UNITÉ D ’ENSEIGNEMENT (UE) : DÉVELOPPEMENT MOBILE

Ch8 : Connectivité

Téléphonie

- **Les fonctions de téléphonie sont relativement simples à utiliser. Elles permettent de récupérer l'état de la fonction de téléphonie (appel en cours, appel entrant, ...), d'être notifié lors d'un changement d'état, de passer des appels et de gérer l'envoi et réception de SMS.**
- **L'état de la téléphonie est géré par la classe TelephonyManager qui permet de récupérer le nom de l'opérateur, du téléphone, et l'état du téléphone. Pour lire ces informations, il est nécessaire de disposer de la permission android.permission.CALL_PHONE.**

```
TelephonyManager tel =
(TelephonyManager)getSystemService(Context.TELEPHONY_SERVICE);
int etat = tel.getCallState();
if (etat == TelephonyManager.CALL_STATE_IDLE)   // RAS
    if (etat == TelephonyManager.CALL_STATE_RINGING)    // Le
téléphone sonne
        String SIMnb = tel.getSimSerialNumber();
```

Téléphonie

- **Il est aussi possible d'être notifié d'un changement d'état en utilisant un écouteur:**

```
public class Ecouteur extends PhoneStateListener {
    public void onCallStateChanged(int etat, String
numero) {
        super.onCallStateChanaged(etat, numero)
        if (etat ==
TelephonyManager.CALL_STATE_OFFHOOK)
        // Le téléphone est en cours d'appel
    }
```

Passer un appel

- Il est possible de passer un appel ou de déléguer l'appel, ces deux actions étant réalisées avec un Intent (attention aux permissions):

```
Uri telnumber = Uri.parse("tel:0021698000000");
Intent call = new Intent(Intent.ACTION_CALL, telnumber);
startActivity(call);
```

```
Uri telnumber = Uri.parse("tel:0021698000000");
Intent call = new Intent(Intent.ACTION_DIAL, telnumber);
startActivity(call);
```

- Les applications qui peuvent passer des appels doivent filtrer ce type d'Intent pour pouvoir être invoquée lorsque l'Intent est lancé:

```
<receiver android:name=".ClasseGerantLAppel">
    <intent-filter>
        <action android:name="Intent.ACTION_CALL"/>
    </intent-filter>
</receiver>
```

Envoyer et recevoir des SMS

- **Si la permission android.permission.SEND_SMS est disponible, il est possible d'envoyer des SMS au travers de SmsManager:**

```
SmsManager manager = SmsManager.getDeault();
manager.sendTextMessage("0021698000000",null,"BONJOUR",null,null)
;
```

- **Inversement, il est possible de créer un filtre d'Intent pour recevoir un SMS qui sera géré par un broadcast receiver. L'action à préciser dans le filtre d'Intent du receveur est android.provider.Telephony.SMS_RECEIVED:**

```
<receiver android:name=".SMSBroadcastReceiver">
  <intent-filter>
    <action
android:name="android.provider.Telephony.SMS_RECEIVED"/>
  </intent-filter>
</receiver>
```

Bluetooth

- **Le bluetooth se gère au travers de principalement 3 classes:**
  - **BluetoothAdapter**: similaire au WifiManager, cette classe permet de gérer les autres appareils bluetooth et d'initier les communications avec ceux-ci.
  - **BluetoothDevice**: objet représentant l'appareil distant.
  - **BluetoothSocket** et **BluetoothServerSocket**: gère une connexion établie.
- **Pour pouvoir utiliser les fonctionnalités bluetooth, il faut activer les permissions android.permission.BLUETOOTH et android.permission.BLUETOOTH_ADMIN pour pouvoir chercher des appareils ou changer la configuration bluetooth du téléphone.**

```
BluetoothAdapter bluetooth = BluetoothAdapter.getDefaultAdapter();
if (!bluetooth.isEnabled())
{
  Intent launchBluetooth = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
  startActivity(launchBluetooth);
}
```

