



Broadcast Receiver

Índice



Broadcast Receiver

1 Introducción	3
2 La clase BroadcastReceiver	4
3 Registro de un BroadcastReceiver	6
4 Envío de SMS desde aplicaciones Android	16
5 BroadcastReceiver en línea	18
6 Notificación de eventos	22

1. Introducción

Los Broadcast Receiver o receptores de sucesos son un tipo de componente Android que nos van a permitir crear aplicaciones capaces de **responder a determinados sucesos que ocurran en el terminal**, como la recepción o envío de una llamada, la llegada de un SMS, etc.

Además de esto, a lo largo de este apartado veremos también como notificar sucesos personalizados desde nuestra aplicación. Estos sucesos serán capturados también por componentes Broadcast Receiver que podrán estar definidos en aplicaciones diferentes.



2. La clase BroadcastReceiver

La clase **BroadcastReceiver** del paquete `android.content` proporciona la base para la creación de este tipo de componentes.

Dispone del método abstracto `onReceive()` que habrá que sobrescribir para definir en él el código de respuesta al suceso que queremos responder. Su formato es el siguiente:

```
void onReceive(Context context, Intent intent)
```

Recibe como primer parámetro el contexto de la aplicación, mientras que el segundo es el objeto `Intent` que contiene los datos con información sobre el suceso.

Dependiendo del tipo de suceso, podemos utilizar los métodos `getXxxExtra()` de `Intent` para extraer los datos relativos al suceso. Por ejemplo, si queremos responder al suceso de recepción de llamada, en la clase `TelephonyManager` de `android.telephony` tenemos una serie de constantes con los nombres de los datos proporcionados en el

`Intent`, entre estas constantes están:

- **TelephonyManager.EXTRA_INCOMING_NUMBER.** Constante utilizada para recuperar el número de teléfono de de la llamada entrante
- **TelephonyManager.EXTRA_STATE.** Constante utilizada para recuperar el estado de la llamada. Entre los posibles estados están:
 - `TelephonyManager.CALL_STATE_RINGING.` Llamando.
 - `TelephonyManager.CALL_STATE_OFFHOOK.` Colgado.

El siguiente bloque de instrucciones correspondería a un BroadcastReceiver que mostrará un mensaje en unToast con el número de teléfono de la llamada entrante, cada vez que se reciba una llamada en el terminal:

En este caso, dado que estamos utilizando el servicio de telefonía del terminal para acceder a los datos de las llamadas, necesitamos otorgar el siguiente permiso a nuestra aplicación:

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

```
public class ReceptorLlamadas extends BroadcastReceiver {

    @Override
    public void onReceive(Context context, Intent intent) {
        String tlf="";
        String estado=intent.getStringExtra(
            TelephonyManager.EXTRA_STATE);
        //si el telefono está sonando,
        //es que se trata de llamada entrante
        if(estado.equals(
            TelephonyManager.EXTRA_STATE_RINGING)){


            tlf=intent.getStringExtra(
                TelephonyManager.EXTRA_INCOMING_NUMBER);
            String msg="Llamada del número: "+tlf;
            Toast.makeText(context, msg,
                Toast.LENGTH_LONG).show();

        }
    }
}
```

3. Registro de un BroadcastReceiver

Para que nuestra subclase de BroadcastReceiver sea utilizada como receptor de un determinado suceso, es necesario registrar el componente en el archivo de manifiesto de la aplicación y asociarle el suceso correspondiente.

El registro se realiza mediante el elemento `<receiver>`, en cuyo atributo `name` se indicará el nombre de la subclase BroadcastReceiver. Para asociar el componente al suceso, utilizaremos el mismo sistema de acciones que empleamos con las actividades, y es que, al igual que las actividades del sistema, los sucesos tienen asociada una acción. Por ejemplo, en el caso del suceso de recepción de llamada, la acción se encuentra definida en la constante `android.intent.action.PHONE_STATE`, por lo que el registro del componente sería:



