Programando para Android: una sencilla aplicación

Java, Android, Apps

Vamos a hablar sobre como programar una aplicación para un dispositivo con **Android** que se va a tratar de una sencilla calculadora.

Como todos sabemos, Android se programa mediante el lenguaje de programación **Java**. Este ejemplo de la calculadora puede parecer demasiado básico, pero con una idea inicial de como hacer una pequeña aplicación para Android y nociones sobre Java, veremos que desarrollar un programa más complejo es muy fácil.

En primer lugar, necesitaremos el **Android SDK** que lo podemos descargar de http://developer.android.com/sdk/index.html

Tras instalarlo deberemos ejecutar el archivo android dentro de la carpeta tools y hacer lo siguiente:

- Instalar la plataforma de la versión Android a emular, lo haremos desde la pestaña *Available packages* y elegiremos la versión que queramos (2.1, 2.2, 2.3, etc)
- Crear un dispositivo virtual, pestaña *Virtual devices*, que será nuestro dispositivo a emular.

Como segundo paso, haremos uso de la plataforma **Eclipse** y su plugin ADT para el desarrollo de Android bajo Eclipse. Podemos hacerlo desde la dirección http://developer.android.com/sdk/eclipse-adt.html

Ahora nos quedaría configurar Eclipse para que supiera la ruta hasta el SDK de Android y tendríamos todo preparado para trabajar. Lo podemos hacer desde el menú Window -> Preferences y dentro de la ventana en la pestaña Android tenemos la opción SDK Location.

Ahora tenemos todo preparado y configurado para empezar a crear aplicaciones. La web del SDK de Android contiene varios ejemplo de como empezar a programar y el ejemplo más sencillo es el del "*Hola Mundo*" donde viene todo el proceso indicado arriba de forma más detallada (lo podéis encontrar aquí http://developer.android.com/resources/tutorials/hello-world.html).

Nosotros nos vamos a basar en uno un poco más avanzado y lo vamos a ampliar hasta crear una aplicación con cierta utilidad. Este ejemplo es el de como posicionar elementos con tamaños y posiciones relativas http://developer.android.com/resources/tutorials/views/hello-relativelayout.html y vamos a añadir algunos elementos y funcionalidades para crear nuestra sencilla calculadora.

Creando nuestra primera aplicación para Android

Para ver cual queremos que sea la interfaz final e intuir el funcionamiento, vamos a mostrar una captura de pantalla del resultado final en el emulador:

donde podemos ver tres elementos: *TextView* (objeto para mostrar texto), *EditText* (caja de entrada de texto) y *Button* (botón).

Empezaremos creando en Eclipse un proyecto para Android, modificaremos el archivo **res/layout/main.xml** en su vista de código de fuente y pegaremos el siguiente código XML:

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <RelativeLayout
3  xmlns:android="http://schemas.android.com/apk/res/android"
4      android:layout_width="fill_parent"
5      android:layout_height="fill_parent">
6      <TextView
7          android:id="@+id/label_a"
8          android:layout_width="fill_parent"
9          android:layout_height="wrap_content"
10         android:text="Operando A:"/>
11     <EditText
12         android:id="@+id/op_a"
13         android:layout_width="fill_parent"
14         android:layout_height="wrap_content"
15         android:background="@android:drawable/editbox_background"
16         android:layout_below="@id/label_a"/>
17      <TextView
18         android:id="@+id/label_b"
19         android:layout_width="fill_parent"
```

