Interação com sensores usando a plataforma Android

Henrique Neves da Silva[1], **Andre Takeshi Endo**[1], **Davi Bernardo Silva**[2]

[1]Universidade Tecnológica Federal do Paraná (UTFPR)
Avenida Alberto Carazzai 1640 – Cornélio Procópio – PR – Brazil

hen123neves@gmail.com, andreendo@utfpr.edu.br

[2]Instituto Federal de Santa Catarina (IFSC)
Avenida Fahdo Thomé, 3000 – 89500-000 – Caçador, SC – Brasil

davi.bernardo@ifsc.edu.br

Abstract. *This material aims to allow the student or teacher to explore the Android technology with a focus on the use of sensors. It is expected that the student has basic knowledge in development for Android using the Java programming language. The method of obtaining knowledge is characterized by being self-taught and experimental.*

Resumo. *Este material tem como principal objetivo permitir ao aluno ou professor explorar a tecnologia Android com enfoque no uso de sensores. Espera-se que o aluno possua conhecimentos básicos em desenvolvimento para Android usando a linguagem de programação Java. O método de obtenção de conhecimento será caracterizado por ser autodidata e experimental.*

1. Introdução

O avanço contínuo da tecnologia produziu, na última década, a popularização de dispositivos eletrônicos e, entre eles, os de computação móvel. Mais conhecidos pelos *smartphones*, *PDAs*, *tablets* e *e-readers*, esses dispositivos levaram o desenvolvimento de software a uma nova classe de aplicações, então chamadas, aplicações móveis [Syer et al. 2015]. Essas aplicações que são executadas nos dispositivos móveis exigem que sistemas operacionais específicos sejam usados, tais como o Google Android [Android.com 2016], o Apple iOS [Apple.com 2016] e o Windows Phone [Windows.com 2016].

Inicialmente desenvolvido para atender a indústria de entretenimento, a computação móvel se tornou um modelo de negócio que atualmente é utilizado nos mais diversos segmentos, inclusive em domínios mais críticos como o da saúde, das finanças e o industrial [Muccini et al. 2012]. O amplo mercado proporcionado pela computação móvel atraiu muitos profissionais de desenvolvimento para a criação de produtos de software móvel [Bhattacharya et al. 2013, Joorabchi et al. 2013]. Nesse contexto, o Sistema Operacional (SO) Android desenvolvido pela Google foi o primeiro a fornecer código-fonte de forma aberta, e torna o mercado Android um importante centro de distribuição em que os usuários têm autonomia para desenvolver e baixar as suas próprias aplicações [Yang and Yang 2012]. O fato de a plataforma Android possuir seu código fonte aberto e não ser mantido por apenas uma empresa contribui para seu aprimoramento, uma vez que desenvolvedores podem colaborar com melhorias no projeto como um todo, acrescentando novas funcionalidades ou simplesmente corrigindo defeitos [Mednieks et al. 2012].

2. Sistema Android

O Google Android é uma plataforma de desenvolvimento para aplicações móveis, composta por um sistema em camadas baseado no *kernel* do Linux, com diversas aplicações já instaladas e um ambiente de desenvolvimento integrado [Ableson et al. 2011]. O *kernel* do Linux também é responsável por gerenciar a memória, os processos, *threads* e a segurança dos arquivos [Lecheta 2013]. As aplicações podem ser desenvolvidas na linguagem Java, interpretadas por uma máquina virtual própria, compiladas, empacotadas, distribuídas e instaladas nos dispositivos. Além de não precisar pagar para utilizar o sistema, os fabricantes de dispositivos móveis podem customizar o SO sem precisar compartilhar o código-fonte, uma vez que o Android tem sua licença baseada na *Apache Software Foundation* (ASF) [Lecheta 2013]. Por disponibilizar o código-fonte, permite que novas funcionalidades sejam desenvolvidas sem que seja necessário uma intervenção da Google. Não existe diferença entre as aplicações que já vem instaladas no SO e as aplicações criadas utilizando o SDK, possibilitando o desenvolvimento de aplicações que acessam os recursos disponíveis no dispositivo. Empresas têm customizado os seus produtos e desenvolvedores têm aproveitado aplicações já desenvolvidas para vários modelos de hardware; essas características têm atraído muitos profissionais e contribuído para melhoria do sistema [Mednieks et al. 2012, Lecheta 2013]. A arquitetura do sistema Android é subdividida em cinco camadas de fundamental importância para facilitar o desenvolvimento de aplicações móveis e contribuir com a construção de soluções mais eficientes [Ableson et al. 2011, Lecheta 2013]. A plataforma Android é composta pelas seguintes camadas (ilustrada na Figura 1):

