

# INFO0306 - Programmation mobile







# Sommaire

- Rappels
- La connectivité réseau
- Les capteurs
- La localisation





# Rappels

- Interface d'une applications Android
- Layout d'applications
  - Agent de placement
  - Fichiers XML
  - Exemples : *LinearLayout, RelativeLayout, TableLayout, ScrollView*
- Composants graphiques d'une application
  - **TextView**
  - **Button**
  - **ImageView & ImageButton**
  - **ETc.**





## La connectivité réseau







# La connectivité réseau

## Les permissions

- Comme pour tout accès dans Android :
  - il faut demander l'autorisation d'accès au réseau avant
  - dans le fichier : **AndroidManifest.xml**
  - Principales autorisations réseau :

```
<!-- Autoriser l'accès à Internet -->  
<uses-permission android:name="android.permission.INTERNET" />  
  
<!-- Autorise la vérification de l'état du réseau -->  
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```





# La connectivité réseau

## L'accès réseau (1/3)

- Avant de tenter une connexion réseau :
  - Il faut vérifier si une connexion réseau est disponible
  - Raisons d'absence de connexion :
    - Hors de portée d'un réseau mobile ou WIFI
    - L'utilisateur peut avoir désactivé le WIFI (Ex : Pour préserver la batterie)
    - L'utilisateur peut avoir désactivé les données mobile (Ex : Pour éviter un surcoût sur le forfait mobile).
    - Etc.
  - Il existe un gestionnaire de connexions réseau : **ConnectivityManager**
    - Récupération : avec la méthode **getSystemService()**
    - Comme tout le reste dans Android : les capteurs, le vibreur, l'alarme, la localisation, etc.

```
ConnectivityManager connectivityManager = (ConnectivityManager) getSystemService(CONNECTIVITY_SERVICE);
```





# La connectivité réseau

## L'accès réseau (2/3)

- La suite :
  - Appel à la méthode : **getActiveNetworkInfo()**
    - permet de récupérer un objet **NetworkInfo**
    - si objet **null** :
      - aucun réseau n'est disponible
      - annuler les accès réseau
      - prévenir l'utilisateur?

```
NetworkInfo networkInfo = connectivityManager.getActiveNetworkInfo();
```

- Sinon appel à la méthode : **getNetworkInfo()**
  - permet de vérifier l'état de chaque interface

```
NetworkInfo mobile = connectivityManager.getNetworkInfo(connectivityManager.TYPE_MOBILE);  
NetworkInfo wifi = connectivityManager.getNetworkInfo(connectivityManager.TYPE_WIFI);
```





# La connectivité réseau

## L'accès réseau (3/3)

- La suite :
  - On peut vérifier l'état du réseau maintenant :
    - **isConnected()** : réseau connecté
    - **isConnectedOrConnecting()** : réseau connecté ou en cours de connexion

```
boolean mobile_isConnected = mobile != null && mobile.isConnected();  
boolean wifi_isConnected = wifi != null && wifi.isConnected();
```

```
if (wifi_isConnected || mobile_isConnected ) {  
    // Disponible + Faire un travail  
} else {  
    // Non disponible + Prévenir l'utilisateur  
}
```





# La connectivité réseau

## Surveillance du réseau (1/3)

- Objectifs de la surveillance réseau :
  - Privilégier le réseau WIFI plutôt que le réseau mobile :
    - Réseau mobile : lent + coûteux
  - Arrêter les téléchargements en Mobile et poursuivre en WIFI
  - Arrêter car plus de réseau tout simplement
  - Etc.
- —> il faut surveiller les différents changements de connectivité :
  - Le système averti les applications :
    - inscrites aux changements d'état du réseau
    - évènement : **android.net.conn.CONNECTIVITY\_CHANGE**
  - On utilise pour cela le **BroadcastReceiver** :
    - il faut instancier un récepteur d'intentions
    - il faut l'inscrire à cet évènement





# La connectivité réseau

## Surveillance du réseau (2/3)

- 2 méthodes de souscription :

### 1. Dans le manifest :

```
<receiver
    android:name=".MyConnectionReceiver"
    android:enabled="true"
    android:exported="true">
    <intent-filter>
        <action android:name="android.net.conn.CONNECTIVITY_CHANGE" />
    </intent-filter>
</receiver>
```

### 2. Dynamiquement dans l'activité en JAVA :

