

# Sensori

Lezione 13

# Categorie di sensori

- Motion sensors (sensori di movimento)
- Environmental sensors (sensori ambientali)
- Position sensors (sensori di posizione)



# Sensori di movimento

---

- Questi sensori misurano le forze di accelerazione e le forze di rotazione lungo tre assi. Questa categoria include accelerometri, sensori di gravità, giroscopi e sensori vettoriali rotazionali
  - Accelerometro
  - Gravità
  - Giroscopio
  - Accelerazione lineare

# Sensori ambientali

---

- Questi sensori misurano vari parametri ambientali, come la temperatura e la pressione dell'aria ambiente, l'illuminazione e l'umidità. Questa categoria include barometri, fotometri e termometri.
  - Luce (Photometers)
  - Pressione (Barometers)
  - Temperatura
  - Umidità

# Sensori di posizionamento

---

- Questi sensori misurano la posizione fisica di un dispositivo. Questa categoria include sensori di orientamento e magnetometri.
  - Posizione fisica del device
  - Magnetometri (sensori di campo geomagnetico)
  - Sensori di prossimità

# Tipi di sensori

---

- Tipi di sensori supportati dalla piattaforma Android
  - Sensori basati su hardware
    - Componente fisico integrato nel dispositivo
    - Deriva i dati misurando direttamente proprietà specifiche
    - Esempi: sensore di luce, sensore di prossimità, magnetometro, accelerometro
  - Sensori basati su software
    - Software: sensore virtuale o composito
    - Deriva i dati da uno o più sensori hardware
    - Esempi: accelerazione lineare, orientamento

**Table 1.** Sensor types supported by the Android platform.

Sensor	Type	Description	Common Uses
TYPE_ACCELEROMETER	Hardware	Measures the acceleration force in $\text{m/s}^2$ that is applied to a device on all three physical axes (x, y, and z), including the force of gravity.	Motion detection (shake, tilt, etc.).
TYPE_AMBIENT_TEMPERATURE	Hardware	Measures the ambient room temperature in degrees Celsius ( $^{\circ}\text{C}$ ). See note below.	Monitoring air temperatures.
TYPE_GRAVITY	Software or Hardware	Measures the force of gravity in $\text{m/s}^2$ that is applied to a device on all three physical axes (x, y, z).	Motion detection (shake, tilt, etc.).
TYPE_GYROSCOPE	Hardware	Measures a device's rate of rotation in $\text{rad/s}$ around each of the three physical axes (x, y, and z).	Rotation detection (spin, turn, etc.).
TYPE_LIGHT	Hardware	Measures the ambient light level (illumination) in lx.	Controlling screen brightness.
TYPE_LINEAR_ACCELERATION	Software or Hardware	Measures the acceleration force in $\text{m/s}^2$ that is applied to a device on all three physical axes (x, y, and z), excluding the force of gravity.	Monitoring acceleration along a single axis.

<b>TYPE_MAGNETIC_FIELD</b>	Hardware	Measures the ambient geomagnetic field for all three physical axes (x, y, z) in $\mu\text{T}$ .	Creating a compass.
<b>TYPE_ORIENTATION</b>	Software	Measures degrees of rotation that a device makes around all three physical axes (x, y, z). As of API level 3 you can obtain the inclination matrix and rotation matrix for a device by using the gravity sensor and the geomagnetic field sensor in conjunction with the <code>getRotationMatrix()</code> method.	Determining device position.
<b>TYPE_PRESSURE</b>	Hardware	Measures the ambient air pressure in hPa or mbar.	Monitoring air pressure changes.
<b>TYPE_PROXIMITY</b>	Hardware	Measures the proximity of an object in cm relative to the view screen of a device. This sensor is typically used to determine whether a handset is being held up to a person's ear.	Phone position during a call.
<b>TYPE_RELATIVE_HUMIDITY</b>	Hardware	Measures the relative ambient humidity in percent (%).	Monitoring dewpoint, absolute, and relative humidity.
<b>TYPE_ROTATION_VECTOR</b>	Software or Hardware	Measures the orientation of a device by providing the three elements of the device's rotation vector.	Motion detection and rotation detection.
<b>TYPE_TEMPERATURE</b>	Hardware	Measures the temperature of the device in degrees Celsius ( $^{\circ}\text{C}$ ). This sensor implementation varies across devices and this sensor was replaced with the <b>TYPE_AMBIENT_TEMPERATURE</b> sensor in API Level 14	Monitoring temperatures.



