

ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
Dipartimento di Informatica – Scienza e Ingegneria (DISI)
C.d.S. in Ingegneria e Scienze Informatiche, Campus di Cesena

Programmazione in Android

Accesso ai Sensori e Geolocalizzazione

Angelo Croatti
a.croatti@unibo.it

Sistemi Embedded e Internet of Things
A.A. 2019 – 2020

Outline

- 1 Android Sensor Framework
 - Sensori Supportati e API
 - Identificazione e Monitoring dei Sensori
- 2 Sensori di Movimento
 - Il Sistema di Riferimento
 - Accelerometro
 - Giroscopio
- 3 NFC
 - Inizializzazione dei componenti
 - Attivazione, Rilascio e Connessione
 - NFC I/O
- 4 Geolocalizzazione in Android
 - Network-based vs. GPS-based
 - Informazioni sulla posizione GPS
 - Permessi



Sensori sui device Android

- La maggior parte dei device Android dispone di una serie di sensori built-in con i quali è possibile interagire.
- Generalmente, i sensori producono stream di dati raw ad elevata accuratezza e precisione.
- In Android sono supportate tre macro-categorie di sensori:
 1. **Sensori di Movimento** (motion sensors), misurano le forze di accelerazione e le forze di rotazione relative ai tre assi del SdR (es. Accelerometro, Giroscopio, ...).
 2. **Sensori Ambientali** (environmental sensors), misurano parametri ambientali come temperatura, pressione e grado di illuminazione (es. barometro, sensore di luce, ...).
 3. **Sensori di Posizione** (position sensors), determinano la posizione fisica del dispositivo (es. sensori di orientamento, magnetometro, ...).

Android Sensor Framework (ASF)

- Costituisce la porzione di Framework Android per l'accesso e la gestione dei sensori di ciascun device Android.
- Tra le altre funzionalità, consente di:
 - ▶ Determinare quali sensori sono disponibili.
 - ▶ Stabilire quali funzionalità sono disponibili per ciascun sensore e configurarne i parametri.
 - ▶ Acquisire i dati raw prodotti continuamente dai sensori (specificandone il rate desiderato).
 - ▶ Registrare listener specifici per ciascun sensore.

» <https://developer.android.com/guide/topics/sensors/>

Sensori HW vs. Sensori SW

- **Sensori Hardware**, sono componenti fisici montati sul device che producono i propri flussi di dati misurando specifiche proprietà e condizioni ambientali.
- **Sensori Software**, non sono associati a nessun componente fisico, bensì propongono i propri flussi dati come combinazione logica dei flussi dati sintetizzati dai sensori HW.

Sensori Supportati nell'ASF

Sensor	Type	Description	Common Uses
TYPE_ACCELEROMETER	Hardware	Measures the acceleration force in 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 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 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 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.
TYPE_MAGNETIC_FIELD	Hardware	Measures the ambient geomagnetic field for all three physical axes (x, y, z) in μT .	Creating a compass.
TYPE_ORIENTATION	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 getRotationMatrix() method.	Determining device position.
TYPE_PRESSURE	Hardware	Measures the ambient air pressure in hPa or mbar.	Monitoring air pressure changes.
TYPE_PROXIMITY	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.
TYPE_RELATIVE_HUMIDITY	Hardware	Measures the relative ambient humidity in percent (%).	Monitoring dewpoint, absolute, and relative humidity.
TYPE_ROTATION_VECTOR	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.
TYPE_TEMPERATURE	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 TYPE_AMBIENT_TEMPERATURE sensor in API Level 14.	Monitoring temperatures.

API per la gestione dei sensori

- L'ASF mette a disposizione una serie di componenti presenti nel package `android.hardware.*`.
- I più significativi:
 - ▶ **SensorManager**, permette di creare l'istanza di un oggetto che rappresenta il servizio associato ad un sensore specifico. Fornisce i metodi per l'accesso ai sensori e per la registrazione di listener.
 - ▶ **Sensor**, permette di creare istanze specifiche per ciascun sensore supportato.
 - ▶ **SensorEvent**, rappresenta l'istanza per ciascun evento propagato da ciascun sensore. Include sia i dati raw prodotti dal sensore sia le informazioni correnti associate al sensore (accuratezza, timestamp, ...).
 - ▶ **SensorEventListener**, interfaccia che deve essere implementata da qualunque oggetto che debba essere progettato per ricevere informazioni dai sensori d'interesse.