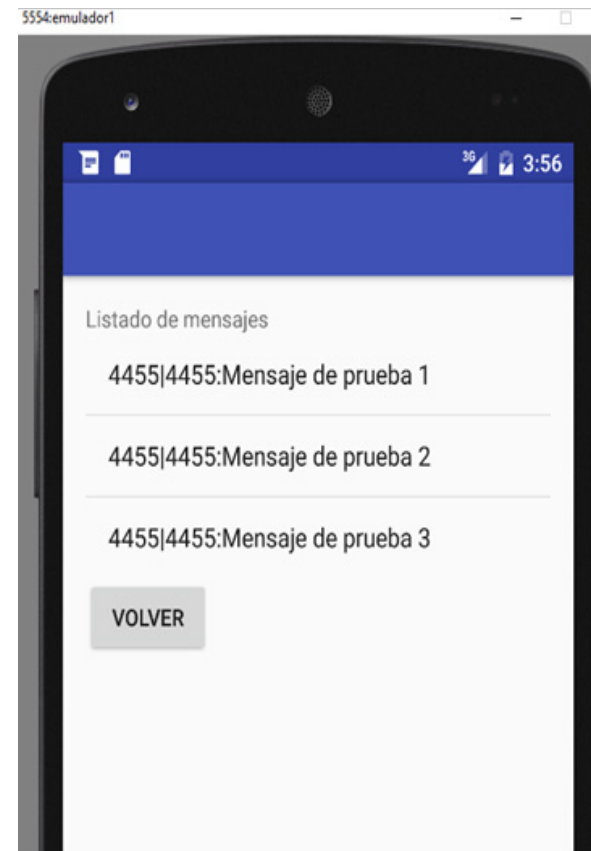
```
<receiver
    android:name=".ReceptorLlamadas"
    android:enabled="true"
    android:exported="true" >
    <intent-filter>
        <action android:name="android.intent.action.
PHONE_STATE"/>
    </intent-filter>
</receiver>
```

Ejercicio resuelto

Vamos a crear un ejercicio completo en el que vamos a implementar un BroadcastReceiver para responder a uno de los sucesos que se producen en el terminal, en este caso, a la llegada de un SMS.

Se trata de una aplicación que cada vez que llegue un SMS registrará en un fichero el texto del mensaje junto con el número desde el que se ha enviado.

Por otro lado, la actividad principal contará con un botón "mostrar mensajes", que al ser pulsado mostrará la lista de mensajes almacenados en el fichero en una lista:



Una vez creado el proyecto Android, lo primero que haremos será crear una clase donde encapsulemos las operaciones sobre el fichero, tal y como hicimos en los ejercicios sobre persistencia de datos que hemos realizado en apartados anteriores. Esta clase la llamaremos GestionFichero, la guardaremos en un paquete modelo y tendrá el siguiente código:

```
public class GestionFichero {
    private Context contexto;
    private String nombre;

    public GestionFichero(Context contexto, String nombre){
        this.contexto=contexto;
        this.nombre=nombre;
    }

    public void guardarMensaje(String mensaje ){
        FileOutputStream fos= null;
        PrintStream out=null;
        try {

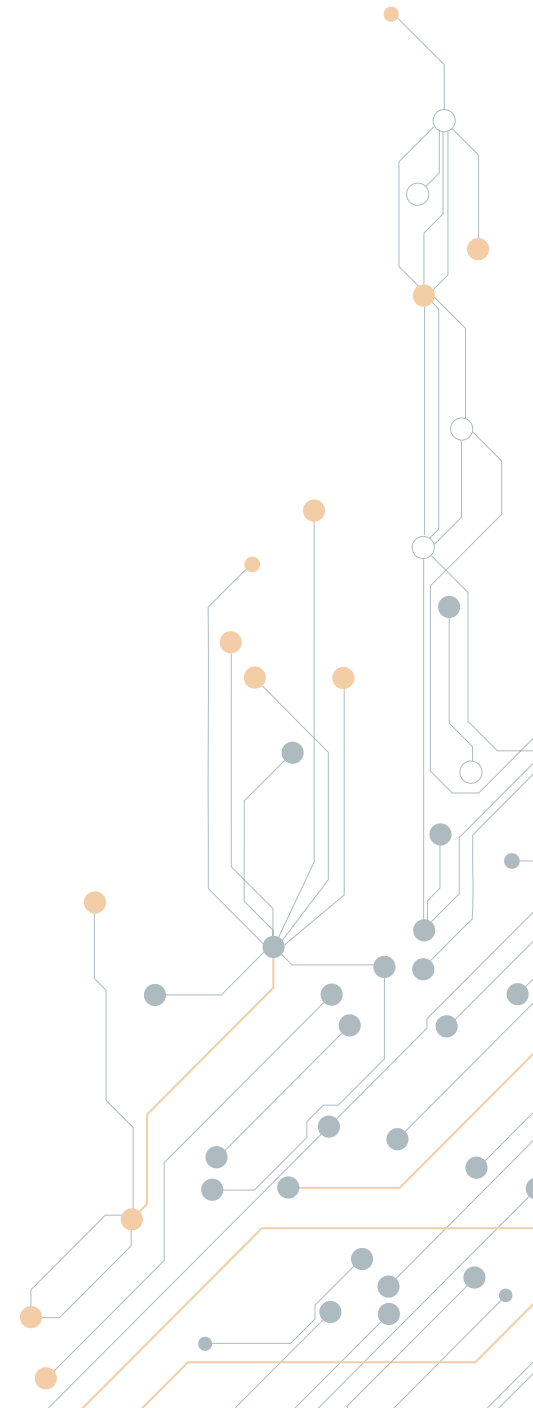
            fos = contexto.openFileOutput(nombre, Context.MODE_
APPEND);
            out=new PrintStream(fos);
            out.println(mensaje);
            out.flush();

        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }
        finally {
            if(out!=null){
                out.close();
            }
        }
    }
}
```

