un elemento menú y dos elementos ítem para cada opción, le pondremos a cada ítem texto y el ícono que trae por defecto la aplicación, el archivo XML debe quedar así:

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

El menú debería verse de la siguiente forma:

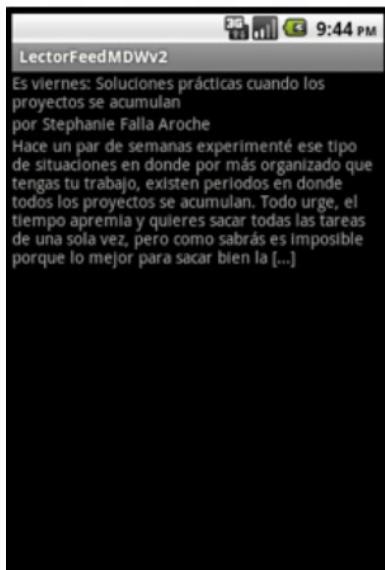


## View para preview: showelement.xml

En esta vista previa mostraremos el título del artículo, el autor y la descripción, necesitamos varios TextView dentro de un LinearLayout con orientación vertical, el archivo XML debe quedar así:

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

La vista previa debería verse de la siguiente forma:





## View para fila: row.xml

En este capítulo en cada una de las filas de la lista no mostraremos dos líneas como en el anterior, ahora vamos a tener una imagen del elemento y su título agrupados por un LinearLayout con orientación horizontal, el archivo XML debe quedar de la siguiente forma:

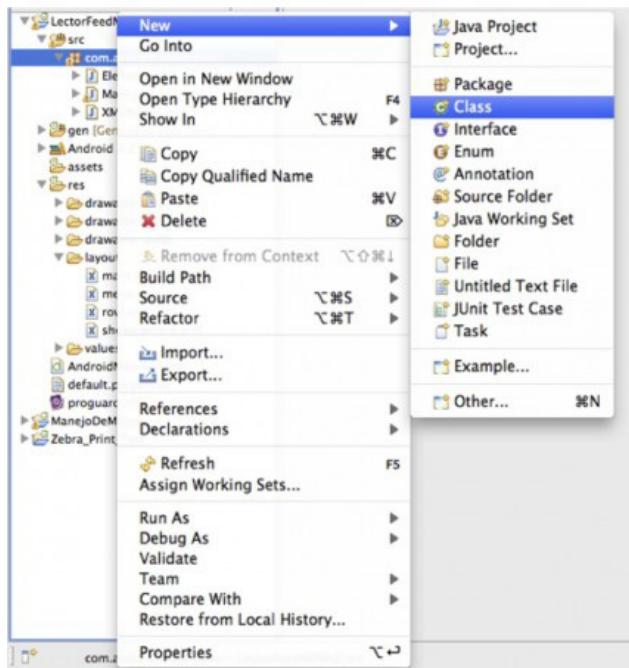
```
<!--?xml version="1.0" encoding="utf-8"?-->
```

## Clases de apoyo

En esta versión del lector de feeds tenemos varias clases que apoyan para facilitar el desarrollo:

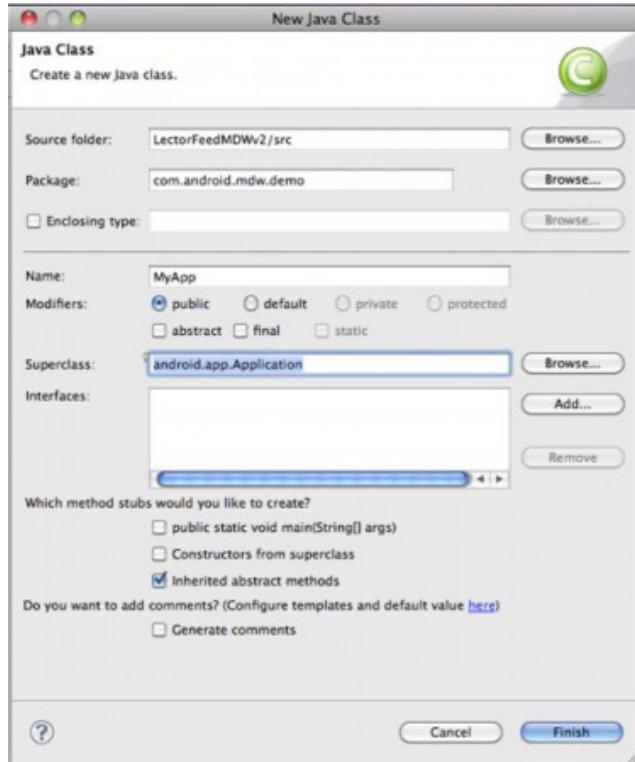
- ▶ **Element**: guarda los datos de cada artículo en el listado.
- ▶ **MyApp**: clase de aplicación para representar datos volátiles dentro de la aplicación.
- ▶ **MyAdapter**: adaptador personalizado que permite mostrar la imagen en cada fila del listado.

Para la vista previa del artículo necesitamos otra Activity que es otra clase que llamaremos ShowElement.java para agregar una clase es necesario hacer click derecho sobre com.android.mdw.demo (bajo src) luego New (nuevo) y por último Class (clase).



Después colocamos el nombre “MyApp” para el ejemplo y la superclase (clase de la que hereda) android.

app.Application para el ejemplo y hacemos click en finalizar.



El proceso es necesario para MyApp, MyAdapter y ShowElement la clase Element va incluida, ahora veamos el contenido de cada una de estas clases:

## Element

Haremos un objeto Element, para aumentar el orden porque ahora guardaremos más información de cada elemento del listado obtenido del feed. Esta clase está incluida en los archivos de prueba y no es muy complicada, maneja métodos para colocar y obtener los atributos de cada elemento como: Título, Enlace, Autor, Descripción, Fecha de publicación, Imagen.

SetImage es el método diferente y se debe a la forma en que obtenemos las imágenes. El parser XML buscará dentro del contenido del artículo la primera ocurrencia de la etiqueta img de HTML y en algunos casos la única es la foto del autor del artículo.

Tomamos prestada la imagen del avatar de Twitter de **@maestros<sup>1</sup>** el URL es:

[http://a1.twimg.com/profile\\_images/82885809/mdw\\_hr\\_reasonably\\_small.png](http://a1.twimg.com/profile_images/82885809/mdw_hr_reasonably_small.png)

La URL se recibe como parámetro en este método, si al tratar de obtener la imagen algo fallara entonces también se establece la imagen del avatar de Twitter. Para obtener la imagen en base a un URL se utiliza la ayuda de la función `loadFromUrl` que devuelve un **Bitmap<sup>2</sup>**.

Este método abre una conexión hacia el URL especificado, luego decodifica el flujo (stream) de bytes recibido y en base a ellos construye un objeto Bitmap. El código de la clase Element (ya incluido en el código base de la guía) es el siguiente:

Código fuente



```
package com.android.mdw.demo;
import java.io.IOException;
import java.io.InputStream;
import java.net.MalformedURLException;
import java.net.URL;
import java.text.SimpleDateFormat;
import java.util.Date;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
public class Element {
    static SimpleDateFormat FORMATTER = new SimpleDateFormat("EEE, dd MMM yyyy
HH:mm:ss Z");
    private String title;
    private String author;
    private String link;
    private Bitmap image;
    private String description;
    private Date date;
    public String getTitle() {
        return this.title;
```

1 <http://twitter.com/maestros>

2 <http://developer.android.com/reference/android/graphics/Bitmap.html>



```
}

public String getLink() {
    return this.link;
}

public String getDescription() {
    return this.description;
}

public String getDate() {
    return FORMATTER.format(this.date);
}

public Bitmap getImage() {
    return this.image;
}

public String getAuthor() {
    return this.author;
}

public void setTitle(String title) {
    this.title = title.trim();
}

public void setLink(String link) {
    this.link = link;
}

public void setDescription(String description) {
    this.description = description.trim();
}

public void setDate(String date) {
    try {
        this.date = FORMATTER.parse(date);
    } catch (java.text.ParseException e) {
        e.printStackTrace();
    }
}

public void setImage(String image) {
    if (image.contains("autor")) {
        image = "http://al.twimg.com/profile_images/82885809
```

```
/mdw_hr_reasonably_small.png";
}

try {
this.image = loadFromUrl(new URL(image));
} catch (Exception e) {
try {
this.image = loadFromUrl(new URL("http://a1.twimg.com
/profile_images/82885809/mdw_hr_reasonably_small.png"));
} catch (MalformedURLException e1) {}
}
}

public void setAuthor(String author) {
this.author = author;
}

public String toString() {
return this.title;
}

private Bitmap loadFromUrl(URL link) {
Bitmap bitmap = null;
InputStream in = null;
try {
in = link.openConnection().getInputStream();
bitmap = BitmapFactory.decodeStream(in, null, null);
in.close();
} catch (IOException e) {}
return bitmap;
}
}
```

## MyApp

En el capítulo anterior el listado de artículo los representamos como una lista estática dentro de la clase de la activity principal, mencioné que esa no era la forma correcta por que debían utilizarse clases de aplicación y para eso utilizaremos MyApp.



No es posible utilizar una variable local o una variable de instancia porque la **Activity**<sup>1</sup> es constantemente destruida y creada de nuevo. Por ejemplo, al rotar el teléfono. A pesar de estar representado dentro de una clase esta información no deja de ser volátil, para almacenamiento permanente es necesario una base de datos de SQLite.

Para decirle a la aplicación que es de tipo MyApp es necesario editar el manifest AndroidManifest.xml y en los atributos de la etiqueta aplicación agregar android:name="MyApp"

Inicialmente decía:

```
<application android:icon="@drawable/icon" android:label="@string/app_name">
```

Al modificarlo debe decir:

```
<application android:icon="@drawable/icon" android:label="@string/app_name"  
    android:name="MyApp">
```

Dentro de la clase MyApp vamos a guardar dos cosas:

- ▶ El listado de los artículos
- ▶ La opción seleccionada por el usuario para visualizar el artículo (ya sea en una vista previa dentro de la aplicación o en el navegador)
- ▶ Además de estas dos variables de instancia, vamos a incluir métodos para guardar y devolver estas variables (getters y setters). Para representar la opción elegida por el usuario utilizaremos enteros dentro de la Activity principal, estos están incluidos en el código base y fueron definidos así:

```
final static int APP_VIEW = 1;  
final static int BROWSER_VIEW = 2;
```

Al iniciar la aplicación, colocaremos el valor de APP\_VIEW en el campo que guarda la preferencia del usuario dentro de la clase de aplicación, el código de la clase MyApp queda de la siguiente forma:

Código fuente



```
package com.android.mdw.demo;  
import java.util.LinkedList;
```

<sup>1</sup> <http://developer.android.com/reference/android/app/Activity.html>

```
import android.app.Application;
public class MyApp extends Application {
private LinkedList data = null;
private int selectedOption = Main.APP_VIEW;
public LinkedList getData() {
return this.data;
}
public void setData(LinkedList d) {
this.data = d;
}
public int getSelectedOption() {
return this.selectedOption;
}
public void setSelectedOption(int selectedOption) {
this.selectedOption = selectedOption;
}
}
```

## MyAdapter

La clase MyAdapter será una personalización de un **ArrayAdapter**<sup>1</sup>, un adaptador que recibe un arreglo (o listado) de elementos. La clase MyAdapter hereda de ArrayAdapter<Element>, dentro de la clase manearemos dos variables de instancia, un **LayoutInflater**<sup>2</sup> esta clase se utilizar para instanciar el diseño a partir de un archivo de XML hacia un View y además un listado (LinkedList) de objetos (en este caso, artículos, representados por la clase Element).

Dentro del constructor, llamamos al padre (super) y asignamos a los campos (variables de instancia). Además, dado que queremos personalizar la representación (rendering) de cada fila, necesitamos sobrecargar revisar (**Override**<sup>3</sup>) el método getView.

Dentro de este método, cuando sea necesario instanciamos el archivo de su diseño correspondiente row.xml y le asignamos valor a sus Views (imagen y texto) a partir de la información guardada en el

1 <http://developer.android.com/reference/android/widget/ArrayAdapter.html>

2 <http://developer.android.com/reference/android/view/LayoutInflator.html>

3 <http://download.oracle.com/javase/tutorial/java/landl/override.html>

listado de artículos.

El código de la clase MyAdapter es el siguiente:

Código fuente



```
package com.android.mdw.demo;
import java.util.LinkedList;
import android.content.Context;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ArrayAdapter;
import android.widget.ImageView;
import android.widget.TextView;
public class MyAdapter extends ArrayAdapter {
    LayoutInflater inf;
    LinkedList objects;
    public MyAdapter(Context context, int resource,
    int textViewResourceId,
    LinkedList objects) {
        super(context, resource, textViewResourceId, objects);
        this.inf = LayoutInflater.from(context);
        this.objects = objects;
    }
    public View getView(int position, View convertView,
    ViewGroup parent) {
        View row = convertView;
        Element currentElement = (Element)objects.get(position);
        if (row == null) {
            row = inf.inflate(R.layout.row, null);
        }
        ImageView iv = (ImageView) row.findViewById(R.id.imgElement);
        iv.setImageBitmap(currentElement.getImage());
        iv.setScaleType(ImageView.ScaleType.FIT_XY);
        TextView tv = (TextView) row.findViewById(R.id.txtElement);
        tv.setText(currentElement.getText());
    }
}
```

```
        tv.setText(currentElement.getTitle());
        return row;
    }
}
```

## ShowElement

Por último tenemos la Activity que nos mostrará la vista previa del artículo seleccionado, cada vez que se agrega un Activity es necesario especificarlo en el manifest AndroidManifest.xml, bajo la etiqueta de aplicación (`<application>`) vamos a colocar:

```
<activity android:name=".ShowElement"></activity>"
```

Para pasar la información entre las Activities utilizaremos la información extra puede llevar un **intent<sup>1</sup>** previo a levantar una Activity. Disponemos de un diccionario que en nuestro caso utilizaremos para mandar un entero representando la posición del elemento que queremos visualizar.

Para empezar obtenemos el intent que levanto esta activity que debería traer la posición del elemento a visualizar, si en caso fuera nulo (el intent que levanto esta aplicación no manda nada) entonces se le asigna el valor por defecto (esperamos que la posición sea mayor o igual que 0, entonces para distinguir un error podemos asignar -1).

```
Intent it = getIntent();
int position = it.getIntExtra(Main.POSITION_KEY, -1);
```

Con la posición recibida, buscamos a través de la clase de aplicación MyApp y llamando al método `getData` obtenemos el artículo que nos interesa:

```
MyApp appState = ((MyApp) getApplication());
Element e = appState.getData().get(position);
```

Luego colocamos esos valores en los campos correspondientes definidos en su diseño `showelement.xml` si en caso no se recibiera una posición válida, se regresa a la Activity principal enviando la posición -1 para indicar que hubo un error:

```
Intent backToMainActivity = new Intent(this, Main.class);
backToMainActivity.putExtra(Main.POSITION_KEY, -1);
```

---

<sup>1</sup> <http://developer.android.com/reference/android/content/Intent.html#getIntExtra%28java.lang.String,%20int%29>

```
startActivity(backToMainActivity);
```

Al finalizar, el código de esta clase es el siguiente:

Código fuente



```
package com.android.mdw.demo;
import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.widget.TextView;
public class ShowElement extends Activity {
@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.showelement);
Intent it = getIntent();
int position = it.getIntExtra(Main.POSITION_KEY, -1);
if (position != -1) {
MyApp appState = ((MyApp) getApplication());
Element e = appState.getData().get(position);
TextView txtTitle = (TextView) findViewById(R.id.txtTitle);
txtTitle.setText(e.getTitle());
TextView txtAuthor = (TextView) findViewById(R.id.txtAuthor);
txtAuthor.setText("por " + e.getAuthor());
TextView txtDesc = (TextView) findViewById(R.id.txtDesc);
txtDesc.setText(e.getDescription());
} else {
Intent backToMainActivity = new Intent(this, Main.class);
backToMainActivity.putExtra(Main.POSITION_KEY, -1);
startActivity(backToMainActivity);
}
}
```

Esas son las clases adicionales luego el trabajo es sobre la Activity principal: la clase Main.java



## Mostrar datos siguiendo el diseño

La función auxiliar setData de la guía anterior se ve drásticamente reducida ya que ahora utilizamos nuestro adaptador y por tener una activity que hereda de ListActivity la asignación al ListView lo hacemos con setListAdapter.

```
private void setData() {
    this.setListAdapter(new MyAdapter(this, R.layout.row, 0,
        appState.getData()));
}
```

Esta herencia de nuestra Activity, también nos permite colocar la función onListItemClick para especificar la operación a realizar cuando el usuario presiona click sobre un elemento. Validamos la opción seleccionada por el usuario guardada en la clase de aplicación y dependiendo del caso levantamos un intent diferente.