1. **Estrutura da aplicação:** é a camada utilizada por desenvolvedores de aplicações, responsável por mapear de forma direta as interfaces da camada de abstração de hardware e fornecer informações úteis sobre a implementação de *drivers*.
2. **Comunicação entre processos:** é a camada utilizada para efetivar a comunicação entre as camadas de estrutura da aplicação e o sistema de serviços Android, permitindo que a estrutura da aplicação invoque o código do sistema de serviços Android. A ideia central é ter um controlador que transfere mensagens entre o espaço de memória de diferentes aplicações.
3. **Sistema de serviços Android:** é a camada utilizada para estabelecer comunicação com os serviços do sistema para acessar o hardware. O Android inclui dois grupos de serviços que possuem componentes modulares: *(i)* o servidor de mídia, que compreende os serviços de áudio, câmera, reprodutor de vídeos, entre outros e *(ii)* o servidor de sistemas, que compreende os serviços de busca, gerenciador de atividades, gerenciador de janelas, etc.
4. **Camada de abstração de hardware:** é a camada utilizada para definir uma interface para que os fornecedores de hardware realizem suas implementações. Ela possibilita implementar funcionalidades sem afetar ou modificar a camada superior do sistema. Para permitir que o sistema Android interaja corretamente com o hardware, deve-se respeitar o contrato definido em cada interface específica de hardware. Exemplos de componentes de hardware são: áudio, câmera, *bluetooth*, imagens, entrada de dados, mídia, sensores e TV.
5. **Kernel do Linux:** é a camada utilizada para o desenvolvimento de *drivers* para o dispositivo. O Android utiliza uma versão do *kernel* do Linux com alguns recursos adicionais que são específicos da plataforma móvel.

Figura 1. Arquitetura do sistema Android

Cada versão do SO Android é vinculada a uma API que inclui recursos de usuário, recursos de desenvolvedor, alterações relacionadas a versão anterior e correções de *bugs*. No momento da escrita deste texto, o SO Android encontra-se na versão 6.0 Marshmallow, referente a API Level 23 [Developers 2016].

2.1. Ambiente de Desenvolvimento

A *Integrated Development Environment* (IDE) desenvolvida e apoiada pela Google para o desenvolvimento de aplicações móveis é chamada Android Studio IntelliJ[1] [Developers 2016]. Para se apoiar o desenvolvimento, existe um conjunto de ferramentas disponíveis no Android SDK. Entre os principais recursos oferecidos pelo Android SDK, estão: ambiente de desenvolvimento Java, emulador, bibliotecas e recursos necessários para construir, testar e depurar aplicações Android [Ableson et al. 2011, Mednieks et al. 2012, Lecheta 2013]. As ferramentas SDK exigem que o projeto tenha uma estrutura característica para compilar e empacotar uma aplicação corretamente [Developers 2016].

Como pré-requisito é necessário possuir o Java instalado na máquina. Além disso, dependendo do SO, é necessário criar variáveis de ambiente. Na Figura 2 é apresentada

[1]O Android Studio está disponível para *download* em: `http://developer.android.com/intl/pt-br/index.html`

a tela de configuração do **SDK Manager**, que consiste em ser um kit de desenvolvimento contendo exemplos de código-fonte, ferramentas de desenvolvimento e um emulador. Para obter tal acesso basta clicar em *Configure* na tela de boas vindas do Android Studio, em seguida selecionar *SDK Manager*. Também, com a IDE já aberta, é possível acessar *Tools*, *Android* e por fim *SDK Manager*. A principal importância da janela *SDK Manager* é realizar o *download* de novos recursos para o desenvolvimento. Por padrão, a IDE traz alguns desses recursos, então por hora, será utilizado apenas o que já está instalado.

