**SENSORI**

**Si descriva l’utilizzo di un sensore spiegando cosa si deve fare per utilizzarlo. Arricchire la spiegazione con frammenti di codice.**

I sensori consentono di misurare il movimento, l'orientamento e varie condizioni ambientali, sono in grado di fornire dati grezzi con elevata precisione e accuratezza e sono utili se si desidera monitorare il movimento o il posizionamento tridimensionale del dispositivo o se si desidera monitorare i cambiamenti nell'ambiente circostante ad un dispositivo. Android supporta tre grandi categorie di sensori:

* ***Sensori di movimento***, misurano le forze di accelerazione e le forze di rotazione lungo tre assi, categoria che comprende accelerometri, sensori di gravità, giroscopi e sensori vettoriali rotazionali.
* ***Sensori ambientali***, misurano vari parametri ambientali, come temperatura e pressione dell'aria ambiente, illuminazione e umidità, categoria che comprende barometri, fotometri e termometri.
* ***Sensori di posizione***, misurano la posizione fisica di un dispositivo, categoria che comprende sensori di orientamento.

Per utilizzare un sensore in un’activity, occorre seguire gli step sottostanti:

1. L’activity deve implementare ***SensorEventListener***:

|  |
| --- |
| *public class SensorActivity extends Activity implements SensorEventListener { … }* |

1. In ***onCreate()*** istanziamo il ***sensorManager***, istanziamo il sensore e controlliamo se esso esiste (es. il sensore ***TYPE\_LIGHT*** per la luminosità dell’ambiente):

|  |
| --- |
| *private SensorManager sensorManager;*  *private Sensor sensorLight;*  *sensorManager = (SensorManager) getSystemService(Context.SENSOR\_SERVICE);*  *sensorLight = sensorManager.getDefaultSensor(Sensor.TYPE\_LIGHT);*  *if (sensorLight != null) {*  *// success*  *} else {*  *// failure*  *}* |

1. Importante è, in ***onResume()*** e ***onPause()***, rispettivamente la registrazione e la deregistrazione del sensor listener:

|  |
| --- |
| *@Override*  *protected void onResume() {*  *super.onResume()*  *sensorManager.registerListener(this, lightSensor, SensorManager.SENSOR\_DELAY\_NORMAL);*  *}*  *@Override*  *protected void onPause() {*  *super.onPause();*  *sensorManager.unregisterListener(this);*  *}* |

1. Eseguire l’override di metodi ***onAccuracyChanged()*** e ***onSensorChanged(),*** dei listener invocati rispettivamente quando cambia la precisione del sensore e quando viene effettuato un campionamento:

|  |
| --- |
| *@Override*  *public final void onAccuracyChanged(Sensor sensor, int accuracy) {*  *// do something when accuracy changes*  *}*  *@Override*  *public final void onSensorChanged(SensorEvent event) {*  *// values are stored in event.values array*  *}* |

**Quando si registra il listener di un sensore è possibile selezionare la velocità di campionamento da utilizzare:**

**– SENSOR\_DELAY\_NORMAL (0,2sec)**

**– SENSOR\_DELAY\_GAME (0,02sec)**

**– SENSOR\_DELAY\_UI (0,06sec)**

**– SENSOR\_FASTEST (0sec )**

**Si discuta dei vantaggi e svantaggi di queste varie possibilità e di quale accortezze deve avere il programmatore per un app che utilizza i sensori.**

La possibilità di selezionare quattro opzioni per la velocità di campionamento dei sensori si basa su due fattori: performance dell’applicazione e durata della batteria del dispositivo. Per “performance” s’intende lo scopo dell’applicazione, ed è opportuno eseguire un trade-off tra i due fattori specificati. Un vantaggio che riguarda la performance indica che un’alta frequenza di campionamento implica una maggiore precisione dei dati ricavati; tuttavia, i sensori richiedono un uso elevato della batteria. Ad esempio, con SENSOR\_DELAY\_NORMAL abbiamo una bassa precisione ed un utilizzo ridotto della batteria rispetto a SENSOR\_DELAY\_GAME, che ha un’alta precisione ed un utilizzo elevato della batteria.

L’accortezza che il programmatore deve avere quando sviluppa un’app che utilizza i sensori è, di conseguenza, la gestione ottimizzata dei sensori: questi vanno registrati e deregistrati programmaticamente.