```
        android:layout_height="wrap_content"
        android:layout_below="@id/op_a"
        android:text="Operando B:"/>
    <EditText
        android:id="@+id/op_b"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:background="@android:drawable/editbox_background"
        android:layout_below="@id/label_b"/>
    <Button
        android:id="@+id/sumar"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/op_b"
        android:layout_alignParentLeft="true"
        android:layout_marginLeft="1dip"
        android:onClick="cSumar"
        android:text="+" />
    <Button
        android:id="@+id/restar"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_toRightOf="@id/sumar"
        android:layout_alignTop="@id/sumar"
        android:onClick="cRestar"
        android:text="-" />
    <Button
        android:id="@+id/multiplicar"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_toRightOf="@id/restar"
        android:layout_alignTop="@id/restar"
        android:onClick="cMultiplicar"
        android:text="*" />
    <Button
        android:id="@+id/dividir"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_toRightOf="@id/multiplicar"
        android:layout_alignTop="@id/sumar"
        android:onClick="cDividir"
        android:text="/" />
    <TextView
        android:id="@+id/texto_resultado"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_below="@id/dividir"
        android:text="Resultado:"/>
    <TextView
        android:id="@+id/resultado"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_below="@id/texto_resultado"
        android:text="Realice operación para obtener
resultado"/>
</RelativeLayout>
```

Los atributos relevantes para este ejemplo de cada elemento son:

- *android:id* identificador de cada objeto
- *android:layout_width* y *android:layout_height*, anchura y altura respectivamente
- *android:layout_below*: indica si el objeto está debajo de otro
- *android:layout_toRightOf* y *android:layout_alignTop*: indican si están varios elementos en la misma fila sobre qué objeto se coloca a su derecha y su posicionamiento en altura
- *android:text* es el texto por defecto en cada elemento
- *android:onClick* es el nombre del método público a ejecutar al pulsar ese botón. Este método debe ser obligatoriamente público y tener como parámetro de entrada la vista, ejemplo public void function cSumar(View view) { ... }.

ahora vamos a programar el funcionamiento de los objetos descritos en el archivo de extensión java creado para el proyecto:

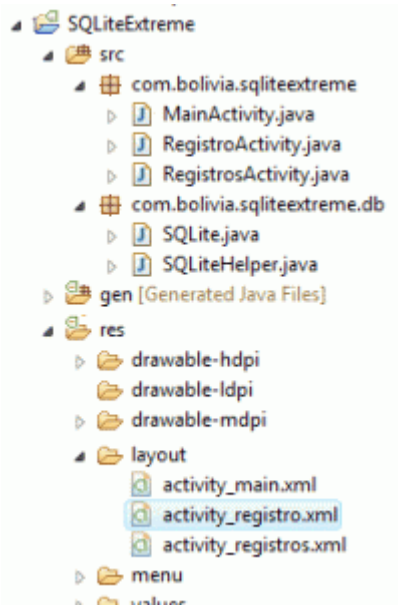
```
package com.android.calculadora;

import android.app.Activity;
import android.os.Bundle;
import android.widget.*;
import android.view.*;

public class Calculadora extends Activity {

    // Instancias de objetos a usar
    private double valor_a, valor_b;
    private EditText op_a, op_b;
    private TextView resultado;

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

        // Asignamos los objetos
        this.op_a = (EditText) findViewById(R.id.op_a);
        this.op_b = (EditText) findViewById(R.id.op_b);
        this.resultado = (TextView)
findViewById(R.id.resultado);
    }

    public void cSumar(View view) {
        if(this.op_a.getText().toString().length() > 0 &&
this.op_b.getText().toString().length() > 0) {
            this.valor_a =
```

```
Double.parseDouble(this.op_a.getText().toString());
            this.valor_b =
Double.parseDouble(this.op_b.getText().toString());
            this.resultado.setText(Double.toString((this.valor_a +
this.valor_b)));
        }
    }

    public void cRestar(View view) {
        if(this.op_a.getText().toString().length() > 0 &&
this.op_b.getText().toString().length() > 0) {
            this.valor_a =
Double.parseDouble(this.op_a.getText().toString());
            this.valor_b =
Double.parseDouble(this.op_b.getText().toString());
            this.resultado.setText(Double.toString((this.valor_a -
this.valor_b)));
        }
    }

    public void cMultiplicar(View view) {
        if(this.op_a.getText().toString().length() > 0 &&
this.op_b.getText().toString().length() > 0) {
            this.valor_a =
Double.parseDouble(this.op_a.getText().toString());
            this.valor_b =
Double.parseDouble(this.op_b.getText().toString());
            this.resultado.setText(Double.toString((this.valor_a *
this.valor_b)));
        }
    }