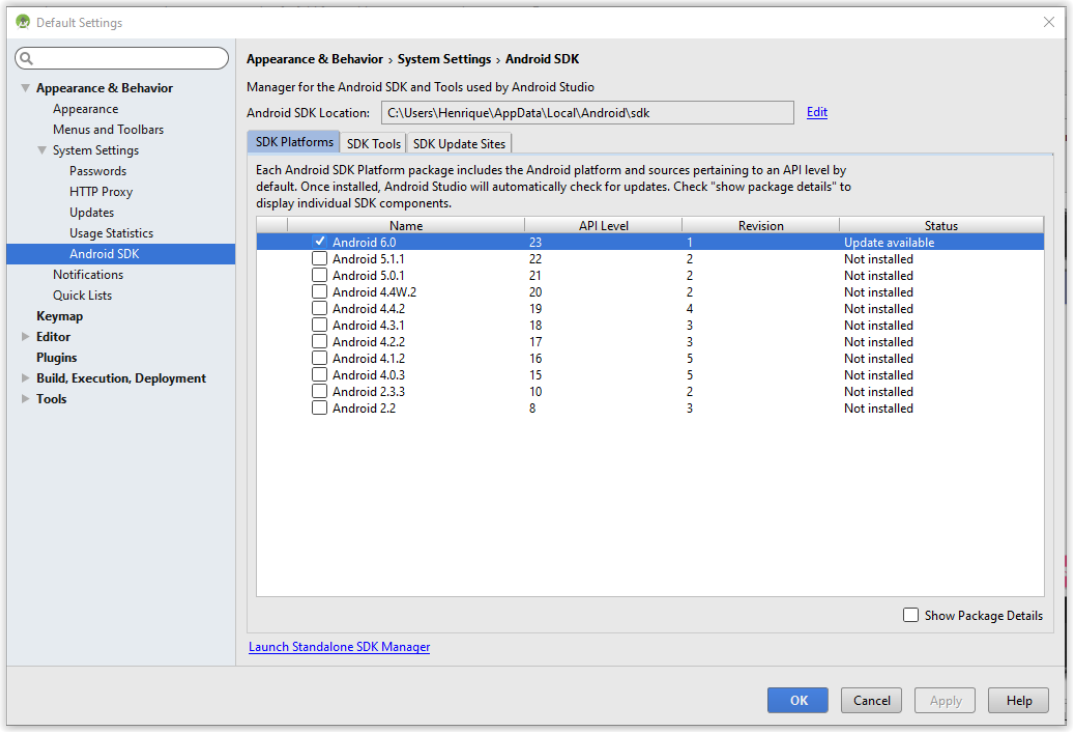

Figura 2. Tela *SDK Manager*

2.2. Estrutura de uma Aplicação Móvel em Android

A raiz da estrutura de uma aplicação móvel em Android é composta por módulos, que são divididos em quatro níveis: *(i)* nível de aplicação Android, que é responsável por apoiar as configurações da aplicação, o código-fonte da aplicação e os arquivos a nível de recursos. Esses arquivos são incorporados a aplicação que é instalada no dispositivo; *(ii)* nível de teste, que é utilizado para testar a aplicação, sendo que no Android Studio é criado um diretório *AndroidTest* para armazenar os testes JUnit; *(iii)* nível de biblioteca, que compartilha código-fonte que pode ser utilizado. Esses arquivos não são instalados no dispositivo, mas transferidos para a aplicação em tempo de compilação; e *(iv)* nível de motores de aplicações, que atribuem a funcionalidade de nuvem, como exemplo o *backup* de dados do usuário.