```
@Override
protected void onResume() {
    super.onResume();
    MyConnectionReceiver receiver = new MyConnectionReceiver();
    IntentFilter intentFilter = new IntentFilter("android.net.conn.CONNECTIVITY_CHANGE");
    registerReceiver(receiver, intentFilter);
}

@Override
protected void onPause() {
    super.onPause();
    unregisterReceiver(receiver);
}
```





# La connectivité réseau

## Surveillance du réseau (3/3)

- Il reste à programmer le **BroadcastReceiver**

```
public class MyConnectionReceiver extends BroadcastReceiver {  
    @Override  
    public void onReceive(Context context, Intent intent) {  
        ConnectivityManager cm =  
            (ConnectivityManager) context.getSystemService(Context.CONNECTIVITY_SERVICE);  
        NetworkInfo activeNetwork = cm.getActiveNetworkInfo();  
        boolean isConnected = activeNetwork != null &&  
            activeNetwork.isConnectedOrConnecting();  
        if (isConnected) {  
            Toast.makeText(context, "Réseau connecté", Toast.LENGTH_LONG).show();  
        } else {  
            Toast.makeText(context, "Réseau changé ou reconnecté", Toast.LENGTH_LONG).show();  
        }  
    }  
}
```





# La connectivité réseau

## Les pages Web

- Charger du HTML :
  - Possible dans un **TextView**, mais limité (ex : pas d'image)
  - Il faut utiliser les **WebView** : gérées par **WebKit**
  - Avec : **loadDataWithBaseURL (String Url, String data, String mimeType, String encoding, String historyUrl)**

```
WebView webview = (WebView) findViewById(R.id.webview);  
webview.loadDataWithBaseURL(null, "<html>"  
    + "<body>La connectivité réseau</body></html>", "text/html",  
    "UTF-8", null);
```

- Charger une page Internet :
  - Il faut utiliser aussi les **WebView**
  - Avec : **loadURL (String url)**
  - Pensez bien à indiquer l'adresse complète avec les **http://** et les **www**

```
webview.loadUrl("https://www.android.com");
```





# La connectivité réseau

## Les Requêtes HTTP (1/3)

- Les clients **HTTP** dans Android :
  - **HttpClient** :
    - fourni par Apache
    - très utilisé pour les accès PC
    - adapté à Android par Apache
  - **HttpURLConnection** :
    - fourni par Android pour **Gingerbread** et plus (API 9, Android 2.3)
    - un client léger et adapté pour le mobile
    - privilégié par Android
- Les requêtes sur le réseau :
  - peuvent être longues (téléchargement fichier multimédia)
  - peuvent entraîner des retards imprévisibles
  - éviter de bloquer l'interface utilisateur (ANR)
  - il est conseillé de toujours effectuer ces opérations dans un thread séparé





# La connectivité réseau

## Les Requêtes HTTP (2/3)

- Etapes de connexion et de chargement :
  1. transformer la chaîne de caractère en objet URL

```
URL url = new URL("https://www.android.com");
```

### 2. ouvrir une connexion dessus avec **openConnection()**

```
URLConnection conn = (URLConnection) url.openConnection();
```

### 3. vérifier si le serveur est accessible

```
if (conn.getResponseCode() == HttpURLConnection.HTTP_OK) {  
    // Chargement infos  
}
```

### 4. récupérer le contenu du flux

