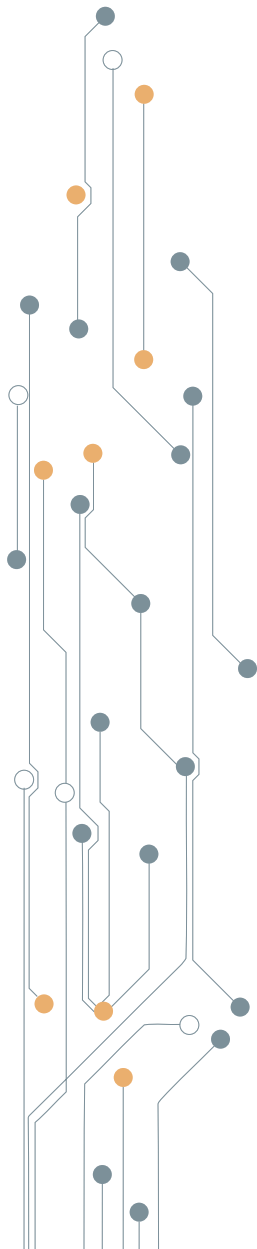




# Comunicaciones

*Telefonica*  
EDUCACIÓN DIGITAL

# Índice



## Comunicaciones

1   Acceso a recursos Web	3
1.1   Los paquetes java.net y java.io	3
1.2   Conexión con el recurso	3
1.3   Intercambio de datos con el recurso	4
1.4   Comunicación en modo asíncrono. La clase AsyncTask	5
2   Acceso a un servicio Web	10
2.1   Acceso a servicios Web XML	11
2.2   Servicios Web JSON	15
3   Comunicaciones mediante sockets	20
3.1   La clase ServerSocket	21
3.2   La clase Socket	22
4   Geolocalización	31
4.1   LocationManager	32
4.2   Obtención de la posición	33
4.3   Cambios de localización	34
5   Google Maps	43
5.1   Google play service	43
5.2   Creación del proyecto para google maps	57
5.3   Utilización API google maps	65

# 1. Acceso a recursos Web

Por recurso Web entendemos cualquier elemento o componente expuesto por una aplicación que se ejecuta sobre un servidor Web. Estos recursos pueden ser páginas HTML, documentos XML, imágenes, etc., y están identificados por una dirección o URL a la que se accede a través del protocolo HTTP.

Por ejemplo, la página inicial de google que se encuentra en la dirección `http://www.google.es` es un ejemplo de recurso Web al que podemos acceder desde una aplicación Android

## 1.1 | Los paquetes `java.net` y `java.io`

Estos dos paquetes, que son adaptaciones de Java SE para Android, proporcionan las clases necesarias para conectarnos con un recurso y recuperar su contenido

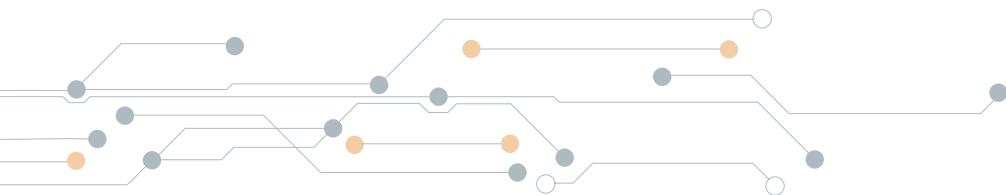
## 1.2 | Conexión con el recurso

En primer lugar, debemos establecer la conexión con el recurso a partir de la URL que lo identifica. La clase `URL` del paquete `java.net` permite crear un objeto `URL` a partir de la cadena de caracteres que representa dicha dirección:

```
URL url=new URL("http://www.google.es");
```

Una vez creado el objeto, la conexión con el recurso se establece a través del método `openConnection()` de la clase `URL`. Este método devuelve un objeto `URLConnection` que representa la conexión con el recurso:

```
URLConnection con=url.openConnection();
```



## 1.3 | Intercambio de datos con el recurso

Una vez que disponemos del objeto `URLConnection` podemos hacer uso de sus métodos para obtener los objetos de entrada y salida con los que poder intercambiar datos con el recurso. Estos métodos son:

- **`getInputStream()`**. Devuelve un objeto `java.io.InputStream` para poder leer la información de la URL. A partir de este objeto podemos crear otros objetos del `java.io`, como `BufferedReader`, que proporcionan métodos para leer cadenas de datos.
- **`getOutputStream()`**. Devuelve un objeto `java.io.OutputStream`, con el que podemos crear objetos `PrintStream` para realizar operaciones de escritura. Dicha clase nos proporciona un juego de métodos `print()` y `println()` por cada tipo de dato Java, con los que podemos enviar información desde nuestra aplicación al recurso.

Por ejemplo, si el recurso fuera un página Web como en el ejemplo indicado antes, el siguiente bloque de instrucciones crearía una cadena de caracteres con el contenido HTML completo de la página:

```
String contenido="";
String cad;
InputStream is=con.getInputStream();
//creación objeto BufferedReader
BufferedReader bf=new BufferedReader(new
InputStreamReader(is));
//va leyendo línea a línea hasta el final
//y concatena las líneas leídas en una cadena
while((s=bf.readLine())!=null){
    cad+=s;
```

Sería lógico pensar, que las instrucciones explicadas en el ejemplo se deberían incluir en alguno de los métodos de ciclo de vida de la actividad o en los de respuesta a alguno de los eventos de la interfaz. Pero no es así, si hacemos esto, **se producirá un error en tiempo de ejecución**.

El motivo es que desde el hilo principal de la actividad no se puede realizar una tarea que pueda bloquear dicho hilo. Esto significa que las tareas de comunicación con un proceso externo deberán realizarse en un hilo independiente, para lo cual contamos con la

## 1.4 | Comunicación en modo asíncrono. La clase AsyncTask

Esta clase, que forma parte del paquete `android.os`, permite definir tareas en hilos independientes para ser ejecutadas de forma asíncrona. Para ello, debemos extender dicha clase de la siguiente manera:

```
class ClaseTarea extends AsyncTask<Tipo_
parametros, Tipo_progreso, Tipo_
resultado>
```

Entre los símbolos < y > se deben especificar los siguientes tipos Java:

- **Tipo\_parametros.** Es el tipo de los parámetros que se pasarán a la tarea cuando se le dé el orden de ejecución.

- **Tipo\_progreso.** Es el tipo de datos devuelto durante la ejecución de la tarea. Y es que, en determinados momentos durante la ejecución de la misma, se puede consultar a la tarea pasándole unos datos adicionales cuyo tipo será el aquí especificado

- **Tipo\_resultado.** Es el tipo de dato del resultado devuelto por la tarea.

El código con las instrucciones a realizar por la tarea las implementaremos en el método `doInBackground()`, definido como abstracto en `AsyncTask` y cuyo formato es el siguiente:

```
Tipo_resultado doInBackground(Tipo_
parametros... params)
```

El método recibirá un array de parámetros cuyo tipo será el indicado en *Tipo\_parametros* y devolverá un resultado del tipo indicado en *Tipo\_resultado*.

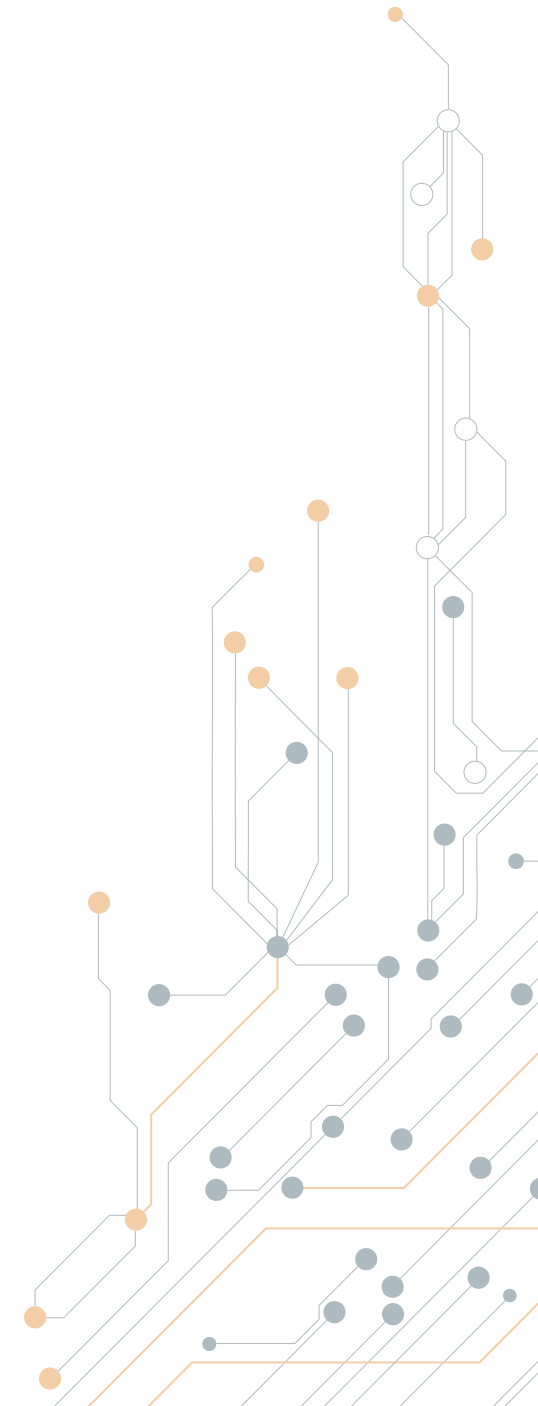
Si la tarea debe modificar algún componente de la interfaz gráfica, esto no podrá hacerse desde el método `doInBackground()`, pues **Android no permite modificar componentes de la interfaz desde un hilo distinto** al de la actividad propietaria.

Si necesitamos hacer esto, tendremos que sobrescribir este otro método de `AsyncTask`:

```
void onPostExecute(Tipo_resultado res)
```

Este método es invocado cuando la tarea se ha completado, es decir, después de finalizar la ejecución de *doInBackground()*, y recibe como parámetro el resultado generado durante la ejecución de la tarea. Como el hilo asociado a la tarea ya ha finalizado, desde **este método si podemos modificar** los componentes gráficos.

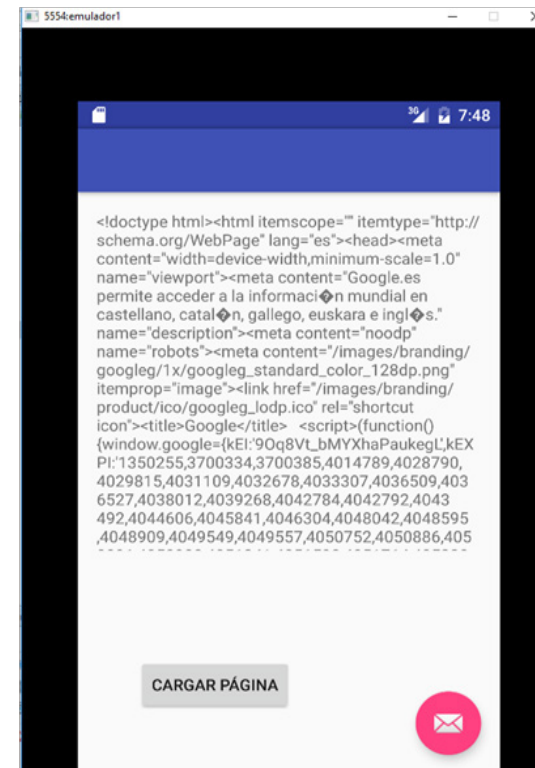
Finalmente, para poner en ejecución una tarea, se debería crear un objeto de la subclase *AsyncTask* y llamar al método *execute()* heredado de esta. Al método *execute()* se le pasarán los parámetros que queremos enviar a la actividad y que, como hemos indicado anteriormente, son recibidos en el método *doInBackground()*.



## Ejercicio resuelto

Para comprobar cómo se comunica una aplicación Android con un recurso Web y aclara todo lo comentado respecto a la comunicación en segundo plano, vamos a realizar un sencillo ejercicio que consiste en una actividad formada por un botón y un TextView.

Al pulsar el botón, la aplicación se conectará con el servidor de google y mostrará el contenido de la página principal en el campo de texto. Lógicamente, mostrará el contenido HTML de la misma, sin interpretar las etiquetas. Su aspecto será similar al que se indica en la siguiente imagen:



A continuación, te presentamos el código de esta actividad:

```

public class MainActivity extends AppCompatActivity {

    TextView tvPagina;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        //referencia al TextView
        tvPagina=(TextView)findViewById(R.id.tvPagina);
    }
    //método de respuesta al click del botón
    public void cargar(View v){
        //iniciar tarea en segundo plano
        ComunicacionTask com=new ComunicacionTask();

        //le pasa como parámetro la dirección
        //de la página
        com.execute("http://www.google.es");

    }
    //el primer tipo es lo que recibirá el doInBackground,
    // que en nuestro caso es la dirección a la que se
    //tiene que conectar. El tercer tipo
    //es el resultado que entregará la tarea, y que en este caso
    //es la página recuperada

```

```

private class ComunicacionTask extends
    AsyncTask<String, Void, String> {

    @Override
    protected String doInBackground(String... params) {
        String pagina="";
        try{
            URL url=new URL(params[0]);
            URLConnection con=url.openConnection();
            //recuperacion de la página línea a línea
            String s;
            InputStream is=con.getInputStream();
            BufferedReader bf=new BufferedReader(
                new InputStreamReader(is));
            while((s=bf.readLine())!=null){
                pagina+=s;
            }
        }
        catch(IOException ex){
            ex.printStackTrace();
        }

        return pagina;
    }

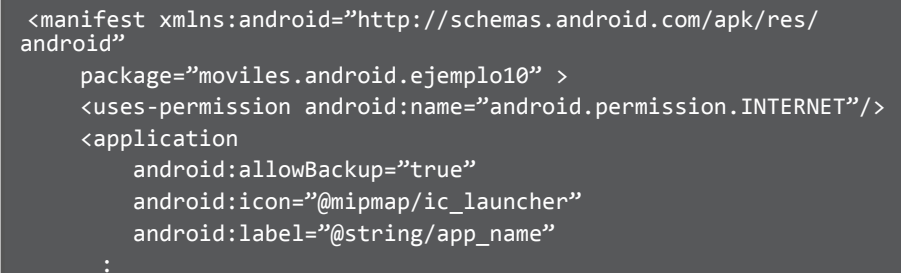
    @Override
    protected void onPostExecute(String result) {
        //volcar el resultado en el TextView
        tvPagina.setText(result);
    }
}

```



Como vemos, la clase que define la tarea asíncrona es definida como una clase interna dentro de la actividad, esto permite al *método `onPostExecute()`* acceder al atributo de la clase de actividad donde se guarda en objeto `TextView`.