Ao longo do desenvolvimento deste texto, será necessário manipular os arquivos da aplicação. Por esse motivo será enfatizado o nível de aplicação Android, o qual é composto, principalmente por: *(i)* configuração ou *AndroidManifest*, responsável por definir as configurações da aplicação; *(ii)* recursos ou *res*, responsável por armazenar os recursos

da aplicação, tais como as imagens, as telas e os arquivos de mensagens; *(iii)* arquivos gerados automaticamente ou *gen*, responsável por armazenar a classe de constantes da aplicação que estabelece a ligação entre código-fonte e interface; e *(iv)* código-fonte ou *src*, responsável por armazenar o código-fonte do projeto, representado por classes Java.

2.2.1. Configuração da Aplicação

As configurações de uma aplicação Android são centralizadas no arquivo *AndroidManifest*, localizado no diretório raiz do projeto e codificado no padrão eXtensible Markup Language (XML). Ele é o arquivo principal do projeto e apresenta, para o SO, informações essenciais sobre a aplicação e sobre o que é necessário para executá-la, por isso deve ser lido antes que qualquer outro código [Monteiro 2012]. Entre as suas utilidades, esse arquivo é responsável por [Developers 2016]: *(i)* definir o nome do pacote Java para a aplicação; *(ii)* descrever os componentes que integram a aplicação e em que condições podem ser utilizados; *(iii)* determinar que processo vai suportar a aplicação; *(iv)* declarar que permissões a aplicação deve ter para acessar partes protegidas da API e interagir com outras aplicações; *(v)* declarar a permissão que outras aplicações são obrigadas a ter, a fim de interagir com os componentes da aplicação que está sendo executada; *(vi)* disponibilizar classes de instrumentação, responsáveis por fornecer informações da aplicação que está sendo desenvolvida e testada; *(vii)* declarar o nível mínimo da API Android que a aplicação requer; e *(viii)* declarar as bibliotecas que a aplicação deve estar vinculada.

O arquivo de configuração é composto por elementos e atributos, sendo que um elemento pode conter outros elementos e todos os seus valores são definidos por meio de atributos. Os atributos são opcionais, no entanto, alguns deles são necessários para que um elemento tenha função. Os principais elementos são descritos a seguir:

- **Manifesto (*manifest*):** é um elemento obrigatório para qualquer aplicação e só pode aparecer uma vez.
- **Aplicação (*application*):** é um elemento obrigatório para qualquer aplicação e só pode aparecer uma vez.
- **Atividade (*activity*):** cada atividade é responsável por implementar parte da interface do usuário. Uma atividade precisa ser configurada como sendo o ponto de partida da aplicação e então poderá ser executada;
- **Instrumentação (*instrumentation*):** declara uma classe *instrumentation* que permite monitorar a interação de uma aplicação com o sistema. O objeto dessa classe deve ser instanciado antes de qualquer um dos componentes da aplicação. Pode ser utilizada, por exemplo, para realizar teste funcional ou executar *profiling*.
- **Filtros de intenção (*intent-filter* e *action*):** são pacotes de informação (objetos *intent*) que ativam os componentes fundamentais de uma aplicação. Eles descrevem uma ação desejada, dados usados em ações, a categoria do componente que deve executar a ação, etc. Antes de iniciar um componente, o sistema Android precisa saber quais intenções ele permite. Por isso, os componentes anunciam seus recursos por meio de filtros de intenção no arquivo de configuração. Os filtros descrevem diferentes recursos e podem aparecer em qualquer quantidade. O Android executa três passos para compartilhar um componente: *(i)* localiza o componente

adequado para atender uma intenção; *(ii)* se necessário, inicia uma nova instância do componente; e *(iii)* passa o componente para o objeto da intenção.

