uc3m | Universidad Carlos III de Madrid
Grupo de investigación:
Computer Security Lab

Mobile Devices Security

Bachelor Degree in Informatics Engineering

2019

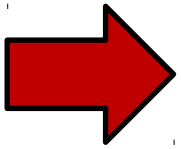
Agenda

- Life Cycle of an Activity
- Broadcast Receiver
- Services
- Content Provider

Clases

Recalling:

After 9 seconds



Life Cycle

What happens at the background of the activity:

- We still can hear the song, do we want that to happen?
- What happens with activity “SeTiBienvenida”?
- How much memory are we currently using?
- What happens if someone calls at this moment?

Life Cycle

Getting back to our application, when we change the form activity "SeTiBienvenida" to the activity "Inicio" :

- The application stops the music
- The application is destroyed so that memory is not overloaded:

```
...      ...
Thread reloj = new Thread(){
        public void run(){
        try{
            sleep(9000);
            Intent abrirInicio = new Intent(SetiBienvenida.this, Inicio.class);
            startActivity(abrirInicio);
            }
            catch (InterruptedException e){
            e.printStackTrace();
            }
        } /** Fin del metodo public void run()  */
     }; /** Fin del Thread reloj = new Thread()  */
     reloj.start();
}  /** Fin del protected void onCreate(Bundle objetoSeti) */
```

```
@Override
protected void onPause() {
        super.onPause();
        miCancion.release(); // paramos la cancion
        finish(); // destruimos la aplicacion setibienvenida
   }  /**  Fin del método protected void onPause() **/
```

```
} /** Fin de la public class SetiBienvenida */
```

Broadcast Receiver

Recalling and expanding our app:

- We create a “Welcome to SDM class” project
- We added new activity:
  - We added new layout “seti.xml”
  - Background with logo “SetiApp”
  - Added some music at the background
  - We called this activity “SetiBienvenida” (java class)
- We changed the Blank “Welcome SDM” activity
  - We modified the “activity_inicio.xml” layout
  - We added two buttons and text that holds a counter
- Now we will add a Broadcast receiver
  - Counter = 10, trigger an Alarm
  - Vibrate the phone for 3 sec. + Toast Message

Create the Receiver class

1. Implement the `MiBCReceiver` class and its methods

To make the phone vibrate we implement the Vibrator class and the Context class to get the current state of the application/object through its method: getSystemService(String name)

```
package com.sdm.setiapp;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.os.Vibrator;
import android.util.Log;
import android.widget.Toast;

public class MiBCReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        Toast.makeText(context, "Cuidado, cuidado ha llegado a 10!!!!.", Toast.LENGTH_LONG).show();
        // hacemos vibrar el móvil
        Vibrator vibrator = (Vibrator) context.getSystemService(Context.VIBRATOR_SERVICE);
        vibrator.vibrate(3000);
        Log.i("Receiver", "Intent Recibido");
    } /** fin del metodo public void onReceive(Context context, Intent intent) **/
} /** Fin de la clase public class MiBCReceiver extends BroadcastReceiver**/
```

http://developer.android.com/reference/android/os/Vibrator.html

Implementing a Receiver

2. Modify the Manisfest.xml file: registering and giving the permission

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.SDM.setiapp"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="19" />

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

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >

        <activity
            android:name=".SetiBienvenida"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

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

        <receiver android:name="MiBCReceiver"></receiver>

    </application>
</manifest>
```

Implementing a Receiver

3. Register the Receiver in the main class where the event occurs

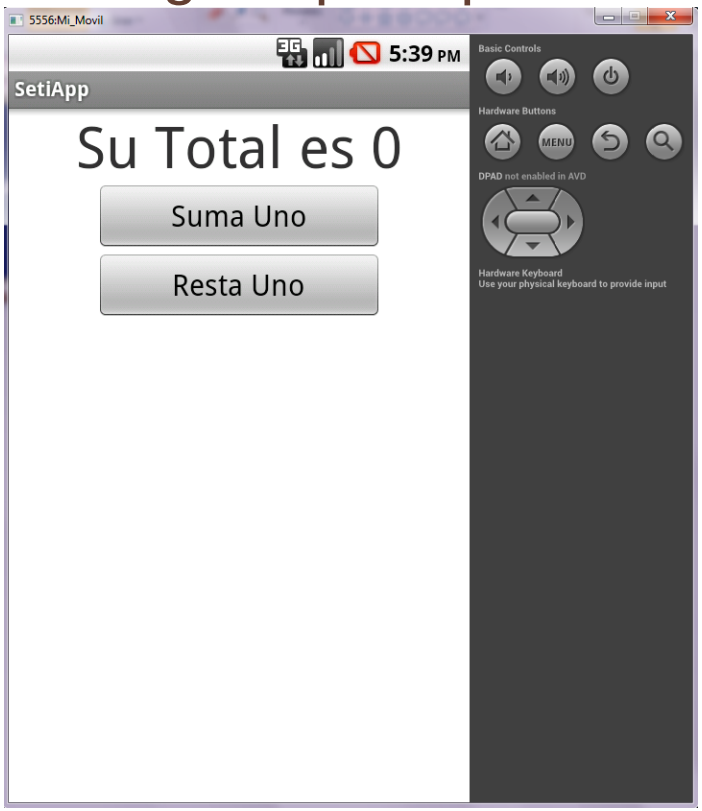
The receiver will be triggered when the counter gets to 10. We create a method to verify the counting

