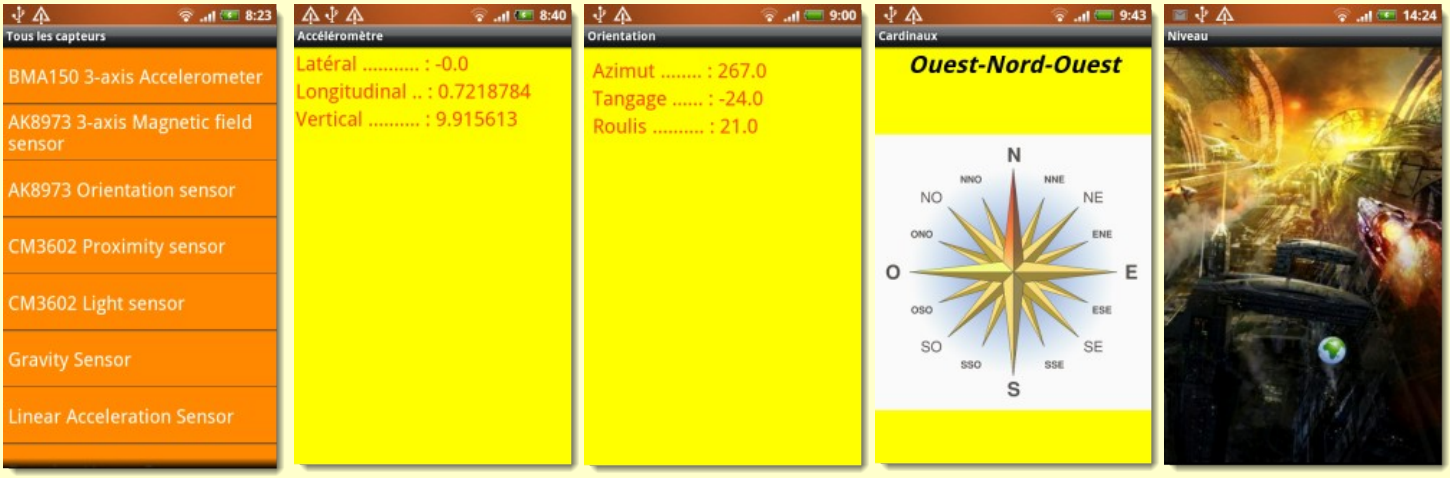
Les dernières générations de téléphones tentent d'embarquer de plus en plus de fonctionnalités matérielles : ***GPS, boussole, gyroscope, Wi-Fi, Bluetooth, capteur magnétique, capteur de température, etc.*** Bien évidemment, en tant que développeur, vous comprendrez immédiatement que ce sont autant d'opportunités de créer des applications toujours plus immersives, communicantes et interactives !

- **Identification des capteurs :** Trouver tous les capteurs disponibles dans un smartphone particulier.
- Détecter les changements d'accélération.
- Déterminer votre orientation.
- Donner le cap.
- Déplacement d'une boule suivant l'horizontalité du mobile.



LES RESSOURCES DISPONIBLES SUR UN APPAREIL ANDROID

La plate-forme Android gère de façon native bon nombre de ressources matérielles telles que le clavier et le trackball qui ne nécessitent pas vraiment d'adaptation lorsque ceux-ci ne sont pas disponibles sur le terminal. Par exemple, le clavier physique sera remplacé par un clavier virtuel sans avoir besoin de reprogrammer l'interface utilisateur.

Liste des principales ressources utilisables depuis Android

Affichage
- *- Appareil photo*
- *- 3D*

Entrées
- *- Clavier*
- *- Matrice tactile*
- *- Boule de commande (trackball)*

Son
- *- Haut parleur / Microphone*
- *- Reconnaissance vocale*
- *- Vibration*

Réseau
- *- 3G*
- *- Wi-Fi*
- *- Bluetooth*
- *- GPS*

Capteurs
- *- Accéléromètre*
- *- Orientation*
- *- Champ magnétique*
- *- Température*

✗ IDENTIFICATION DES CAPTEURS

Le panel des capteurs s'est étoffé avec les versions successives d'Android : ***accéléromètre, capteur de pression, capteur de proximité...*** Ceci dit, il ne faut pas croire que tous les capteurs sont disponibles sur tous les terminaux, loin de là !