- **Permissões (*permission* e *uses-permission*):** aplicações compartilham dados e recursos do dispositivo (por exemplo, o acesso a câmera), e as permissões de segurança são responsáveis por realizar declarações que podem ser usadas para limitar o acesso aos componentes ou características da aplicação. O excesso de permissões atribui vulnerabilidade ao sistema, portanto recomenda-se projetar aplicações para não requerer permissões (por exemplo, é preferível realizar armazenamento interno que externo e depender de permissões específicas). Um recurso pode ser protegido por, no máximo, uma permissão. Cada permissão é identificada por uma etiqueta exclusiva que geralmente indica a ação que foi restringida (por exemplo: `android.permission.CALL_EMERGENCY_NUMBERS`, que restringe as chamadas de emergência).
- **Suporte a telas (*supports-screens*):** permite especificar o tamanho de tela permitido pela aplicação, ou então habilitar o modo de compatibilidade de tela para telas maiores do que a suportada pela aplicação.
- **Versão da API (*uses-sdk*):** dentro do arquivo de configuração de cada aplicação pode ser especificado, por meio do atributo `minSDKVersion`, a versão miníma da plataforma Android em que a aplicação será executada. Assim como também é possível utilizar o atributo `targetSDKVersion` para especificar a versão para a qual a aplicação foi projetada.

2.2.2. Recursos da Aplicação

Na pasta *res*, é onde adicionamos recursos que irão compor nosso projeto, como arquivos de som, imagens, ícones e o principal, o arquivo que define o layout da *Activity*. Nesta mesma pasta também é possível encontrar o arquivo menu main, que gera um menu para a aplicação.

Todos os recursos utilizados pela aplicação são armazenados no diretório */res*, sendo que três principais categorias podem ser encontradas nesse diretório: *(i) layout*, contém os arquivos XML para construir as telas da aplicação; *(ii) drawable*, contém as imagens utilizadas na aplicação; e *(iii) values*, contém valores estáticos, no formato chave-valor, que podem ser utilizados por outros arquivos da aplicação.

Um *layout* é um *ViewGroup* responsável por definir a estrutura visual para uma interface de usuário. A classe *ViewGroup* serve como base para as subclasses chamadas *layouts*, que oferecem diferentes tipos de arquitetura de *layout*, tais como *LinearLayout*, *TableLayout* e *RelativeLayout*. Um objeto *ViewGroup* é uma estrutura de dados cujas propriedades armazenam os parâmetros de *layout* e conteúdo para uma área retangular específica de a tela. O *LinearLayout*, por exemplo, é um tipo especial de componente que pode organizar outros componentes exibidos na tela e funciona como um gerenciador de *layout*. Por padrão os componentes são distribuídos na vertical por meio do atributo *android:orientation="vertical"*. Na plataforma Android, a *activity* de uma tela é definida utilizando uma hierarquia de nós *View* e *ViewGroup*, como apresentado no diagrama da Figura 3. Essa árvore de hierarquia pode ser simples ou complexa dependendo do conjunto de *widgets* e *layouts* utilizados.

