Geolocalización [Java]

Los dispositivos móviles son capaces de establecer su posición geográfica (latitud y longitud) por diferentes medios. Muchos cuentan con un GPS capaz de darnos la posición con un error de metros. El problema del GPS es que solo funciona a cielo abierto y usa la triangulación de los satélites disponibles. Si estamos a cubierto o no disponemos de PS, es posible establecer nuestra posición mediante las antenas de telefonía móvil y los puntos de acceso Wifi, con un margen de error mucho mayor.

Para usar los servicios de geolocalización es preciso previamente solicitar los permisos de acceso correspondientes en el **AndroidManifest**. Necesitaremos ACCESS_FINE_LOCATION para la localización precisa o GPS, y la ACCESS_COARSE_LOCATION para la localización aproximada o mediante las antenas de red móvil o Wifi.

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

El servicio de localización se considera 'peligroso' por lo que hay que gestionarlo con cuidado. La clase que se encarga de gestionar este servicio es **LocationManager** (tanto en Java como en Kotlin). No se instancia, sino que se trata como un servicio del sistema que tenemos que solicitar.

```
// The Location Manager
LocationManager locationManager = (LocationManager) getSystemService(LOCATION_SERVICE);
```

Los proveedores

La geolocalización nos la ofrece un **proveedor**. Es posible obtener la información de los proveedores a través del **LocationManager**. En la imagen, el primero es el proveedor de GPS y el segundo es el de red móvil o wifi.

```
LocationProvider gpsProvider = locationManager.getProvider(LocationManager.GPS_PROVIDER);
LocationProvider networkProvider = locationManager.getProvider(LocationManager.NETWORK_PROVIDER);
```

Cada instancia de **LocationProvider** dispondrá de toda la información necesaria de cada proveedor. Uno de los datos más interesantes es la precisión (accuracy), que podrá ser ACCURACY_FINE o ACCURACY_COARSE.

En ocasiones y según el dispositivo, es posible que no dispongamos de algún proveedor o que existan otros disponibles. En ese caso podemos hacer uso de **getAllProviders ()** para obtener el listado de todos los proveedores disponibles. El método **isProviderEnabled ()** nos servirá para averiguar si un proveedor está actualmente disponible o habilitado, aunque el dispositivo cuente con él.

```
private List<Provider> getAllProviders(LocationManager locationManager) {
    List<Provider> ret = null;

    List<String> providers = locationManager.getAllProviders();
    if ((null != providers) && (providers.size() > 0)) ret = new ArrayList<>();

    if (providers != null) {
        for (String providerName : providers) {
            Provider provider = new Provider();
            provider.setName(providerName);
            provider.setProviderEnabled(locationManager.isProviderEnabled(providerName));
            provider.setAccuracy(Provider.ACCURACY[Math.max(0, (locationManager.getProvider(pro
```

Consultar la localización

Para averiguar la posición disponemos de varias estrategias. Por ejemplo, es posible recuperar la última posición conocida del dispositivo registrada por un proveedor. Sin embargo, el problema es que no sabemos de cuándo es dicha posición.

```
Location myLastLocation = locationManager.getLastKnownLocation(LocationManager.GPS_PROVIDER);
```

Para recibir una posición actualizada lo primero es definir el *listener* que va a escuchar a los eventos del proveedor. Para eso podemos implementar la interfaz **LocationListener** o crear una clase separada. La actualización de la localización se puede hacer mediante el método **requestLocationUpdates ()**

```
locationManager.requestLocationUpdates(chosenProvider, TIME_MIN, DISTANCE_MIN, listener: this);
```

Los métodos del **LocationListener** son:

```
@Override
public void onProviderDisabled(String provider) {
}

@Override
public void onProviderEnabled(String provider) {
}

@Override
public void onStatusChanged(String provider, int status, Bundle extras) {
}

@Override
public void onLocationChanged(@NonNull Location location) {
}
```

Fuse location provider

Finalmente, también podemos recurrir a la clase **FusedLocationProviderClient** para automatizar ciertas tareas y gestionar mejor el consumo de la batería del dispositivo. Esta opción no se encuentra siempre disponible ya que depende enteramente de los servicios de Google.

Práctica 50

[Java] Instala la app de ejemplo **UD9 – Geoloc**. Esta app es un ejemplo sencillo de cómo obtener la posición y los proveedores. Modifícala para que muestre un listado de los proveedores disponibles y después lance una petición únicamente al proveedor escogido. Muestra la posición al usuario.

Práctica 51

[Kotlin] Lo mismo que antes, pero en Kotlin.