# Disponibilità sensori

---

- La disponibilità del sensore varia da dispositivo a dispositivo, può anche variare tra le versioni di Android
- La maggior parte dei dispositivi ha accelerometro e magnetometro
- Alcuni dispositivi hanno barometri o termometri
- Il dispositivo può avere più di un sensore di un determinato tipo
- La disponibilità varia tra le versioni di Android

**Table 2.** Sensor availability by platform.

Sensor	Android 4.0 (API Level 14)	Android 2.3 (API Level 9)	Android 2.2 (API Level 8)	Android 1.5 (API Level 3)
TYPE_ACCELEROMETER	Yes	Yes	Yes	Yes
TYPE_AMBIENT_TEMPERATURE	Yes	n/a	n/a	n/a
TYPE_GRAVITY	Yes	Yes	n/a	n/a
TYPE_GYROSCOPE	Yes	Yes	n/a <sup>1</sup>	n/a <sup>1</sup>
TYPE_LIGHT	Yes	Yes	Yes	Yes
TYPE_LINEAR_ACCELERATION	Yes	Yes	n/a	n/a
TYPE_MAGNETIC_FIELD	Yes	Yes	Yes	Yes
TYPE_ORIENTATION	Yes <sup>2</sup>	Yes <sup>2</sup>	Yes <sup>2</sup>	Yes
TYPE_PRESSURE	Yes	Yes	n/a <sup>1</sup>	n/a <sup>1</sup>
TYPE_PROXIMITY	Yes	Yes	Yes	Yes
TYPE_RELATIVE_HUMIDITY	Yes	n/a	n/a	n/a
TYPE_ROTATION_VECTOR	Yes	Yes	n/a	n/a
TYPE_TEMPERATURE	Yes <sup>2</sup>	Yes	Yes	Yes

# Sensor Framework

---

- Si può accedere ai sensori e acquisire dati di sensori grezzi utilizzando il framework Sensor di Android
- Il framework Sensor fa parte del pacchetto *android.hardware* e include le seguenti classi e interfacce:
  - SensorManager
    - È possibile utilizzare questa classe per creare **un'istanza del servizio del sensore**. Questa classe fornisce vari metodi per accedere ed elencare i sensori, registrare e annullare la registrazione dei listener di eventi dei sensori e acquisire informazioni sull'orientamento. Questa classe fornisce anche diverse costanti del sensore che vengono utilizzate per segnalare l'accuratezza del sensore, impostare le velocità di acquisizione dei dati e calibrare i sensori

# SensorManager

---

- Accedere ai sensori e ai «listener» dei sensori
- Registrare e annullare la registrazione dei listener di eventi del sensore
- Acquisire informazioni sull'orientamento
- Fornisce costanti per accuratezza, velocità di acquisizione dei dati e calibrazione

# Altre classi

---

- Sensor
  - Per creare **un'istanza di un sensore specifico**
  - Questa classe fornisce vari metodi che consentono di determinare le capacità di un sensore
- SensorEvent
  - Per creare un **oggetto evento sensore**, che fornisce informazioni su un evento sensore
  - Un oggetto evento sensore include le seguenti informazioni: i dati del sensore non elaborati, il tipo di sensore che ha generato l'evento, l'accuratezza dei dati e il timestamp per l'evento

# SensorEventListener

---

- SensorEventListener
  - È possibile utilizzare questa interfaccia per creare due metodi di callback che ricevono notifiche (eventi del sensore) quando i valori del sensore cambiano (**nuovi dati**) o quando **l'accuratezza** del sensore cambia

# Usare i sensori

---

1. Determinare quali sensori sono disponibili sul dispositivo
2. Determinare le capacità di un singolo sensore
3. Portata massima, produttore, requisiti di alimentazione, risoluzione
4. Registrare i listener di eventi del sensore
5. Acquisire dati del sensore non elaborati (raw)
6. Definire anche la velocità minima per l'acquisizione dei dati del sensore
7. Annullare la registrazione dei listener di eventi del sensore

# Identificare i sensori nel dispositivo

```
private lateinit var sensorManager: SensorManager
```

```
...
```

```
sensorManager = getSystemService(Context.SENSOR_SERVICE) as  
SensorManager
```

```
val deviceSensors: List<Sensor> =  
sensorManager.getSensorList(Sensor.TYPE_ALL)
```