```
InputStream is = conn.getInputStream();
```





# La connectivité réseau

## Les Requêtes HTTP (3/3)

- Les méthodes de la classe ***HttpURLConnection*** :

Méthodes	Description
<b>setConnectTimeout(int)</b>	<ul style="list-style-type: none"> <li>- Fixe le temps maximum de la connexion</li> <li>- Exprimé en milliseconde</li> </ul>
<b>setReadTimeout(int)</b>	<ul style="list-style-type: none"> <li>- Fixe le temps maximum de chargement de la page</li> <li>- Exprimé en milliseconde</li> </ul>
<b>setUsesCaches (boolean)</b>	<ul style="list-style-type: none"> <li>- Usage ou non du cache</li> </ul>
<b>setRequestMode(String)</b>	<ul style="list-style-type: none"> <li>- Fixe la méthode HTTP à utiliser</li> <li>- GET/POST</li> </ul>
<b>setDoOutput(boolean)</b>	<ul style="list-style-type: none"> <li>- Autorise ou non les flux sortants</li> <li>- Utile quand on souhaite envoyer des informations vers un serveur.</li> </ul>





## Les capteurs







# Les capteurs

## Diversité des capteurs

- Un grand nombre d'appareils Android disposent de capteurs :
  - Mouvement
  - Orientation
  - Conditions environnementales
- Des informations avec une grande précision, proposer aux utilisateurs des :
  - nouvelles possibilités
  - de nouveaux services
- Capteurs divers et variés :
  - des accéléromètres
  - des gyroscopes
  - des capteurs de luminosité
  - des capteurs de température
  - des capteurs de pression
  - détecteur d'empreinte digitale, Etc.





# Les capteurs

## Prise en charge des capteurs

- Framework des capteurs dans Android :
  - Contient :
    - plusieurs classes
    - plusieurs interfaces
  - Permet :
    - détecter la présence ou non d'un capteur
    - reconnaître ses capacités
    - recueillir ses données
- Types de capteurs :
  - capteurs matériels : physiquement présents sur l'appareil
  - capteurs logiciels (virtuels) : calculent leurs valeurs en fonction d'un ou plusieurs autres capteurs





# Les capteurs

## Liste des capteurs

Nom	Description et usage	Type
<b>Accéléromètre</b>	<ul style="list-style-type: none"> <li>-TYPE_ACCELEROMETER</li> <li>-Mesure la force d'accélération sur les 3 axes (x,y,z).</li> <li>-Capable d'en déduire la force de gravitation en m/s<sup>2</sup>.</li> <li>-Détecter les inclinaisons (et la ligne d'alignement, etc.)</li> </ul>	Matériel
<b>Gyroscope</b>	<ul style="list-style-type: none"> <li>-TYPE_GYROSCOPE</li> <li>-Mesure la rotation sur les 3 axes (x, y, z) en rad/s.</li> <li>-Détecter l'orientation de l'appareil.</li> </ul>	Matériel
<b>Orientation</b>	<ul style="list-style-type: none"> <li>-TYPE_ORIENTATION</li> <li>-Mesure la rotation sur les 3 axes (x, y, z) en degrés.</li> <li>-Préférer l'usage de getOrientation() depuis API 8.</li> <li>-Détecter l'orientation de l'appareil.</li> </ul>	Logiciel
<b>Proximité</b>	<ul style="list-style-type: none"> <li>-TYPE_PROXIMITY</li> <li>-Mesure la proximité d'un objet en cm.</li> <li>-Détecter si l'utilisateur porte l'appareil à son oreille.</li> </ul>	Matériel
<b>Magnétomètre</b>	<ul style="list-style-type: none"> <li>-TYPE_MAGNETIC_FIELD</li> <li>-Mesure le champ géomagnétique sur les 3 axes en micro-telsa (μT).</li> <li>-Création d'une boussole</li> </ul>	Matériel
<b>Photomètre</b>	<ul style="list-style-type: none"> <li>-TYPE_LIGHT</li> <li>-Mesure le niveau de lumière ambiante en lux.</li> <li>-Contrôler la luminosité de l'écran.</li> </ul>	Matériel
<b>Baromètre</b>	<ul style="list-style-type: none"> <li>-TYPE_PRESSURE</li> <li>-Mesure la pression de l'air ambiant.</li> <li>-Surveiller le changement de pression atmosphérique.</li> </ul>	Matériel
<b>Thermomètre</b>	<ul style="list-style-type: none"> <li>-TYPE_TEMPERATURE / TYPE_AMBIANT_TEMPERATURE (&gt;API 14)</li> <li>-Mesure la température en degré Celsius (°C).</li> <li>-Contrôler la température.</li> </ul>	Matériel
<b>Tout</b>	<ul style="list-style-type: none"> <li>-TYPE_ALL</li> <li>-Liste : <a href="http://developer.android.com/reference/android/hardware/Sensor.html">http://developer.android.com/reference/android/hardware/Sensor.html</a></li> </ul>	Matériel Logiciel





# Les capteurs

## Filtre Google Play Store

- Lors de la mise en ligne de l'application :
  - filtrer son affichage uniquement en présence de certains capteurs nécessaire à son bon fonctionnement
  - on utilise la balise **<uses-feature>** dans le manifest
  - exemple :

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

- Les attributs :
  - name :
    - spécifie le nom du capteur
    - Liste dans le manifest : accéléromètre, baromètre, magnétomètre, gyroscope, photomètre et proximité
  - required :
    - spécifie la présence du capteur est indisponible ou pas
    - sinon : il faut désactiver les parties du capteur si non dispo





# Les capteurs

## Le Framework des capteurs

- Framework des capteurs dans Android fournit :
  - L'identification des capteurs et des capacités de détection
  - Surveiller les évènements des capteurs

Classe/interface	Description
<b>SensorManager</b>	<ul style="list-style-type: none"> <li>-Création de l'instance du service du capteur</li> <li>-Accès aux capteurs, listings des capteurs</li> <li>-Constante spécifiant la précision du capteur et l'étalonnage</li> <li>-Permet l'ajout ou la suppression d'un écouteur d'événements.</li> </ul>
<b>Sensor</b>	<ul style="list-style-type: none"> <li>-Création d'un capteur spécifique (en fonction du capteur)</li> <li>-Permet de déterminer les capacités du capteur.</li> </ul>
<b>SensorEvent</b>	<ul style="list-style-type: none"> <li>-Création d'un événement de capteur</li> <li>-Informations sur un événement de capteur</li> <li>-Utiliser par le système pour publier les données de capteur</li> <li>-Fourni les données des capteurs, le type de capteur ayant généré l'événement, la précision des données et l'horodatage de l'événement.</li> </ul>
<b>SensorEventListener</b>	<ul style="list-style-type: none"> <li>-Interface disposant de deux méthodes de rappels                             <ul style="list-style-type: none"> <li>✓ Une pour détecter le changement de précision</li> <li>✓ Une pour détecter le changement de valeurs</li> </ul> </li> </ul>





# Les capteurs

Dispo

Capteur	Android 4.0	Android 2.3	Android 2.2	Android 1.5
TYPE_ACCELEROMETER	Oui	Oui	Oui	Oui
TYPE_AMBIENT_TEMPERATURE	Oui	Non	Non	Non
TYPE_GRAVITY	Oui	Oui	Non	Non
TYPE_GYROSCOPE	Oui	Oui	Non <sup>2</sup>	Non <sup>2</sup>
TYPE_LIGHT	Oui	Oui	Oui	Oui
TYPE_LINEAR_ACCELERATION	Oui	Oui	Non	Non
TYPE_MAGNETIC_FIELD	Oui	Oui	Oui	Oui
TYPE_ORIENTATION	Oui <sup>1</sup>	Oui <sup>1</sup>	Oui <sup>1</sup>	Oui
TYPE_PRESSURE	Oui	Oui	Non <sup>2</sup>	Non <sup>2</sup>
TYPE_PROXIMITY	Oui	Oui	Oui	Oui
TYPE_RELATIVE_HUMIDITY	Oui	Non	Non	Non
TYPE_ROTATION_VECTOR	Oui	Oui	Non	Non
TYPE_TEMPERATURE	Oui <sup>1</sup>	Oui	Oui	Oui

## Légende

Dispo

Non Dispo

Dispo  
déprécié

Dispo 1.5  
utilisable 2.3





# Les capteurs

## Identifier les capteurs

### 1. Récupérer le manager des capteurs