Código fuente



```
protected void onListItemClick(ListView l, View v, int position, long id) {
    super.onListItemClick(l, v, position, id);
    Intent nextActivity = null;
    if (appState.getSelectedOption() == APP_VIEW) {
        nextActivity = new Intent(this, ShowElement.class);
        nextActivity.putExtra(POSITION_KEY, position);
    } else {
        LinkedList data = appState.getData();
        nextActivity = new Intent(Intent.ACTION_VIEW,
            Uri.parse(data.get(position).getLink()));
    }
    this.startActivity(nextActivity);
}
```

Permanece la variable de instancia para el diálogo de progreso y el manejador es ligeramente distinto porque ya no recibe los datos a través del mensaje.

Código fuente



```
private final Handler progressHandler = new Handler() {
```



```
public void handleMessage(Message msg) {  
    setData();  
    progressDialog.dismiss();  
}  
};
```

## Carga de datos

La carga de datos no cambia mucho, seguimos teniendo el diálogo de progreso pero ahora ya no necesitamos mandar los datos reconocidos por el parser a través del manejador si no puedo guardarlos en la clase de aplicación directamente y mando un mensaje vacío.

Código fuente



```
private void loadData() {  
    progressDialog = ProgressDialog.show(  
        Main.this,  
        "",  
        "Por favor espere mientras se cargan los datos...",  
        true);  
    new Thread(new Runnable() {  
        @Override  
        public void run() {  
            XMLParser parser = new XMLParser(feedUrl);  
            appState.setData(parser.parse());  
            progressHandler.sendEmptyMessage(0);  
        }  
    }).start();  
}
```

## Agregando el menú de opciones

Necesitamos indicarle qué hacer cuando el usuario presione la tecla de menú en el teléfono, como en este caso construimos el menú en un XML solo es necesario crear una instancia.

Código fuente



```
@Override
```

```
public boolean onCreateOptionsMenu(Menu menu) {  
    MenuInflater inflater = getMenuInflater();  
    inflater.inflate(R.layout.menu, menu);  
    return true;  
}
```

Además, requerimos sobrecargar otra función que se dispara cuando el usuario elige alguna de las opciones del menú. Aquí guardaremos en la clase de aplicación lo que sea que el usuario haya elegido.

  
Código fuente

```
@Override  
public boolean onOptionsItemSelected(MenuItem item) {  
    switch (item.getItemId()) {  
        case R.id.mmElementApp:  
            AppState.setSelectedOption(APP_VIEW);  
            break;  
        case R.id.mmElementBrw:  
            AppState.setSelectedOption(BROWSER_VIEW);  
            break;  
    }  
    return true;  
}
```

## Llamando todo desde la función onCreate

Dentro del cuerpo de la función onCreate inicializamos la variable para nuestra clase de aplicación:

```
AppState = ((MyApp) getApplication());
```

Validamos si el intent lo levantó alguna otra Activity y si viene un -1 en el mensaje mostramos un error:

```
Intent it = getIntent();  
int fromShowElement = it.getIntExtra(POSITION_KEY, 0);  
if (fromShowElement == -1) {  
    Toast.makeText(this, "Error, imposible visualizar el elemento",  
    Toast.LENGTH_LONG);  
}
```



La acción sobre el click del botón no cambia mucho, la verificación la hemos cambiado y en lugar de ver si el adaptar ya tiene datos revisamos si la clase de aplicación devuelve algo diferente de null:

```
LinkedList data = appState.getData();  
if (data != null) { ... }
```



Pregunta en el Foro Android  
de Foros del Web

<http://www.forosdelweb.com/f165/>

Descarga el código  
completo en

```
AlertDialog.Builder builder = new  
AlertDialog.Builder(Main.this);<br />  
builder.setMessage("ya ha cargado datos. ¿Está seguro de  
hacerlo de nuevo?");<br />  
builder.setCancelable(false);<br />  
builder.setPositiveButton("Sí",  
DialogInterface.OnClickListener() {<br />  
public void onClick(DialogInterface dialog, int id) {<br />
```



<https://github.com/androidMDW/guia2completo>



## Conclusión

- **ListActivity:** aprendimos la utilización de este tipo de actividad que nos da algunas ayudas cuando nuestra actividad muestra una lista de datos.
- **ImageView, Menu y MenuItem:** en la parte de diseño, manejamos ImageView para mostrar imágenes, en el caso de nuestra aplicación, la imagen se obtiene de una dirección de internet y los bytes recibidos se asignan para mostrarla. También agregamos un menú con opciones MenuItem para elegir y colocar una configuración muy simple de nuestra aplicación.
- **Guardar datos a través de clases de aplicación:** como un medio de guardar el estado de la aplicación vimos la forma de utilizar una clase creada por nosotros y también los mecanismos para guardar y recuperar los datos guardados.
- **Sobrecargar clases para personalización (MyAdapter):** con el fin de personalizar la forma en que se muestran los datos, utilizamos como base una clase ya existente y modificamos lo necesario para que el rendering fuera a nuestro gusto.
- **Incluir Activities a nuestra aplicación:** la aplicación puede contener más de una activity, vimos lo necesario para agregar y configurar en el manifest nuevas activities.
- **Enviar datos a través de intents:** utilizamos intents no solo para levantar una nueva Activity si no también para enviarle datos de una Activity a otra.
- **Recibir datos a través de intents:** para lograr la comunicación entre mis activities no solo fue necesario enviar los datos si no en la activity llamada también obtenerlos para su utilización.

# 3

capítulo



## Trabajando con ímágenes (cámara y galería)



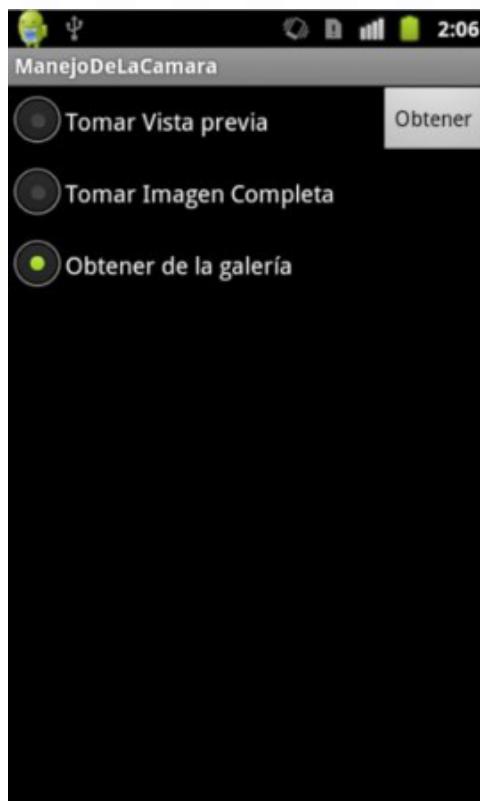
## CAPÍTULO 3

## Trabajando con imágenes (cámara y galería)

En el tercer capítulo trabajamos con el hardware de los teléfonos y empezaremos con el manejo de la cámara.

### Mostrar una imagen de la cámara o de la galería

La aplicación que realizaremos nos permitirá mostrar una imagen, podemos obtenerla desde la cámara o desde la galería del teléfono. Queremos que al finalizar se vea así:



Con una imagen cargada se verá de la siguiente forma:



## Disposición inicial

Empezaremos **descargando el código<sup>1</sup>** que debe importarse hacia un proyecto nuevo. Trabajamos sobre un teléfono con Android y cámara para desarrollar el ejemplo de este artículo. Para el deployment hacia el teléfono es necesario que el sistema operativo lo reconozca y además debe colocarse en el manifest como una propiedad de la etiqueta `<application>` el valor `android:debuggable="true"`.

La etiqueta debe lucir de la siguiente forma:

```
<application android:icon="@drawable/icon" android:label="@string/app_name"  
    android:debuggable="true">
```

En la parte de diseño vamos a empezar con un `RelativeLayout`, el archivo `/res/layout/main.xml` debe estar así:

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

<sup>1</sup> <https://github.com/androidMDW/guia3base>



## Diseño

Trabajamos con otra disposición de elementos para el diseño llamado RelativeLayout y agregaremos otros elementos de interfaz de usuario.

- ▶ **RelativeLayout<sup>1</sup>**: con este esquema los elementos se colocan en posición relativa a otros elementos o hacia el padre.
- ▶ **RadioButton<sup>2</sup>**: es un botón de dos estados (marcado y des-marcado) a diferencia del CheckButton este no permite des-marcarlo y cuando se encuentra en grupo solo uno de los botones del grupo puede estar marcado a la vez.
- ▶ **RadioGroup<sup>3</sup>**: permite agrupar un conjunto de RadioButtons para que solo uno a la vez esté seleccionado.

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.

El código completo del layout es el siguiente:

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

Hemos orientado el botón hacia la derecha y el RadioGroup hacia la izquierda. Luego el ImageView abajo del RadioGroup.

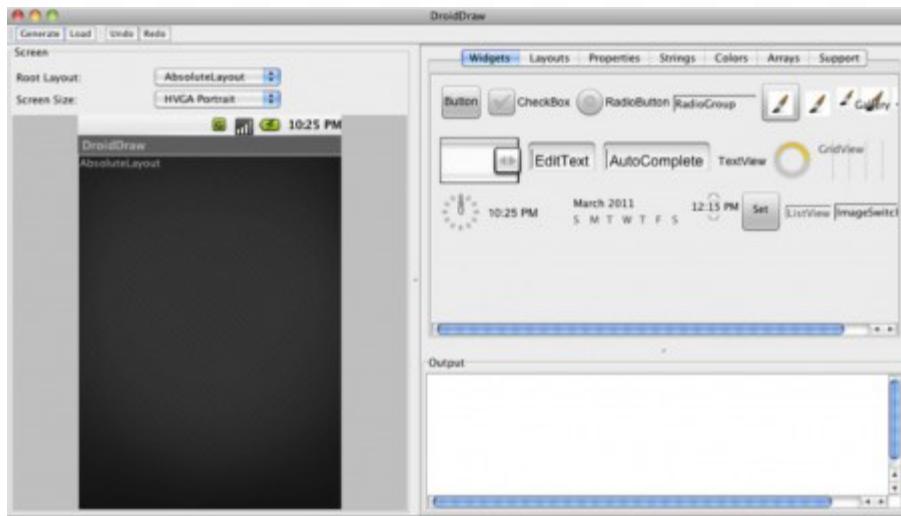
El diseño de interfaces de usuario en ocasiones se vuelve complicado con eclipse por ello utilizaremos la herramienta gratuita **DroidDraw<sup>4</sup>** que permite exportar e importar archivos XML para luego solo colocarlos en el archivo de diseño en eclipse, además tiene ejecutables para Windows, Linux y OS X.

1 <http://developer.android.com/reference/android/widget/RelativeLayout.html>

2 <http://developer.android.com/reference/android/widget/RadioButton.html>

3 <http://developer.android.com/reference/android/widget/RadioGroup.html>

4 <http://www.droiddraw.org/>



## Agregando código para funcionalidad

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 Store**<sup>1</sup> 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.

### Código fuente

```
name = Environment.getExternalStorageDirectory() + "/test.jpg";
Button btnAction = (Button)findViewById(R.id.btnPic);
```

<sup>1</sup> <http://developer.android.com/reference/android/provider/MediaStore.html>



```
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 accedemos la cámara con la acción `ACTION_IMAGE_CAPTURE`, si accedemos 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_CONTENT_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.

```
@Override 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.imageView);
iv.setImageBitmap(BitmapFactory.decodeFile(name));
```

Si quisieramos incluir esa imagen en nuestra galería, utilizamos un **MediaScannerConnectionClient**<sup>1</sup>.

  
Código fuente

```
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:

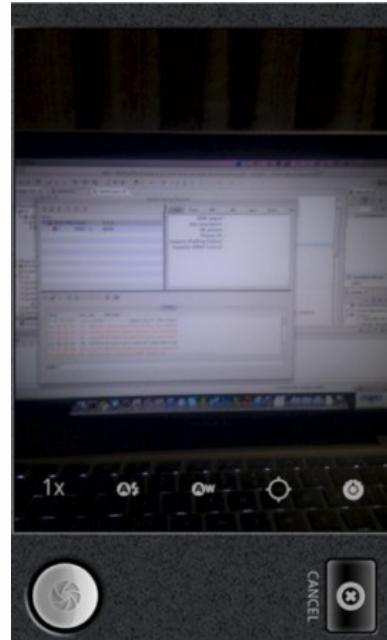
  
Código fuente

```
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.imageView);  
    iv.setImageBitmap(bitmap);  
} catch (FileNotFoundException e) {}
```

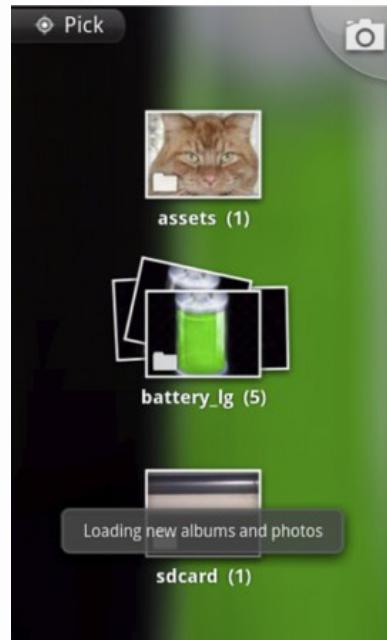
Si estamos utilizando la cámara la vista será de la siguiente forma:

---

<sup>1</sup> <http://developer.android.com/reference/android/media/MediaScannerConnection.MediaScannerConnectionClient.html>



Si estamos capturando de la galería será así:





¿Dudas?

Pregunta en el Foro Android  
de Foros del Web



<http://www.forosdelweb.com/f165/>

Descarga el código  
completo en

```
if (true) {  
    AlertDialog.Builder builder = new  
    AlertDialog.Builder(Main.this);<br />  
    builder.setMessage("ya ha cargado datos. ¿Está seguro de  
    hacerlo de nuevo?");<br />  
    builder.setCancelable(false);<br />  
    builder.setPositiveButton("Si",  
        new DialogInterface.OnClickListener() {<br />  
            public void onClick(DialogInterface dialog, int id) {<br />  
                //...  
            }  
        });  
    builder.show();  
}
```

**github**  
SOCIAL CODING

<https://github.com/androidMDW/guia3completo>



## Conclusión

- **RelativeLayout, RadioButton y RadioGroup:** aprendimos cómo pueden acomodarse componentes ubicándolos en el diseño con posiciones relativas con respecto a otros componentes y también los controles necesarios para el manejo de radio buttons y su agrupación.
- **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.
- **Utilización de la cámara de fotos:** a través de un ejemplo sencillo, aprendimos como utilizar parte del hardware del teléfono. A partir de aquí podemos trabajar las imágenes guardadas y modificarlas o utilizarla para lo que nos interese.

4

capítulo



# Grabación y reproducción de vídeo



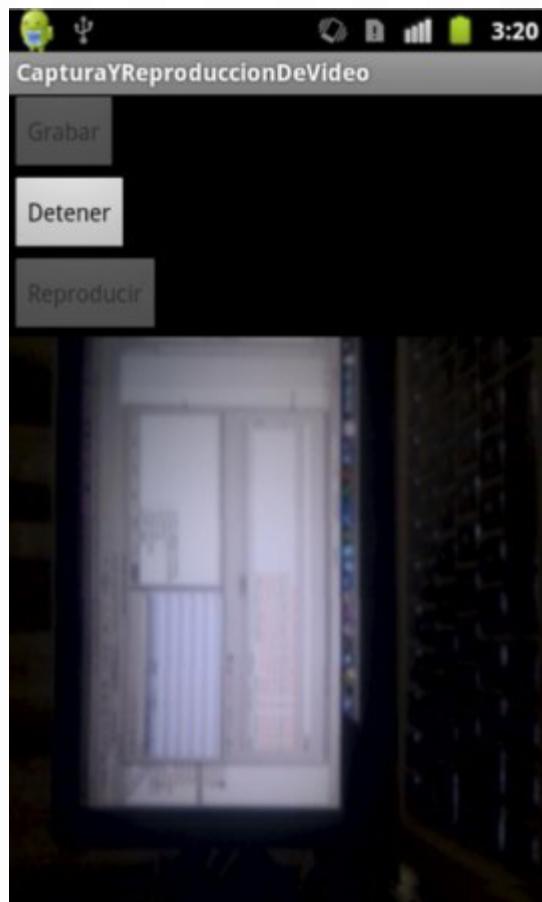
## CAPÍTULO 4

## Grabación y reproducción de vídeo

En el cuarto capítulo seguiremos trabajando con el hardware de los teléfonos, ahora aprenderemos sobre cómo grabar vídeo desde la cámara y reproducirlo.