Se desideri elencare tutti i sensori di un determinato tipo, puoi usare un'altra costante invece di TYPE\_ALL come TYPE\_GYROSCOPE, TYPE\_LINEAR\_ACCELERATION o TYPE\_GRAVITY



# Controllare l'esistenza di un sensore

```
private lateinit var sensorManager: SensorManager
...
sensorManager = getSystemService(Context.SENSOR_SERVICE) as
SensorManager
if (sensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD)
!= null) {
    // Success! There's a magnetometer.
} else {
    // Failure! No magnetometer.
}
```

# Capacità di un sensore

---

- Oltre ad elencare i sensori presenti su un dispositivo, è possibile utilizzare i metodi pubblici della classe `Sensor` per determinare le capacità e gli attributi dei singoli sensori
- Ciò è utile se si desidera che l'applicazione si comporti in modo diverso in base ai sensori o alle capacità dei sensori disponibili su un dispositivo
- Ad esempio, è possibile utilizzare i metodi `getResolution()` e `getMaximumRange()` per ottenere la risoluzione di un sensore e l'intervallo massimo di misurazione
- È inoltre possibile utilizzare il metodo `getPower()` per ottenere i requisiti di alimentazione di un sensore

# Capacità e caratteristiche sensore

---

- Metodi della classe Sensor
  - getResolution() per la risoluzione del sensore
  - getMaximumRange() per il massimo intervallo di misurazione
  - getPower() per i requisiti di alimentazione del sensore
  - getVendor() e getVersion() per l'ottimizzazione per diversi sensori o diverse versioni di sensore
  - getMinDelay() per determinare la velocità massima alla quale il sensore può acquisire dati
    - Nota! La velocità massima di acquisizione dei dati di un sensore non è necessariamente la velocità con cui il framework del sensore fornisce i dati del sensore all'applicazione. La struttura del sensore riporta i dati attraverso gli eventi del sensore e diversi fattori influenzano la velocità con cui l'applicazione riceve gli eventi del sensore

# Esempio

```
private lateinit var sensorManager: SensorManager
private var mSensor: Sensor? = null

...

sensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager

if (sensorManager.getDefaultSensor(Sensor.TYPE_GRAVITY) != null) {
    val gravSensors: List<Sensor> = sensorManager.getSensorList(Sensor.TYPE_GRAVITY)
    // Use the version 3 gravity sensor.
    mSensor = gravSensors.firstOrNull { it.vendor.contains("Google LLC") && it.version == 3 }
}
if (mSensor == null) {
    // Use the accelerometer.
    mSensor = if (sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER) != null) {
        sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER)
    } else {
        // Sorry, there are no accelerometers on your device.
        // You can't play this game.
        null
    }
}
```

In questo esempio, stiamo cercando un sensore di gravità che ha Google LLC come fornitore e ha un numero di versione pari a 3. Se quel particolare sensore non è presente sul dispositivo, proviamo a utilizzare l'accelerometro

# Monitoraggio eventi del sensore

---

- Per monitorare i raw dati del sensore è necessario implementare due metodi di callback esposti attraverso l'interfaccia `SensorEventListener`:
  - `onAccuracyChanged ()`
    - Chiamato quando l'accuratezza del sensore è cambiata
  - `onSensorChanged ()`
    - Chiamato quando il sensore ha riportato un nuovo valore

# onAccuracyChanged()

---

- L'accuratezza del sensore è cambiata
  - In questo caso il sistema richiama il metodo `onAccuracyChanged()`, fornendo un riferimento all'oggetto `Sensor` che è cambiato e alla nuova precisione del sensore
  - La precisione è rappresentata da una delle quattro costanti di stato:  
`SENSOR_STATUS_ACCURACY_LOW`,  
`SENSOR_STATUS_ACCURACY_MEDIUM`,  
`SENSOR_STATUS_ACCURACY_HIGH`  
o `SENSOR_STATUS_UNRELIABLE`.

# onSensorChanged()

---

- Il sensore ha riportato un nuovo valore
  - In questo caso il sistema invoca il metodo `onSensorChanged()`, fornendo un oggetto `SensorEvent`
  - Un oggetto `SensorEvent` contiene informazioni sui nuovi dati del sensore, tra cui: l'accuratezza dei dati, il sensore che ha generato i dati, il timestamp in cui i dati sono stati generati e i nuovi dati registrati dal sensore

# Esempio – parte 1

```
class SensorActivity : Activity(), SensorEventListener {  
    private lateinit var sensorManager: SensorManager  
    private var mLight: Sensor? = null  
  
    public override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.main)  
  
        sensorManager = getSystemService(Context.SENSOR_SERVICE) as  
SensorManager  
        mLight = sensorManager.getDefaultSensor(Sensor.TYPE_LIGHT)  
    }  
  
    override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {  
        // Do something here if sensor accuracy changes.  
    }  
}
```

Che sensore  
è?