Localisation

- **Comme pour le réseau, Android permet d'utiliser plusieurs moyens de localisation. Cela permet de rendre transparent l'utilisation du GPS, des antennes GSM ou des accès au Wifi. La classe LocationManger permet de gérer ces différents fournisseurs de position.**
  - LocationManager.GPS_PROVIDER: fournisseur GPS
  - LocationManager.NETWORK_PROVIDER: fournisseur basé réseau
- **La liste de tous les fournisseurs s'obtient au travers de la méthode getAllProviders() ou getAllProviders(true) pour les fournisseurs activés:**
- **Les permissions associées pour la localisation sont:**
  - android.permission.ACCESS_FINE_LOCATION via le GPS
  - android.permission.ACCESS_COARSE_LOCATION via le réseau

Coordonnées

- **A partir du nom d'un fournisseur de position actif, il est possible d'interroger la dernière localisation en utilisant l'objet Location.**

```
Location localisation =
manager.getLastKnownLocation("gps");

Toast.makeText(getApplicationContext(),
"Latitude" + localisation.getLatitude(),
Toast.LENGTH_SHORT).show();

Toast.makeText(getApplicationContext(),
"Longitude" + localisation.getLongitude(),
Toast.LENGTH_SHORT).show();
```

Coordonnées

- **Il est possible de réagir à un changement de position en créant un écouteur qui sera appelé à intervalles réguliers et pour une distante minimum donnée:**

```
manager.requestLocationUpdates("gps", 6000, 100, new LocationListener() {
  public void onStatusChanged(String provider, int status, Bundle extras)
{//Appeler quand le status d'une source change. Il existe 3 statuts
(OUT_OF_SERVICE, TEMPORARILY_UNAVAILABLE, AVAILABLE)
  }
public void onProviderEnabled(String provider) {//appelée quand une
source de localisation est activée
}
public void onProviderDisabled(String provider) {//appelée quand une
source de localisation est désactivée
}
public void onLocationChanged(Location location) {//appelée quand la
localisation de l'utilisateur est mise à jour.
}
```

Alerte de proximité

- **Il est possible de préparer un événement en vu de réagir à la proximité du téléphone à une zone. Pour cela, il faut utiliser la méthode addProximityAlert de LocationManager qui permet d'enregistrer un Intent qui sera envoyé lorsque des conditions de localisation sont réunies. Cette alerte possède une durée d'expiration qui la désactive automatiquement. La signature de cette méthode est:**

```
addProximityAlert(double latitude, double longitude, float radius, long expiration, PendingIntent intent)
```

- **Il faut ensuite filtrer l'intent préparé:**

```
IntentFilter filtre = new IntentFilter(PROXIMITY_ALERT);
registerReceiver(new MyProximityAlertReceiver(), filtre);
```

- **La classe gérant l'alerte est alors:**

```
public class MyProximityAlertReceiver extends BroadcastReiceiver {
  public void onReceive(Context context, Intent intent) {
    String key = LocationManager.KEY_PROXIMITY_ENTERING;
    Boolean entering = intent.getBooleanExtra(key, false);
  }}
```

Capteurs

- **La majorité des Smartphones sont dotés de capteurs : accéléromètre, capteur de lumière, capteur d'orientation, etc.**
- **Sont utiles pour construire différents types d'applications sensibles aux contextes (sports, tourisme, etc.), les jeux, etc.**
- **Peu d'appareils possèdent tous les capteurs gérés par la plateforme Android.**
  - La majorité des Smartphones et tablettes ont un accéléromètre et un magnétomètre mais peu possèdent un baromètre où un thermomètre.
- **Un appareil peut avoir plusieurs capteurs du même type.**

Types de capteurs

- **Capteurs de mouvements**
  - Mesurent les forces d'accélération et de rotation autour de trois axes.
  - Inclut les capteurs de : accéléromètre, gravité, gyroscope et rotation.
- **Capteurs d'environnement**
  - Mesurent différents paramètres environnementaux tels que la température ambiante et la pression, la luminosité et l'humidité.
  - Inclut les capteurs de : pression (baromètre), photomètre et thermomètre.
- **Capteurs de position**
  - Mesurent la position physique de l'appareil.
  - Inclut les capteurs de : orientation et champs magnétique (magnétomètre).

Framework (API) d'Android pour gestion des capteurs

- **Android propose un Framework (Une API) pour accéder aux capteurs disponibles sur un appareil et acquérir les données capturées par ces capteurs.**
- **Exemples de services :**
  - Déterminer les capteurs disponibles sur un appareil.
  - Déterminer les propriétés d'un capteur donné telles que la portée, le constructeur, les besoins en énergie et la résolution.
  - Acquérir les données des capteurs.
  - Enregistrer et dé-enregistrer un écouteur d'événement pour gérer les changements signalés par un capteur.

