

# TP2 : Les capteurs

## Programmation mobile

Mohamad Satea Almallouhi - Tony Nguyen  
*M1 Génie Logiciel*  
Faculté des Sciences  
Université de Montpellier.

5 mars 2024



### Résumé

*Nous avons réalisé une application Android en Java afin de démontrer l'utilisation des capteurs intégré.*

## Table des matières

<b>Introduction</b>	<b>3</b>
<b>Démonstration</b>	<b>3</b>
1   Liste des capteurs ( <i>Index</i> )	3
2   Détection de la disponibilité des capteurs ( <i>SensorAvailability</i> )	4
3   Accéléromètre ( <i>Accelerometer</i> )	4
4   Direction ( <i>Direction</i> )	5
5   Secouer un appareil ( <i>ShakeDevice</i> )	5
6   Proximité ( <i>Proximity</i> )	6
7   Géolocalisation ( <i>Geolocation</i> )	6
8   Extra	6
8.1   Icone . . . . .	6
8.2   Retour . . . . .	7

# Introduction

Dans ce TP, nous allons l'utilisation des capteurs intégré dans nos smartphones.

Nous allons voir comment manipuler les différents types de capteur comme le GPS, la boussole, le gyroscope, etc ...

Pour réaliser ce tp, nous choisissons de créer une unique application avec un écran d'accueil (*ChooseApplication.java*) qui mène aux différentes activités correspondant à chaque exercice.



Les sections du rapport suit les exercices.

## Démonstration

En ligne sur Youtube, à l'adresse URL <https://youtu.be/NjIdJrN-Rm8> une démonstration vidéo de notre travail.

## 1 Liste des capteurs (*Index*)

Lorsque l'on déploie une application sur un smartphone, nous ne pouvons pas être sûr des capteurs que l'on aura à notre disposition. Dans cette section, nous allons regarder tout les capteurs disponible sur un appareil donné.

Tout d'abord, il est nécessaire d'accéder au sensorManager par lequel les informations ainsi que l'accès aux capteurs passent.

```
setContentView(R.layout.activity_list);
//Récupère une instance du service de gestion des capteurs
sensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
listSensor();

ListView listView = findViewById(R.id.sensorListView);
ArrayAdapter<String> adapter = new ArrayAdapter<>(
    context: this,
    android.R.layout.simple_list_item_1,
    displayArray);
listView.setAdapter(adapter);
```

Ensuite, une fois qu'on a le sensorManager, nous pouvons faire appel à la fonction **getSensorList(Sensor.TYPE\_ALL)** qui nous renvoie un tableau de tous les capteurs.

Pour accéder individuellement aux informations de chaque capteur représenté par un objet de type Sensor, nous invoquons les méthodes **getName()**, **getType()** et **getVersion()**.

```
private void listSensor() {
    //Lister les capteurs :
    sensors = sensorManager.getSensorList(Sensor.TYPE_ALL);
    displayArray = new String[sensors.size()];
    int i=0;
    for (Sensor sensor : sensors) {
        displayArray[i]="";
        displayArray[i] += "Name: " + sensor.getName() +
        "\r\n"+ "Type: " +
        getType(sensor.getType()) +
        "\r\n"+ "Version: " +
        sensor.getVersion();
        i++;
    }
}
```

Enfin, pour afficher à l'écran le résultat, nous utilisons une Vue de type ListView que nous avons au préalable déclarer dans le layout.

```
<LinearLayout xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".Index">

    <ListView
        android:id="@+id/sensorListView"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />

</LinearLayout>
```



forme d'un numéro d'index. On récupère le capteur qui correspond à cet index puis regarde si ce capteur est disponible ou pas. C'est la fonction `getDefaultSensor()` qui nous assure qu'un capteur est indisponible quand il renvoie null.