# Esempio – parte 1

```
class SensorActivity : Activity(), SensorEventListener {  
    private lateinit var sensorManager: SensorManager  
    private var mLight: Sensor? = null  
  
    public override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.main)  
  
        sensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager  
        mLight = sensorManager.getDefaultSensor(Sensor.TYPE_LIGHT)  
    }  
  
    override fun onAccuracyChanged(sensor: Sensor, accuracy: Int) {  
        // Do something here if sensor accuracy changes.  
    }  
}
```

Sensore di  
luce

# Esempio – parte 2

```
override fun onSensorChanged(event: SensorEvent) {  
    // The light sensor returns a single value.  
    // Many sensors return 3 values, one for each axis.  
    val lux = event.values[0]  
    // Do something with this sensor value.  
}  
  
override fun onResume() {  
    super.onResume()  
    mLight?.also { light ->  
        sensorManager.registerListener(this, light,  
            SensorManager.SENSOR_DELAY_NORMAL)  
    }  
}  
  
override fun onPause() {  
    super.onPause()  
    sensorManager.unregisterListener(this)  
}  
}
```

Il data delay (o frequenza di campionamento) controlla l'intervallo in cui gli eventi del sensore vengono inviati all'applicazione tramite il metodo di callback `onSensorChanged()`. La frequenza di campionamento predefinito dei dati è adatto per monitorare le tipiche modifiche all'orientamento dello schermo e utilizza un ritardo di 200.000 microsecondi. È possibile specificare altri ritardi di dati, come `SENSOR_DELAY_GAME` (ritardo di 20.000 microsecondi), `SENSOR_DELAY_UI` (ritardo di 60.000 microsecondi) o `SENSOR_DELAY_FASTEST` (ritardo di 0 microsecondi). A partire da Android 3.0 (Livello API 11) puoi anche specificare il ritardo come valore assoluto (in microsecondi)

# Dettagli e Best practice

- Il delay specificato è solo un ritardo suggerito; il sistema Android e altre applicazioni possono modificare questo ritardo
  - Come best practice, è necessario specificare il ritardo maggiore possibile poiché il sistema utilizza in genere un ritardo inferiore a quello specificato (ovvero, è necessario scegliere la **frequenza di campionamento più lenta che soddisfa ancora le esigenze dell'applicazione**). L'uso di un ritardo maggiore comporta un carico inferiore sul processore e quindi consuma meno energia
- Come best practice dovresti sempre **disabilitare i sensori che non ti servono**, specialmente quando la tua attività è in pausa. In caso contrario si può scaricare la batteria in poche ore perché alcuni sensori hanno requisiti di alimentazione sostanziali e possono consumare rapidamente la batteria. Il sistema non disabiliterà i sensori automaticamente quando lo schermo si spegne!

# Gestire le diverse configurazioni

---

- Android non specifica una configurazione di sensore standard per i dispositivi, il che significa che i produttori di dispositivi possono incorporare qualsiasi configurazione di sensore che desiderano nei loro dispositivi Android
- Di conseguenza, i dispositivi possono includere una varietà di sensori in una vasta gamma di configurazioni
- Se l'applicazione si basa su un tipo specifico di sensore, è necessario assicurarsi che il sensore sia presente su un dispositivo in modo che l'app possa funzionare correttamente.
- Sono disponibili **due opzioni** per garantire che un determinato **sensore sia presente** su un dispositivo:
  - **Rilevare i sensori in fase di esecuzione** e abilitare o disabilitare le funzionalità dell'applicazione come appropriato
  - **Utilizzare i filtri di Google Play** per indirizzare i dispositivi con configurazioni specifiche del sensore

# Rilevare sensori a runtime

```
private lateinit var sensorManager: SensorManager
...
sensorManager = getSystemService(Context.SENSOR_SERVICE) as
SensorManager

if (sensorManager.getDefaultSensor(Sensor.TYPE_PRESSURE) != null) {
    // Success! There's a pressure sensor.
} else {
    // Failure! No pressure sensor.
}
```

Si può usare in particolare  
quando il sensore NON è  
essenziale per il funzionamento  
dell'applicazione