Nota : L'API d'**Android 1.6** a donc introduit une nouvelle classe ***Sensor*** responsable de la récupération des données, sous la houlette d'un autre classe nommée ***SensorManager*** qui comme son nom l'indique chapeaute le groupe de capteurs au travers d'un service commun à tout le système. SensorManager est ainsi utilisé pour gérer l'ensemble des capteurs disponibles sur les appareils Android.

✗ Utilisez la méthode ***getSystemService()*** pour renvoyer une référence vers le service ***SensorManager*** :

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

✗ Ensuite, la classe ***Sensor*** contient un ensemble de constantes utilisées pour décrire quel type de capteur matériel est représenté par un objet de type Sensor. Ces constantes sont de la forme **Sensor.TYPE_<Type>.** La section suivante décrit chaque type de capteur supporté. Vous apprendrez ensuite comment trouver et utiliser chacun d'eux.

Liste des différents types de capteurs disponible

Sensor.TYPE_ACCELEROMETER	Permet d'évaluer les mouvements du terminal ou tout simplement la gravité. Accéléromètre à trois axes renvoyant l'accélération courante en **m/s2**.
Sensor.TYPE_GYROSCOPE	Permet de connaître la position angulaire en degré du terminal en fonction des trois axes **x, y et z**.
Sensor.TYPE_LIGHT	Le capteur de lumière permet d'évaluer l'exposition lumineuse subie par le terminal. Capteur de lumière ambiante renvoyant une valeur exprimée en **lux**. Le capteur de lumière est couramment utilisé pour contrôler la luminosité de l'écran.
Sensor.TYPE_MAGNETIC_FIELD	Permet de détecter les modifications des champs magnétiques environnants. Capteur de champ magnétique renvoyant le champ en **microteslas** suivant les trois axes.
Sensor.TYPE_ORIENTATION	Editer une donnée.Permet de déterminer la position du terminal en fonction des trois axes **x, y et z**.
Sensor.TYPE_PRESSURE	Indique la pression atmosphérique. Capteur de pression renvoyant une valeur exprimée en **kilopascals**.
Sensor.TYPE_PROXIMITY	Indique la distance du terminal par rapport à un objet. Capteur de proximité indiquant la distance entre l'appareil et un objet cible en **mètres**. La façon dont l'objet est sélectionné et la distance supportée dépendront de l'implémentation matérielle du détecteur de proximité. Un usage typique est la détection de l'approche de l'appareil de l'oreille de l'utilisateur, l'ajustement automatique de la luminosité de l'écran ou le lancement d'une commande vocale.
Sensor.TYPE_TEMPERATURE	Indique la température à proximité du capteur. Thermomètre renvoyant la température en **degré Celsius**. Il peut s'agir de la température ambiante, de celle de la batterie ou d'un capteur distant en fonction de l'implémentation matérielle.

✗ TROUVER LES CAPTEURS

Un appareil Android peut contenir plusieurs implémentations d'un type de capteur particulier. Pour déterminer l'implémentation par défaut, utilisez la méthode ***getDefaultSensor()*** issue de la classe ***SensorManager*** en lui passant le type de capteur requis sous la forme d'une des constantes décrites dans la section précédente.

✗ Si le capteur n'est pas disponible sur le terminal, la méthode ***getDefaultSensor()*** renverra une valeur **null** nous permettant d'agir en conséquence.

```
SensorManager gestionCapteurs =  (SensorManager) getSystemService(Context.SENSOR_SERVICE);
Sensor gyroscope = gestionCapteurs.getDefaultSensor(Sensor.TYPE_GYROSCOPE);
```

✗ De façon alternative, utilisez plutôt la méthode ***getSensorList()*** pour renvoyer la liste de tous les capteurs d'un type donné comme dans le code suivant, qui renvoie tous les capteurs de pression :

```
SensorManager gestionCapteurs =  (SensorManager) getSystemService(Context.SENSOR_SERVICE);
List<Sensor> capteursPression = gestionCapteurs.getSensorList(Sensor.TYPE_PRESSURE);
```