Para que el programa funcione, es necesario asignar el permiso de acceso a Internet dentro del archivo de manifiesto:

A screenshot of a code editor window showing the AndroidManifest.xml file. The window has a title bar with standard minimize, maximize, and close buttons. The code is as follows:

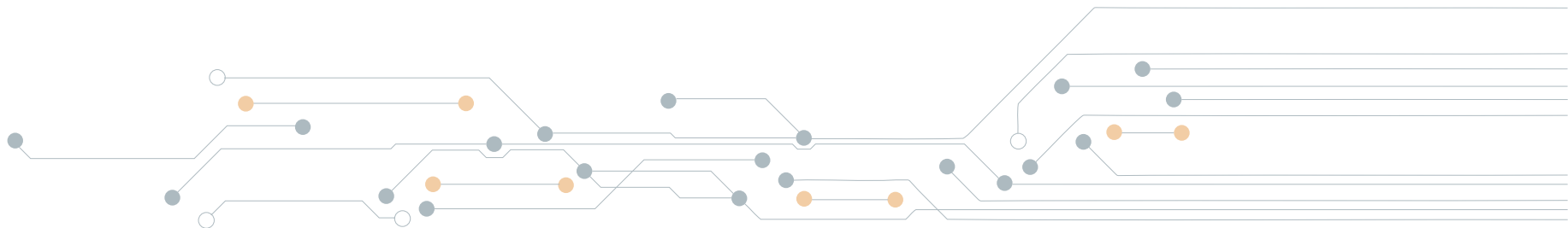
```
<manifest xmlns:android="http://schemas.android.com/apk/res/
android"
    package="moviles.android.ejemplo10" >
    <uses-permission android:name="android.permission.INTERNET"/>
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        :
    >
```

## 2. Acceso a un servicio Web

Un servicio Web es una aplicación que sirve datos a otras aplicaciones a través de la Web. Por tanto, a diferencia de las aplicaciones Web estándares los servicios Web no están orientados a generar páginas Web para navegadores, sino a ofrecer datos a otros programas que pueden utilizarlos para cualquier fin.

De cara a conseguir que los datos puedan ser manipulados por cualquier aplicación, independientemente de cómo estén implementados el servicio y el cliente, se emplea el estándar XML para marcar dichos datos. Actualmente, se está utilizando también el **formato JSON** (JavaScript Object Notation) para el envío de los datos desde el servicio, ya que reduce el número de bytes de meta información enviada por la red.

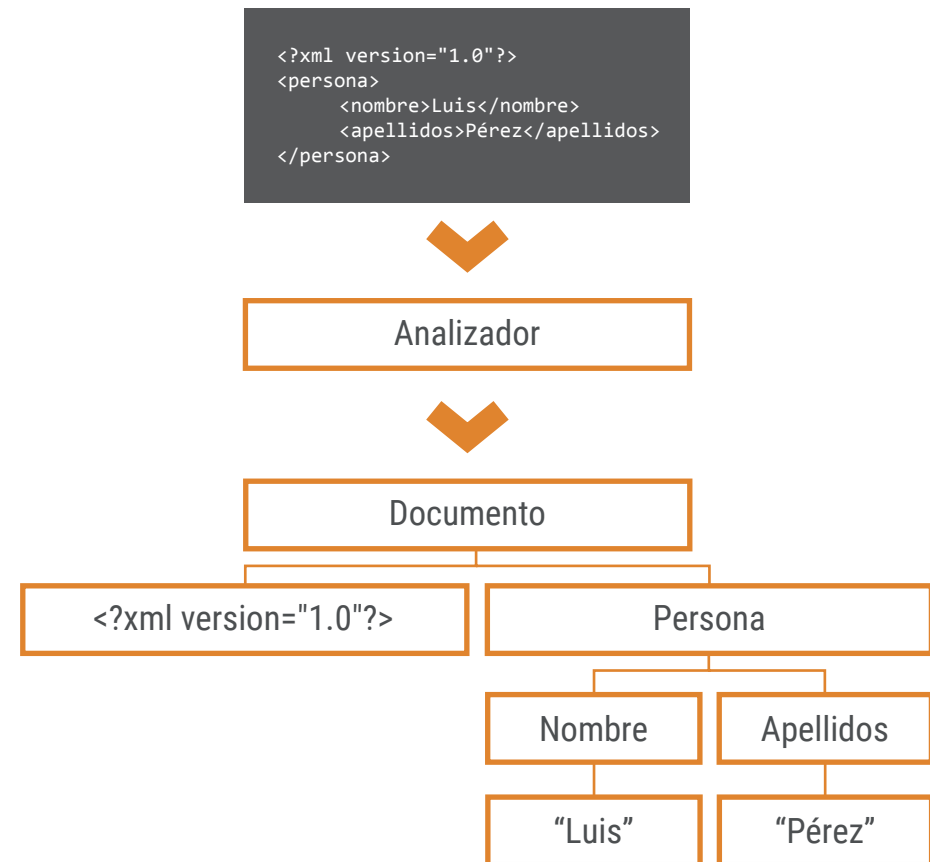
Entre estos programas clientes de servicios Web están las aplicaciones Android. Una aplicación Android podría conectarse, por ejemplo, a un servicio Web de información meteorológica para ofrecer dicha información de forma personalizada al usuario en el terminal, a una fuente RSS, que es un tipo de servicio Web, para poder tener acceso a noticias en tiempo real. Las posibilidades que ofrecen los servicios Web al mundo de las aplicaciones móviles es enorme.



## 2.1 | Acceso a servicios Web XML

Como se ha indicado, un servicio Web puede entregar datos en formato XML o JSON. En el caso de los servicios Web XML, para acceder a los datos necesitaríamos utilizar un **analizador DOM** que analizase el documento XML y lo convirtiera en un árbol de objetos, siguiendo el estándar DOM, para poderlo manipular desde la aplicación.

En Android disponemos de los paquetes `javax.xml.parsers` y `org.w3c.dom`. El primero proporciona un analizador DOM que transforma el documento XML en un árbol de objetos compatibles con éste estándar. El paquete `org.w3c.dom`, proporciona las clases Java que implementan el estándar DOM y a través de las cuales podemos desplazarnos por el árbol de objetos del documento y acceder al contenido de elementos y atributos.



## Ejemplo práctico

Vamos realizar una aplicación de ejemplo en la que accederemos a un servicio Web XML de información meteorológica. El servicio sirve un documento XML con los datos de precipitación, temperatura, etc. de una ciudad. El documento generado por el servicio se encuentra en la siguiente dirección:

[http://www.aemet.es/xml/municipios/localidad\\_28079.xml](http://www.aemet.es/xml/municipios/localidad_28079.xml)

Y su estructura es la siguiente:

```
<root id="28079" version="1.0" xsi:noNamespaceSchemaLocation="http://www.aemet.es/xsd/localidades.xsd">
  <origen>
    <productor>Agencia Estatal de Meteorología - AEMET. Gobierno de España</productor>
    <web>http://www.aemet.es</web>
    <enlace>http://www.aemet.es/es/eltiempo/prediccion/municipios/madrid-id28079</enlace>
    <language>es</language>
    <copyright>© AEMET. Autorizado el uso de la información y su reproducción citando a AEMET como autora de la misma.</copyright>
    <nota_legal>http://www.aemet.es/es/nota_legal</nota_legal>
  </origen>
  <elaborado>2016-02-13T08:15:01</elaborado>
  <nombre>Madrid</nombre>
  <provincia>Madrid</provincia>
  <prediccion><dia fecha="2016-02-13">
    <prob_precipitacion periodo="00-24">100</prob_precipitacion>
  </dia>
  <prob_precipitacion periodo="00-12">75</prob_precipitacion>
  <prob_precipitacion periodo="12-24">100</prob_precipitacion>
  <prob_precipitacion periodo="00-06">30</prob_precipitacion>
  <prob_precipitacion periodo="06-12">35</prob_precipitacion>
  <prob_precipitacion periodo="12-18">80</prob_precipitacion>
  <prob_precipitacion periodo="18-24">95</prob_precipitacion>
  <cota_nieve_prov periodo="00-24">2100</cota_nieve_prov>
  <cota_nieve_prov periodo="00-12">2100</cota_nieve_prov>
  <cota_nieve_prov periodo="12-24">2000</cota_nieve_prov>
  </prediccion>
</root>
```

La aplicación consistirá en una actividad con un botón y un campo de texto TextView. Al pulsar el botón, el programa se conectará con el servicio Web, leerá el documento XML y mostrará en el campo de texto el nombre de la ciudad y la media de precipitaciones registradas en el documento:



Al igual que en el ejemplo anterior, las operaciones de comunicación con el servicio y manipulación de resultados serán programadas en una subclase de AsyncTask. He aquí el código completo de la actividad:


```
public class MainActivity extends AppCompatActivity {

    TextView tvDatos;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        //referencia al campo de texto TextView
        tvDatos=(TextView)this.findViewById(R.id.tvDatos);
    }

    public void datos(View v){
        ComunicacionTask com=new ComunicacionTask();
        com.execute("http://www.aemet.es/xml/municipios/localidad_28079.xml");
    }

    //definición de la tarea encargada de conectarse con el servicio
    //y procesar los datos
    private class ComunicacionTask extends
        AsyncTask<String, Void, String> {



        @Override
        protected String doInBackground(String... params) {
            String cadena="";
            try{
                URL url=new URL(params[0]);
            }
        }
    }
}
```



```

URLConnection con=url.openConnection();
InputStream is=con.getInputStream();
Document doc;
//creación del objeto DocumentBuilder encargado de
// analizar el documento XML
DocumentBuilderFactory factory=
    DocumentBuilderFactory.newInstance();
DocumentBuilder builder;
builder=factory.newDocumentBuilder();
doc=builder.parse(is);
//utilizando el DOM, recorremos colecciones de objetos
// a partir de su nombre
NodeList listaNombre=
    doc.getElementsByTagName("nombre");
//recuperamos nombre de la localidad
cadena+="Localidad: "+
    listaNombre.item(0).getTextContent()+"\n";
//recuperamos la colección de objetos probabilidad
//de precipitación
NodeList listaPrecip=
    doc.getElementsByTagName("prob_precipitacion");
double media=0;
//calculamos la media de las precipitaciones
for(int i=0;i<listaPrecip.getLength();i++){
    String valor=listaPrecip.item(i).
        getTextContent();
    if(valor==null || valor.equals("")){
        valor="0";
    }
}

```

```

media+=Double.parseDouble(valor);
    }
media=media/listaPrecip.getLength();
cadena+=" Media prob. precipitación: "+media;
    }
    catch(Exception ex){
        ex.printStackTrace();
    }

    return cadena;
}

@Override
protected void onPostExecute(String result) {
    //presentamos los resultados en el TextView
    tvDatos.setText(result);
}

}

```

## 2.2 | Servicios Web JSON

JSON es un estándar para representación de objetos y arrays, que inicialmente se empezó a utilizar en programación Web cliente con JavaScript (JSON son las siglas de JavaScript Object Notation), pero que hoy en día se emplea en múltiples contextos y con diferentes lenguajes.

Uno de los motivos de la amplia extensión de este estándar para representar objetos es su sencillez. Veámoslo con un ejemplo, si quisiéramos representar un objeto de tipo JavaBean, que encapsula los datos de una Persona, como su nombre, email y edad, en formato JSON sería:

```
{nombre:"Javier Sánchez", email:"jjjss@gmail.com", edad:24}
```

Ahí podemos comprobar lo sencillo que resulta la representación de este tipo de objetos con JSON; entre llaves se van indicando las propiedades del objeto en forma propiedad:valor, separandolas por una ",".

Además de estos método get() que nos También podemos representar de forma sencilla un array, basta con indicar los valores del mismo separados por una ",", delimitando todo el conjunto por corchetes "[" y "]":

```
[4, 8, 1, 3]
```

Combinando ambas notaciones, podríamos representar fácilmente un conjunto de objetos JSON:

```
[{nombre:"Javier Sánchez", email:"jjjss@gmail.com", edad:24},  
{nombre:"Ana López", email:"asl@gmail.com", edad:31},  
{nombre:"María Martín", email:"mml@gmail.com", edad:26}  
]
```

El hecho de que en muchos casos los servicios Web tengan que devolver conjuntos de datos con las mismas propiedades a las aplicaciones clientes, hacen que JSON sea el formato más apropiado para el envío de estos datos.

## El paquete org.json

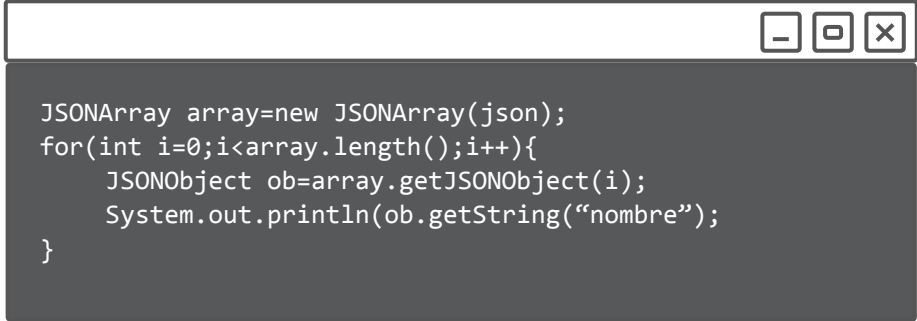
Para manipular cadenas JSON en una aplicación Android, disponemos del paquete org.json, que proporciona fundamentalmente dos clases:

- **JSONArray.** Representa un array JSON. Se puede crear un objeto de esta clase a partir de una cadena de caracteres que represente el array. Una vez disponemos del objeto JSONArray, se puede iterar con una instrucción for, accediendo a cada una de las posiciones del array con su método get(). Si el contenido de cada posición del array es un objeto JSON, utilizaríamos el método getJSONObject() , que nos devuelve el objeto como JSONObject.
- **JSONObject.** Representa un objeto JSON. A partir de los métodos getString(), getInt(), etc., puede accederse a los valores de sus propiedades. Estos métodos reciben como parámetro el nombre de la propiedad cuyo valor se quiere obtener.

Para entender el funcionamiento de estos objetos, supongamos que tenemos la siguiente cadena de caracteres conteniendo un array de objetos JSON:

```
String json="{\"nombre\":\"Javier Sánchez\", \"email\":\"jjjss@gmail.com\", \"edad\":24},{\"nombre\":\"Ana López\", \"email\":\"asl@gmail.com\", \"edad\":31},{\"nombre\":\"María Martín\", \"email\":\"mml@gmail.com\", \"edad\":26}}"
```

Si quisiéramos mostrar el nombre de cada persona en la consola LogCat sería:



```
JSONArray array=new JSONArray(json);
for(int i=0;i<array.length();i++){
    JSONObject ob=array.getJSONObject(i);
    System.out.println(ob.getString("nombre"));
}
```



## Ejercicio resuelto

Vamos a ver un ejemplo de conexión a un servicio Web JSON real desde una aplicación Android. El servicio en cuestión forma parte de un conjunto de servicios incluidos en el **catálogo de datos de utilidad pública** ofrecidos por <http://datos.gob.es/catalogo>.

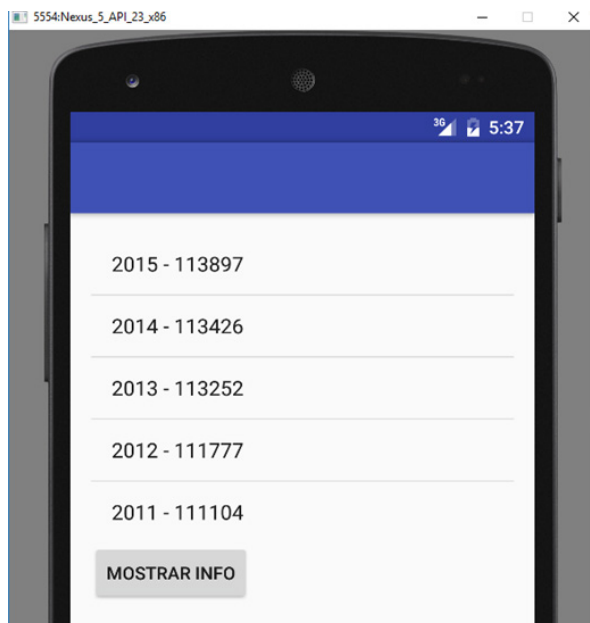
Entre sus muchos servicios, encontramos uno que nos ofrece los datos de empadronamiento del ayuntamiento de Alcobendas (Madrid- España), desde el año 2011 hasta 2015. El servicio se encuentra en la siguiente dirección:

<http://datos.alcobendas.org/dataset/c3002859-2d57-42b9-8aaf-a69e905e93fb/resource/8cac8866-91c2-43c3-baa0-0908ba2d0328/download/evolucionanualdelapoblacionpordistritos20112015.json>

Y genera una respuesta JSON con este aspecto:

```
[
  {
    "Año":2015,
    "Centro":44789,
    "Norte":45488,
    "Urbanizaciones":23620
  },
  {
    "Año":2014,
    "Centro":44294,
    "Norte":45631,
    "Urbanizaciones":23501
  },
  :
]
```

En este ejercicio realizaremos una aplicación Android que se conecte con el servicio y muestre en una tabla el total de vecinos empadronados cada año. El aspecto de la actividad será el que se muestra en la siguiente imagen:



Según se desprende de la imagen anterior, la actividad consta de un ListView y un botón de pulsación, que al ser pulsado, se conectará con el servicio, realizará la lectura de los datos y creará un array con los resultados a mostrar en la lista.