Sensor	Type	Description	Common Uses
TYPE_ACCELEROMETER	Hardware	Measures the acceleration force in m/s^2 that is applied to a device on all three physical axes (x, y, and z), including the force of gravity.	Motion detection (shake, tilt, etc.).
TYPE_AMBIENT_TEMPERATURE	Hardware	Measures the ambient room temperature in degrees Celsius (°C). See note below.	Monitoring air temperatures.
TYPE_GRAVITY	Software or Hardware	Measures the force of gravity in m/s^2 that is applied to a device on all three physical axes (x, y, z).	Motion detection (shake, tilt, etc.).
TYPE_GYROSCOPE	Hardware	Measures a device's rate of rotation in rad/s around each of the three physical axes (x, y, and z).	Rotation detection (spin, turn, etc.).
TYPE_LIGHT	Hardware	Measures the ambient light level (illumination) in lx.	Controlling screen brightness.
TYPE_LINEAR_ACCELERATION	Software or Hardware	Measures the acceleration force in m/s^2 that is applied to a device on all three physical axes (x, y, and z), excluding the force of gravity.	Monitoring acceleration along a single axis.
TYPE_MAGNETIC_FIELD	Hardware	Measures the ambient geomagnetic field for all three physical axes (x, y, z) in μT.	Creating a compass.

TYPE_ORIENTATION	Software	Measures degrees of rotation that a device makes around all three physical axes (x, y, z). As of API level 3 you can obtain the inclination matrix and rotation matrix for a device by using the gravity sensor and the geomagnetic field sensor in conjunction with the getRotationMatrix() method.	Determining device position.
TYPE_PRESSURE	Hardware	Measures the ambient air pressure in hPa or mbar.	Monitoring air pressure changes.
TYPE_PROXIMITY	Hardware	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	Phone position during a call.
TYPE_RELATIVE_HUMIDITY	Hardware	Measures the relative ambient humidity in percent (%).	Monitoring dewpoint, absolute, and relative humidity.
TYPE_ROTATION_VECTOR	Software or Hardware	Measures the orientation of a device by providing the three elements of the device's rotation vector.	Motion detection and rotation detection.
TYPE_TEMPERATURE	Hardware	Measures the temperature of the device in degrees Celsius (°C). This sensor implementation varies across devices and this sensor was replaced with the TYPE_AMBIENT_TEMPERATURE sensor in API Level 14	Monitoring temperatures.

Les principales classes de l'API

- **SensorManager :**
  - Utilisée pour créer une instance du service commun de gestion des capteurs.
  - Fournit :
    - Des méthodes pour accéder aux capteurs ou les lister, enregistrer et de-enregistrer les écouteurs etc..
    - Des constantes permettant d'accéder aux valeurs telles que la précision du capteur ou le taux d'acquisition des données, etc.
- **Sensor :**
  - Utilisée pour créer une instance d'un capteur donné.
  - Fournit des méthodes pour déterminer les propriétés d'un capteur.

Les principales classes de l'API

- **SensorEvent:**
  - Utilisée pour créer un objet événement de capteur, qui fournit des informations sur l'événement du capteur.
    - Cet objet inclut les informations suivantes : les données capturées par le capteur, le type du capteur qui a générer l'événement, la précision des données et l'horodatage de l'événement.
    - Un événement capteur apparaît chaque fois que le capteur détecte un changement du paramètre qu'il mesure.
- **SensorEventListener :**
  - Est une interface utilisée pour créer deux méthodes de callback
  - Ces méthodes reçoivent des notifications quand les valeurs ou la précision de ce capteur changent.