```
sensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);
```

### 2. Récupérer la liste des capteurs ou un capteur en particulier

```
List<Sensor> sensorList = sensorManager.getSensorList(Sensor.TYPE_ALL);  
Sensor accelerationSensor = sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
```

### 3. vérifier si le capteur est disponible

```
if (accelerationSensor!=null) {  
    // un accéléromètre est disponible  
}  
else{  
    // aucun accéléromètre n'est disponible  
}
```





# Les capteurs

## Surveiller les capteurs (1/3)

- Afin de surveiller les capteurs :
  - On utilise l'interface : **SensorEventListener**
    - Méthodes :
      - **onAccuracyChanged()**
      - **onSensorChanged()**

Méthode	Description
<b>onAccuracyChanged (Sensor, int)</b>	-Appelée quand la précision du capteur change. - <i>Sensor</i> : Capteur - <i>int</i> : précision du capteur
<b>onSensorChanged (SensorEvent)</b>	-Appelé quand il y a un événement sur le capteur - <i>SensorEvent</i> : un événement <ul style="list-style-type: none"> <li>➡ <i>accuracy</i> pour indiquer la précision de la mesure</li> <li>➡ <i>sensor</i> comme référence au capteur ayant pris la mesure</li> <li>➡ <i>timestamp</i> pour l'instant de prise de mesure</li> <li>➡ <i>values</i>, la plus importante, pour les valeurs (tableau d'entiers)</li> </ul>





# Les capteurs

## Surveiller les capteurs (2/3)

- La précision du capteur peut prendre :
  - **`SensorManager.SENSOR_STATUS_ACCURACY_LOW`** : Faible précision.
  - **`SensorManager.SENSOR_STATUS_ACCURACY_MEDIUM`** : Précision moyenne.
  - **`SensorManager.SENSOR_STATUS_ACCURACY_HIGH`** : Précision maximale.
  - **`SensorManager.SENSOR_STATUS_ACCURACY_UNRELIABLE`** : il ne faut pas faire confiance à ce capteur par manque de fiabilité des données ou dû à un mauvais étalonnage par exemple on utilisera.
- Il faut déclarer quel capteur nous voulons surveiller :