# Filtrare con Google Play

---

- Se stai pubblicando la tua applicazione su Google Play, puoi utilizzare l'elemento `<uses-feature>` nel tuo file `manifest.xml` per non far installare e visualizzare l'applicazione a dispositivi che non dispongono della configurazione del sensore appropriata per la tua applicazione
- L'elemento `<uses-feature>` ha diversi descrittori hardware che ti consentono di filtrare le applicazioni in base alla presenza di sensori specifici
- I sensori che puoi elencare includono: accelerometer, barometer, compass (geomagnetic field), gyroscope, light, and proximity

# Esempio

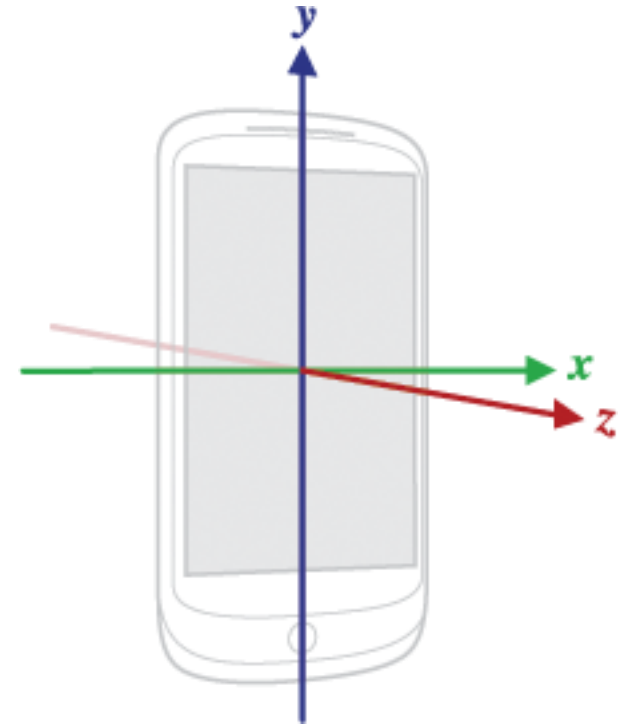
---

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

Gli utenti vedranno la tua applicazione su Google Play solo se il loro dispositivo ha un accelerometr

# Sistema di coordinate

- In generale, la struttura del sensore utilizza un sistema di coordinate standard a 3 assi per esprimere i valori dei dati
- Per la maggior parte dei sensori, il sistema di coordinate viene definito in relazione allo schermo del dispositivo quando il dispositivo viene mantenuto nell'orientamento predefinito (vedere la figura)
- Quando un dispositivo viene mantenuto nel suo orientamento predefinito, l'asse X è orizzontale e punta verso destra, l'asse Y è verticale e punta verso l'alto e l'asse Z punta verso l'esterno della faccia dello schermo
- In questo sistema, le coordinate dietro lo schermo hanno valori Z negativi. Questo sistema di coordinate viene utilizzato dai seguenti sensori:
  - Acceleration sensor
  - Gravity sensor
  - Gyroscope
  - Linear acceleration sensor
  - Geomagnetic field sensor





# Sistema di coordinate

- La cosa più importante da capire su questo sistema di coordinate è che gli assi non vengono scambiati quando cambia l'orientamento dello schermo del dispositivo, ovvero il sistema di coordinate del sensore non cambia mai mentre il dispositivo si muove
- Un altro punto importante è che l'applicazione non deve presumere che l'orientamento naturale (predefinito) di un dispositivo sia verticale. L'orientamento naturale per molti dispositivi tablet è orizzontale. E il sistema di coordinate del sensore si basa sempre sull'orientamento naturale di un dispositivo
- Infine, se l'applicazione abbina i dati del sensore alla visualizzazione su schermo, è necessario utilizzare il metodo `getRotation()` per determinare la rotazione dello schermo, quindi utilizzare il metodo `remapCoordinateSystem()` per mappare le coordinate del sensore alle coordinate dello schermo
- Occorre fare ciò anche se il manifest specifica `portrait-only display`

# Best practice

---

## 1. Raccogliere solo i dati del sensore quando l'applicazione è in primo piano (foreground)

- Sui dispositivi con Android 9 (livello API 28 o successivo), le app in esecuzione in background hanno le seguenti restrizioni:
  - I sensori che utilizzano la modalità di segnalazione continua, come accelerometri e giroscopi, non ricevono eventi
  - I sensori che utilizzano le modalità di reporting al cambio o one-shot non ricevono eventi
- Date queste restrizioni, è meglio rilevare gli eventi del sensore quando l'app è in primo piano o come parte di un servizio in primo piano