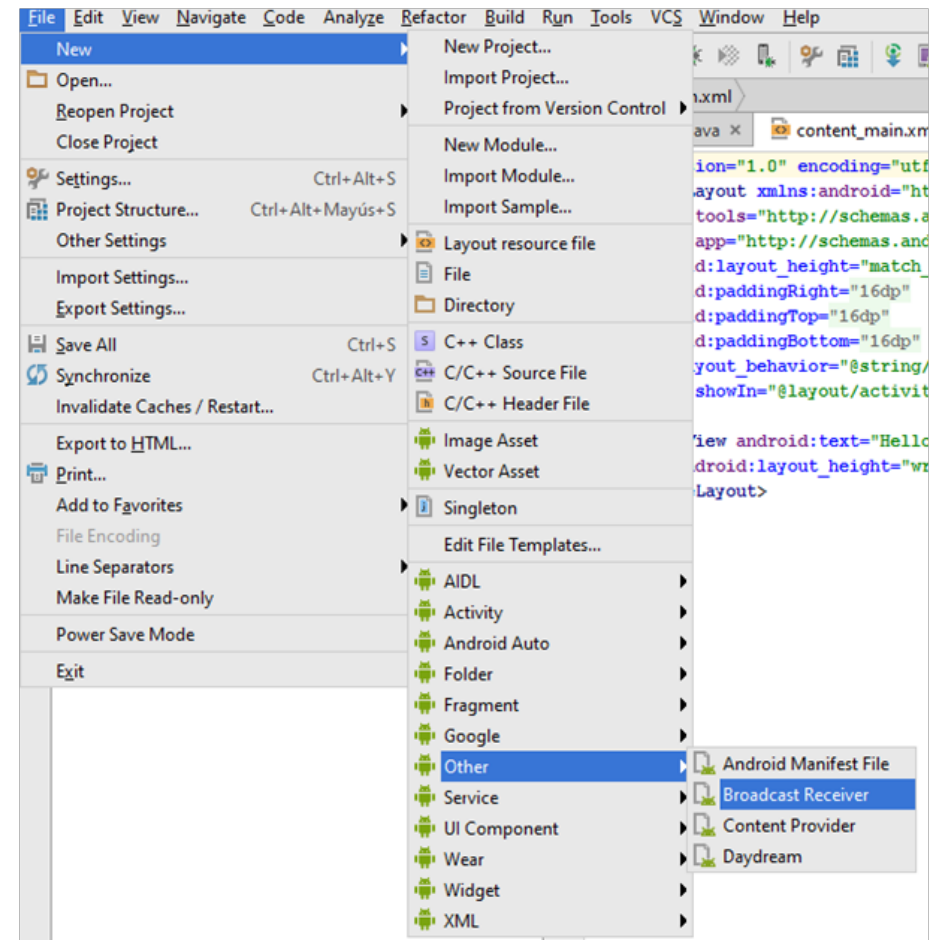



```
public ArrayList<String> recuperarMensajes(){
    ArrayList<String> notas=new ArrayList<>();
    FileInputStream fis=null;
    BufferedReader bf=null;
    try{
        fis=contexto.openFileInput(nombre);
        bf=new BufferedReader(new InputStreamReader(fis));
        String men;

        while((men=bf.readLine())!=null){
            notas.add(men);
        }
    }
    catch(IOException ex){
        ex.printStackTrace();
    }
    return notas;
}
}
```



A continuación, vamos a crear el BroadcastReceiver encargado de responder a la llegada de los SMS. Para ello, nos situamos sobre la carpeta *app* del proyecto dentro de Android Studio y en el menú que aparece al pulsar el botón derecho del ratón elegiremos File->New->Other-> Broadcast Receiver:



Nos pedirá el nombre de la clase, a la que llamaremos ReceptorSms.