```
@Override
protected void onResume() {
    super.onResume();
    sensorManager.registerListener(accelerationEventListener, accelerationSensor,
    SensorManager.SENSOR_DELAY_UI);
}

@Override
protected void onPause() {
    super.onPause();
    sensorManager.unregisterListener(accelerationEventListener);
}
```





# Les capteurs

## Surveiller les capteurs (3/3)

- Délai de rafraîchissement :
  - Entier qui peut prendre ces valeurs :

Constante	Description
<b>SENSOR_DELAY_NORMAL</b>	-Rafraichissement normal -0,2 seconde
<b>SENSOR_DELAY_UI</b>	-Rafraichissement lent -0,6 seconde -Convient bien aux interfaces utilisateurs
<b>SENSOR_DELAY_GAME</b>	-Rafraichissement rapide -0,02 seconde -Convient bien aux jeux
<b>SENSOR_DELAY_FASTEST</b>	-Rafraichissement le plus rapide possible -0 seconde -Consommatrice d'énergie





# Les capteurs

## Système de coordonnées des capteurs (1/2)

- Un **SensorEvent** indique les données des capteurs :
  - un système de coordonnées standard à 3 axes, où **X**, **Y** et **Z** sont représentés par **values[0]**, **values[1]** et **values[2]**
  - Pour certains capteurs ne délivrant qu'une seule information :
    - seule **values[0]** est utilisée
    - exemple : lumière, température, proximité, pression, etc.