Identificazione dei sensori disponibili I

- Accedendo all'istanza del `SensorManager` è possibile stabilire quali sensori sono attualmente disponibili nel device.
 - ▶ Tale istanza è ottenibile tramite il metodo `getSystemService()`, specificando come parametro `Context.SENSOR_SERVICE`.

Esempio – Lista dei Sensori disponibili

```
private SensorManager sm;

@Override
protected void onCreate(Bundle savedInstanceState){
    /* ... */

    sm = (SensorManager) getSystemService(Context.SENSOR_SERVICE);

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


Identificazione dei sensori disponibili II

- Viceversa, è possibile verificare la disponibilità di ogni singolo sensore sfruttando la funzione `getDefaultSensor()` fornita dal `SensorManager`.

Esempio – Disponibilità di uno specifico sensore

```
private SensorManager sm;
private Sensor accelerometer;

@Override
protected void onCreate(Bundle savedInstanceState){
    /* ... */
    accelerometer = sm.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);

    if(accelerometer != null){
        // The Accelerometer sensor is available for this device.
    }
}
```

Monitoring dei dati prodotti dai sensori

Affinché sia possibile monitorare i dati prodotti da uno specifico sensore è necessario:

1. Definire un apposito listener, implementando l'interfaccia **SensorEventListener** costituita da due specifiche callback:
 - ▶ **onAccuracyChanged()**, invocata dal sistema quando qualche parametro dello specifico sensore viene modificato in relazione alla sua accuratezza nel produrre lo stream di dati raw.
 - ▶ **onSensorChanged()**, invocata dal sistema quando un nuovo dato dello stream è disponibile per essere letto. Le informazioni sono propagate al listener attraverso un parametro di tipo **SensorEvent**.
2. Registrare il listener presso il **SensorManager**.

Monitoring dei dati prodotti dai sensori – Esempio I

Esempio – Activity che usa il sensore di luminosità ambientale

```
public class MainActivity extends Activity {  
  
    private SensorManager sm;  
    private Sensor lightSensor;  
    private LightSensorListener lsListener;  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(/* ... */);  
  
        sm = (SensorManager) getSystemService(Context.SENSOR_SERVICE);  
        lightSensor = sm.getDefaultSensor(Sensor.TYPE_LIGHT);  
  
        if (lightSensor != null)  
            lsListener = new LightSensorListener();  
    }  
}
```

Monitoring dei dati prodotti dai sensori – Esempio II

```
@Override
protected void onResume() {
    super.onResume();

    if(lightSensor != null)
        sm.registerListener(lsListener, lightSensor,
            SensorManager.SENSOR_DELAY_NORMAL);
}

@Override
protected void onPause() {
    super.onPause();

    if(lightSensor != null)
        sm.unregisterListener(lsListener);
}
```

Monitoring dei dati prodotti dai sensori – Esempio III

Esempio – Definizione del Listener

```
public class LightSensorListener implements SensorEventListener{

    private static final String LOG_TAG = "app-tag";

    @Override
    public void onSensorChanged(SensorEvent event) {
        float actualvalue = event.values[0];