    public void cDividir(View view) {
        if(this.op_a.getText().toString().length() > 0 &&
this.op_b.getText().toString().length() > 0) {
            this.valor_a =
Double.parseDouble(this.op_a.getText().toString());
            this.valor_b =
Double.parseDouble(this.op_b.getText().toString());
            if(this.valor_b != 0) {
                this.resultado.setText(Double.toString((this.valor_a
/ this.valor_b)));
            }
            else {
                this.resultado.setText("Infinito");
            }
        }
    }
}
```

El método onCreate se ejecuta al crear la aplicación y es donde asignamos los objetos declarados en el XML a objetos Java. Tras ello declaramos las funciones definidas en los atributos onClick de cada botón.

Los programadores android un ejemplo sencillo de una aplicación que hace uso de **base de datos *SQLite***. En este proyecto, se cuenta con las opciones, de *inserción*, *consulta* y *eliminación* de registros, ademas se hace uso de paso de parámetros entre activities, llenado de registros en un ***ListView***, ***Toast***, **DatePicker** y **Dialog**.

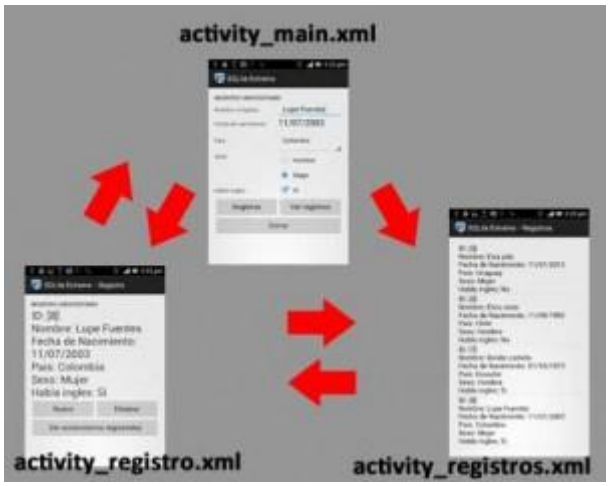
Esta desarrollado bajo Eclipse Indigo



La aplicación, hace uso de una sola tabla **UNIVERSITARIO** que consta de 6 campos, donde la llave primaria es autoincrementable de tipo entero.

