AGENDA 7

SENSORES

GEEaD - Grupo de Estudos de Educação a Distância

Centro de Educação Tecnológica Paula Souza

GOVERNO DO ESTADO DE SÃO PAULO

EIXO TECNOLÓGICO DE INFORMAÇÃO E COMUNICAÇÃO

CURSO TÉCNICO EM DESENVOLVIMENTO DE SISTEMAS

PROGRAMAÇÃO MOBILE I

Expediente

Autor:

GUILHERME HENRIQUE GIROLLI

Revisão Técnica:

Eliana Cristina Nogueira Barion

Revisão Gramatical:

Juçara Maria Montenegro Simonsen Santos

Editoração e Diagramação:

Flávio Biazim

São Paulo – SP, 2020

MERGULHANDO NO TEMA...

Tipos de sensores

Os sensores são componentes de hardware e/ou software que permitem ao dispositivo uma interação com o ambiente. Capturando dados, analisando e devolvendo informações para o aparelho e para o usuário.

Encontramos sensores a nível de hardware, ou seja, sensores que são componentes eletrônicos físicos para capturar determinada informação, como por exemplo, a temperatura ou até mesmo a iluminação de um determinado ambiente.

Existem também sensores baseados em software ou sensores virtuais, ou seja, são sensores que para entregar um determinado dado, captura informações de um ou mais sensores de hardware, processa essas informações para entregar um resultado. Um exemplo é o sensor de aceleração linear.

O sistema operacional Android suporta a grande maioria dos sensores encontrados nos aparelhos, e é importante comentar que os aparelhos não possuem todos os tipos de sensor, isso depende muito do fabricante e da versão do dispositivo.

A seguir a imagem 1, retirada do site oficial do Android Studio, mostra os tipos de sensor suportados pelo sistema operacional, com sua classificação de tipos e breve explicação sobre seu funcionamento. Para examinar o conteúdo completo sobre sensores acesse o site: https://developer.android.com/guide/topics/sensors/sensors_overview

Sensor	Tipo	Descrição	Usos Comuns
TYPE_ACCELEROMETER	Hardware	Mede a força de aceleração em m / s ^{2} que é aplicada a um dispositivo em todos os três eixos físicos (x, yez), incluindo a força da gravidade.	Detecção de movimento (agitar, inclinar, etc.)
TYPE_AMBIENT_TEMPERATURE	Hardware	Mede a temperatura ambiente em graus Celsius (° C). Ver nota abaixo.	Monitorando a temperatura do ar.
TYPE_GRAVITY	Software ou Hardware	Mede a força da gravidade em m / s ^{2} que é aplicada a um dispositivo em todos os três eixos físicos (x, y, z).	Detecção de movimento (agitar, inclinar, etc.)
TYPE_GYROSCOPE	Hardware	Mede a taxa de rotação de um dispositivo em rad / s em torno de cada um dos três eixos físicos (x, y e z).	Detecção de rotação (girar, girar, etc.)
TYPE_LIGHT	Hardware	Mede o nível de luz ambiente (iluminação) em lx.	Controlando o brilho da tela.
TYPE_LINEAR_ACCELERATION	Software ou Hardware	Mede a força de aceleração em m / s ^{2} que é aplicada a um dispositivo em todos os três eixos físicos (x, yez), excluindo a força da gravidade.	Monitorando a aceleração ao longo de um único eixo.
TYPE_MAGNETIC_FIELD	Hardware	Mede o campo geomagnético do ambiente para todos os três eixos físicos (x, y, z) em μT.	Criando uma bússola.
TYPE_ORIENTATION	Programas	Mede os graus de rotação que um dispositivo faz em torno dos três eixos físicos (x, y, z). A partir do nível API 3, você pode obter a matriz de inclinação e a matriz de rotação de um dispositivo usando o sensor de gravidade e o sensor de campo geomagnético em conjunto com o `getRotationMatrix()` método.	Determinando a posição do dispositivo.
TYPE_PRESSURE	Hardware	Mede a pressão do ar ambiente em hPa ou mbar.	Monitorando as mudanças de pressão de ar.
TYPE_PROXIMITY	Hardware	Mede a proximidade de um objeto em cm em relação à tela de exibição de um dispositivo. Este sensor é normalmente usado para determinar se um fone está sendo conectado ao ouvido de uma pessoa.	Posição do telefone durante uma chamada.
TYPE_RELATIVE_HUMIDITY	Hardware	Mede a umidade relativa do ar em porcentagem (%).	Monitoramento do ponto de orvalho, absoluta e umidade relativa.
TYPE_ROTATION_VECTOR	Software ou Hardware	Mede a orientação de um dispositivo fornecendo os três elementos do vetor de rotação do dispositivo.	Detecção de movimento e detecção de rotação.