# Best practice

---

## 2. Unregister i listeners dei sensori

- Assicurati di annullare la registrazione del listener di un sensore quando hai finito di usare il sensore o quando l'attività del sensore si interrompe
- Se un listener di sensore è registrato e la sua attività è in pausa, il sensore continuerà ad acquisire dati e utilizzerà le risorse della batteria a meno che non si annulli la registrazione del sensore

# Esempio

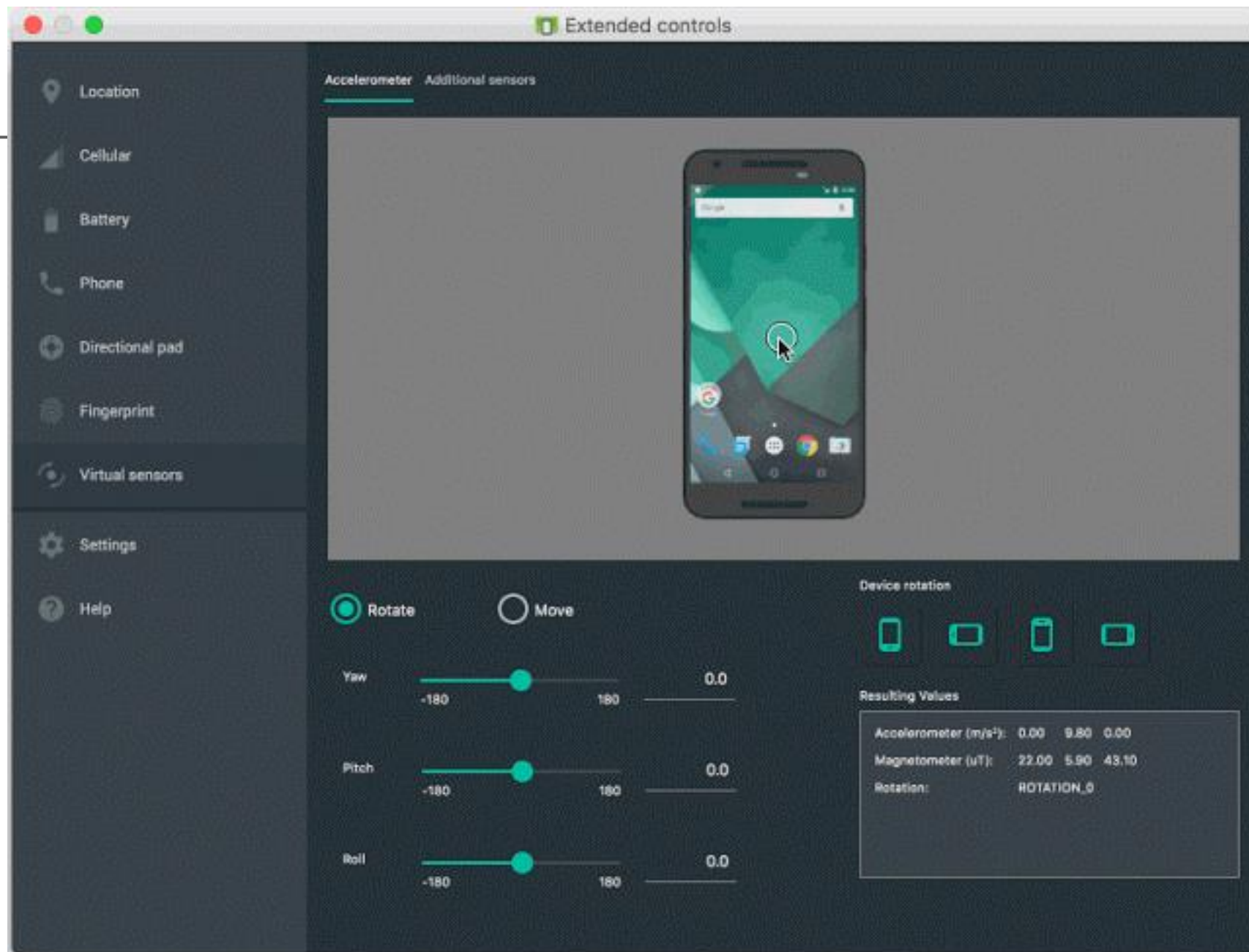
---

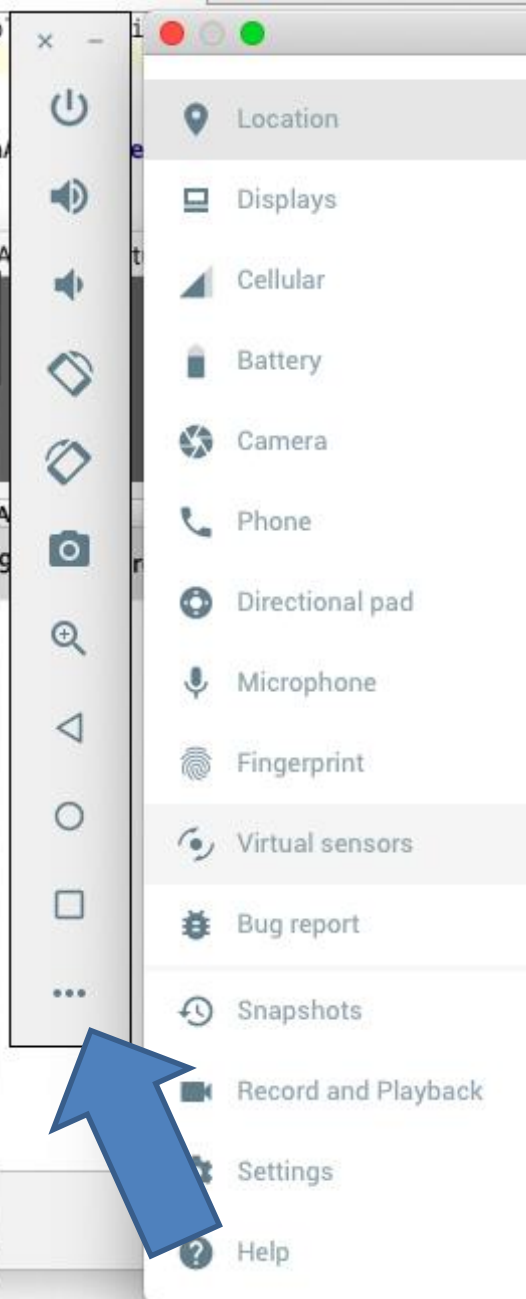
```
private lateinit var sensorManager: SensorManager
...
override fun onPause() {
    super.onPause()
    sensorManager.unregisterListener(this)
}
```

# Best practice

---


3. L'emulatore Android include una serie di controlli di sensori virtuali che consentono di testare sensori come accelerometro, temperatura ambiente, magnetometro, prossimità, luce e altro ancora





### Extended controls - Nexus\_S\_API\_29:5554

Accelerometer Additional sensors



☒ Rotate ☐ Move

Z-Rot -180 180 0.0

X-Rot -180 180 -4.8

Y-Rot -180 180 0.0

Device rotation

☐ Portrait ☐ Landscape ☐ Portrait ☐ Landscape

Resulting values