✗ Enfin, pour déterminer tous les capteurs disponibles sur une plate-forme hôte, utilisez de nouveau la méthode ***getSensorList()***, en lui passant cette fois-ci la constante **Sensor.TYPE_ALL**. Cette technique vous permet de déterminer quels capteurs et types de capteurs sont disponibles sur l'appareil.

```
SensorManager gestionCapteurs =  (SensorManager) getSystemService(Context.SENSOR_SERVICE);
List<Sensor> capteurs = gestionCapteurs.getSensorList(Sensor.TYPE_ALL);
```

Je vous propose de réaliser un premier projet qui permet de recenser tous les capteurs que vous avez à votre disposition sur votre terminal.

res/layout/main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<ListView xmlns:android = "http://schemas.android.com/apk/res/android"
      android:id="@android:id/list"
      android:layout_width = "fill_parent"
      android:layout_height = "fill_parent"
      android:background = "#FF8800"
/>
```

fr.btsiris.capteurs.Capteur.java

```
package fr.btsiris.capteurs;

import android.app.ListActivity;
import android.content.Context;
import android.hardware.*;
import android.os.Bundle;
import android.widget.ArrayAdapter;
import java.util.List;

public class Capteur extends ListActivity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        SensorManager gestionCapteurs =  (SensorManager) getSystemService(Context.SENSOR_SERVICE);
        List<Sensor> capteurs = gestionCapteurs.getSensorList(Sensor.TYPE_ALL);
        String[] listeCapteurs = new String[capteurs.size()];
        for (int i=0; i<capteurs.size(); i++) listeCapteurs[i] = capteurs.get(i).getName();
        setListAdapter(new ArrayAdapter<String>(this, android.R.layout.simple_list_item_1, listeCapteurs));
    }
}
```

COMMENT UTILISER LES CAPTEURS

Pour accéder aux capteurs, après avoir récupérer l'instance du service, vous devez vous abonner aux événements du capteur qui vous intéressent.

Pour cela, vous devez implémenter l'interface ***SensorEventListener*** *et redéfinir les méthodes adpatées à ce type d'événements :*

- ***onSensorChanged()*** pour monitorer les valeurs du capteur et d'autre part la méthode
- ***onAccuracyChanged()*** pour réagir à ses changements de précision.

```
class Scrutation implements SensorEventListener {
    public void onSensorChanged(SensorEvent evt) {
        // A faire : Monitorer les changements dans le capteur
    }

    public void onAccuracyChanged(Sensor capteur, int accuracy) {
        // A faire : Réagir aux demandes de changement de précision du capteur
    }
}
```

Nota *: Le paramètre de type* ***SensorEvent*** *dans la méthode* ***onSensorChanged()*** *contient quatre propriétés utilisées pour décrire un événement particulier du capteur.*

- **sensor** : L'objet de type **Sensor** ayant déclanché l'événement.
- **accuracy** : La précision du capteur lorsque l'événement est survenu (*faible, moyenne, haute ou non fiable*).
- **values** : Tableau de flottants contenant la ou les nouvelles valeurs détectées.
- **timestamp** : L'horodatage (*en nanosecondes*) auquel l'événement est survenu.

Nota *: Vous pouvez monitorer séparément les changements dans la précision d'un capteur en utilisant la méthode* ***onAccuracyChanged()****. Dans les deux gestionnaires, la valeur* ***accuracy*** *représente la précision du capteur monitoré sous la forme de constantes prédéfinies dont la liste est proposée ci-dessous.*

Précision du capteur

- **SensorManager.SENSOR_STATUS_ACCURACY_LOW** : Indique que le capteur possède une faible précision et doit être recalibré.
- **SensorManager.SENSOR_STATUS_ACCURACY_MEDIUM** : Indique que les données du capteur sont de précision intermédiaire et qu'une recalibration améliorerait la lecture.
- **SensorManager.SENSOR_STATUS_ACCURACY_HIGH** : Indique que le capteur a la précision la plus élevée possible.
- **SensorManager.SENSOR_STATUS_UNRELIABLE** : Indique que les données du capteur ne sont pas fiable et qu'une calibration est nécessaire ou que la lecture n'est pas possible.

Nota *: Pour obtenir les données, nous devons nous enregistrer auprès du service comme écouteur de type* ***SensorEventListener*** *à l'aide de la méthode* ***registerListener()*** *de la classe* ***SensorManager****. Spécifiez alors le capteur à observer et la fréquence à laquelle vous voulez recevoir les mises à jour.*

