

TABLA DE CONTENIDOS

| | |
|--|-----------|
| ALMACENAMIENTO DE DATOS | 2 |
| 1 BASES DE DATOS..... | 2 |
| 1.1. Creación de la base de datos..... | 3 |
| 1.2. Manipulación de datos (Insertar, actualizar y eliminar) | 5 |
| 1.3. Consulta y recuperación de registros..... | 6 |
| 1.4. Ejercicio 1 | 7 |
| 1.5. Ejercicio Guiado 1 | 8 |
| 1.6. Ejercicio Guiado 2 | 10 |
| 1.7. Ejercicio 2 | 14 |
| 2. CONTENT PROVIDERS..... | 15 |
| 2.1. Crear un Content Provider | 15 |
| 2.1.1. Definir estructura de almacenamiento..... | 16 |
| 2.1.2. Crear la clase extendiendo ContentProvider | 17 |
| 2.1.3. Declarar el Content Provider en AndroidManifest.xml..... | 20 |
| 2.1.4. Utilización de Content Providers..... | 20 |

ALMACENAMIENTO DE DATOS

En Android tenemos varias posibilidades para almacenar información de una forma permanente:

- Mediante **ficheros almacenados** en diferentes lugares:
 - En la memoria interna del teléfono.
 - En la tarjeta SD.
 - En los recursos. Estos sólo podrán ser leídos por lo que no son útiles para almacenar información desde la aplicación.
- Mediante el **uso de XML** para almacenar la información de manera estructurada usando dos herramientas alternativas:
 - Las librerías SAX (Java's Simple API for XML)
 - Las librerías DOM (Document Object Model)
- Mediante **bases de datos**. Android incorpora la librería SQLite, que nos permitirá crear y manipular nuestras propias bases de datos de forma muy sencilla.
- Mediante la **clases ContentProvider**. Consiste en un mecanismo introducido en Android para poder compartir datos entre aplicaciones.

En este capítulo nos vamos a centrar en las dos herramientas principales y diferentes a otras plataformas que proporciona Android para almacenar y consultar datos estructurados. Por un lado las bases de datos SQLite y por otro lado los Content Providers.

Las otras dos posibilidades (ficheros y uso de XML) las dejaré al finalizar el curso en un capítulo anexo con aquellas cuestiones que se nos queden en el tintero por falta de tiempo.

1 BASES DE DATOS

Android incorpora la librería SQLite que nos permite utilizar bases de datos mediante el lenguaje SQL, de una forma sencilla y utilizando muy pocos recursos del sistema.

Para crear una base de datos en Android utilizamos la clase SQLiteOpenHelper que permite la creación de la base de datos y el trabajo con futuras versiones de la misma. Lo que se debe hacer es crear una subclase que debe implementar los métodos onCreate(), onUpgrade() y opcionalmente onOpen().

La clase SQLiteOpenHelper se encarga de abrir la base de datos o de crearla si no existe. También se encarga de actualizar la versión de la base de datos si decidimos crear una nueva estructura de la misma. La clase SQLiteOpenHelper también tiene dos métodos getReadableDatabase() y getWritableDatabase() que abren la base de datos en modo sólo lectura o lectura y escritura.

1.1. Creación de la base de datos

Vamos a crear como ejemplo una base de datos muy sencilla que se llamará DBClientes, que solo contendrá una tabla llamada Clientes que contendrá así mismo los campos nombre y teléfono.

Crearemos por tanto una nueva aplicación que llamaremos **EjemploBaseDatos** (y que os paso en fichero de Códigos Fuente) donde lo primero será crear una clase derivada de SQLiteOpenHelper que llamaremos ClientesSQLiteHelper, donde sobreescribiremos los métodos onCreate() y onUpgrade() para adaptarlos a la estructura de datos indicada.

El código de esta clase sería el siguiente:

```
package org.ejemplo.aguilar;

import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteDatabase.CursorFactory;
import android.database.sqlite.SQLiteOpenHelper;

public class ClientesSQLiteHelper extends SQLiteOpenHelper {
    //Cadena con la sentencia SQL que permite crear la tabla Clientes
    String cadSQL = "CREATE TABLE Clientes (codigo INTEGER, nombre TEXT, telefono TEXT)";

    public ClientesSQLiteHelper(Context contexto, String nombre, CursorFactory almacen, int version){
        super(contexto, nombre, almacen, version);
    }

    //Este método se ejecuta automáticamente cuando sea necesaria la creación de la base de datos.
    //Es decir, se ejecutará cuando la base de datos todavía no exista.
    //Aquí deben crearse todas las tablas necesarias e insertar los datos iniciales si es necesario.
    @Override
    public void onCreate(SQLiteDatabase bd) {
        //Ejecutamos la sentencia SQL para crear la tabla Clientes
        //El método execSQL se limita a ejecutar directamente el código SQL que le pasemos.
        bd.execSQL(cadSQL);
    }

    //Este método se lanza automáticamente cuando es necesaria una actualización de la estructura
    //de la base de datos o una conversión de los datos.
    @Override
    public void onUpgrade(SQLiteDatabase bd, int versionAnterior, int versionNueva) {
        //NOTA: Para simplificar este ejemplo eliminamos la tabla anterior y la creamos de nuevo
        // con el nuevo formato.
        // Lo normal sería realizar una migración o traspaso de los datos de la tabla antigua
        // a la nueva, con la consiguiente complicación que ello conlleva.

        //Eliminamos la versión anterior de la tabla
        bd.execSQL("DROP TABLE IF EXISTS Clientes");

        //Creamos la nueva versión de la tabla
        bd.execSQL(cadSQL);
    }
}
```

Una vez que hemos definido la clase helper, abrir la base de datos desde nuestra aplicación es muy simple. Tan sólo será necesario crear un objeto de la clase ClientesSQLiteHelper pasándole los siguientes parámetros:

- El contexto de la aplicación. En nuestro ejemplo una referencia a la actividad principal.
- El nombre de la base de datos.
- Un objeto CursorFactory que normalmente no necesitaremos (pasamos null).
- La versión de la base de datos que necesitamos.

Únicamente creando este objeto se pueden producir las siguientes acciones:

- Si la base de datos ya existe y su versión actual coincide con la que se pide se realiza la conexión con ella.
- Si la base de datos ya existe pero su versión actual es anterior a la solicitada, se llama de forma automática al método `onUpgrade()` para convertir la base de datos a la nueva versión y se realiza la conexión con la base de datos convertida.
- Si la base de datos no existe, se llama de forma automática al método `onCreate()` para crearla y se realiza la conexión la base de datos creada.

Una vez que tenemos el objeto `CientesSQLiteHelper`, llamaremos al método `getReadableDatabase()` o al método `getWritableDatabase()` para obtener una referencia a la base de datos que nos permita sólo consultarla o realizar también modificaciones respectivamente.

Una vez que tenemos la referencia a la base de datos ya podemos realizar las acciones que necesitemos. En nuestro caso insertaremos 3 registros de prueba y al final cerraremos la conexión con la base de datos llamando al método `close()`.