        Log.d(LOG_TAG, "Actual Value: " + actualvalue);
    }

    @Override
    public void onAccuracyChanged(Sensor sensor, int accuracy) {
        //do something
    }
}
```

Monitoring dei dati prodotti dai sensori – Esempio IV

- La registrazione del listener per uno specifico sensore può essere fatta attraverso l'utilizzo del metodo `registerListener()` fornito dal `SensorManager`
 - ▶ A tale metodo devono essere passate come parametro le istanza del listener e del sensore. Inoltre, il terzo parametro fa riferimento al rate di emissione del valore desiderato.
- Dualmente, il listener può essere de-registrato invocando il metodo `unregisterListener()`.
- Nel listener, è possibile accedere ai valori prodotti dallo specifico sensore attraverso il vettore di float ottenibile dal parametro di tipo `SensorEvent` della callback `onSensorChanged()`.
 - ▶ Il numero di valori presenti in tale vettore variano a seconda del tipo di sensore che si sta monitorando.

Best-practices per l'uso dei Sensori

- Deregistrare sempre i listener per i sensori nella callback `onPause()` dell'activity che utilizza il sensore.
- Non inserire meccanismi bloccanti nella funzione `onSensorChanged()`.
 - ▶ Il listener può tuttavia utilizzare task asincroni per eseguire compiti long-running sui dati prodotti dai sensori.
- Verificare sempre la presenza dei sensori prima di utilizzarli.

Outline

- 1 Android Sensor Framework
 - Sensori Supportati e API
 - Identificazione e Monitoring dei Sensori
- 2 Sensori di Movimento
 - Il Sistema di Riferimento
 - Accelerometro
 - Giroscopio
- 3 NFC
 - Inizializzazione dei componenti
 - Attivazione, Rilascio e Connessione
 - NFC I/O
- 4 Geolocalizzazione in Android
 - Network-based vs. GPS-based
 - Informazioni sulla posizione GPS
 - Permessi



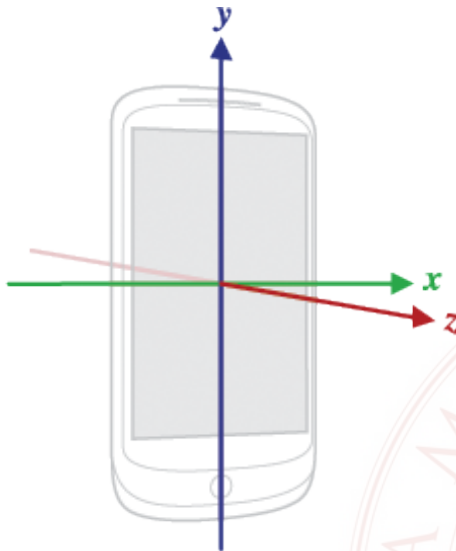
I Sensori di Movimento

- La categoria di sensori più utilizzata e maggiormente diffusa sui diversi device è quella che fa riferimento ai sensori di movimento.
 - ▶ Appartengono a questa categoria, tra gli altri, **accelerometro** e **giroscopio**, generalmente disponibili come sensori HW.
- I sensori di movimento possono essere utilizzati per identificare il movimento del dispositivo con riferimento alle coordinate spaziali definite in termini solidali al dispositivo stesso.
- Esempio d'applicazione dei sensori di movimento:
 - ▶ Determinare se un dispositivo viene agitato (shaking).
 - ▶ Determinare la rotazione del dispositivo rispetto all'utente.
 - ▶ Determinare se si sta viaggiando in auto oppure se si sta camminando.
 - ▶ ...

Sistema di riferimento per i Sensori I

- Si utilizza un sistema di coordinate basato su un SdR a tre assi (X,Y,Z) solidali con il dispositivo.
- Il SdR è definito in funzione dell'orientamento standard dello specifico dispositivo (*portrait* per la maggior parte dei dispositivi).
- Quando il device è nella posizione standard, vale che:
 - ▶ L'asse X è orizzontale e il suo asse positivo è definito verso destra
 - ▶ L'asse Y è verticale e il suo asse positivo è identificato verso l'alto
 - ▶ L'asse Z esce dallo schermo del device ortogonalmente agli altri due assi con direzione positiva.
- Si noti che non viene fatto nessuno swap degli assi X e Y quando il dispositivo è ruotato.

Sistema di riferimento per i Sensori II



Sistema di riferimento per i Sensori III

- Non è assolutamente detto che l'orientamento di default sia portrait. Molti dispositivi (tra cui alcuni tablet e tutti gli smart glasses) assumono come orientamento di default il landscape.
- Alcuni sensori restituiscono i propri valori con riferimento al SdR terrestre e dunque non quello solidale al device.
 - ▶ Risulta sempre opportuno verificare questo aspetto prima di interpretare i valori restituiti da un sensore e applicare eventualmente matrici di trasformazione.

» [android-developers.blogspot.it/2010/09/
one-screen-turn-deserves-another.html](http://android-developers.blogspot.it/2010/09/one-screen-turn-deserves-another.html)

Accelerometro I

- Concettualmente, misura l'accelerazione (in m/s^2) applicata al dispositivo lungo i tre assi del SdR, includendo anche la forza di gravità.
 - ▶ In condizione di equilibrio (device fermo, appoggiato su una superficie piana con lo schermo rivolto verso l'alto) i valori di accelerazione per gli assi X e Y sono prossimi al valore zero mentre il valore dell'accelerazione lungo l'asse Z è prossimo al valore (assoluto) dell'accelerazione di gravità 9,81.

Esempio – Istanza per l'accelerometro