El siguiente listado corresponde a la implementación de esta clase. Seguidamente explicaremos las instrucciones más importantes:

```
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.telephony.SmsMessage;
import modelo.GestionFichero;

public class ReceptorSms extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        String origen="";
        String textoSms="";
        //los SMS se envían en trozos que forman un array de
objetos
        //el conjunto de trozos se llama pdu
        Object[] pdu=(Object[])intent.getExtras().get("pdu");
        //recorremos los trozos para montar el mensaje completo
        for(int i=0;i<pdu.length;i++){
            //obtenemos el objeto SmsMessage de cada pdu
            SmsMessage sm= SmsMessage.createFromPdu(
                (byte[]) pdu[i],"3gpp");
            //recuperamos el número desde el que se
            //ha enviado el mensaje
            origen=sm.getOriginatingAddress();
            //recuperamos el texto de ese trozo de mensaje
            textoSms+=sm.getOriginatingAddress()+":"+sm.getMessageBody().toString();
        }
        GestionFichero gestion=new
            GestionFichero(context,"mensajes.txt");
        gestion.guardarMensaje(origen+"|"+textoSms);
    }
}
```

En primer lugar, vemos que para recuperar el mensaje recogemos del intent el **extra "pdus"**, que consiste en un array de objetos con los trozos que forman el mensaje. Y es el mensaje enviado desde el emisor del suceso se trocea en varias partes, dependiendo del tamaño del mismo.

Cada parte del mensaje es un **objeto de tipo SmsMessage**. Para crear el objeto SmsMessage a partir de un pdu, recurrimos al método estático de la clase SmsMessage *createFromPdu()*, que recibe como parámetro un array de byte con los datos del pdu y el formato de codificación del mismo ("3gpp"):

```
SmsMessage sm= SmsMessage.  
createFromPdu((byte[]) pdus[i],"3gpp");
```

La clase SmsMessage dispone de los siguiente métodos para obtener información sobre el mensaje:

- **getOriginatingAddress()**. Número de teléfono del emisor del mensaje
- **getMessageBody()**. Cuerpo del mensaje

En cuanto al registro de la clase ReceptorSms, al crearla con el asistente se registra automáticamente en el AndroidManifest.xml, sin embargo, no tiene asociada ninguna acción. Para que responda a la llegada de SMS es **necesario asignarle la acción android.provider.Telephony.SMS_RECEIVED**. Así pues, el registro del BroadcastReceiver debe quedar:

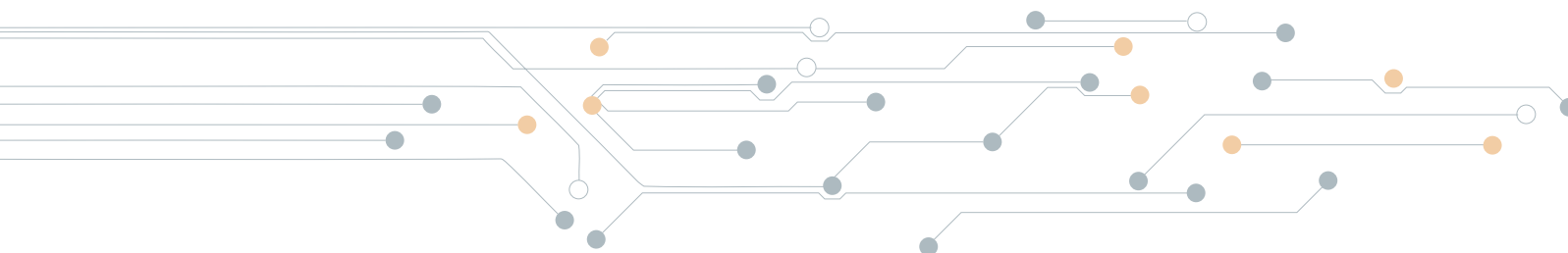
```
<receiver  
    android:name=".ReceptorSms"  
    android:enabled="true"  
    android:exported="true" >  
    <intent-filter>  
        <action android:name="android.  
provider.Telephony.SMS_RECEIVED" />  
    </intent-filter>  
</receiver>
```

Además del registro del BroadcastReceiver, para poder realizar la lectura de los SMS es necesario asignar el siguiente permiso en el AndroidManifest.xml:

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

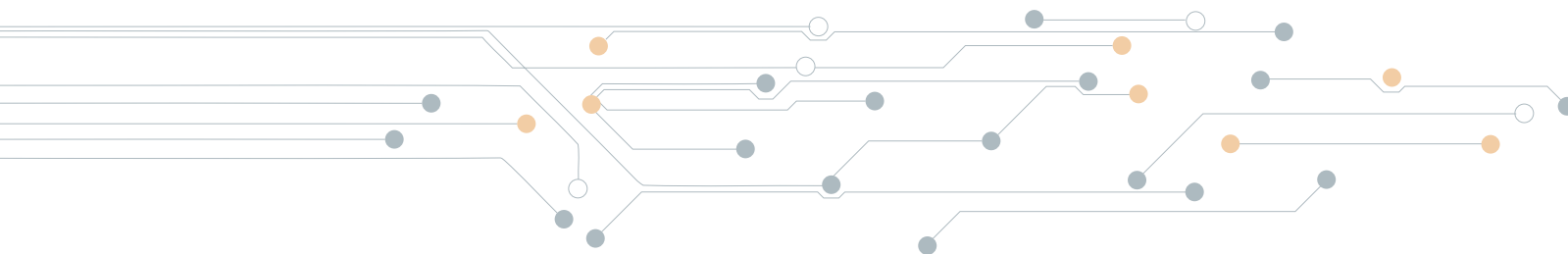
En cuanto a las actividades, la actividad principal simplemente incluye un botón para lanzar la actividad de listado. El código de la misma será el siguiente:

```
public class MainActivity extends AppCompatActivity {  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
    }  
    public void mostrarMensajes(View v){  
        Intent intent=new Intent(this,ListadoActivity.class);  
        this.startActivity(intent);  
    }  
}
```



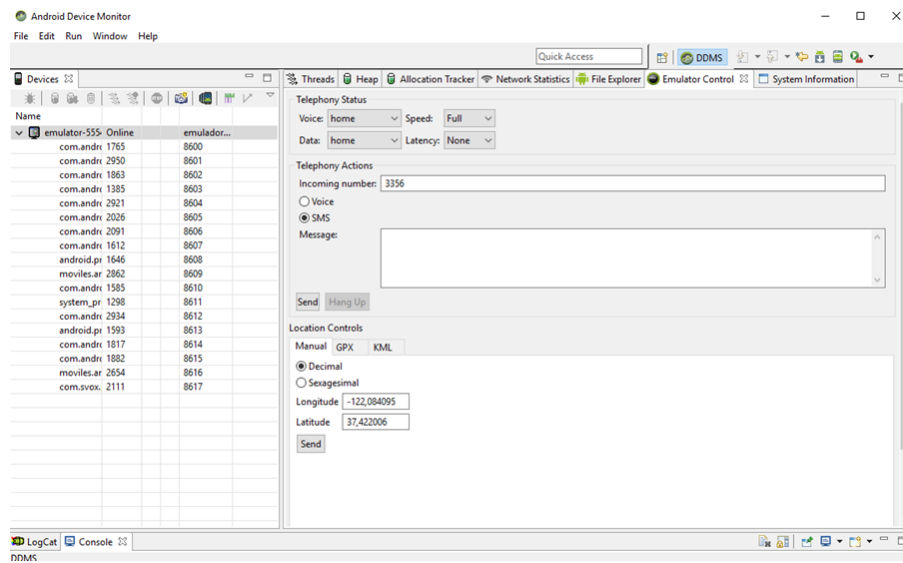
El siguiente listado corresponderá a la actividad de listado de mensajes:

```
public class ListadoActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_listado);
        //instancia GestionFichero
        GestionFichero af=new GestionFichero(this,"mensajes.
txt");
        //Recuperamos los mensajes almacenados
        ArrayList<String> mensajes=af.recuperarMensajes();
        ListView lvNotas=(ListView)this.findViewById(R.
id.lvMensajes);
        //creamos un arrayadapter con los mensajes
        //y lo asignamos a la lista
        ArrayAdapter<String> adapter= new
ArrayAdapter<String>(this, android.R.layout.simple_list_item_1,
mensajes);
        lvNotas.setAdapter(adapter);
    }
    public void volver(View v){
        this.finish();
    }
}
```



Para probar el proyecto en el emulador, podemos simular el envío de sms a través del Android Device Monitor, en la pestaña "Emulator Control". En la casilla "Incoming number" introduciremos el número de teléfono del remitente, mientras que en la caja de texto "message" escribiremos el texto del mensaje, marcando previamente la opción "SMS" en el grupo de radiobutton:

El mismo cuadro de diálogo puede utilizarse para simular llamadas de teléfono.



4. Envío de SMS desde aplicaciones Android

Aprovechando que hemos estado tratando en el ejercicio anterior la manera de acceder a la información de los SMS recibidos, vamos a ver en este punto como enviar SMS desde una aplicación Android.