El código de la clase en la que haremos esto con la actividad principal sería el siguiente:

```
package org.ejemplo.aguilar;

import android.app.Activity;

public class EjemploBaseDatos extends Activity {

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

        //Abrimos la base de datos en modo escritura
        CientesSQLiteHelper cliBDh = new CientesSQLiteHelper(this, "DBClientes", null, 1);

        //Obtenemos referencia a la base de datos para poder modificarla.
        SQLiteDatabase bd = cliBDh.getWritableDatabase();

        //En caso de abrir de forma correcta la base de datos
        if (bd!=null) {
            //Introducimos 3 clientes de ejemplo
            for (int cont=1; cont<=3; cont++) {
                //Creamos los datos
                int codigo = cont;
                String nombre = "Cliente" + cont;
                String telefono = cont + "XXXXXXX";

                //Introducimos los datos en la tabla Clientes
                bd.execSQL("INSERT INTO Clientes (codigo, nombre, telefono) " +
                    "VALUES (" + codigo + ", '" + nombre + "', '" + telefono + "')");
            }

            //Cerramos la base de datos
            bd.close();
        }
    }
}
```

1.2. Manipulación de datos (Insertar, actualizar y eliminar)

En la API de SQLite de Android podemos encontrar dos formas de realizar operaciones sobre la base de datos que no produzcan resultados (insertar, actualizar y eliminar registros, crear tablas, crear índices, etc.):

1. Utilizar el método `execSQL()` de la clase `SQLiteDatabase`. Este método permite ejecutar cualquier sentencia sobre la base de datos siempre que ésta no devuelva resultados.

```
//Insertar un registro
bd.execSQL("INSERT INTO Clientes (cliente, telefono) VALUES ('cli1','11111') ");

//Actualizar un registro
bd.execSQL("UPDATE Clientes SET telefono='00000' WHERE cliente='cli1' ");

//Eliminar un registro
bd.execSQL("DELETE FROM Clientes WHERE cliente='cli1' ");
```

2. Utilizar los métodos `insert()`, `update()` y `delete()` que también proporciona la clase `SQLiteDatabase`.

- a. Ejemplo de utilización del método `insert()`

```
//Creamos el registro que queremos insertar utilizando un objeto ContentValues
ContentValues nuevoRegistro = new ContentValues();
nuevoRegistro.put("cliente","cli10");
nuevoRegistro.put("telefono", "101010");

//Insertamos el registro en la base de datos
//El primer parámetro es el nombre de la tabla donde insertaremos.
//El segundo parámetro lo obviemos ya que solo sirve en caso de querer introducir
//un registro vacío.
//El tercer parámetro es el objeto ContentValues que contiene el registro a insertar
bd.insert("Clientes", null, nuevoRegistro);
```

- b. Ejemplo utilización del método `update()`

```
//Establecemos los campos-valores que vamos a actualizar
ContentValues valores = new ContentValues();
valores.put("telefono", "101010XX");

//Actualizamos el registro en la base de datos
//El tercer argumento es la condición del UPDATE tal como lo haríamos en la cláusula
//WHERE en una sentencia SQL normal.
//El cuarto argumento son partes variables de la sentencia SQL que aportamos en un
//vector de valores aparte
String[] args = new String[]{"cli1", "cli2"};
bd.update("Clientes", valores, "cliente=? OR cliente=?", args);
```

- c. Ejemplo utilización del método `delete()`

```
//Eliminamos el registro del cliente 'cli2'
String[] args2 = new String[]{"cli2"};
bd.delete("Clientes", "usuario=?", args2);
```

1.3. Consulta y recuperación de registros

Tenemos dos opciones para recuperar registros de una base de datos SQLite en Android:

1. Utilizando directamente un comando de selección SQL.
 - a. Utilizamos el método `rawQuery()` de la clase `SQLiteDatabase`.
 - b. Este método recibe directamente como parámetro un comando SQL completo, donde indicamos los campos a recuperar y los criterios de selección.
 - c. El resultado de la consulta lo obtenemos en forma de cursor que después podemos recorrer para procesar los registros recuperados.

```
String[] args = new String[]{"cli1"};
Cursor c = bd.rawQuery("SELECT cliente,telefono FROM Clientes WHERE cliente=? ", args);
```

2. Utilizando un método específico donde parametrizamos la consulta a la base de datos.
 - a. Utilizamos el método `query()` de la clase `SQLiteDatabase`.
 - b. Este método recibe varios parámetros: el nombre de la tabla, un vector con los nombres de campos a recuperar, la cláusula `WHERE`, un vector con los argumentos variables incluidos en el `WHERE` (si los hay, null en caso contrario), la cláusula `GROUP BY` si existe, la cláusula `HAVING` si existe, y por último la cláusula `ORDER BY` si existe.
 - c. Opcionalmente se puede incluir un parámetro más al final donde indicamos el número máximo de registros que queremos obtener.

```
//Ejemplo Select2
String[] campos = new String[] {"cliente", "telefono"};
String[] args = new String[] {"cli1"};
Cursor c = bd.query("Clientes", campos, "usuario=?", args, null, null, null);
```

Para recorrer el cursor podemos utilizar los siguientes métodos de la clase `Cursor`:

- `moveToFirst()`: Mueve el puntero del cursor al primer registro devuelto.
- `moveToNext()`: Mueve el puntero del cursor al siguiente registro devuelto.

Una vez posicionados en cada registro podemos utilizar cualquiera de los métodos `getXXX(índice_columna)` existente para cada tipo de dato para recuperar el dato de cada campo del registro actual del cursor.

En el siguiente código tenemos un ejemplo:

```
String[] campos = new String[] {"nombre", "telefono"};
String[] args = new String[] {"cli1"};
Cursor c = bd.query("Clientes", campos, "nombre=?", args, null, null, null);
//Nos aseguramos de que exista al menos un registro
if (c.moveToFirst()) {
    //Recorremos el cursor hasta que no haya más registros
    do {
        String nombreCli = c.getString(0);
        String telefonoCli = c.getString(1);
    } while (c.moveToNext());
}
```

Para comprobar la creación y el contenido de las tablas creadas en alguno de los ejemplos realizados se puede seguir el siguiente procedimiento una vez se haya ejecutado el proyecto:

1. La base de datos se habrá creado en el directorio:
/data/data/com.slashmobility.curso.android/databases
2. Para comprobarlo podemos ejecutar lo siguiente para entrar como root:
adb -s emulator-5554 shell
3. Ejecutar lo siguiente para abrir la base de datos:
>sqlite3 /data/data/com.slashmobility.curso.android/databases/mibasededatos
4. Una vez en este modo ya podemos ejecutar sentencias SQL:
>SELECT * from ejemplo;

1.4. Ejercicio 1

Se trata de utilizar una base de datos para guardar las puntuaciones obtenidas en nuestra aplicación Solobici Didáctico.

En primer lugar crearemos una interfaz para tener métodos en común y así poder cambiar más adelante el método de almacenamiento de una forma sencilla.

