



Lezione 18



Programmazione Android



- **Interfaccia col mondo**
 - Accesso ai sensori
 - Servizi di localizzazione
 - Video e audio



Sviluppo Applicazioni Mobili
Vincenzo Gervasi – a.a. 2012/13

Accesso ai sensori



Gestione dei sensori



- Android implementa un sistema di gestione dei sensori del tutto **generico**
 - Pronto per essere esteso a tipi di sensori diversi
 - Meccanismi simili, valori **da interpretare**
- Servizio di sistema: **SensorManager**

```
SensorManager sm =
```

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

- Come per altri servizi: una volta ottenuto un handle, si possono chiamare metodi



Sensor discovery



- Per recuperare l'elenco di tutti i sensori presenti sul dispositivo:

```
List<Sensor> list = sm.getSensorList(Sensor.TYPE_ALL) ;
```

- Ogni Sensor descrive un particolare sensore
 - Type
 - Name, vendor, version
 - Max range, min delay, resolution, power
- Si può avere più di un sensore per tipo
 - Uno è quello usato per default



Sensor discovery



- Per ottenere il sensore di default (quello “standard”) per un certo tipo:

```
Sensor sens = sm.getDefaultSensor(tipo);
```

- Per ottenere la lista di tutti i sensori di un certo tipo:

```
List<Sensor> list = sm.getSensorList(tipo);
```



Tipi di sensori



TYPE_ACCELEROMETER	A constant describing an accelerometer sensor type.
TYPE_AMBIENT_TEMPERATURE	A constant describing an ambient temperature sensor type
TYPE_GRAVITY	A constant describing a gravity sensor type.
TYPE_GYROSCOPE	A constant describing a gyroscope sensor type
TYPE_LIGHT	A constant describing a light sensor type.
TYPE_LINEAR_ACCELERATION	A constant describing a linear acceleration sensor type.
TYPE_MAGNETIC_FIELD	A constant describing a magnetic field sensor type.
TYPE_ORIENTATION	<i>This constant was deprecated in API level 8. use <code>SensorManager.getOrientation()</code> instead.</i>
TYPE_PRESSURE	A constant describing a pressure sensor type
TYPE_PROXIMITY	A constant describing a proximity sensor type.
TYPE_RELATIVE_HUMIDITY	A constant describing a relative humidity sensor type.
TYPE_ROTATION_VECTOR	A constant describing a rotation vector sensor type.
TYPE_TEMPERATURE	<i>This constant was deprecated in API level 14. use <code>Sensor.TYPE_AMBIENT_TEMPERATURE</code> instead.</i>



Tipi di sensori



- Alcuni di questi sensori saranno veri **sensori hardware**
- Altri saranno **sensori software**
 - Utilizzano i sensori hardware per ottenere informazioni
 - Esempi:
 - Integrando un sensore di accelerazione, si può ottenere un sensore di velocità
 - Con un filtro passa-basso sull'accelerazione, si può determinare la forza di gravità
- Non occorre distinguerli (dipende dai dispositivi)



Leggere i sensori



- Come è di regola su Android, è il sistema a chiamare il nostro codice, non viceversa
- Si registra un listener (indicando anche con che frequenza vogliamo essere chiamati)
- Il `SensorManager` chiamerà poi i nostri listener
 - *Forse* con la frequenza richiesta, più o meno...
 - In genere, con frequenza non minore di quella richiesta
 - Ma si sa, il multitasking... lo scheduler... il garbage collector... arriva una chiamata... gli alieni... le cavallette...



Casi di callback



- **SensorEventListener**

- **onSensorChanged()**

- Cambia il valore letto dal sensore
- Per esempio, leggo la bussola e il telefono viene ruotato

- **onAccuracyChanged()**

- Cambia l'accuracy del sensore
- Per esempio, passo dalla localizzazione GPS a quella radio
 - Psst... il GPS non è letto come un sensore, è solo un esempio!

