BRÚJULA Y GPS CON ANDROID

Jorge Chamorro Padial | Germán Iglesias Padial

17/12/14

1. INTRODUCCIÓN

El problema de la geolocalización aborda fundamentalmente dos problemas:

- Conocer la posición en la que se encuentra el usuario, un punto sobre el planeta tierra.
- Conocer la orientación del usuario. Su distancia angular respecto al norte.

Los smartphone de hoy en día suelen incluir dos sensores, el de **localización** y el de **brújula**, que solucionan estos problemas. Ambos sensores van a ser explicados en el tutorial.

En el ejemplo que hemos resuelto se muestra el uso de la brújula y el del sensor de localización. El usuario puede ver sus coordenadas y mostrarlas en un mapa, así como la orientación del mismo.

2. TUTORIAL

El código del tutorial lo tenéis en https://github.com/jorgechpugr/NPI-Android-Practica3

2.1 GPS

Vamos a trabajar con dos sensores de Android, el de la brújula y el de la geolocalización. Lo primero que vamos a hacer es añadir los permisos necesarios, para el caso de la localización, se llama `ACCESS_FINE_LOCATION.`

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

Y ya está, no necesitamos nada más. Es posible que veas que alguna aplicaciones usan también **ACCESS_COARSE_LOCATION,** pero aquí se explica en detalle por qué solamente es necesario un único permiso (http://developer.android.com/guide/topics/location/strategies.html)

Trabajaremos con dos clases para el caso de la localización. "*LocationManager*" y "*LocationListener*". La primera permite el acceso a los servicios de localización de Android, permitiendo actualizar el estado de los mismos de forma periódica (es decir, actualizar la posición del usuario con el paso del tiempo).

Hay que tener en cuenta que **NO SE DEBE** crear un objeto de esta clase. Debemos usar el que nos proporciona el sistema.

```
locManager = (LocationManager) getSystemService(Context.LOCATION_SERVICE);
```

Obtener la posición es muy sencillo. Solamente tenemos que llamar al método getLastKnownLocation().

```
Location loc = locManager
            .getLastKnownLocation(LocationManager.GPS_PROVIDER);
```

Un objeto de la clase Location tiene todos los detalles sobre una localización geográfica. Tiene métodos como getLatitude() o getLongitude(), getAccuracy(), que utilizamos en la aplicación. Pero tiene más como velocidad, tiempo y el proveedor que proporcionó esa ubicación.

También podemos utilizarlo para crear nuestras propias localizaciones, que luego podamos representar sobre un mapa. En el siguiente enlace tienes más información al respecto.

http://developer.android.com/reference/android/location/Location.html

Hasta aquí se ha descrito la forma de acceso estática a la aplicación. Pero es posible que deseemos monitorizar continuamente la posición del usuario (*ejemplo: un navegador para el coche, peatón, bicicleta...*).

En ese caso, entra en juego la clase **LocationListener**, que "escucha" al servicio de geoposicionamiento. Es un sistema basado en eventos, esto quiere decir que determinadas acciones provocan un evento, este se activa y ejecuta un código que nosotros hemos de definir. En Java esto se hace con clases anónimas. Y debemos definir los eventos: ***onLocationChanged*** (cambio de ubicación del usuario), ***onProviderDisabled*** (pérdida de conectividad GPS), ***onProviderEnabled*** (conectividad con GPS), ***onStatusChanged*** (cambio en la conectividad al conectar con un determinado proveedor).

```
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) {
              Log.i("", "Provider Status: " + status);
              lblEstado.setText("Provider Status: " + status);
```

```
        }
    };
```

Ahora solo nos queda añadir el objeto Listener al listado de de LocManager, para que le "avise" cuando se produzcan cambios.

```
    locManager.requestLocationUpdates(LocationManager.GPS_PROVIDER, 30000,
                0, locListener);
```

El método requestLocationUpdates() tiene muchas definiciones, podemos indicarle el proveedor, el tiempo mínimo que ha de transcurrir antes de que se realice una actualización y la distancia que debe consentir antes de considerar un cambio de posición. Pero se pueden definir diferentes criterios de actualización de la posición. Nuevamente, os recomendamos que consultéis http://developer.android.com/reference/android/location/LocationManager.html para más información.