Utilisation des capteurs: Les étapes

1) **Récupérer une instance du service de gestion des capteurs (SensorManager)**
2) **Vérifier si le capteur qui nous intéresse est disponible sur l'appareil → utilisation des méthodes de SensorManager.**
3) **Si le capteur en question existe, récupérer un objet qui représente le capteur qui nous intéresse → instance de la classe Sensor.**
4) **S'abonner aux événements du capteur qui nous intéresse → la classe utilisateur doit implémenter l'interface SensorEventListener.**
5) **Dans les méthodes de callBack de l'interface SensorEventListener, utiliser les informations de l'objet, instance de la classe SensorEvent qui encapsule les données reçues du capteur et faire une action.**

Utilisation des capteurs : SensorManager

- **Récupérer une instance du service de gestion des capteurs (SensorManager)**

```
private SensorManager mSensorManager;

...

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

- **Lister les capteurs présents sur un appareil**

```
final SensorManager sensorManager =
(SensorManager)getSystemService(Context.SENSOR_SERVICE
);

List<Sensor> sensorsList =
sensorManager.getSensorList(Sensor.TYPE_ALL);
```

Utilisation des capteurs: SensorManager

- **Déterminer la présence d'un capteur d'un type donné sur un appareil**
  - Utilisation de la méthode getDefaultSensor()
  - Avec un paramètre de type constante du capteur en question.
    - TYPE_GYROSCOPE, TYPE_LINEAR_ACCELERATION, TYPE_GRAVITY, ...
- **Si l'appareil possède plus d'un capteur du même type, il doit être désigné comme capteur par défaut.**
- **Exemple du besoin de vérifier la présence d'un capteur**
  - Une application de guidage (de navigation) peut utiliser les capteurs de température, de pression, capteur GPS, capteur de champs magnétique,
  - Si un appareil n'a pas de capteur de pression, l'application doit détecter dynamiquement l'absence de ce capteur et désactiver la partie de l'application qui utilise ce capteur

Utilisation des capteurs: classe Sensor

- **Déterminer les propriétés d'un capteur : utilisation des méthodes de la classe Sensor**
  - Permet d'adapter dynamiquement les services d'une application en fonction de la disponibilité ou non de certains capteurs sur l'appareil
  - Exemples
    - getResolution() et getMaximumRange() : retournent respectivement les mesures de la résolution et de la portée maximale.
    - getPower() retourne la mesure du besoin énergétique (batterie) d'un capteur.
    - getVendor() et getVersion().
- **Exemple :**
  - Chercher si un capteur de gravité est disponible et dont le fabricant est Google Inc. et la version est 3. Si ce capteur n'est pas présent, utilisation de l'accéléromètre

Utilisation des capteurs: classe Sensor

- **Déterminer la sensibilité d'un capteur**
  - La sensibilité d'un capteur est l'intervalle de temps minimum (en microsecondes) qu'un capteur utilise pour acquérir ses données.
  - **getMinDelay**() retourne la sensibilité d'un capteur :
    - Un capteur qui retourne une valeur différente de zéro est un capteur de flux continu (streaming sensor).
    - Un capteur qui retourne la valeur 0, n'est pas un capteur de flux continu mais ce capteur qui avertit ses écouteurs en cas de changement.
- **Remarque : la valeur de sensibilité retournée par un capteur n'est pas nécessairement celle de livraison des données vers une application.**
  - L'API livre les données à une application via des événements dont le ration dépend d'autres facteurs.

Agir par rapport aux événements d’un capteur

- **Pour obtenir les données à partir d’un capteur, il est nécessaire de :**

1) S’enregistrer auprès du service comme écouteur d’événements

2) Implémenter deux méthodes de callback définies dans l’interface

S’enregistrer auprès du service comme écouteur d’événements

- **Utilisation de la méthode `registerListener()`.**
- **« data delay ou rate » est un des paramètres de registerListener() = l’intervalle de temps, souhaité par l’utilisateur, entre chaque envoi de données (événements) par le capteur à l’application via la méthode onSensorChanged().**
  - SENSOR_DELAY_FASTEST : le taux de rafraîchissement le plus élevé. Les données sont récupérées aussi vite que possible par le système ;
  - SENSOR_DELAY_GAME : taux utilisable pour des applications nécessitant des données précises telles que les jeux ; 20,000 microsecondes
  - SENSOR_DELAY_NORMAL : taux normal (par défaut) utilisé par exemple par le système pour détecter les changements d’orientation ; 200,000 microsecondes
  - SENSOR_DELAY_UI : le taux le plus bas utilisé dans les applications ne nécessitant qu’une faible précision. 60,000 microsecondes
- **Plus le taux est élevé, plus les données sont rafraîchies et précises, et plus la consommation d’énergie est élevée.**
- **Après utilisation de ces données ou avant de quitter l’application, il est indispensable de se dé-s’enregistrer**
  - Utilisation de la méthode unregisterListener.

Implémenter deux méthodes de callback définies dans l'interface

- **onAccuracyChanged() : Appelée quand la précision du capteur change**
  - Le système fournit à cette méthode une référence vers l'objet Sensor concerné et la nouvelle valeur de la précision
  - La précision a une des 4 valeurs suivantes :
    - SENSOR_STATUS_ACCURACY_LOW,
    - SENSOR_STATUS_ACCURACY_MEDIUM,
    - SENSOR_STATUS_ACCURACY_HIGH,
    - SENSOR_STATUS_UNRELIABLE.
- **onSensorChanged() : Appelée quand le capteur retourne une nouvelle donnée.**
  - Le système fournit à cette méthode l'objet SensorEvent qui contient les informations suivantes :
    - La donnée générée;
    - Le capteur qui a généré cette donnée;
    - L'horodatage de la donnée générée;
    - La précision de cette donnée.

Agir par rapport aux événements d'un capteur

Nom	Dimension du vecteur	Unité	Sémantique	Valeurs[]
Accelerometer	3	m/s2	Mesure de l'accélération (gravité incluse)	[0] axe x [1] axe y [2] axe z
Gyroscope	3	Radian/seconde	Mesure la rotation en termes de vitesse autour de chaque axe	[0] vitesse angulaire autour de x [1] vitesse angulaire autour de y [2] vitesse angulaire autour de z
Light	1	Lux	Mesure de la luminosité	[0]valeur
Magnetic_Field	3	µTesla	Mesure du champ magnétique	[0] axe x [1] axe y [2] axe z
Orientation	3	degrés	Mesure l'angle entre le nord magnétique	[0] Azimut entre l'axe y et le nord [1] Rotation autour de l'axe x (-180,180) [2] Rotation autour de l'axe y (-90,90)
Pressure	1	KPascal	Mesure la pression	[0]valeur
Proximity	1	mètre	Mesure la distance entre l'appareil et un objet cible	[0]valeur
Temperature	1	Celsius	Mesure la température	[0]valeur

Exemple : Réception des données

- **Utilisation de l'accéléromètre**