- È bene registrare il listener nella **onResume()** e de-registrarlo nella **onPause()**



Registrazione & Deregistrazione



```
boolean success = sm.registerListener(  
    SensorEventListener l,  
    Sensor sens,  
    int rate  
);
```

- Registra **l** per essere informato degli eventi relativi a **sens**, con una frequenza di circa **rate**
 - **rate** è espresso in microsecondi (1M = 1 sec)
 - SENSOR_DELAY_NORMAL = 200.000 μ s = 5 per sec
 - SENSOR_DELAY_GAME = 20.000 μ s = 50 per sec
 - SENSOR_DELAY_UI = 60.000 μ s = 16.7 per sec
 - SENSOR_DELAY_FASTEST = 0 μ s = max possibile

3.0+





Registrazione & Deregistrazione



```
boolean success = sm.registerListener(  
    SensorEventListener l,  
    Sensor sens,  
    int rate  
    );
```

- Registra **l** per essere informato degli eventi relativi a **sens**, con una frequenza di circa **rate**
 - **l** può essere un listener creato ad-hoc
 - Anonymous inner class
 - Lo stesso listener può essere registrato su più sensori
 - Come al solito, l'Activity stessa può essere il listener



Registrazione & Deregistrazione



- Per deregistrare un listener:
 - Da uno specifico sensore
`sm.unregisterListener(l, sens) ;`
 - Da tutti i sensori su cui è registrato
`sm.unregisterListener(l) ;`
- Deregistrare i listener quando non state usando i sensori è critico
 - Altrimenti, la CPU non va ma in sleep
 - Batteria esaurita prima che ve ne accorgiate...



SensorEventListener



```
package android.hardware;  
  
public interface SensorEventListener {  
    public void onSensorChanged(SensorEvent event);  
    public void onAccuracyChanged(Sensor sens, int acc);  
}
```

- L'interfaccia del listener offre metodi per ricevere gli **eventi** (letture di valori) e le variazioni di **accuracy**
- Gli **eventi** sono codificati come SensorEvent
 - Contengono un campo **accuracy**...
- **L'accuracy** è codificata da valori discreti



L'accuracy



- **Non** si tratta di un valore di accuratezza del valore
 - Per quello c'è `sens.getResolution()`
- È un'indicazione relativa allo stato del sensore:

<code>SENSOR_STATUS_ACCURACY_HIGH</code>	This sensor is reporting data with maximum accuracy
<code>SENSOR_STATUS_ACCURACY_LOW</code>	This sensor is reporting data with low accuracy, calibration with the environment is needed
<code>SENSOR_STATUS_ACCURACY_MEDIUM</code>	This sensor is reporting data with an average level of accuracy, calibration with the environment may improve the readings
<code>SENSOR_STATUS_UNRELIABLE</code>	The values returned by this sensor cannot be trusted, calibration is needed or the environment doesn't allow readings



L'accuracy



- Ogni singola lettura di valore contiene anche l'accuracy che il sensore aveva in quel momento
- Spesso `onAccuracyChanged()` non serve
 - Quando serve, è per invitare l'utente a iniziare un processo di ri-calibrazione
 - Più spesso, si da un'implementazione vuota
- Implementarla è comunque obbligatorio!



I SensorEvent



```
package android.hardware;  
  
public class SensorEvent {  
    public final float[] values;  
    public Sensor sensor;  
    public int accuracy;  
    public long timestamp;  
  
    SensorEvent(int size) {  
        values = new float[size];  
    }  
}
```

- Un SensorEvent incapsula:
 - Un numero **variabile** di valori float che rappresentano la lettura del sensore
 - Il sensore da cui vengono i valori
 - L'accuratezza del sensore al momento della lettura
 - L'istante della lettura