La interfaz se denominará “AlmacenPuntuaciones” y la añadiremos al proyecto con el fichero “AlmacenPuntuaciones.java” que tendrá el siguiente código:

```
package juego.solobici;

import java.util.Vector;

public interface AlmacenPuntuaciones {
    public void guardarPuntuacion(int puntos, String nombre, long fecha);

    public Vector<String> listaPuntuaciones(int cantidad);
}
```

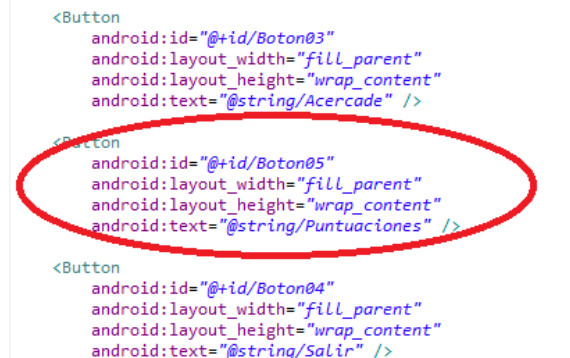
Realizar una nueva clase AlmacenPuntuacionesSQLite que implemente la interfaz anterior y que posea los siguientes métodos:

- a. Constructor.
- b. onCreate() donde crearemos la tabla “puntuaciones en la base de datos.
- c. onUpgrade() que estará vacío.
- d. guardarPuntuacion(puntos, nombre, fecha) que guardará dentro de la tabla un registro nuevo con los valores que le pasemos.
- e. listaPuntuaciones(cantidad) que devuelva un Vector<String> con un número “cantidad” de registros correspondientes a los primeros de la lista de puntuaciones.

1.5. Ejercicio Guiado 1

Una vez realizada la clase AlmacenPuntuacionesSQLite lo que vamos a hacer es utilizarla en nuestra aplicación. Para ello vamos a seguir los siguientes pasos:

1. Añadir un nuevo botón antes del botón “Salir” con el texto Puntuaciones.



```
<Button
    android:id="@+id/Boton03"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/Acercade" />

<Button
    android:id="@+id/Boton05"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/Puntuaciones" />

<Button
    android:id="@+id/Boton04"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/Salir" />
```

2. En la actividad Solobici incluimos un escuchador asociado al botón “Puntuaciones” que llame al siguiente método:

```
public void lanzarPuntuaciones(){
    Intent i = new Intent(this, Puntuaciones.class);
    startActivity(i);
}
```

3. En la actividad Solobici declaramos un atributo para almacenar las puntuaciones.

```
//Variable para almacenar las puntuaciones
public static AlmacenPuntuaciones almacen;
```

y en el constructor lo inicializamos:

```
//Almacén de datos
almacen = new AlmacenPuntuacionesSQLite(this);
```

4. Creamos un layout para la nueva actividad que llamaremos puntuaciones.xml y que nos servirá para mostrar la lista de puntuaciones del juego.

```
<?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"
    android:background="@drawable/degradado">

    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Puntuaciones"
        android:gravity="center"
        android:layout_margin="10px"
        android:textSize="10pt"/>
```



```

<FrameLayout
    android:layout_width="fill_parent"
    android:layout_height="0dip"
    android:layout_weight="1">
    <ListView
        android:id="@android:id/list"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:drawSelectorOnTop="false" />
    <TextView
        android:id="@android:id/empty"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:text="No hay puntuaciones" />
</FrameLayout>
</LinearLayout>

```

5. Creamos la actividad Puntuaciones con una lista para mostrar las puntuaciones.

```

package juego.solobici;

import android.app.ListActivity;

public class Puntuaciones extends ListActivity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.puntuaciones);
        setListAdapter(new ArrayAdapter<String>(this, android.R.layout.simple_list_item_1,
                                                Solobici.almacen.listaPuntuaciones(10)));
    }
}

```

6. Registramos la actividad Puntuaciones en AndroidManifest.xml

```

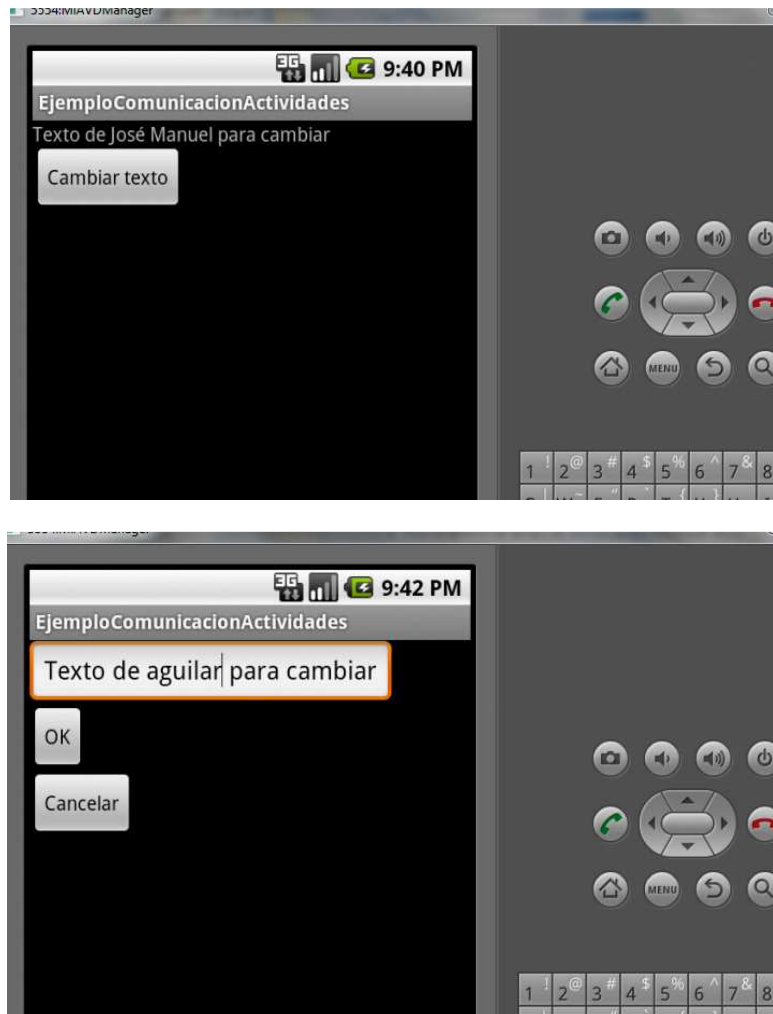
<activity android:name=".Puntuaciones"
    android:label="Puntuaciones"
    android:theme="@android:style/Theme.Dialog">
</activity>

```

En este momento nos interesaría comunicar información entre dos actividades. En concreto se trataría de pasar la puntuación desde la actividad "Juego" a la actividad "Solobici" que la llamó. Pero esto lo haréis vosotros como ejercicio una vez que veamos un ejemplo de cómo pasar información entre dos actividades.

1.6. Ejercicio Guiado 2

En el ejemplo que os paso (**EjemploComunicacionActividades**) conseguiremos tener dos ventanas que se pasan información.



En la actividad principal, la que se mostrará al principio de la ejecución de nuestra aplicación, tenemos dos elementos:

3. Un texto, en el que mostraremos el texto que nos será enviado como respuesta desde la subactividad.
4. Un botón, que lanza la subactividad.