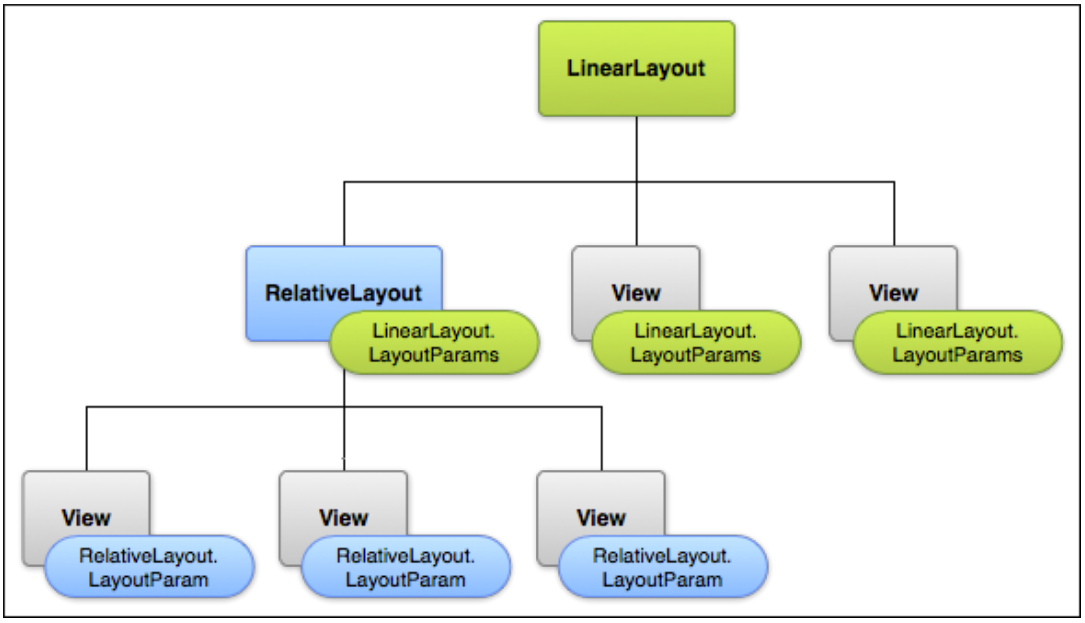

Figura 3. Visualização de uma hierarquia de exibição com parâmetros de *layout*

Existe uma grande diversidade na resolução de telas nos dispositivos móveis e, por meio do diretório *drawable* é possível customizar as imagens para ficarem do tamanho exato em cada resolução de forma automática. Nesse diretório existem seis outros diretórios para armazenamento das imagens: *(i) drawable_ldpi*, contém imagens baixa qualidade; *(ii) drawable_mdpi*, contém imagens média qualidade; *(iii) drawable_hdpi*, contém imagens alta qualidade; *(iv) drawable_xhdpi*, contém imagens extra-alta qualidade; *(v) drawable_xxhdpi*, contém imagens extra-extra-alta qualidade; e *(vi) drawable_xxxhdpi*, contém imagens extra-extra-extra-alta qualidade.

Ainda, existe um terceiro diretório importante dentro dos recursos, o *values*. Esse diretório contém os arquivos de mensagens ou de internacionalização, em especial o arquivo *strings.xml*. Esse arquivo centraliza os valores estáticos que são úteis quando necessário portar a aplicação para outro idioma, pois apenas um arquivo precisa ser traduzido e então todas as referências são atualizadas [Ableson et al. 2011]. Para que a aplicação tenha as suas mensagens suportadas em Inglês, é necessário criar um novo diretório (*/res/values-en*) com o arquivo de *strings* traduzido, essa configuração pode ser repetida para outras línguas. A partir do código de uma *activity* o padrão para acessar essas mensagens é "*@string/nome_da_chave*".

2.2.3. Primeiro Projeto

Mantendo a tradição em que o primeiro projeto deve ser um "Olá Mundo", a IDE traz consigo o código fonte deste famoso exemplo, no atual contexto *Hello World*.

Na tela de boas vindas ou na IDE já aberta clique na opção *Create New Project*, a janela (Figura 4) deve aparecer em sua tela.

Na janela de criação de projeto é solicitado o nome do projeto (*Application name*): que será o nome da aplicação; o campo (*Company name*): onde estarão armazenado as

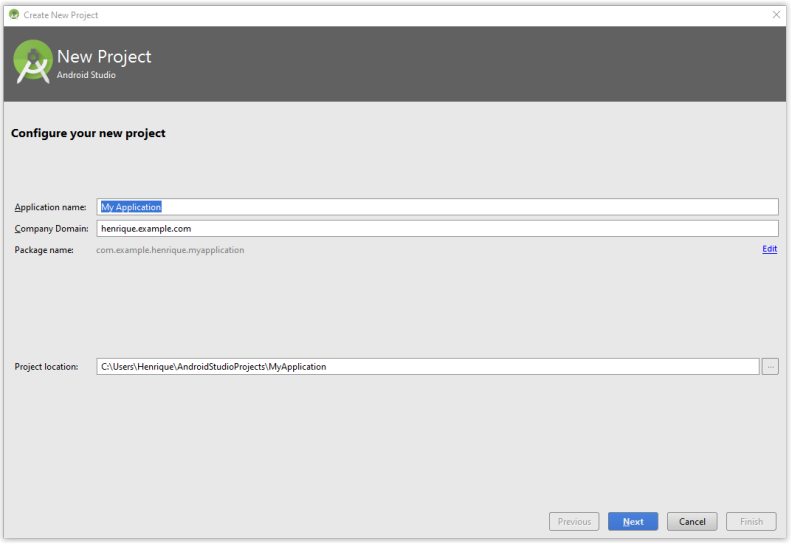

Figura 4. Tela de novo projeto

classes Java, e também indicará o domínio para a utilização do aplicativo na *Google Play* e seu nome para instalação em um smartphone; Por fim, o campo *Project location*, o qual indica o local de armezanagem dos arquivos de código fonte do projeto.