Todas las tareas se programarán, como en el caso anterior, en una subclase de AsyncTask. En el método `doInBackground()` realizaremos la conexión con el servicio y recuperaremos la cadena JSON de respuesta. Esta cadena se pasará como resultado al método `onPostExecute()`, que se encargará de parsearla a un JSONArray para manipular su contenido y formar el array que posteriormente se pasará al ListView para que visualice los datos.

Este sería el código completo de la actividad:

```
public class MainActivity extends AppCompatActivity {

    ListView lvCiudades;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        lvCiudades=(ListView)this.findViewById(R.id.lvCiudades);
    }

    public void mostrar(View v){
        ComunicacionTask com=new ComunicacionTask();
        com.execute("http://datos.alcobendas.org/dataset/c3002859-2d57-42b9-8aaf-a69e905e93fb/resource/8cac8866-91c2-43c3-baa0-0908ba2d0328/download/evolucionanualdelapoblacionpordistritos20112015.json");
    }
}
```

```

private class ComunicacionTask extends AsyncTask<String, Void,
String> {

    @Override
    protected String doInBackground(String... params) {
        String cadenaJson="";
        try{
            URL url=new URL(params[0]);
            URLConnection con=url.openConnection();
            //recuperacion de la respuesta JSON
            String s;
            InputStream is=con.getInputStream();
            //utilizamos UTF-8 para que interprete
            //correctamente las ñ y acentos
            BufferedReader bf=new BufferedReader(
            new InputStreamReader(is, Charset.forName("UTF-8")));
            while((s=bf.readLine())!=null){
                cadenaJson+=s;
            }
        }
        catch(IOException ex){
            ex.printStackTrace();
        }

        return cadenaJson;
    }

    @Override
    protected void onPostExecute(String result) {
        String[] datosCiudad=null;
        try{
            //creamos un array JSON a partir de la cadena
recibida

```

```

JSONArray jarray=new JSONArray(result);
//creamos el array de String con el tamaño
//del array JSON
datosCiudad=new String[jarray.length()];
for(int i=0;i<jarray.length();i++){
    JSONObject job=jarray.getJSONObject(i);
    //por cada objeto JSON, creamos una cadena
    //con la propiedad año y la suma de
    //habitantes de cada zona,el resultado lo
    //añadimos al array
    int habitantes=job.getInt("Centro")+
    job.getInt("Norte")+
    job.getInt("Urbanizaciones");
    datosCiudad[i]=job.getString("Año")+
    " - "+habitantes;
}
cargarLista(datosCiudad);
}
catch(JSONException ex){
    ex.printStackTrace();
}
}

private void cargarLista(String[] datos){
    //creamos un arrayadapter con los datos del array
    // y lo asignamos al ListView
    ArrayAdapter<String> adp=new ArrayAdapter<String>(
    MainActivity.this, android.R.layout.simple_list_item_1,datos);
    lvCiudades.setAdapter(adp);
}
}
}

```

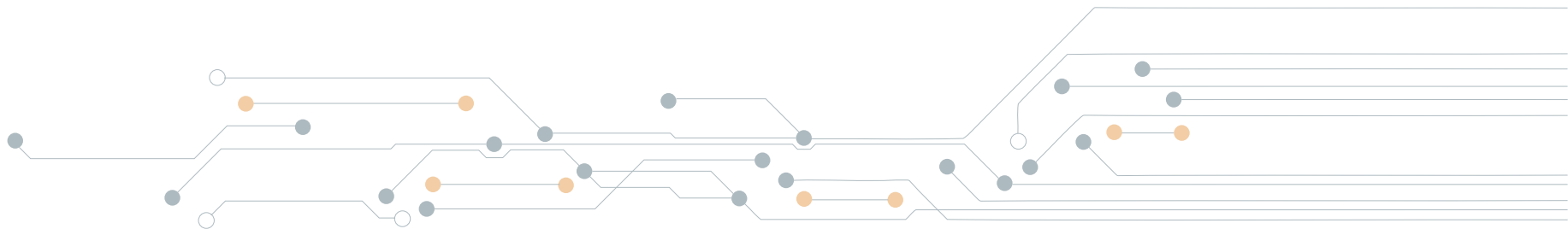
Como en los ejemplos anteriores, sería necesario otorgar el permiso de acceso a Internet.

### 3. Comunicaciones mediante sockets

Utilizando `URLConnection` podemos acceder a cualquier recurso Web identificado por una URL, pero si lo que queremos es comunicarnos con una aplicación independiente que se está ejecutando en una máquina remota y con la que queremos intercambiar datos, deberíamos emplear sockets.

Un socket es un objeto que permite conectarnos con una aplicación que se ejecuta en un equipo remoto y que se encuentra escuchando en un determinado puerto. Una vez creado el socket, podemos obtener a partir de él los streams de entrada y salida para realizar el intercambio de datos entre nuestro programa Android y la aplicación remota.

Este sistema de comunicación mediante sockets es el que emplean aplicaciones tan populares como WhatsApp, que utilizan sockets para conectarse a una aplicación servidor remota, que es la que se encarga de gestionar la recepción de los mensajes y su envío a los terminales destinatarios.



## 3.1 | La clase ServerSocket

Antes de ver cómo crear un socket desde una aplicación Android para conectarnos con un programa remoto, vamos a ver precisamente como crear un programa de Java estándar que haga de servidor, es decir, con el que se conecten nuestras aplicaciones Android

Para ello, el paquete java.net de Java SE proporciona la clase ServerSocket, cuyo constructor tiene el siguiente formato:

```
ServerSocket(int port)
```

El único parámetro que se necesita para crear un socket de servidor es el puerto de escucha, su valor puede ser cualquier número desde 1024 a aproximadamente 64000, siempre que dicho número no esté siendo usado como puerto por otra aplicación que se esté ejecutando en la misma máquina.

Una vez creado el objeto ServerSocket, está listo para aceptar llamadas desde el exterior, para lo cual tendríamos que llamar al método accept(). El método **accept()** **bloquea el programa a la espera de recibir una conexión externa**. Cuando la conexión se produce, el método accept() se desbloquea y devuelve un objeto Socket para comunicarse con el cliente que ha realizado la conexión.

El siguiente bloque de instrucciones de ejemplo, que estarían definidas en una aplicación Java del servidor, muestran el proceso:

```
ServerSocket server=new ServerSocket(5000);
```

```
Socket sc=server.accept();
```

```
//instrucciones que se ejecutarán cuando se reciba llamada
```

## 3.2 | La clase Socket

Se encuentra también en el paquete `java.net`. Un objeto `Socket` representa una conexión con un punto remoto. Como hemos comentado anteriormente, el método `accept()` del socket del servidor devuelve un objeto de esta clase para que el servidor pueda comunicarse con la aplicación cliente. Del lado cliente, en nuestro caso desde una aplicación Android (la librería de Android dispone de la clase `java.net.Socket`), para conectarnos a una aplicación remota crearemos un objeto `Socket` utilizando el siguiente constructor:

```
Socket(String host, int port)
```

Donde `host` es el nombre del equipo remoto o su dirección ip y `port` es el número de puerto por el que se encuentra escuchando el socket de servidor.

Donde `host` es el nombre del equipo remoto o su dirección ip y `port` es el número de puerto por el que se encuentra escuchando el socket de servidor.

La siguiente instrucción crearía un `Socket` para comunicarse con una aplicación que estuviera ejecutándose en el equipo con dirección `www.ejemplo.com` y escuchando llamadas a través del puerto 5000:

```
Socket sck=new Socket("www.ejemplo.com", 5000);
```

En la siguiente imagen se muestra la secuencia de pasos para conectar dos aplicaciones mediante sockets:

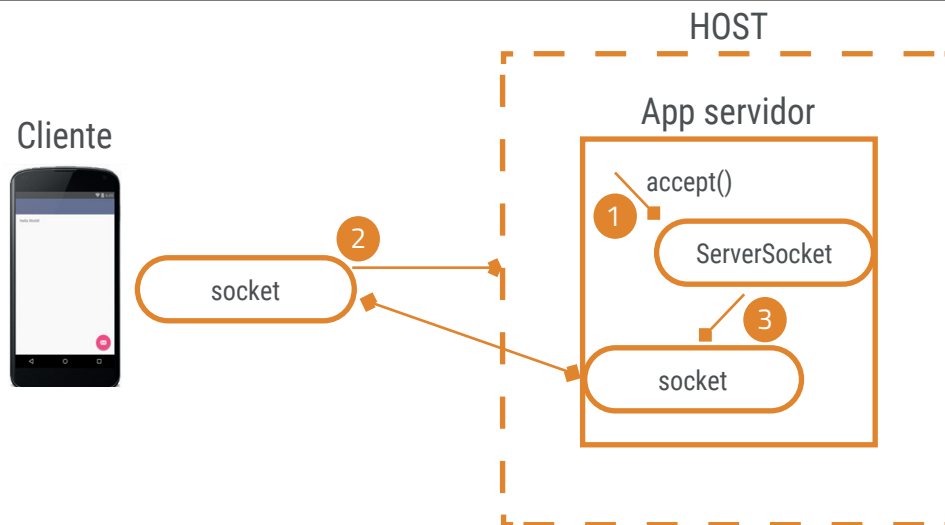
En primer lugar, la aplicación que va a hacer de servidor, creará el `ServerSocket` y llamará al método `accept()` sobre este para quedar a la escucha de llamadas externas. A partir de ahí, las aplicaciones clientes pueden conectar con la aplicación de servidor, esto se llevará a cabo creando un objeto `Socket` en el que se indicará el host y puerto al que queremos conectarnos.

La creación del socket en el cliente provoca una llamada desde esta aplicación a la del servidor, que provocará el desbloqueo del método `accept()` y que se genere un socket en el servidor para comunicarse con el cliente.

A partir de ahí, las aplicaciones ya pueden intercambiar datos. Para ello, la clase `Socket` proporciona los siguientes métodos:

- **`getInputStream()`**. Devuelve un `InputStream` para leer datos a través del socket, enviados por el socket remoto
- **`getOutputStream()`**. Devuelve un objeto `PrintStream` para enviar datos al socket remoto.

Como vemos, una vez creado el socket, las operaciones de envío y recepción de datos se convierten en operaciones de entrada/salida estándar.



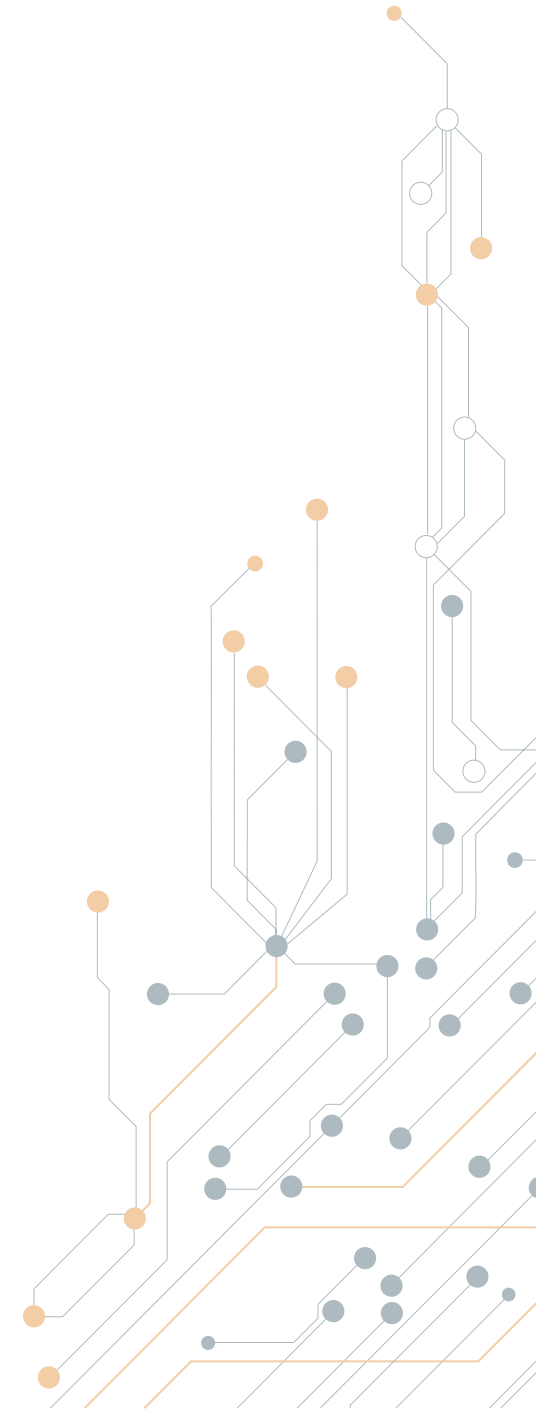
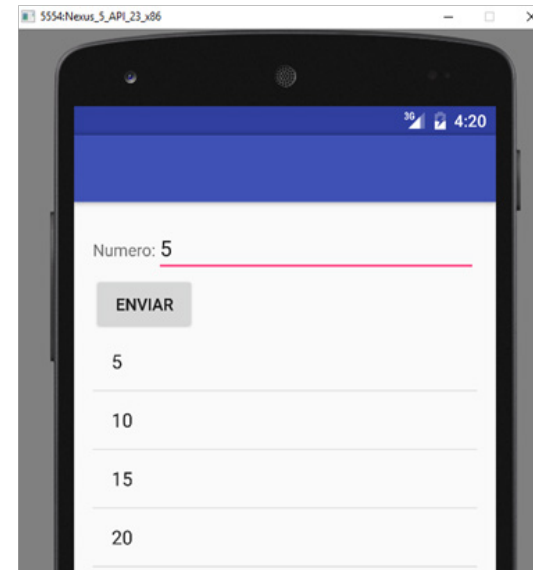
1. Llamada a `accept()` en la aplicación servidor.
2. Cliente crea un socket para conectarse con servidor.
3. El método `accept()` devuelve en la aplicación del servidor un socket para comunicar con el cliente.

## Ejercicio de ejemplo

Como siempre, realizaremos una aplicación de ejemplo en la que pondremos en práctica el uso de sockets.

En primer lugar, realizaremos una aplicación de servidor que, a partir de un número recibido desde las aplicaciones clientes, calculará la tabla de multiplicar de dicho número y se la enviará al cliente como un array JSON.

