Universidade da Beira Interior
Faculdade de Engenharia
Departamento de Informática

Programação de Dispositivos Móveis
Aula 8
Licenciatura em Engenharia Informática
Licenciatura em Informática Web

Programming of Mobile Devices
Lecture 8
Degree in Computer Science and Engineering
Degree in Web Informatics

Sumário
Acesso a sensores em dispositivos móveis. Discussão da *framework* que abstrai esse acesso na plataforma Android™, mais especificamente através das classes `SensorManager`, `Sensor` e `SensorEvent`, bem como da interface `SensorEventListener`.

Summary
Access to sensors in mobile devices. Discussion of the framework that abstracts the access to sensors in the Android™ platform, namely through the `SensorManager`, `Sensor` and `SensorEvent` classes, as well as via the `SensorEventListener` interface.

1 Sensores em Dispositivos Móveis
Sensors in Mobile Devices

1.1 Introdução
Introduction

A **maior parte dos dispositivos móveis de hoje integram sensores** que **medem o movimento, a localização, a orientação e vários parâmetros ambientais, como a temperatura ou a humidade**. Os valores devolvidos por estes sensores podem, aparte algumas limitações, **ser usados no âmbito de aplicações móveis**, de modo a tornar a **experiência mais pessoal**, **transparente**, **simples e integrada** para o utilizador:

- **Pessoal**, porque determinada aplicação pode **mostrar dados que têm a ver com o ambiente** onde o utilizador está (e.g., uma aplicação mostra os restaurantes que estão na redondeza) **ou com o que o utilizador está a fazer** (e.g., uma aplicação faz sugestões para abrandar ou acelerar o ritmo quando o utilizador está a correr);
- **Transparente**, porque determinada aplicação pode **ajustar automaticamente o *layout* ou decidir fluxos de execução** de acordo com alguns valores dos sensores (e.g., estiver frio, uma aplicação de sugestão de compras pode escolher não mostrar roupa para tempo quente nesse dia);
- **Simples e integrada**, porque determinada aplicação pode usar alguns sensores para **tornar a navegação mais intuitiva ao utilizador** (e.g., um jogo de corridas pode usar o sensor de orientação para permitir a condução do veículo, evitando a utilização de botões para o mesmo efeito);

1.2 Sensores em Android™
Sensors in Android™

A **plataforma Android™**[1] **suporta três categorias principais de sensores**, estruturadas da seguinte forma:

1. **Sensores de Movimento**, que **incluem os sensores que medem forças de aceleração e rotacionais em três eixos**. É nesta categoria que são **incluídos os acelerómetros, giroscópios, sensores de gravidade e os rotacionais**;
2. **Sensores de Ambiente**, que **incluem os sensores que medem parâmetros de ambiente**, como a temperatura, a pressão do ar, a humidade ou a iluminação. Aqui se **incluem os barómetros, fotómetros e termómetros**;
3. **Sensores de Posicionamento**, que medem/identificam **a posição física dos dispositivos, e que incluem o *Global Positioning Sensor* (GPS), sensores de orientação e os magnetómetros**.

Como seria de esperar, a plataforma Android™ permite **um acesso bastante simplificado** a todos os sensores disponíveis no dispositivo móvel a **partir da *framework* de sensores Android™**. Esta *framework* **contém classes e interfaces** que podem ser usadas para **adquirir os valores em estado bruto** desses sensores, bem como **outras funcionalidades**, nomeadamente **registo de recetores de eventos** para sensores em particular.

Na verdade, a explicação referente aos sensores constitui também uma boa oportunidade para referir (e relembrar) **a forma padrão** de como bastantes funcionalidades da plataforma Android™ podem ser conseguidas. Já

[1] A explicação subsequente é sobretudo inspirada na documentação oficial da plataforma Android™ sobre este assunto, nomeadamente na discussão contida no URL `http://developer.android.com/guide/topics/sensors/sensors_overview.html`.

antes foi referido o facto do sistema operativo executar automaticamente **vários gestores, que mais não são dos que aplicações que vêm nativas com o sistema**, situadas na parte da ***framework* aplicacional da pilha da plataforma** (ver aula 3). **Não é tipicamente possível aceder diretamente a um sensor** ou, de uma forma geral, a qualquer recurso para o qual haja um gestor disponível. O procedimento elabora em:

1. **Declarar um objeto da classe do gestor** pretendido na aplicação (mas **não instanciá-lo** diretamente);
2. **Pedir** a instância do gestor **ao sistema**;
3. **Aceder a funcionalidades disponibilizadas pelo gestor**, como obter valores, enviar mensagens, registar ou eliminar o registo para um recetor.

