Tema 10. Ubicación

Prof. Manuel Castillo

Programación de Dispositivos Móviles

Escuela Profesional de Ciencias de la Computación

Facultad de Ciencias

Universidad Nacional de Ingeniería

Objetivos

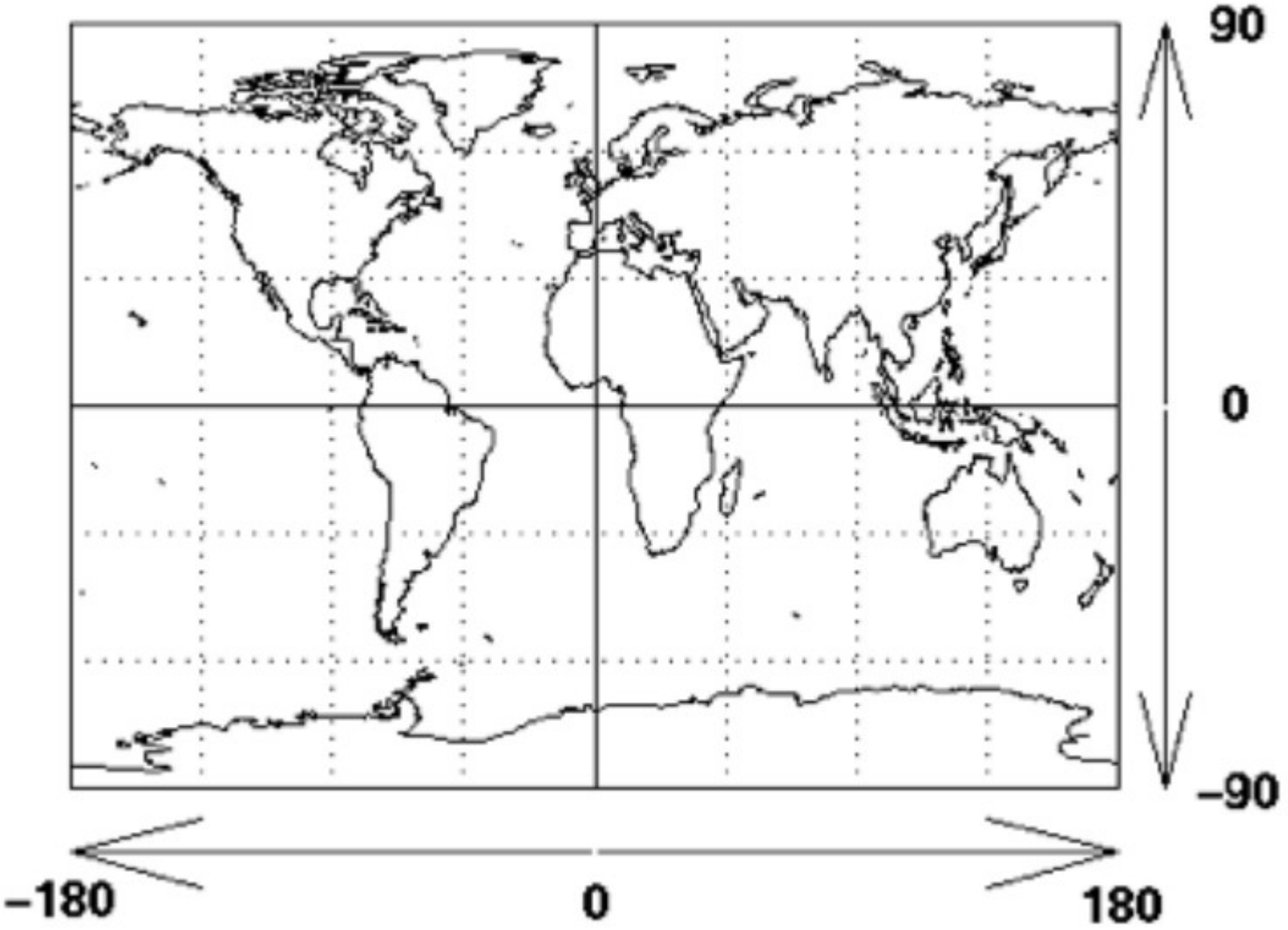
- Conocer las bases de la ubicación en Android.
- Saber cómo crear un proyecto que incluya mapas.
- Saber cómo geolocalizar un dispositivo Android.