```
sumar.setOnClickListener(new View.OnClickListener() {
@Override
   public void onClick(View v) {

     if(contador == 10)
     contador++;
     activarReceiver(contador);
     mostrar.setText("Su Total es " + contador);


   }
}); /** Fin del metodo sumar.setOnClickListener(new View.OnClickListener() */

restar.setOnClickListener(new View.OnClickListener() {
@Override
   public void onClick(View v) {
     if(contador == 10)
     contador--;
     activarReceiver(contador);
     mostrar.setText("Su Total es " + contador);


   }
});  /** Fin del metodo restar.setOnClickListener(new View.OnClickListener() */
```

```
public void activarReceiver(int c) {
   int i = c;
   Intent intent = new Intent(this, MiBCReceiver.class);
   sendBroadcast(intent);
   Log.i("Receiver", “Intent Enviado");
} /** Fin del public void activarReceiver(int c) */
```

Receiver: Example

4. We run our application. First we notify that the alarm is set to 10. Afterwards the phone vibrates for 3 seconds and a message is prompt to confirm it.



Services

- It is an application that is executed in the background, it doesn't need any user interaction
  - It continues even when the activity that launched it is destroyed
  - It finishes when it is disconnected from all applications
  - It is used when something has to be done/executed while the user doesn´t interact with the application, otherwise threads are used
- Allows multiples applications to communicate through a common interface
- It has to be declared in the "manisfest.xml"
- Similarly to the services in Windows or daemon in Unix. No UI: check events e.g.: new email etc.
- It has its own life cycle structure

Implementing a service

In order to get a better idea how services work, let´s create a simple service project

Our app:

- An interface with two buttons, one to start the service the other to shut it down
- To notice the service, when the service is started a song will play, and when it stops the song will stop as well
- When the service starts or stops we use a Toast message

Implementing a service

We create the Layout:

```xml
<?xml version="1.0" encoding="utf-8"?>
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/tlTable"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <TableRow
        android:id="@+id/tr0"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:alpha="0"
        android:minHeight="50dp" >
    </TableRow>

    <TableRow
        android:id="@+id/tr1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:padding="5dip" >

    <Button
            android:id="@+id/bIniciarSer"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:gravity="center"
            android:text="@string/iniciar_service" />
    </TableRow>

   <TableRow
        android:id="@+id/tr2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:alpha="0"
        android:minHeight="20dp" >
    </TableRow>

    <TableRow
        android:id="@+id/tr3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:padding="5dip" >

    <Button
        android:id="@+id/bPararSer"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/parar_service" />
    </TableRow>

</TableLayout>
```

Implementing a service

We call it: maser_act.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/tlTable"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <TableRow
        android:id="@+id/tr0"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:alpha="0"
        android:minHeight="50dp" >
    </TableRow>

    <TableRow
        android:id="@+id/tr1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:padding="5dip" >

    <Button
            android:id="@+id/bIniciarSer"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:gravity="center"
            android:text="@string/iniciar_service" />
    </TableRow>

   <TableRow
        android:id="@+id/tr2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:alpha="0"
        android:minHeight="20dp" >
    </TableRow>

    <TableRow
        android:id="@+id/tr3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:padding="5dip" >

    <Button
        android:id="@+id/bPararSer"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/parar_service" />
    </TableRow>

</TableLayout>
```

Implementing a service

We create the class for the service and we call it: "MiServicio"

```
package com.SDM.example;

import android.app.Service;
import android.content.Intent;
import android.os.IBinder;
import android.widget.Toast;

public class MiServicio extends Service {

   @Override
   public IBinder onBind(Intent arg0) {

        return null;
   } /** Fin del metodo public IBinder onBind(Intent arg0) ***/
  @Override
  public int onStartCommand(Intent intent, int flags, int startId) {
  // This service will execute until we stop it explicitly
  Toast.makeText(this, "El Servicio iniciado", Toast.LENGTH_SHORT).show();
  return START_STICKY;
  }  /** Fin del metodo public int onStartCommand(...) ***/


} /** Fin de la clase public class MiServicio extends Service
***/
```