En cuanto al cliente, realizaremos una aplicación Android consistente en una actividad con un campo de texto EditText en el que se solicitará al usuario la introducción del número. El número será enviado al servidor y la tabla de multiplicar enviada como respuesta por éste será mostrada en un ListView:



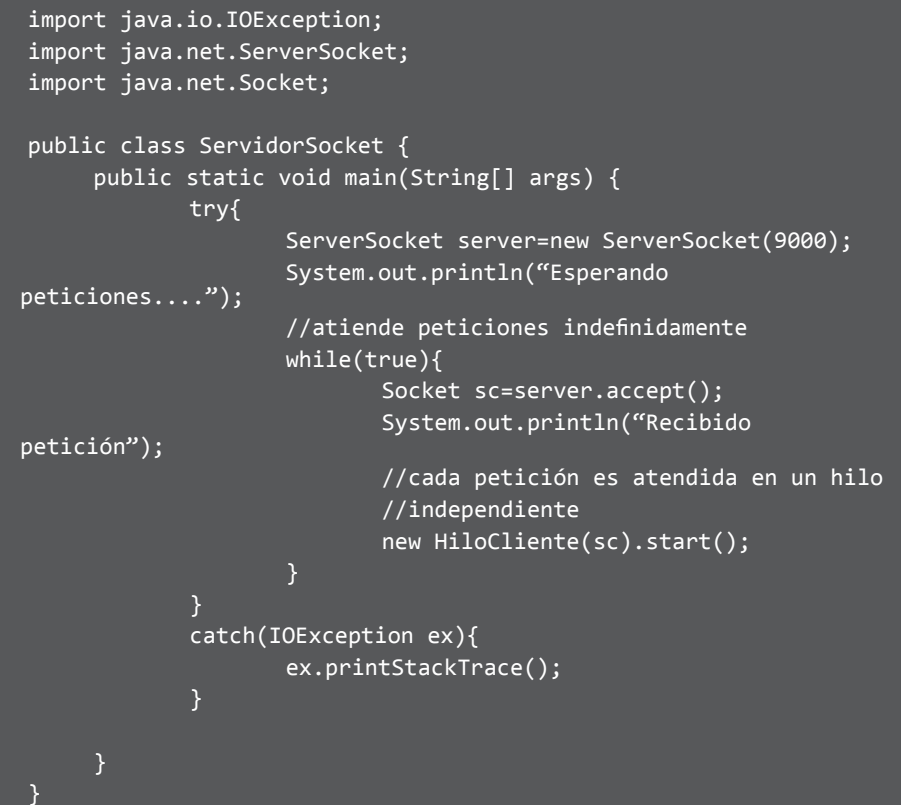


## Aplicación servidor

Utilizando un entorno de desarrollo Java, como eclipse o netbeans, crearemos una aplicación de Java estándar en la que implementaremos la funcionalidad del servidor de sockets.

La idea es que el servidor de sockets esté accesible permanentemente para cualquier aplicación cliente que quiera conectarse al mismo, por ello, cada vez que se produzca una conexión con el servidor, se creará un hilo para atender dicha petición y que se ejecute concurrentemente con los hilos de las peticiones del resto de clientes.

La creación del servidor de sockets se realizará en el método main de la aplicación. Este sería el código de la clase principal:



```
import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;

public class ServidorSocket {
    public static void main(String[] args) {
        try{
            ServerSocket server=new ServerSocket(9000);
            System.out.println("Esperando
peticiones....");
            //atiende peticiones indefinidamente
            while(true){
                Socket sc=server.accept();
                System.out.println("Recibido
petición");
                //cada petición es atendida en un hilo
                //independiente
                new HiloCliente(sc).start();
            }
        }
        catch(IOException ex){
            ex.printStackTrace();
        }
    }
}
```

Podemos observar la existencia del bucle infinito `while(true)` que permite atender peticiones de forma indefinida. Cada vez que llega una llamada, el método `accept()` devuelve un objeto `Socket` que es utilizado como argumento del nuevo hilo que atenderá la petición, después se vuelve a llamar de nuevo a `accept()` a la espera de una nueva llamada, y así indefinidamente.

El hilo que gestiona la comunicación con el cliente se implementa en una clase que hereda `Thread`. Este es el código de esta clase:

```
public class HiloCliente extends Thread {
    Socket sc;
    public HiloCliente(Socket sc){
        this.sc=sc;
    }

    @Override
    public void run() {
        InputStream entrada=null;
        PrintStream salida=null;

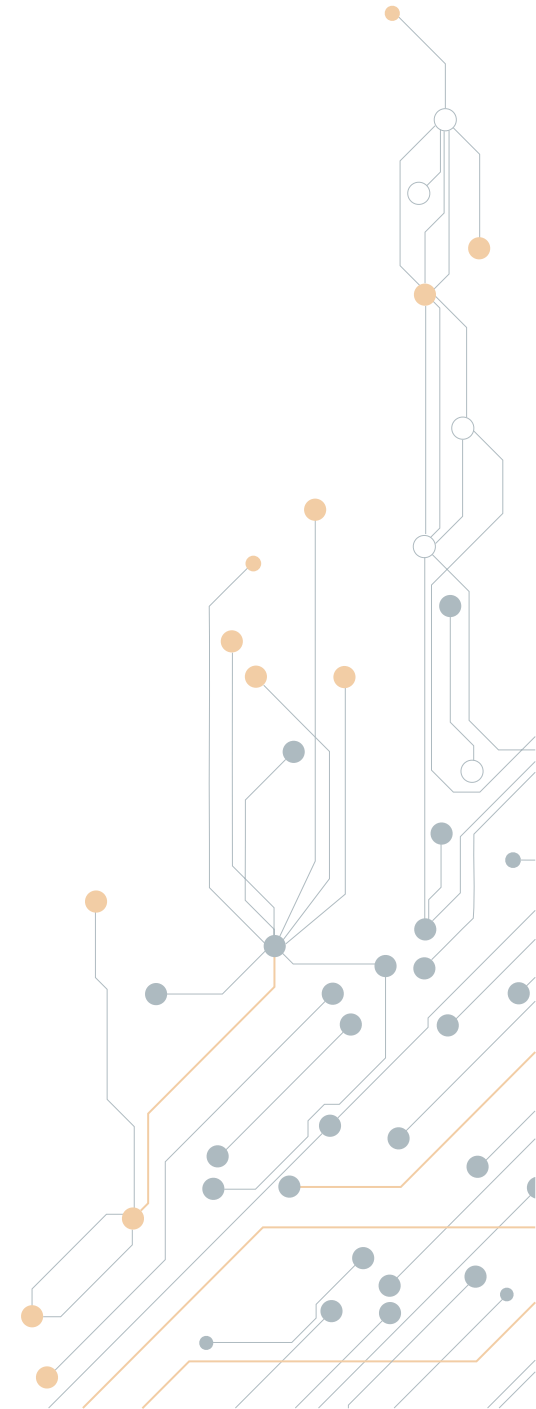
        try {
            entrada = sc.getInputStream();
            salida=new PrintStream(sc.getOutputStream());
            BufferedReader bf=new BufferedReader(
                new InputStreamReader(entrada));
            //leemos el número enviado por la aplicación cliente
            int num=Integer.parseInt(bf.readLine());

            StringBuilder sb=new StringBuilder();
            sb.append("[");
            for(int i=1;i<=10;i++){
                sb.append(num*i+",");
            }
            //elimina la última coma
            sb.deleteCharAt(sb.length()-1);
            sb.append("]");
            //envia la cadena de respuesta
            salida.println(sb.toString());
            salida.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```





```
} catch (IOException ex) {  
    ex.printStackTrace();  
} finally {  
    try {  
        sc.close();  
  
        } catch (IOException ex) {  
            ex.printStackTrace();  
        }  
    }  
}
```



Para poner en funcionamiento la aplicación del servidor simplemente tenemos que ejecutar la clase `ServidorSocket`. A partir de ese momento, el programa quedará bloqueado en la instrucción de llamada al método *accept()* a la espera de recibir peticiones de conexión por parte de aplicaciones clientes a través del puerto 9000.

### Aplicación cliente

La aplicación cliente será una aplicación Android con la funcionalidad descrita anteriormente.

Desde el método de respuesta al evento clic del botón, pondremos en marcha una tarea `AsyncTask` que creará un `Socket` para conectarse con la aplicación del servidor, enviarle el número introducido en el `EditText` y recuperar la cadena de respuesta.

Aquí tenemos el código de la actividad al completo:

```
public class MainActivity extends AppCompatActivity {
    //listView
    ListView lvTabla;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        lvTabla=(ListView)this.findViewById(R.id.lvMulti);
    }

    public void enviar(View v){
        //recogemos el número
        EditText edtNumero=(EditText)this.findViewById(R.
id.edtNumero);
        String numero=edtNumero.getText().toString();
        ComunicacionTask com=new ComunicacionTask();
        //ejecutamos la tarea que se comunica con el servidor
        com.execute("10.0.2.2","9000",numero);
    }

    private class ComunicacionTask extends AsyncTask<String, Void,
String> {
        @Override
        protected String doInBackground(String... params) {
            BufferedReader bf=null;
            Socket sc=null;
            String res="";

```

```

        try{
            sc=new Socket(params[0],Integer.
parseInt(params[1
));
        PrintStream ps=new PrintStream(sc.
getOutputStream());
        ps.println(params[2]);
        ps.flush();
        //recuperamos la respuesta y la devolvemos para
        //que la procese onPostExecute()
        InputStream is=sc.getInputStream();
        bf=new BufferedReader(new InputStreamReader(is));
        res= bf.readLine();
    }
    catch(IOException ex){
        ex.printStackTrace();
    }
    finally {//cierre de objetos en el finally
        if(bf!=null){
            try {
                bf.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
        if(sc!=null){
            try {
                sc.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}

```

```

@Override
protected void onPostExecute(String result) {
    Integer[] tabla=null;
    try{
        //creamos un array JSON a partir de la cadena
        JSONArray jarray=new JSONArray(result);
        //creamos el array de String con el tamaño
        //del array JSON
        tabla=new Integer[jarray.length()];
        for(int i=0;i<jarray.length();i++){
            //añadimos al array los resultados de la
            //tabla de multiplicar que vienen en el array
            tabla[i]=jarray.getInt(i);
        }
        cargarLista(tabla);
    }
    catch(JSONException ex){
        ex.printStackTrace();
    }
}

private void cargarLista(Integer[] datos){
    //creamos un arrayadapter con los datos del array
    // y lo asignamos al ListView
    ArrayAdapter<Integer> adp=
        new ArrayAdapter<Integer>(MainActivity.this,
            android.R.layout.simple_list_item_1,datos);
    lvTabla.setAdapter(adp);
}
}

```

El primer parámetro que se pasa al método `execute()` del `AsyncTask` corresponde a la dirección IP del equipo donde se encuentra la aplicación de servidor. En este caso, la dirección 10.0.2.2 corresponde con la del equipo local, es decir, el mismo donde se encuentra el propio emulador. que se indica a continuación:

Y es que para probar esta aplicación desde un terminal real, deberíamos indicar la dirección ip real del equipo donde se está ejecutando la aplicación del servidor.

Como en los casos anteriores de comunicación con el exterior, tenemos que declarar el permiso de acceso a Internet en el archivo de manifiesto.

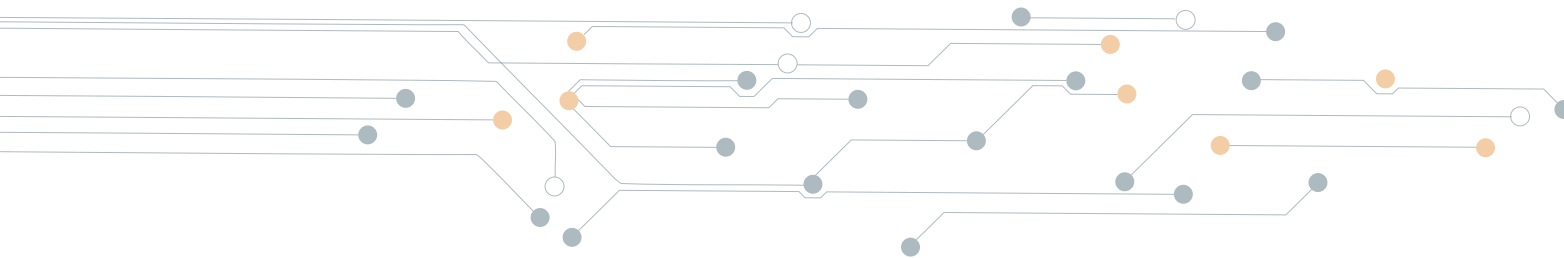
---

Desde el método de respuesta al evento clic del botón, pondremos en marcha una tarea `AsyncTask` que creará un `Socket` para conectarse con la aplicación del servidor, enviarle el número introducido en el `EditText` y recuperar la cadena de respuesta.

---

## 4. Geolocalización

La obtención de los datos de posicionamiento del terminal desde una aplicación Android no está directamente relacionado con las tareas de comunicaciones, sin embargo, el hecho de poder enviar nuestra posición a una aplicación remota es algo que ofrece múltiples posibilidades a los desarrolladores, por eso lo vamos a tratar dentro de este apartado.



## 4.1 | LocationManager

La clase `android.location.LocationManager` nos proporciona acceso al servicio de localización del dispositivo.

Esta clase no puede instanciarse directamente, para obtener un objeto `LocationManager` recurriremos al método `getSystemService()` de `Context` que nos proporciona acceso a diferentes servicios del sistema. A este método habría que pasarle el servicio que queremos obtener, dichos servicios se encuentran definidos en constantes de `Context`:

```
public abstract Object getSystemService (String name)
Return the handle to a system-level service by name. The class of the returned object varies

WINDOW_SERVICE ("window")
The top-level window manager in which you can place custom windows. The returned

LAYOUT_INFLATER_SERVICE ("layout_inflater")
A LayoutInflater for inflating layout resources in this context.

ACTIVITY_SERVICE ("activity")
A ActivityManager for interacting with the global activity state of the system.

POWER_SERVICE ("power")
A PowerManager for controlling power management.

ALARM_SERVICE ("alarm")
A AlarmManager for receiving intents at the time of your choosing.

NOTIFICATION_SERVICE ("notification")
A NotificationManager for informing the user of background events.

KEYGUARD_SERVICE ("keyguard")
A KeyguardManager for controlling keyguard.