```
SensorManager gestionCapteurs =  (SensorManager) getSystemService(Context.SENSOR_SERVICE);
Sensor gyroscope = gestionCapteurs.getDefaultSensor(Sensor.TYPE_GYROSCOPE);
Scrutation scrutation = new Scrutation();
gestionCapteurs.registerListener(scrutation, gyroscope, SensorManager.SENSOR_DELAY_NORMAL);
```

> ***Nota*** *: Comme nous le constatons, la récupération d'une instance de capteur s'associe à un taux de rafraîchissement des données qui peut être plus ou moins précis. Ce taux influence directement les performances de l'application ainsi que l'usage de la batterie. Plus le taux est élevé, plus les données sont rafraîchies et précises, et plus la consommation d'énergie est élevée. Le* ***SensorManager*** *contient les constantes suivantes (dans l'ordre décroissant de réactivité) pour sélectionner une fréquence appropriée :*

Taux de rafraîchissement des mesures effectuées

- **SensorManager.SENSOR_DELAY_FASTER** : Le taux de rafraîchissement le plus élevé. Les données sont récupérées aussi vite que possible par le système.
- **SensorManager.SENSOR_DELAY_GAME** : Taux utilisable pour des applications nécessitant des données précises telles que les jeux.
- **SensorManager.SENSOR_DELAY_NORMAL** : Taux normal utilisé par exemple par le système pour détecter les changements d'orientation.
- **SensorManager.SENSOR_DELAY_UI** : Le taux le plus bas utilisé dans les applications ne nécessitant qu'une faible précision ou ne sollicitant que très peu de données, comme les composants graphiques d'une interface utilisateur.

> *Nota : La fréquence que vous sélectionnez n'est pas obligatoire. Le* ***SensorManager*** *peut répondre plus ou moins vite que ce que vous avez spécifié, mais il a tendance à être plus rapide. Pour minimiser le coût en ressources d'utilisation d'un capteur dans votre application, sélectionnez la fréquence la plus faible.*

- **IMPORTANT** : Il est également important de désenregistrer votre **SensorEventListener** lorsque votre application n'a plus besoin de recevoir de mise à jour, en utilisant la méthode opposée ***unregisterListener()***. Il est d'ailleurs conseillé d'enregistrer et de désenregistrer votre **SensorEventListener** dans les méthodes de rappel ***onResume()*** et ***onPause()*** de vos activités pour garantir qu'ils ne sont utilisés que lorsque votre activité est active.

INTERPRÉTER LES VALEURS DES CAPTEURS

La longueur et la composition des valeurs renvoyées par l'événement ***onSensorChanged()*** varient selon le capteur monitoré. Les détails sont donnés ci-dessous. Des détails complémentaires , du capteur d'orientation et du capteur de champ magnétique seront donnés dans les sections suivantes.

Type de capteur	Nombre	Contenu des valeurs	Commentaire
TYPE_ACCELEROMETER	3	value[0] : Latéral value[1] : Longitudinal value[2] : Vertical	Accélération suivant trois axes en m/s^2. Le ***SensorManager*** contient un ensemble de constantes de gravité de la forme ***SensorManager.GRAVITY_**.
TYPE_GYROSCOPE	3	value[0] : Azimut value[1] : Tangage value[2] : Roulis	Orientation de l'appareil en degrés suivant trois axes.
TYPE_LIGHT	1	value[0] : Luminosité	Mesurée en lux. Le ***SensorManger*** contient un ensemble de constantes représentant différentes luminosités de la forme ***SensorManager.LIGNT_**.
TYPE_MAGNETIC_FIELD	3	value[0] : Latéral value[1] : Longitudinal value[2] : Vertical	Champ magnétique local mesuré en microteslas (µT).
TYPE_ORIENTATION	3	value[0] : Azimut value[1] : Tangage value[2] : Roulis	Orientation de l'appareil en degrés suivant trois axes.
TYPE_PRESSURE	1	value[0] : Pression	Mesurée en kilopascals (KP).
TYPE_PROXIMITY	1	value[0] : Distance	Mesurée en mètres.
TYPE_TEMPERATURE	1	value[0] : Température	Mesurée en degré Celsius.

ACCÉLÉROMÈTRES

Les accéléromètres, comme leur nom l'indique, sont utilisés pour mesurer l'accélération. Ils sont aussi parfois nommés capteurs de gravité.

> ***Nota*** *: Les accéléromètres sont appelés capteurs de gravité en raison de leur incapacité à distinguer une accélération due à un mouvement d'une accélération due à la gravité. Par conséquent, une accélération détectant une accélération sur l'axe z (vertical) lira* ***-9.81 m/s2*** *lorsque l'appareil sera immobile (cette valeur est disponible dans la constante* ***SensorManager.STANDARD_GRAVITY****).*

- L'accélération est définie comme **la modification de la vitesse d'un mouvement** et les accéléromètres mesureront donc comment la vitesse de l'appareil change dans une direction donnée. En utilisant un accéléromètre, vous pourrez détecter les mouvements et, plus utilement, la variation de leur vitesse.
- Il est important de noter que les accéléromètres ne mesurent pas la vélocité, et vous ne pourrez donc pas mesurer directement la vitesse par une seule lecture de l'accéléromètre. A la place, vous devrez mesurer les changements de vitesse dans le temps.

> *Généralement, vous serez intéressé par les changements de vitesse par rapport à un état repos ou par des mouvements rapides (repérables par des changements rapides dans l'accélération) comme des gestes de l'utilisateur. Dans le premier cas, vous devrez souvent calibrer l'appareil pour calculer l'orientation et l'accélération initiales afin de les prendre en compte dans les résultats futurs.*

DÉTECTER LES CHANGEMENTS D'ACCÉLÉRATION

Comme nous l'avons découvert dans le tableau un peu plus haut, le **SensorManager** renvoie les modifications selon trois axes. Les valeurs de la propriété values du paramètre de type **SensorEvent** issu de l'écouteur **SensorEventListener** représente latérale, longitudinale et verticale dans cet ordre.

> *Le* ***SensorManager*** *considère qu'un appareil est* ***"au repos"*** *lorsqu'il est posé face vers le haut en mode portrait sur une surface plane. L'accélération peut donc être mesurée selon trois directions :*

- **latéral** : *gauche-droite* : Accélération vers la gauche ou la droite, les valeurs positives représentent la droite et les négatives, la gauche. Une accélération latérale positive sera détectée sur un appareil posé à plat sur son dos en mode portrait et déplacé vers la droite.
- **longitudinal** : *avant-arriere* : Accélération vers l'avant ou vers l'arrrière, une valeur positive représentant l'avant. Dans la même configuration que ci-dessus, une accélération longitudinale positive sera créée en déplaçant l'appareil vers l'avant.
- **verticale** : *haut-bas* : Accélération vers le haut ou vers le bas, une valeur positive représentant le haut. Au repos, l'accéléromètre enregistre une valeur de **9.81 m/s2** du fait de la gravité.

*Comme nous l'avons décrit plus haut, vous pouvez monitorer les changements dans l'accélération en utilisant un **SensorEventListener**. Enregistrer une implémentation de ce type d'écouteur avec le **SensorManager** en utilisant un objet **Sensor** de type **Sensor.TYPE_ACCELEROMETER** pour interroger l'accéléromètre.*

*Au travers d'un **deuxième petit projet,** nous allons tester simplement l'accéléromètre intégré du terminal en affichant sous forme de texte la valeur des trois axes représentatifs. Nous prenons en compte l'accéléromètre par défaut en utilisant une fréquence normale.*

res/layout/main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
      android:orientation="vertical"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:padding="2px"
      android:background="#FFFF00" >
    <TextView
        android:id="@+id/lateral"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:textSize="23dp"
        android:textColor="#FF4400" />
    <TextView
        android:id="@+id/longitudinal"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:textSize="23dp"
        android:textColor="#FF4400" />
    <TextView
        android:id="@+id/vertical"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:textSize="23dp"
        android:textColor="#FF4400" />
</LinearLayout>
```