En la subactividad, que será lanzada por el botón “Cambiar texto”, tenemos tres elementos:

5. Una caja de texto, en la que pondremos la cadena que queramos.
6. Un botón OK, que actualizará el texto de la actividad principal.
7. Un botón Cancelar, que no actualizará el texto de la actividad principal.

Recuerda que para que todo funcione bien, las dos actividades deben estar definidas en AndroidManifest.xml

El código de la actividad principal es el siguiente:

```
package org.ejemplo.aguilar;

import android.app.Activity;

public class EjemploComunicacionActividades extends Activity {
    private static final int TEXTO_ENVIADO = 0;
    private TextView t;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        t = (TextView) this.findViewById(R.id.textview);
        Button b = (Button) this.findViewById(R.id.botonCambiar);
        b.setOnClickListener(new OnClickListener() {

            @Override
            public void onClick(View v) {
                Intent i = new Intent(EjemploComunicacionActividades.this, SubActividad.class );
                i.putExtra("text", t.getText());
                EjemploComunicacionActividades.this.startActivityForResult(i, TEXTO_ENVIADO);
            }
        });
    }

    @Override
    protected void onActivityResult(int codigoEnviado, int codigoResultado, Intent datos) {
        if ( codigoEnviado == TEXTO_ENVIADO ){
            if ( codigoResultado == Activity.RESULT_OK ){
                t.setText(datos.getExtras().get("text").toString());
            }
        }
    }
}
```

El código de la subactividad es el siguiente:

```
package org.ejemplo.aguilar;

import android.app.Activity;

public class SubActividad extends Activity {
    private EditText textoEditable;

    public void onCreate( Bundle savedInstanceState ){
        super.onCreate(savedInstanceState);
        this.setContentView(R.layout.text);
        textoEditable = (EditText) this.findViewById(R.id.editar);
        textoEditable.setText(this.getIntent().getExtras().getString("text"));

        Button ok = (Button) this.findViewById(R.id.BotonOK);
        ok.setOnClickListener(new OnClickListener(){

            @Override
            public void onClick(View v) {
                Intent i = new Intent( SubActividad.this, SubActividad.class );
                i.putExtra("text", textoEditable.getText());
                setResult( Activity.RESULT_OK, i );
                SubActividad.this.finish();
            }
        });
    }
}
```

```

        Button cancel = (Button) this.findViewById(R.id.BotonCancelar);
        cancel.setOnClickListener(new OnClickListener(){

            @Override
            public void onClick(View v) {
                Intent i = new Intent( SubActividad.this, SubActividad.class );
                i.putExtra("text", textoEditable.getText());
                setResult( Activity.RESULT_CANCELED, i );
                SubActividad.this.finish();
            }

        });
    }
}

```

Veamos cada una de las partes de las que se compone el código que acabamos de presentar:

1. Lanzando la subactividad

Lo primero que debemos hacer es iniciar **SubActividad** desde **EjemploComunicacionActividades** cuando el botón “Cambiar texto” sea pulsado:

```

@Override
public void onClick(View v) {
    Intent i = new Intent(EjemploComunicacionActividades.this, SubActividad.class );
    i.putExtra("text", t.getText());
    EjemploComunicacionActividades.this.startActivityForResult(i, TEXTO_ENVIADO);
}

```

- a) Creamos un Intent que requiere como parámetros la actividad padre y la clase de la actividad a lanzar. ¿Qué es un Intent? *Algo que puede ser lanzado desde la aplicación padre*. En este caso, la actividad SubActividad.
- b) **i.putExtra("text", t.getText())**: Con esta acción estamos añadiendo datos a nuestro Intent. **Estos datos estarán disponibles en la subactividad. Con este paso creamos la comunicación entre actividad y subactividad.** En el ejemplo, añadimos un dato que tiene como identificador “text” y como valor el texto contenido por la TextView. Lo utilizaremos para inicializar la caja de texto con el texto actual.
- c) **startActivityForResult(i, TEXTO_ENVIADO)**: Lanzamos la actividad a ejecución. Como parámetros, el **Intent** ya definido y un número identificativo de la actividad. Este número nos servirá para procesar la respuesta de la actividad. Es importante usar esta función y no **startActivity**. Esta función es la que provoca que, cuando la subactividad acabe, se ejecute **onActivityResult**. Podemos utilizar **startActivity** cuando no esperamos ningún resultado de la actividad lanzada.

2) Procesando datos de la actividad padre en la subactividad

```

public void onCreate( Bundle savedInstanceState ){
    ...
    editText.setText(this getIntent().getExtras().getString("text"));
    ...
}

```

Con este código accedemos al valor que inicializamos en el paso anterior. Así actualizamos el valor de la caja de texto con lo que actualmente haya en el TextView de EjemploComunicacionActividades.

3) Devolviendo los datos a la actividad principal

Desde **SubActividad**, añadimos sendos listeners a los botones:

```
@Override
public void onClick(View v) {
    Intent i = new Intent( SubActividad.this, SubActividad.class );
    i.putExtra("text", textoEditable.getText());
    setResult( Activity.RESULT_OK, i );
    SubActividad.this.finish();
}
```

- a) Como queremos devolver información a la actividad padre, debemos crear un Intent para albergarla. Añadimos un dato con identificador "text" y cuyo valor es el texto del **textoEditable**.
- b) **setResult(...)**: A través de este método, indicamos el resultado de la actividad. Puede ser un valor predefinido, como **RESULT_OK** o **RESULT_CANCELED**, o alguno definido por el usuario. Este valor nos será devuelto a la hora de procesar el resultado de la actividad.
- c) **finish()**: Hemos hecho todo lo que teníamos que hacer, finalizamos la actividad y regresamos a la actividad padre.

4) Procesando el resultado de la subactividad

Procesamos el resultado en EjemploComunicacionActividades con:

```
@Override
protected void onActivityResult(int codigoEnviado, int codigoResultado, Intent datos) {
    if ( codigoEnviado == TEXTO_ENVIADO ){
        if ( codigoResultado == Activity.RESULT_OK ){
            t.setText(datos.getExtras().get("text").toString());
        }
    }
}
```

Comprobamos que el **codigoEnviado** es de la actividad adecuada, el código de resultado fue **RESULT_OK** y actualizamos con los datos recibidos.