Note que este procedimento já foi antes discutido, aquando da menção ao gestor de notificações. Nesse caso, obtinha-se uma instância do gestor pedindo-o ao sistema através do método `getSystemService(Context.NOTIFICATION_SERVICE)`, **fornecido com o contexto**. Depois, era invocado um dos métodos do gestor para lhe entregar uma mensagem (nomeadamente o método `notify()`):

```
...
NotificationManager oNM = (NotificationManager)
  getSystemService(Context.NOTIFICATION_SERVICE);
oNM.notify(iID, oBuilder.build());
```

O uso dos sensores será, portanto, semelhante.

No caso específico dos sensores, a *framework* fornece as seguintes funcionalidades:

- Permite **determinar que sensores estão disponíveis** no dispositivo;
- Permite **obter as capacidades de cada sensor**, tal como o fabricante, os requisitos de energia, resolução e alcance;
- Permite **adquirir os dados em bruto e definir**, em alguns casos, **a cadência com que os dados devem ser adquiridos**; e
- Permite **registar ou eliminar o registo de *listeners* para determinado tipo de eventos** relacionados com sensores, e forma a detetar mudanças nos seus valores.

Os sensores disponíveis num dispositivo com Android™ **podem ainda ser divididos em dois tipos principais**, embora tal facto não mude a forma como estes são utilizados, **nomeadamente *hardware-* e *software-based sensors***. No primeiro caso, os valores devolvidos pelo sensor são **derivados diretamente de um dispositivo físico**, enquanto que, para o segundo caso, **os valores são derivados de um ou mais dispositivos físicos, podendo ser tratados antes de devolvidos**. O **acelerómetro ou magnetómetro são exemplos de sensores de** `hardware`, enquanto que **o sensor gravitacional constitui um exemplo do segundo**. Note que **nem todos os dispositivos integram todos os sensores suportados** pela plataforma, e que novos sensores podem vir a ser adicionados com o tempo. Por exemplo, **a maior parte dos dispositivos móveis atuais contêm um giroscópio**, mas **poucos integram um termómetro ou um barómetro**.

1.3 Constituição da *Framework* de Sensores

Constitution of the Sensor Framework

A *framework* de sensores é constituída por três classes principais e uma interface, disponibilizadas no **pacote** `android.hardware`:

1. A `SensorManager` (Gestor de Sensores), que é **a classe que encapsula os métodos que podem ser usados para interagir com o gestor em execução** no sistema, e que **constitui o ponto de partida para aceder e listar sensores**, ou registar escutas para os mesmos.
2. A `Sensor`, que é **a classe que permite criar uma instância de um sensor específico** no âmbito da aplicação, e que **fornece os vários métodos que podem ser usados para verificar as características do sensor** instanciado. Enquanto que **uma instância da classe referida no ponto anterior permite verificar se determinado sensor está disponível no sistema, é esta classe que permite**, por exemplo, **verificar qual é que é a marca ou versão de determinado sensor**, ou a energia (em mA) que este utiliza, entre outras. Note que **não é um objeto da classe** `Sensor` **que irá permitir obter valores** dos sensores;
3. A `SensorEvent`, que é **a classe que permite**, no fundo, **obter valores em estado bruto (*raw*) de determinado sensor**. Um evento desta classe **inclui informação como os valores, o tipo de sensor que gerou esses dados, a precisão e o momento** (selo temporal) em que foram adquiridos;
4. Finalmente, a `SensorEventListener` é **a interface que pode ser implementada para automaticamente receber eventos quando os valores ou a precisão dos sensores mudam**.