### Capturar vídeo y reproducirlo

La aplicación que realizaremos nos permitirá grabar un vídeo, tener una vista previa mientras se graba y luego reproducirlo. Queremos que al finalizar se vea así:



## Disposición inicial

Iniciamos **descargando el código<sup>1</sup>** que debe ser importado hacia un nuevo proyecto, la configuración inicial para este ejemplo tiene algunas características importantes para revisar:

- ▶ **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:

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

- ▶ En el archivo principal de diseño res/layout/main.xml incluimos un LinearLayout.
- ▶ 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:

### Código fuente



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

@Override
public void surfaceCreated(SurfaceHolder arg0) {
}

@Override
public void surfaceDestroyed(SurfaceHolder arg0) {
}
```

---

<sup>1</sup> <https://github.com/androidMDW/guia4base>



## Diseño

Seguimos trabajando sobre un LinearLayout y agregamos un nuevo elemento:

- **SurfaceView**<sup>1</sup>: 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>
```

## Agregando código para funcionalidad

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.

```
@Override
```

---

<sup>1</sup> <http://developer.android.com/reference/android/view/SurfaceView.html>



```
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:

Código fuente



```
@Override  
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.

Código fuente



```
public void prepareRecorder(){  
    mediaRecorder.set AudioSource(MediaRecorder.AudioSource.MIC);  
    mediaRecorder.set VideoSource(MediaRecorder.VideoSource.CAMERA);  
    mediaRecorder.set OutputFormat(MediaRecorder.OutputFormat.MPEG_4);  
    mediaRecorder.set AudioEncoder(MediaRecorder.AudioEncoder.DEFAULT);  
    mediaRecorder.set VideoEncoder(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.

Código fuente



```
btnRec.setOnClickListener(new OnClickListener() {
    @Override
    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.

## Código fuente



```
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();  
    }  
});
```



Pregunta en el Foro Android  
de Foros del Web



<http://www.forosdelweb.com/f165/>

Descarga el código  
completo en

```
if (true) {  
    AlertDialog.Builder builder = new  
    AlertDialog.Builder(Main.this);  
    builder.setMessage("ya ha cargado datos. ¿Está seguro de  
    hacerlo de nuevo?");  
    builder.setCancelable(false);  
    builder.setPositiveButton("SI",  
        DialogInterface.OnClickListener(){  
            public void onClick(DialogInterface dialog, int id){  
                //...  
            }  
        });  
    builder.show();  
}
```

github  
SOCIAL CODING



<https://github.com/androidMDW/guia4completo>



## Conclusión

- **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.

# 5

capítulo



# Geolocalización y utilización de mapas de Google



## CAPÍTULO 5

## Geolocalización y utilización de mapas de Google

En el quinto capítulo seguiremos trabajando con el hardware de los teléfonos, ahora nos corresponde aprovechar el GPS y mostrar la ubicación actual en un mapa.

La aplicación que realizaremos nos permitirá mostrar la ubicación y actualizarse cada vez que exista movimiento. Es importante notar que las pruebas de este capítulo al igual que los capítulos 4 y 5 son para realizarlas en un teléfono con Android es posible utilizar el emulador con algunos arreglos.

**Ejemplo:** obtener localización, monitorear sus cambios y mostrarla en un mapa a través de markers. Queremos que al finalizar se vea de la siguiente forma:



# Disposición inicial

Iniciamos **descargando el código<sup>1</sup>** que debe ser importado hacia un nuevo proyecto y que tiene algunas características importantes.

Permisos en el Manifest, por el acceso a internet para el mapa y la ubicación del GPS requerimos estos 2 permisos bajo el tag:

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

Uso de la librería de mapas, también en el Manifest bajo el tag:

```
<uses-library android:name="com.google.android.maps" />
```

En la clase de la actividad principal, utilizaremos herencia de MapActivity para facilitarnos el manejo del mapa e implementaremos la interfaz LocationListener para detectar los cambios en la localización.

Por heredar de MapActivity debemos implementar el método:

```
@Override
protected boolean isRouteDisplayed() {
    return false;
}
```

Por implementar LocationListener debemos realizar los siguientes métodos:

Código fuente



```
@Override
public void onLocationChanged(Location location) {}

@Override
public void onProviderDisabled(String provider) {}

@Override
public void onStatusChanged(String provider, int status, Bundle extras) {}

@Override
public void onProviderEnabled(String provider) {}
```

---

<sup>1</sup> <https://github.com/androidMDW/guia5base>

## Diseño

Utilizaremos una vista especial para el mapa es necesario un **key para la utilización del servicio de Google Maps<sup>1</sup>** y depende del certificado utilizado para firmar las aplicaciones, el valor del atributo android:apiKey que aparece en el código es válida para mi certificado.

Código fuente



```
<com.google.android.maps.MapView  
    xmlns:android="http://schemas.android.com/apk/res/android"  
    android:id="@+id/mapview"  
    android:layout_width="fill_parent"  
    android:layout_height="fill_parent"  
    android:clickable="true"  
    android:apiKey="0xJ4uwRGDV296srCn4Iiy46oFmd1jecLr07BsAA"  
    />
```

## Agregando código

La clase de la Activity para nuestra aplicación extenderá de MapActivity para facilitarnos el manejo de la visualización del mapa e implementará LocationListener para el manejo de las actualizaciones de ubicación.

```
public class Main extends MapActivity implements LocationListener
```

De los métodos disponibles, utilizaremos onProviderDisabled para forzar al usuario a tener el GPS funcionando antes de que la aplicación inicie.

```
@Override  
public void onProviderDisabled(String provider) {  
    Intent intent = new Intent(  
        android.provider.Settings.ACTION_LOCATION_SOURCE_SETTINGS);  
    startActivity(intent);  
}
```

Vamos a mostrar un marker en las diferentes ubicaciones detectadas, para ello haremos una clase que

---

<sup>1</sup> <http://code.google.com/android/add-ons/google-apis/mapkey.html>

represente este Overlay mostrando la imagen seleccionada. Para este ejemplo se utilizará la imagen del ícono, al definirla dentro de la misma clase de la Activity principal entonces será una clase privada.

```
class MyOverlay extends Overlay { }
```

Tendremos una variable de instancia representando el punto donde se colocará el marcador y será un parámetro recibido por el constructor:

```
GeoPoint point;  
/* El constructor recibe el punto donde se dibujará el marker */  
public MyOverlay(GeoPoint point) {  
super();  
this.point = point;  
}
```

Sobrecargaremos el método draw para dibujar nuestro marker:

Código fuente



```
@Override  
public boolean draw(Canvas canvas, MapView mapView, boolean shadow, long when)  
{  
super.draw(canvas, mapView, shadow);  
//se traduce el punto geo localizado a un punto en la pantalla  
Point scrnPoint = new Point();  
mapView.getProjection().toPixels(this.point, scrnPoint);  
//se construye un bitmap a partir de la imagen  
Bitmap marker = BitmapFactory.decodeResource(getResources(), R.drawable.icon);  
//se dibuja la imagen del marker  
canvas.drawBitmap(marker, scrnPoint.x - image.getWidth() / 2, scrnPoint.y -  
marker.getHeight() / 2, null);  
return true;  
}
```

El código completo de la clase para los overlays es el siguiente:

Código fuente



```
class MyOverlay extends Overlay { }
```

```
GeoPoint point;
//El constructor recibe el punto donde se dibujará el marker
public MyOverlay(GeoPoint point) {
super();
this.point = point;
}
@Override
public boolean draw(Canvas canvas, MapView mapView, boolean shadow, long when)
{
super.draw(canvas, mapView, shadow);
//se traduce el punto geolocalizado a un punto en la pantalla */
Point scrnPoint = new Point();
mapView.getProjection().toPixels(this.point, scrnPoint);
//se construye un bitmap a partir de la imagen
Bitmap marker = BitmapFactory.decodeResource(getResources(),
R.drawable.icon);
// se dibuja la imagen del marker
canvas.drawBitmap(marker, scrnPoint.x - marker.getWidth() / 2,
scrnPoint.y - marker.getHeight() / 2, null);
return true;
}
}
```

Para ir dibujando estos markers al principio y cada vez que haya una actualización de la ubicación, utilizaremos un método nuevo con la firma:

```
protected void updateLocation(Location location)
```

Obtenemos el MapView y de él un MapController:

```
MapView mapView = (MapView) findViewById(R.id.mapview);
MapController mapController = mapView.getController();
```

Construimos un punto a partir de la latitud y longitud del Location recibido:

```
GeoPoint point = new GeoPoint((int) (location.getLatitude() * 1E6), (int)
(location.getLongitude() * 1E6));
```

Movemos el centro del mapa hacia esta ubicación:

```
mapController.animateTo(point);  
mapController.setZoom(15);
```

Posterior a esto, intentaremos hacer geolocalización del punto, obtener una dirección a partir de las coordenadas del punto. Primero instanciamos un objeto Geocoder que responderá de acuerdo a la localidad configurada en el teléfono.

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

A través de este Geocoder y con el punto de la ubicación intentamos obtener alguna dirección asociada y se lo hacemos saber al usuario por medio de un toast (aviso).

#### Código fuente



```
try {  
    List<Address> addresses = geoCoder.getFromLocation(point.getLatitudeE6() /  
        1E6, point.getLongitudeE6() / 1E6, 1);  
    String address = "";  
    if (addresses.size() > 0) {  
        for (int i = 0; i < addresses.get(0).getMaxAddressLineIndex(); i++)  
            address += addresses.get(0).getAddressLine(i) + "\n";  
    }  
    Toast.makeText(this, address, Toast.LENGTH_SHORT).show();  
} catch (IOException e) {  
    e.printStackTrace();  
}
```

Por último, instanciamos la clase MyOverlay construída previamente para agregar un nuevo marker en el listado de overlays del mapa e invalidamos la vista para que vuelva a dibujarse todo.

```
List<Overlay> mapOverlays = mapView.getOverlays();  
MyOverlay marker = new MyOverlay(point);  
mapOverlays.add(marker);  
mapView.invalidate();
```

En el método onCreate vamos a instanciar un LocationManager para mostrar el primer marker y solicitar actualizaciones:

## Código fuente



```
MapView mapView = (MapView) findViewById(R.id.mapview);
//habilitamos el control de zoom
mapView.setBuiltInZoomControls(true);
LocationManager locationManager =
(LocationManager) getSystemService(Context.LOCATION_SERVICE);
updateLocation(locationManager.getLastKnownLocation(LocationManager.GPS_
PROVIDER));
/* se actualizará cada minuto y 50 metros de cambio en la localización
mientras más pequeños sean estos valores más frecuentes serán las
actualizaciones */
locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER, 6000, 50,
this);
```

Luego en el método onLocationChanged llamamos a nuestro método updateLocation para que dibuje un nuevo marker cada vez que se actualice la localización:

```
@Override
public void onLocationChanged(Location location) {
updateLocation(location);
}
```



¿Dudas?

Pregunta en el Foro Android  
de Foros del Web



Descarga el código  
completo en

```
git@github.com:androidMDW/guia5completo.git
```

github  
SOCIAL CODING

<http://www.forosdelweb.com/f165/>

<https://github.com/androidMDW/guia5completo>

## Conclusión

En este capítulo aprendimos sobre:

- Cómo manejar un mapa, la necesidad de una vista especial y solicitar un API Key.
- Cómo dibujar markers a través de overlays sobre el mapa, además personalizamos la imagen del marker.
- Cómo solicitar la ubicación actual y actualizaciones ante el cambio de ubicación.

6

capítulo



# Trabajar con el acelerómetro

## CAPÍTULO 6

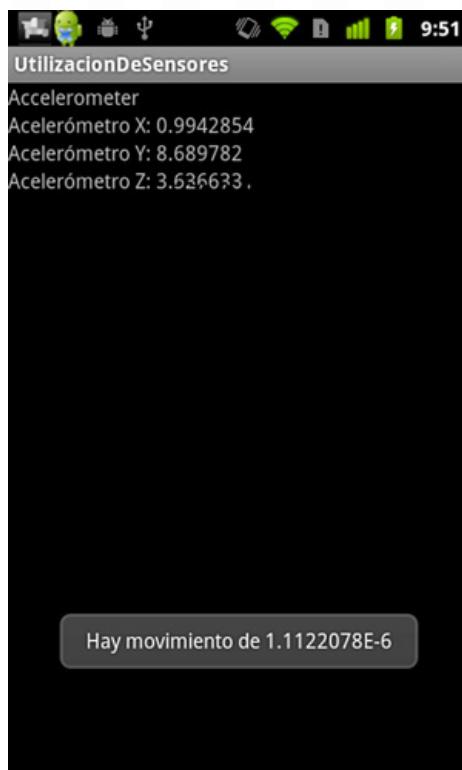
## Trabajar con el acelerómetro

En el sexto capítulo seguiremos trabajando con el hardware, ahora nos corresponde aprovechar el acelerómetro y usarlo para detectar movimiento del teléfono.

### Ejemplo: Utilizando el acelerómetro

La aplicación que realizaremos nos permitirá ver en pantalla los valores del acelerómetro y mostrar un aviso cuando se detecte cierto movimiento del teléfono.

Es importante notar que el teléfono cuenta con varios sensores (acelerómetro, orientación, campo magnético, luz, etc.) y de ellos vamos a utilizar únicamente uno el del acelerómetro. Un **acelerómetro**<sup>1</sup> es un sensor que mide la aceleración relativa a la caída libre como marco de referencia.



<sup>1</sup> <http://es.wikipedia.org/wiki/Aceler%C3%B3metro>

En este caso, nuestra interfaz de usuario será muy sencilla por lo que no utilizaremos ningún código base.

## Diseño

Mostraremos los valores de X, Y y Z del acelerómetro, para ello necesitamos 3 etiquetas TextView.

Código fuente



```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Valor de X"
        android:id="@+id/txtAccX"
    />
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Valor de Y"
        android:id="@+id/txtAccY"
    />
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Valor de Z"
        android:id="@+id/txtAccZ"
    />
</LinearLayout>
```

## Agregando código

La clase de la Activity principal para nuestra aplicación implementará SensorEventListener para el manejo de las actualizaciones de los sensores, en nuestro caso específico nos interesa el acelerómetro.

```
public class Main extends Activity implements SensorEventListener
```

Utilizaremos algunas variables de instancia para el control de los valores de los 3 ejes que mide el acelerómetro (valor anterior y valor actual) y otras para guardar el timestamp de la última actualización y la última vez que se detectó movimiento.

```
private long last_update = 0, last_movement = 0;  
private float prevX = 0, prevY = 0, prevZ = 0;  
private float curX = 0, curY = 0, curZ = 0;
```

Al implementar esta interface debemos sobrecargar dos métodos:

```
@Override  
public void onAccuracyChanged(Sensor sensor, int accuracy) {}  
  
@Override  
public void onSensorChanged(SensorEvent event) {}
```

Vamos a trabajar sobre onSensorChanged nuestro código será un synchronized statement (más información en la [documentación de Java](#)<sup>1</sup>) para evitar problemas de concurrencia al estar trabajando con los sensores.

```
synchronized (this) {  
}
```

A partir del evento recibido como parámetro vamos a obtener el timestamp de la fecha/hora actual (se usará más adelante) y los valores para los 3 ejes del acelerómetro (X, Y, Z).

```
long current_time = event.timestamp;  
curX = event.values[0];  
curY = event.values[1];  
curZ = event.values[2];
```

Luego, revisamos si este código ya se ha ejecutado alguna vez, es decir, si las variables que guardan los

---

<sup>1</sup> <http://download.oracle.com/javase/tutorial/essential/concurrency/locksync.html>

valores anteriores de X, Y y Z tiene valores diferentes de cero. Esto debería ejecutarse sólo la primera vez.

```
if (prevX == 0 && prevY == 0 && prevZ == 0) {  
    last_update = current_time;  
    last_movement = current_time;  
    prevX = curX;  
    prevY = curY;  
    prevZ = curZ;  
}
```

Obtenemos la diferencia entre la última actualización y el timestamp actual, esto nos servirá no solo para ver que si existe una diferencia de tiempos en las mediciones si no también para calcular el movimiento. Para ello, tomamos la posición actual (con las 3 coordenadas) y la restamos a la posición anterior, puede ser que el movimiento sea en distintas direcciones por eso nos es útil el valor absoluto.

```
long time_difference = current_time - last_update;  
if (time_difference > 0) {  
    float movement = Math.abs((curX + curY + curZ) - (prevX + prevY + prevZ)) /  
        time_difference;  
    ...  
}
```

Para decidir en que momento mostramos un aviso Toast indicando el movimiento vamos a usar como valor de frontera de movimiento mínimo  $1 \times 10^{-6}$ ; este valor es arbitrario mientras mayor sea, se necesitará más movimiento y mientras menor sea más sensible será el display del aviso.