```
private SensorEventListener accelerationEventListener = new SensorEventListener() {  
    @Override  
    public void onSensorChanged(SensorEvent event) {  
        float[] values = event.values;  
        float Ax = values[0];  
        float Ay = values[1];  
        float Az = values[2];  
        // traiter les données  
    }  
    @Override  
    public void onAccuracyChanged(Sensor sensor, int accuracy) {  
    }  
};
```

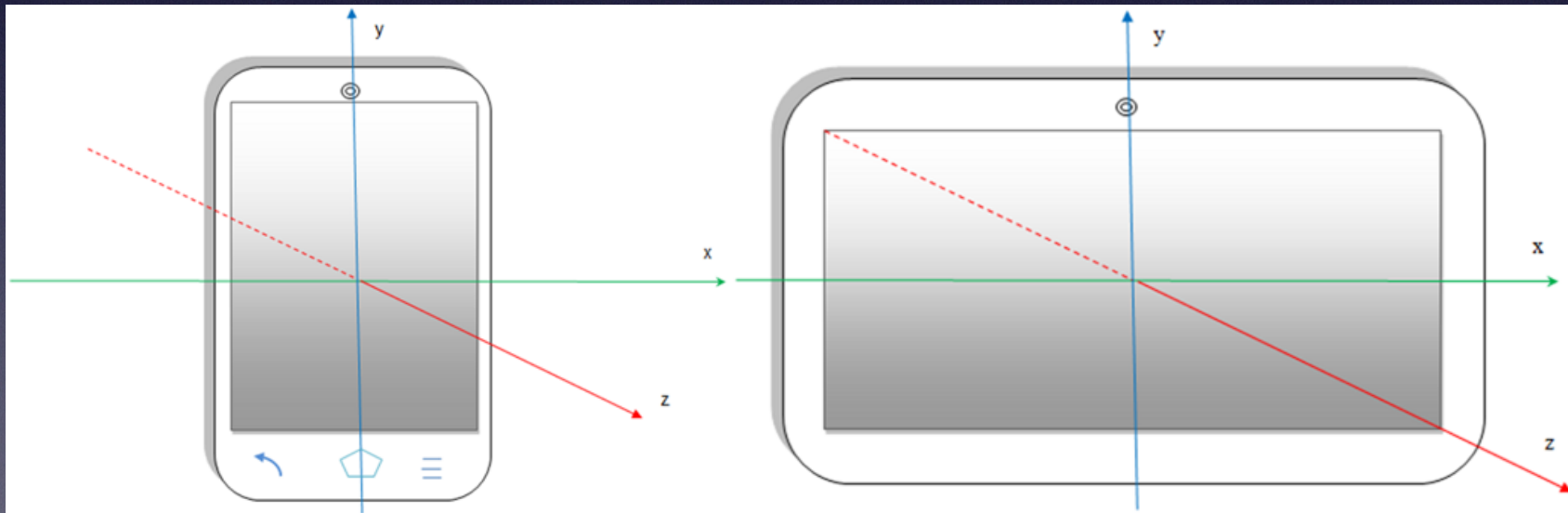




# Les capteurs

## Système de coordonnées des capteurs (2/2)

- Orientation :
  - Smartphone : mode portrait
  - Tablette : mode paysage



- Attention : le système des coordonnées du capteurs ne change jamais même lorsque l'appareil bouge ou change d'orientation!

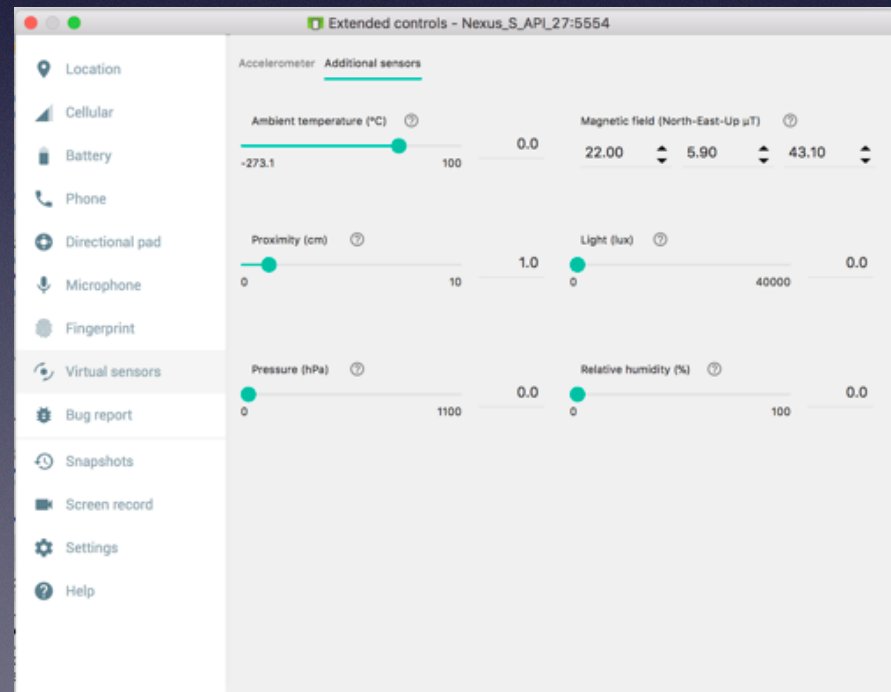
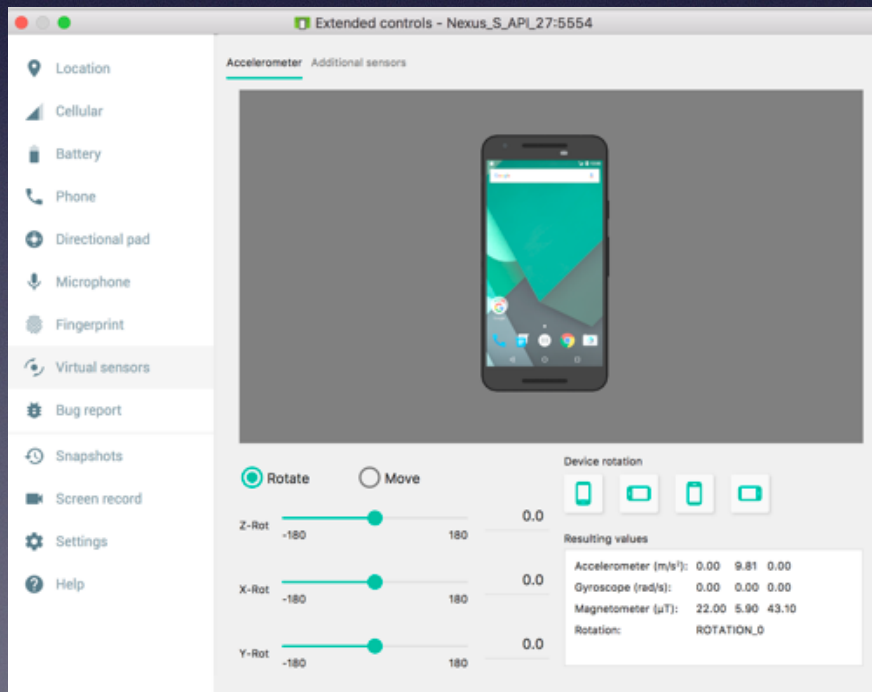




# Les capteurs

## Tester les capteurs

- Tests :
  - Smartphones réels
  - Émulateurs : pas possible avant sur Eclipse, mais possible sur Android Studio







# Les capteurs

## Les bonnes pratiques

- Se désinscrire :
  - de la surveillance d'un ou de plusieurs capteurs
  - quand plus besoin
  - avec la méthode ***unregisterListener()***
  - souvent dans la méthode ***onPause()***
- Vérifier les capteurs :
  - toujours vérifier la présence des capteurs !
  - même avec la mise en œuvre d'un filtre Google Play
- Rafraichissement :
  - bien choisir le type de délai correspondant à votre application
  - sinon consommation excessive de la batterie...





## La localisation







# La localisation

## Généralités

- La localisation ou plutôt géolocalisation :
  - un procédé permettant de positionner un objet sur une carte ou sur un plan
  - basé sur les coordonnées géographiques
  - peut enrichir grandement une application :
    - application intelligente
    - offrir à vos utilisateurs des informations adaptées en fonction de la position de l'appareil Android.
  - la solution la plus efficace est l'usage du GPS :
    - il est également possible de déterminer une position à l'aide du WIFI
    - ou d'un réseau GSM





# La localisation

## Les permissions

- Deux types de permissions :
  - géolocalisation par GPS (**GPS\_PROVIDER**)
  - une localisation moins précise par le WIFI et les antennes relais (**NETWORK\_PROVIDER**)
  - la première englobe la deuxième

