

Programmation mobile

Cours 4 : Cas d'étude d'un `Sensor`, l'accéléromètre

Julien Grange <julien.grange@lacl.fr>

Mardi 22 octobre 2024

Contrairement aux logiciels de bureau, une application mobile se base souvent sur l'environnement de l'appareil.

Pour récupérer ces informations extérieures, on utilise des capteurs divers, qui sont représentés par des `Sensor`.

L'ensemble des `Sensor` est géré par le `SensorManager`.

- accéléromètre (`TYPE_ACCELEROMETER`)
- capteur de lumière (`TYPE_LIGHT`)
- thermomètre (`TYPE_TEMPERATURE`)
- compteur de pas (`TYPE_STEP_COUNTER`)

- accéléromètre (`TYPE_ACCELEROMETER`)
- capteur de lumière (`TYPE_LIGHT`)
- thermomètre (`TYPE_TEMPERATURE`)
- compteur de pas (`TYPE_STEP_COUNTER`)

Certains `Sensor` sont **physiques**

- accéléromètre (`TYPE_ACCELEROMETER`)
- capteur de lumière (`TYPE_LIGHT`)
- thermomètre (`TYPE_TEMPERATURE`)
- compteur de pas (`TYPE_STEP_COUNTER`)

Certains `Sensor` sont **physiques**, d'autre **logiques**.

Un appareil ne possède jamais tous les `Sensor` !

Avant d'utiliser un `Sensor`, il faut vérifier qu'il est bien présent sur l'appareil. Pour cela, deux possibilités :

- limiter l'installation de l'application aux appareils le possédant :

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

- vérifier sa présence à runtime

Un appareil ne possède jamais tous les `Sensor` !

Avant d'utiliser un `Sensor`, il faut vérifier qu'il est bien présent sur l'appareil. Pour cela, deux possibilités :

- limiter l'installation de l'application aux appareils le possédant :

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

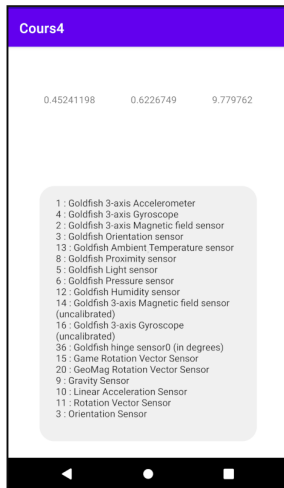
- vérifier sa présence à runtime

Certains `Sensor` (e.g. le capteur de pouls) demandent une permission particulière, à déclarer dans le Manifest.

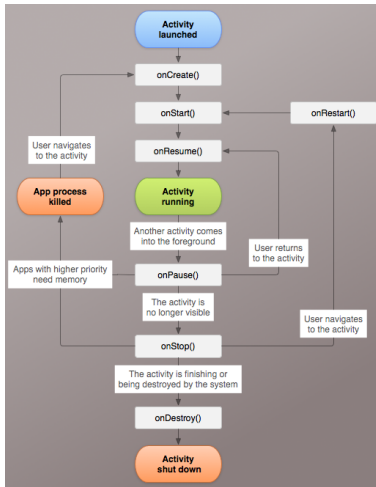
Lister les `Sensor`, tester l'existence d'un `Sensor`

```
// on récupère et on affiche la liste des senseurs disponibles
SensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
List<Sensor> sensorList = sensorManager.getSensorList(Sensor.TYPE_ALL);
StringBuilder stringBuilder = new StringBuilder();
for(Sensor sensor : sensorList) {
    stringBuilder.append(sensor.getType()+" : "+sensor.getName()+"\n");
}
Toast.makeText(context: this, stringBuilder.toString(), Toast.LENGTH_LONG).show();

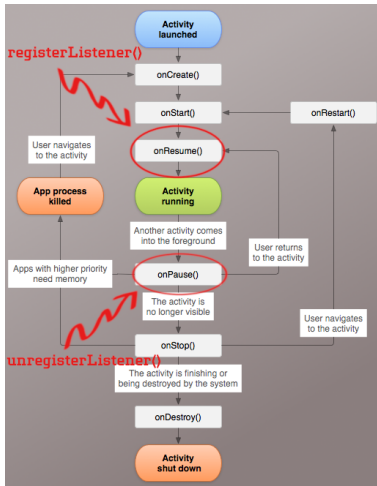
// s'il n'y a pas d'accéléromètre, on quitte brutalement
if((sensor = sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER)) == null){
    Log.v(tag: "Cours4", msg: "Pas d'accéléromètre");
    finish();
}
```