Índice de contenido

1. Introducción

- A parte de estos sensores habituales tenemos otro tipo de componentes hardware que nos permiten saber dónde estamos:
  - Chip *GPS* y *GLONASS*. *GLONASS* es el sistema de posicionamiento ruso, se combina con el *GPS* para mejorar el posicionamiento del dispositivo.
  - Brújula Digital.
- Aunque también hay otros medios de geolocalizar los dispositivos como son:
  - Redes *Wi-Fi*
  - Redes móviles

1. Introducción



1.1. Proveedores

- Obener proveedores disponibles en el dispositivo es mediante una llamada al método *getAllProviders()* de la clase *LocationManager*.

```
LocationManager locManager = (LocationManager)
      getSystemService(LOCATION_SERVICE);
List<String> listaProviders = locManager.getAllProviders();
```

1.2. Criterios de búsqueda

- Indica las características mínimas del proveedor que necesitamos utilizar.

```
//  buscar uno con precisión alta y que nos
// proporcione la altitud definiríamos el siguiente criterio de búsqueda:
Criteria req = new Criteria();
req.setAccuracy(Criteria.ACCURACY_FINE);
req.setAltitudeRequired(true);

//Mejor proveedor por criterio
String mejorProviderCrit = locManager.getBestProvider(req, false);
//Lista de proveedores por criterio
List<String> listaProvidersCrit = locManager.getProviders(req, false);
```

Nos devuelve los que están activados

1.3. Posición actual

- Activar el proveedor de localización.
- Suscribirnos a sus notificaciones de cambio de posición.
- Se realiza mediante mediante una llamada al método *requestLocationUpdates()*, al que deberemos pasar 4 parámetros distintos:
  - Nombre del proveedor de localización al que nos queremos suscribir.
  - Tiempo mínimo entre actualizaciones, en milisegundos.
  - Distancia mínima entre actualizaciones, en metros.
  - Instancia de un objeto *LocationListener*, que tendremos que implementar previamente para definir las acciones a realizar al recibir cada nueva actualización de la posición.

1.4. Objeto *LocationListener* (I)

- Una vez inicializado el localizador podemos esperar que también nos avise cuando suceda algún cambio en la posición en un determinado tiempo límite que pongamos
  - Debemos implementar el interfaz *LocationListener*
  - Deberemos implementar el método *onLocationChanged(Location)*
    - Donde nos pasan un objeto con la latitud y la longitud como en el caso del *getLastLocation*().

1.4. Objeto *LocationListener* (III)

- Inicialización y parada del cliente de actualizaciones
- Para activar el sistema de monitorización de la posición
  - *requestLocationUpdates*
- Para parar la monitorización
  - *removeLocationUpdates*
- En caso de cambiar la posición
  - *onLocationChanged(Location location)*
    - *location.getLongitude()*
    - *location.getLatitude()*

1.4. Objeto *LocationListener* (IV)


```
LocationListener locListener = new LocationListener() {
    public void onLocationChanged(Location location) {
      mostrarPosicion(location);
    }
    public void onProviderDisabled(String provider){
      lblEstado.setText("Provider OFF");
    }
    public void onProviderEnabled(String provider){
      lblEstado.setText("Provider ON");
    }
    public void onStatusChanged(String provider,
          int status, Bundle extras){
      lblEstado.setText("Provider Status: " + status);
    }
};
```

Lanzado cada vez que se recibe una actualización de la posición.