1. All services (classes) implement the "Service" class
2. onBind(Intent arg0) – allows a service to connect to an activity, this makes possible for the activity to have access to the members and methods of the service class
3. To start explicitly a service we use the method: `onStartCommand(....)`
4. `START_STICKY;` - variable that allow to keep the service in execution until we want to stop it explicitly

Implementing a service

Continue …

```
package com.SDM.example;

import android.app.Service;
import android.content.Intent;
import android.os.IBinder;
import android.widget.Toast;

public class MiServicio extends Service {

   @Override
   public IBinder onBind(Intent arg0) {

        return null;
   } /** Fin del metodo public IBinder onBind(Intent arg0) ***/

   @Override
   public int onStartCommand(Intent intent, int flags, int startId) {
   // This service will execute until we stop it explicitly
   Toast.makeText(this, "El Servicio iniciado", Toast.LENGTH_SHORT).show();
   return START_STICKY;
  }  /** Fin del metodo public int onStartCommand(...) ***/


   @Override
   public void onDestroy() {
   super.onDestroy();
   Toast.makeText(this, "Servicio parado",
   Toast.LENGTH_SHORT).show();
   }  /** Fin del metodo public int onStartCommand(...) ***/
} /** Fin de la clase public class MiServicio extends Service ***/
```

5. Once we click on the “Iniciar Servicio” button, the method `startService(...);` in the main activity is triggered, this will call the `onStartCommand(...);` method in the service class

6. The service will stop when the user clicks on “Parar Servicio“ button, which execute the `stopService(...);` method written in the main activity. This method will call the `onDestroy();` method written in the service class

Implementing a service

Writing the code for our main activity: “Actividad_Principal”

```
package com.SDM.example;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;

public class Actividad_Principal extends Activity{
@Override
protected void onCreate(Bundle objServicio) {
super.onCreate(objServicio);
setContentView(R.layout.maser_act);
```

1. `onStartCommand(....);` - in the service class it is called from the `startService(...);` method in the main class

2. `onDestroy();` - it is used to stop the service in the service class. This method is called from the `stopService(...);` method in main class

```
Button bIniciar = (Button) findViewById(R.id.bIniciarSer);
 bIniciar.setOnClickListener(new View.OnClickListener() {

   public void onClick(View v) {
      startService(new Intent(getBaseContext(), MiServicio.class));
   }
 }); /** fin del metodo bIniciar.setOnClickListener(...) **/
```

```
Button bParar = (Button) findViewById(R.id.bPararSer);
 bParar.setOnClickListener(new View.OnClickListener() {

   public void onClick(View v) {
     stopService(new Intent(getBaseContext(), MiServicio.class));
   }
 }); /** fin del metodo bParar.setOnClickListener(...) **/
}  /** fin del metodo protected void onCreate(Bundle objServicio) **/

} /** fin de la clase public class Actividad_Principal extends Activity**/
```

Shall we try our App?

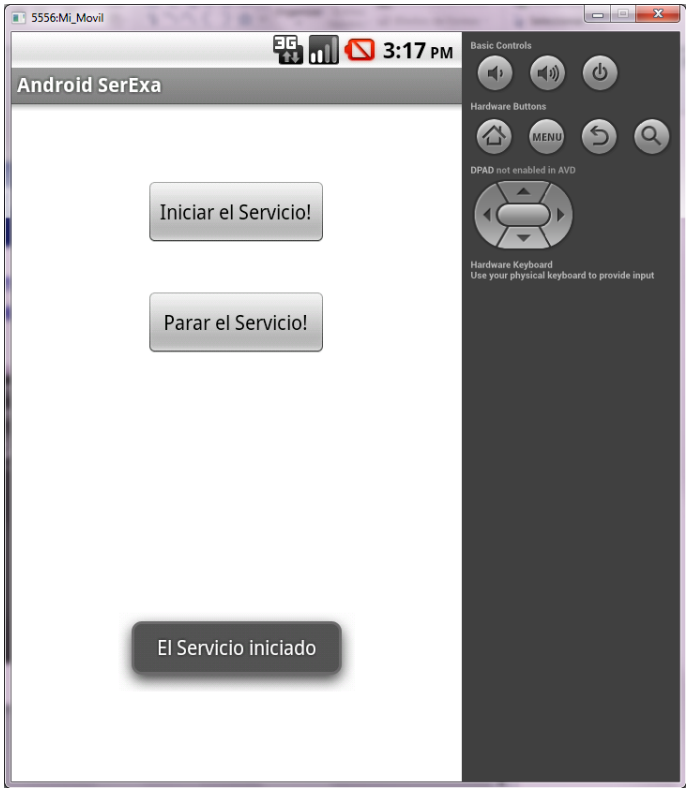
Implementing a service

We need to modify the Manisfest.xml: registering the service

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="sdm.com.service">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity
            android:name=".Actividad_Principal"
            android:label="@string/title_MiServicio">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <service android:name="MiServicio"></service>