Clicando em *Next* a janela (Figura 5) mostrará as versões disponíveis de API de *Android*. Deve-se escolher uma das versões, tendo em mente que a versão escolhida será definida como configuração mínima (pré requisito) de um sistema operacional *Android* para executar a aplicação que está sendo criada.

Observação: toda vez que uma versão de API diferente é selecionada, ele traz como informação, a percentagem de aparelhos *Android* que estão em uso e são compatíveis, nos induzindo à criação de um aplicativo com maior portabilidade.

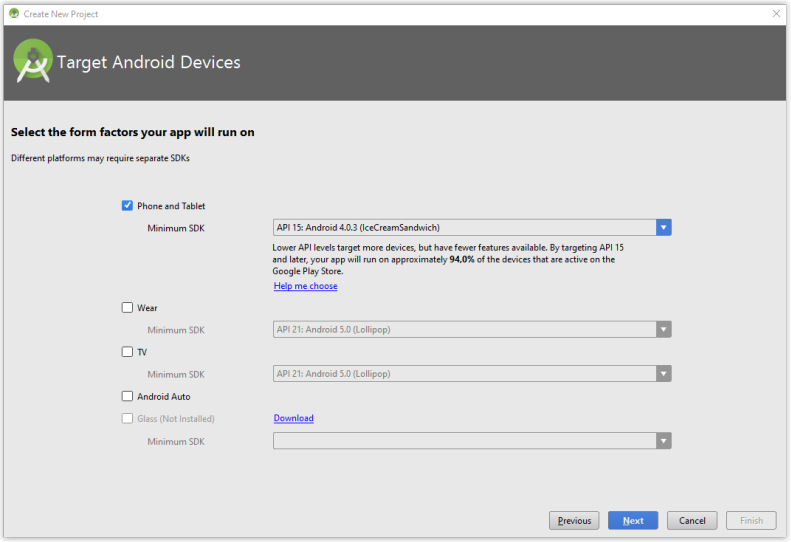

Figura 5. Tela *Target Android Devices*

Na próxima tela (Figura 6) é exibido diversas opções de template que se pode

selecionar para o desenvolvimento do aplicativo. Neste exemplo, selecione *Blank Activity* e prosseguir.

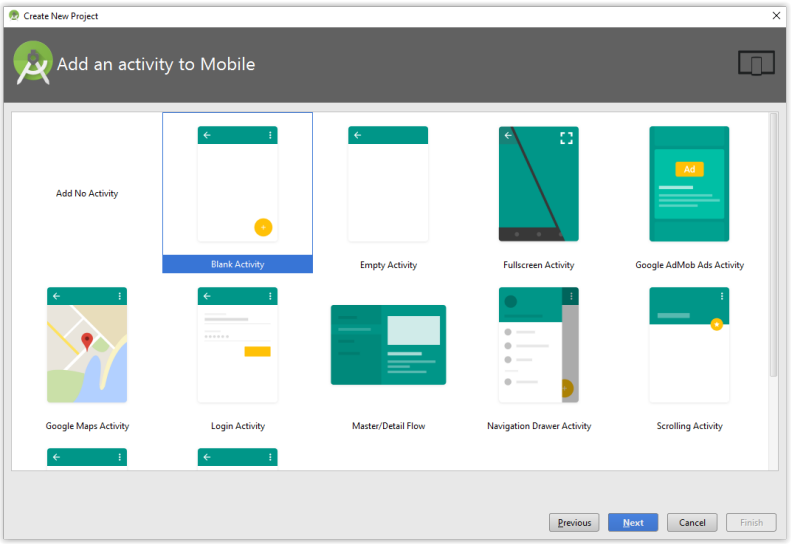

Figura 6. Tela de seleção de *Template*

É necessário preencher algumas informações para a *Activity* que está sendo criada (Figura 7), neste caso mantenha os nomes como ele sugeriu. Caso seja necessário, é possível criar outras *Activity*, mas no momento o objetivo é apenas criar uma principal. Clique em *Finish*.