Manejamos también 2 variables para el control de tiempo, una para saber la última actualización (last\_update) y otra para conocer la última vez que se registró movimiento last\_movement y en esta parte del código actualizamos su valor según sea conveniente.

#### Código fuente

```
int limit = 1500;  
float min_movement = 1E-6f;  
if (movement > min_movement) {  
    if (current_time - last_movement >= limit) {  
        Toast.makeText(getApplicationContext(), "Hay movimiento de " + movement,
```

```
Toast.LENGTH_SHORT).show();  
}  
last_movement = current_time;  
}
```

Por último actualizamos los valores de X, Y y Z anteriores para la próxima vez que se registre cambio en los sensores.

```
prevX = curX;  
prevY = curY;  
prevZ = curZ;  
last_update = current_time;
```

También actualizamos los valores de los 3 ejes del acelerómetro en las etiquetas para visualizarlo en pantalla.

```
((TextView) findViewById(R.id.txtAccX)).setText("Acelerómetro X: " + curX);  
((TextView) findViewById(R.id.txtAccY)).setText("Acelerómetro Y: " + curY);  
((TextView) findViewById(R.id.txtAccZ)).setText("Acelerómetro Z: " + curZ);
```

El código completo queda de la siguiente forma:

#### Código fuente



```
@Override  
public void onSensorChanged(SensorEvent event) {  
    synchronized (this) {  
        long current_time = event.timestamp;  
        curX = event.values[0];  
        curY = event.values[1];  
        curZ = event.values[2];  
        if (prevX == 0 && prevY == 0 && prevZ == 0) {  
            last_update = current_time;  
            last_movement = current_time;  
            prevX = curX;  
            prevY = curY;  
            prevZ = curZ;  
        }  
    }  
}
```

```
long time_difference = current_time - last_update;
if (time_difference > 0) {
    float movement = Math.abs((curX + curY + curZ) - (prevX - prevY - prevZ)) / time_difference;
    int limit = 1500;
    float min_movement = 1E-6f;
    if (movement > min_movement) {
        if (current_time - last_movement >= limit) {
            Toast.makeText(getApplicationContext(), "Hay movimiento de " +
                + movement, Toast.LENGTH_SHORT).show();
        }
        last_movement = current_time;
    }
    prevX = curX;
    prevY = curY;
    prevZ = curZ;
    last_update = current_time;
}
((TextView) findViewById(R.id.txtAccX)).setText("Acelerómetro X: " +
    curX);
((TextView) findViewById(R.id.txtAccY)).setText("Acelerómetro Y: " +
    curY);
((TextView) findViewById(R.id.txtAccZ)).setText("Acelerómetro Z: " +
    curZ);
}
```

Una vez listo esto, es necesario registrar y anular el registro del Listener para el sensor según corresponda, esto lo haremos en los métodos `onResume` y `onStop` de la actividad (más información del ciclo de vida de las Activities en la [documentación oficial<sup>1</sup>](#) ).

Para el registro, obtenemos primero el servicio del sistema de sensores para asignarlo en un SensorManager y a partir de él obtenemos el acceso al acelerómetro. Al realizar el registro del aceleróme-

---

<sup>1</sup> <http://developer.android.com/reference/android/app/Activity.html>

tro es necesario indicar una tasa de lectura de datos, en nuestro caso vamos a utilizar SensorManager.SENSOR\_DELAY\_GAME que es la velocidad mínima para que el acelerómetro pueda usarse en un juego.



### Código fuente

```
@Override  
protected void onResume() {  
    super.onResume();  
    SensorManager sm = (SensorManager) getSystemService(SENSOR_SERVICE);  
    List<Sensor> sensors = sm.getSensorList(Sensor.TYPE_ACCELEROMETER);  
    if (sensors.size() > 0) {  
        sm.registerListener(this, sensors.get(0),  
SensorManagerSENSOR_DELAY_GAME);  
    }  
}
```

Para anular el registro únicamente es necesario indicar que la clase actual (que implementa a SensorEventListener) ya no está interesada en recibir actualizaciones de sensores.

```
@Override  
protected void onStop() {  
    SensorManager sm = (SensorManager) getSystemService(SENSOR_SERVICE);  
    sm.unregisterListener(this);  
    super.onStop();  
}
```

Para terminar, en el método onCreate de la Activity vamos a bloquear la orientación para que al mover el teléfono la pantalla no se mueva y la interfaz permanezca tal como la vemos.

```
@Override  
public void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.main);  
    this.setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);  
}
```



¿Dudas?

Pregunta en el Foro Android  
de Foros del Web



<http://www.forosdelweb.com/f165/>

Descarga el código  
completo en

```
if (true) {  
    AlertDialog.Builder builder = new  
    AlertDialog.Builder(Main.this);<br />  
    builder.setMessage("ya ha cargado datos. ¿Está seguro de  
    hacerlo de nuevo?");<br />  
    builder.setCancelable(false);<br />  
    builder.setPositiveButton("Si",  
        DialogInterface.OnClickListener() {<br />  
            public void onClick(DialogInterface dialog, int id) {<br />  
                //...  
            }  
        });  
    builder.show();  
}
```

**github**  
SOCIAL CODING

<https://github.com/androidMDW/guia6completo>

## Conclusión

En esta entrega del curso Android hemos aprendido a trabajar con el acelerómetro, uno de los varios sensores incluídos en los teléfonos, a través de un Listener, nuestro código de respuesta al evento se ejecuta cuando hay algún cambio en los sensores y detectamos cuando hay cierto movimiento.

7

capítulo



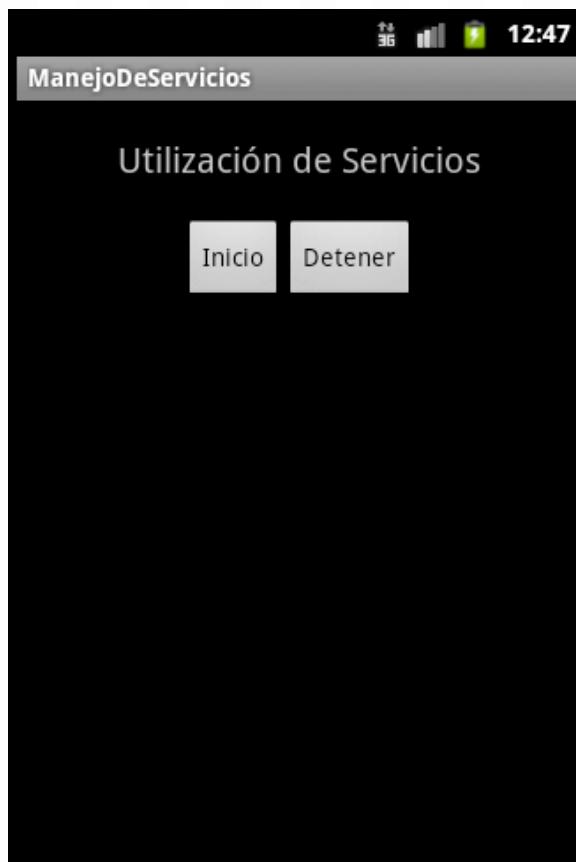
# Reproducción de sonido en un ciclo infinito



## CAPÍTULO 7

## Reproducción de sonido en un ciclo infinito

En el séptimo seguiremos trabajando con el hardware veremos cómo funcionan los servicios en Android. La aplicación que realizaremos nos permitirá iniciar y detener un servicio, como ejemplo este servicio se encargará de reproducir un sonido en un ciclo infinito.



## Disposición inicial

En este caso, nuestra interfaz de usuario será muy sencilla por lo que no utilizaremos ningún código base. Para configurar el servicio es necesario modificar el Manifest y agregar: `<service android:enabled="true" android:name=".ElServicio" />`



## Diseño

El diseño presenta dos botones, uno para detener y otro para iniciar el servicio. Se sugiere utilizar layouts, en nuestro caso el código completo del layout es:

Código fuente



```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"><TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Utilización de Servicios"
        android:gravity="center"
        android:textSize="20sp"
        android:padding="20dp"/>
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:gravity="center">
    <Button
        android:text="Inicio"
        android:id="@+id/btnInicio"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content">
    </Button>
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Detener"
        android:id="@+id/btnFin">
    </Button>
</LinearLayout>
</LinearLayout>
```

## Agregando código

Necesitamos de una clase adicional que represente al servicio, la llamaremos ElServicio y debe heredar de Service (en `android.app`). Esta clase debe sobrecargar el método onBind pero en este capítulo no nos interesa que haga algo así que únicamente devolverá null.

Para que la clase funcione de la manera deseada, vamos a sobrecargar tres métodos: `onCreate`, `onDestroy` y `onStart`, en cada uno de ellos a través de un toast vamos a verificar cuando el flujo pasa por estos puntos.

En `onCreate` vamos a inicializar lo que hará el servicio más adelante, en nuestro ejemplo utilizaremos el media player ya reproduciendo un sonido, como demo he descargado uno desde [soundjay](#)<sup>1</sup>. luego renombrado a `water_droplet` por el convenio de nombres que no incluye guiones y por último colocado en `/res/raw/`

En `onStart` vamos a iniciar la reproducción del sonido y en `onDestroy` vamos a detener el sonido que se estaba reproduciendo. En resumen, lo necesario para cada uno de estos métodos es:

- ▶ **onCreate**: Inicializar el Media Player e indicar que funcionará con un loop reproduciendo el sonido.
- ▶ **onDestroy**: Detener la reproducción del sonido
- ▶ **onStart**: Iniciar la reproducción del audio.

Es decir, el código de nuestra clase de servicios completo queda de la siguiente forma:

Código fuente



```
public class ElServicio extends Service {  
    private MediaPlayer player;  
    @Override  
    public IBinder onBind(Intent intent) {  
        // TODO Auto-generated method stub  
        return null;  
    }  
    @Override
```

<sup>1</sup> <http://www.soundjay.com/nature/sounds/water-droplet-1.mp3>

```
public void onCreate() {  
    Toast.makeText(this, "Servicio Creado", Toast.LENGTH_LONG).show();  
    player = MediaPlayer.create(this, R.raw.water_droplet);  
    player.setLooping(true);  
}  
  
@Override  
public void onDestroy() {  
    Toast.makeText(this, "Servicio Detenido", Toast.LENGTH_LONG).show();  
    player.stop();  
}  
  
@Override  
public void onStart(Intent intent, int startid) {  
    Toast.makeText(this, "Servicio Iniciado", Toast.LENGTH_LONG).show();  
    player.start();  
}  
}
```

Este código del servicio se ejecutará como una tarea en Background es decir podríamos cerrar la aplicación y la ejecución continuaría. Por lo mismo, adicional al servicio, es necesario construir código para iniciarla y detenerla.

Esto lo haremos en nuestra actividad principal asociando acciones a los botones previamente mencionados en el diseño. Dado que tenemos 2 botones que tendrán el mismo listener en lugar de instanciar un *OnClickListener* como una clase anónima nuestra Actividad implementará a *OnClickListener*.

```
public class Main extends Activity implements OnClickListener
```

Dentro del método *onCreate* de la actividad asociamos el listener a los botones.

```
Button btnInicio = (Button) findViewById(R.id.btnInicio);  
Button btnFin = (Button) findViewById(R.id.btnFin);  
btnInicio.setOnClickListener(this);  
btnFin.setOnClickListener(this);
```

Por último es necesario sobrecargar el método *onClick* y dentro de él distinguimos que botón fue presionado y realizamos la acción asociada en cada caso.

## Código fuente



```
public void onClick(View src) {  
    switch (src.getId()) {  
        case R.id.btnInicio:  
            startService(new Intent(this, ElServicio.class));  
            break;  
        case R.id.btnFin:  
            stopService(new Intent(this, ElServicio.class));  
            break;  
    }  
}
```



Pregunta en el Foro Android  
de Foros del Web



Descarga el código  
completo en

```
git clone https://github.com/androidMDW/guia7completo  
github SOCIAL CODING
```



<http://www.forosdelweb.com/f165/>

<https://github.com/androidMDW/guia7completo>



## Conclusión

En este capítulo hemos visto varias cosas sobre los servicios, estos son componentes que representan ciertas acciones que se ejecutarán durante un tiempo muy largo (o indefinido) por lo que funcionan en segundo plano (background). Para interactuar con los servicios es necesaria una actividad que lo inicie y detenga.

8

capítulo



# Envío de email utilizando Android

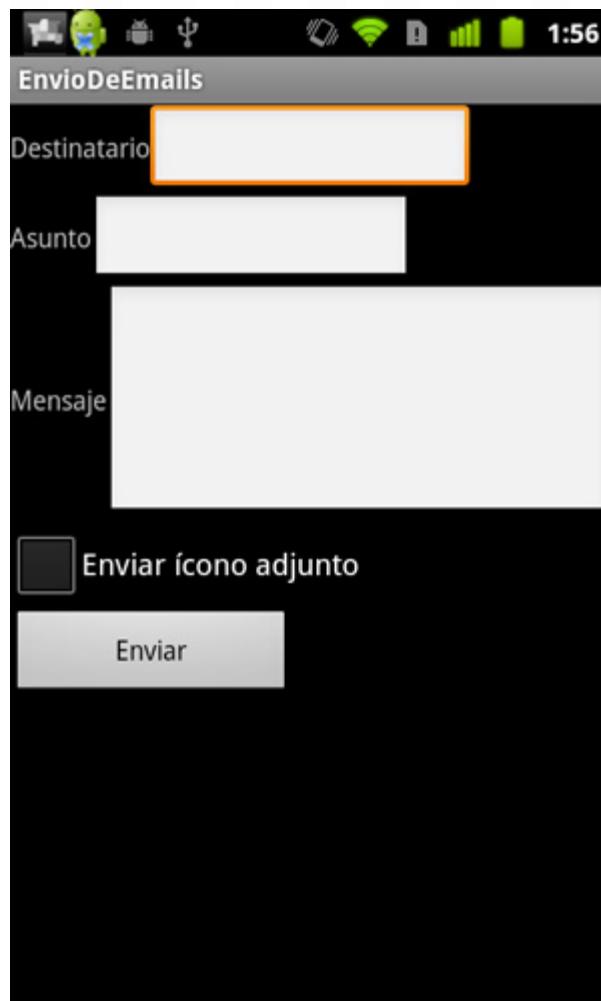


## CAPÍTULO 8

## Envío de email utilizando Android

En el octavo capítulo veremos cómo enviar emails utilizando Android la aplicación de ejemplo que realizaremos nos permitirá colocar la información para un correo electrónico (destinatario, asunto, contenido del correo, adjuntos) y luego enviarlo a través de la aplicación de correo electrónico.

Es importante notar que el ejemplo debe ejecutarse en un teléfono o bien instalar una aplicación para envío de emails en el emulador (por defecto no está instalada pero es relativamente simple). Queremos que al finalizar se vea así:





# Disposición inicial

En este caso, nuestra interfaz de usuario será muy sencilla por lo que no utilizaremos ningún código base tampoco realizaremos ninguna configuración adicional.