I valori degli eventi



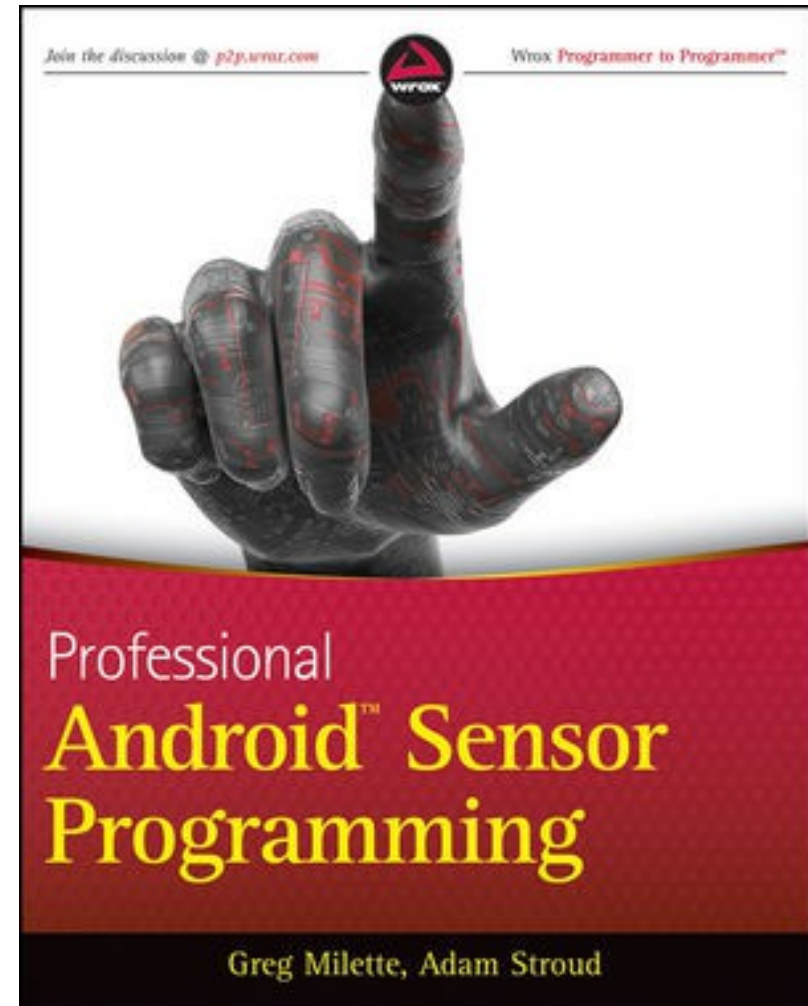
- L'*interpretazione* di **values** dipende dal sensore
- Esempi:
 - TYPE_AMBIENT_TEMPERATURE
 - **values**[0] = temperatura ambiente in °C
 - TYPE_MAGNETIC_FIELD
 - **values**[0] = componente x del campo magnetico in μT
 - **values**[1] = componente y del campo magnetico in μT
 - **values**[2] = componente z del campo magnetico in μT
- Indispensabile rifarsi alla documentazione



Sensori posizionali e di movimento



- In particolare, la gestione di coordinate, velocità e accelerazioni 3D è **complicata**
 - Diversi sistemi di riferimento
 - Il dispositivo può essere a sua volta ruotato, o tenuto in mano in modi diversi, o può essere a bordo di un veicolo in movimento, o su un aereo in picchiata...