```
<!-- Autorise la géolocalisation à partir du GPS -->  
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />  
  
<!-- Autorise la géolocalisation à partir du Cellulaire et du Wifi -->  
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
```





# La localisation

## Mise à jour de la position (1/2)

### 1. Récupérer le manager de la localisation

```
locationManager = (LocationManager) getSystemService(Context.LOCATION_SERVICE);
```

### 2. Préparer le LocationListener

```
private LocationListener locationManager = new LocationListener() {  
    @Override  
    public void onLocationChanged(Location location) {  
        float lat = (float) (location.getLatitude());  
        float lng = (float) (location.getLongitude());  
    }  
    @Override  
    public void onStatusChanged(String provider, int status, Bundle extras) {  
    }  
    @Override  
    public void onProviderEnabled(String provider) {  
    }  
    @Override  
    public void onProviderDisabled(String provider) {  
    }  
};
```





# La localisation

## Mise à jour de la position (2/2)

### 3. s'enregistrer pour récupérer les mises à jour de la localisation

```
@Override  
protected void onResume() {  
    super.onResume();  
    locationManager.requestLocationUpdates(provider, 1000 * 60 * 5, 1000 * 2, locationManagerListener);  
}
```

### 4. se désinscrire si plus besoin de la position

```
@Override  
protected void onPause() {  
    super.onPause();  
    locationManager.removeUpdates(locationManagerListener);  
}
```





# La localisation

## Les paramètres de la méthode ***requestLocationUpdates***

Paramètres	Description
<b>String provider</b>	<ul style="list-style-type: none"> <li>–LocationManager.GPS_PROVIDER</li> <li>–LocationManager.NETWORK_PROVIDER</li> </ul>
<b>Long minTime</b>	<ul style="list-style-type: none"> <li>–Intervalle minimum entre 2 mises à jour</li> <li>–Exprimé en millisecondes.</li> <li>–Attention : plus la fréquence est basse plus la consommation d'énergie est importante.</li> </ul>
<b>Float minDistance</b>	<ul style="list-style-type: none"> <li>–Distance minimale entre 2 mises à jour</li> <li>–Exprimée en mètres.</li> <li>–Attention : plus la distance est courte plus la consommation d'énergie est importante.</li> </ul>
<b>LocationListener listener</b>	<ul style="list-style-type: none"> <li>–L'écouteur (listener) appelé à chaque mise à jour d'emplacement.</li> </ul>





# La localisation

## Les bonnes pratiques (1/2)

- Quand commencer l'écoute ?
  - Problématique :
    - Une longue écoute du GPS consommera beaucoup de batterie
    - Une courte période d'écoute du GPS manquera de précision
  - Durée d'écoute dépend du type d'application (navigation Vs. Store Locator)
  - Choix de **minTime** et **minDistance** a un impact très important sur la consommation de la batterie
  - Donc compromis entre précision et consommation!
  - En général, démarrage dans **onResume()**
- Quand arrêter l'écoute ?
  - Dès que possible (plus besoin)
  - En général, arrêt dans **onPause()**





# La localisation

## Les bonnes pratiques (2/2)

- Une solution rapide :
  - récupérer dernière position connue même si géolocalisation à l'arrêt
  - position disponible dans le cache récupérée grâce à ***getLastKnownLocation()***
  - plus rapide