Para já, fica claro que, para poder usar sensores numa aplicação, é **primeiro necessário instanciar o gestor de sensores**, que mais não seja para testar se o sensor a utilizar está disponível ou não. O excerto de código seguinte mostra precisamente a forma de instanciar o `SensorManager` e de **apurar se determinado sensor está ou não disponível** no dispositivo móvel onde a aplicação está instalada, **durante a própria execução**:

```
package pt.di.ubi.pmd.exsensors1;

import android.app.Activity;
import android.hardware.SensorManager;
import android.hardware.Sensor;
import android.widget.Toast;

public class SensorActivity extends Activity {
  private SensorManager oSM;

  @Override
  public final void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    oSM = (SensorManager)
      getSystemService(Context.SENSOR_SERVICE);
    if(oSM.getDefaultSensor(Sensor.TYPE_LIGHT) != null){
      Toast.makeText(
        this,
        "You HAVE a light sensor!",
        Toast.LENGTH_SHORT
        ).show();
    }
    else{
      Toast.makeText(
        this,
        "You DO NOT HAVE a light sensor!",
        Toast.LENGTH_SHORT
        ).show();
    }
  }
}
```

Repare nos **dois *imports* necessários** à compilação correta da aplicação e também na forma de instanciar um `SensorManager`, dado por:

```
private SensorManager oSM;
...
oSM = (SensorManager)
    getSystemService(Context.SENSOR_SERVICE);
```

A aplicação exemplificada antes, se executada, mostra uma mensagem *toast* a dizer `You HAVE a light sensor!` caso este sensor esteja disponível; ou `You DO NOT HAVE a light sensor!` no caso contrário.

Caso se queira **obter uma lista de todos os sensores disponíveis no sistema** em que a aplicação está instalada, pode recorrer-se ao **método `getSensorList(Sensor.TYPE_ALL)`**, conforme se mostra a seguir, que devolve um objeto da classe `List`:

```
List<Sensor> lDS = mSensorManager.getSensorList(
    Sensor.TYPE_ALL);
```

A **declaração da lista anterior faz uso de *genéricos* em Java**, que basicamente **permitem comunicar ao compilador o tipo dos objetos que esta classe (`List`) vai conter**. Significa que sempre que um objeto for obtido desta lista, e.g., fazendo `lDS.get(1)`, **já não será necessário fazer *cast* do mesmo para que este seja considerado um objeto da classe `Sensor`**. Note que uma `List` pode albergar objetos de quaisquer classe, ou até vários objetos de classes diferentes simultaneamente (o que não é o caso). No exemplo dado, já é sabida a classe dos objetos que irá guardar, pelo que é **mais seguro e cómodo defini-los imediatamente recorrendo à notação que usa parêntesis angulares**.

Atente ainda no código da Atividade mostrada antes, nomeadamente na linha com `oSM.getDefaultSensor(Sensor.TYPE_LIGHT)`. **O método `getDefaultSensor(.)` devolve (ou tenta devolver) um objeto da classe `Sensor`**, embora não tenha sido usado explicitamente no exemplo. **Caso** o sensor específico **não exista, o método devolve `null`, caso exista um ou mais, o método devolve o primeiro que encontrar**. Nas ocasiões em que existe mais do que um sensor de determinado tipo (e.g., sensor de luz), pode usar-se o método `getSensorList()`, como antes, mas especificando o tipo, em vez de os requisitar a todos:

```
List<Sensor> lDS = mSensorManager.getSensorList(
    Sensor.TYPE_LIGHT);
```

A documentação oficial do Android™ constituirá sempre o melhor recurso para se obter a ideia de quais os sensores suportados pela plataforma[2]. Contudo, para referência, inclui-se a seguir **uma lista de alguns dos tipos de sensores mais importantes e disponíveis atualmente**:

- `TYPE_ACCELEROMETER`, *hardware-based*, que **mede a aceleração em m/s^2 aplicada ao dispositivo em <u>três</u> eixos** (x, y, z) e que pode ser usado para **detetar movimento, vibrações**, etc.;
- `TYPE_AMBIENT_TEMPERATURE`, *hardware-based*, que **mede a temperatura ambiente** em graus Celsius (žC);
- `TYPE_GRAVITY`, *hardware-* ou ***software**-based*, que **mede a força da gravidade** em m/s^2 aplicada a cada um dos eixos (x, y, z), e que também pode ser usado para detetar movimento, vibrações, etc.;
- `TYPE_GYROSCOPE`, *hardware-based*, que **mede o rácio de rotação em rad/s para cada um dos <u>três</u> eixos** (x, y, z), e que pode ser usado para **detetar movimento** em termos de rotação;
- `TYPE_LIGHT`, *hardware-based*, que **mede a intensidade da luz no ambiente (iluminação)** em lx, e que pode ser usado, por exemplo, para **controlar a luminosidade do ecrã**;
- `TYPE_LINEAR_ACCELERATION`, *hardware-* ou ***software**-based*, que aceleração em m/s^2 aplicada a cada um dos eixos (x, y, z), mas **excluindo a força da gravidade**;
- `TYPE_MAGNETIC_FIELD`, *hardware-based*, que **mede o campo eletromagnético do ambiente** em que o dispositivo se insere para **cada um dos eixos** (x, y, z);
- `TYPE_PRESSURE`, *hardware-based*, que mede **a pressão do ar** em mbar;

[2]Ver `http://developer.android.com/reference/android/hardware/Sensor.html`.

- `TYPE_PROXIMITY`, *hardware-based*, que que mede **a proximidade de um objeto**, em cm, relativamente ao local onde o sensor físico está colocado (normalmente está colocado na parte superior do ecrã). Este sensor é, por exemplo, usado por **aplicações que querem saber se o telefone está junto ao ouvido** ou não (e.g., a aplicação de gestão de chamadas de voz pode fazer uso desta funcionalidade para **desativar o ecrã tátil**, evitando que sejam pressionadas funcionalidades acidentalmente);

- `TYPE_RELATIVE_HUMIDITY`, *hardware-based*, que m**ede a humidade relativa do ar no ambiente** em que o dispositivo móvel se insere, **em percentagem (%)**. Os valores devolvidos por este sensor são tipicamente **usados para determinar o ponto de orvalho**.

Repare que **os vários tipos de sensores estão definidos, por comodidade, como *Strings* estáticas na classe** `Sensor`.

1.4 A Classe `Sensor`

The `Sensor` Class

O método `getDefaultSensor(Sensor.TYPE_LIGHT)`, já incluído no exemplo da secção anterior, devolve uma instância da classe `Sensor` que pode ser atribuída, caso o objeto haja sido declarado antes. O exemplo seguinte mostra precisamente essa atribuição, elaborando no que já foi discutido, e adicionando mais detalhes discutidos em baixo:

```
package pt.di.ubi.pmd.exsensors1;

import android.app.Activity;
import android.hardware.SensorManager;
import android.hardware.Sensor;
import android.util.Log;

public class SensorActivity extends Activity {
  private SensorManager oSM;
  private Sensor oL;

  @Override
  public final void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    oSM = (SensorManager)
      getSystemService(Context.SENSOR_SERVICE);
    oL = oSM.getDefaultSensor(Sensor.TYPE_LIGHT);
    if(oL != null){
      String sName = oL.getName();
      String sVendor = oL.getVendor();
      float fP = oL.getPower();
      int iVersion = oL.getVersion();
      Log.i("SENSORACTIVITY",
        "Sensor found! Specs - Name=" + sName +
        " Vendor=" + sVendor +
        " Power=" + fP +
        " Version=" + iVersion
        );
    }
    else{
      Log.e("SENSORACTIVITY","Sensor not found!");
    }
  }
}
```

Como poderá reparar, desta feita, o objeto `oL` foi declarado como sendo da classe `Sensor`, e privado à `SensorActivity`, para depois lhe ser atribuído o *output* do método `getDefaultSensor(.)`. Para melhorar a legibilidade do código e poupar espaço, neste exemplo é usada a classe `Log`, ao invés da `Toast`. Caso esta aplicação seja executada num dispositivo Android™ com sensor de luminosidade, o código guardado por `(oL != null)` é executado, sendo impressos no *logcat* algumas das especificações do sensor, nomeadamente **a *String* que concretiza o seu nome, o nome do fabricante, a versão e a energia que o sensor consome**. Conforme prometido, é **o objeto da classe `Sensor` que permite obter estas informações, mas não os valores em bruto que são por ele medidos**.