Lanzado cuando el proveedor se deshabilita.

Lanzado cuando el proveedor se habilita.

Lanzado cada vez que el proveedor cambia su estado, que puede variar entre OUT_OF_SERVICE, TEMPORARILY_UNAVAILABLE, AVAILABLE.

2. Creación del proyecto

- Creamos un proyecto nuevo
  - Indicamos que será para la versión 2.3 con los *GOOGLE API*.
  - Incluimos una *activity* del tipo *blank activity.*
- Con esto deberíamos tener un proyecto simple para comenzar el desarrollo.

3. Importación de las dependencias de mapas (I)

- Primero deberemos descargas los *SDK* y las bibliotecas que necesitamos para que funcionen bien los proyectos de mapas
- Abrimos el *SDK Manager*. Necesitamos descargar dos componentes principales.
  - *Android Support Repository*
  - *Android Support Library*
  - *Google Play services*
  - *Google Repository*

3. Importación de las dependencias de mapas (II)

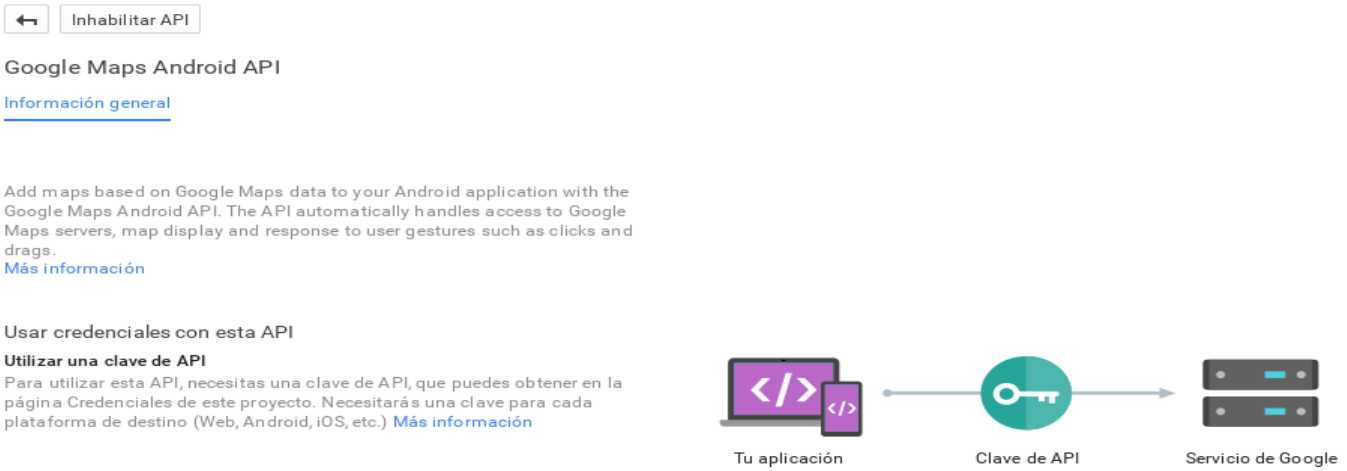
- Deberemos conseguir una API Key
  - https://desarrollandoandroid.wordpress.com/2014/06/18/obteniendo-la-api-key-de-google/
- Una vez copiada debemos pegar nuestra API Key en su lugar correspondiente en el *AndroidManifest.xml* en Android Studio.
- También deberemos de incluir dentro de la etiqueta <*application*> del *AndroidManifest.xml*
  - *<meta-data android:name="com.google.android.maps.v2.API_KEY" android:value=YOUR_KEY_HERE">*