```
SensorManager sm = ...  
Sensor accelerometer = sm.getDefaultSensor(  
    Sensor.TYPE_ACCELEROMETER);
```

Accelerometro II

- Qualora si vogliano ottenere i valori di accelerazione senza considerare la forza di gravità, si può far riferimento al sensore SW **accelerometro lineare**.

Esempio – Istanza per l'accelerometro lineare

```
SensorManager sm = ...  
Sensor linear_accelerometer = sm.getDefaultSensor(  
    Sensor.TYPE_LINEAR_ACCELEROMETER);
```

- In entrambi i casi, nella callback `onSensorChanged()`, il vettore di float restituito mediante il campo `values` del parametro di tipo `SensorEvent` proporrà rispettivamente l'accelerazione lungo X,Y e Z nei primi tre elementi del vettore.

Giroscopio

- Concettualmente, misura la rotazione (in *rad/s*) rispetto ai tre assi del dispositivo.
 - ▶ La rotazione positiva è quella che è eseguita in direzione oraria.
- Generalmente, i valori ottenuti dal giroscopio sono combinati con i dati temporali per calcolare la rotazione del dispositivo con l'evolvere del tempo.

Esempio – Istanza per il giroscopio

```
SensorManager sm = ...  
Sensor gyroscope = sm.getDefaultSensor(  
    Sensor.TYPE_GYROSCOPE);
```

- Anche in questo caso, i tre valori delle rotazioni rispetto agli assi X, Y e Z sono forniti rispettivamente come primo, secondo e terzo elemento del vettore di float nella relativa callback.

Outline

- 1 Android Sensor Framework
 - Sensori Supportati e API
 - Identificazione e Monitoring dei Sensori
- 2 Sensori di Movimento
 - Il Sistema di Riferimento
 - Accelerometro
 - Giroscopio
- 3 **NFC**
 - Inizializzazione dei componenti
 - Attivazione, Rilascio e Connessione
 - NFC I/O
- 4 Geolocalizzazione in Android
 - Network-based vs. GPS-based
 - Informazioni sulla posizione GPS
 - Permessi

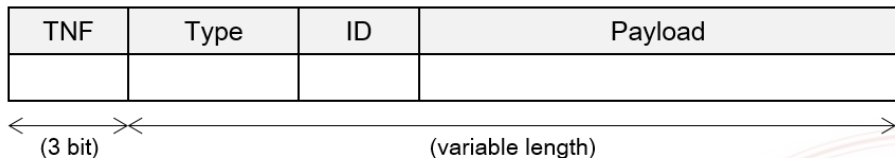


Near Field Communication (NFC)

- Tecnologia per connettività wireless a corto raggio di tipo contact-less.
 - ▶ Quando due dispositivi dotati di sensori NFC (rispettivamente *initiator* e *target/tag*) si trovano nelle vicinanze (entro i 5cm), tra i due viene creata una rete ad-hoc di tipo P2P per lo scambio di un quantitativo limitato di dati.
- La lettura di un generico tag NFC può essere attuata interpretando i valori di uno o più record NDEF (NFC Data Exchange Format) memorizzati nel tag.
- In Android è possibile accedere in lettura e in scrittura ai tag NFC abilitati avvalendosi della libreria implementata nel package `android.nfc`.

Struttura di un record NDEF I

Near Field Communication NDEF Record



- **TNF (Type Name Format)**, specifica come interpretare il successivo campo **Type**. Può assumere valori di default tra i quali: **TNF_EMPTY**, **TNF_ABSOLUTE_URI**, **TNF_EXTERNAL_TYPE**, **TNF_WELL_KNOWN**,
- **Type**, descrive il tipo specifico assunto dal record NDEF.

Struttura di un record NDEF II

- ▶ Nel caso più comune in cui al precedente campo TNF sia stato associato il valore `TNF_WELL_KNOWN`, il campo type deve essere utilizzato per specificare un valore valido di RDT (Record Type Description).
- ▶ In generale, si tende a specificare un valore RDT pari a `RDT_TEXT` che corrisponde al tipo MIME `text/plain`.
- **ID**, identificatore univoco (opzionale) per il record.
- **Payload**, il contenuto del record che sarà letto/scritto dall'initiator.

Permessi

- Per utilizzare il supporto NFC offerto dal sistema operativo è necessario dichiararne l'intenzione nel File Manifest, specificando i seguenti permessi.

Permessi per NFC