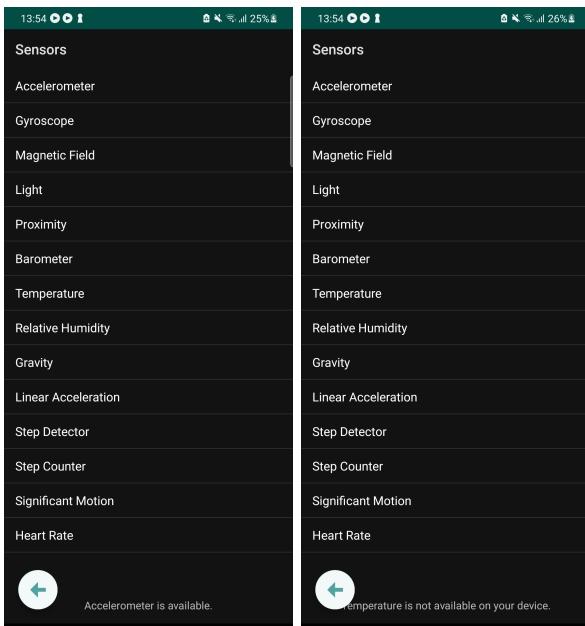
```

sensorListView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
        String selectedSensor = sensorTypes[position];
        checkSensorAvailability(selectedSensor);
    }
});

private void checkSensorAvailability(String sensorName) {
    int sensorType = sensorTypeMap.get(sensorName);
    Sensor sensor = sensorManager.getDefaultSensor(sensorType);
    if (sensor != null) {
        sensorStatusTextView.setText(sensorName + " is available.");
    } else {
        sensorStatusTextView.setText(sensorName + " is not available on your device.");
    }
}

```

## 2 Détection de la disponibilité des capteurs (*Sensor Availability*)



Dans ce nouvel exercice, nous allons afficher la disponibilité (*en bas de l'écran*) de un capteur en particulier.

Pour cela, nous réutilisons un `ListView` où chaque élément correspondra à un type de capteur. Puis nous redéfinissons la méthode `onItemClick()` pour modifier le comportement "cliquer" sur chaque item de la vue. La fonction nous donne en argument quelle item a été cliquer dessus sous

## 3 Accéléromètre (*Accelerometer*)

Nous avons fait une petite application qui change la couleur en fond en fonction de l'accélération du téléphone.

Nous commençons par récupérer le `sensorManager` et le capteur accéléromètre. Afin de récupérer les données du capteur, il nous suffit de s'enregistrer (avec `registerListener()`) auprès du service comme écouteurs d'événements et implémenter les méthodes de callback `onSensorChanged()` et `onAccuracyChanged()`.

Nous choisissons d'utiliser `SensorManager.SENSOR_DELAY_UI` pour économiser de la batterie.

Quand le capteur détecte un changement, la méthode `onSensorChanged()` est appelé. Si une accélération suffisamment grande est détecté, on change la couleur du fond d'écran en rouge ou noir avec `view.setBackgroundColor()`.

```

sensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
sensor_Acc = sensorManager.getDefaultSensor(TYPE_ACCELEROMETER);

sensorManager.registerListener( listener: this, sensor_Acc, SensorManager.SENSOR_DELAY_UI);

```

```

@Override
public void onSensorChanged(SensorEvent event) {
    // Check if the sensor change is from the accelerometer
    if (event.sensor.getType() == TYPE_ACCELEROMETER) { onAccelerometerChanged(event); }
}

Usage : SateaMail
private void onAccelerometerChanged(SensorEvent event) {
    // Get accelerometer values
    float[] values = event.values;
    float x = values[0];
    float y = values[1];
    float z = values[2];

    float magnitude = (float) Math.sqrt(x * x + y * y + z * z) / SensorManager.GRAVITY_EARTH;

    if (magnitude < THRESHOLD_MEDIUM) {
        // Low values, set background to green
        view.setBackgroundColor(Color.GREEN);
    } else if (magnitude < THRESHOLD_HIGH) {
        // Medium values, set background to black
        view.setBackgroundColor(Color.BLACK);
    } else {
        // High values, set background to red
        view.setBackgroundColor(Color.RED);
    }
}

```

## 4 Direction (*Direction*)

De façon très similaire à la section précédente, nous récupérons les capteurs qui nous intéressent et nous les écoutons.