5. Google API's: Mapas para Android (I)

- Ahora es cuando queremos generar una clave para el uso de los mapas en Android
- Es fundamental disponer de una Google ID para podernos loguear en la consola
- Para entrar a la Consola de Google API's iremos:
  - *https://console.developers.google.com*
  - Dentro de la consola podemos dar de alta nuevos proyectos mediante el botón *Create Project*
    - Pondremos un nombre
    - Nos generará un *Project ID*
    - Pulsamos en el botón *Create*
  - Una vez creado nos pasará a la configuración del proyecto

5. Google API's: Mapas para Android (II)

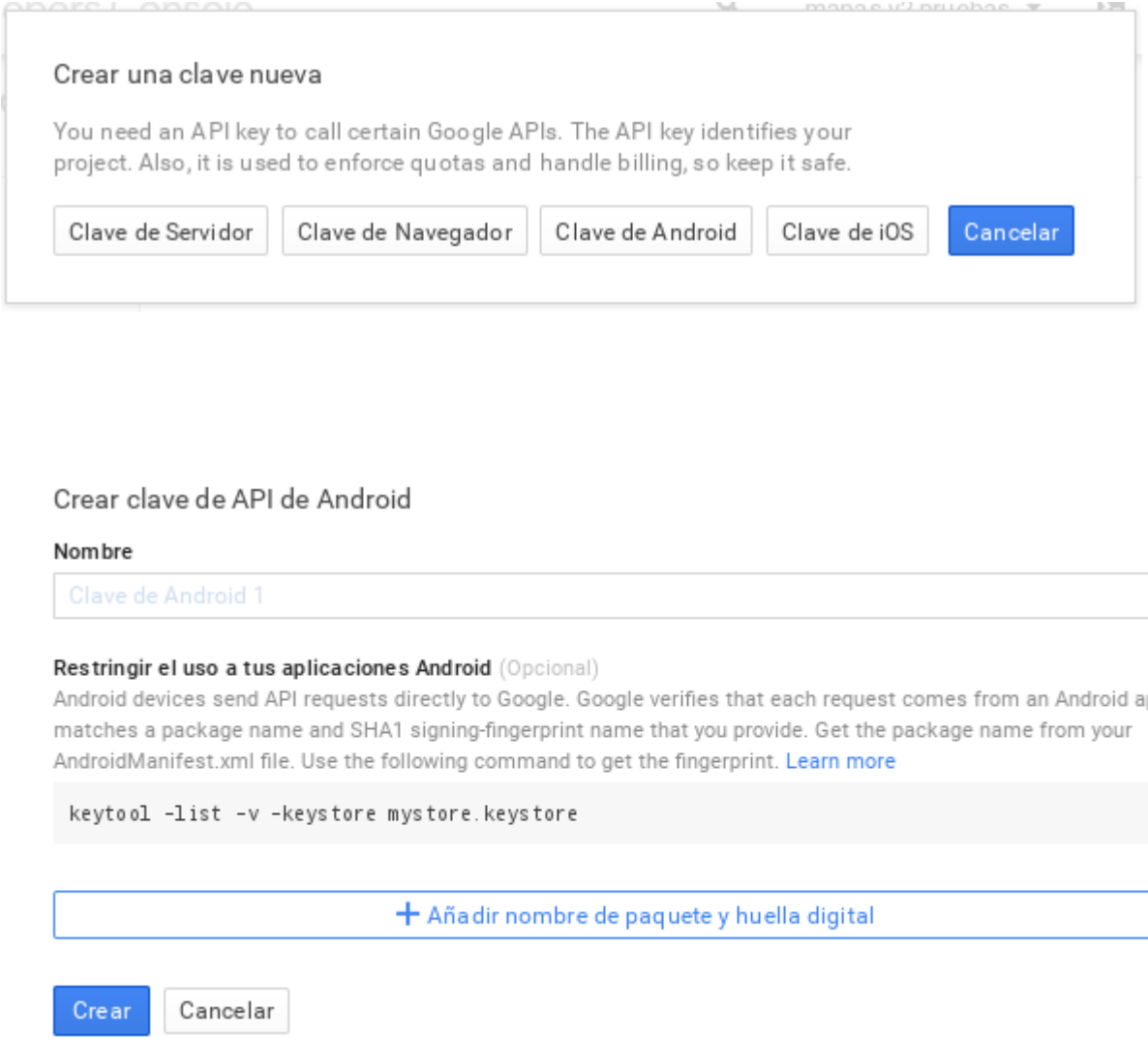
- En la configuración del proyecto de *API's*.
  - En el *API Google Maps Android API*
    - Pulsamos el botón de la derecha para que ponga *ON* en verde
  - Esto permitirá acceder al *API* de Mapas, como podemos ver no tiene ninguna cuota de uso, esto significa que no debemos de pagar por tener acceso a los mapas desde Android
  - Aquí podríamos activar otros servicios que necesitáramos para acceder desde nuestro proyecto desde los *Google API's*.