Finalmente, si Eclipse o vuestro IDE no lo ha hecho automáticamente, no olvidéis hacer los imports correspondientes:

```
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
```

2.2 Brújula

Ahora vamos con la brújula. La idea es exactamente igual, pero aquí hemos de jugar con ángulos y posiciones. No está todo tan "mascado" como con la Localización. En nuestro código usamos un gestor de sensores, osea, un objeto de la clase SensorManager.

```
    mSensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);
```

Para obtener la información de la brújula debemos trabajar con el sensor de orientación. Y le debemos decir que lo añada al listado eventos a escuchar.

```
mSensorManager.registerListener(this,mSensorManager.getDefaultSensor(Sensor.TYPE_ORIENTATIO
N), SensorManager.SENSOR_DELAY_GAME)
```

Atentos al matiz del *this*, necesita, al igual que en el caso de Localization, una clase que sirva de escuchante. Clase que, esta vez, va a ser la propia Activity. Simplemente tenemos que definir los métodos apropiados, que son *onResume*(), *onPause*(), *onSensorChanged*() y *onAccuracyChanged*() estos métodos son autodescriptivos. Los dos primeros especifican qué hacer en caso de que la brújula se apague o se encienda, el tercero nos informa de los cambios de información de la brújula y el último, de los cambios en la precisión.

En el caso de **onSensorChanged**(), trabajamos con un objeto de la clase SensorEvent que es un array, su primer elemento es el ángulo

Finalmente, si tienes algún problema de dependencias, los import son:

```
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
```

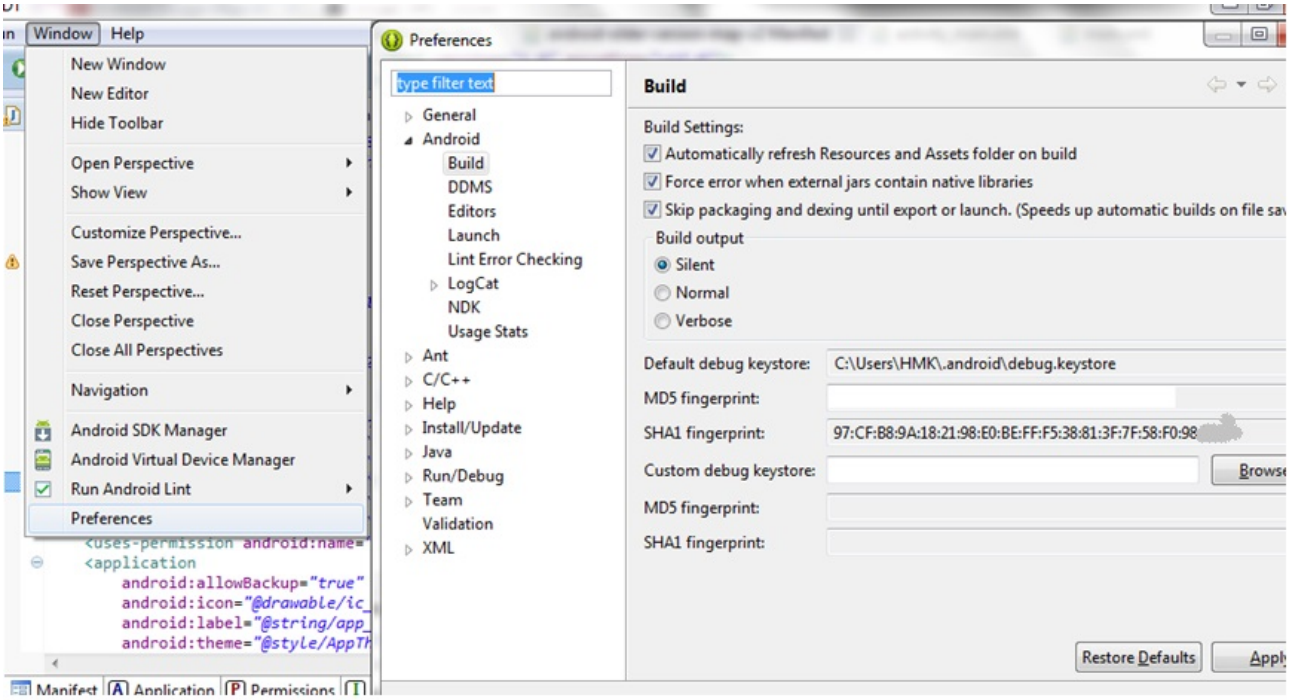
2.3 Mapa de Google

Vamos ahora con el mapa, en concreto los mapas de Google. En primer lugar necesitas obtener una clave para el acceso a la API de los servidores de Google, si ya la tienes, puedes saltarte esta sección y empezar leyendo directamente la siguiente.