De même on implémente la méthode `onSensorChanged` afin d'implémenter le comportement que l'on souhaite. Lors d'un mouvement nous affichons dans quelle direction le smartphone se déplace.

Nous avons déclarer une balise `TextView` dans le layout correspondant à cette activité. On le récupère dans le code avec `directionTextView = findViewById(R.id.directionTextView)`; puis on change le texte qu'il affiche avec du code : `directionTextView.setText("Left")`;



## 5 Secouer un appareil (*ShakeDevice*)

Cette activité (*ShakeDevice*) nous permet de faire basculer le flash entre allumé/éteint quand on secoue le smartphone.

Nous procémons comme dans les exercices précédent. On récupère le `sensorManager`. On récupère les capteurs (*l'accéléromètre et la caméra*). On s'enregistre auprès de lui. Il faut remarquer qu'on doit d'abord récupérer le manger de caméra et utiliser un try catch car il est possible qu'on ait plusieurs ou 0 caméra.

Nous avons choisi d'utiliser `SensorManager.SENSOR_DELAY_UI` pour pouvoir détecter les mouvements de manière plus fine.

```

sensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
sensor_Acc = sensorManager.getDefaultSensor(TYPE_ACCELEROMETER);
cameraManager = (CameraManager) getSystemService(CAMERA_SERVICE);
try {
    cameraId = cameraManager.getCameraIdList()[0];
} catch (CameraAccessException e) {
    e.printStackTrace();
}

sensorManager.registerListener(listener, this, sensor_Acc, SensorManager.SENSOR_DELAY_GAME);

```

On implémente les méthodes de callback. On remarquera aussi l'utilisation de `System.currentTimeMillis()` pour empêcher que le flash ne s'allume et s'éteigne au cours du même mouvement.

```

@Override
public void onSensorChanged(SensorEvent event) {
    if (event.sensor.getType() == TYPE_ACCELEROMETER) { onAccelerometerChanged(event); }
}

Usage : SateaMail
private void onAccelerometerChanged(SensorEvent event) {
    long currentTimeMillis = System.currentTimeMillis();
    if ((currentTimeMillis - lastShakeTimestamp) > 1000) {
        float x = event.values[0];
        float y = event.values[1];
        float z = event.values[2];

        float shakeMagnitude = (float) Math.sqrt(x * x + y * y + z * z) / SensorManager.GRAVITY_EARTH;
        if (shakeMagnitude > SHAKE_THRESHOLD) {
            toggleFlashlight();
            lastShakeTimestamp = currentTimeMillis;
        }
    }
}

```

Si le smartphone est suffisamment secoué, on fait basculer l'état du flash comme sur la capture d'écran suivante :

```

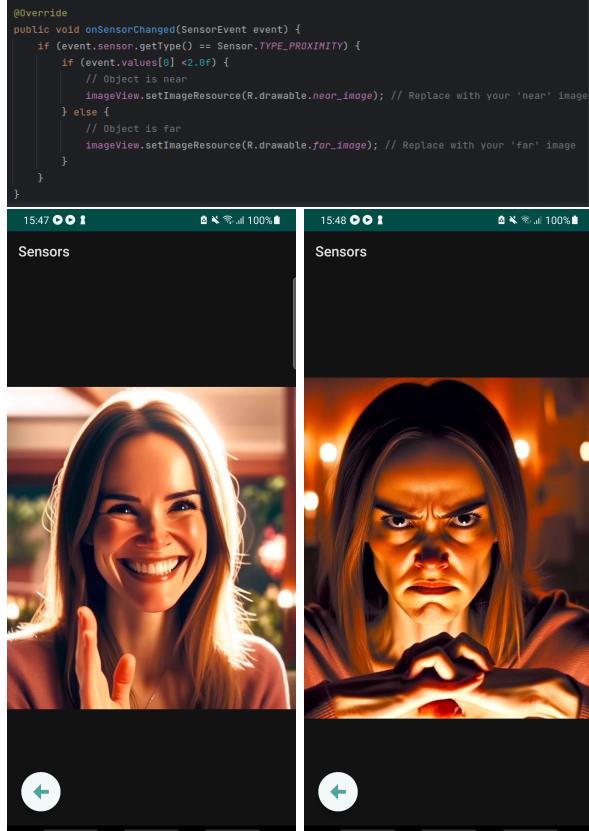
private void toggleFlashlight() {
    isFlashlightOn = !isFlashlightOn;
    try {
        cameraManager.setTorchMode(cameraId, isFlashlightOn);
    } catch (CameraAccessException e) {
        e.printStackTrace();
    }
}