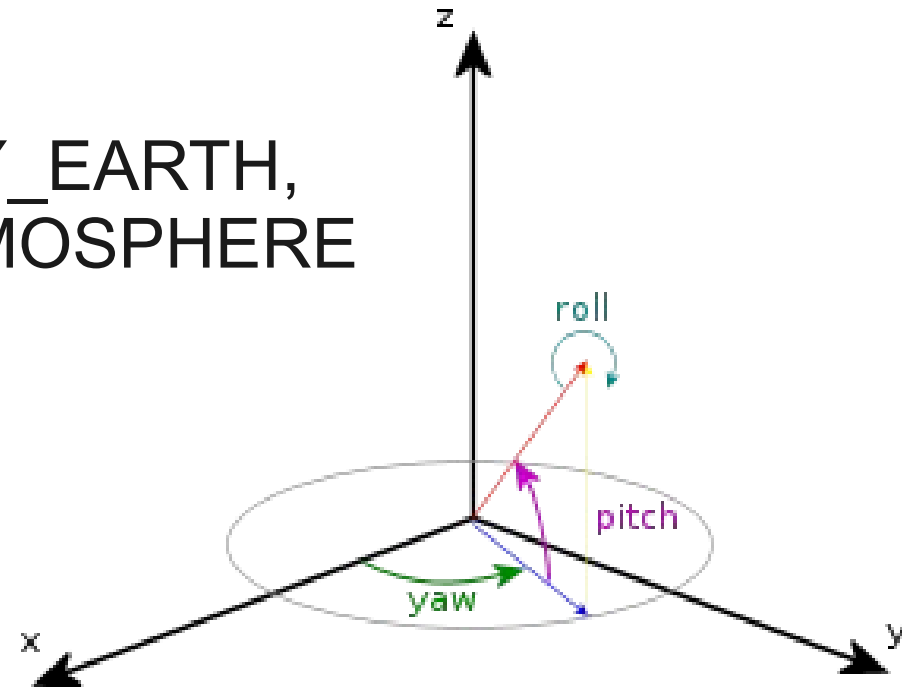


Sensori posizionali e di movimento



Sviluppo Applicazioni Mobili
Vincenzo Gervasi – a.a. 2012/13

- La classe `SensorManager` fornisce un certo numero di metodi e costanti di utilità
 - `getInclination()`, `getOrientation()`,
`getQuaternionFromVector()`,
`getRotationMatrixFromVector()`
 - `LIGHT_FULLMOON`, `GRAVITY_EARTH`,
`PRESSURE_STANDARD_ATMOSPHERE`
- E altri di inutilità...
 - `SENSOR_TRICORDER`,
`GRAVITY_SATURN`,
`GRAVITY_DEATH_START`





Cautele varie



- In generale
 - + precisione → + batteria
 - + frequenza → + batteria
- Mai lasciare i listener registrati più del dovuto
- Considerare approcci adattivi
 - Leggo poco e male quando non serve, aumento frequenza e precisione quando serve
- Non dedicarsi a operazioni lente nei metodi on...()
 - Semmai, memorizzare il valore in una struttura dati e processarlo su un altro thread
 - Sincronizzazione!
 - Il thread di processing può avere un suo tempo di ciclo, diverso da quello di lettura del sensore



Cautele varie



- Il `SensorEvent` passato al listener rimane di proprietà del `SensorManager`
 - Mai tenere dei riferimenti all'evento che restano anche dopo il ritorno da `onSensorChanged()`
- Il `SensorManager` potrebbe usare un *pool* di `SensorEvent`
 - Per evitare di fare una **new** per ogni lettura
 - In questo caso, gli stessi oggetti vengono riciclati più volte



Cautele varie



- Il `SensorEvent` passato al listener rimane di proprietà del `SensorManager`
 - Mai tenere dei riferimenti all'evento che restano anche dopo il ritorno da `onSensorChanged()`
- Il `SensorManager` potrebbe usare un *pool* di `SensorEvent`
 - Per evitare di fare una **new** per ogni lettura
 - In questo caso, gli stessi oggetti vengono riciclati più volte



Cautele varie



```
SensorEvent getFromPool() {
    SensorEvent t = null;
    synchronized (this) {
        if (mNumItemsInPool > 0) {
            // remove the "top" item from the pool
            final int index = mPoolSize - mNumItemsInPool;
            t = mPool[index];
            mPool[index] = null;
            mNumItemsInPool--;
        }
    }
    if (t == null) {
        // the pool was empty or this item
        // was removed from the pool for
        // the first time. In any case,
        // we need to create a new item.
        t = createSensorEvent();
    }
    return t;
}
```

protected static final class SensorEventPool
(dentro SensorManager)

```
void returnToPool(SensorEvent t) {
    synchronized (this) {
        // is there space left in the pool?
        if (mNumItemsInPool < mPoolSize) {
            // if so, return the item to the pool
            mNumItemsInPool++;
            final int index = mPoolSize - mNumItemsInPool;
            mPool[index] = t;
        }
    }
}
```