Cycle de vie d'une **Activity** et monitoring d'un **Sensor**



Cycle de vie d'une **Activity** et monitoring d'un **Sensor**



`SensorManager::registerListener()` :

```
// on s'enregistre auprès du senseur à chaque fois que l'application repasse au premier plan
@Override
protected void onResume(){
    super.onResume();
    sensorManager.registerListener(listener, sensor, sensorManager.SENSOR_DELAY_NORMAL);
}
```

`SensorManager::unregisterListener()` :

```
// pour économiser la batterie, on évite d'écouter un senseur quand on n'est pas au premier plan
@Override
protected void onPause() {
    super.onPause();
    sensorManager.unregisterListener(listener);
}
```

SensorEventListener

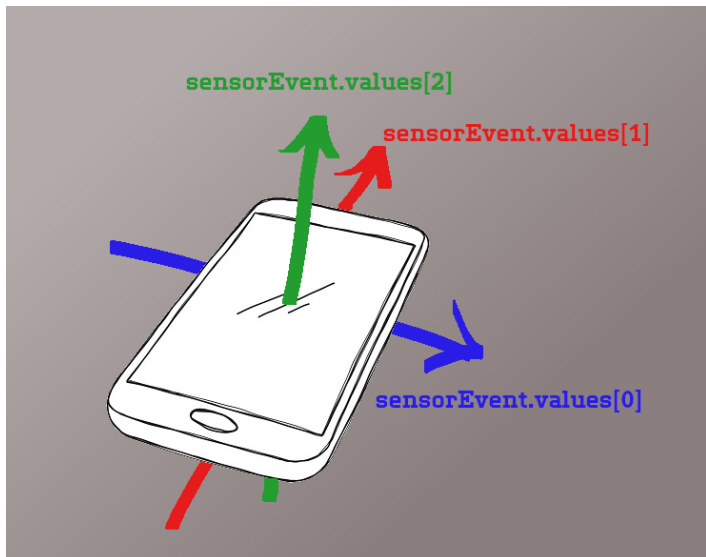
Le premier argument de `SensorManager::registerListener()` doit implémenter l'interface `SensorEventListener` :

```
// on se souvient de ses cours de physique :  $g \approx 9,81 \text{ m/s}^2$ 
private static final float g = 9.81f;

private SensorEventListener listener = new SensorEventListener() {
    @Override
    public void onSensorChanged(SensorEvent sensorEvent) {
        // pour chaque accélération détectée...
        for (int i = 0; i < 3; i++) {
            // ...on affiche les coordonnées du vecteur...
            views[i].setText(Float.toString(sensorEvent.values[i]));
        }
        //...on enregistre une nouvelle position...
        dessin.newPosition(x: sensorEvent.values[0]/g, y: sensorEvent.values[1]/g);
        //...et on demande à la View de se redessiner
        dessin.invalidate();
    }

    @Override
    public void onAccuracyChanged(Sensor sensor, int i) {}
};
```

Zoom sur un `Sensor` particulier : l'accéléromètre



Première expérience

```
@Override  
public void onSensorChanged(SensorEvent sensorEvent) {  
    // pour chaque accélération détectée...  
    for (int i = 0; i < 3; i++) {  
        // ...on affiche les coordonnées du vecteur...  
        views[i].setText(Float.toString(sensorEvent.values[i]));  
    }  
}
```