Imagem 1 - Tipos de sensores suportados pela plataforma Android. Site: https://developer.android.com/guide/topics/sensors/sensors_overview Acesso em: 5 de dezembro de 2018.

Estrutura do sensor

O sensor do dispositivo é utilizado através de uma estrutura padrão, estabelecida pelo Android. O acesso aos seus dados brutos é feito através de classes Java para facilitar o processo de desenvolvimento de aplicativos. A seguir vamos verificar as principais classes para trabalho com sensores.

SensorManager: É utilizado para criar uma instância dos serviços dos sensores presentes no aparelho, com métodos para facilitar o desenvolvimento do aplicativo. Através dela é possível iniciar e parar o uso do sensor, calibrar, verificar a precisão, entre outras funções.

Sensor: Essa classe é utilizada para trabalhar com métodos de um sensor em específico, obtido através do **SensorManager**.

SensorEvent: É uma classe utilizada para criar um objeto sobre determinado evento do sensor, com os dados, horário, precisão do sensor, qual tipo de sensor foi utilizado, entre outras informações.

SensorEventListener: É utilizada para criar duas ações sobre um determinado sensor que esteja em uso. Essas ações são disparadas quando o valor do sensor que está sendo verificado sofre uma mudança chamado de **onSensorChanged()**, e outra ação que é disparada quando a precisão do sensor sofre uma alteração chamada de **onAccuracyChanged()**.

Boas práticas ao utilizar sensores

Ao utilizar um ou mais sensores, o desenvolvedor precisa levar em consideração alguns fatores, como por exemplo o uso desnecessário da bateria do aparelho. O Android por padrão, não desabilita um sensor quando a tela do dispositivo é desligada, desta forma o desenvolvedor mobile necessita trabalhar com métodos para que o sensor seja desligado quando o mesmo não estiver em uso.

Alguns sensores exigem um alto uso da bateria do aparelho durante seu funcionamento, e deixar esses sensores ativos mesmo não sendo utilizados pode gerar alguns problemas e insatisfação do usuário com o aplicativo. Os métodos utilizados para trabalhar com essas boas práticas são o **onResume()** e o **onPause().**

Para recapitular o que foi visto anteriormente sobre os métodos padrões do Android, o **onPause()** é disparado quando o sistema vai trabalhar com uma atividade anterior ao aplicativo que está em execução, ou seja, ele sai do topo da pilha de execuções. E o **onResume()** é chamado quando o aplicativo vai iniciar a interação com o usuário, ou seja, ele está no topo da pilha de execuções.

Na imagem 2, mostramos um exemplo de desligamento do sensor através do método **onPause().**

```java
@Override
protected void onPause() {
    super.onPause();
    sensorManager.unregisterListener(lightEventListener);
}
```

Imagem 2 – onPause da classe Java.

Na imagem 3, encontramos um exemplo de retomada do uso do sensor através do método **onResume().**

```java
@Override
protected void onResume() {
    super.onResume();
    sensorManager.registerListener(lightEventListener, lightSensor, SensorManager.SENSOR_DELAY_FASTEST);
}
```

Imagem 3 – onResume da classe Java.

Trabalhando com sensores

Primeiramente o desenvolvedor necessita pesquisar a documentação oficial do Android para conhecer a forma correta de trabalho para cada sensor. Por exemplo os sensores de giroscópio utilizam coordenadas para passar os dados referentes a localização do dispositivo. A imagem 4 mostra os eixos x, y e z de um aparelho. Em contrapartida, um sensor de luz, que vamos utilizar para desenvolver o projeto da Ana, utiliza um único dado, para informar a quantidade de iluminação recebida por um dispositivo.