```
<uses-permission  
    android:name="android.permission.NFC"/>  
  
<uses-feature  
    android:name="android.hardware.nfc" android:required="true"/>
```

Nota. Il supporto all'NFC è disponibile in Android a partire dall'API Level 10. Pertanto per utilizzare l'NFC il minSdkVersion specificato deve essere uguale o superiore a 10.

Inizializzazione del supporto NFC I

- In Android, l'entry point per l'uso del sensore NFC è fornito dalla classe **NfcAdapter**.
 - ▶ Consente di attivare il sensore NFC, di identificare un eventuale tag nelle vicinanze e di leggere/scrivere tale tag.
- L'accesso al sensore NFC è esclusivo. Ogni applicazione deve ottenere e rilasciare il sensore esplicitamente.
- L'identificazione di un tag è retro-propagata all'activity mediante un Intent, opportunamente filtrato sul tipo/categoria.

Inizializzazione del supporto NFC II

Esempio - Inizializzazione

```
private static final String MIME_TEXT_PLAIN = "text/plain";

private NfcAdapter nfcAdapter;

@Override
protected void onCreate(Bundle savedInstanceState){
    super.onCreate(savedInstanceState);
    setContentView(/* ... */);

    nfcAdapter = NfcAdapter.getDefaultAdapter(this);

    if(nfcAdapter == null){
        //NFC not supported
        finish();
    }
}
```

Attivazione del supporto NFC I

- L'attivazione dell'NFC deve essere fatta contestualmente all'ingresso dell'activity nello stato di Running (ovvero, nella chiamata di onResume)
 - ▶ Si basa sulla definizione di una serie di Intent che predispongono l'applicazione per poter identificare eventuali tag NFC avvicinati al sensore.

Esempio - Start NFC Dispatching

```
@Override
public void onResume(){
    super.onResume();
    startNFCDispatch(this, nfcAdapter);
}
```

Attivazione del supporto NFC II

```
private void startNFCDispatch(Activity a, NfcAdapter adapter){
    Context ctx = a.getApplicationContext();

    final Intent i = new Intent(ctx, a.getClass());
    i.setFlags(Intent.FLAG_ACTIVITY_SINGLE_TOP);

    final PendingIntent pi = PendingIntent.getActivity(ctx, 0, i, 0);

    IntentFilter[] filters = new IntentFilter[1];
    filters[0] = new IntentFilter();
    filters[0].addAction(NfcAdapter.ACTION_NDEF_DISCOVERED);
    filters[0].addCategory(Intent.CATEGORY_DEFAULT);

    try {
        filters[0].addDataType(MIME_TEXT_PLAIN);
    } catch (MalformedMimeTypeException e) { /* ... */ }

    String[][] techList = new String[][]{};
    adapter.enableForegroundDispatch(a, pi, filters, techList);
}
```


Rilascio del supporto NFC I

- Il rilascio dell'NFC deve essere fatto contestualmente all'uscita dell'activity dallo stato di Running (ovvero, nella chiamata di onPause)
- Diversamente, nessun'altra applicazione potrà utilizzare il sensore.

Esempio - Stop NFC Dispatching

```
@Override
protected void onPause(){
    stopNFCDispatch(this, nfcAdapter);
    super.onPause();
}

private void stopNFCDispatch(Activity a, NfcAdapter adapter){
    adapter.disableForegroundDispatch(a);
}
```

Connessione ad un Tag NFC I

- Su qualunque activity è possibile ridefinire la callback `onNewIntent(Intent i)`
 - ▶ Il sistema richiama tale metodo ogni qualvolta è riconosciuto uno specifico Intent (PendingIntent) opportunamente registrato dall'activity.
- È il caso dell'NFC, nella cui fase di attivazione è stato registrato un Pending Intent specifico per i tag NFC desiderati.
 - ▶ In particolare, è stato definito un Pending Intent associato all'azione di riconoscimento dei record di tipo NDEF.

Connessione ad un Tag NFC II

Esempio - Ridefinizione del metodo onNewIntent()