5. Google API's: Mapas para Android (III)

- En el apartado de *Credentials* → Dentro de la parte de *Public API Access* → Deberemos pulsar en *Create new Key*
  - Seleccionamos Android Key.
  - Ahora nos pide introducir una huella SHA1 de nuestra aplicación y el nombre del paquete principal separados por un ";"
- Para conseguir el *SHA1* podemos utilizar el del certificado que utilicemos a la hora de firmar el *APK* del proyecto desde
  - Preferencias/*Android*/*Build*/*SHA1 FingerPrint*
- Para conseguir el paquete principal basta con copiar el valor desde el *AndroidManifest.xml*
- Una vez introducido el dato pulsamos en *Create*
- Nos generará una clave para la aplicación que estamos haciendo en el campo *API Key*, esta es la cadena que deberemos utilizar en la aplicación.

5. Google API's: Mapas para Android (III)



6. Configuración del Manifesto (I)

- Dentro del *AndroidManifest.xml* deberemos realizar una serie de cambios
  - Inclusión de permisos dentro de la etiqueta <*manifest*>

```
<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"/>

<!-- Los siguientes permisos son para la geolocalización -->
<uses-permission
    android:name="android.permission.ACCESS_COARSE_LOCATION "/>
<uses-permission
    android:name="android.permission.ACCESS_FINE_LOCATION"/>
```

6. Configuración del Manifesto (I)

- Dentro del *AndroidManifest.xml* deberemos realizar una serie de cambios
  - Uso de OpenGL 2 dentro de la etiqueta