Extended Controls - Pixel_XL_API_30:5554

Device Pose Additional sensors



Rotate Move

Z-Rot -180 180 0.0

X-Rot -180 180 0.0

Y-Rot -180 180 0.0

Rotation



Sensor values

Accelerometer (m/s ²):	0. 9.81 0.00
	00
Gyroscope (rad/s):	0. 0.00 0.00
	00
Magnetometer (μT):	0. 5.90 -48.40
	00

Cours4

0.0 9.809989 0.0




Deuxième expérience

```
@Override
public void onSensorChanged(SensorEvent sensorEvent) {
    // pour chaque accélération détectée...
    for (int i = 0; i < 3; i++) {
        // ...on affiche les coordonnées du vecteur...
        views[i].setText(Float.toString(sensorEvent.values[i]));
    }
}
```

Extended Controls - Pixel_XL_API_30:5554

Location
Displays
Cellular
Battery
Camera
Phone
Directional pad
Microphone
Fingerprint
Virtual sensors
Bug report
Snapshots
Record and Playback
Settings
Help

Device Pose Additional sensors



Rotate Move

Rotation

Z-Rot 0.0

X-Rot -90.0

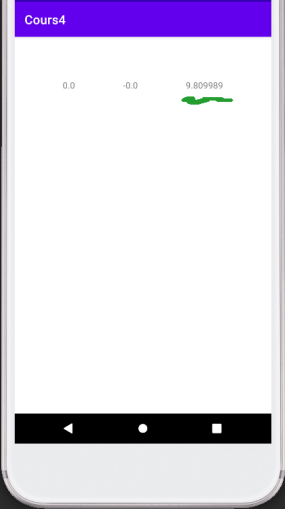
Y-Rot 0.0

Sensor values

Accelerometer (m/s ²):	0.00	-0.00	9.8100
Gyroscope (rad/s):	0.00	0.00	0.00
Magnetometer (µT):	0.00	48.40	5.9000

Cours4

0.0 -0.0 9.809989

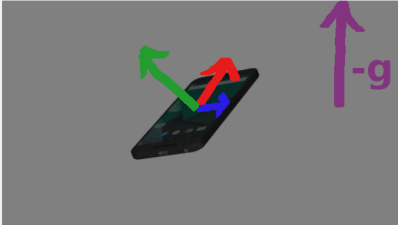


Troisième expérience

```
@Override
public void onSensorChanged(SensorEvent sensorEvent) {
    // pour chaque accélération détectée...
    for (int i = 0; i < 3; i++) {
        // ...on affiche les coordonnées du vecteur...
        views[i].setText(Float.toString(sensorEvent.values[i]));
    }
}
```

Extended Controls - Pixel_XL_API_30:5554

Device Pose Additional sensors



Rotation

Rotate Move

Z-Rot -180 180 -24.8

X-Rot -180 180 -68.2


Y-Rot -180 180 -20.3

Sensor values

Accelerometer (m/s ²):	1.34	4.63	8.54
Gyroscope (rad/s):	0.00	0.00	0.00
Magnetometer (μT):	-23.73	40.96	-11.71
Rotation:	ROTATION_0		

Cours4

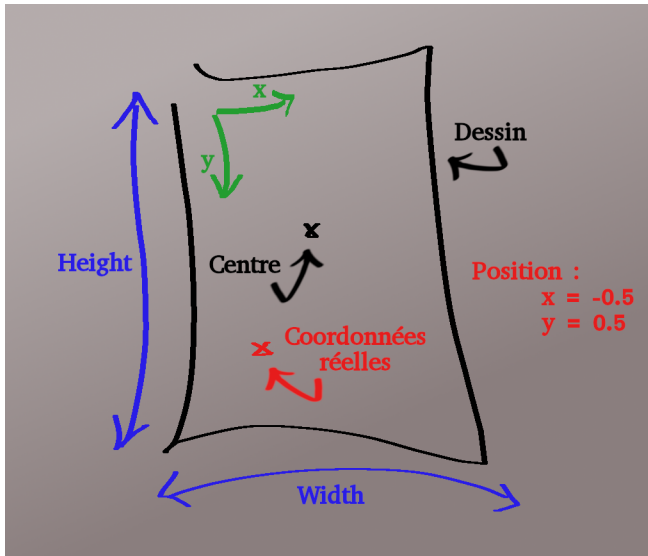
1.3377047 4.6307354 8.544178



On veut tracer à l'écran une bulle de niveau, pour nous aider à bricoler.