LOCATION_SERVICE ("location")
A LocationManager for controlling location (e.g., GPS) updates.
```

Así pues, para obtener el `LocationManager` sería:

```
LocationManager lm=
```

```
(LocationManager)this.getSystemService(Context.LOCATION_SERVICE);
```

El acceso al servicio de localización requiere un permiso especial por parte de la aplicación. Debemos registrar el siguiente permiso en el archivo de manifiesto:

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



## 4.2 | Obtención de la posición

A partir del objeto `LocationManager`, podemos obtener la última posición conocida del terminal utilizando el método `getLastKnownLocation()`, al que le proporcionaremos una cadena con el proveedor de localización que queremos utilizar (satélite, red, ...). Estos proveedores se encuentran definidos en constantes dentro de `LocationManager`:

- **NETWORK\_PROVIDER.** Nombre del proveedor de localización por red
- **GPS\_PROVIDER.** Nombre del proveedor de localización por GPS

La posición es devuelta por el método `getLastKnownLocation()` en forma de objeto `Location`.

La clase `android.location.Location` encapsula los datos de localización del terminal. Estos datos podemos obtenerlos a través de los métodos:

- **getLatitude().** Devuelve la latitud
- **getLongitude().** Devuelve la longitud
- **getAltitude().** Devuelve la altitud.

La siguiente instrucción permite obtener la última posición conocida a través del proveedor GPS y mostrarla en un Toast:

```
Location loc=
    lm.getLastKnownLocation(LocationManager.PASSIVE_
        PROVIDER);

String datos="Altitud:"+loc.getAltitude()+
    " Longitud:"+loc.getLongitude()+" Latitud: "+loc.getLatitude()

Toast.makeText(this, datos, Toast.LENGTH_LONG).show();
```

## 4.3 | Cambios de localización

Otra de las posibilidades que nos ofrece el objeto `LocationManager` es responder a los cambios de localización que se producen en el dispositivo, para lo cual, debemos implementar la interfaz `LocationListener`. Esta interfaz dispone de los siguientes métodos:

- **`onLocationChanged(Location location)`.**

Es llamado cada vez que se produce un cambio de posición en el dispositivo. Recibe como parámetro el objeto `Location` con la nueva localización

- **`onProviderDisabled(String provider)`.**

Es llamado cuando un proveedor de localización se desactiva en el terminal. Recibe como parámetro el nombre del proveedor desactivado.

- **`onProviderEnabled(String provider)`.**

Es llamado cuando un proveedor de localización se activa. Recibe como parámetro el nombre del proveedor activado.

- **`onStatusChanged(String provider, int status, Bundle extras)`.**

Es llamado cuando un proveedor cambia de estado.

Para que la aplicación responda a los sucesos anteriores, habrá que registrar el escuchador a través del método `requestLocationUpdates()` de `LocationManager`. El formato de este método es el siguiente:

**`requestLocationUpdates(String provider, long minTime, float minDistance, LocationListener listener)`**

El significado de los parámetros es el siguiente:

- **`provider`.** Nombre del proveedor utilizado para la determinación de la localización.

- **`minTime`.** Tiempo mínimo, en milisegundos, entre actualizaciones de localización.

- **`minDistance`.** Distancia mínima, en metros, entre actualizaciones de localización.

- **`listener`.** Escuchador que implementa `LocationListener`.

Por otro lado, si queremos que la aplicación deje de responder a los sucesos definidos en el escuchador, utilizaremos el método `removeUpdates()` de `LocationManager` pasándole el objeto escuchador.

## Ejercicio de ejemplo

A continuación realizaremos una aplicación Android en la que utilizaremos el servicio de localización del terminal para enviar nuestra posición a un programa remoto.

La idea es que cada vez que nuestro terminal cambie de posición, le envíe las nuevas coordenadas a una aplicación que se encargará de registrar en un fichero los datos de localización del dispositivo, junto con la hora de envío y el nombre del equipo.

Lo primero que haremos será crear la aplicación de servidor con nuestro entorno Java habitual.

Al igual que en el ejercicio de sockets anterior, la clase principal del servidor creará un hilo por cada petición cliente para gestionar dicha llamada. Este sería el código de esta clase principal:

```
public class ServidorSocket {  
  
    public static void main(String[] args) {  
        try{  
            ServerSocket server=new ServerSocket(9000);  
            System.out.println("Esperando  
peticiones...");  
            while(true){  
                Socket sc=server.accept();  
                System.out.println("Recibido  
petición");  
                new HiloCliente(sc).start();  
            }  
        }  
        catch(IOException ex){  
            ex.printStackTrace();  
        }  
    }  
}
```

En cuanto a la clase HiloCliente, aquí tenemos su implementación:

```

public class HiloCliente extends Thread {
    Socket sc;
    public HiloCliente(Socket sc){
        this.sc=sc;
    }

    @Override
    public void run() {
        InputStream entrada=null;
        FileWriter fw=null;
        try {
            entrada = sc.getInputStream();
            BufferedReader bf=new BufferedReader(
                new InputStreamReader(entrada));
            //leemos la información enviada por la aplicación

cliente
            String datos=bf.readLine();
            //agregamos la hora
            SimpleDateFormat df=new SimpleDateFormat("hh:mm");
            datos+="|"+df.format(new Date());
            datos+="|"+sc.getInetAddress().getHostName();
            //guardamos en un fichero los datos de localización del
cliente
        }
    }
}

```

```

//agregándolo a la información existente
        fw=new FileWriter("locations.txt", true);
        fw.write(datos+System.lineSeparator());
        fw.flush();
    } catch (IOException ex) {
        ex.printStackTrace();
    } finally {
        try {
            fw.close();
            sc.close();

        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }
}
}
}

```

A los datos recibidos desde la aplicación cliente, le añadimos la hora actual, formateada a través de la clase SimpleDateFormat, y la dirección ip o nombre del terminal cliente, que se obtiene a través del método getHostName() del objeto InetAddress asociado al Socket cliente. Finalmente, la cadena resultante es almacenada en el fichero locations.txt.

En cuanto a la aplicación Android, en este caso no vamos a incorporar ningún componente gráfico a la actividad, puesto que la tarea a realizar, que es enviar los datos de localización al servidor, no requerirá ninguna acción del usuario.

Las instrucciones de comunicación con el servidor se programarán en una tarea asíncrona AsyncTask como en otras ocasiones. Esta tarea será activada desde el evento de cambio de localización, puesto que la idea es que cada vez que se produzca este suceso se envíe la información al servidor para su almacenamiento.

El registro del escuchador en el LocationManager lo realizaremos en el método onResume() de la actividad, mientras que en onPause() procederemos a desregistrar el escuchador. De esta manera, la aplicación solo estará respondiendo a los eventos de cambio de localización mientras la actividad esté activa.



```
public class MainActivity extends AppCompatActivity {
    LocationManager lm;
    LocationListener escuchadorPosicion=new LocationListener(){

        @Override
        public void onLocationChanged(Location loc) {
            String datos="Altitud:"+loc.getAltitude()+
                "|Longitud:"+loc.getLongitude()+
                "|Latitud: "+loc.getLatitude();
            //muestra los datos a enviar
            Toast.makeText(MainActivity.this, datos, Toast.LENGTH_
LONG).show();
            ComunicacionTask com=new ComunicacionTask();
            //Ejecuta la tarea con la dirección del servidor,
            //puerto y datos a enviar
            com.execute("10.0.2.2","9000",datos);
        }
        @Override
        public void onStatusChanged(String provider, int status,
Bundle extras) {
        }
        @Override
        public void onProviderEnabled(String provider) {
        }
    }
}
```

```

@Override
public void onProviderDisabled(String provider) {
}

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    //obtener objeto LocationManager que representa
    //el servicio de localización
    lm=(LocationManager)this.
        getSystemService(Context.LOCATION_SERVICE);
}

@Override
protected void onPause() {
    super.onPause();
    //elimina la asociación con el escuchador cuando
    //la actividad no está activa
    lm.removeUpdates(escuchadorPosicion);
}

@Override
protected void onResume() {
    super.onResume();
    //asociar el LocationManager al escuchador
    lm.requestLocationUpdates(LocationManager.GPS_PROVIDER,
    10000, 10, escuchadorPosicion);
}

//tarea de comunicación con el servidor.
//como no se van a procesar resultados, el tipo
//de devolución es Void y no se implementa onPostExecute

```

```

private class ComunicacionTask extends AsyncTask<String, Void,
Void> {

    @Override
    protected Void doInBackground(String... params) {
        BufferedReader bf=null;
        Socket sc=null;
        String res="";
        try{
            sc=new Socket(params[0],Integer.
            parseInt(params[1]));
            PrintStream ps=new PrintStream(sc.
            getOutputStream());
            //enviamos los datos
            ps.println(params[2]);
            ps.flush();

        }
        catch(IOException ex){
            ex.printStackTrace();
        }
        finally { //cierre de objetos en el finally
            if(sc!=null){
                try {
                    sc.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }
        //debe devolver null por ser Void
        //y no void
        return null;
    }
}

```

En cuanto al código de la actividad, queremos que los contactos aparezcan desde el principio al cargarse la actividad, por lo que el código para acceder al proveedor, recuperar los contactos y mostrarlos en la lista se tendrá que ejecutar en el método *onCreate()* de la actividad. Lo que haremos en este ejercicio es codificar estas instrucciones en un método a parte que será llamado desde el *onCreate()*. Así es como quedará el código de la actividad:

```
import android.content.ContentResolver;
import android.database.Cursor;
import android.os.Bundle;
import android.provider.ContactsContract.CommonDataKinds.Phone;
import android.provider.ContactsContract.Data;
import android.support.v4.widget.CursorAdapter;
import android.support.v7.app.AppCompatActivity;
import android.widget.ListView;
import android.widget.SimpleCursorAdapter;

public class MainActivity extends AppCompatActivity {

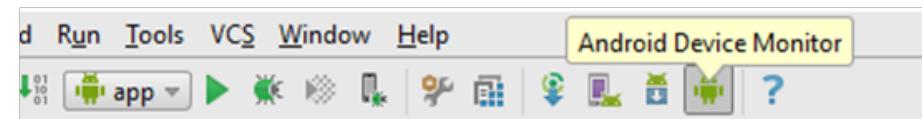
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        cargarContactos();
    }