```
@Override
protected void onNewIntent(Intent intent) {
    Tag tag = (Tag)intent.getParcelableExtra(NfcAdapter.EXTRA_TAG);

    if(NfcAdapter.ACTION_NDEF_DISCOVERED.equals(intent.getAction()))
        if(MIME_TEXT_PLAIN.equals(intent.getType())){
            //do something (I/O on tag)
        }
}
```

- Tramite l'intent è possibile recuperare un'istanza di tipo **Tag** che resta valida finché il tag NFC resta in prossimità del sensore.
- Tramite questa istanza è possibile leggere o scrivere il tag NFC.

Lettura di un Tag NFC I

Esempio - Lettura del contenuto di un tag

```
public String readNFCTag(Tag tag) throws Exception {
    Ndef ndef = Ndef.get(tag);

    if (ndef == null)
        return null;

    NdefRecord[] records = ndef.getCachedNdefMessage().getRecords();

    for(NdefRecord r : records)
        if(r.getTnf() == NdefRecord.TNF_WELL_KNOWN
            && Arrays.equals(r.getType(), NdefRecord.RTD_TEXT))
            return analyzePayload(r.getPayload());

    return null;
}
```

Lettura di un Tag NFC II

```
private String analyzePayload(byte[] payload) throws Exception {  
    String encoding = ((payload[0] & 0200) == 0) ? "UTF-8":"UTF-16";  
    int langCodeLen = payload[0] & 0077;  
  
    String tagContent = new String(payload, langCodeLen + 1,  
                                   payload.length - langCodeLen - 1, encoding);  
  
    return tagContent;  
}
```

- Ogni tag NFC può contenere uno o più record NDEF (in funzione della dimensione/capienza del tag).
- Qualora il record NDEF sia del tipo interessato, ovvero se specifica i parametri desiderati per TNF e Type è possibile analizzare il Payload per estrarne il contenuto.

Scrittura di un Tag NFC I

- Durante la scrittura di un tag, il tag deve rimanere in prossimità del sensore NFC per tutta la durata del processo.
- Ciascun record NDEF opportunamente creato può essere poi scritto sul tag, avvalendosi dell'istanza del tag, come nel caso dell'operazione di lettura.
- La creazione del record NDEF deve avvenire specificando tutti i parametri richiesti, opportunamente codificati.

Scrittura di un Tag NFC II

Esempio - Scrittura del contenuto di un tag

```
public void writeTag(String content, Tag tag) throws Exception{
    NdefRecord[] records = new NdefRecord[1];
    records[0] = createRecord(content);

    NdefMessage message = new NdefMessage(records);

    Ndef ndef = Ndef.get(tag);
    ndef.connect();
    ndef.writeNdefMessage(message);
    ndef.close();
}
```

Scrittura di un Tag NFC III

```
private NdefRecord createRecord(String text) throws Exception{
    String lang = "en";

    byte[] textBytes = text.getBytes();
    byte[] langBytes = lang.getBytes("US-ASCII");

    int textLength = textBytes.length;
    int langLength = langBytes.length;

    byte[] payload = new byte[1 + langLength + textLength];
    payload[0] = (byte) langLength;

    System.arraycopy(langBytes, 0, payload, 1, langLength);
    System.arraycopy(textBytes, 0, payload, 1 + langLength,
        textLength);

    NdefRecord recordNFC = new NdefRecord(NdefRecord.TNF_WELL_KNOWN,
        NdefRecord.RTD_TEXT, new byte[0], payload);