```
CREATE TABLE "Universitario" (
                    "id" INTEGER PRIMARY KEY  AUTOINCREMENT  NOT NULL
UNIQUE ,
                    "Nombre" TEXT, "FechaNac" DATETIME, "Pais" TEXT,
"Sexo" TEXT, "Ingles" TEXT )
```

El proyecto hace uso de tres *layout* siendo **activity_main** el primero en mostrarse y donde se registran nuevos alumnos, si el registro tuvo éxito, se mostrara el layout **activity_registro**, este *layout* permite volver al layout anterior para agregar nuevos registros o eliminar el registro que este visible, también cuenta con un botón para ver la lista de registros en el *layout* **activity_registros** donde se llenaran en un ListView, cuando se realice un clic en cualquier item, se mostraran sus datos en el layout activity_registro.



Para una mejor compresión del proyecto, este esta basado en código de post anteriores:

- **Paso de parámetros entre dos Activity**

- **Introducción a SQLite**

- **SQLite INSERT, UPDATE, DELETE, QUERY**

Descarga el proyecto **AQUI**

Servicios avanzados - Ejercicios

Servicio reproductor de música

Vamos a crear un servicio que inicie la reproducción de un recurso audio al arrancarse, y que detenga la reproducción al pararse.

- Descargad las plantillas de la sesión. En el proyecto `ServicioMusica` tenemos una actividad principal que muestra un botón Start y un botón Stop. En sus respectivos `OnClickListener`'s tendremos que iniciar y parar el servicio con los métodos `startService(...)` y `stopService(...)`, pasándoles en ambos casos un `new Intent(main, MiAudioServicio.class)` como parámetro. Pero para ello tendremos que crear antes la clase que define el servicio:
- Creamos una nueva clase Java que se llame `MiAudioServicio` y sobrecargamos los métodos `onStartCommand`, `onCreate`, `onDestroy` y `onBind`, ayudándonos de las herramientas que proporciona Eclipse.
- Declaramos un campo `private MediaPlayer mediaPlayer;` en la clase del servicio.
- Cuando iniciemos el servicio desde la actividad, primero se creará y se invocará al método `onCreate(...)`. En él crearemos el reproductor:
- `Toast.makeText(this,"Servicio creado ...",`
- `Toast.LENGTH_LONG).show();`
- `mediaPlayer = MediaPlayer.create(getApplicationContext(),`
- `R.raw.ubuntu);`
- `mediaPlayer.setLooping(true);`

  mostrando un `Toast` para quedarnos tranquilos de que el servicio se ha iniciado. El recurso R.raw.ubuntu es un archivo .ogg que se incluye en la carpeta `res/raw` de las plantillas del proyecto. También podía haber sido un mp3.

- Una vez creado, se ejecutará el método `onStartCommand(...)`. En él iniciaremos la reproducción y devolveremos el valor `Service.START_STICKY`.
- `mediaPlayer.start();`
- `return Service.START_STICKY;`
- Finalmente, al destruir el servicio, detendremos la reproducción y mostraremos un `Toast`:
- `Toast.makeText(this,"onDestroy: Servicio destruido.",`
- `Toast.LENGTH_LONG).show();`
- `mediaPlayer.stop();`
- En cuanto al método `onBind`, devolveremos `null`, que indica que el servicio no tiene definido un interfaz AIDL para comunicarse con otros.

- Para que el servicio funcione en la aplicación, habrá que declararlo en el `AndroidManifest.xml`, dentro de `application`:
- `...`
- `<service android:enabled="true"`
- `android:name=".MiAudioServicio"/>`
- `</application>`

Si todo ha ido bien, y si hemos implementado los listeners de los botones que inician y detienen el servicio, debería funcionar. Probad iniciar el servicio y salir de la aplicación, entrar en otras, etc. El sonido seguirá reproduciéndose. Para detenerlo, volvemos a abrir la aplicación y lo detenemos.

Servicio con proceso en background. Contador

Los servicios se utilizan para ejecutar algún tipo de procesamiento en background. En el anterior ejercicio utilizamos el reproductor del sistema y simplemente le indicamos cuándo iniciarse y cuándo detenerse. En este ejercicio vamos a crear nuestro propio proceso que ejecute determinada tarea, en este caso, que vaya contando desde 1 hasta 100, deteniéndose 5 segundos antes de cada incremento. En cada incremento mostraremos un `Toast` que nos informe de la cuenta.

En las plantillas tenemos el proyecto `ServicioContador` que ya incluye la declaración del servicio en el manifest, la actividad que inicia y detiene el servicio, y el esqueleto del servicio `MiCuentaServicio`.