Outros dois métodos que a classe disponibiliza e que podem ser **úteis em determinados contextos são**:

- `getMaxDelay()`, que **devolve o tempo máximo, em microssegundos, entre dois eventos do sensor** (apenas definido para sensores que fazem medições de forma contínua);

- `getResolution()`, que **devolve a precisão do sensor na unidade para a qual está definido**.

Enquanto que a **utilidade desta classe** ainda não está suficientemente explicita neste ponto, é possível elaborar um pouco mais neste aspeto imediatamente. O facto é que **as aplicações podem ser otimizadas para diferentes tipos de sensores ou até para fabricantes, bem como para a precisão de cada um**. Assim, pode ser **útil obter estes dados em tempo de execução, e redirecionar o fluxo de acordo** com os mesmos, e antes de usar os seus valores. Para concretizar esta discussão com um exemplo, pode imaginar-se uma aplicação que precisa saber se o utilizador agita o telemóvel em determinada situação. Este evento pode ser detetado por um sensor de gravidade (melhor solução) ou por um acelerómetro, e o código terá de ser diferente na presença de um ou de outro, pelo que convém testar qual deles está disponível. Em alguns casos, a versão ou fabricante de determinado sensor, bem como a sua precisão, podem também ser cruciais para os objetivos da aplicação. E.g., a experiência de utilização de um jogo de condução que use o sensor de rotação será tanto melhor quanto mais preciso este for.

1.5 A Interface `SensorEventListener`

The `SensorEventListener` Interface

Até aqui, a explicação focou-se na obtenção do gestor de sensores e na verificação da existência e características destes últimos. Esta secção elabora em como se podem efetivamente obter valores dos mesmos.

A **obtenção de valores de sensores requer sempre que se registe um *Listener* para o sensor do tipo desejado** (e.g., `TYPE_LIGHT`) **através do método**

`registerListener()`, **providenciado pelo gestor de sensores** (i.e., pelo `SensorManager`). O *Listener* que é registado **é um objeto de uma classe que obrigatoriamente implementa a interface** `SensorEventListener`, cuja definição determina também que **os métodos** `onAccuracyChanged()` **e** `onSensorChanged()` **estejam concretizados neste objeto**. De uma forma geral, pode estruturar-se o procedimento da seguinte forma:

1. **Declaram-se os objetos** das classes `SensorManager` e `Sensor`;
2. **Obtém-se a instância do Gestor de Sensores** a correr no sistema conforme discutido acima;
3. **Obtém-se a instância do sensor pretendido**, através do método `getDefaultSensor(int)`, também como já foi discutido antes;
4. **Cria-se uma nova classe que implemente a interface** `SensorEventListener` **ou**, alternativamente, **define-se que a própria Atividade implementa esta classe** (neste caso, os métodos são definidos dentro da classe que estende a `Activity` – ver exemplo seguinte);
5. **Implementam-se os dois métodos da interface** `SensorEventListener`, nomeadamente o `onAccuracyChanged()` e o `onSensorChanged()`;
6. **Instancia-se um novo objeto da classe referida** (se o objeto for a própria Atividade, este passo não é necessário);
7. Usa-se o método `registerListener()`, disponível no objeto da classe `SensorManager`, para **registar o consumidor** (*Listener*).

Para além dos passos enunciados, não deve ser esquecido que é também necessário importar todas as classes necessárias, bem como a interface.

O método `registerListener(.,.,.)` **aceita 3 parâmetros** de entrada: **um objeto da classe que implementa** `SensorEventListener`**, o objeto da classe** `Sensor` **e um inteiro, que determina a frequência com que o sistema deve tentar entregar os eventos do sensor a esta aplicação**. Este último valor é meramente indicativo, já que os eventos podem depois ser entregues mais ou menos rápido, embora o sistema tente responder com a celeridade pedida. Este método **devolve verdadeiro caso o registo tenha sido bem sucedido, e falso no caso contrário**.