## Diseño

El diseño presenta los campos necesarios para el envío del mail dentro de un contenedor global (en este caso utilizamos un LinearLayout) por cada fila tenemos otro contenedor que a su vez dentro tiene 2 elementos: una TextView y un EditText eso se repite para la dirección de correo del destinatario, el asunto y el cuerpo del correo.

Por último, se tiene una CheckBox para indicar si se desea enviar un adjunto (el ícono de la aplicación). El XML completo del diseño es el siguiente:

Código fuente



```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout android:id="@+id/LinearLayout01"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical"
    xmlns:android="http://schemas.android.com/apk/res/android">
    <LinearLayout android:id="@+id/LinearLayout02"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="horizontal">
        <TextView android:text="Destinatario"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:id="@+id/txtForEmail"></TextView>
        <EditText android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:width="170dip"
            android:id="@+id/etEmail">
```



```
</EditText>
</LinearLayout>
<LinearLayout android:id="@+id/LinearLayout03"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="horizontal">
    <TextView android:text="Asunto"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/txtForSubject"></TextView>
    <EditText android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:width="170dip"
        android:id="@+id/etSubject">
    </EditText>
</LinearLayout>
<LinearLayout android:id="@+id/LinearLayout04"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="horizontal">
    <TextView android:text="Mensaje"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/txtForBody"></TextView>
    <EditText android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:lines="5"
        android:width="300dip"
        android:id="@+id/etBody">
    </EditText>
</LinearLayout>
<CheckBox android:text="Enviar ícono adjunto"
    android:id="@+id/chkAttachment"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"></CheckBox>
```



```
<Button android:layout_width="wrap_content"  
        android:layout_height="wrap_content"  
        android:id="@+id/btnSend"  
        android:text="Enviar"  
        android:width="150dip">  
</Button>  
</LinearLayout>
```

## Agregando funcionalidad

Asociaremos todo el código del envío del email al evento de click sobre el botón “Enviar”:

```
Button btnSend = (Button) findViewById(R.id.btnSend);  
btnSend.setOnClickListener(new OnClickListener() {  
    @Override  
    public void onClick(View v) {  
    }  
});
```

Obtenemos primero los elementos del form con los datos el email (destinatario, asunto, cuerpo del mail y adjunto):

```
// obtenemos los datos para el envío del correo  
EditText etEmail = (EditText) findViewById(R.id.etEmail);  
EditText etSubject = (EditText) findViewById(R.id.etSubject);  
EditText etBody = (EditText) findViewById(R.id.etBody);  
CheckBox chkAttachment = (CheckBox) findViewById(R.id.chkAttachment);
```

Luego construimos un intent que luego utilizaremos para levantar la Activity para el envío del correo, este debe ser del tipo ACTION\_SEND, posteriormente indicamos cuál será el tipo de dato a enviar.

```
// es necesario un intent que levante la actividad deseada  
Intent itSend = new Intent(android.content.Intent.ACTION_SEND);  
// vamos a enviar texto plano a menos que el checkbox esté marcado  
itSend.setType("plain/text");
```

Colocamos todos los datos obtenidos del form, incluyendo el posible adjunto en caso de que el CheckBox esté marcado.



## Código fuente



```
// colocamos los datos para el envío
itSend.putExtra(android.content.Intent.EXTRA_EMAIL, new String[]{etEmail.getText().toString()});
itSend.putExtra(android.content.Intent.EXTRA_SUBJECT, etSubject.getText().toString());
itSend.putExtra(android.content.Intent.EXTRA_TEXT, etBody.getText());
// revisamos si el checkbox está marcado enviamos el ícono de la aplicación como
// adjunto
if (chkAttachment.isChecked()) {
    // colocamos el adjunto en el stream
    itSend.putExtra(Intent.EXTRA_STREAM, Uri.parse("android.resource://" + getPackageName() + "/" + R.drawable.icon));
    // indicamos el tipo de dato
    itSend.setType("image/png");
}
```

Por último iniciamos la Activity para el envío.

```
startActivity(itSend);
```

El código completo es el siguiente:

## Código fuente



```
Button btnSend = (Button) findViewById(R.id.btnSend);
btnSend.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View v) {
        //obtenemos los datos para el envío del correo
        EditText etEmail = (EditText) findViewById(R.id.etEmail);
        EditText etSubject = (EditText) findViewById(R.id.etSubject);
        EditText etBody = (EditText) findViewById(R.id.etBody);
        CheckBox chkAttachment = (CheckBox) findViewById(R.id.chkAttachment);
        //es necesario un intent que levante la actividad deseada
        Intent itSend = new Intent(android.content.Intent.ACTION_SEND);
```



```
//vamos a enviar texto plano a menos que el checkbox esté marcado
itSend.setType("plain/text");
//colocamos los datos para el envío
itSend.putExtra(android.content.Intent.EXTRA_EMAIL, new String[]{etEmail.getText().toString()});
itSend.putExtra(android.content.Intent.EXTRA_SUBJECT,
etSubject.getText().toString());
itSend.putExtra(android.content.Intent.EXTRA_TEXT, etBody.getText());
//revisamos si el checkbox está marcado enviamos el ícono de la
aplicación como adjunto
if (chkAttachment.isChecked()) {
//colocamos el adjunto en el stream
itSend.putExtra(Intent.EXTRA_STREAM, Uri.parse("android.resource://" +
+ getPackageName() + "/" + R.drawable.icon));
//indicamos el tipo de dato
itSend.setType("image/png");
}
//iniciamos la actividad
startActivity(itSend);
}
});
```



Pregunta en el Foro Android  
de Foros del Web



<http://www.forosdelweb.com/f165/>

Descarga el código  
completo en

```
if (true) {  
    AlertDialog.Builder builder = new  
    AlertDialog.Builder(Main.this);<br />  
    builder.setMessage("ya ha cargado datos. ¿Está seguro de  
    hacerlo de nuevo?");<br />  
    builder.setCancelable(false);<br />  
    builder.setPositiveButton("Si",  
        new DialogInterface.OnClickListener() {<br />  
            public void onClick(DialogInterface dialog, int id) {<br />  
                //...  
            }  
        });  
    builder.setNegativeButton("No",  
        new DialogInterface.OnClickListener() {<br />  
            public void onClick(DialogInterface dialog, int id) {<br />  
                //...  
            }  
        });  
    builder.show();  
}
```

github  
SOCIAL CODING

<https://github.com/androidMDW/guia8completo>

## Conclusión

En esta ocasión hemos visto la forma de enviar un correo electrónico (incluyendo un adjunto opcional) todo incluído en los extras a través de un intent utilizado para iniciar la Activity para el envío del mail.

# 9

capítulo



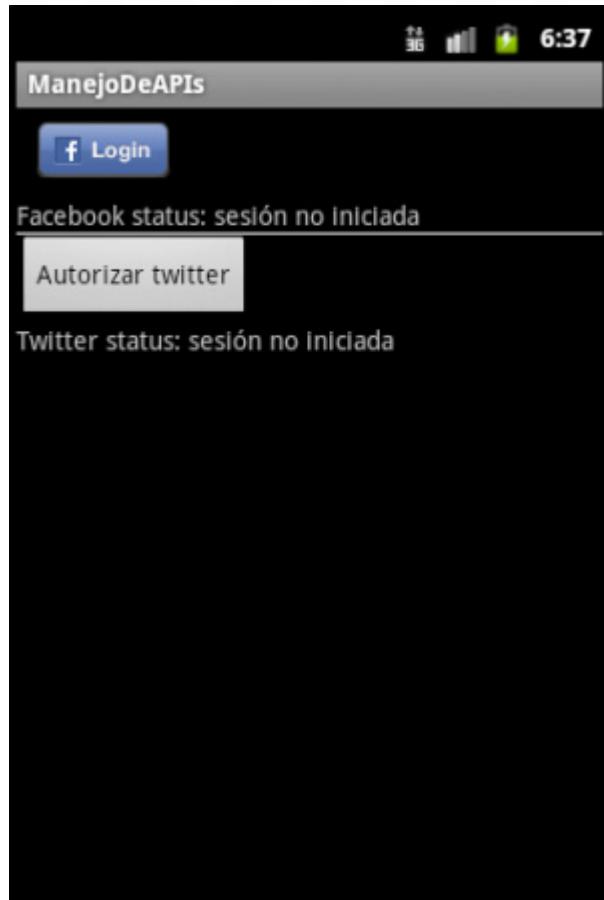
## Trabajando con APIs (Facebook y Twitter)

## CAPÍTULO 9

## Trabajando con APIs (Facebook y Twitter)

En el noveno capítulo utilizaremos los APIs de Facebook y Twitter para autenticación. La aplicación que realizaremos nos permitirá iniciar sesión y obtener algunos datos.

Queremos que al finalizar se vea así:



## Disposición inicial

Para realizar la autenticación es necesario **solicitar una aplicación en Facebook<sup>1</sup>** y **una aplicación en Twitter<sup>2</sup>**. En el código completo se incluyen el identificador de la aplicación de Facebook de demo del

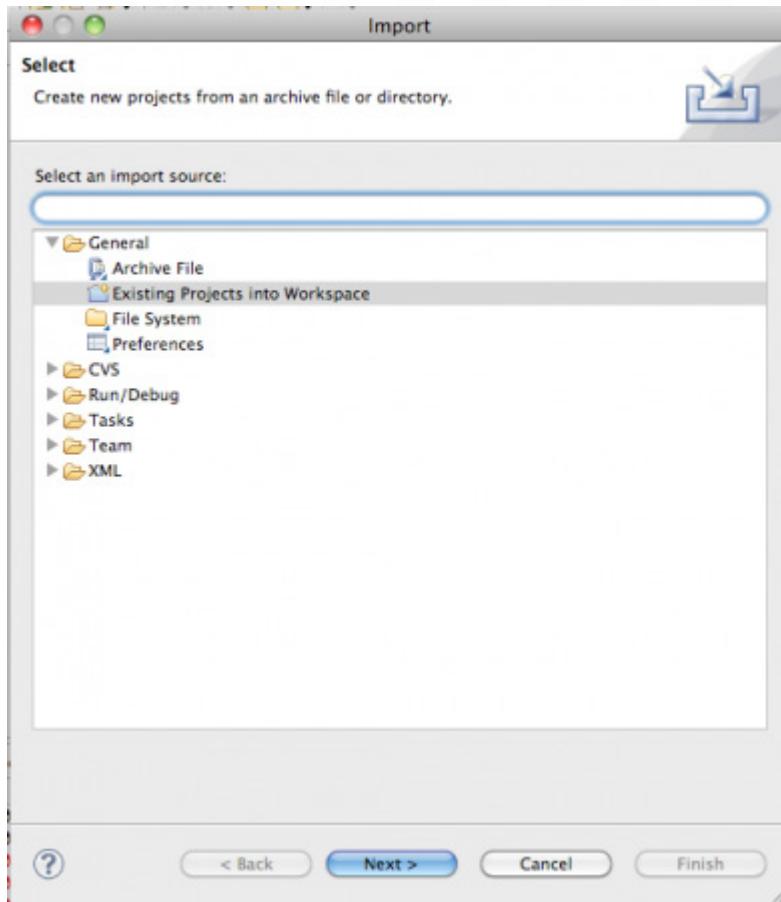
<sup>1</sup> <http://www.facebook.com/developers/createapp.php>

<sup>2</sup> <https://dev.twitter.com/apps/new>

SDK y un token/secret de consumer de una aplicación de Twitter que han sido expirados y si no los reemplazan por los suyos la aplicación no funcionará.

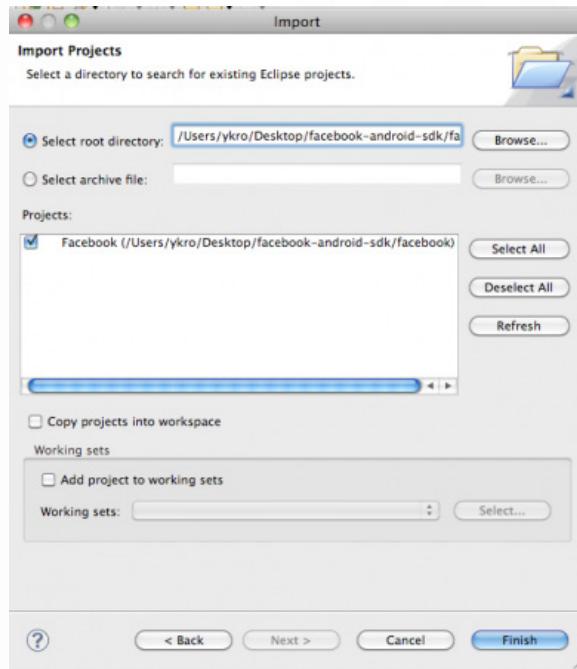
En esta ocasión no utilizaremos código base si no que realizaremos toda la configuración inicial paso a paso. Vamos a utilizar el SDK de Facebook para Android que pueden [descargar desde GitHub](#)<sup>1</sup>. Para nuestro ejemplo vamos a tomar no solo el SDK si no también algunos archivos del ejemplo simple que se incluye en el mismo SDK (El botón de inicio de sesión, el manejo y almacenamiento de la sesión y el listener).

Importamos el proyecto a nuestro workspace:



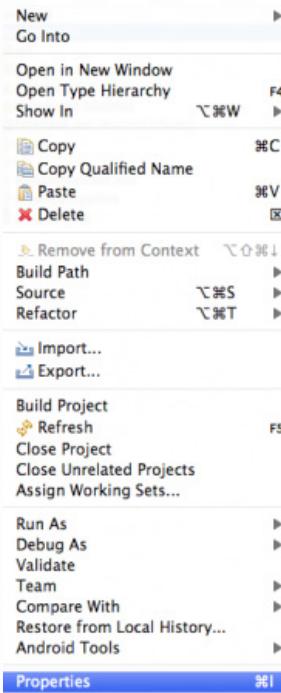
Seleccionando el directorio con los archivos descargados:

1 <https://github.com/facebook/facebook-android-sdk/>

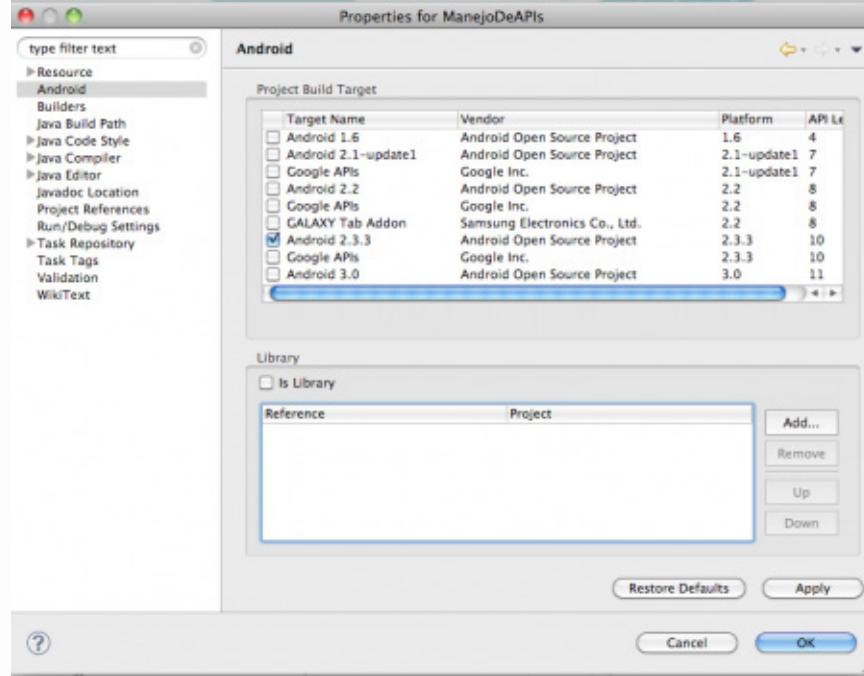


Hacemos un proyecto nuevo, como lo hemos visto en guías anteriores, en mi caso le llamaré **Manejo-DeAPIs**.

