Telefonía (1)

Aunque hoy en día Android se puede ejecutar en diversos dispositivos, su origen está en los teléfonos móviles y por tanto debe disponer de herramientas para gestionar la telefonía, permitiendo realizar las siguientes operaciones:

- Obtener características del teléfono.
- Gestionar llamadas telefónicas.
- Gestionar mensajes de texto (SMS).

71

Telefonía (2)

- Obtener características del teléfono

Por medio de la clase *TelephonyManager* podemos recuperar información relativa al dispositivo:

- Tipo de teléfono: CDMA, GSM, SIP, NONE.
- Código IMEI del teléfono.
- Número de teléfono.

72

Telefonía (3)

- Obtener características del teléfono

En primer lugar debemos solicitar el permiso READ_PHONE_STATE en nuestro Manifiesto.

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

A continuación, crearemos un objeto de la clase *TelephonyManager*, mediante *getSystemService*.

```
TelephonyManager tm =
(TelephonyManager)getSystemService(Context.TELEPHONY_SERVICE);
```

A partir de este objeto podemos acceder a toda la información a través de sus métodos *get*:
https://developer.android.com/reference/android/telephony/TelephonyManager.html

73

Telefonía (4)

- Obtener características del teléfono

Por ejemplo, podemos obtener el tipo de teléfono mediante *getPhoneType*.

```
int tipoTelefono = tm.getPhoneType();
```

Esté método devuelve un entero, cuyos posibles valores son:

- 0: PHONE_TYPE_NONE
- 1: PHONE_TYPE_GSM
- 2: PHONE_TYPE_CDMA
- 3: PHONE_TYPE_SIP

74

Telefonía (5)

- Obtener características del teléfono

Para obtener el nombre del operador utilizaremos *getSimOperatorName*.

```
String operador = tm.getSimOperatorName();
```

Algunos de los valores (número de teléfono, IMEI, etc.) son considerados información sensible, por lo que antes de acceder a ellos deberemos comprobar que disponemos del permiso READ_PHONE_STATE.

```
if (ContextCompat.checkSelfPermission(MainActivity.this,
    Manifest.permission.READ_PHONE_STATE)
    == PackageManager.PERMISSION_GRANTED) {
       String numeroTlf = tm.getLine1Number();
}
```

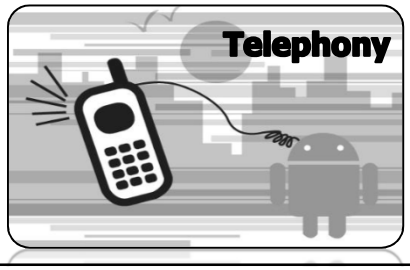
75

Telefonía (6)

- Gestión de llamadas y mensajes

La API de Android que gestiona la voz y los datos (llamadas, SMS, datos, etc.) se llama *Telephony*.

En el comienzo de Android todos los dispositivos disponían de esta API, pero desde la aparición de dispositivos equipados con Wi-Fi, el uso de esta API depende del dispositivo.



76

Telefonía (7)

- Gestión de llamadas y mensajes

Si nuestra aplicación sólo puede utilizarse en dispositivos que dispongan de esta API, deberemos especificarlo en el manifiesto para que *Google Play* no la ofrezca en dispositivos que no la tengan.

```
<uses-feature android:name="android.hardware.telephony" android:required="true"/>
```

Si el uso de la telefonía no es imprescindible para la aplicación, podemos comprobar dinámicamente la ausencia de las funcionalidades para desactivarlas.

```
PackageManager pm = getPackageManager();
boolean hasTelephony = pm.hasSystemFeature(pm.FEATURE_TELEPHONY);
```

77

Telefonía (8)

- Realizar llamadas

Para realizar una llamada desde nuestra aplicación basta con crear un *Intent* cuya acción sea ACTION_DIAL o ACTION_CALL, y como dato reciba un objeto Uri que contenga el prefijo "tel:" seguido del número al que queremos llamar.

```
Uri tlf = Uri.parse("tel:" + numTel);
Intent call = new Intent(Intent.ACTION_DIAL, tlf);
startActivity(call);
```

NOTA: el uso de la acción ACTION_CALL requiere del permiso CALL_PHONE (Manifiesto).

78

Telefonía (9)

- Gestionar llamadas entrantes

En ocasiones nuestras aplicaciones necesitarán que se les notifique la recepción de una llamada para poder realizar operaciones como guardar cambios o el progreso de un juego, etc.