1.7. Ejercicio 2

Bueno, ahora os toca a vosotros. Se trata de pasar la puntuación desde la actividad “Juego” a la actividad “Solobici” que la llamó. Los pasos que debéis seguir son los siguientes:

1. Crear una variable global en la clase VistaJuego que se llame “puntuación” y que inicializamos a cero. Cuando se destruye un coche lo que haremos es incrementar esta variable.
2. Preparamos la actividad Solobici para que al lanzar la actividad Juego se pueda recoger la puntuación obtenida. Con la puntuación obtenida llamaremos al método “guardarPuntuacion” de la clase AlmacenPuntuacionesSQLite pasándole la puntuación recibida, el nombre del jugador (podemos pasar uno constante y más adelante pediremos el nombre al usuario) y el momento en el que realizamos la llamada (System.currentTimeMillis()).
3. En la actividad llamada, o sea, en la actividad Juego debemos devolver la puntuación. Sin embargo, es en VistaJuego donde podemos hacer esto de forma más sencilla puesto que allí disponemos de la variable puntuación. El problema es que la clase VistaJuego es una vista y no una actividad. Podemos resolver el problema introduciendo el siguiente código en VistaJuego:

```
private Activity padre;
```

```
public void setPadre(Activity padre) {  
    this.padre = padre;  
}
```

4. Al detectar una condición de victoria o derrota, o sea, al terminar la partida llamamos al siguiente método que debemos incorporar en VistaJuego:

```
private void TerminarJuego() {  
    if (padre != null) {  
        Bundle datos = new Bundle();  
        datos.putInt("puntuacion", puntuacion);  
        Intent intent = new Intent();  
        intent.putExtras(datos);  
        padre.setResult(Activity.RESULT_OK, intent);  
    }  
}
```

5. En nuestro caso la condición para terminar el juego será al pulsar en cualquier momento la tecla “q”. Debéis implementar este comportamiento al pulsar dicha tecla de forma que se llame al método del punto anterior al pulsar la “q”.
6. En el método onCreate de la clase Juego introducimos el siguiente código para que desde la clase VistaJuego se tenga acceso a la clase Juego:

```
vistaJuego.setPadre(this);
```

LA ENTREGA DE ESTA PRÁCTICA CONSISTE EN UNA CAPTURA DE PANTALLA DONDE SE PUEDA APRECIAR LA PANTALLA DE PUNTUACIONES EN LAS QUE APAREZCAN AL MENOS 3 PUNTUACIONES Y TODAS ELLAS CON VUESTRO NOMBRE.

2. CONTENT PROVIDERS

Un Content Provider es un mecanismo que ofrece la plataforma Android para compartir información entre aplicaciones.

Muchas aplicaciones estándar utilizan este mecanismo, como por ejemplo la lista de contactos, la aplicación de SMS, o el calendario/agenda. De esta forma, podemos acceder a los datos que gestionan estas aplicaciones desde nuestras propias aplicaciones ya que ofrecen Content Providers a los que podemos acceder.

Con los Content Providers podemos hacer dos cosas:

7. Construir un Content Provider nuevo para compartir datos de nuestra aplicación con otras aplicaciones.
8. Utilizar un Content Provider que ya exista para acceder a los datos de otras aplicaciones.

2.1. Crear un Content Provider

Para añadir un Content Provider a nuestra aplicación tenemos que realizar las siguientes tareas:

1. Definir una estructura de almacenamiento para los datos. Lo más cómodo es usar una base de datos SQLite.
2. Crear una nueva clase que extienda a la clase Android ContentProvider.
3. Declarar el nuevo Content Provider en el fichero AndroidManifest.xml

Como ejemplo crearemos una aplicación que facilitará datos de usuarios a otras aplicaciones. Los datos tendrán la siguiente estructura:

| _ID | Usuario | Contraseña | Email |
|-----|----------|------------|--|
| 1 | usuario1 | passw1 | email1@cefire.com |
| 4 | usuario2 | passw2 | email2@cefire.com |
| 5 | usuario3 | passw3 | email3@cefire.com |

NOTA: Los registros de datos proporcionados por un content provider deben contar **siempre** con un campo llamado _ID que los identifique de forma unívoca del resto de registros.

2.1.1. Definir estructura de almacenamiento

En una nueva aplicación que os paso como ejemplo (**EjemploContentProvider**) tenemos la siguiente clase:

```
package org.ejemplo.aguilar;

import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteDatabase.CursorFactory;
import android.database.sqlite.SQLiteOpenHelper;

public class UsuariosSqliteHelper extends SQLiteOpenHelper {
    //Sentencia SQL para crear la tabla de Usuarios
    String sql = "CREATE TABLE Usuarios " +
        "(_id INTEGER PRIMARY KEY AUTOINCREMENT, " +
        "usuario TEXT, " +
        "password TEXT, " +
        "email TEXT )";

    public UsuariosSqliteHelper(Context contexto, String nombre, CursorFactory almacen, int version) {
        super(contexto, nombre, almacen, version);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        //Ejecutamos la sentencia para crear la tabla
        db.execSQL(sql);

        //Introducimos 10 usuarios de ejemplo
        for (int i=1; i<=10; i++) {
            //Creamos los datos
            String usuario = "usuario" + i;
            String password = "passw" + i;
            String email = "email" + i + "@cefire.com";
            //Introducimos los datos en la tabla Usuarios
            db.execSQL("INSERT INTO Usuarios (usuario, password, email) " +
                "VALUES ('" + usuario + "', '" + password + "', '" +
                email + "')");
        }
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int versionAnterior, int versionNueva) {
        //Eliminamos la versión anterior de la tabla
        db.execSQL("DROP TABLE IF EXISTS Usuarios");
        //Creamos la nueva versión de la tabla
        db.execSQL(sql);
    }
}
```

La aplicación principal del ejemplo no hará nada puesto que lo que nos interesa es tener un Content Provider que otra aplicación utilizará para acceder a los datos que hemos creado de prueba.

2.1.2.Crear la clase extendiendo ContentProvider

El acceso a un content provider se realiza mediante una URI. Esto es una cadena de texto similar a cualquiera de las direcciones web que utilizamos en nuestro navegador. Para acceder a un content provider utilizaremos una dirección similar a "content://net.aguilar.android.ejemplo/usuarios".

Las direcciones URI de los content providers están formadas por 3 partes:

9. Prefijo "content://" -> Indica que dicho recurso deberá ser tratado por un content provider.
10. Authority -> El identificador del content provider. Este dato debe ser único por lo que puede utilizarse un "nombre de clase java invertido", en mi caso "org.ejemplo.aguilar.EjemploContentProvider".
11. Entidad concreta a la que queremos acceder dentro de los datos que proporciona el content provider. En nuestro caso es la tabla "usuarios".
12. En una URI se puede hacer referencia directamente a un registro concreto de la entidad seleccionada. Esto se hace indicando al final de la URI el ID de dicho registro. Por ejemplo la URI "content:// org.ejemplo.aguilar.EjemploContentProvider /usuarios/10" haría referencia directa al cliente con _ID=10.

Los pasos a seguir a continuación son los siguientes:

1. Creamos una nueva clase UsuariosProvider que extienda de ContentProvider.
2. Definimos la URI con la que se va a acceder a nuestro content provider. En este caso será "content:// org.ejemplo.aguilar.EjemploContentProvider /usuarios".
3. Encapsulamos esta dirección en un objeto estático de tipo Uri llamado CONTENT_URI:

```
//Definición del CONTENT_URI
private static final String uri = "content://org.ejemplo.aguilar.EjemploContentProvider/usuarios";
public static final Uri CONTENT_URI = Uri.parse(uri);
```

4. Definimos constantes con los nombres de las columnas de los datos proporcionados por nuestro content provider. Las columnas predefinidas que deben tener todos los content providers, como por ejemplo la columna "_ID" están definidas en la clase BaseColumns. Así que definimos otra clase a partir de BaseColumns añadiendo las columnas propias de nuestro content provider.

```
//Clase interna para declarar las columnas
public static final class Usuarios implements BaseColumns {
    private Usuarios() {}

    //Nombres de columnas
    public static final String COL_USUARIO = "usuario";
    public static final String COL_PASSWORD = "password";
    public static final String COL_EMAIL = "email";
}
```