```

## 6 Proximité (*Proximity*)

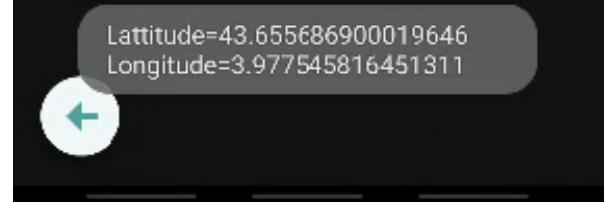
Nous affichons une image indiquant si l'objet est proche ou loin.

On récupère le sensorManager, le capteur et on s'enregistre auprès de lui encore une fois. On implémente les méthodes de callback. On déclare une <Imageview> dans le layout (*activity\_proximity.xml*) qu'on manipulera ensuite en java avec la méthode **setImageRessource()**.



Puis, quand on reçoit l'autorisation (en redéfinissant **onRequestPermissionsResult()**), On appelle la fonction **getLastKnownLocation(GPS\_PROVIDER)** qui nous rend une *Location*. Il nous suffit maintenant d'utiliser les fonctions **getLatitude()** et **getLongitude()**.

```
@Override  
public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions, @NonNull int[] grantResults) {  
    if (requestCode == 0) {  
        if (ActivityCompat.checkSelfPermission(this, Manifest.permission.ACCESS_FINE_LOCATION) != PackageManager.PERMISSION_GRANTED &&  
            ActivityCompat.checkSelfPermission(this, Manifest.permission.ACCESS_COARSE_LOCATION) != PackageManager.PERMISSION_GRANTED) {  
            return;  
        }  
        Location location = locationManager  
            .getLastKnownLocation(LocationManager.GPS_PROVIDER);  
        String lat = null;  
        String longi = null;  
        if (location != null) {  
            lat = String.valueOf(location.getLatitude());  
            longi = String.valueOf(location.getLongitude());  
        }  
  
        String s = "Latitude=" + lat + " Longitude=" + longi;  
        Toast.makeText(this, s, Toast.LENGTH_SHORT).show();  
  
        //locationTextView.setText(s);  
    }  
    super.onRequestPermissionsResult(requestCode, permissions, grantResults);  
}
```



## 7 Géolocalisation (*Geolocation*)

Dans de nombreuses applications, il est souhaitable de connaître la position de l'utilisateur. Pour ce faire, nous allons manipuler le gps.

Pour commencer nous accédons à un objet de type *LocationManager* grâce à la fonction **getSystemService(Context.LOCATION\_SERVICE)**.

```
locationManager = (LocationManager) getSystemService(Context.LOCATION_SERVICE);  
  
ActivityCompat.requestPermissions(activity, new String[] {  
    Manifest.permission.ACCESS_NETWORK_STATE,  
    Manifest.permission.ACCESS_FINE_LOCATION,  
    Manifest.permission.ACCESS_COARSE_LOCATION},  
    requestCode);
```