Para ello podemos crear una clase que herede de *PhoneStateListener* y sobrecargar el método *onCallStateChanged*, que notificará del cambio de estado de las llamadas permitiendo reaccionar a cada uno de esos estados:

- Inactivo: CALL_STATE_IDLE
- Sonando: CALL_STATE_RINGING
- Descolgado/Comunicando: CALL_STATE_OFFHOOK

79

Telefonía (10)

- Gestionar llamadas entrantes

```
private class MyPhoneListener extends PhoneStateListener
{
    @Override
    public void onCallStateChanged(int state, String phoneNumber)
    {
        switch (state) {
            case TelephonyManager.CALL_STATE_RINGING:
                // Sonando
                break;
            case TelephonyManager.CALL_STATE_OFFHOOK:
                // Descolgado/Comunicando
                break;
            case TelephonyManager.CALL_STATE_IDLE:
                // Inactivo
                break;
        }
    }
}
```

80

Telefonía (11)

- Gestionar llamadas entrantes

En la actividad en la que queramos gestionar las llamadas entrantes tendremos que crear un objeto de la clase que hemos creado, y un objeto *TelephonyManager* al que añadiremos el método *listen* para que escuche los cambios de estado de nuestro objeto.

```
// Gestión de llamadas entrantes
MyPhoneListener phoneListener = new MyPhoneListener();
TelephonyManager tm =
  (TelephonyManager) this.getSystemService(Context.TELEPHONY_SERVICE);
tm.listen(phoneListener, PhoneStateListener.LISTEN_CALL_STATE);
```

Este *listener* sólo funcionará cuando nuestra aplicación se esté ejecutando.

81

Telefonía (12)

- Gestionar llamadas entrantes

Por último, no debemos olvidar añadir el permiso de lectura de estados READ_PHONE_STATE a nuestro archivo de manifiesto.

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

También podemos supervisar las llamadas entrantes utilizando un Broadcast Receiver.

```
<receiver android:name="MyPhoneStateReceiver">
    <intent-filter>
        <action android:name="android.intent.action.PHONE_STATE"/>
    </intent-filter>
</receiver>
```

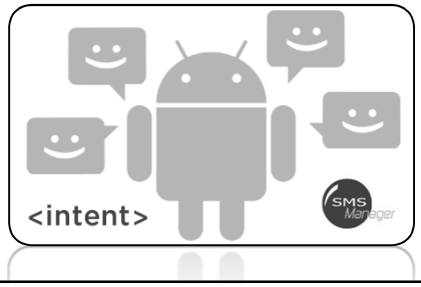
82

Telefonía (13)

- Enviar SMS

Una aplicación Android puede enviar mensajes a través de dos mecanismos:

- Utilizando un *Intent* que invoque a una aplicación de mensajería.
- Utilizando la clase *SMSManager*.



83

Telefonía (14)

- Enviar SMS

El primer método consiste en crear un *Intent* con los parámetros *ACTION_SENDTO* y una URI con el número del destinatario: "sms:" + número.

Además podemos añadir el contenido del SMS mediante *putExtra*.

Este método no necesita ningún permiso porque no es nuestra aplicación la que realiza el envío del mensaje, sino la aplicación de SMS por defecto.

```
Uri tlf = Uri.parse("sms:123456789");
Intent sendSMS = new Intent(Intent.ACTION_SENDTO, tlf);
sendSMS.putExtra("sms_body", "Texto del mensaje");
startActivity(sendSMS);
```

84

Telefonía (15)

- Enviar SMS

El segundo método consiste en enviar los mensajes directamente desde la aplicación utilizando la clase *SmsManager*, y por tanto necesitaremos el permiso *SEND_SMS*.

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

En primer lugar obtendremos una instancia de la clase *SmsManager* mediante *getDefault* porque es una clase *singleton*, y a continuación utilizaremos el método *sendTextMessage* para enviar el SMS.

```
String tlf = "123456789";
String mensa = "Texto del mensaje";
SmsManager smsm = SmsManager.getDefault();
smsm.sendTextMessage(tlf, null, mensa, null, null);
```

Telefonía (16)

- Enviar SMS

El método *sendTextMessage* recibe los siguientes parámetros:

- Destino (*String*): destinatario del mensaje.
- Servicio (*String*): servicio utilizado para el envío del mensaje, mediante *null* se establece que se utilice el servicio por defecto.
- Mensaje (*String*): texto del mensaje.
- Enviado (*PendingIntent*): si no es nulo, se ejecutará cuando el mensaje es enviado indicando éxito o fracaso.
- Recibido (*PendingIntent*): si no es nulo, se ejecutará cuando el mensaje es recibido por el destinatario.