5. Definimos atributos para almacenar el nombre de la base de datos, la versión, y la tabla a la que accederá nuestro content provider:

```
//Base de datos
private UsuariosSqliteHelper usdbh;
private static final String BD_USUARIO = "DBUsuarios";
private static final int BD_VERSION = 1;
private static final String TABLA_USUARIOS = "Usuarios";
```

6. Definimos un objeto UriMatcher y dos nuevas constantes que representan los dos tipos de URI que podemos encontrarnos: acceso genérico a tabla o acceso a un cliente por su ID.

```
//UriMatcher
private static final int USUARIOS = 1;
private static final int USUARIOS_ID = 2;
private static final UriMatcher uriMatcher;
//Inicializamos el UriMatcher indicándole el formato de los dos tipos de acceso:
//genérico a tabla o directo a un registro, indicándole el formato de ambos tipos
//de URI, de forma que pueda diferenciarlos y devolvernos su tipo.
static {
    uriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
    uriMatcher.addURI("net.aguilar.android.ejemplo", "usuarios", USUARIOS);
    uriMatcher.addURI("net.aguilar.android.ejemplo", "usuarios/#", USUARIOS_ID);
}
```

7. Implementamos los métodos necesarios:

- a. Método onCreate: Para inicializar la base de datos, a través de su nombre y versión.

```
@Override
public boolean onCreate() {
    usdbh = new UsuariosSqliteHelper(getContext(), BD_NOMBRE, null, BD_VERSION);
    return true;
}
```

- b. Método query: Debe devolver los datos solicitados según la URI indicada y los criterios de selección y ordenación pasados como parámetro.

```
@Override
public Cursor query(Uri uri, String[] proyeccion, String seleccion,
    String[] seleccionArgs, String ordenacion) {
    //Si es una consulta a un ID concreto construimos el WHERE
    String where = seleccion;
    if (uriMatcher.match(uri) == USUARIOS_ID) {
        where = "_id=" + uri.getLastPathSegment();
    }

    SQLiteDatabase db = usdbh.getWritableDatabase();

    Cursor c = db.query(TABLA_USUARIOS, proyeccion, where, seleccionArgs, null, null, ordenacion);

    return c;
}
```

- c. Método update: Idem al anterior pero utilizado para actualizar datos.

```
@Override
public int update(Uri uri, ContentValues valores, String seleccion, String[] seleccionArgs) {
    int cont;

    //Si es una consulta a un ID concreto construimos el WHERE
    String where = seleccion;
    if (uriMatcher.match(uri) == USUARIOS_ID) {
        where = "_id" + uri.getLastPathSegment();
    }

    SQLiteDatabase db = usdbh.getWritableDatabase();

    cont = db.update(TABLA_USUARIOS, valores, where, seleccionArgs);

    return cont;
}
```

- d. Método delete: Idem al anterior pero utilizado para eliminar datos.

```
@Override
public int delete(Uri uri, String seleccion, String[] seleccionArgs) {
    int cont;

    //Si es una consulta a un ID concreto construimos el WHERE
    String where = seleccion;
    if (uriMatcher.match(uri) == USUARIOS_ID) {
        where = "_id" + uri.getLastPathSegment();
    }

    SQLiteDatabase db = usdbh.getWritableDatabase();

    cont = db.delete(TABLA_USUARIOS, where, seleccionArgs);

    return cont;
}
```

- e. Método insert: Idem al anterior pero utilizado para insertar registros y además debe devolver la URI que hace referencia al nuevo registro insertado.

```
@Override
public Uri insert(Uri uri, ContentValues valores) {
    long regId = 1;

    SQLiteDatabase db = usdbh.getWritableDatabase();

    regId = db.insert(TABLA_USUARIOS, null, valores);

    Uri nuevaUri = ContentUris.withAppendedId(CONTENT_URI, regId);

    return nuevaUri;
}
```

- f. Método `getType`: Se utiliza para identificar el tipo de datos que devuelve el content provider. Existirán dos tipos de datos distintos para cada entidad del content provider según si se devuelve un solo registro o una lista de registros:

```
@Override
public String getType(Uri uri) {
    int match = uriMatcher.match(uri);

    switch(match) {
        case USUARIOS:
            return "vnd.android.cursor.dir/vnd.aguilar.usuario";
        case USUARIOS_ID:
            return "vnd.android.cursor.item/vnd.aguilar.usuario";
        default:
            return null;
    }
}
```

2.1.3.Declarar el Content Provider en AndroidManifest.xml

Debemos insertar un nuevo elemento `<provider>` dentro de `<application>` indicando el nombre del content provider y su authority:

```
<provider android:name="UsuariosProvider"
    android:authorities="org.ejemplo.aguilar.EjemploContentProvider" />
```

De esta forma tenemos completa la construcción de nuestro content provider mediante el cual otras aplicaciones del sistema pueden acceder a los datos almacenados por nuestra aplicación.

2.1.4.Utilización de Content Providers

En este apartado veremos cómo utilizar un Content Provider que ya exista para acceder a datos de otras aplicaciones. Utilizar un Content Provider ya existente es muy sencillo, sobre todo comparado con el costoso proceso de construcción de uno nuevo como ya hemos visto. Únicamente debemos utilizar el método `getContentResolver()` para obtener la referencia indicada y después utilizar los métodos `query()`, `update()`, `insert()` y `delete()`.

Como ejemplo crearemos una nueva aplicación que también os paso con su código fuente (**EjemploContentProvider2**) que accederá a los datos de usuarios del content provider creado en el anterior apartado y al historial de llamadas del dispositivo. En esta nueva aplicación utilizaremos cuatro botones: uno para consultar todos los usuarios, otro para introducir nuevos usuarios, el tercero para eliminar todos los usuarios nuevos introducidos con el segundo botón y el último para ver el historial de llamadas.

El layout main.xml de esta nueva aplicación sería:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <LinearLayout android:id="@+id/linearLayout1"
        android:layout_height="wrap_content"
        android:layout_width="wrap_content">

        <Button android:layout_height="wrap_content" android:text="Consultar"
            android:id="@+id/BtnConsultar" android:layout_width="wrap_content"></Button>
        <Button android:layout_height="wrap_content" android:text="Insertar"
            android:id="@+id/BtnInsertar" android:layout_width="wrap_content"></Button>
        <Button android:layout_height="wrap_content" android:text="Eliminar"
            android:id="@+id/BtnEliminar" android:layout_width="wrap_content"></Button>
        <Button android:layout_height="wrap_content" android:text="Llamadas"
            android:id="@+id/BtnLlamadas" android:layout_width="wrap_content"></Button>