fr.btsiris.capteurs.Accelerometre.java

```
package fr.btsiris.capteurs;

import android.app.Activity;
import android.content.Context;
import android.hardware.*;
import android.os.Bundle;
import android.widget.TextView;

public class Accelerometre extends Activity implements SensorEventListener {
    private SensorManager gestionCapteurs;
    private Sensor accelerometre;
    private TextView lateral, longitudinal, vertical;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        lateral = (TextView) findViewById(R.id.lateral);
        longitudinal = (TextView) findViewById(R.id.longitudinal);
        vertical = (TextView) findViewById(R.id.vertical);
        gestionCapteurs =  (SensorManager) getSystemService(Context.SENSOR_SERVICE);
        accelerometre = gestionCapteurs.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
    }

    @Override
    protected void onStart() {
        super.onStart();
        gestionCapteurs.registerListener(this, accelerometre, SensorManager.SENSOR_DELAY_NORMAL);
    }
```

```
    @Override
    protected void onStop() {
        super.onStop();
        gestionCapteurs.unregisterListener(this);
    }

    public void onSensorChanged(SensorEvent evt) {
        if (evt.sensor.getType() == Sensor.TYPE_ACCELEROMETER) {
            lateral.setText("Latéral ..... : "+evt.values[0]);
            longitudinal.setText("Longitudinal ... : "+evt.values[1]);
            vertical.setText("Vertical ..... : "+evt.values[2]);
        }
    }

    public void onAccuracyChanged(Sensor capteur, int precision) {  }
}
```