2.3.1 OBTENCIÓN DE LA CLAVE DE LA API DE GOOGLE [1]

Para el uso de Google Maps es necesario que Google nos autorice a usar sus servidores obteniendo una clave para su API, esto es un proceso automatizado y gratuíto y una vez que se obtiene puedes usarla en todas tus aplicaciones que requieran el uso de mapas. Lo primero que necesitamos es una **firma SHA1** generada por nosotros. Esto lo debemos hacer desde el IDE (Eclipse en nuestro caso) . Seleccionando **Windows → Preferences → Android → Build → SHA1 fingerprin**t.

1 Esta sección es una traducción y un resumen de este tutorial: http://hmkcode.com/getting-android-google-maps-v2-api-key/

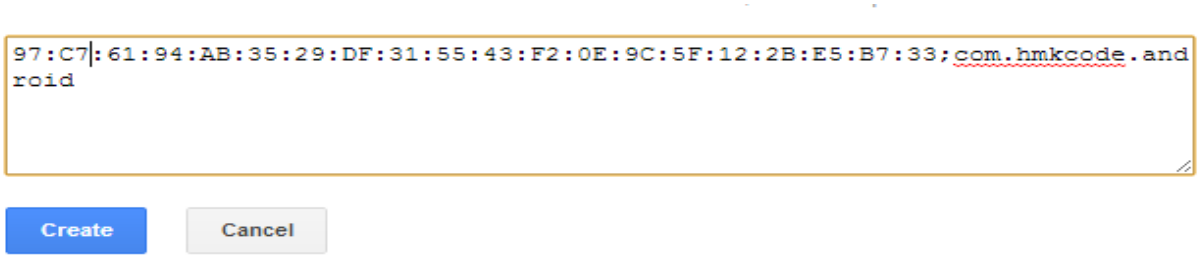


Ahora vamos a solicitar a Google que nos admita el uso de mapas para aplicaciones con esta firma que acabamos de construir (*por cierto, la firma es privada, si alguien la obtiene, puede suplantar vuestra identidad*).

Nos vamos a Google Cloud, https://cloud.google.com/console y creamos un proyecto. Solo necesitamos poner un nombre y definir el id de nuestro proyecto. Una vez que lo hayamos creado, clicamos en él y accedemos a su consola, nos vamos a **API & auth** >> **APIs** >> **Google Maps Android API v2 y activamos al API v2 de android.**

En esa misma sección, debe aparecer un apartado llamado "*credentials*". Que nos permitirá obtener una clave para la API. Nos saldrá un pequeño texto con las condiciones de uso de esa clave, que debemos aceptar haciendo click sobre "Create new key". Hay varios tipos de claves. Podemos obtener una para Android, otra para iOS, para servidores y para aplicaciones de navegador. En nuestro caso obtenemos la clave para Android.

A continuación, nos pide firma que anteriormente habíamos generado.



En cuanto pulsemos el botón "create" podremos ver nuestra clave de acceso a la API. Ya solo falta indicarselo a la aplicación. Esto se hace desde el manifiest, en el apartado <application>. Tenemos que añadir una etiqueta <meta-data> de la siguiente manera.

```
<application>
        ..............
        <meta-data
            android:name="com.google.android.maps.v2.API_KEY"
            android:value="AIzaSyD0bnkRinqfKgRcdHAJTovrm9GtY.......... />
</application>
```

2.3.2 Implementación del mapa

Para el uso del mapa en nuestro código necesitamos tres objetos:

- LocationManager
- LocationListener
- GoogleMap

La clase Activity define una serie de métodos que se ejecutan en base a eventos que le suceden a la actividad. Como en nuestra actividad queremos cargar un mapa, un buen momento de cargarlo es a la par que la actividad es generada, para ello, redefinimos el método onCreate()

```
            super.onCreate(savedInstanceState);
            setContentView(R.layout.activity_map);

            //Obtenemos el fragment del mapa
            MapFragment mapFragment = (MapFragment) getFragmentManager()
                        .findFragmentById(R.id.map);

            //Inicializamos el objeto del fragmento
            googleMap = mapFragment.getMap();

            //Habilitamos MyLocation en el mapa
            googleMap.setMyLocationEnabled(true);

            comenzarLocalizacion();
```

Lo que estamos haciendo aquí es indicar que se va a mostrar como contenido un mapa, tenemos un objeto **MapFragment** que nos sirve para obtener diferentes fragmentos del mapa. Esto se hace con el método findFragmentById() de la clase FragmentManager(). Esta clase nos permite interactuar con **Fragments**, que son elementos que forman parte de un Activity y que admiten interacción con el usuario. En este caso, nuestro mapa es un objeto tipo MapFragment (*hereda de Fragment*) y la interacción con el usuario se realiza por medio del la clase FragmentManager, que obtenemos con el método global getFragmentManager() .

El objeto **googleMap** recoge el MapFragment con el que vamos a trabajar, y tiene diversos métodos para establecer la latitud, longitud, zoom... Puedes ver más aquí http://developer.android.com/reference/com/google/android/gms/maps/MapFragment.html

googleMap.setMyLocationEnabled(true) indica que queremos geolocalizar al usuario. De lo contrario, se mostraría un mapa del mundo genérico o bien la última localización que haya tenido el usuario (depende de las preferencias del usuario y si ha usado previamente google maps en el mismo dispositivo).

Anteriormente, para el apartado de geolocalización, se habló del LocationManager y del LocationListener. El primero era el objeto a usar para obtener nuestra ubicación, el segundo permitía "*vigilar*" los cambios de la misma para recibirlos y procesarlos.

Pues, solamente queda aplicar nuevamente esa idea. Obteniendo el sistema el LocationManager y definiendo los eventos del LocationListener para que obtenga continuamente nuestra nueva ubicación.

```
// Obtenemos una referencia al LocationManager
locManager = (LocationManager) getSystemService(Context.LOCATION_SERVICE);
// Obtenemos la última posición conocida
Location loc = locManager
             .getLastKnownLocation(LocationManager.GPS_PROVIDER);
// Mostramos la última posición conocida
mostrarPosicion(loc);
// Nos registramos para recibir actualizaciones de la posición
locListener = new LocationListener() {
       public void onLocationChanged(Location location) {
              mostrarPosicion(location);
       }

       @Override
       public void onStatusChanged(String provider, int status,
                     Bundle extras) {
              // TODO Auto-generated method stub

       }

       @Override
       public void onProviderEnabled(String provider) {
              // TODO Auto-generated method stub

       }

       @Override
       public void onProviderDisabled(String provider) {
              // TODO Auto-generated method stub

       }
};
locManager.requestLocationUpdates(LocationManager.GPS_PROVIDER, 30000,
```

```
                        0, locListener);
    }
```

Para mostrar la posición, hemos definido el método mostrarPosición(), básicamente volvemos a hacer lo que hacíamos en la parte de geolocalización.

```
private void mostrarPosicion(Location loc) {
        if (loc != null) {
                //Texto de localizacion
                TextView tvLocation = (TextView) findViewById(R.id.tv_location);

                //Obtenemos la latitud
                double latitude = loc.getLatitude();

                //Obtenemos la longitud
                double longitude = loc.getLongitude();

                //Creamos un objeto latLng para usarlo en el mapa
                LatLng latLng = new LatLng(latitude, longitude);

                //Mostramos la localizacion en el mapa
                googleMap.moveCamera(CameraUpdateFactory.newLatLng(latLng));

                //Hacemos zoom en el mapa
                googleMap.animateCamera(CameraUpdateFactory.zoomTo(15));

                tvLocation.setText("Latitude:" + latitude + ", Longitude:"
                                + longitude);
        }
}
```