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

- Añadir dentro de la etiqueta <*application*>

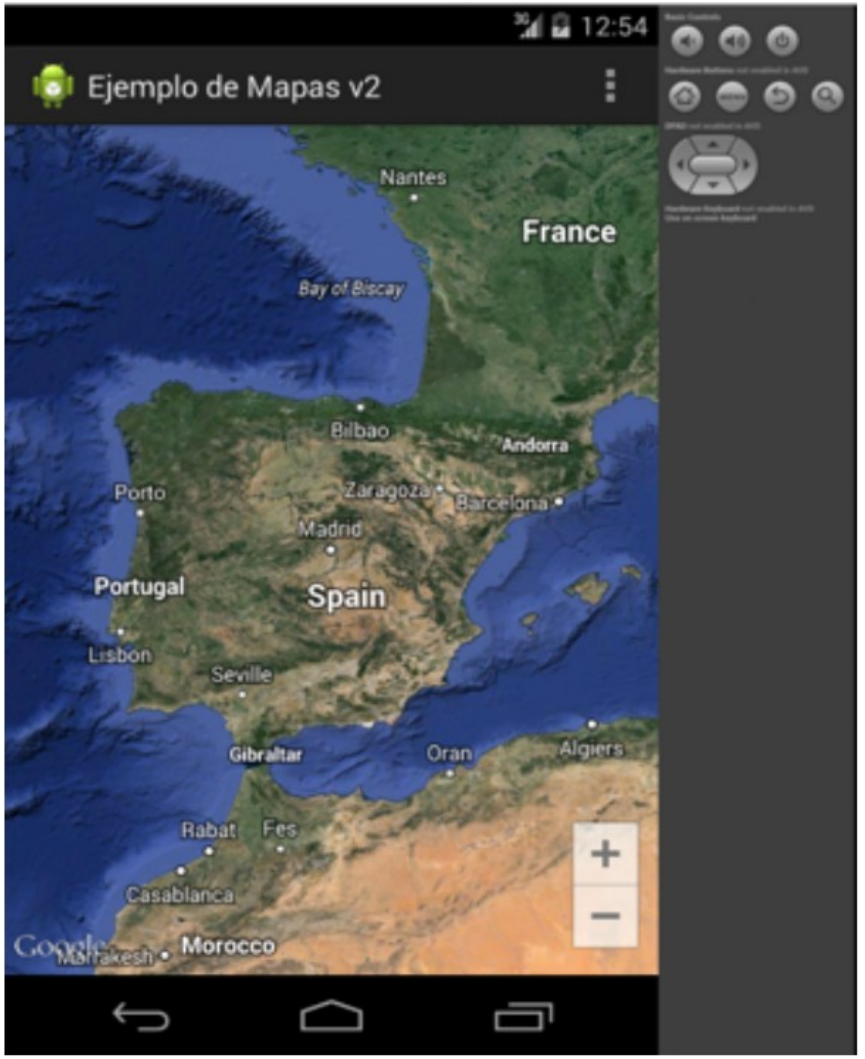
```
<meta-data
    android:name="com.google.android.maps.v2.API_KEY"
    android:value="AQUIVALAAPIKEY" />
<uses-library android:name="com.google.android.maps" />
```

7. Inclusión del mapa en el Layout

- Dentro del fichero del *layout* de la *Activity* deberemos incluir un fragmento que permite incluir el mapa en pantalla

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

Ejecución de la aplicación



8. Mapas

- El objeto *GoogleMap*.
- Se accede llamando al método *getMap()* del fragmento *MapFragment* que contiene nuestro mapa.

```
import com.google.android.gms.maps.GoogleMap;

GoogleMap mapa = ((SupportMapFragment) getSupportFragmentManager()
                .findFragmentById(R.id.map)).getMap();
```

8.1. Modificación del mapa

- A través del método *setMapType()*.
  - *MAP_TYPE_NORMAL*.
  - *MAP_TYPE_HYBRID*.
  - *MAP_TYPE_SATELLITE*.
  - *MAP_TYPE_TERRAIN*.

```
mapa.setMapType(GoogleMap.MAP_TYPE_NORMAL);
mapa.setMapType(GoogleMap.MAP_TYPE_HYBRID);
mapa.setMapType(GoogleMap.MAP_TYPE_SATELLITE);
mapa.setMapType(GoogleMap.MAP_TYPE_TERRAIN);
```

8.2. Objeto *CameraUpdate*

- Movimiento de la cámara
- Cambiar zoom:
  - *CameraUpdateFactory.zoomIn()*. Aumenta en 1 el nivel de zoom.
  - *CameraUpdateFactory.zoomOut()*. Disminuye en 1 el nivel de zoom.
  - *CameraUpdateFactory.zoomTo(nivel_de_zoom)*. Establece el nivel de zoom.
- Actualizar la latitud-longitud
  - *CameraUpdateFactory.newLatLng(lat, long)*. Establece la lat-lng expresadas en grados.
- Modifica los parámetros anteriores en conjunción
  - *CameraUpdateFactory.newLatLngZoom(lat, long, zoom)*. Establece la lat-lng y el zoom.
- Para movernos lateralmente por el mapa (panning) podríamos utilizar los métodos de scroll:
  - *CameraUpdateFactory.scrollBy(scrollHorizontal, scrollVertical)*. Scroll expresado en píxeles.