DÉTERMINER VOTRE ORIENTATION

Le capteur d'orientation est une combinaison de capteurs de champ magnétique qui fonctionne comme un compas électronique et un accéléromètre pour déterminer le tangage et le roulis.

En réalité, Android fournit deux alternatives pour déterminer l'orientation de l'appareil. Vous pouvez directement interroger le capteur d'orientation ou dériver celle-ci de l'accéléromètre et du capteur de champ magnétique. La dernière option prend plus de temps mais offre l'avantage d'une plus grande précision et de pouvoir modifier la trame de référence lorsque vous déterminer votre orientation.

En utilisant la trame de référence standard, l'orientation de l'appareil est donnée selon trois dimensions comme l'illustre la liste ci-dessous :

- **Azimut** : Axe qui pointe vers le haut : L'azimut (ou lacet) est la direction à laquelle l'appareil fait face suivant l'axe des x où ***0/360*** degrés est le nord, ***90*** degré l'est, ***180*** degré le sud et ***270*** degré l'ouest.
- **Tangage** : Axe qui pointe vers l'avant : Le tangage représente l'ange de l'appareil autour de l'axe des y. Lorsque l'appareil est à plat sur son dos, la valeur est ***0***. Lorsqu'il est ***"debout"*** (sommet de l'appareil vers le haut), la valeur est ***-90***. Lorsqu'il est vers le bas, la valeur est ***90*** et ***180/-180*** lorsqu'il est retourné.
- **Roulis** : Axe qui pointe vers le côté droit : Le roulis représente le mouvement de l'appareil autour de l'axe des z. Lorsque l'appareil est posé à plat sur son dos, la valeur est ***0***, ***-90*** lorsque l'écran fait face à la gauche et ***90*** lorsque l'écran fait face à la droite.

DÉTERMINER L'ORIENTATION AVEC LE CAPTEUR D'ORIENTATION

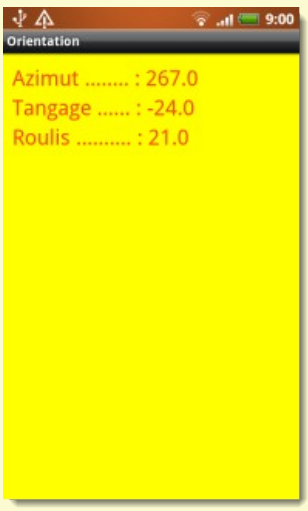
La façon la plus simple de monitorer l'orientation de l'appareil est d'utiliser le capteur d'orientation dédié. Créer et enregistrer un **SensorEventListener** avec le **SensorManager** en utilisant le capteur d'orientation par défaut.

```
SensorManager gestionCapteurs =  (SensorManager) getSystemService(Context.SENSOR_SERVICE);
Sensor orientation = gestionCapteurs.getDefaultSensor(Sensor.TYPE_ORIENTATION);
gestionCapteurs.registerListener(this, orientation, SensorManager.SENSOR_DELAY_NORMAL);
```