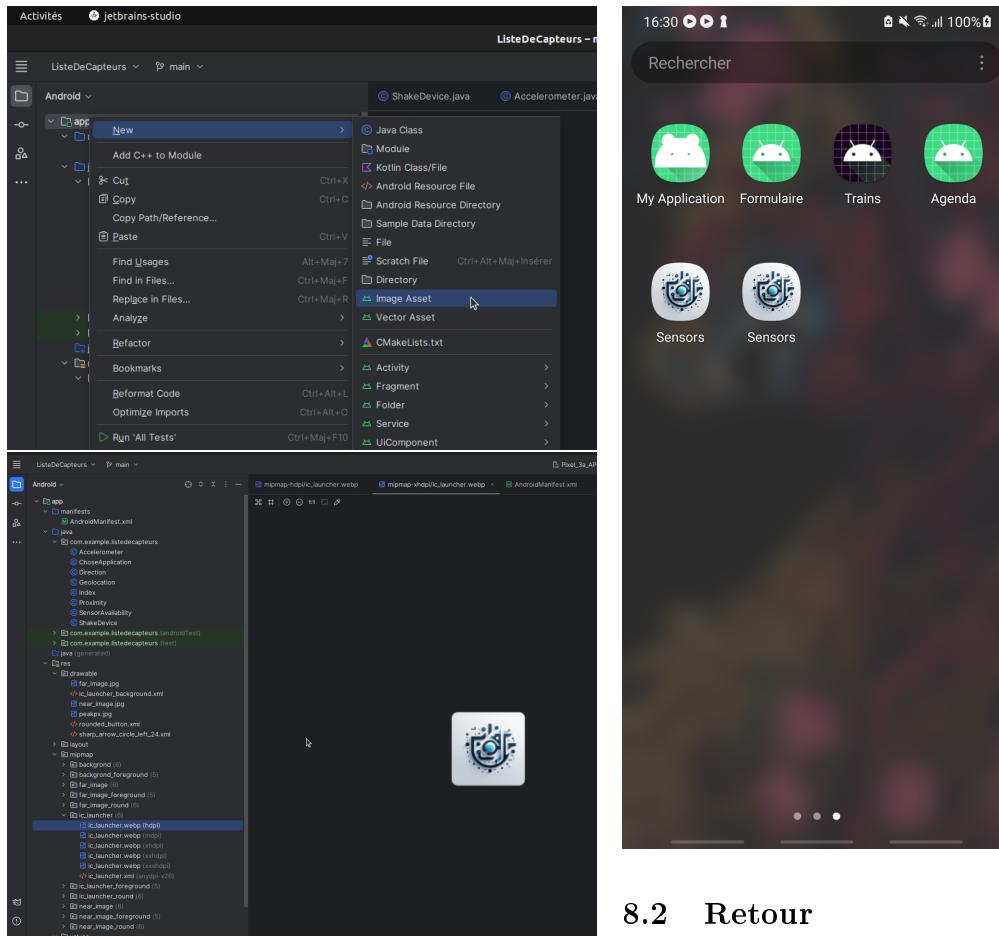
Ensuite, on demande les permissions pour avoir la localisation de l'utilisateur.

## 8 Extra

### 8.1 Icône

Nous avons choisi de modifier l'icône de notre application. Nous allons maintenant expliquer comment nous cela a été fait.

Dans Android Studio, faire un click droit sur le dossier app. Aller dans New > Image Asset.



## 8.2 Retour

Nous avons placer un bouton de retour vers l'écran d'accueil.

Dans le layout, nous ajoutons un bouton flottant (*FloatingActionButton*) dans le coin inférieur gauche de la vue. Et à l'aide des connaissances du 1er tp, on redéfinit la méthode `onClick()`. On comment l'activité correspondant à l'accueil avec un `Intent` et la méthode `startActivity()`.

```
<com.google.android.material.floatingactionbutton.FloatingActionButton
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:id="@+id/my_button"
    android:contentDescription="Home"
    app:srcCompat="@drawable/sharp_arrow_circle_left_24"
    app:maxImageSize="67dp"
    app:tint="#EEFFFFF"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    android:layout_marginStart="20dp"
    android:layout_marginBottom="20dp"
/>

myButton = findViewById(R.id.my_button);
@Override
public void onClick(View view) {
    Intent intent = new Intent(getApplicationContext(), MainActivity.class);
    startActivity(intent);
}
```