Telefonía (17)

- Enviar SMS

El tamaño de los mensajes está limitado a 160 caracteres. Si queremos enviar un mensaje mayor tendremos que dividirlo en partes mediante *divideMessage* de recibe un *String* y devuelve un *ArrayList* de *String* de tamaño adecuado, y una vez dividido utilizaremos *sendMultipartTextMessage* que funciona igual que *sendTextMessage* sustituyendo el texto del mensaje por el *ArrayList*.

```
String tlf = "123456789";
String mensa = "Texto con más de 160 caracteres... ";
SmsManager smsm = SmsManager.getDefault();
ArrayList<String> textoDividido = smsm.divideMessage(textoSMS);
smsm.sendMultipartTextMessage(tlf, null, textoDividido, null, null);
```

87

Telefonía (18)

- Recibir SMS

Una aplicación puede reaccionar ante la recepción de un mensaje mediante un *Broadcast Receiver*.

Para ello incluiremos en el archivo de manifiesto tanto el permiso *RECEIVE_SMS*, como un receiver en el que indicaremos la clase encargada de su gestión.

```
<application
   ...
   <receiver android:name="MySMSReceiver">
      <intent-filter>
         <action android:name="android.provider.Telephony.SMS_RECEIVED"/>
      </intent-filter>
   </receiver>
</application>
<uses-permission android:name="android.permission.RECEIVE_SMS" />
```

88

Telefonía (19)

- Recibir SMS

Una vez declarado el *receiver* implementaremos la clase *MySMSReceiver*, encargada de la gestión de la recepción de mensajes.

Esta clase extiende a *BroadcastReceiver* y tendrá que sobrecargar su método *onReceive*.

Los mensajes recibidos se obtienen mediante el extra "pdu" del *Intent* en forma de lista de Objetos. Recorreremos esta lista e iremos obteniendo cada SMS en un objeto *SmsMessage* mediante el método *createFromPdu*, y accederemos a la info del SMS:

- Origen: *getDisplayOriginatingAddress*
- Texto: *getDisplayMessageBody*

89

Telefonía (20)

- Recibir SMS

La clase *MySMSReceiver* sería similar a la siguiente:

```
public class MySMSReceiver extends BroadcastReceiver {
  @Override
  public void onReceive(Context context, Intent intent) {
    Bundle extras = intent.getExtras();
    if (extras != null) {
      Object[] pdus = (Object[])extras.get("pdus");
      for (int i=0; i<pdus.length; i++) {
        SmsMessage sms = SmsMessage.createFromPdu((byte[])pdus[i]);
        String numTlf = sms.getDisplayOriginatingAddress();
        String textoSMS = sms.getDisplayMessageBody();
        String mensa = numTlf + ": " + textoSMS;
        int duration = Toast.LENGTH_LONG;
        Toast toast = Toast.makeText(context, mensa, duration);
        toast.show();
      }
    }
  }
}
```

90

Telefonía (21)

- Recibir SMS

El método *createFromPdu* fue modificado a partir de Android 6 (*Marshmallow*), así que sería conveniente comprobar la versión y hacer la llamada adecuada.

```
if (Build.VERSION.SDK_INT>= Build.VERSION_CODES.M) {
  String format = extras.getString("format");
  sms = SmsMessage.createFromPdu((byte[])pdus[i],format);
} else {
  sms = SmsMessage.createFromPdu((byte[])pdus[i]);
}
```

91

Telefonía (22)

- Leer SMS

También podemos acceder a los SMS que tenemos almacenados en nuestro dispositivo.

Para ello, crearemos un cursor realizando una *query* sobre *getContentResolver*, utilizando un objeto Uri con el contenido que deseemos como parámetro:

- Todos los SMS: "*content://sms/*"
- SMS recibidos: "*content://sms/inbox*"
- SMS enviados: "*content://sms/sent*"
- Borradores: "*content://sms/draft*"

Por último, recorreremos el cursor y utilizaremos los métodos: *getColumnCount*, *getColumnName* y *getString*, para acceder a la información.

92

Telefonía (23)

- Leer SMS

```
Uri uri = Uri.parse("content://sms/inbox");
Cursor cur = getContentResolver().query(uri, null, null, null, null);
if (cur.moveToFirst()) {
  do {
    // Mostrar todos los campos del SMS
    String msg = "";
    for (int i = 0; i < cur.getColumnCount(); i++) {
      msg = msg+i+" => "+cur.getColumnName(i)+": "+cur.getString(i)+"\n";
    }
    Toast.makeText(MainActivity.this, msg, Toast.LENGTH_SHORT).show();
  } while (cur.moveToNext());
}
```

Para poder leer los SMS tendremos que solicitar el permiso READ_SMS:

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

93