Accelerometer (m/s <sup>2</sup> ):	0.00	9.78	0.81
Gyroscope (rad/s):	0.00	0.00	0.00
Magnetometer (μT):	0.00	9.89	-47.75

# Best practice

---

## 4. Non «bloccare» il metodo `onSensorChanged()`

- I dati del sensore possono cambiare a un ritmo elevato, il che significa che il sistema può chiamare il metodo `onSensorChanged(SensorEvent)` abbastanza spesso. Come best practice, dovresti fare il meno possibile all'interno del metodo `onSensorChanged(SensorEvent)` in modo da non bloccarlo. Se l'applicazione richiede di eseguire il filtraggio o la riduzione dei dati dei sensori, è necessario eseguire tale operazione al di fuori del metodo `onSensorChanged(SensorEvent)`



# Best practice

---

## 5. Evitare l'uso di metodi o tipi di sensori obsoleti

- Diversi metodi e costanti sono stati deprecati. In particolare, il tipo di sensore `TYPE_ORIENTATION` è stato deprecato. Per ottenere i dati di orientamento è necessario utilizzare invece il metodo `getOrientation()`
- Allo stesso modo, il tipo di sensore `TYPE_TEMPERATURE` è stato deprecato. Dovresti utilizzare il tipo di sensore `TYPE_AMBIENT_TEMPERATURE` invece su dispositivi che eseguono Android 4.0 e superiori