    return recordNFC;
}
```


Outline

- 1 Android Sensor Framework
 - Sensori Supportati e API
 - Identificazione e Monitoring dei Sensori
- 2 Sensori di Movimento
 - Il Sistema di Riferimento
 - Accelerometro
 - Giroscopio
- 3 NFC
 - Inizializzazione dei componenti
 - Attivazione, Rilascio e Connessione
 - NFC I/O
- 4 Geolocalizzazione in Android
 - Network-based vs. GPS-based
 - Informazioni sulla posizione GPS
 - Permessi



Geolocalizzazione in Android

- In android l'accesso ai servizi di localizzazione, tramite il GPS e non solo, è fornito tramite istanze di oggetti definiti nel package `android.location.*`.
- Come nel caso dei sensori, anche per la localizzazione esiste un `LocationManager` con cui è possibile istanziare i componenti necessari per ottenere informazioni sulla posizione dell'utente.
- Ottenuta l'istanza del `LocationManager`, è possibile:
 - ▶ Interrogare l'istanza di un oggetto di tipo `LocationProvider` per conoscere l'ultima posizione nota determinata dal sistema di geolocalizzazione.
 - ▶ Registrarsi per ottenere aggiornamenti periodici.
 - ▶ Registrare Intent per essere notificati quando il device si trova in prossimità di specifiche posizioni (specificate secondo latitudine e longitudine).

Network-based vs. GPS-based

- Per sviluppare applicazioni location-aware è possibile utilizzare il segnale GPS (**location GPS-Based**) e/o sfruttare le informazioni che provengono dal provider dell'accesso alla rete Internet (**location Network-based**).
- Location GPS-based:
 - ▶ Informazioni molto precisa (entro qualche metro),
 - ▶ Può essere utilizzato quasi esclusivamente in ambienti outdoor,
 - ▶ Elevato consumo della batteria,
 - ▶ Non fornisce una risposta veloce sulla posizione dell'utente.
- Location Network-based
 - ▶ Garantisce un funzionamento sia indoor che outdoor,
 - ▶ Ha un minore consumo di batteria,
 - ▶ Garantisce tempi di risposta molto più brevi,
 - ▶ Meccanismo molto meno preciso rispetto al precedente.

Richiedere gli aggiornamenti sulla posizione I

- Ottenuta l'istanza per il `LocationManager` procedendo analogamente a quanto visto per i sensori, è possibile invocare il metodo `requestLocationUpdates()` per registrare un opportuno listener.
- Tale metodo accetta come parametri (in ordine):
 - ▶ Il tipo di localizzazione che si vuole utilizzare, alternativamente scelta tra `LocationManager.NETWORK_PROVIDER` oppure `LocationManager.GPS_PROVIDER`,
 - ▶ il minimo intervallo di tempo, in millisecondi, che deve trascorrere tra un aggiornamento ed il successivo,
 - ▶ il minimo intervallo spaziale, in metri, tra un aggiornamento e il successivo,
 - ▶ l'istanza del listener, di tipo `LocationListener`.

Richiedere gli aggiornamenti sulla posizione II

Esempio – Listener per gli aggiornamenti sulla posizione

```
LocationListener listener = new LocationListener(){  
    public void onLocationChanged(Location location){  
        makeUseOfNewLocation(location);  
    }  
  
    public void onStatusChanged(String provider, int status, Bundle  
        extras) { /* ... */ }  
    public void onProviderEnabled(String provider) { /* ... */ }  
    public void onProviderDisabled(String provider) { /* ... */ }  
};  
  
LocationManager lm = (LocationManager) getSystemService(  
    Context.LOCATION_SERVICE);  
  
lm.requestLocationUpdates(LocationManager.GPS_PROVIDER, 0, 0,  
    listener);
```

Richiedere l'ultima posizione rilevata

- Spesso può essere necessario conoscere l'ultima posizione rilevata dal servizio di localizzazione senza voler essere notificati ogni qualvolta si rilevi una nuova posizione.
- in questo caso è possibile ottenere l'ultima posizione nota direttamente dall'istanza del `LocationManager`.

Esempio – Ottenere l'ultima posizione rilevata

```
LocationManager lm = ...

String lp = LocationManager.NETWORK_PROVIDER;
//or LocationManager.GPS_PROVIDER

Location lastKnownLocation = lm.getLastKnownLocation(lp);
```

Interrompere gli aggiornamenti

- Anche in questo caso è bene interrompere gli aggiornamenti quando l'Activity interessata a riceverli viene spostata in background, oppure quando tali aggiornamenti non sono più necessari.
- Può essere richiamato il metodo `removeUpdates()`, sull'istanza del `LocationManager`, al quale deve essere passato il riferimento al listener dedicato.

Esempio – Interruzione delle notifiche

```
LocationManager lm = ...  
LocationListener listener = ...  
  