- En el esqueleto que se proporciona, viene definida una extensión de `AsyncTask` llamada `MiTarea`. Los métodos `onPreExecute`, `doInBackground`, `onProgressUpdate` y `onCancelled` están sobrecargados pero están vacíos. Se pide implementarlos, el primero de ellos inicializando el campo `i` que se utiliza para la cuenta, el segundo ejecutando un bucle desde 1 hasta 100, y en cada iteración pidiendo mostrar el progreso y durmiento después 5 segundos con `Thread.sleep(5000)`. El tercer método, `onProgressUpdate` mostrará el `Toast` con el progreso, y por último el método de cancelación pondrá el valor máximo de la cuenta para que se salga del bucle.
- En los métodos del servicio, `onCreate`, `onStartCommand` y `onDestroy`, introduciremos la creacion de la nueva `MiTarea`, su ejecución (método `execute()` de la tarea) y la cancelación de su ejecución (método `cancel()` de la tarea).

Una vez más, el servicio deberá seguir funcionando aunque se salga de la aplicación y podrá ser parado entrando de nuevo en la aplicación y pulsando Stop.

Servicio con notificaciones. Números primos

Este ejercicio es una extensión del anterior, pero vamos a utilizar un nuevo proyecto plantilla, el `ServicioNotificaciones`. En lugar de mostrar cualquier número de la cuenta, vamos a mostrarlos sólo si son primos. Además, en lugar de mostrar un `Toast`, vamos a mostrar una `Notification` que aparecerá en la barra de tareas y se actualizará con la llegada de cada nuevo número. Si salimos de la aplicación sin parar el servicio, seguirán apareciendo notificaciones, y si pulsamos sobre la notificación, volverá a lanzar la actividad, cerrándose la notificación que hemos pulsado.

- Dentro del servicio `MiNumerosPrimosServicio` se encuentra declarada la `AsyncTask` llamada `MiTarea`. En ella tenemos como campos de la clase una `Notification` y un `NotificationManager`. Hay que darles valores en el método `onPreExecute()`.
- El método `doInBackground(...)` ejecutará un bucle que irá incrementando `i` mientras su valor sea menor de `MAXCOUNT`. En cada iteración, si el número es primo (función incluida en la plantilla), pedirá que se muestre el progreso, pasándole como parámetro el nuevo primo encontrado.

- Implementar el método `onProgressUpdate(...)` para que muestre la notificación. Para ello habrá que actualizar la notificación con el método `setLatestEventInfo`, al cuál le pasaremos en un String la información del último primo descubierto y le pasaremos un `PendingIntent` para que al pulsar sobre la notificación, nos devuelva a la actividad de la aplicación, por si la hemos cerrado. Para crear el PendingIntent utilizaremos el método `PendingIntent.getActivity(...)` al cuál le tenemos que pasar un `new Intent(getApplicationContext(),Main.class)`.
- La aplicación debería funcionar en este punto, mostrando las notificaciones y relanzando la aplicación si son pulsadas, pero no cerrándolas al pulsarlas. Para ello simplemente tenemos que llamar al método `cancel(id)` del `notificationManager` y pasarle la constante NOTIF_ID para que la notificación no se muestre como una nueva, sino como actualización de la que ya habíamos puesto. Una manera de hacerlo es en un método estático del `MiNumerosPrimosServicio`, que podemos llamar `cerrarMiNotificacion(NotificationManager nm)`. Este método será invocado desde el `Main.onResume()`.

IP AppWidget

En programación de Android se denomina `Widget` a los componentes de alto nivel de la interfaz de usuario, y `AppWidgets` a los widgets que se pueden añadir al escritorio del sistema operativo, como el reloj, pequeños controles, etc.

Vamos crear un proyecto `AppWidget` para construir un AppWidget de Android, que nos muestre en todo momento la IP que el dispositivo está usando en este momento. No necesitaremos ninguna actividad, así que podemos desmarcar la casilla "Create

activity", o bien eliminar la actividad después (no sólo la clase, sino también la declaración en el manifest).

En el proyecto pulsamos con el boton derecho y añadimos un nuevo Android XML File, de tipo AppWidget Provider, que se llame `miwidget.xml`. El editor nos permite pulsar sobre el AppWidget Provider y editar sus atributos. Ponemos los siguientes:

```
android:minWidth="146dip"
android:minHeight="72dip"
android:updatePeriodMillis="600000"
android:initialLayout="@layout/miwidget_layout"
```

El miwidget_layout lo tenemos que crear, o dará error. Así que creamos un nuevo Android XML File de tipo Layout llamado `miwidget_layout.xml` y le añadimos un campo de texto `TextView` con el texto vacío.

Creamos una clase `MiWidget` que herede de `AppWidgetProvider`, en el paquete `es.ua.jtech.daa.appwidget`. Sobrecargamos su método `onUpdate(...)` y actualizamos en él el campo de texto, usando `RemoteViews` y pasándoselos al `AppWidgetManager`:

```
RemoteViews updateViews = new RemoteViews(context.getPackageName(),
                                          R.layout.miwidget_layout);
updateViews.setTextViewText(R.id.TextView01,"Hola");
ComponentName thisWidget = new ComponentName(context, MiWidget.class);
AppWidgetManager.getInstance(context).updateAppWidget(thisWidget,
updateViews);
```

Antes de probar el widget hay que declararlo en el `AndroidManifest.xml`, dentro de `application`:

```
    <receiver android:name=".MiWidget" android:label="Mi Widget">
        <intent-filter>
            <action
android:name="android.appwidget.action.APPWIDGET_UPDATE" />
        </intent-filter>
        <meta-data android:name="android.appwidget.provider"
            android:resource="@xml/miwidget" />
    </receiver>
```

Ejecutamos el widget desde Eclipse, como aplicación android, y comprobamos que no ocurra ningún error en la consola de Eclipse. Ya se puede añadir el widget en el escritorio, efectuando una pulsación larga sobre una porción de área libre del escritorio, y seleccionando nuestro widget.

Si todo funciona correctamente, vamos a implementar en el `MiWidget` un servicio `UpdateService` que realizará la actualización del widget, evitando así bloqueos debidos a la velocidad de la red. El servicio recogerá la información que le devuelve en texto plano la página http://www.whatismyip.org y la mostrará en el campo de texto del widget.

Instrucciones para programar el servicio que se pide:

- Creamos la clase `public static class UpdateService extends Service` dentro de la clase `MiWidget` y sobrecargamos los métodos `onBind` (que es

obligatorio, pero devolverá `null`) y `onStartCommand` que devolverá `Service.START_STICKY`.

- Hay que declarar el servicio en el `AndroidManifest.xml`, dentro de `application`, con:
- `<service android:name=".MiWidget$UpdateService" />`
- Del método `MiWidget.onUpdate(...)` podemos cortar todas las líneas y sustituirlas por la llamada al servicio:
- `context.startService(new Intent(context, UpdateService.class));`
- En el método `onStartCommand` del servicio, pegaremos las líneas que actualizan los RemoteViews, pero las modificaremos para que obtengan el contexto y el paquete del widget, quedando el método así:

```
@Override
public int onStartCommand(Intent intent, int flags, int startId)
{
  RemoteViews updateViews = new RemoteViews(getPackageName(),
          R.layout.miwidget_layout);
  updateViews.setTextViewText(R.id.TextView01,"Hola Serv");
  ComponentName thisWidget = new ComponentName(this,
MiWidget.class);
  AppWidgetManager.getInstance(this).updateAppWidget(thisWidget,
          updateViews);
  return Service.START_STICKY;
}
```

Ahora podemos volver a probar el widget, ejecutándolo desde Eclipse. Si funciona, podemos pasar a sustituir la línea

```
updateViews.setTextViewText(R.id.TextView01,"Hola Serv");
```

por el código que accede a la URL por HTTP, obteniendo un InputStream y convirtiendo los bytes a String para mostrarlo:

```
String ipstring = "Unknown IP";

try {
    URL url = new URL("http://icanhazip.com/");
    HttpURLConnection http = (HttpURLConnection)url.openConnection();
    InputStream is = http.getInputStream();
    byte[] buffer = new byte[20];
    is.read(buffer, 0, 20);
    ipstring = "IP: "+new String(buffer);
} catch (MalformedURLException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
}

updateViews.setTextViewText(R.id.TextView01,ipstring);
```

Antes de probarlo hay que añadir el permiso de Internet en el `AndroidManifest.xml`, fuera de `application`:

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

Ejecutamos y observamos el resultado:

Se puede añadir un comportamiento al pulsar sobre algún componente del widget. Por ejemplo, para que se abra un navegador con la web consultada, añadiríamos las siguientes líneas para actualizar el `updateViews`:

```
Intent defineIntent = new Intent(Intent.ACTION_VIEW,
    Uri.parse("www.whatismyip.org"));
PendingIntent pendingIntent = PendingIntent.getActivity(
    getApplicationContext(), 0, defineIntent, 0);
```