    </LinearLayout>

    <TextView android:id="@+id/TxtResultados"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="" />
</LinearLayout>
```

En el método onCreate() de la actividad del nuevo proyecto además de las referencias a los controles de la interfaz gráfica, iremos incorporando los escuchadores correspondientes para realizar las tareas que se piden.

El código para el botón "Consultar" sería el siguiente:

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

    @Override
    public void onClick(View arg0) {
        //Columnas de la tabla a recuperar
        String[] proyeccion = new String[] {
            "_id",
            "usuario",
            "password",
            "email" };

        //Incorporamos el uri al que queremos acceder
        String uri = "content://org.ejemplo.aguilar.EjemploContentProvider/usuarios";
        Uri usuariosUri = Uri.parse(uri);
        Log.e("ContentProvider2", "LLEGAMOS");
        //Obtenemos una referencia al content provider al que queremos acceder
        ContentResolver cr = getContentResolver();

        //Hacemos la consulta
        Cursor cur = cr.query(usuariosUri,
            proyeccion, //Columnas a devolver
            null,        //Condición de la query
            null,        //Argumentos variables de la query
            null);       //Orden de los resultados
    }
});
```

```

        if (cur.moveToFirst())
        {
            String usuario;
            String password;
            String email;

            int colUsuario = cur.getColumnIndex("usuario");
            int colPassword = cur.getColumnIndex("password");
            int colEmail = cur.getColumnIndex("email");

            txtResultados.setText("");

            do
            {
                usuario = cur.getString(colUsuario);
                password = cur.getString(colPassword);
                email = cur.getString(colEmail);

                txtResultados.append(usuario + " - " + password + " - " + email + "\n");

            } while (cur.moveToNext());
        }
    });
}

```

El código para el botón “Insertar” sería el siguiente:

```

btnInsertar.setOnClickListener(new OnClickListener() {

    @Override
    public void onClick(View arg0) {
        //Creamos los datos de un usuario nuevo
        ContentValues values = new ContentValues();
        values.put("usuario", "UsuarioN");
        values.put("password", "PasswordXXX");
        values.put("email", "nuevo@cefire.com");

        //Incorporamos el uri al que queremos acceder
        String uri = "content://org.ejemplo.aguilar.EjemploContentProvider/usuarios";
        Uri usuariosUri = Uri.parse(uri);
        //Obtenemos una referencia al content provider al que queremos acceder
        ContentResolver cr = getContentResolver();

        //Insertamos un usuario nuevo
        cr.insert(usuariosUri, values);
    }
});

```

El código para el botón “Eliminar” sería el siguiente:

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

    @Override
    public void onClick(View arg0) {
        //Incorporamos el uri al que queremos acceder
        String uri = "content://org.ejemplo.aguilar.EjemploContentProvider/usuarios";
        Uri usuariosUri = Uri.parse(uri);
        //Obtenemos una referencia al content provider al que queremos acceder
        ContentResolver cr = getContentResolver();

        //Eliminamos el usuario introducido con el botón de Insertar
        cr.delete(usuariosUri, "usuario" + " = 'UsuarioN'", null);
    }
});
```

El código para el botón “Llamadas” que nos permitirá obtener el historial de llamadas de nuestro teléfono sería el siguiente:

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

    @Override
    public void onClick(View arg0) {

        //Columnas que queremos recuperar
        String[] proyeccion = new String[] {
            Calls.TYPE,
            Calls.NUMBER };

        //Uri a la que queremos acceder. En este caso es la del historial de llamadas
        Uri llamadasUri = Calls.CONTENT_URI;

        //Referencia al content provider al que queremos acceder
        ContentResolver cr = getContentResolver();

        Cursor cur = cr.query(llamadasUri,
            proyeccion, //Columnas a devolver
            null,        //Condición de la query
            null,        //Argumentos variables de la query
            null);       //Orden de los resultados

        if (cur.moveToFirst())
        {
            int tipo;
            String tipollamada = "";
            String telefono;

            int colTipo = cur.getColumnIndex(Calls.TYPE);
            int colTelefono = cur.getColumnIndex(Calls.NUMBER);

            txtResultados.setText("");

            do
            {
                tipo = cur.getInt(colTipo);
                telefono = cur.getString(colTelefono);

                if(tipo == Calls.INCOMING_TYPE)
                    tipollamada = "ENTRADA";
                else if(tipo == Calls.OUTGOING_TYPE)
                    tipollamada = "SALIDA";
                else if(tipo == Calls.MISSED_TYPE)
                    tipollamada = "PERDIDA";
            }
        }
    }
});
```

```

        txtResultados.append(tipoLlamada + " - " + telefono + "\n");
    } while (cur.moveToNext());
}
}
});
}
}
}

```

No debemos olvidar añadir al fichero AndroidManifest.xml fuera de las etiquetas <application> las siguientes líneas para dar permisos de acceso al registro de llamadas.

```

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

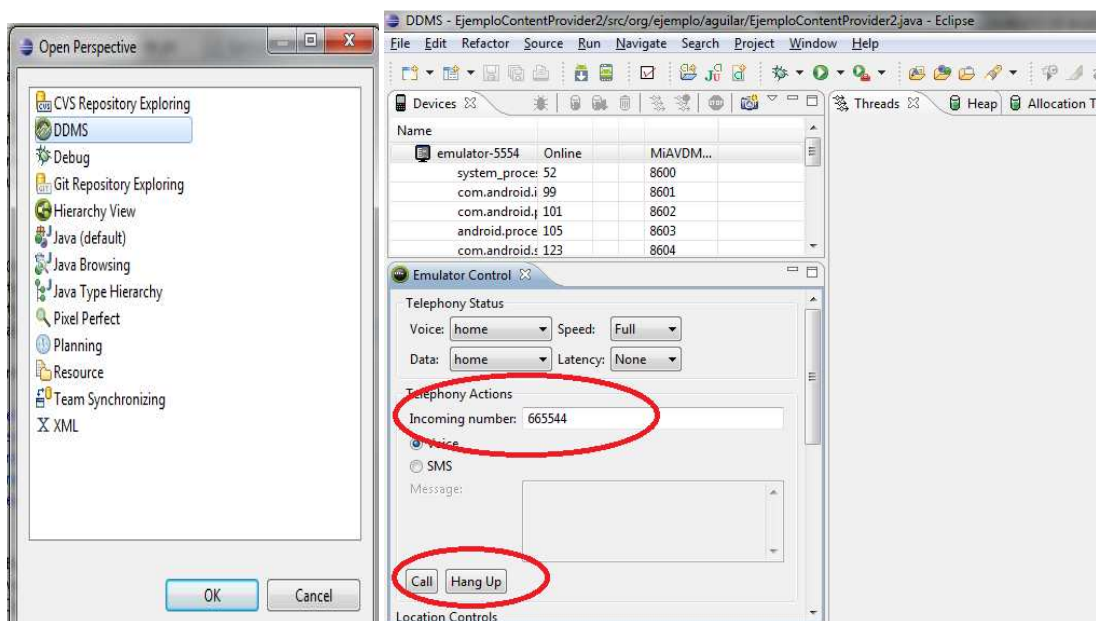
```

Como podréis suponer, esta última función del historial de llamadas podemos probarla en el emulador siempre que simulemos antes llamadas entrantes y salientes.