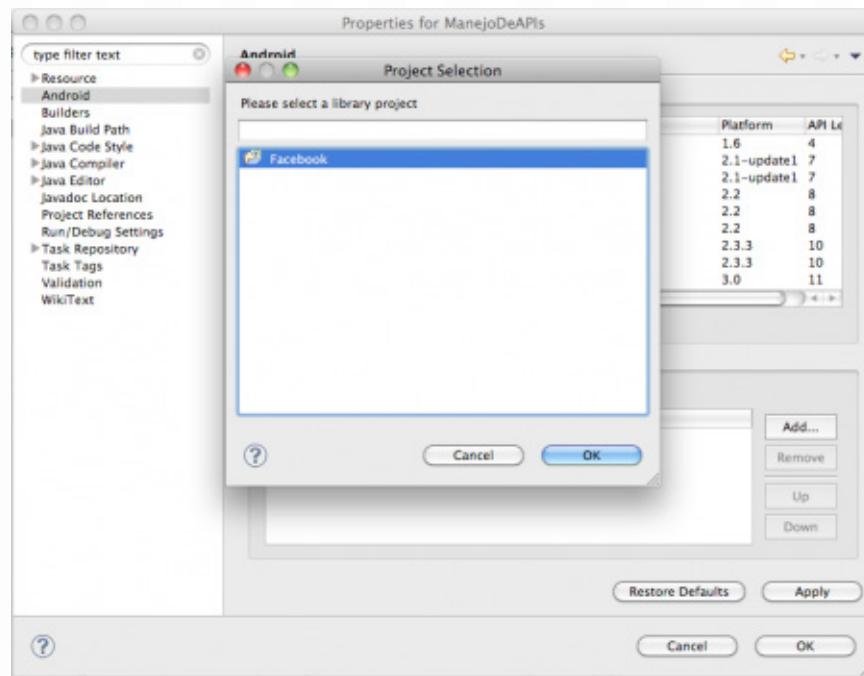
Luego de que está listo el proyecto debemos indicarle que tome la librería del SDK desde otro proyecto, hacemos click derecho sobre el proyecto y luego en propiedades



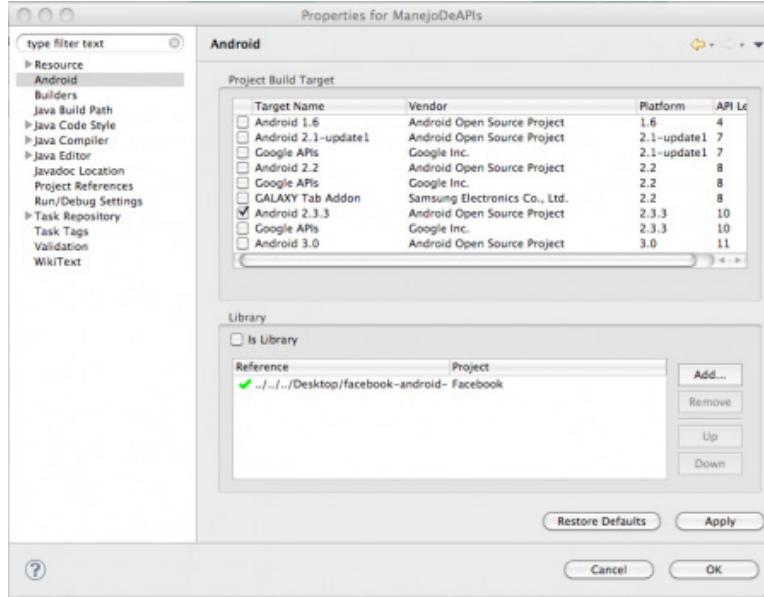
En la pantalla de las opciones buscamos la parte de Android:



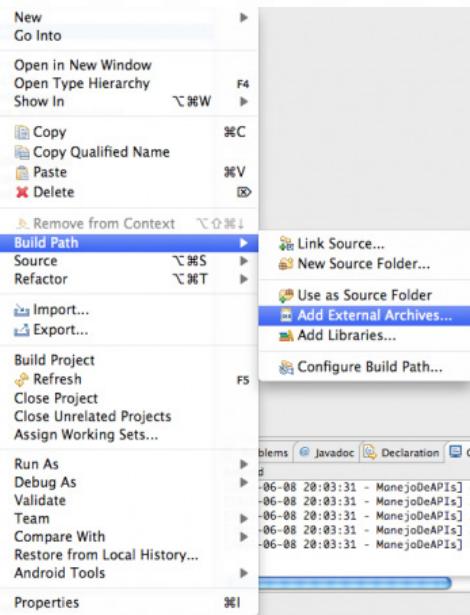
Hacemos click sobre “Add” en la parte de Library:



Deberá aparecernos el proyecto llamado “Facebook”, lo seleccionamos y estará lista la referencia:



Para la autenticación con Twitter hay varias opciones en nuestro caso estaremos utilizando **OAuth Sign-Post**<sup>1</sup> y para hacerlo parte de nuestro proyecto necesitamos las librerías de Core y Apache Common, ambos son archivos **JAR descargables**<sup>2</sup>, la forma de incluirlos es haciendo click derecho sobre el proyecto, luego “Build Path” y por último “Add External Archives”.



1 <http://code.google.com/p/oauth-signpost/>

2 <http://code.google.com/p/oauth-signpost/downloads/list>



Por último, vamos a configurar algunas cosas en el **Manifest** necesitamos permisos para internet:

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

Para la autenticación de Twitter utilizaremos un intent filter que permita que el callback luego de la autorización sea nuestra aplicación, para ello usamos el tag de data con un esquema y un host definido por nosotros, es decir, el URL de callback será *mdw://twitter*

```
<intent-filter>
<action android:name="android.intent.action.VIEW"></action>
<category android:name="android.intent.category.DEFAULT"></category>
<category android:name="android.intent.category.BROWSABLE"></category>
<data android:scheme="mdw" android:host="twitter"></data>
</intent-filter>
```

## Diseño

El diseño tendrá 2 botones para **iniciar/cerrar** sesión y 2 etiquetas para mostrar el status. En el caso de Facebook, vamos a utilizar el botón incluido en el ejemplo del SDK implementado en una clase llamada **LoginButton**.

Código fuente



```
<?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"
    >
    <com.facebook.android.LoginButton
        android:id="@+id/btnFbLogin"
        android:src="@drawable/login_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        />
    <TextView
        android:layout_height="wrap_content"
        android:text="Facebook status: sesión no iniciada"
        />

```

```
    android:id="@+id/txtFbStatus"
    android:layout_width="fill_parent"></TextView>
<View
    android:layout_height="1dip"
    android:layout_width="fill_parent"
    android:background="#FFFFFF"
/>
<Button
    android:text="Twitter"
    android:id="@+id/btnTwLogin"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"></Button>
<TextView
    android:layout_height="wrap_content"
    android:text="Twitter status: sesión no iniciada"
    android:id="@+id/txtTwStatus"
    android:layout_width="fill_parent"></TextView>
</LinearLayout>
```

## Agregando código

Inicialmente, vamos a definir algunas variables globales dentro de nuestra Activity. El manejo del botón de **autorizar/de-autorizar** de Twitter lo haremos nosotros (el de Facebook tiene una implementación en el código importado que vamos a aprovechar) para ellos lo declaramos como una variable global y declaramos los 2 listeners que utilizaremos.

```
private Button btnTwLogin;
private OnClickListener twitter_auth, twitter_clearauth;
```

Además requerimos de banderas para chequear el estatus de autenticación y etiquetas para mostrar el mismo.

```
private TextView txtFbStatus, txtTwStatus;
private boolean twitter_active = false, facebook_active = false;
```

Las peticiones para interactuar con el Graph API de Facebook son asincrónicas y las realizamos a través de una instancia de la clase AsyncFacebookRunner.

```
private AsyncFacebookRunner mAsyncRunner;
```

También definimos el identificador de la aplicación de Facebook (recuerden cambiarlo por el la aplicación que ustedes solicitaron).

```
public static final String APP_ID = "175729095772478";
```

Para la parte de autenticación con Twitter, necesitamos un provider y un consumer configurados con los URLs (no cambian), el consumer key (es necesario cambiarlo por el de ustedes) y el consumer secret (también es necesario cambiarlo por la aplicación que ustedes solicitaron).

#### Código fuente



```
private static CommonsHttpOAuthProvider provider =
new CommonsHttpOAuthProvider(
    "https://api.twitter.com/oauth/request_token",
    "https://api.twitter.com/oauth/access_token",
    "https://api.twitter.com/oauth/authorize");
private static CommonsHttpOAuthConsumer consumer =
new CommonsHttpOAuthConsumer(
    "7iEjG84wItGvXaIZFXAyZg",
    "sZKCJaUN8BgmYy4r9Z7h1I4BEHV8aAd6Ujw3hofQ4k");
```

Una vez realizada la autorización del usuario con su cuenta de Twitter, recibimos de vuelta un key y un secret de acceso y es necesario guardarlos para cualquier futura interacción.

```
private static String ACCESS_KEY = null;
private static String ACCESS_SECRET = null;
```

Estas últimas 4 variables son estáticas debido al funcionamiento de la librería OAuth SignPost en algunas ocasiones se pierden los valores luego de autenticar y previo a recibir el acceso entonces puede guardarse temporalmente y luego restaurarse o utilizarse variables estáticas, en este caso hemos tomado la segunda forma de solucionarlo.

Cuando el usuario se autentique con sus credenciales de Facebook, haremos una petición al Graph API para obtener sus datos (identificador y nombre), esta requisición es asincrónica y requiere un Request-Listener (parte del SDK de Facebook) para realizar alguna acción una vez se ha recibido respuesta.

Vamos a encapsular en un método esta llamada y en concreto nos concentraremos en el método onComplete del Listener para realizar lo que queremos. La respuesta recibida en JSON es necesario reconocerla (parsing) y luego enviar los valores que nos interesan (ID y nombre) al hilo de ejecución(thread) principal para que modifique la vista(únicamente él, el thread principal, puede hacer este cambio en el contenido del TextView) .

## Código fuente



```
private void updateFbStatus() {
    mAsyncRunner.request("me", new RequestListener() {
        @Override
        public void onMalformedURLException(MalformedURLException e, Object state) {}
        @Override
        public void onIOException(IOException e, Object state) {}
        @Override
        public void onFileNotFoundException(FileNotFoundException e, Object state) {}
        @Override
        public void onFacebookError(FacebookError e, Object state) {}
        @Override
        public void onComplete(String response, Object state) {
            try {
                JSONObject json = Util.parseJson(response);
                final String id = json.getString("id");
                final String name = json.getString("name");
                Main.this.runOnUiThread(new Runnable() {
                    @Override
                    public void run() {
                        txtFbStatus.setText("Facebook status: sesión
iniciada como " + name + " con el id " + id);
                    }
                });
            } catch (JSONException e) {
                e.printStackTrace();
            } catch (FacebookError e) {
```

```
e.printStackTrace();  
}  
}  
});  
}
```

Además de este método, vamos a trabajar en onCreate. Primero obtenemos botones y etiquetas (TextView) de status tanto de Facebook como de Twitter.

```
LoginButton mLoginButton = (LoginButton) findViewById(R.id.btnFbLogin);  
btnTwLogin = (Button) findViewById(R.id.btnTwLogin);  
txtTwStatus = (TextView) this.findViewById(R.id.txtTwStatus);  
txtFbStatus = (TextView) this.findViewById(R.id.txtFbStatus);
```

Luego, construimos nuestro objeto Facebook (parte del SDK descargado) y a partir de él un objeto AsyncFacebookRunner que vamos a utilizar para las peticiones al GraphAPI.

```
Facebook mFacebook = new Facebook(APP_ID);  
mAsyncRunner = new AsyncFacebookRunner(mFacebook);
```

Vamos a guardar las credenciales de Twitter como preferencias de la aplicación (funciona como un diccionario, key/value), más adelante vemos la forma de guardarlas al realizar la autenticación, en onCreate vamos a recuperarlas y si no existen tenemos un valor por defecto que es un String vacío. Si existen las credenciales entonces habilitamos la variable boolean twitter\_active.

```
SharedPreferences prefs = PreferenceManager.getDefaultSharedPreferences(this);  
String stored_keys = prefs.getString("KEY", "");  
String stored_secret = prefs.getString("SECRET", "");  
if (!stored_keys.equals("") && !stored_secret.equals("")) {  
    twitter_active = true;  
}
```

Realizamos un proceso similar para el caso de Facebook, aquí es más sencillo porque no es tarea nuestra la administración de las credenciales, nos apoyamos en el código del ejemplo del SDK. Restauramos la sesión y luego validamos la misma, en base a esto habilitamos la bandera facebook\_active y si la sesión está activa entonces mostraremos los datos del usuario llamando a updateFbStatus().

```
SessionStore.restore(mFacebook, this);  
facebook_active = mFacebook.isSessionValid();
```

```
if (facebook_active) {  
    updateFbStatus();  
}
```

Para el manejo del inicio de sesión de Facebook es necesario un Listener para autenticación y otro para cuando se finaliza sesión. Para el primero (AuthListener) es necesario sobrecargar métodos dependiendo si tuvo éxito o no, en caso de tener éxito llamamos a updateFbStatus y en caso de fallar reportamos el error a través del TextView de estatus.

  
Código fuente

```
SessionEvents.addAuthListener(new AuthListener() {  
    @Override  
    public void onAuthSucceeded() {  
        updateFbStatus();  
    }  
    @Override  
    public void onAuthFailed(String error) {  
        txtFbStatus.setText("Facebook status: imposible iniciar sesión " +  
            error);  
    }  
});
```

En el caso del listener para finalizar la sesión, necesitamos sobrecargar métodos para el inicio y finalización del proceso de cierre de sesión, le reportaremos al usuario estos eventos a través del TextView de estatus.

  
Código fuente

```
SessionEvents.addLogoutListener(new LogoutListener() {  
    @Override  
    public void onLogoutFinish() {  
        txtFbStatus.setText("Facebook status: sesión no iniciada");  
    }  
    @Override  
    public void onLogoutBegin() {  
        txtFbStatus.setText("Facebook status: cerrando sesión...");  
    }  
});
```

```
 }  
});
```

Finalizamos con la inicialización el botón de login de Facebook:

```
mLoginButton.init(this, mFacebook);
```

Para la parte de Twitter manejaremos 2 Listeners para el evento del click, esta vez vamos a definirlos y asignarles nombre en vez de usar clases internas y anónimas porque los asignaremos varias veces (dependiendo del estado de la autenticación).

El primer listener, el utilizado cuando aun no se inicia sesión, requiere que al darle click al botón obtenga el requestToken y a través de un intent con el URL recibido levante una actividad (el navegador) y muestre al usuario la pantalla de Twitter solicitando su aprobación.

#### Código fuente



```
twitter_auth = new OnClickListener() {  
    @Override  
    public void onClick(View v) {  
        txtTwStatus.setText("Twitter status: iniciando sesión");  
        try {  
            String authUrl = provider.retrieveRequestToken(consumer,  
                "mdw://twitter");  
            startActivity(new Intent(Intent.ACTION_VIEW, Uri.parse(authUrl)));  
        } catch (OAuthMessageSignerException e) {  
            e.printStackTrace();  
        } catch (OAuthNotAuthorizedException e) {  
            e.printStackTrace();  
        } catch (OAuthExpectationFailedException e) {  
            e.printStackTrace();  
        } catch (OAuthCommunicationException e) {  
            e.printStackTrace();  
        }  
    }  
};
```

El segundo listener, una vez ya ha ocurrido la autenticación, borrará las credenciales guardadas en los SharedPreferences.



#### Código fuente

```
twitter_auth = new OnClickListener() {
    @Override
    public void onClick(View v) {
        txtTwStatus.setText("Twitter status: iniciando sesión");
        try {
            String authUrl = provider.retrieveRequestToken(consumer,
                    "mdw://twitter");
            startActivity(new Intent(Intent.ACTION_VIEW, Uri.parse(authUrl)));
        } catch (OAuthMessageSignerException e) {
            e.printStackTrace();
        } catch (OAuthNotAuthorizedException e) {
            e.printStackTrace();
        } catch (OAuthExpectationFailedException e) {
            e.printStackTrace();
        } catch (OAuthCommunicationException e) {
            e.printStackTrace();
        }
    }
};
```

Es importante notar que las credenciales recibidas de Twitter (key y secret) no expiran y son válidas mientras el usuario no revoque los permisos de la aplicación, lo que hacemos al desautorizar es borrar las credenciales guardadas por lo que la siguiente vez que presionemos el botón solicitaremos nuevas.

Para terminar el método onCreate revisamos el valor de twitter\_active y actualizamos el texto del botón y del TextView de estatus.