La clave está en la utilización de la clase **SmsManager** del paquete `android.telephony`. Esta clase no dispone de constructor, para obtener un objeto de la misma utilizaremos su método estático `getDefault()` que nos devuelve una implementación por defecto:

```
SmsManager sm=SmsManager.getDefault();
```

El envío de los mensajes de texto se realizará utilizando el método `sendTextMessage()`, cuyo formato es el siguiente:

```
sendTextMessage(String destinationAddress, String scAddress, String text,  
PendingIntent sentIntent, PendingIntent deliveryIntent)
```

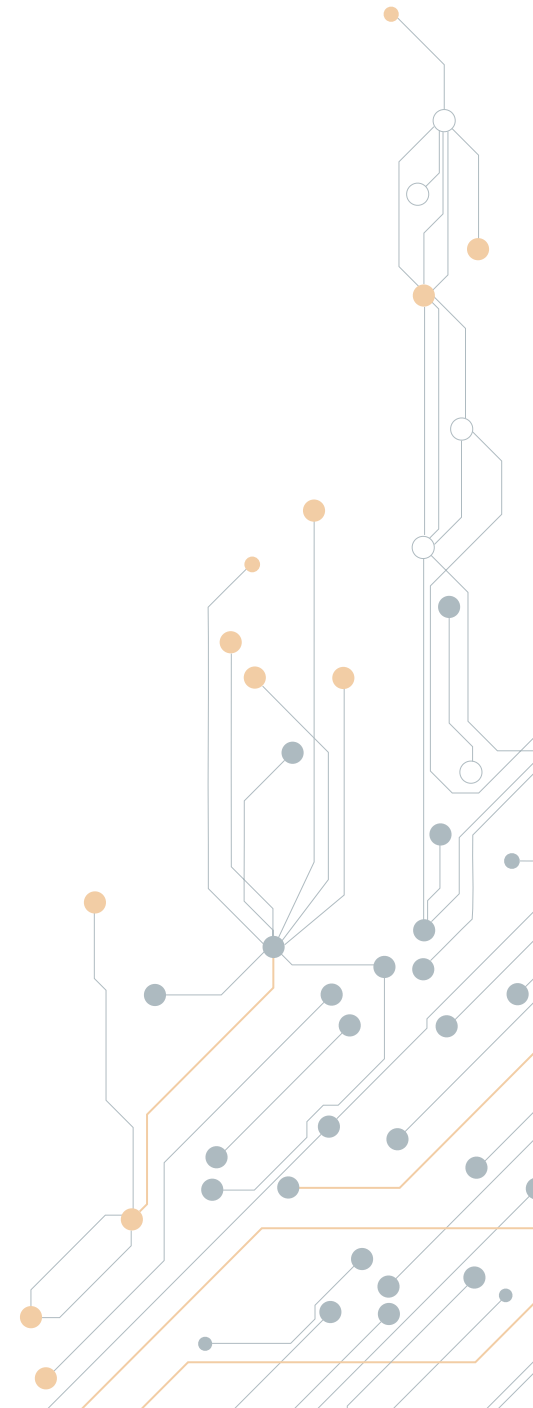
El significado de los parámetros es el siguiente:

- **destinationAddress.** Número de teléfono del destinatario del mensaje
- **scAddress.** Dirección del centro de servicio. Utilizaremos null para usar el centro por defecto.
- **text.** Texto del mensaje
- **sentIntent.** PendingIntent que será procesado cuando el mensaje se envíe. Un PendingIntent encapsula un Intent con una acción asociada, como lanzamiento de una actividad o notificación de suceso.
- **deliveryIntent.** PendingIntent que será procesado cuando el mensaje llegue al destino.

Por ejemplo, para enviar un mensaje de saludo al número 1111, sería tan simple como esto:

```
sm.sendMessage("1111", null, "mensaje de saludo", null, null);
```

Hay que tener en cuenta que para que la aplicación pueda enviar mensajes necesita el permiso **android.permission.SEND_SMS**.



5. BroadcastReceiver en línea

Se puede definir un BroadcastReceiver dentro de otro componente, por ejemplo una actividad, a través de una clase anónima, sin tener que crear una clase independiente. Esto permite al método *onReceive()* del BroadcastReceiver acceder a los componentes de la actividad.