    public void cargarContactos(){
        //obtenemos el contentresolver para acceder a proveedores
        ContentResolver resolver=this.getContentResolver();
        //recuperamos todos los contactos
        //se establece una condición para que,
        //de todas las combinaciones obtenidas,
        // solamente se queda con aquellas
        //que corresponden al tipo teléfono
        Cursor c=resolver.query(Data.CONTENT_URI,
            null,
            Data.MIMETYPE+"='"+Phone.CONTENT_ITEM_TYPE+"'",
            null,
            null);
        //array con el nombre de las columnas a mostrar
```

Finalmente, es necesario otorgar los siguientes permisos en el AndroidManifest.xml para que la aplicación funcione:

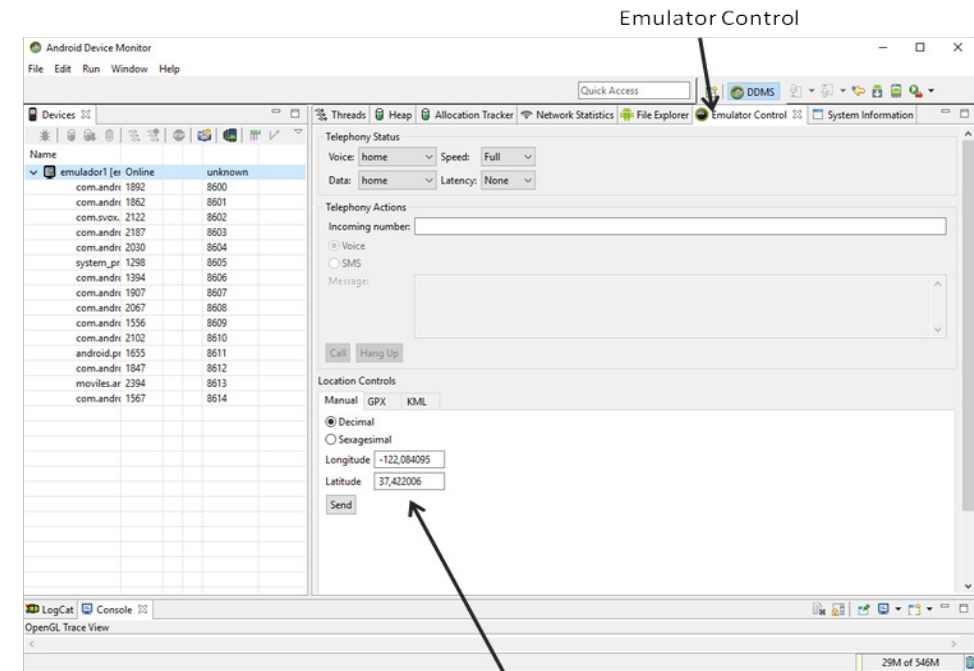
```
import android.content.ContentResolver;
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
<uses-permission android:name="android.permission.INTERNET"/>
```

De cara a probar la aplicación, primeramente pondremos en marcha la aplicación del servidor para que quede a la espera de recibir llamadas externas. Desde la aplicación Android, para simular los cambios de posición desde el emulador tendremos que acceder al Android Device Monitor pulsando el penúltimo botón de la barra de herramientas del IDE:



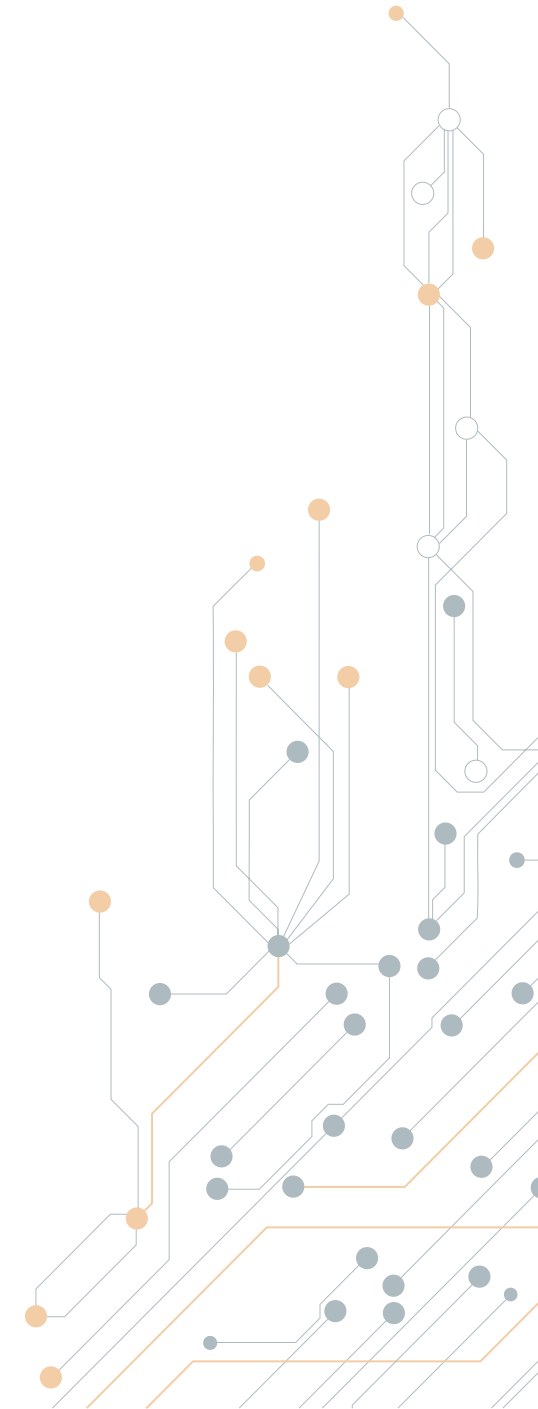
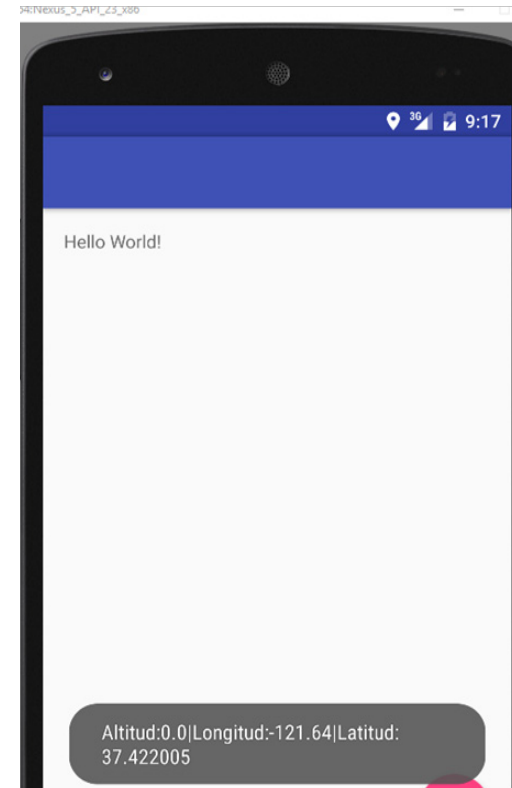


En el cuadro de diálogo que aparece a continuación, nos desplazaremos hasta la pestaña Emulator Control. Ahí, en la parte inferior tenemos unos campos de texto para introducir las coordenadas que queremos establecer en el emulador.



Coordenadas de posición

Dichas coordenadas serán establecidas al pulsar el botón send, lo que provocará el envío de los datos al servidor y que se muestre un cuadro de texto en la pantalla con los valores de posición:



## 5. Google Maps

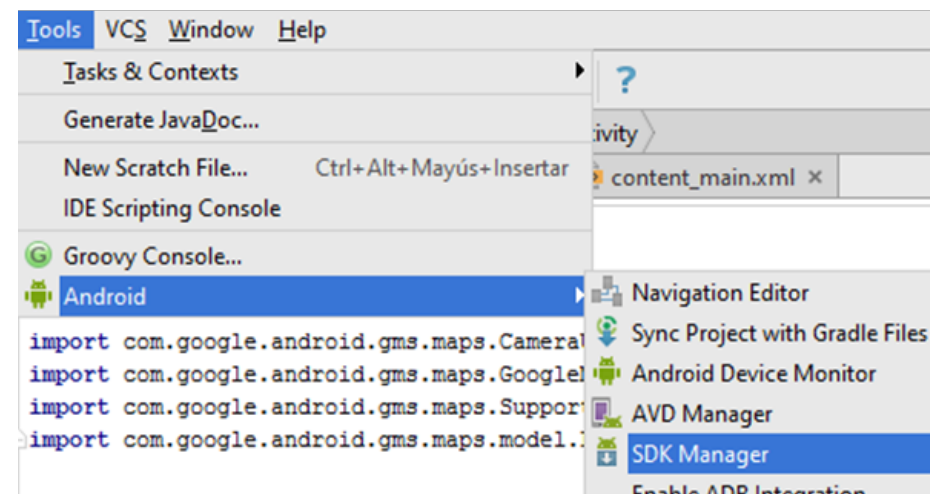
Ya que hemos estado tratando el tema de la geolocalización, vamos a abordar otro interesantísimo aspecto de la programación de aplicaciones para Android que es el uso del API de Google maps, con el que podremos incorporar mapas a nuestras aplicaciones.

Sin embargo, la utilización de la versión 2 de google maps requiere la realización de una serie de tareas previas de obtención de claves y configuración de la aplicación que vamos a describir a continuación.

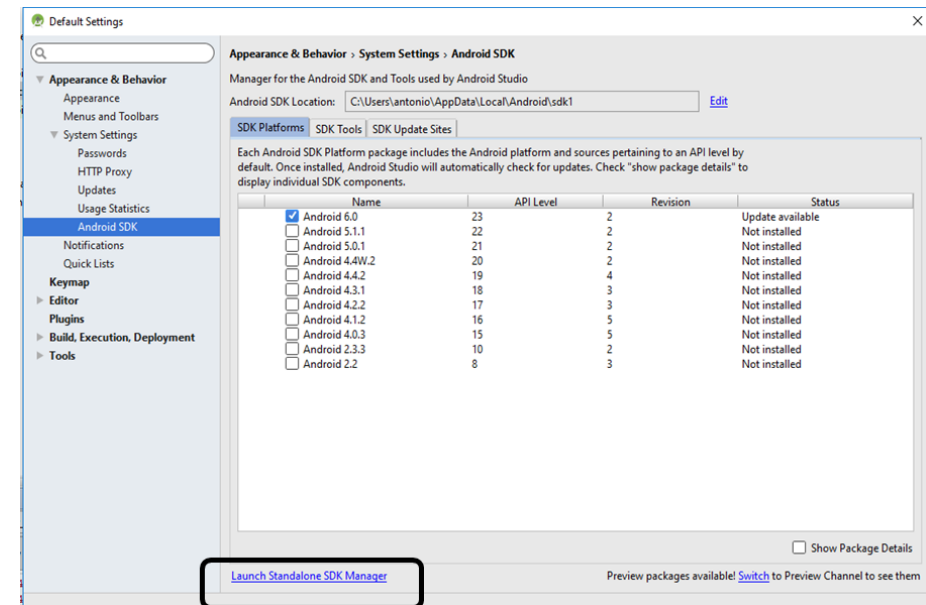
### 5.1 | Google play service

Lo primero que haremos será verificar que contamos con las librerías necesarias para utilizar este API. Estas librerías forman parte del **Google Play Service**, por tanto, nos vamos a ir al SDK Manager y vamos a comprobar si las tenemos instaladas y, en caso de que no sea así, procederemos a su instalación.

Podemos acceder al SDK Manager desde el menú de Android Studio, opción Tools->Android->SDK Manager



En el cuadro de diálogo que aparece a continuación pulsamos sobre el enlace “Launch Standalone SDK Manager” que aparece en la parte inferior de dicho cuadro:



Se abrirá el SDK Manager, nos desplazaremos a la zona inferior y verificaremos que tenemos instalado el Google play service y el Google repository:

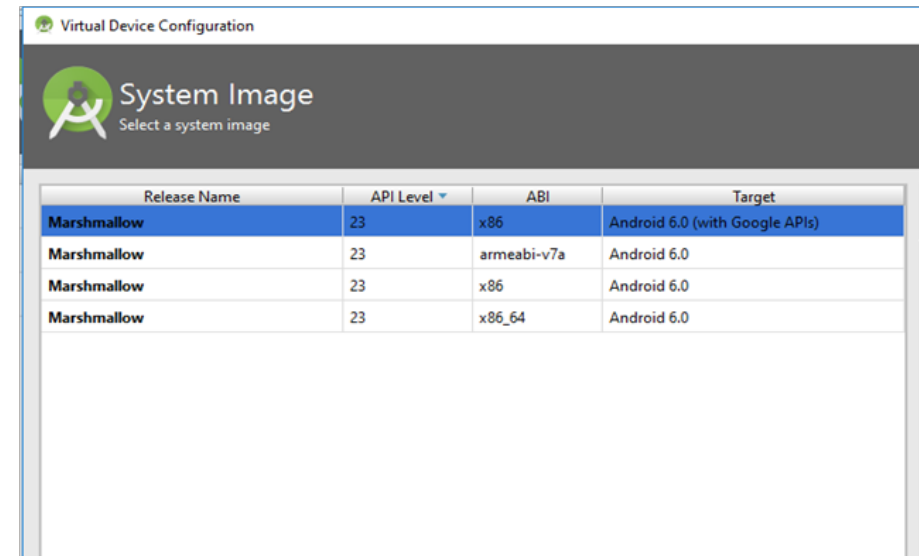
<input type="checkbox"/>		Google Play services	29	<input checked="" type="checkbox"/> Installed
<input type="checkbox"/>		Google Repository	24	<input checked="" type="checkbox"/> Installed
<input type="checkbox"/>		Google Play APK Expansion Library	3	<input type="checkbox"/> Not installed

Si no lo tenemos instalado, marcamos las opciones y pulsamos el botón install para instalarlas.

Además de esto, para probar nuestras aplicaciones basadas en google maps, necesitamos un emulador que incluya el soporte para google play service, así pues, nos desplazaremos a Tools->Android->AVD Manager y crearemos un nuevo dispositivo virtual en el cuadro de diálogo que aparece a continuación.



Tras elegir el tipo de dispositivo, en el cuadro de diálogo “Virtual Device Configuration”, elegiremos la primera opción, que es la que incluye el api de google. Finalmente, le daremos un nombre al nuevo dispositivo virtual y ya lo tendremos listo para probar las aplicaciones.



## Obtención de la clave de google maps

Para poder utilizar el api google maps v2 en una aplicación Android, necesitamos obtener una clave de google que luego tendremos que proporcionar en el archivo de manifiesto de nuestra aplicación.

La obtención de esta clave es un proceso bastante largo y pesado, pero que solo tendremos que realizar una sola vez.

Lo primero que necesitamos para obtener la clave es la **huella del certificado digital de depuración** utilizado por nuestro entorno de desarrollo, para ello, nos desplazamos a la carpeta .android de nuestro usuario, y ahí encontraremos un archivo llamado debug.keystore. Necesitamos la ruta de este archivo para el procedimiento que realizaremos a continuación, por lo que podemos copiarla en el portapapeles. En windows esa ruta será algo similar a esto:

```
c:\users\nombre_usuario\.android\debug.keystore
```

Seguidamente, utilizando la línea de comandos, nos desplazaremos hasta la carpeta \bin del jdk, que estará en una ubicación similar a esta:

```
C:\Program Files\Java\jdk1.8.0_11\bin
```

allí debemos ejecutar el programa el programa **keytool** para extraer la huella digital del archivo keystore, para lo cual teclearemos lo siguiente:

```
>keytool -v -list -keystore c:\users\nombre_usuario\.android\debug.keystore
```

Si nos pide una contraseña la dejaremos en blanco.

Tras ejecutar el comando, se mostrará en la consola lo siguiente:

```
Símbolo del sistema
Número de serie: 2511b242
Válido desde: Sun Nov 08 21:13:26 CET 2015 hasta: Tue Oct 31 21:13:26 CET 2045
Huellas digitales del Certificado:
    MD5: 44:5C:13:58:19:7E:80:06:1F:7D:60:97:48:56:2C:63
    SHA1: 59:A3:82:11:F4:2D:AE:F4:DE:3E:DE:59:37:D2:C5:1D:E7:F0:B5:D7
    SHA256: 4C:7B:8B:CF:03:B4:E3:D1:B2:92:7E:53:C1:A2:36:70:44:05:60:6F:D6:
    D0:26:0A:18:2A:A4:9A:5E:36:7E:05
    Nombre del Algoritmo de Firma: SHA256withRSA
    Versión: 3

Extensiones:

#1: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: E3 7B C2 CC 8A B5 6A 38  2D 50 2D B1 FC 45 3C 59  .....j8-P...E<Y
0010: BB 99 1C 16                ....
]
]
]
.....
.....
```





Como vemos, en la parte superior nos muestra las huellas digitales del certificado. Nosotros necesitamos la huella SHA1, por lo que copiaremos al portapapeles la secuencia de caracteres correspondiente a esa codificación:

**59:A3:82:11:F4:2D:AE:F4:DE:3E:DE:  
59:37:D2:C5:1D:E7:F0:B5:D7**

Ahora que tenemos la huella, iremos a la consola de desarrollador de google para obtener nuestra clave. El acceso a la consola se realiza a través de la siguiente dirección:

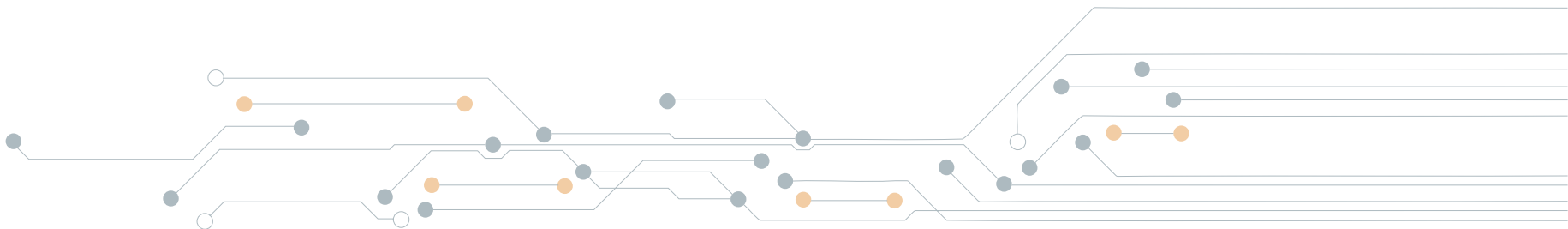
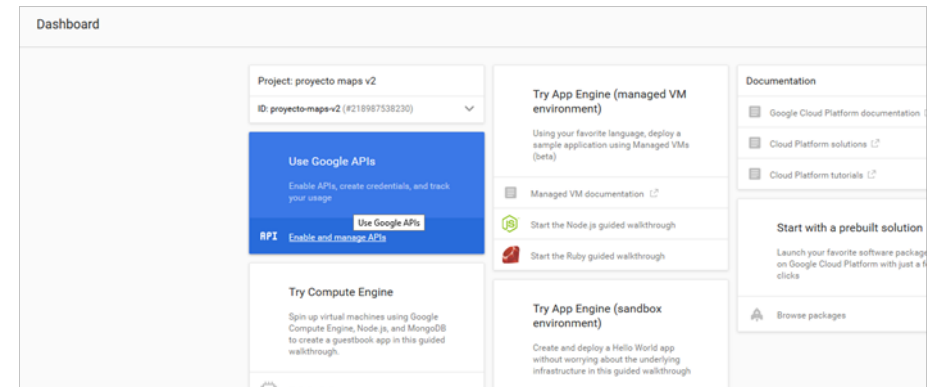
**<https://code.google.com/apis/console/>**

Para acceder a esta dirección necesitamos tener un usuario de google. Si al acceder nos pide que nos autentiquemos, le proporcionaremos los credenciales de nuestro usuario .

Una vez en la consola, crearemos un nuevo proyecto pulsando el botón "create project" y nos pedirá el nombre del proyecto, donde le daremos, por ejemplo, "proyecto maps v2".



Una vez creado el proyecto, nos aparece la siguiente página:



En ella pulsaremos sobre el cuadro "Use google APIs", y en la página de opciones que nos aparece a continuación, pulsaremos sobre el enlace "Google Maps Android API":

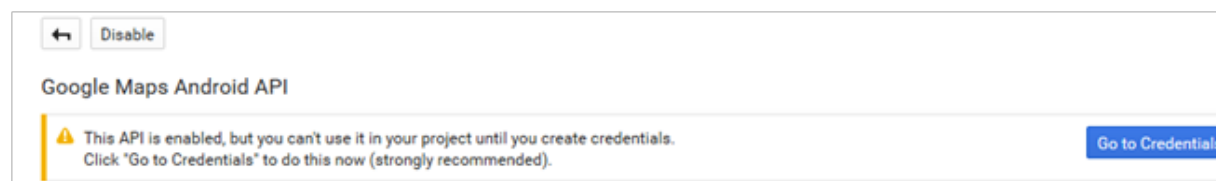
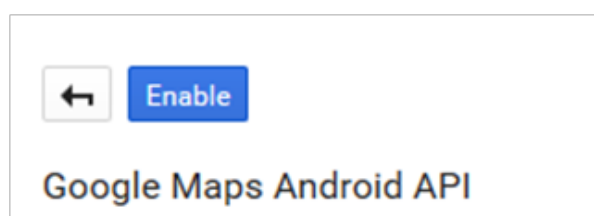
Pulsamos aquí

The screenshot displays the Google APIs console interface. At the top, there's a search bar labeled "Search all 100+ APIs". Below it, the "Popular APIs" section is visible. The APIs are organized into several categories, each with a representative icon:

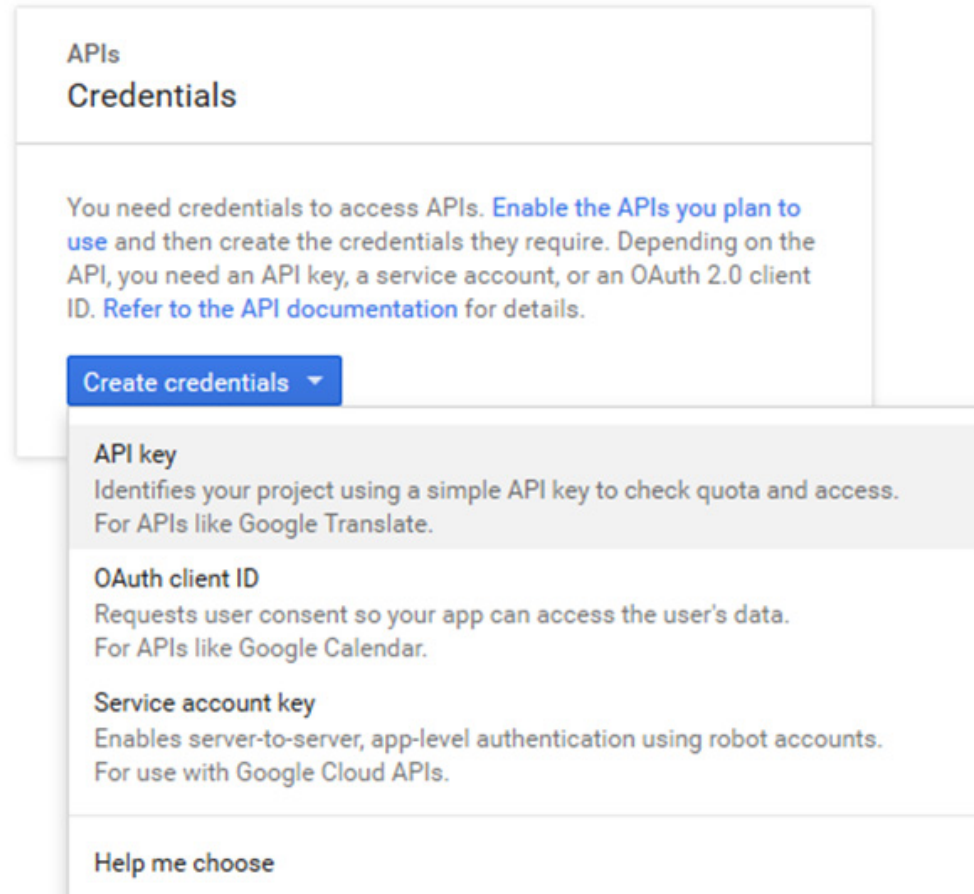
- Google Cloud APIs** (Google Cloud icon): Compute Engine API, BigQuery API, Cloud Storage Service, Cloud Datastore API, Cloud Deployment Manager API, Cloud DNS API, and a "More" link.
- Google Maps APIs** (Google Maps icon): Google Maps Android API (highlighted with an arrow), Google Maps SDK for iOS, Google Maps JavaScript API, Google Places API for Android, Google Places API for iOS, Google Maps Roads API, and a "More" link.
- Google Apps APIs** (Google Apps icon): Drive API, Calendar API, Gmail API, Google Apps Marketplace SDK, Admin SDK, Contacts API, and CalDAV API.
- YouTube APIs** (YouTube icon): YouTube Data API, YouTube Analytics API, and YouTube Reporting API.
- Advertising APIs** (Advertising icon): AdSense Management API, DCM/DFA Reporting And Trafficking API, Ad Exchange Seller API, Ad Exchange Buyer API, DoubleClick Search API, and DoubleClick Bid Manager API.
- Other popular APIs** (API icon): Analytics API, Translate API, Custom Search API, URL Shortener API, PageSpeed Insights API, Fusion Tables API, and Web Fonts Developer API.

Seguidamente, se nos mostrará una página en la que debemos pulsar sobre el botón "Enable" para habilitar el uso de este.

Tras ello, quedará habilitado el uso del API y nos aparecerá en la página un botón "Go to Credentials" que pulsaremos a continuación.



En la página que nos aparece a continuación, pulsaremos el botón "Cancel" para saltar este paso. Después, aparecerá una nueva página con una lista "Create Credentials" en la que elegiremos la opción "API Key":



**APIs**  
**Credentials**

You need credentials to access APIs. [Enable the APIs you plan to use](#) and then create the credentials they require. Depending on the API, you need an API key, a service account, or an OAuth 2.0 client ID. [Refer to the API documentation](#) for details.

Create credentials ▾

- API key**  
Identifies your project using a simple API key to check quota and access.  
For APIs like Google Translate.
- OAuth client ID**  
Requests user consent so your app can access the user's data.  
For APIs like Google Calendar.
- Service account key**  
Enables server-to-server, app-level authentication using robot accounts.  
For use with Google Cloud APIs.

[Help me choose](#)

En el cuadro de diálogo que se muestra a continuación, pulsaremos sobre el botón “Android Key”.

### Create a new key

You need an API key to call certain Google APIs. The API key identifies your project. Also, it is used to enforce quotas and handle billing, so keep it safe.

Server key

Browser key

Android key

iOS key



Finalmente, nos pedirá un nombre para la clave que vamos a generar, la huella digital de nuestro certificado y un nombre de paquete:

### Credentials



#### Create Android API key

Name

Android key 1

#### Restrict usage to your Android apps (Optional)

Android devices send API requests directly to Google. Google verifies that each request comes from an Android app that matches a package name and SHA-1 signing-fingerprint name that you provide. Get the package name from your AndroidManifest.xml file. Use the following command to get the fingerprint. [Learn more](#)