```
Location location = locationManager.getLastKnownLocation(provider);
```





# La localisation

## Les alertes de proximités (1/2)

- Description :
  - Les alertes de proximités vous permettent d'avertir les utilisateurs dès qu'ils entrent ou sortent d'une zone
  - Cette zone sera définie par un cercle dont on doit préciser son centre et son rayon



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





# La localisation

## Les alertes de proximités (2/2)

Paramètres	Description
<b>Double latitude</b>	-Latitude du centre
<b>Double longitude</b>	-Longitude du centre
<b>Float radius</b>	-Rayon de la zone (du cercle) -Exprimé en mètre
<b>Long expiration</b>	-Durée pour laquelle l'alerte est valable. -Exprimé en milliseconde -Valeur négative = aucune expiration.
<b>PendingIntent intent</b>	-Le pendingIntent qui sera lancé quand l'alerte sera déclenchée.

- Une alerte de proximité est envoyée dès lors que l'utilisateur entre ou sort d'une zone uniquement (même au démarrage)
- Pour limiter la consommation de la batterie, il est, conseillé de fixer une durée de validité





# La localisation

The image shows a composite of two screenshots. On the left is an Android phone home screen with a Google search bar, a dock with icons for Google, NFA025, and Settings, and a bottom dock with Phone, Messages, App Drawer, Browser, and Camera. On the right is a PC window titled 'Extended controls' showing GPS data. The 'Location' menu is open on the phone. The PC window displays the following data:

GPS data point

Coordinate system: Decimal

Longitude: -122.084

Latitude: 37.422

Altitude (meters): 0.0

Currently reported location

Longitude: -122.0840

Latitude: 37.4220

Altitude: 0.0

GPS data playback

Delay (sec)	Latitude	Longitude	Elevation	Name	Description
-------------	----------	-----------	-----------	------	-------------

Speed 1X

SEND

LOAD GPX/KML





# La localisation

## Tester la localisation (2/2)

- Comme pour les capteurs :
  - Smartphones réels
  - Émulateurs : 2 méthodes
    1. Android Studio
      - ...
    2. Telnet
      - En utilisant la ligne de commande, c'est plus drôle !
      - Connectez-vous à l'appareil avec :  
`$ telnet localhost 5554`
      - Puis envoyez vos coordonnées avec :  
`$ geo fix <latitude> <longitude>`
      - Exemple :  
`$ geo fix 49.244178 4.058911`
      - La commande retourne ok en cas de réussite et ko en cas d'échec





# La connectivité réseau / Les capteurs / La localisation

## Post-it

- Pour afficher une page web ou effectuer une requête HTTP, il est indispensable de demander l'autorisation à INTERNET dans le manifest
- Une bonne pratique reste à vérifier l'état de la connexion et d'avertir l'utilisateur.
- Une requête doit toujours charger ses informations dans un thread séparé afin d'éviter les ANR
- Le manager des capteurs facilite beaucoup leur utilisation dans Android
- Il existe des capteurs physiques et d'autres virtuels
- Il faut vérifier toujours la présence ou non d'un capteur avant de l'utiliser
- Il est possible de surveiller et de récupérer les données capteurs via le service de gestion des capteurs
- Les appareils Android peuvent déterminer leur position avec plus ou moins de précision
- La géolocalisation consomme beaucoup d'énergie et a un impact important sur l'autonomie de la batterie
- Il est conseillé de :
  - ✓ maximiser l'intervalle de temps et de distance entre deux mises à jour de la position.
  - ✓ utiliser la dernière position connue sans réactiver le GPS.
  - ✓ stopper l'écoute dès qu'on en a plus besoin.