#### Código fuente

```
if (twitter_active) {
    txtTwStatus.setText("Twitter status: sesión iniciada ");
    btnTwLogin.setText("Deautorizar twitter");
}
```

```
btnTwLogin.setOnClickListener(twitter_clearauth);
} else {
btnTwLogin.setText("Autorizar twitter");
btnTwLogin.setOnClickListener(twitter_auth);
}
```

Además del trabajo realizado en OnCreate, necesitamos modificar OnResume, una vez autorizado el uso de la aplicación de Twitter por parte del usuario, al recibir el redirect al callback nuestra aplicación lo recibirá por el intent-filter colocado previamente y se ejecutará el método OnResume.

Para distinguir si la ejecución es por cualquier interrupción de la aplicación o porque ya nos autenticamos con Twitter revisamos la data del intent y que esta sea un URI de la forma que definimos (mdw://twitter), si ese fuera el caso entonces recibimos token y secret de acceso y lo guardamos.

```
provider.retrieveAccessToken(consumer, verifier);
ACCESS_KEY = consumer.getToken();
ACCESS_SECRET = consumer.getTokenSecret();
```

Para almacenar las credenciales de forma persistente utilizaremos **SharedPreferences**<sup>1</sup> es necesario obtener un editor, guardar la data y luego realizar commit.

```
SharedPreferences prefs = PreferenceManager.getDefaultSharedPreferences(this);
SharedPreferences.Editor editor = prefs.edit();
editor.putString("KEY", ACCESS_KEY);
editor.putString("SECRET", ACCESS_SECRET);
editor.commit();
```

Luego actualizamos el TextView del estado, el texto del botón y la acción asociada al mismo.

```
TextView txtTwStatus = (TextView) this.findViewById(R.id.txtTwStatus);
txtTwStatus.setText("Twitter status: sesión iniciada ");
btnTwLogin.setText("Deautorizar twitter");
btnTwLogin.setOnClickListener(twitter_clearauth);
```

Adicional a esto y fuera de la condición de que el intent tenga data, es posible que el método onResumbe sea llamado y los valores del TextView de status de Facebook se hayan perdido entonces es necesario

---

<sup>1</sup> <http://developer.android.com/reference/android/content/SharedPreferences.html>

revisarlo.

```
if (facebook_active) {  
    updateFbStatus();  
}
```

El método completo onResume queda de la siguiente forma:

Código fuente



```
@Override  
public void onResume() {  
    super.onResume();  
    Uri uri = this.getIntent().getData();  
    if (uri != null && uri.toString().startsWith("mdw://twitter")) {  
        String verifier = uri.getQueryParameter(OAuth.OAUTH_VERIFIER);  
        try {  
            provider.retrieveAccessToken(consumer, verifier);  
            ACCESS_KEY = consumer.getToken();  
            ACCESS_SECRET = consumer.getTokenSecret();  
            SharedPreferences prefs =  
                PreferenceManager.getDefaultSharedPreferences(this);  
            SharedPreferences.Editor editor = prefs.edit();  
            editor.putString("KEY", ACCESS_KEY);  
            editor.putString("SECRET", ACCESS_SECRET);  
            editor.commit();  
            TextView txtTwStatus = (TextView)  
                this.findViewById(R.id.txtTwStatus);  
            txtTwStatus.setText("Twitter status: sesión iniciada");  
            btnTwLogin.setText("Deautorizar twitter");  
            btnTwLogin.setOnClickListener(twitter_clearauth);  
        } catch (OAuthMessageSignerException e) {  
            e.printStackTrace();  
        } catch (OAuthNotAuthorizedException e) {  
            e.printStackTrace();  
        } catch (OAuthExpectationFailedException e) {  
            e.printStackTrace();  
        } catch (Exception e) {  
            e.printStackTrace();  
        }  
    }  
}
```

```
    } catch (OAuthCommunicationException e) {  
        e.printStackTrace();  
    }  
    }  
    if (facebook_active) {  
        updateFbStatus();  
    }  
}
```



Pregunta en el Foro Android  
de Foros del Web



Descarga el código  
completo en

```
git clone https://github.com/androidMDW/guia9completo  
github SOCIAL CODING
```



## Conclusión

En este capítulo hemos visto varias cosas:

- Utilización de un proyecto como librería (SDK de Facebook para Android), es posible crear un proyecto sin Activities si no únicamente con código para ser utilizado por terceros. En este caso, nosotros aprovechamos este código creado por alguien más para nuestra aplicación.
- Uso de librerías externas a través de archivos JAR que se vuelven parte del proyecto, en nuestra aplicación de demo de esta forma incluimos OAuth SignPost.
- Conexión con APIs de terceros (Facebook & Twitter) para lograr la autenticación, vimos dos opciones que podemos manejar para poder identificar a los usuarios de nuestras aplicaciones. La librería de OAuth SignPost es posible utilizarla también con algunos otros proveedores que trabajan también con OAuth.

# 10

capítulo



## Conectándonos con Apis de Google

## CAPÍTULO 10

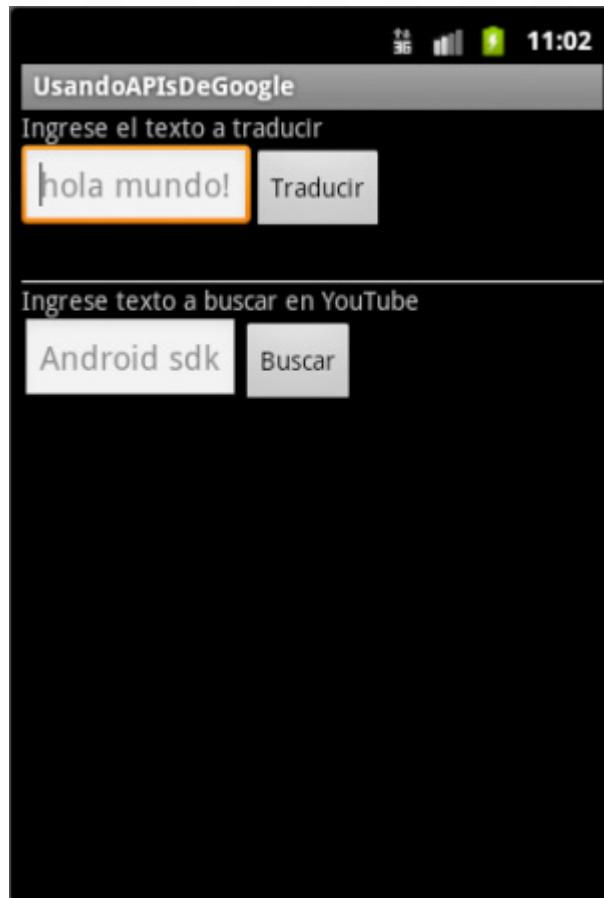
## Conectándonos con Apis de Google

En el decimo capítulo seguimos trabajando con APIs esta vez utilizaremos conexión con 2 APIs de Google: el de traducción y el de búsqueda de vídeos en YouTube.

## Conexión con APIs de Google

La aplicación que realizaremos nos permitirá realizar la traducción de un texto pequeño de español a inglés y además realizar una búsqueda en YouTube.

Queremos que al finalizar se vea así:



## Disposición inicial

El tipo de requisiciones que realizaremos a estos APIs no necesita de autenticación entonces no necesitamos validar ningún tipo de credenciales. Si quisiéramos, por ejemplo, subir un video a YouTube sí sería necesario autenticarnos.

Para no realizar el trabajo desde cero nos apoyaremos en 2 librerías, una para cada API:

- » Página del proyecto: [google-api-translate-java](#) <sup>1</sup>
- » Código fuente disponible en [Github para descargar](#) <sup>2</sup>
- » Página del proyecto: [google-api-java-client](#) <sup>3</sup>
- » La [librería para descargar](#) <sup>4</sup>

Vamos a usar esta librería en este demo sin preocuparnos mucho del tamaño final de la aplicación, al usarla en un ambiente de producción es muy importante utilizar ProGuard para reducir el tamaño, encuentran instrucciones detalladas en <http://code.google.com/p/google-api-client/wiki/Setup>

Vamos a importar varias librerías al proyecto, no utilizaremos código base de nuevo para seguir paso a paso la configuración:

- » Para la parte de traducción, es un único archivo llamado: google-api-translate-java-0.95
- » Para la parte de YouTube, del archivo zip que descargamos vamos a elegir 4 archivos, esta descarga trae varias librerías y sus fuentes, la descripción la encuentran en <http://code.google.com/p/google-api-client/wiki/Setup> nosotros vamos a utilizar del directorio raíz.
- » google-api-client-1.4.1-beta
- » google-api-client-googleapis-1.4.1-beta

Además, para la conexión HTTP y el parsing de JSON otras 2 de las dependencias: del archivo descargado de google-api-client dentro de la carpeta dependencies necesitamos:

- » guava-r09

1 <http://google-api-translate-java.googlecode.com/files/google-api-translate-java-0.95.jar>

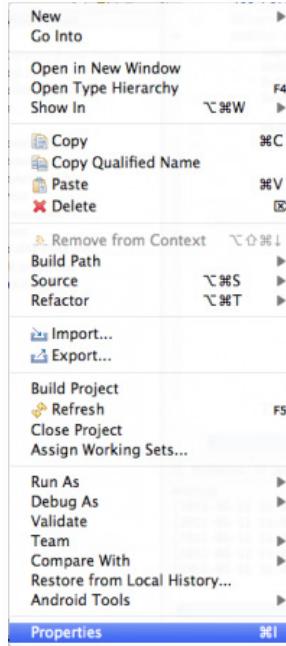
2 <http://github.com/richmidwinter/google-api-translate-java>

3 <http://code.google.com/p/google-api-client/>

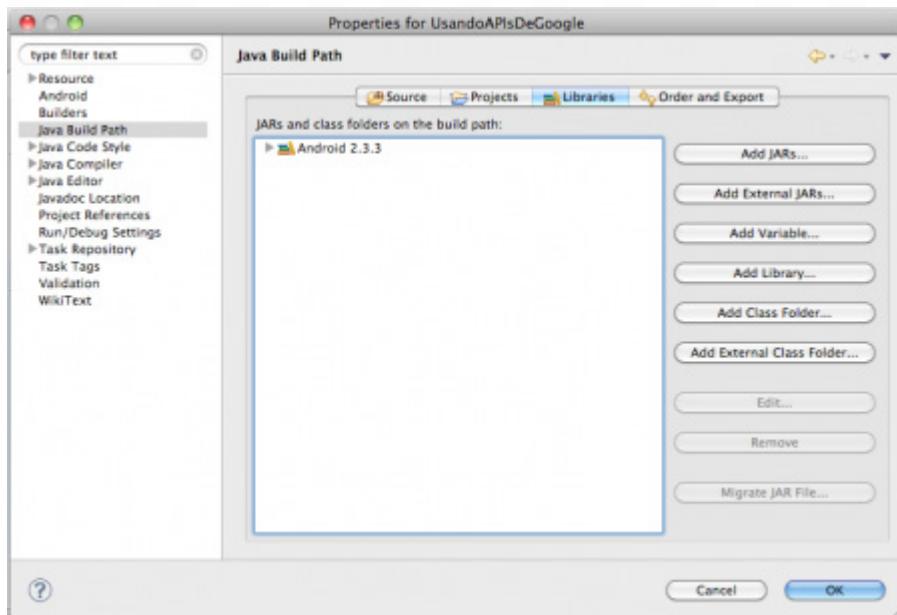
4 <http://google-api-client.googlecode.com/files/google-api-client-1.4.1-beta.zip>

## ➤ jackson-core-asl-1.6.7

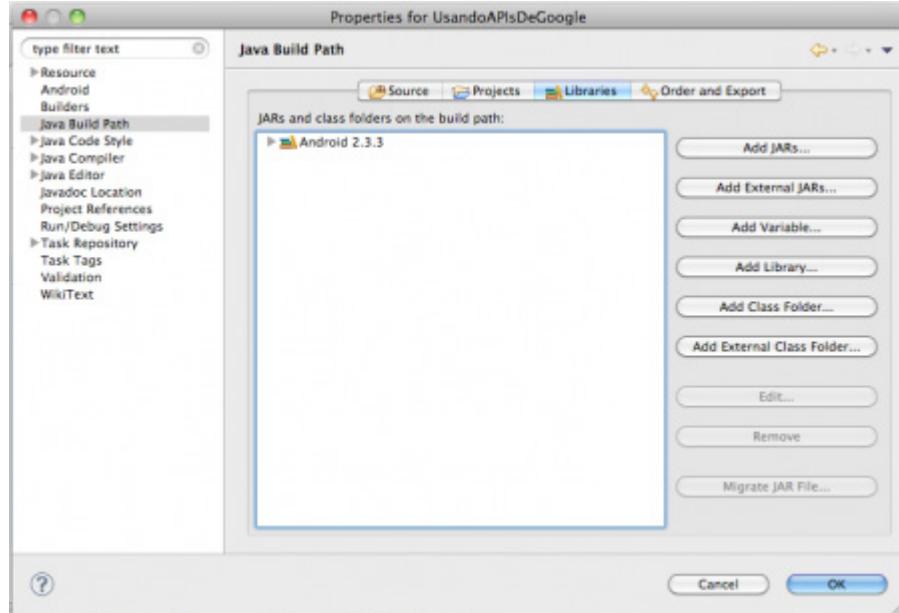
Para incluir nuestras librerías necesitamos hacer click derecho sobre el proyecto, luego propiedades:



Buscamos la pestaña de “Java Build Path” y de los botones del lado derecho hacemos click en “Add External JARs”:



Seleccionamos todas los archivos **JAR** de las librerías ya descritas y estamos listos:



Por último, vamos a configurar algunas cosas en el Manifest necesitamos permisos para Internet:

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

## Diseño

El diseño tendrá 2 partes, para la traducción utilizaremos un TextView que le indique al usuario que hacer

```
<TextView  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    android:text="Ingrese el texto a traducir"  
/>
```

Luego un LinearLayout para mostrar en fila un EditText para recibir la cadena a traducir y un botón para realizar la traducción:

Código fuente



```
<LinearLayout android:layout_width="match_parent"  
    android:layout_height="wrap_content" android:id="@+id/linearLayout1">  
    <EditText android:id="@+id/etTextToTranslate"  
        android:hint="hola mundo!"  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"></EditText>  
    <Button android:text="Traducir"  
        android:id="@+id/btnTranslate"  
        android:layout_width="wrap_content"  
        android:layout_height="wrap_content"></Button>  
</LinearLayout>
```

Por último otro TextView que muestre el resultado.

```
<TextView android:id="@+id/txtTranslatedText"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:textSize="20dp"></TextView>
```

Separamos de la siguiente sección a través de una línea horizontal:

```
<View  
    android:layout_height="1dip"
```

```
    android:layout_width="fill_parent"
    android:background="#FFFFFF"
    />
```

Para la siguiente sección, la de búsqueda en YouTube de manera similar que el caso anterior indicamos al usuario a través de un TextView que debe ingresar el parámetro de búsqueda:

```
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Ingrese texto a buscar en YouTube"
    />
```

Luego, con otro LinearLayout horizontal que dentro tiene un EditText esperamos la cadena que será el parámetro de búsqueda y un botón para realizar la búsqueda:

Código fuente



```
<LinearLayout android:layout_width="match_parent"
    android:layout_height="wrap_content" android:id="@+id/linearLayout2">
    <EditText android:id="@+id/etQueryText"
        android:hint="Android sdk"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"></EditText>
    <Button android:text="Buscar"
        android:id="@+id/btnSearch"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"></Button>
</LinearLayout>
```

Terminamos con un ListView para mostrar el resultado de la búsqueda:

```
<ListView android:layout_height="wrap_content"
    android:id="@+id/lstVideo"
    android:layout_width="match_parent"></ListView>
```

El diseño completo queda de la siguiente forma:

Código fuente



```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Ingrese el texto a traducir"
        />
    <LinearLayout android:layout_width="match_parent"
        android:layout_height="wrap_content" android:id="@+id/linearLayout1">
        <EditText android:id="@+id/etTextToTranslate"
            android:hint="hola mundo!"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"></EditText>
        <Button android:text="Traducir"
            android:id="@+id/btnTranslate"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"></Button>
    </LinearLayout>
    <TextView android:id="@+id/txtTranslatedText"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="20dp"></TextView>
    <View
        android:layout_height="1dip"
        android:layout_width="fill_parent"
        android:background="#FFFFFF"
        />
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
```

```
        android:text="Ingrese texto a buscar en YouTube"
    />
<LinearLayout android:layout_width="match_parent"
    android:layout_height="wrap_content" android:id="@+id/linearLayout2">
    <EditText android:id="@+id/etQueryText"
        android:hint="Android sdk"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"></EditText>
    <Button android:text="Buscar"
        android:id="@+id/btnSearch"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"></Button>
</LinearLayout>
<ListView android:layout_height="wrap_content"
    android:id="@+id/lstVideo"
    android:layout_width="match_parent"></ListView>
</LinearLayout>
```

## Agregando código

Para el manejo de la respuesta de YouTube (en JSON) vamos a construir clases de Java que representen esta respuesta de tal forma que el proceso de parsing sea más sencillo. Estas clases serán internas y estáticas (más información en [download.oracle.com<sup>1</sup>](http://download.oracle.com/javase/tutorial/java/javaOO/nested.html)) dentro de la clase de nuestra Activity principal.