*Lorsque l'orientation change, la méthode **onSensorChanged()** de votre **SensorEventListener** est déclenchée. Le paramètre **SensorEvent** contient un tableau de **float** qui fournit l'orientation de l'appareil suivant les trois axes, respectivement l'azimut, le tangage et le roulis.*

res/layout/main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:padding="10dp"
    android:background="#FFFF00">
    <TextView
        android:id="@+id/azimut"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:textSize="23dp"
        android:textColor="#FF4400"/>
    <TextView
        android:id="@+id/tangage"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:textSize="23dp"
        android:textColor="#FF4400" />
```



```
    <TextView
        android:id="@+id/roulis"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:textSize="23dp"
        android:textColor="#FF4400"/>
</LinearLayout>
```

fr.btsiris.capteurs.Orientation.java


```
package fr.btsiris.orientation;

import android.app.Activity;
import android.content.Context;
import android.hardware.*;
import android.os.Bundle;
import android.widget.TextView;

public class Orientation extends Activity implements SensorEventListener {
    private SensorManager gestionCapteurs;
    private Sensor orientation;
    private TextView azimut, tangage, roulis;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        azimut = (TextView) findViewById(R.id.azimut);
        tangage = (TextView) findViewById(R.id.tangage);
        roulis = (TextView) findViewById(R.id.roulis);
        gestionCapteurs =  (SensorManager) getSystemService(Context.SENSOR_SERVICE);
        orientation = gestionCapteurs.getDefaultSensor(Sensor.TYPE_ORIENTATION);
    }

    @Override
    protected void onStart() {
        super.onStart();
        gestionCapteurs.registerListener(this, orientation, SensorManager.SENSOR_DELAY_NORMAL);
    }

    @Override
    protected void onStop() {
        super.onStop();
        gestionCapteurs.unregisterListener(this);
    }

    public void onSensorChanged(SensorEvent evt) {
        if (evt.sensor.getType() == Sensor.TYPE_ORIENTATION) {
            azimut.setText("Azimut ..... : "+evt.values[0]);
            tangage.setText("Tangage ... : "+evt.values[1]);
            roulis.setText("Roulis ..... : "+evt.values[2]);
        }
    }

    public void onAccuracyChanged(Sensor capteur, int precision) {  }
}
```

x DONNER LE CAP

Nous allons modifier le projet précédent pour prendre en compte uniquement l'azimut et de donner ainsi le cap suivant les points cardinaux avec une rose des vents complète.

res/layout/main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="#FFFF00">
```

```
    <TextView
        android:id="@+id/azimut"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:textSize="28sp"
        android:textStyle="italic|bold"
        android:textColor="#000000"
        android:gravity="center" />
    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:src="@drawable/rosedesvents" />
</LinearLayout>
```