On veut tracer à l'écran une bulle de niveau, pour nous aider à bricoler.

- 1 `Dessin` : hérite de `View`, contient les coordonnées de la bulle
- 2 un relevé de l'accéléromètre → une nouvelle position pour la bulle
- 3 à chaque `onDraw()` du `Dessin`, on dessine la bulle



Dessin et SensorEventListener

```
ty_main.xml x MainActivity.java x Dessin.java x
package com.example.cours4;

import ...

public class MainActivity extends AppCompatActivity {

    private SensorManager sensorManager;
    private Sensor sensor;
    private TextView[] views = new TextView[3];
    private ConstraintLayout constraintLayout;
    private Dessin dessin;
    // on se souvient de ses cours de physique : g == 9,81 m/s^2
    private static final float g = 9.81f;

    private SensorEventListener listener = new SensorEventListener() {
        @Override
        public void onSensorChanged(SensorEvent sensorEvent) {
            // pour chaque accélération détectée...
            for (int i = 0; i < 3; i++) {
                // ...on affiche les coordonnées du vecteur...
                views[i].setText(Float.toString(sensorEvent.values[i]));
            }
            //...on enregistre une nouvelle position...
            dessin.newPosition(x sensorEvent.values[0]/g, y sensorEvent.values[1]/g);
            //...et on demande à la View de se redessiner
            dessin.invalidate();
        }

        @Override
        public void onAccuracyChanged(Sensor sensor, int i) {}
    };

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        public class Dessin extends View {

            private float[] bulle;
            private int[] dimensions;
            private int[] center;
            Paint paint_disque, paint_cercle;

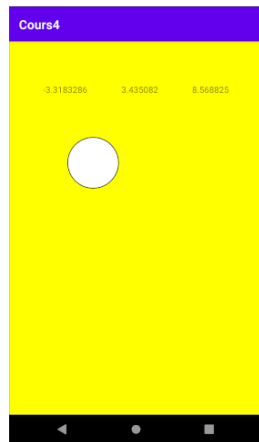
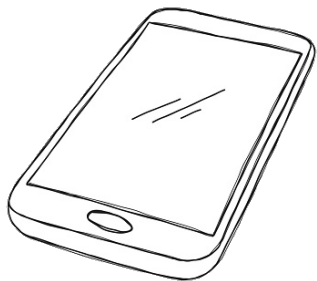
            public Dessin(Context context){
                super(context);
                bulle = new float[2];
                dimensions = new int[2]; center = new int[2];
                paint_disque = new Paint();
                paint_disque.setColor(Color.WHITE);
                paint_disque.setStyle(Paint.Style.FILL);
                paint_cercle = new Paint();
                paint_cercle.setColor(Color.BLACK);
                paint_cercle.setStyle(Paint.Style.STROKE);
                paint_cercle.setStrokeWidth(5f);
                setBackgroundColor(Color.YELLOW);
            }

            public void newPosition(float x, float y) {
                bulle[0] = center[0] + x * dimensions[0] / 2;
                bulle[1] = center[1] - y * dimensions[1] / 2;
            }

            private void computeView(){
                dimensions[0] = getWidth();
                dimensions[1] = getHeight();
                center[0] = dimensions[0]/2;
                center[1] = dimensions[1]/2;
            }

            @Override
            public void onDraw(Canvas canvas) {
                super.onDraw(canvas);
                computeView();
                // on trace la bulle
                float tailleBulle = Math.min(dimensions[0], dimensions[1])/10;
                canvas.drawCircle(bulle[0], bulle[1], tailleBulle, paint_disque);
                canvas.drawCircle(bulle[0], bulle[1], tailleBulle, paint_cercle);
            }
        }
    }
}
```

Essai sur le simulateur



Essai sur le simulateur