### Construiremos 3 clases:

Una para representar el resultado completo del request, una lista de ítems se llamará *VideoFeed*.

```
public static class VideoFeed {
    @Key List<Video> items;
}
```

Otra para representar cada ítem del resultado, del que nos interesa el título(atributo title) y el URL del video (atributo player contiene el URL por defecto y el móvil), se llamará *Video*. Esta clase además

---

<sup>1</sup> <http://download.oracle.com/javase/tutorial/java/javaOO/nested.html>

implementará el método `toString` para mostrar el título en un `ListView` más adelante.

```
public static class Video {  
    @Key String title;  
    @Key Player player;  
    public String toString(){  
        return this.title;  
    }  
}
```

Una para representar el player utilizado en el caso anterior y que contiene la URL que vamos a utilizar para mostrar el video cuando el usuario haga click sobre el título se llamará: `Player`.

```
public static class Player {  
    @Key String mobile;  
}
```

La funcionalidad de la aplicación la trabajaremos dentro del método `onCreate`, en específico en el método `onClick` de `Listeners` asociados a los botones. Para la sección de traducción, primero le indicamos a Google desde que URL vamos a usar su **API de Translate**:

```
Translate.setHttpReferrer("www.ejemplo.com");
```

Luego, vamos a traer lo que ingresó el usuario en el `EditText` y escondemos el teclado para tener libre la pantalla para ver nuestra aplicación.

```
EditText etTextToTranslate = (EditText) findViewById(R.id.etTextToTranslate);  
String textToTranslate = etTextToTranslate.getText().toString();  
InputMethodManager imm =  
(InputMethodManager) getSystemService(Context.INPUT_METHOD_SERVICE);  
imm.hideSoftInputFromWindow(etTextToTranslate.getWindowToken(), 0);
```

Con esta cadena, usamos la librería para acceder al API de traducción indicando que el idioma fuente es español y el idioma objetivo es inglés.

```
String translatedText = Translate.execute(textToTranslate, Language.SPANISH,  
Language.ENGLISH);
```

Una vez recibida la respuesta, la mostramos al usuario en un `TextView`:

```
TextView txt = (TextView) findViewById(R.id.txtTranslatedText);
```

```
txt.setText(translatedText);
```

Para la sección de la búsqueda en YouTube iniciamos obteniendo el parámetro de búsqueda y ocultando el teclado.

```
EditText etQueryText = (EditText) findViewById(R.id.etQueryText);
String queryText = etQueryText.getText().toString();
InputMethodManager imm =
(InputMethodManager) getSystemService(Context.INPUT_METHOD_SERVICE);
imm.hideSoftInputFromWindow(etQueryText.getWindowToken(), 0);
```

Utilizando este parámetro de búsqueda, preparamos el URL del request, indicamos que la respuesta la queremos en JSON y que la cantidad máxima de resultados será 5.

```
GenericUrl url = new GenericUrl("https://gdata.youtube.com/feeds
/api/videos?alt=json&max-results=5&q=" + queryText);
```

Preparamos la requisición que requiere de transporte http y el parser de JSON:

```
HttpTransport transport = new NetHttpTransport();
final JsonFactory jsonFactory = new JacksonFactory();
```

Usando la instancia recién creada de HttpTransport, vamos a construir un RequestFactory indicándole el tipo de parser y los encabezados para el request.

Código fuente



```
HttpRequestFactory factory = transport.createRequestFactory(new
HttpRequestInitializer() {
@Override
public void initialize(HttpRequest request) {
//indicamos el parser a utilizar
JsonCParser parser = new JsonCParser();
parser.jsonFactory = jsonFactory;
request.addParser(parser);
//indicamos el nombre de la aplicación y la versión de gData
//JSON es soportado solo en versión 2 de gData
GoogleHeaders headers = new GoogleHeaders();
headers.setApplicationName("Maestros-Del-Web-Android/1.0");
```

```
headers.gdataVersion = "2";
request.headers = headers;
}
});
```

Con los parámetros listos, construimos el request, lo ejecutamos y lo hacemos el parsing (reconocemos) indicando la clase VideoFeed previamente construida como resultado que deseamos.

```
HttpRequest request = factory.buildGetRequest(url);
final VideoFeed feed = request.execute().parseAs(VideoFeed.class);
```

El resultado guardado en esta variable feed (un listado de ítems Video) lo utilizaremos para construir un ArrayAdapter que luego se lo asociamos a nuestro ListView para poder visualizar el resultado en forma de listado en la aplicación.

```
ArrayAdapter<Video> adpList = new ArrayAdapter<Video>(
getApplicationContext(),
android.R.layout.simple_list_item_1,
feed.items);
ListView videoList = (ListView) findViewById(R.id.lstVideo);
videoList.setAdapter(adpList);
```

Para finalizar, queremos que al presionar sobre el título de cualquier video del resultado podemos verlo, para esto hacemos uso del método setOnItemClickListener del ListView y en el método onItemClick obtenemos de la lista el elemento seleccionado y de él la URL móvil que tiene asociada.

Con esta URL iniciamos una actividad que nos llevará a poder visualizar el video.

```
Video item = feed.items.get(position);
startActivity(new Intent(Intent.ACTION_VIEW, Uri.parse(item.player.mobile)));
```

El código del método onCreate queda de la siguiente forma:

Código fuente



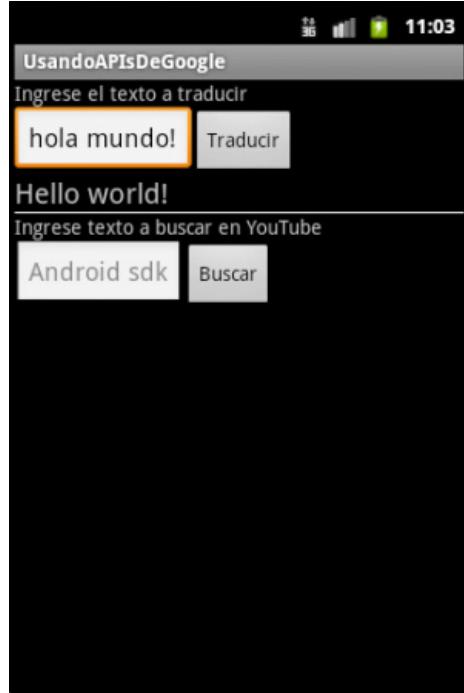
```
@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.main);
```

```
Button btnTranslate = (Button) findViewById(R.id.btnTranslate);
Button btnSearch = (Button) findViewById(R.id.btnSearch);
btnTranslate.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View v) {
        Translate.setHttpReferrer("www.ejemplo.com");
        try {
            EditText etTextToTranslate =
                    (EditText) findViewById(R.id.etTextToTranslate);
            String textToTranslate = etTextToTranslate.getText().toString();
            InputMethodManager imm =
                    (InputMethodManager) getSystemService(Context.INPUT_METHOD_SERVICE);
            imm.hideSoftInputFromWindow(etTextToTranslate.getWindowToken(), 0);
            String translatedText = Translate.execute(textToTranslate,
                    Language.SPANISH, Language.ENGLISH);
            TextView txt = (TextView) findViewById(R.id.txtTranslatedText);
            txt.setText(translatedText);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
});
btnSearch.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View v) {
        EditText etQueryText = (EditText) findViewById(R.id.etQueryText);
        String queryText = etQueryText.getText().toString();
        InputMethodManager imm =
                    (InputMethodManager) getSystemService(Context.INPUT_METHOD_SERVICE);
        imm.hideSoftInputFromWindow(etQueryText.getWindowToken(), 0);
        HttpTransport transport = new NetHttpTransport();
        GenericUrl url = new GenericUrl("https://gdata.youtube.com/feeds
/api/videos?alt=json&max-results=2&q=" + queryText);
        final JsonFactory jsonFactory = new JacksonFactory();
```

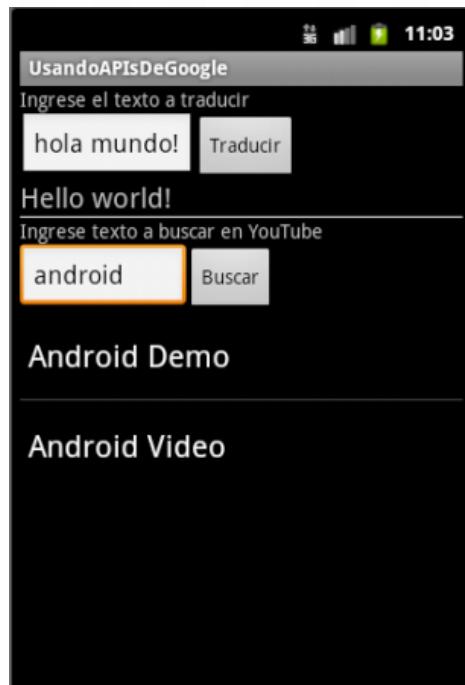
```
HttpRequestFactory factory = transport.createRequestFactory(new
HttpRequestInitializer() {
@Override
public void initialize(HttpRequest request) {
JsonCParser parser = new JsonCParser();
parser.jsonFactory = jsonFactory;
request.addParser(parser);
GoogleHeaders headers = new GoogleHeaders();
headers.setApplicationName("Maestros-Del-Web-Android/1.0");
headers.gdataVersion = "2";
request.headers = headers;
}});
try {
HttpRequest request = factory.buildGetRequest(url);
final VideoFeed feed =
request.execute().parseAs(VideoFeed.class);
ListView videoList = (ListView)findViewById(R.id.lstVideo);
ArrayAdapter<Video> adpList = new ArrayAdapter<Video>
(getApplicationContext(), android.R.layout.simple_list_item_1, feed.items);
videoList.setAdapter(adpList);
videoList.setOnItemClickListener(new OnItemClickListener() {
@Override
public void onItemClick(AdapterView<?> arg0,
View arg1, int position, long arg3) {
Video item = feed.items.get(position);
startActivity(new Intent(Intent.ACTION_VIEW,
Uri.parse(item.player.mobile)));
}});
} catch (IOException e) {
e.printStackTrace();
}}
});
```

Al ejecutar nuestra aplicación, probamos individualmente cada una de las 2 partes, primero la traduc-

ción:



Luego la búsqueda en YouTube:





Pregunta en el Foro Android  
de Foros del Web



<http://www.forosdelweb.com/f165/>

Descarga el código  
completo en

```
if (true) {  
    AlertDialog.Builder builder = new  
    AlertDialog.Builder(Main.this);  
    builder.setMessage("ya ha cargado datos. ¿Está seguro de  
    hacerlo de nuevo?");  
    builder.setCancelable(false);  
    builder.setPositiveButton("SI",  
        DialogInterface.OnClickListener(){  
            public void onClick(DialogInterface dialog, int id){  
                //...  
            }  
        });  
    builder.show();  
}
```

github  
SOCIAL CODING



<https://github.com/androidMDW/guia10completo>

## Conclusión

En este capítulo hemos aprendido a trabajar con 2 de los muchos APIs de google (<http://code.google.com/more/table/>) a través del uso de librerías desarrolladas por terceros. Para la parte de traducción utilizamos una librería específica (no olviden que este API está próximo a morir <http://code.google.com/apis/language/translate/overview.html>) y para la parte de YouTube usamos una librería que permite conexión también a Latitude, Buzz, Books, Moderator y más (<http://code.google.com/p/google-api-java-client/wiki/APILibraries>).

## COMUNIDADES ANDROID

## Curso Android

El objetivo del **Curso Android**<sup>1</sup> era introducirte en el desarrollar para Android, pasamos de lo general (Activities, Intents, UI) a cosas más específicas (cámara, fotos, videos, mapas, sensores, servicios, emails y APIs) y que cada uno se lance a desarrollar con mucho entusiasmo.

Android sigue creciendo y junto a él los desarrolladores que apuestan por esta plataforma, sigue a **@maestros**<sup>2</sup> y **@forosdelweb**<sup>3</sup> para seguir aprendiendo sobre Android y sobre otras tecnologías que están marcando tendencias.

## Más comunidades Android

### And.roid.es

<http://and.roid.es/>



La comunidad española And.roid.es fue creada en octubre del 2008 por Luis Moreno (@carthesian) y **Israel Ferrer**<sup>4</sup> (@rallat) quienes han organizado varios eventos como: and.roid.es **Community Meetup**<sup>5</sup>, Androides Barcelona y Barcelona GTUG.

### Cultura Android Guatemala

<http://cultura-androidgt.org/>



Comunidad de entusiastas guatemaltecos creada por **Adrián Catalán**<sup>6</sup> (@ykaro) que fomentan la utilización y desarrollo de la plataforma móvil Android. Realizan reuniones con el objetivo de promover el desarrollo y en su blog encontrarás información sobre desarrollo y novedades.

1 <http://www.maestrosdelweb.com/editorial/curso-android/>

2 <http://twitter.com/maestros>

3 <http://twitter.com/forosdelweb>

4 <http://www.maestrosdelweb.com/editorial/israel-ferrer-camacho>

5 <http://and.roid.es/meetup>

6 <http://www.maestrosdelweb.com/autores/acatalan/>



Droid.cl  
<http://droid.cl/>

Comunidad chilena del sistema operativo Android en donde encontrarás información sobre las aplicaciones, dispositivos y las últimas noticias sobre Android.



Android Perú  
<http://www.android-peru.com/>

El objetivo de Android-Peru.com es fomentar el desarrollo de Android en el Perú y Latinoamérica compartir experiencias, problemas y proyectos.



AndroidTitlan  
<http://androidtitlan.org/>

Es una comunidad Latinoamericana creada por Ángel Espinoza, Community Manager de Google Technologies User Group México City y el desarrollador de tecnología móvil Enrique Díaz. AndroidTitlan quiere decir “Lugar de los Androids” o “Región de los Andorids” y tiene como objetivo principal fomentar el desarrollo para la plataforma móvil Android SDK.

## OTRAS GUÍAS DE MAESTROS DEL WEB

## Guías online



### Adictos a la comunicación

Utiliza las herramientas sociales en Internet para crear proyectos de comunicación independientes.

#### [Visita Adictos a la comunicación](#)

<http://mdw.li/guiacomunica>



### iPhone

Crea tu propia aplicación para dispositivos móviles basados en iOS, y descubre las posibilidades de trabajo que este mercado ofrece a los desarrolladores.

#### [Visita la Guía iPhone](#)

<http://bit.ly/ManualiPhone>



### Startup

Aprende las oportunidades, retos y estrategias que toda persona debe conocer al momento de emprender.

#### [Visita la Guía Startup](#)

<http://mdw.li/gkTDom>



### Mapas

Aprende a utilizar el API de Google Maps para el desarrollo de tus proyectos con mapas

#### [Visita la Guía Mapas](#)

<http://bit.ly/guiamapas>



### Zend

Zend Framework es un framework de código abierto para desarrollar aplicaciones y servicios web con PHP 5.

#### [Visita la Guía Zend](#)

<http://mdw.li/guiazend>



### ASP.NET

ASP.NET es un modelo de desarrollo Web unificado creado por Microsoft para el desarrollo de sitios y aplicaciones web dinámicas con un mínimo de código. ASP.NET

#### [Visita la Guía ASP.NET](#)

<http://mdw.li/guaaspnet>



## Producción de Video

Tendencias en la creación de videos, herramientas, consejos y referencias en la producción de videos para la web.

[Visita la Guía de Producción de Vídeo](#)

<http://bit.ly/produccionvideo>