    </application>
</manifest>
```

Implementing a service

We click on “Iniciar el Servicio!” button



Implementing a service

We click on“Parar el Servicio! button”

Implementing a service

MiServicio.java File

```
public class MiServicio extends Service {

    MediaPlayer miCancion;
    private static final String TAG = "Clase MiServicio ";

    @Override
    public IBinder onBind(Intent arg0) {

        return null;
    } /** Fin del metodo public IBinder onBind(Intent arg0) ***/

    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
// Este servicio se ejecutará hasta que se forma explicita se pare
        Toast.makeText(this, "El Servicio iniciado", Toast.LENGTH_SHORT).show();
        miCancion = MediaPlayer.create(MiServicio.this, R.raw.ontheroadagain);
        miCancion.start();
        Log.i(TAG, "El Servicio se ha iniciado");
        return START_STICKY;
    }  /** Fin del metodo public int onStartCommand(...) ***/

    @Override
    public void onDestroy() {
        super.onDestroy();
        Toast.makeText(this, "Servicio parado", Toast.LENGTH_SHORT).show();
        miCancion.stop();
        Log.i(TAG, "El Servicio se ha parado");
    }  /** Fin del metodo public int onStartCommand(...) ***/

} /** Fin de la clase public class MiServicio extends Service ***/
```

Implementing a service

In order to see that the service has been stopped. We can make use of the Log class and its methods:

```
private static final String TAG = "Clase MiServicio ";
MediaPlayer miCancion;
...    ..       ..   ...
@Override
public int onStartCommand(Intent intent, int flags, int startId) {
  // This service will execute until we stop it explicitly
  Toast.makeText(this, "El Servicio iniciado", Toast.LENGTH_SHORT).show();
  miCancion = MediaPlayer.create(MiServicio.this, R.raw.ontheroadagain);
  miCancion.start();
  Log.i(TAG, "El Servicio se ha iniciado");
  return START_STICKY;
}
```

We play our song!!

Adding this code line to make use of the Log method

Declaration | Console | LogCat | Devices

Search for messages. Accepts Java regexes. Prefix with pid:, app:, tag: or text: to limit scope. info

	PID	TID	Application	Tag	Text
861	347	354		AndroidRuntime	NOTE: attach of thread 'Binder Thread #3' fai
790	68	92	system_process	ActivityManager	Displayed activity com.SDM.example/.Actividad otal 2046 ms)
730	353	353	com.SDM.example	Clase MiServicio	El Servicio se ha iniciado

http://developer.android.com/reference/android/util/Log.html

Implementing a service

We do the same when we stop the service:

```
private static final String TAG = "Clase MiServicio ";
MediaPlayer miCancion;
...    ..       ..   ...
@Override
public int onDestroy() {
  // This service will execute until we stop it explicitly
  Toast.makeText(this, "Servicio parado", Toast.LENGTH_SHORT).show();
  miCancion.stop();
  Log.i(TAG, "El Servicio se ha parado");
} /** Fin del metodo public void onDestroy() ***/
```

We stop our song!!

We add the same line of code in order to see in LogCat the use of the Log method

Declaration | Console | LogCat | Devices

Search for messages. Accepts Java regexes. Prefix with pid:, app:, tag: or text: to limit scope. info

	PID	TID	Application	Tag	Text
790	68	92	system_process	ActivityManager	Displayed activity com.SDM.example/.Actividad_ otal 2046 ms)
730	353	353	com.SDM.example	Clase MiServicio	El Servicio se ha iniciado
790	353	353	com.SDM.example	Clase MiServicio	El Servicio se ha parado

http://developer.android.com/reference/android/util/Log.html

Content Provider

To retrieve all contacts stored on the device contacts app, and then showing them in the console:

```
package com.SDM.setiprovider;

import android.app.ListActivity;
import android.database.Cursor;
import android.net.Uri;
import android.os.Bundle;
import android.provider.ContactsContract;
import android.provider.ContactsContract.Contacts;
import android.util.Log;

public class ContactsActivity extends Activity{
@Override
protected void onCreate(Bundle objProvider) {
   super.onCreate(objProvider);
   setContentView(R.layout.main_act);
   Uri todosContactos = Uri.parse("content://contacts/people");_
   Cursor c = getContentResolver().query(todosContactos, null, null, null, Contacts.DISPLAY_NAME);
   PrintContactos(c); // mandamos a imprimir
} /** fin del metodo protected void onCreate(Bundle savedInstanceState) **/


private void PrintContactos(Cursor c) {
  if(c.moveToFirst()){
    do{
        String contID = c.getString(c.getColumnIndex(ContactsContract.Contacts._ID));
        String contDameNombre = c.getString(c.getColumnIndex(ContactsContract.Contacts.DISPLAY_NAME));
        Log.v("Content Providers", contID + ", "+ contDameNombre);
      }while(c.moveToNext());
    } /** fin del bucle if **/

  } /** fin del metodo private void PrintContactos(Cursor c) **/

} /** fin de la clase public class ContactsActivity extends ListActivity **/
```

- Cursor - is an interface that provides us with a means to access the data set returned by a query
  - It is downloaded itself when the app is stopped
  - It is likewise called when the app is restarted
  - To query CP we use the methods:
    - getContentResolver.query()
    - Activity.managedQuery()

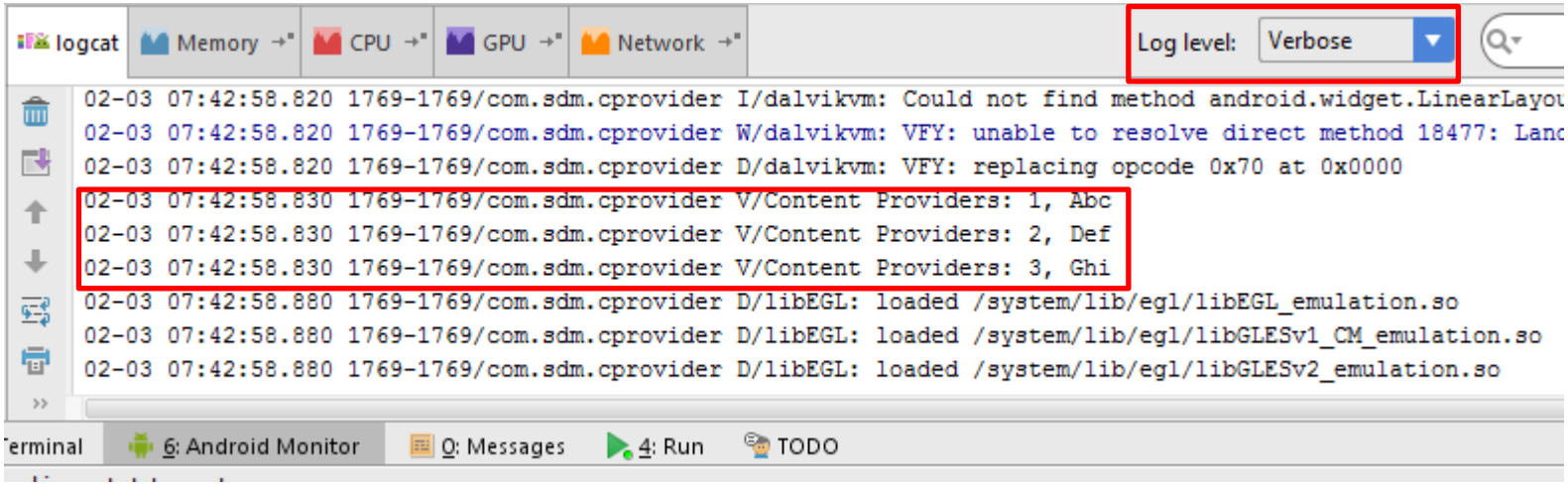
Content Provider

- We have to give the permission in the Manisfest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.SDM.setiprovider“ android:versionCode="1" android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="15"
        android:targetSdkVersion="19" />

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

    <application
        android:allowBackup="true“ android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" android:theme="@style/AppTheme" >

        <activity
            android:name=".ContactsActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

    </application>

</manifest>
```

Content Provider

- Execute your application and verify the result with the help of the LogCat



- Task: how to make visible the contacts in an interface?
  - **Using an Adapter: ListAdapter**

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

uc3m | Universidad Carlos III de Madrid

Grupo de investigación:
Computer Security Lab

Mobile Devices Security

Bachelor Degree in Informatics Engineering

2019