Os **dois métodos da interface** `SensorEventListener` são os que irão conter **a lógica computacional que permite atuar sobre os valores do sensor**, e **é o próprio sistema Android™ que os invoca** aquando da entrega dos valores ou alterações nos sensores:

- **O método** `onAccuracyChanged(.,.)` **recebe dois parâmetros e é invocado sempre que a precisão do sensor sofre alteração** (e.g., por ter sido calibrado). O primeiro parâmetro é um objeto da classe `Sensor`, e define o próprio sensor em que se deu a alteração, enquanto que **o segundo parâmetro é um inteiro que informa a nova precisão**. O inteiro pode ser um dos 3 valores definidos estaticamente na classe `SensorManager`, nomeadamente `SENSOR_STATUS_ACCURACY_HIGH`, `SENSOR_STATUS_ACCURACY_LOW` ou `SENSOR_STATUS_ACCURACY_MEDIUM`;
- **O método** `onSensorChanged(.)` **recebe apenas um parâmetro e é invocado quando o sensor reporta novos valores**. Em algumas APIs, este método também é invocado quando o *Listener* é registado, para se obter uma leitura imediata dos valores. **O parâmetro é um objeto da classe** `SensorEvent`, já discutido anteriormente. Normalmente é possível **obter os valores em bruto do sensor acedendo ao *array*** `values` **desse** `SensorEvent` (e.g., `event.value[0]` corresponde ao valor medido no eixo dos xx no sensor giroscópio).

De maneira a concretizar melhor o procedimento desrito antes, inclui-se, em baixo, um exemplo de uma aplicação Android™ com uma única Atividade que implementa os métodos da interface na própria Atividade, após o qual se elabora nos detalhes mais importantes para o entendimento deste assunto:

```
package pt.di.ubi.pmd.exsensors2;

import android.app.Activity;
import android.hardware.SensorManager;
import android.hardware.Sensor;
import android.hardware.SensorEventListener;
import android.hardware.SensorEvent;
import android.util.Log;

public class SensorActivity extends Activity
    implements SensorEventListener {
  private SensorManager oSM;
  private Sensor oL;
  private boolean bMessage = false;

  @Override
  public final void onCreate(Bundle
      savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    oSM = (SensorManager) getSystemService(Context.
        SENSOR_SERVICE);
    oL = oSM.getDefaultSensor(Sensor.TYPE_LIGHT);
  }

  @Override
  public final void onAccuracyChanged(Sensor sensor,
      int accuracy) {
    Toast.makeText(
        this,
        "Accuracy has changed!",
        Toast.LENGTH_SHORT
        ).show();
  }

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

```
    // 0.27 -- 1.0 lux Full moon on a clear night
    if( ( lux < 1.0 ) && (lux > 0.27))
      if( !bMessage ){
        Toast.makeText(
            this,
            "What a beautiful full moon!",
            Toast.LENGTH_SHORT
            ).show();
        bMessage = true;
      }
  }

  @Override
  protected void onResume() {
    super.onResume();
    mSensorManager.registerListener(this, oL,
        SensorManager.SENSOR_DELAY_NORMAL);
  }

  @Override
  protected void onPause() {
    super.onPause();
    oSM.unregisterListener(this);
  }
}
```

O funcionamento desta aplicação é simples de explicar. A Atividade regista um *Listener* para o sensor do tipo `TYPE_LIGHT`. **Na primeira vez que** os valores da luminosidade no sensor mudam para entre 0.27 e 1.0, a aplicação lança uma mensagem `Toast` a dizer `"What a beautiful full moon!"`. Para evitar que esta mensagem apareça mais vezes, uma variável de controlo (`bMessage`) é então colocada a `true`, impedindo que tal se repita.

Note que **foi decidido**, neste exemplo, **que seria a própria Atividade** `SensorActivity` **a implementar a interface** `SensorEventListener`. Assim, **os dois métodos da interface estão definidos logo a seguir ao** `onCreate()`, e **os métodos** `registerListener(.,.,.)` **e** `unregisterListener(.)` **aceitam**, como parâmetro, **a *keyword* `this`, que lhes injeta a própria classe que os invoca**. Dado que a classe `SensorActivity` não pode ser, dado o seu objetivo, abstrata (`Abstract`), esta **é obrigada a implementar ambos os métodos da interface**, ainda que pudesse deixar um deles vazio. Por exemplo, a implementação do método `onSensorChanged()` seguinte era válida:

```
@Override
public final void onAccuracyChanged(Sensor sensor,
    int accuracy) {
}
```

Também de enfatizar são os locais onde o *Listener* é registado ou onde o seu registo é eliminado. No exemplo, a primeira operação acontece no método `onResume()`, enquanto que a segunda acontece no ponto simétrico ao primeiro, i.e., no método `onPause()`. **É importante que a aplicação contenha os métodos tanto para ativar, como para desativar os sensores nos locais certos, e recomendado que esta os desative quando não forem necessários**, já que **o seu funcionamento consome energia**. Na verdade, **o Android™ não desativa automaticamente os sensores quando uma aplicação é pausada ou mesmo quando o ecrã é desligado**, já que estes podem ser necessários por alguma aplicação. No exemplo anterior, e visto que não necessitarmos do sensor enquanto a aplicação está pausada ou parada, o seu registo é eliminado no método `onPause()`.

O **método de eliminação de registo** ainda não havia sido mencionado anteriormente:

```
private SensorManager oSM;
  ...
@Override
protected void onPause() {
  super.onPause();
  oSM.unregisterListener(this);
}
```

O `unregisterListener(.)`, também providenciado pela classe `SensorManager`, **aceita, como único parâmetro de entrada, o objeto que implementa a interface** `SensorEventListener` (que, em cima, era a própria Atividade, logo o uso da *keyword* `this`).

1.6 A Classe `SensorEvent`
The `SensorEvent` Class

A classe `SensorEvent` **não disponibiliza qualquer método para além dos que herda da sua superclasse** (`Object`), contudo, **todos os seus atributos são públicos**, e é dessa forma que os disponibiliza à chegada, no método `onSensorChanged()`. Os **4 atributos** dos objetos desta classe são:

- `accuracy`, um inteiro (`int`) que determina a **precisão** da captura;
- `sensor`, um objeto da classe `Sensor` que identifica **o sensor onde se deu a captura**;
- `timestamp`, um inteiro (`long`) que representa **o momento, em nanossegundos**, em que se seu a captura;
- `values`, um vetor de decimais (`floats`) com **os valores da captura**.

O **tamanho e conteúdo do vetor de valores depende do tipo de sensor que está a ser usado**, e convém verificar a documentação para se saber quantos valores esperar. Para o sensor da luminosidade, por exemplo, o tamanho do vetor é 1, enquanto que para o rotacional ou gravitacional teria um tamanho de 3.

Convém referir ainda a forma como o sistema define os três eixos para sensores que medem valores para 3 dimensões. **O sistema define como o eixo dos xx como o que segue a linha horizontal do ecrã** (i.e., o lado mais

pequeno quando o dispositivo móvel está em modo retrato, ou o lado maior quando este está em modo paisagem); define **o eixo dos yy como aquele que segue a linha vertical do ecrã** (i.e., o lado maior quando o dispositivo móvel está em modo retrato, ou o lado menor quando este está em modo paisagem); e, finalmente, define **o eixo dos zz como sendo aquele que aponta para o céu quando o dispositivo está deitado com as costas viradas para o chão**.

1.7 Testar a Implementação
Testing the Implementation

Note que **não é normalmente boa ideia testar as aplicações que utilizem sensores em emuladores Android™**, já que **estes não simulam**, tipicamente ou em toda a plenitude, **os *outputs* que aqueles produzem** de uma forma satisfatória. Neste caso, **o ideal será testar a aplicação num dispositivo físico** ou, opcionalmente, certificar-se previamente que o dispositivo virtual que vai utilizar está **apetrechado com simuladores de sensores** adequados ao seus testes.

Nota: o conteúdo exposto na aula e aqui contido não é (nem deve ser considerado) suficiente para total entendimento do conteúdo programático desta unidade curricular e deve ser complementado com algum empenho e investigação pessoal.