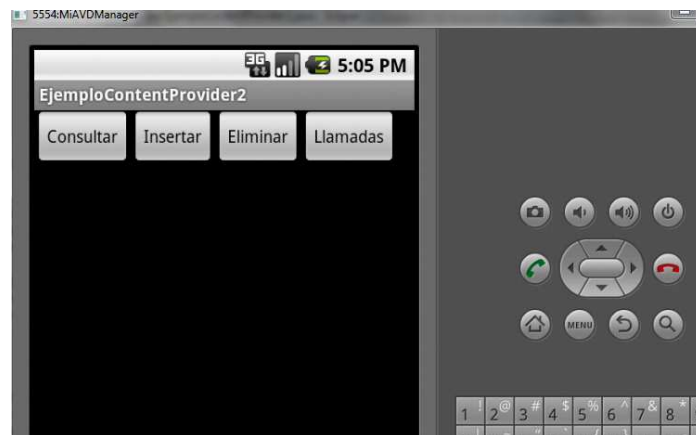
Las llamadas salientes son sencillas, simplemente vamos al emulador, accedemos al teléfono, marcamos y descolgamos igual que lo haríamos en un dispositivo físico.

La emulación de llamadas entrantes la podemos hacer desde Eclipse, accediendo a la vista del DDMS. En esta vista, si accedemos a la sección “Emulator control” veremos un apartado llamado “Telephony Actions”. Desde éste, podemos introducir un número de teléfono origen cualquiera y pulsar el botón “Call” para conseguir que nuestro emulador reciba una llamada entrante.

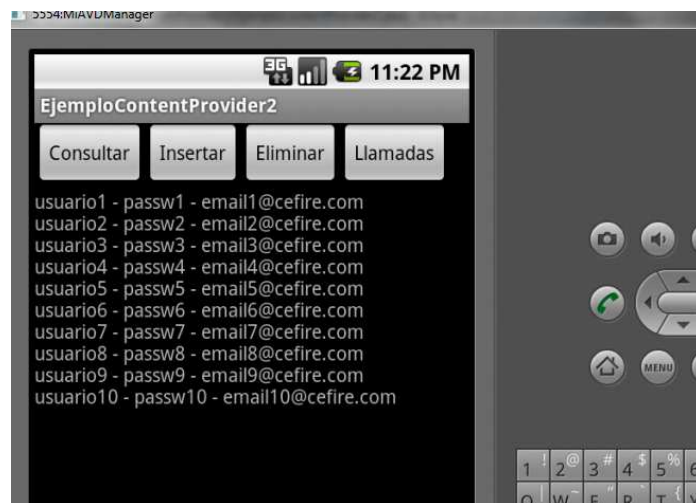
La emulación de llamadas perdidas será igual que la de llamadas entrantes pero en el emulador pulsaremos “Hang up” para terminar la llamada simulando así una llamada perdida.



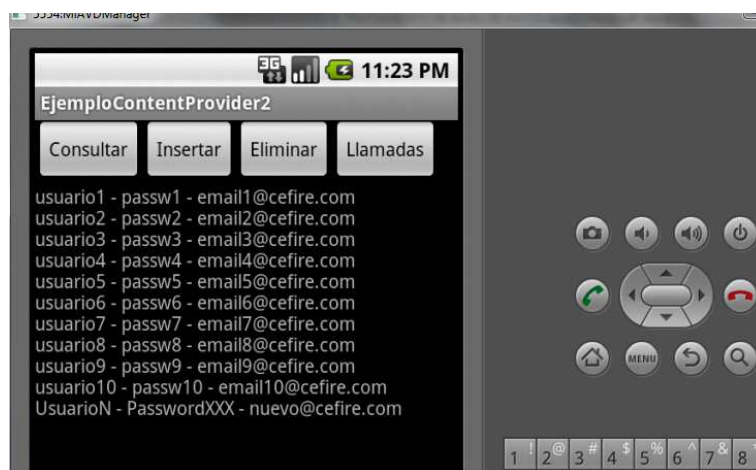
Una vez que hemos realizado varias llamadas entrantes, salientes y perdidas, podemos ejecutar la aplicación para verlas. El resultado sería algo parecido a esto:



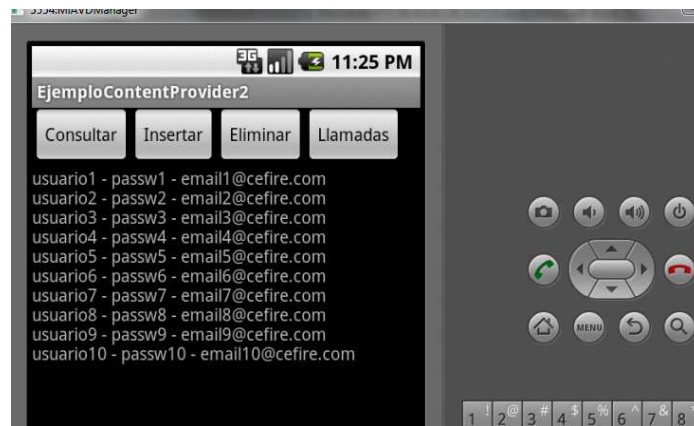
Al pulsar en el botón “Consultar” obtenemos:



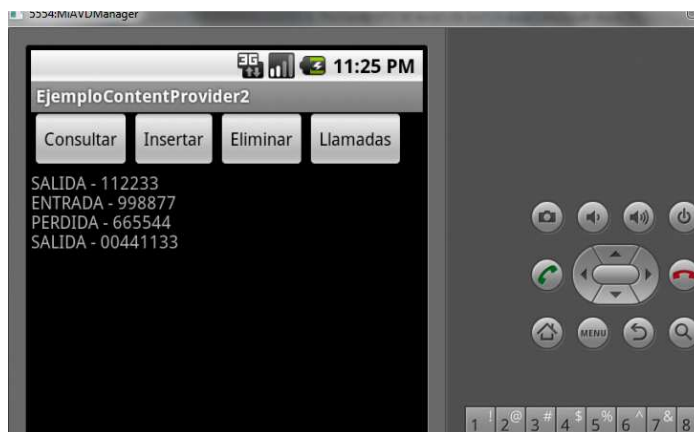
Al pulsar en el botón “Insertar” y después en el botón “Consultar” para ver qué ha ocurrido obtenemos:



Al pulsar en el botón “Eliminar” y después en “Consultar” para ver qué ha ocurrido obtenemos:



Al pulsar en el botón “Llamadas” una vez que tengamos llamadas entrantes, salientes y perdidas en el terminal, obtenemos algo parecido a:



En el ejemplo hemos accedido al historial de llamadas pero tenemos la posibilidad de acceder a otra información del teléfono. Para eso necesitamos saber las URIs de los Content Providers que contienen los datos. En la siguiente tabla tenemos las disponibles en Android:

| CLASE | INFORMACIÓN ALMACENADA | EJEMPLOS DE URIs |
|------------|---|--|
| Browser | Enlaces favoritos, historial de navegación, de búsquedas. | content://browser/bookmarks |
| CallLog | Llamadas entrantes, salientes y perdidas. | content://call_log/calls |
| Contacts | Lista de contactos del usuario. | content://contacts/people |
| MediaStore | Ficheros de audio, vídeo e imágenes, almacenados en dispositivos de almacenamiento internos y externos. | content://media/internal/images content://media/external/video content://media/*/audio |
| Setting | Preferencias del sistema. | content://settings/system/ringtone content://settings/system/notification_sound |