```
keytool -list -v -keystore mystore.keystore
```

Package name

claves.ejercicio

SHA-1 certificate fingerprint

59:A3:82:11:F4:2D:AE:F4:DE:3E:DE:59:37:D2:C5:1D:E7:F0:B9:D7



+ Add package name and fingerprint

Note: It may take up to 5 minutes for settings to take effect

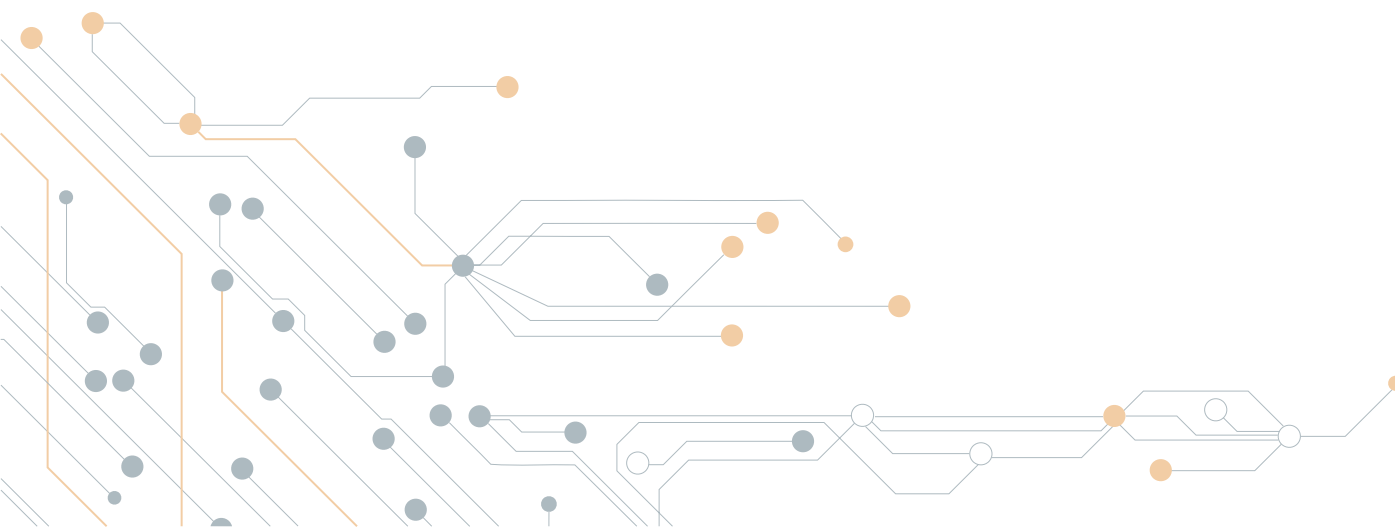
Create

Cancel

Huella digital

Tras pulsar el botón “Create”, por fin se generará nuestra clave:

Conviene que copiemos el valor de la misma en algún archivo de texto a fin de poder utilizarla en los diferentes proyectos en los que vayamos a hacer uso de ella.





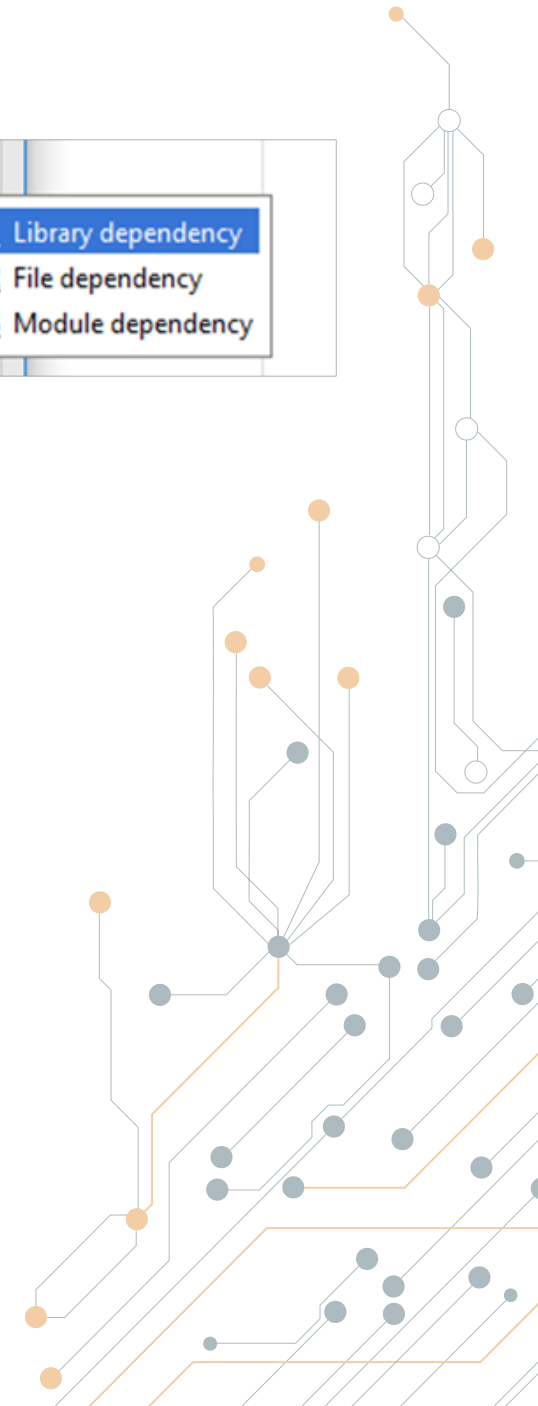
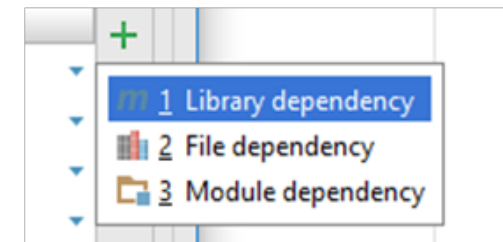
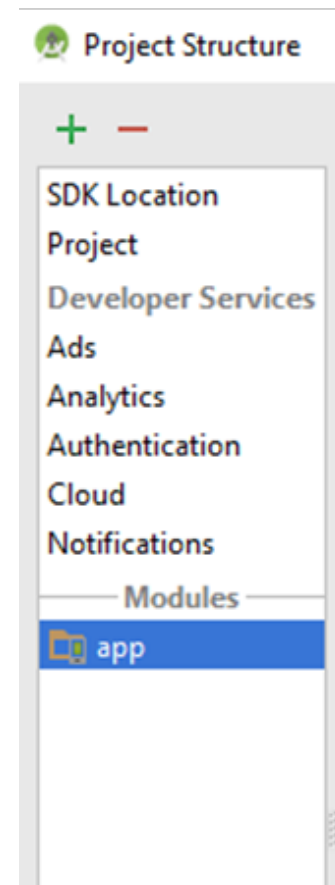
## 5.2 | Creación del proyecto para google maps

Ahora ya podemos crear un proyecto en el que podremos hacer uso del api google maps. Procedemos a crear un proyecto normal con Android Studio, con una actividad en blanco.

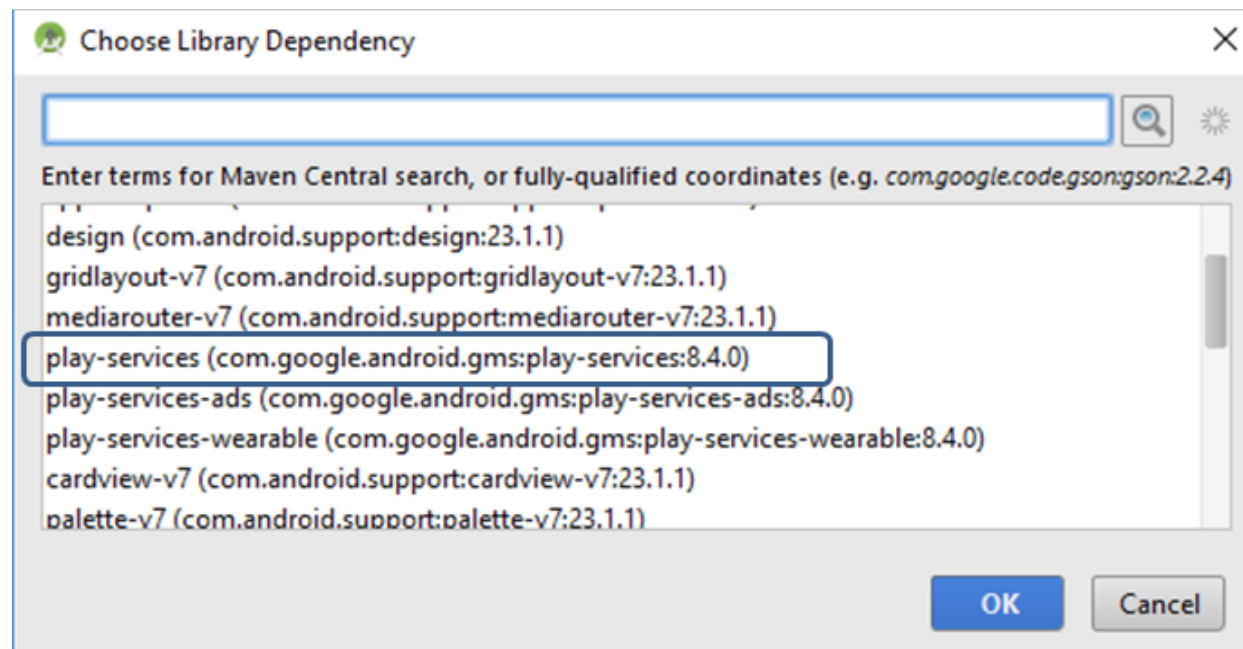
Una vez creado el proyecto, debemos **incorporar al mismo las librerías del Google play service** que ya tenemos instaladas a través del SDK Manager. Para hacer esto, nos vamos a la opción de menú File->Project Structure.

En el cuadro de diálogo que nos aparece a continuación, seleccionamos la pestaña "Dependencies" y dentro de esta, pulsamos sobre la opción App que se encuentra debajo de "Modules" en el menú de la izquierda:

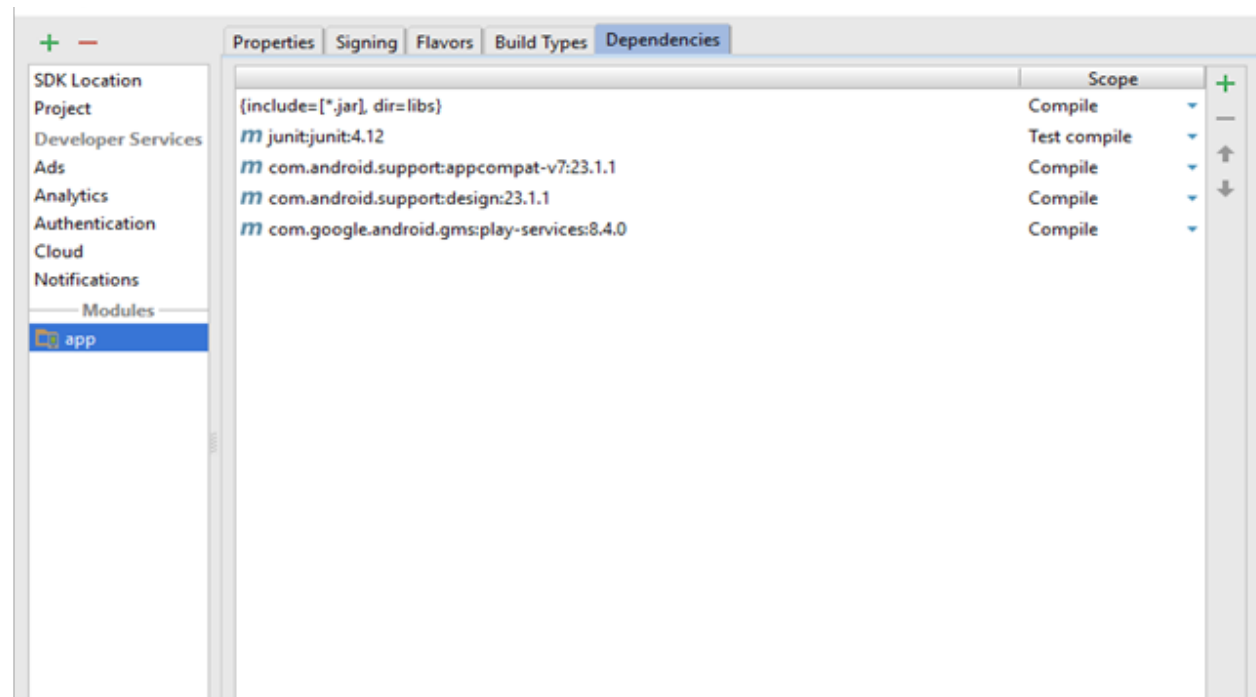
Se nos mostrará una lista con las dependencias de nuestro proyecto. Debemos pulsar el botón + que aparece en la zona superior derecha y elegir "Library dependency"



En la lista que aparece a continuación, elegiremos play services.

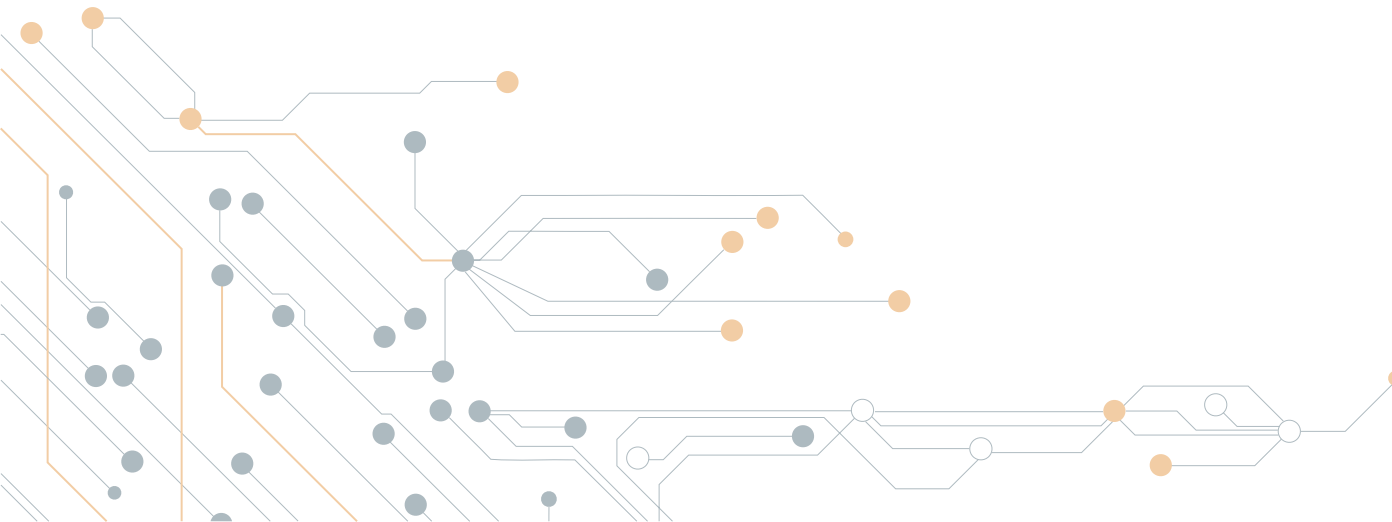


Veremos como la librería se incorpora a la lista de dependencias de nuestro proyecto:



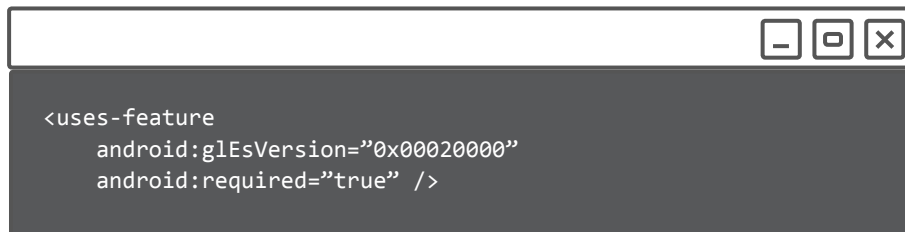
Una vez incorporadas las dependencias, debemos realizar una serie de ajustes en el archivo de manifiesto. Lo primero, será incorporar la lista de permisos:

```
<permission
    android:name="moviles.android.ejemplo_mapas.permission.MAPS_RECEIVE"
    android:protectionLevel="signature"/>
<uses-permission
    android:name="moviles.android.ejemplo_mapas.permission.MAPS_
RECEIVE"/>
<uses-permission
    android:name="com.google.android.providers.gsf.permission.READ_
GSERVICES"/>
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_
STORAGE"/>
<uses-permission android:name="android.permission.ACCESS_COARSE_
LOCATION"/>
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
```



La primera línea corresponde a la definición del permiso MAPS\_RECEIVE para nuestro proyecto. Observa que la parte de la ruta `mobiles.android.ejemplo_mapas` correspondería con el paquete principal del proyecto, el resto de líneas corresponden a los permisos que hay que otorgar a la aplicación

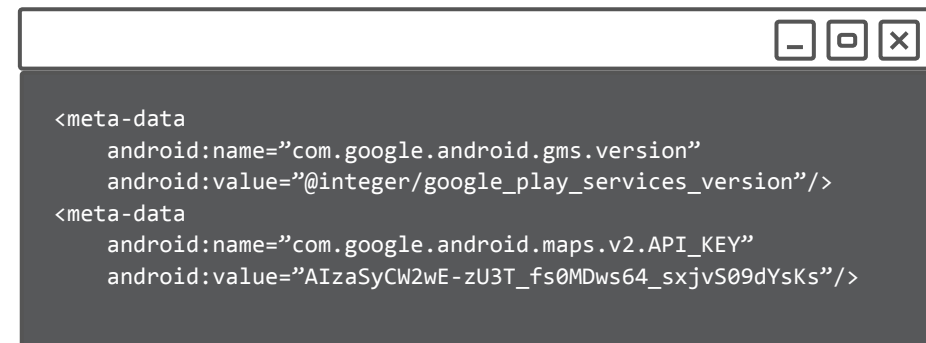
A continuación de los permisos, incluimos el siguiente elemento:



```
<uses-feature
    android:glEsVersion="0x00020000"
    android:required="true" />
```

Que sirve para informarle a Android que el mapa hará uso de la librería gráfica OpenGL.

Dentro del elemento `<application>`, antes de la declaración de la actividad, incluimos las siguientes etiquetas:



```
<meta-data
    android:name="com.google.android.gms.version"
    android:value="@integer/google_play_services_version"/>
<meta-data
    android:name="com.google.android.maps.v2.API_KEY"
    android:value="AIzaSyCW2wE-zU3T_fs0MDws64_sxjvS09dYsKs"/>
```

La segunda de ellas es la que proporciona nuestra clave que hemos generado en google para poder hacer uso del google maps API.

En cuanto al archivo de plantilla, incluiremos en el interior del layout principal la siguiente etiqueta:

```
<fragment
    android:id="@+id/map"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    class="com.google.android.gms.maps.SupportMapFragment"/>
```

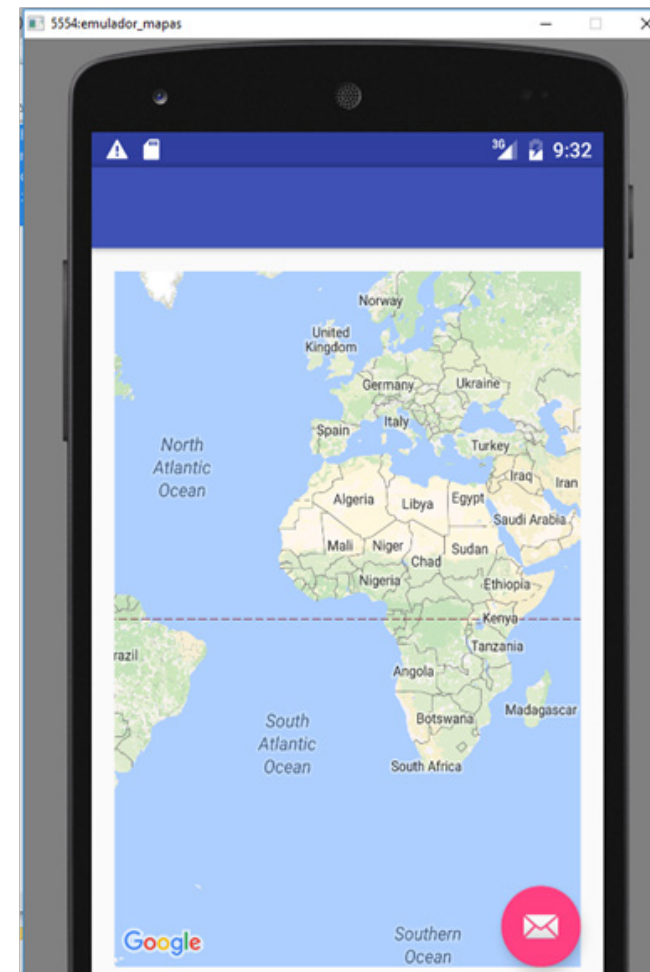
Y es que el mapa se define en Android dentro de lo que se conoce como un fragmento.

Por último, nos desplazaremos al código de la actividad y cambiaremos la definición de la misma para que extienda de `FragmentManager` en vez de `Activity`:

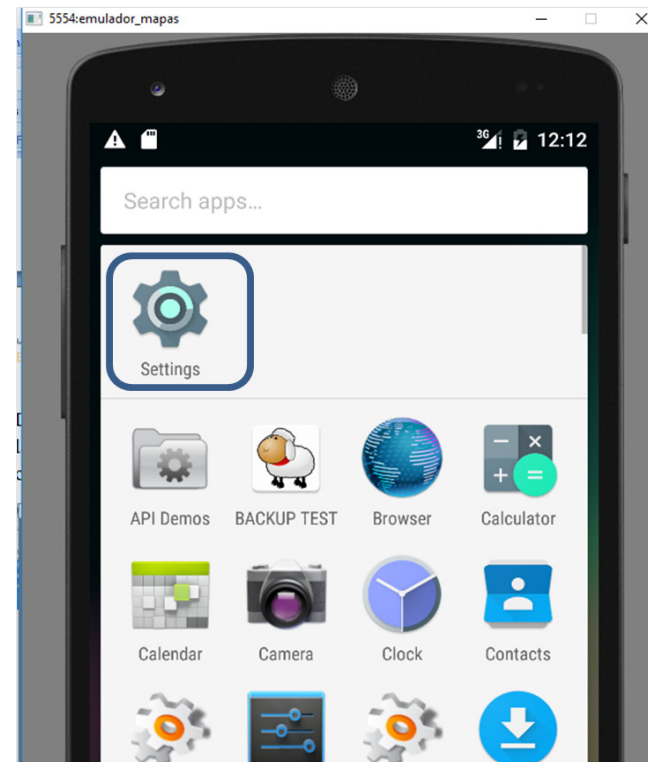
```
public class MainActivity extends FragmentActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

Lo único que hemos definido en el onCreate() de la actividad es la carga de la plantilla, por lo que al ejecutarse el programa en el emulador que hemos crear con soporte para google maps, se tendrá que mostrar lo siguiente:



Como nos ha sucedido alguna otra vez en la que hemos requerido permisos en la aplicación, es posible que a pesar de haberlos asignado en el `AndroidManifest.xml` el programa falle al ejecutarse en un emulador (no en un terminal real). Si es así, habrá que ir a la configuración "Settings" del emulador y añadir manualmente el permiso como ya hiciéramos en otros ejercicios anteriores.





## 5.3 | Utilización API google maps

Una vez que ya hemos visto todos los pasos necesarios para poder utilizar google maps, vamos a comentar alguno de los elementos más importantes de este API que nos permitan realizar algunas tareas interesantes, como posicionarnos en una determinada localización.

Las clases más importantes se encuentran en el paquete **com.google.android.gms.maps**. La clase principal de este paquete es `GoogleMap`, que representa el punto de entrada para manipular el mapa. Para obtener un objeto de esta clase, necesitaremos realizar lo siguiente:

**1. Obtener objeto `FragmentManager`.** Dado que el mapa se va a visualizar sobre un fragmento, necesitamos un objeto de esta clase porque es el que nos permite manipular fragmentos desde código. Esta clase se encuentra en el paquete de soporte `android.support.v4.app`. Podemos obtener un `FragmentManager` a través del método `getSupportFragmentManager()` de `FragmentActivity`:

```
FragmentManager fm=this.  
getSupportFragmentManager();
```

**2. Obtener objeto `SupportMapFragment`.** Este objeto es el que nos dará acceso al `GoogleMap`. Para obtener un `SupportMapFragment`, recurriremos al método `findFragmentById()` de `FragmentManager`, al que le pasaremos el id del fragmento definido en el archivo de plantilla:

```
SupportMapFragment smf=(SupportMapFragment)  
fm.findFragmentById(R.id.map);
```

**3.** Obtener GoogleMap. Utilizando el método `getMap()` de `SupportMapFragment` obtenemos el objeto `GoogleMap` para poder manipular el mapa:

```
GoogleMap gm=smf.getMap();
```

A partir de ahí, podemos utilizar los diversos métodos de `GoogleMap`, entre los que destacamos:

- **`setMapType()`**. Establece la forma en la que debería verse el mapa. Entre los posibles valores están las constantes:



- `MAP_TYPE_NORMAL`. Vista estándar del mapa. Es el modo predeterminado
- `MAP_TYPE_SATELLITE`. Vista satélite con indicación de calles
- `MAP_TYPE_TERRAIN`. Vista real
- **`getUISettings()`**. Devuelve un objeto `UISettings` para establecer características de visualización, por ejemplo, llamando al método *`setZoomControlsEnabled()`* sobre este objeto con el valor `true`, activamos el control de zoom.
- **`moveCamera()`**. Permite desplazar la cámara a una determinada posición, haciendo que el mapa nos muestre esa posición. Como parámetro le pasaremos un objeto `CameraUpdate` apuntando a la posición que queremos mostrar.

Si en el ejercicio de ejemplo anterior quisiéramos posicionar el mapa en una determinada localización, por ejemplo de la provincia de Zaragoza, este sería el código de la actividad:

```
public class MainActivity extends FragmentActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        FragmentManager fm=this.getSupportFragmentManager();
        SupportMapFragment smf=(SupportMapFragment)
            fm.findFragmentById(R.id.map);
        GoogleMap gm=smf.getMap();
        gm.setMapType(GoogleMap.MAP_TYPE_NORMAL);
        gm.getUISettings().setZoomControlsEnabled(true);
        gm.moveCamera(CameraUpdateFactory.newLatLngZoom(
            new LatLng(41.56, -0.7), 10));

    }

}
```

Como vemos, a través del método `newLatLngZoom()` de `CameraUpdateFactory` obtenemos un objeto `CameraUpdate` apuntando a una determinada latitud y longitud a un nivel de zoom de 10.

*Telefonica*

---

EDUCACIÓN DIGITAL