```
private void onAccelerometerChanged(SensorEvent event){
float x,y,z;
x = event.values[0];
y = event.values[1];
z = event.values[2];
((TextView)findViewById(R.id.axex)).setText("Axe X:
"+x+"m/s^2");
((TextView)findViewById(R.id.axey)).setText("Axe Y:
"+y+"m/s^2");
((TextView)findViewById(R.id.axez)).setText("Axe Z:
"+z+"m/s^2");
}
```

Exemple : Réception des données

- **Utilisation du capteur d'orientation**

```
private void onOrientationChanged(SensorEvent event){
float azimuth,pitch,roll;
azimuth = event.values[0];
pitch = event.values[1];
roll = event.values[2];
((TextView)findViewById(R.id.azimuth)).setText("Azimuth:
"+azimuth+"°");
((TextView)findViewById(R.id.pitch)).setText("Pitch:
"+pitch+"°");
((TextView)findViewById(R.id.roll)).setText("Roll: "+roll+"°");
}
```

Droits

- **Pour déclarer l'utilisateur d'un capteur et avoir le droit d'accéder aux valeurs lues par le hardware, il ne faut pas oublier d'ajouter dans le Manifest la déclaration adéquat à l'aide du tag uses-feature. Cela permet à Google Play de filtrer les applications compatibles avec l'appareil de l'utilisateur. Par exemple, pour l'accéléromètre:**

```
<uses-feature
android:name="android.hardware.sensor.accelerometer"></us
es-feature>
```

- **La directive n'est cependant pas utilisée lors de l'installation, ce qui signifie qu'il est possible d'installer une application utilisant le bluetooth sans posséder de hardware bluetooth. Évidemment, il risque d'y avoir une exception ou des dysfonctionnements. Un booléen supplémentaire permet alors d'indiquer si ce hardware est indispensable au fonctionnement de l'application:**

```
<uses-feature android:name="xxx"
android:required="true"></uses-feature>
```

Caméra

- **Il est souvent nécessaire de construire une zone de preview de ce que la caméra voit, avant par exemple de prendre une photo. Avant toute chose, il faut prévoir dans le manifest la permission nécessaire android.permission.CAMERA. Ensuite les grandes étapes sont les suivantes:**
  - Dans l'activité principale on ajoute une ViewGroup spéciale et on ouvre la caméra (arrière, d'id 0, frontale, d'id 1):