fr.btsiris.capteurs.PointsCardinaux.java

```
package fr.btsiris.cardinaux;

import android.app.Activity;
...

public class PointsCardinaux extends Activity implements SensorEventListener {
    private SensorManager gestionCapteurs;
    private Sensor orientation;
    private TextView azimut;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        azimut = (TextView) findViewById(R.id.azimut);
        gestionCapteurs =  (SensorManager) getSystemService(Context.SENSOR_SERVICE);
        orientation = gestionCapteurs.getDefaultSensor(Sensor.TYPE_ORIENTATION);
    }

    @Override
    protected void onStart() {
        super.onStart();
        gestionCapteurs.registerListener(this, orientation, SensorManager.SENSOR_DELAY_FASTEST);
    }

    @Override
    protected void onStop() {
        super.onStop();
        gestionCapteurs.unregisterListener(this);
    }

    public void onSensorChanged(SensorEvent evt) {
        if (evt.sensor.getType() == Sensor.TYPE_ORIENTATION) {
            String cap = "";
            float x = evt.values[0];
            if (x<11.25 || x>=348.75) cap ="Nord";
            else if (x>11.25 && x<33.75) cap ="Nord-Nord-Est";
            else if (x>33.75 && x<56.25) cap ="Nord-Est";
            else if (x>56.25 && x<78.75) cap ="Est-Nord-Est";
            else if (x>78.75 && x<101.25) cap ="Est";
            else if (x>101.25 && x<123.75) cap ="Est-Sud-Est";
            else if (x>123.75 && x<146.25) cap ="Sud-Est";
            else if (x>146.25 && x<168.75) cap ="Sud-Sud-Est";
            else if (x>168.75 && x<191.25) cap ="Sud";
            else if (x>191.25 && x<213.75) cap ="Sud-Sud-Ouest";
            else if (x>213.75 && x<236.25) cap ="Sud-Ouest";
            else if (x>236.25 && x<258.75) cap ="Ouest-Sud-Ouest";
            else if (x>258.75 && x<281.25) cap ="Ouest";
            else if (x>281.25 && x<303.75) cap ="Ouest-Nord-Ouest";
            else if (x>303.75 && x<326.25) cap ="Nord-Ouest";
            else if (x>326.25 && x<348.75) cap ="Nord-Nord-Ouest";
            azimut.setText(cap);
        }
    }

    public void onAccuracyChanged(Sensor capteur, int precision) {  }
}
```

x DÉPLACEMENT D'UNE BOULE SUIVANT L'HORIZONTALITÉ DU MOBILE

Nous allons réutiliser l'accéléromètre avec cette fois-ci une boule que se déplace suivant l'horizontalité de votre smartphone. Pour ce projet Il est nécessaire de récupérer une image de fond et une image de la boule que vous placerez dans la ressource ***drawable***.

fr.btsiris.niveau.Niveau.java


```
package fr.btsiris.niveau;

import android.view.*;

public class Accelerometre extends Activity implements SensorEventListener {
    private SensorManager gestionCapteurs;
    private Sensor accelerometre;
    private SurfaceView surface;
    private SurfaceHolder holder;
    private Bitmap boule, fond;
    private float bx, by, vx, vy, tx, ty;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        boule = BitmapFactory.decodeResource(getResources(), R.drawable.boule);
        fond = BitmapFactory.decodeResource(getResources(), R.drawable.fond);
        surface = new SurfaceView(this);
        holder = surface.getHolder();
        gestionCapteurs =  (SensorManager) getSystemService(Context.SENSOR_SERVICE);
        accelerometre = gestionCapteurs.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
        setContentView(surface);
    }

    @Override
    protected void onStart() {
        super.onStart();
        gestionCapteurs.registerListener(this, accelerometre, SensorManager.SENSOR_DELAY_GAME);
    }
    @Override
    protected void onStop() {
        super.onStop();
        gestionCapteurs.unregisterListener(this, accelerometre);
    }

    public void onSensorChanged(SensorEvent evt) {
        if (evt.sensor.getType() == Sensor.TYPE_ACCELEROMETER) {

            // Ajout les valeurs du capteur aux variables
            vx -= evt.values[0];
            vy += evt.values[1];

            // Déplacement de la boule
            bx += vx;
            by += vy;

            // Récupération de la taille de l'écran
            tx = surface.getWidth() - boule.getWidth();
            ty = surface.getHeight() - boule.getHeight();

            // Empêcher la boule de sortir
            if (bx<0)  { bx = 0; vx = 0; }
            if (by<0)  { by = 0; vy = 0; }
            if (bx>tx) { bx = tx; vx = 0; }
            if (by>ty) { by = ty; vy = 0; }

            // Redessiner
            Canvas dessin = holder.lockCanvas();
            if (dessin==null) return;
            dessin.drawBitmap(fond, 0, 0, null);
            dessin.drawBitmap(boule, bx, by, null);
            holder.unlockCanvasAndPost(dessin);
        }
    }

    public void onAccuracyChanged(Sensor capteur, int precision) {  }
}
```