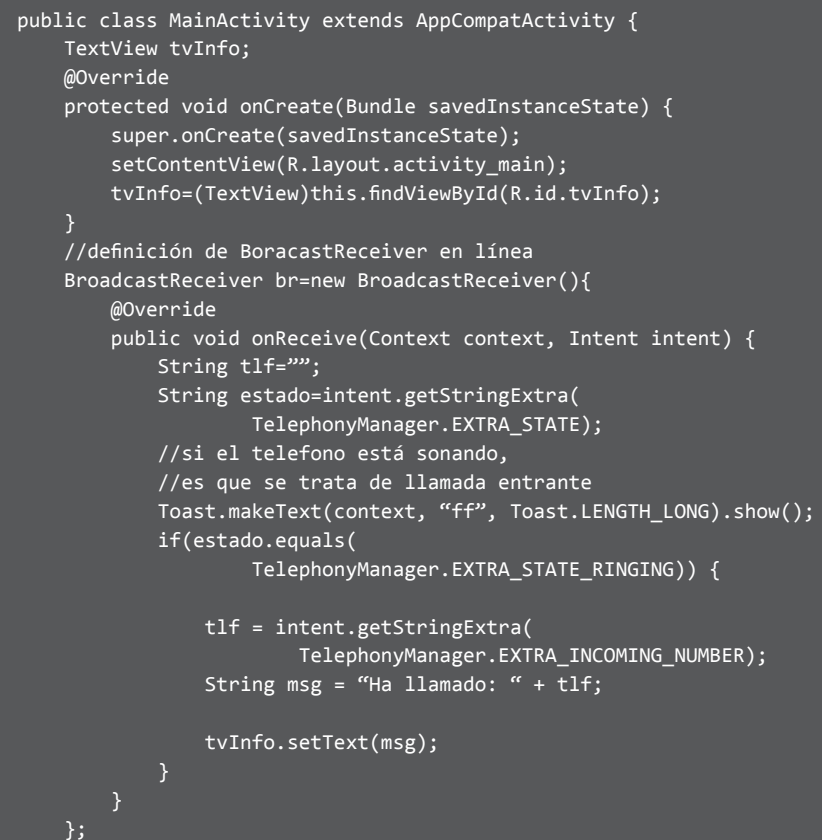
Dado que se define como una clase anónima, un BroadcastReceiver en línea no se puede registrar a través del archivo de manifiesto, habrá que hacerlo desde código a través del método *registerReceiver()* de Context, y cuyo formato es el siguiente:

```
registerReceiver(BroadcastReceiver receiver, IntentFilter filter)
```

El primer parámetro recibido por el método es el objeto BroadcastReceiver, mientras que el segundo representa el IntentFilter asociado a la acción con la que se quiere registrar. Para crear un objeto android.content.IntentFilter utilizaremos el siguiente constructor:

```
IntentFilter(String action)
```

El siguiente código corresponde a una actividad con un campo de texto TextView en el que mostramos un mensaje con el número de teléfono entrante cada vez que llega una llamada al terminal:



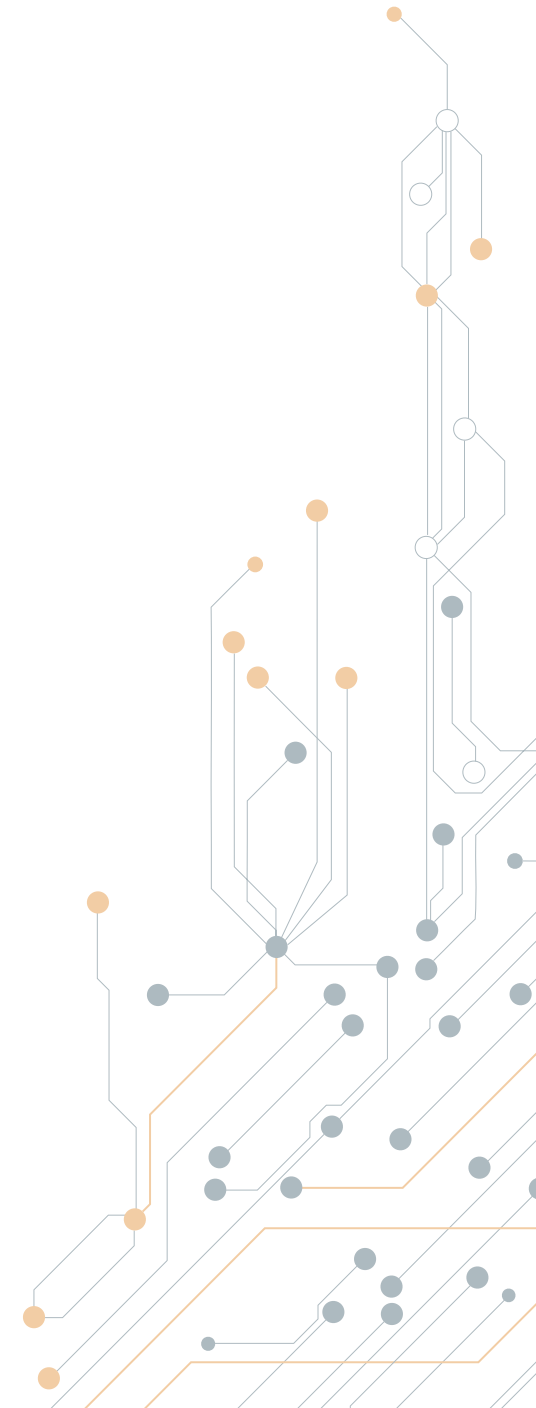
```
public class MainActivity extends AppCompatActivity {
    TextView tvInfo;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        tvInfo=(TextView)this.findViewById(R.id.tvInfo);
    }
    //definición de BroadcastReceiver en línea
    BroadcastReceiver br=new BroadcastReceiver(){
        @Override
        public void onReceive(Context context, Intent intent) {
            String tlf="";
            String estado=intent.getStringExtra(
                TelephonyManager.EXTRA_STATE);
            //si el telefono está sonando,
            //es que se trata de llamada entrante
            Toast.makeText(context, "ff", Toast.LENGTH_LONG).show();
            if(estado.equals(
                TelephonyManager.EXTRA_STATE_RINGING)) {

                tlf = intent.getStringExtra(
                    TelephonyManager.EXTRA_INCOMING_NUMBER);
                String msg = "Ha llamado: " + tlf;

                tvInfo.setText(msg);
            }
        }
    };
};
```



```
@Override
protected void onResume() {
    // registramos el receiver antes de activarse
    //la actividad
    super.onResume();
    this.registerReceiver(br, new IntentFilter("android.
intent.action.PHONE_STATE"));
}
}
```