```
updateViews.setOnClickPendingIntent(R.id.miwidgetlayout,
pendingIntent);
```

Para que la referencia al recurso `R.id.miwidgetlayout` funcione, se tiene que definir el atributo `android:id="@+id/miwidgetlayout"` del `LinearLayout` del widget, que se encuentra en el archivo `miwidget_layout.xml`.

Intefaz de usuario - Ejercicios

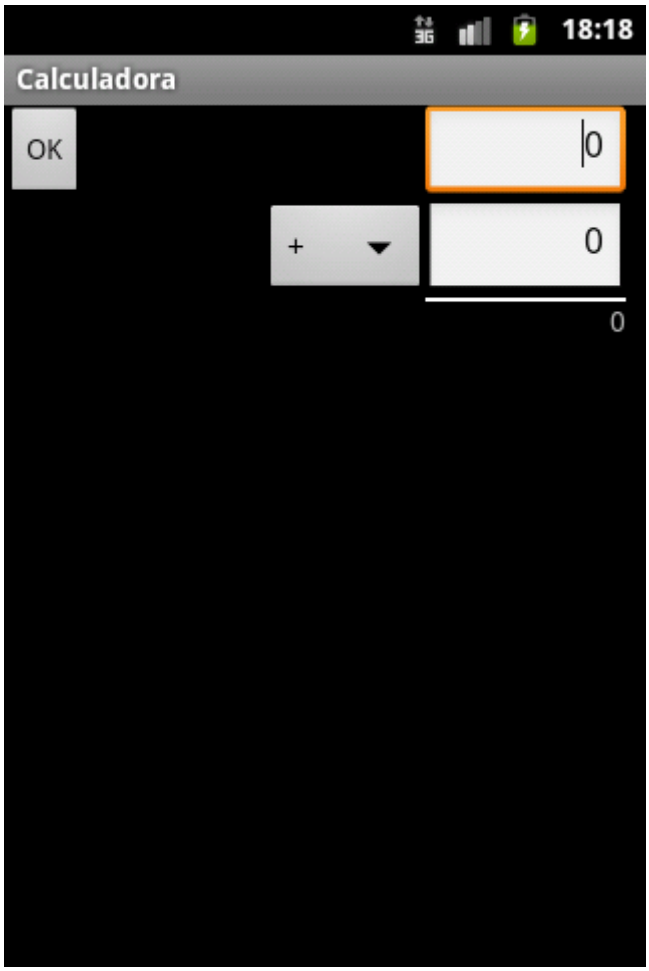
LinearLayout

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

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

Calculadora sencilla

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



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

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

Ciudades

En este ejercicio practicaremos con los elementos de tipo `Spinner`. Crea una nueva aplicación llamada *Ciudades* con una única actividad. La interfaz de dicha actividad estará compuesta por un `TextView` con `android:id="@+id/texto"` y dos elementos de tipo `Spinner` con identificadores `android:id="@+id/paises"` y

`android:id="@+id/ciudades"`. El primero de ellos permitirá escoger entre tres países cualquiera (inicialmente ningún país estará seleccionado). El segundo permitirá escoger una ciudad según el país seleccionado en el anterior. Cada vez que se seleccione un país en el primer `Spinner` deberán cambiar las opciones del segundo, mostrando dos ciudades del país seleccionado.

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

Para completar el ejercicio debes seguir los siguientes pasos:

- Añade el `TextView` y los dos `Spinner` al recurso layout de la aplicación, sin olvidar añadir a estos dos últimos su correspondiente atributo `android:prompt` (con los textos "Selecciona país" y "Selecciona ciudad" respectivamente).
- Crea las opciones para los `Spinner` en el archivo *arrays.xml* de los recursos con el siguiente contenido:

```
<resources>
    <string-array name="paises">
        <item>España</item>
        <item>Alemania</item>
        <item>Francia</item>
    </string-array>

    <string-array name="ciudadesespana">
        <item>Madrid</item>
        <item>Barcelona</item>
    </string-array>

    <string-array name="ciudadesalemania">
        <item>Berlin</item>
        <item>Wacken</item>
    </string-array>

    <string-array name="ciudadesfrancia">
        <item>París</item>
        <item>Marsella</item>
    </string-array>
</resources>
```

- Completa las opciones correspondientes del primer `Spinner` para que cargue el array de países.
- Rellena el segundo `Spinner` con las ciudades correspondintes al primer país. Esto debes hacerlo así porque siempre que inicies la actividad será el primer país el que se encuentre seleccionado.
- Asígnale al TextView como valor inicial el nombre de la primera ciudad, pues será la que se encontrará seleccionada al iniciar la actividad.
- Crea un evento para el `Spinner` de paises para que cada vez que se seleccione una opción se muestren las opciones adecuadas en el `Spinner` de ciudades.
- Añade un manejador al `Spinner` de ciudades para que cada vez que se seleccione una opción se muestre en el `TextView`.

Para obtener el texto correspondiente a la opción seleccionada en el `Spinner` puedes utilizar el método `getSelectedItem` del mismo. Una vez hecho esto puedes llamar al método `toString` para obtener la cadena correspondiente.

Abrir actividades

En este ejercicio vamos a crear una aplicación llamada *Activities* para navegar entre tres actividades. Las tres actividades tendrán de nombre *Actividad1*, *Actividad2* y *Actividad3*. La interfaz gráfica de cada una de ellas consistirá únicamente en un `TextView` mostrando su nombre y dos botones para cambiar a las otras dos actividades posibles. Los botones de cada actividad serán distintos: en la actividad 1 tendremos botones para ir a la actividad 2 y 3, en la 2 para ir a la 1 y la 3, y en la 3 para ir a la 1 y la 2.

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

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

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