# Best practice

---

## 6. Verificare i sensori prima di utilizzarli

- Verificare sempre l'esistenza di un sensore su un dispositivo prima di tentare di acquisire dati da esso
- Non dare per scontato che esista un sensore semplicemente perché è un sensore utilizzato di frequente. I produttori di dispositivi non sono tenuti a fornire particolari sensori nei loro dispositivi

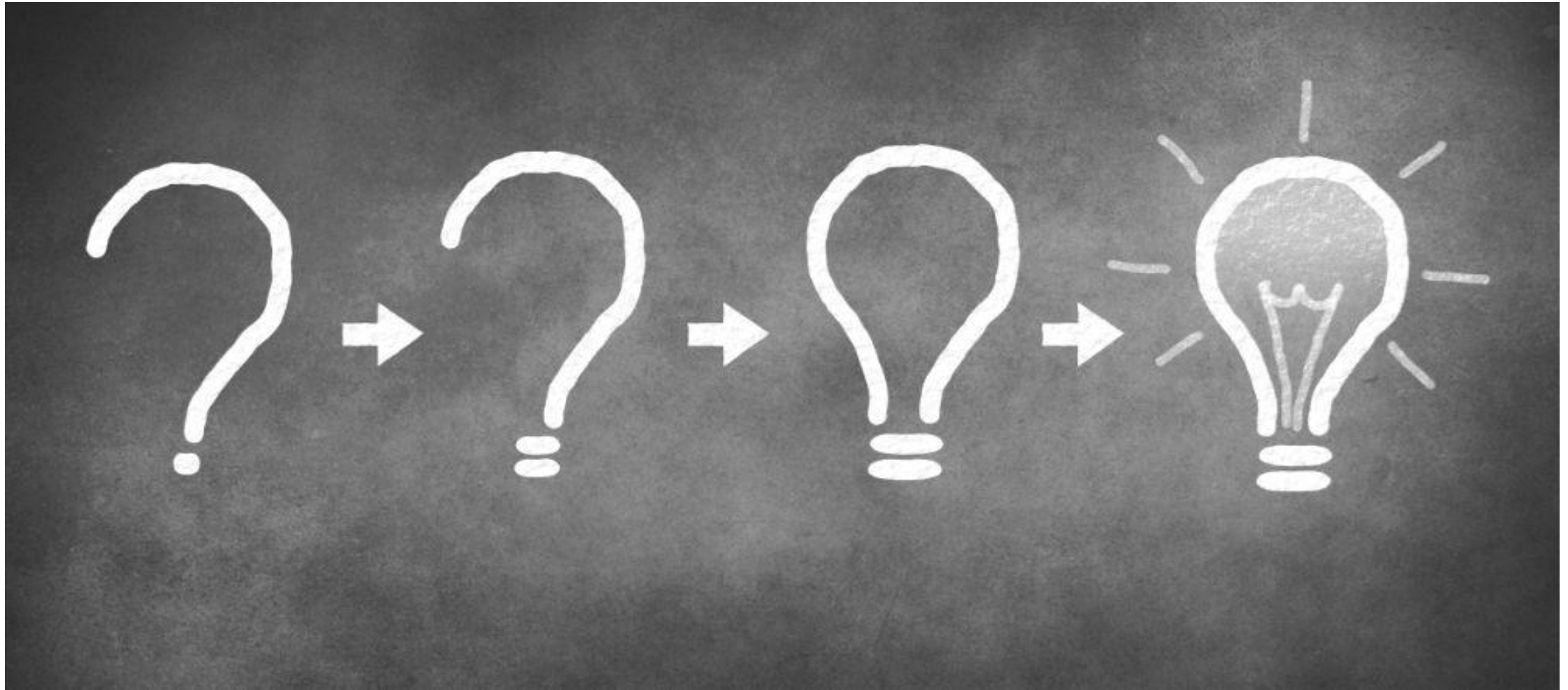
# Best practice

---

## 7. Scegli attentamente i ritardi (delay) del sensore

- Quando registri un sensore con il metodo `registerListener()`, assicurati di scegliere una velocità di consegna adatta alla tua applicazione o al tuo caso d'uso
- I sensori possono fornire dati a velocità molto elevate. Consentire al sistema di inviare dati extra non necessari spreca risorse di sistema e consuma la batteria

# Domande???



# Riferimenti

---

- <https://developer.android.com/guide/topics/sensors>
- [https://developer.android.com/guide/topics/sensors/sensors overview](https://developer.android.com/guide/topics/sensors/sensors-overview)