lm.removeUpdates(listener);
```

Permessi per l'accesso ai servizi di localizzazione

- Per accedere ai servizi di localizzazione è necessario specificare alcuni permessi nel File Manifest dell'applicazione.

Esempio – Permessi

```
<uses-permission  
    android:name="android.permission.ACCESS_COARSE_LOCATION"/>  
<uses-permission  
    android:name="android.permission.ACCESS_FINE_LOCATION"/>
```

- In particolare l'accesso “coarse” serve per l'uso della localizzazione network-based mentre l'accesso “fine” serve per richiedere la localizzazione basata su GPS.
- Nel caso si vogliano utilizzare entrambe le localizzazioni in maniera combinata è sufficiente specificare solo il secondo permesso (che include il primo).

Nota sui Permessi

- A partire dalla versione di Android 6.0 (API Level 23) i permessi sono stati suddivisi in due categorie (a seconda che si richieda l'accesso a feature che coinvolgano o meno la “privacy” dell'utente):
 - ▶ **NORMAL**, devono essere richiesti mediante dichiarazione esplicita nel file manifest
 - ▶ **DANGEROUS**, devono essere richiesti mediante dichiarazione esplicita nel file manifest e l'utente deve esplicitamente accettare tale richiesta

Esempi di Permessi NORMAL

ACCESS_NETWORK_STATE, BLUETOOTH, BLUETOOTH_ADMIN, INTERNET, NFC, ...

Esempi di Permessi DANGEROUS

CAMERA, ACCESS_FINE_LOCATION, ACCESS_COARSE_LOCATION, RECORD_AUDIO, READ_CONTACTS, READ_EXTERNAL_STORAGE, WRITE_EXTERNAL_STORAGE, ...

Permessi DANGEROUS

- Il sistema genera una `SecurityException` se si utilizza codice che richiede un permesso DANGEROUS per essere eseguito, senza aver preventivamente verificato se l'utente abbia o meno concesso esplicitamente tale permesso
- Nel caso in cui l'utente non abbia concesso tale permesso è possibile richiederne l'attivazione direttamente all'utente
 - ▶ Il sistema è in grado di notificare l'applicazione in merito al fatto che l'utente abbia concesso o ignorato la richiesta di utilizzo del permesso

» <https://developer.android.com/training/permissions/requesting>

Richiesta per Permessi DANGEROUS I

Esempio relativo al permesso ACCESS_FINE_LOCATION

```
final int ACCESS_FINE_LOCATION_REQUEST = 1234;

int permission = ContextCompat.checkSelfPermission(this,
    Manifest.permission.ACCESS_FINE_LOCATION);

if (permission == PackageManager.PERMISSION_GRANTED){
    //OK, user has confirmed the permission
} else {
    ActivityCompat.requestPermissions(this,
        new String[] {
            Manifest.permission.ACCESS_FINE_LOCATION
        },
        ACCESS_FINE_LOCATION_REQUEST);
}
```





Richiesta per Permessi DANGEROUS II

Callback di Notifica

```
@Override
public void onRequestPermissionsResult(int reqCode, String
    permissions[], int[] res) {

    switch (reqCode) {
        case ACCESS_FINE_LOCATION_REQUEST:
            // If request is cancelled, the result arrays are empty
            if (res.length > 0 && res[0] == PackageManager.
                PERMISSION_GRANTED){
                // permission was granted!
            } else {
                // permission denied!
            }
            break;
    }
}
```

Riferimenti - Risorse Online

-  **Android Developers - Guide**
» <https://developer.android.com/guide/>
-  **Android Developers - API Reference**
» <https://developer.android.com/reference/>
-  **Android Developers - Samples**
» <https://developer.android.com/samples/>
-  **Android Developers - Design & Quality**
» <https://developer.android.com/design/>

Riferimenti - Libri

-  Zigurd Mednieks, Laird Dornin, G. Blake Meike, Masumi Nakamura
Programming Android
O'Reilly, 2011
-  Chris Haseman, Kevin Grant
Beginning Android Programming: Develop and Design
Peachpit Press, 2013
-  Ronan Schwarz, Phil Dutson, James Steele, Nelson To
The Android Developer's Cookbook : Building Applications with the Android SDK
Addison-Wesley, 2013
-  Theresa Neil
Mobile Design Pattern Gallery: UI Patterns for Smartphone App
O'Reilly, Second Edition, 2014