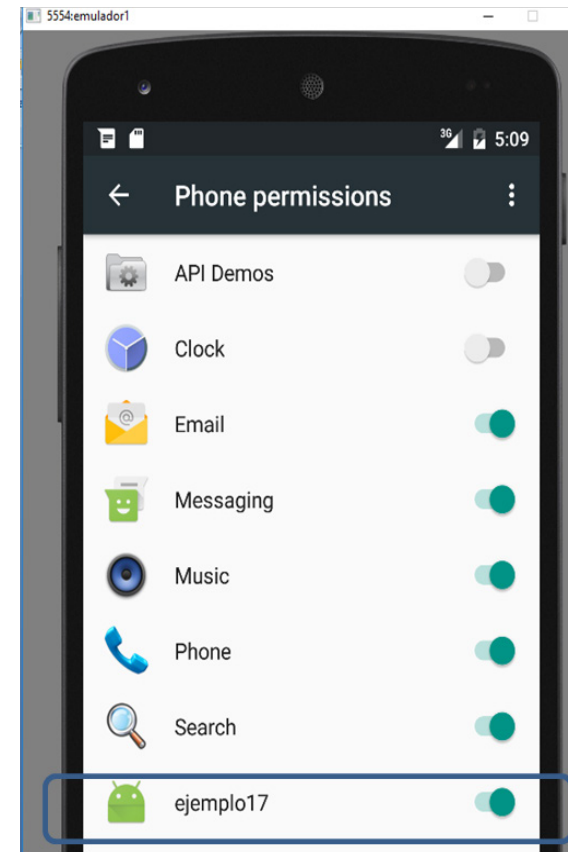


Según podemos apreciar en este programa, el registro del BroadcastReceiver lo hacemos en el método *onResume()*, mientras que en *onPause()* lo desregistramos a través del método *onRegisterReceiver()* de Context. Esto hace que el BroadcastReceiver solo esté escuchando sucesos de llamada mientras la actividad esté activa.

Al igual que con los BroadcastReceiver independientes, es necesario otorgar a la aplicación los permisos necesarios, en este caso, el de lectura del estado del teléfono:

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

Si a pesar de asignar el permiso en el archivo de manifiesto la aplicación no funciona en el emulador, será necesario entrar en las propiedades de las aplicaciones y activar el permiso del teléfono para nuestra aplicación:



6. Notificación de eventos

Desde una aplicación Android también se pueden notificar eventos para que sean capturados por BroadcastReceivers que hayan sido registrados para tal fin en otras aplicaciones instaladas en el terminal.

El proceso no es muy diferente al de lanzar una actividad; se debe crear un Intent a partir de la acción asociada al evento y después llamar al método *sendBroadcast()* de Context para lanzarlo.

El siguiente código realizaría el lanzamiento de una acción de nombre "com.ejemplo.accion1" y enviaría como dato extra en el intent un valor numérico:

```
Intent in=new Intent("com.ejemplo.accion1");  
in.putExtra("dato", 3000);  
this.sendBroadcast(in);
```

Todo BroadcastReceiver existente en alguna de las aplicaciones instaladas en el terminal, y que esté registrado con la acción "com.ejemplo.accion1", responderá a este suceso.

De cara a definir los nombres de las acciones debemos ser cuidadosos a fin de no repetir nombres. Por ello, suele utilizarse como convenio el siguiente formato de nombre:

```
com.nombre_proyecto.nombre_accion
```

Telefonica

EDUCACIÓN DIGITAL