La novededad está en trasladar esa ubicación al mapa. En primer lugar, nos creamos un objeto de la clase LatLng que gestione la latitud y la longitud. Y este objeto se le pasa a googleMap de la siguiente forma: googleMap.moveCamera(CameraUpdateFactory.newLatLng(latLng));

Queda definir el zoom que se aplica sobre la zona, googleMap.animateCamera(CameraUpdateFactory.zoomTo(15));

2.4 PROBLEMAS

- Nuestro primer problema al trabajar con estos sensores se deriva de la multitud de cambios que han ido sufriendo a lo largo del tiempo. Entonces, realmente según la versión de Android hay diferentes formas de hacerlo, a veces cambian bastante. Si nos pusiéramos a hacer un tutorial para cada versión no acabaríamos de escribir nunca. Así que hemos puesto la forma de hacerlo en Android a partir de la versión 4.1 y hasta que vuelva a cambiar. Para versiones anteriores, os recomendamos que consultéis en internet puesto que no es nada difícil encontrar tutoriales específicos. Por esta misma regla, tened cuidado al trabajar con estos sensores con la información que encontréis en internet, es altamente probable que se encuentre desfasada y hay que filtrar por versiones.
- Los móviles “chinos” experimentan problemas con estos sensores. Especialmente los de hace varios años. Muchos de ellos carecen de sensor GPS. Otros algo más modernos tienen problemas con la brújula. En principio los nuevos modelos ya funcionan correctamente en su mayoría, pero tenedlo en cuenta cuando desarrolléis ya que siempre es bueno vigilar si los sensores están funcionando. Que el usuario sepa que su móvil no funciona le genera frustración, pero que lo descubra tras esperar un largo rato mientras vuestra aplicación hace como que funciona, puede provocar también un enfado.
- RegisterListener() tiene varias definiciones, la que hemos puesto en este tutorial es la única que debéis usar. El resto están totalmente desfasadas (API 3)
- Recordad los import que debéis utilizar , para la brújula

```
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
```

y para la Localización:

```
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
```

- El permiso a añadir para el uso de geolocalización y brújula es únicamente este `<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />` (¡ojo!, en versiones antiguas se requieren permisos diferentes).

2.5 LECTURAS RECOMENDADAS Y REFERENCIAS

No sabemos el nivel de dificultad de los otros sensores, pero estos son relativamente sencillos, asi que casi toda la información la hemos sacado de la documentación oficial de la API de Android. Recapitulamos algunos enlaces que ya os hemos pasado:

- El tutorial de Android sobre estrategias para trabajar con geolocalización: http://developer.android.com/guide/topics/location/strategies.html
- Información sobre el objeto Location, es interesante la idea de que las Localizaciones no solo se obtienen, sino que vosotros las podéis construir para trabajar con localizaciones (por ejemplo, imaginad que quereís controlar determinados puntos en un mapa para comprobar si un corredor ha pasado por esos puntos). http://developer.android.com/reference/android/location/Location.html
- La API del LocationManager http://developer.android.com/reference/android/location/LocationManager.html
- El curso de Android la UPV es muy completo, y paso a paso toca muchos temas, es muy recomendable seguir este curso si vais a desarrollar herramientas de geolocalización: http://www.androidcurso.com/index.php/tutoriales-android/41-unidad-7-seguridad-y-posicionamiento/284-localizacion
- El blog de Salvador Gómez Oliver ha sido el blog desde el que hemos aprendido a realizar el código que se explica en el tutorial. http://www.sgoliver.net/blog/?p=1887
- Aquí podéis ver una explicación más matemática y teórica del funcionamiento de la brújula de android. https://sites.google.com/site/appinventormegusta/ejemplos/brujula---sensor-de-orientacion-y-sprite
- Para obtener la clave de la API de google, hemos seguido este tutorial donde viene explicado el proceso con todo detalle http://hmkcode.com/getting-android-google-maps-v2-api-key/
- Y para terminar, recordad que la geolocalizción es un tema candente en la sociedad, y hay mucho debate sobre ello. En determinados países podéis tener problemas legales con sistemas de geolocalización. Y en España, si la información de geolocalización puede permitir identificar a una persona, tenéis que cumplir la Ley de Protección de datos Personales. Finalmente, un artículo al respecto de este tema http://bordercriminologies.law.ox.ac.uk/no-place-to-hide/