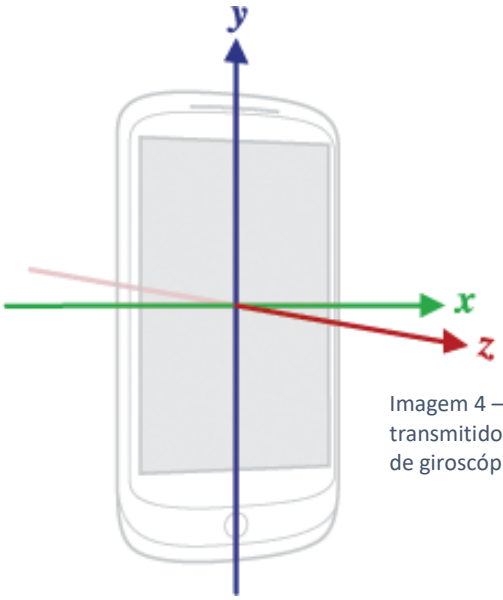


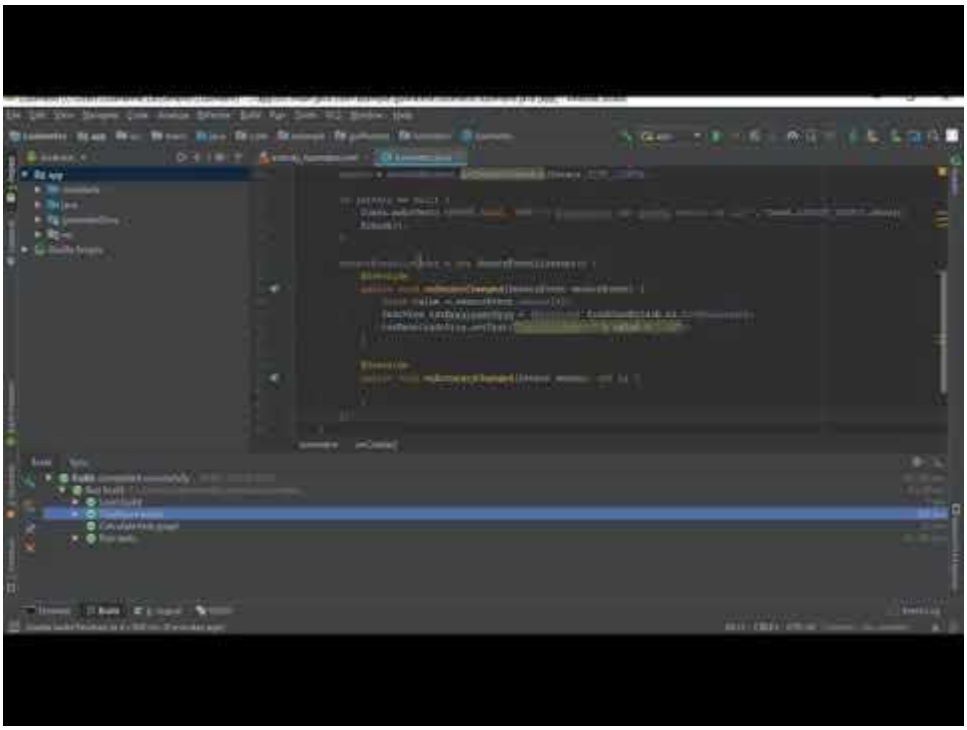
Imagem 4 – Eixos x, y e z transmitidos por um sensor de giroscópio.

A documentação oficial do Android aborda todas as formas de utilização presente no sistema operacional, e está disponível pelo site: https://developer.android.com/guide/topics/sensors/

No projeto recebido por Ana, vamos utilizar um sensor de luz, que captura o valor de iluminação recebida por um determinado dispositivo. Esse sensor também é utilizado para o processo de iluminação da tela, você já deve ter notado que em ambientes escuros, o celular retira

automaticamente o brilho da tela para o conforto do usuário e economizar energia. E em ambientes claros o sistema operacional aumenta o brilho da tela para facilitar a visualização do usuário.

O vídeo a seguir exemplifica a utilização do sensor de luz, vamos colocar em prática o conteúdo adquirido nesse material.



Disponível em: https://youtu.be/r8_NTwasyb4

Confira o código da classe Java do projeto “Luximetro” criado no vídeo anterior.

```
package com.example.guilherme.luximetro;

import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.TextView;
import android.widget.Toast;

public class luximetro extends AppCompatActivity {

    private SensorManager sensorManager;
    private Sensor sensor;
    private SensorEventListener sensorEventListener;

    @Override
```

```
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_luximetro);

        sensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);
        sensor = sensorManager.getDefaultSensor(Sensor.TYPE_LIGHT);

        if (sensor == null) {
            Toast.makeText(this, "O dispositivo não possui sensor de luz!",
Toast.LENGTH_SHORT).show();
            finish();
        }

        sensorEventListener = new SensorEventListener() {
            @Override
            public void onSensorChanged(SensorEvent sensorEvent) {
                float value = sensorEvent.values[0];
                TextView txtResultadoProg = (TextView)
findViewById(R.id.txtResultado);
                txtResultadoProg.setText("Luminosidade: " + value + " lx");
            }

            @Override
            public void onAccuracyChanged(Sensor sensor, int i) {

            }
        };
    }

    @Override
    protected void onResume() {
        super.onResume();
        sensorManager.registerListener(sensorEventListener, sensor,
SensorManager.SENSOR_DELAY_FASTEST);
    }

    @Override
    protected void onPause() {
        super.onPause();
        sensorManager.unregisterListener(sensorEventListener);
    }
}
```