```
Camera mCamera;

Preview mPreview = new Preview(this);

setContentView(mPreview);

Camera.open(id);

mPreview.setCamera(mPreview);
```

  - Avec une classe Preview héritant de ViewGroup et implémentant SurfaceHolder.Callback. Cela oblige à recoder les méthodes surfaceChanged, surfaceCreated, surfaceDestroyed et onLayout qui seront appelées quand la surface de rendu est effectivement créée. Notamment, il faut accrocher un objet SurfaceHolder à l'objet Camera, puis dire à l'objet SurfaceHolder que le callback est l'objet Preview.

https://developer.android.com/guide/topics/media/camera.html

L'API Google Maps

- **Un des grands avantages d'Android est de pouvoir bénéficier des principales applications déjà développées par Google. L'une d'elle sont les Google maps**
- **Toute une API permet d'utiliser ces cartes Google**
- **On utilise les Google Maps Android API v2. La page d'accueil de cette technologie est à l'URL https://developers.google.com/maps/documentation/android-api/**
- **Cette API permet de manipuler des cartes terrestres. Ces classes se trouvent dans le package com.google.android.gms.maps**
- **Pour afficher une carte, on utilise les fragments**

La bibliothèque Google Maps

- **Cette API gère les entrées clavier, le zoom, le toucher sur une carte affichée On peut ajouter des dessins, des images sur la carte**
- **Pour utiliser cette API, on doit être enregistré auprès du service Google Maps et avoir obtenu une clé Maps API v2**
- **On peut commencer à étudier cette API à partir de l'URL**

https://developers.google.com/maps/documentation/android/intro

https://developers.google.com/maps/documentation/android/start#the_google_maps_api_key

Pour utiliser les cartes de Google

- **Il faut avoir un compte Google : un gmail suffit**
- **Si on veut faire afficher le résultat dans un AVD, c'est possible ! Mais cet AVD doit avoir les APIs Google**
- **Pour utiliser les cartes Google avec Android Studio, il suffit de créer un projet dans le cloud Google qui gère les Google maps et d'obtenir une Maps API v2 Key**
- **Toute la procédure est indiquée à https://developers.google.com/maps/documentation/android/start**

  https://developers.google.com/maps/documentation/android-api/signup

Obtenir une Maps API v2 Key

- **Une Maps API key est obtenue en envoyant le couple (réduit du certificat de la clé d'encryptage de l'application, nom du paquetage de l'application)**
- **Comme toute application Android est signée, les clés Google API sont liées à ce couple (clé d'encryptage de l'app, paquetage de l'application). Elles ne dépendent pas des utilisateurs (et de leur nombre)**
- **Créer une activité qui va afficher une carte Google par New | Google | Google Maps Activity**
  - Il est alors créé :
    - une activité adaptée
    - un fichier google_maps_api.xml

Obtenir une Maps API v2 Key

```xml
<resources>
    <!--
    TODO: Before you run your application, you need a Google Maps API key.

    To get one, follow this link, follow the directions and press "Create" at the end:

    https://console.developers.google.com/flows/enableapi?apiid=maps_android_backend&keyType=CLIENT_SIDE_ANDROID&r=

    You can also add your credentials to an existing key, using this line:
    30:C7:44:35:B6:12:37:7C:87:D8:37:CF:F6:53:BC:90:E7:FC:C8:FF;com.example.khouja.loca

    Alternatively, follow the directions here:
    https://developers.google.com/maps/documentation/android/start#get-key

    Once you have your key (it starts with "AIza"), replace the "google_maps_key"
    string in this file.
    -->
    <string name="google_maps_key" templateMergeStrategy="preserve" translatable="false">
        YOUR_KEY_HERE
    </string>
</resources>
```

Exécution du programme

- **En lançant l'activité qui gère la google map on obtient l’image de la carte dans l’interface**
  - tout a été bien configuré par Android Studio à savoir
    - la création du projet Google cloud pour les google maps
    - la configuration de l'AndroidManifest.xml
    - le code de l'activité et son fichier d'IHM XML associé
- **Vérifier ces parties dans votre app**

La documentation des classes Google Maps Android :

http://developer.android.com/reference/com/google/android/gms/maps/package-summary.html