Observação: Para quem está acostumado com a linguagem Java, a *Activity* se assemelha com a ideia do *JFrame*, tendo como objetivo, ser a *interface* de uma tela.

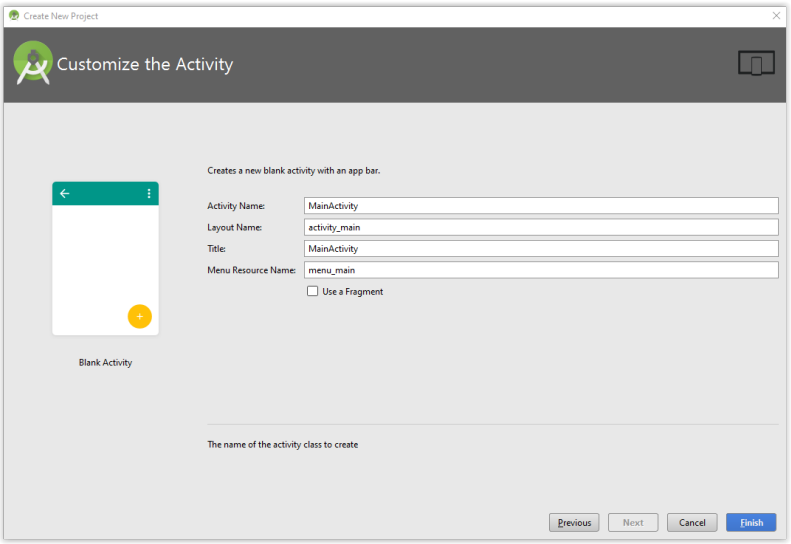

Figura 7. Informações da *Activity*

Após alguns instantes, o projeto é criado e a tela de desenvolvimento é aberta (Figura 8). No arquivo *activity main* defina-se o layout do aplicativo, em suma, os componentes gráficos que farão parte da ”tela”. Pode-se construir a tela utilizando a paleta de

componentes como auxiliar (clicando e arrastando o componente desejado) ou inserir os componentes diretamente via código, permitindo uma melhor personalização, a Figura 9 mostra com mais detalhes.

Observação: na parte direita, encontra-se a guia *Component Tree*, onde se localizam todos os componentes que já foram adicionados na *Activity*. Toda vez que um componente é selecionado nesta área, abre-se uma nova guia *Properties*, que nos dá mais opções de customização.

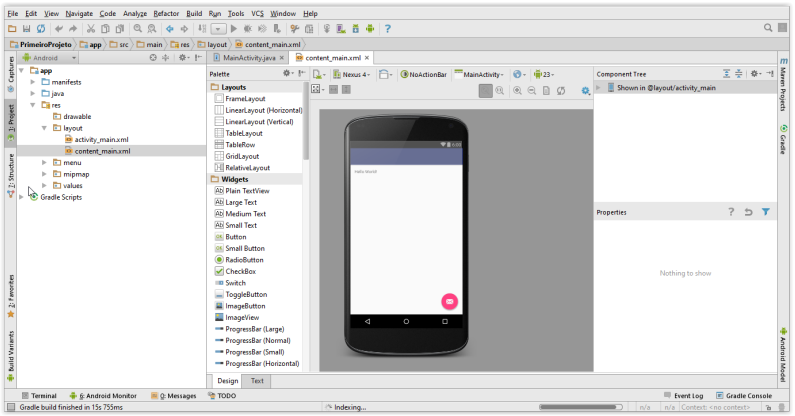

Figura 8. Tela principal do projeto

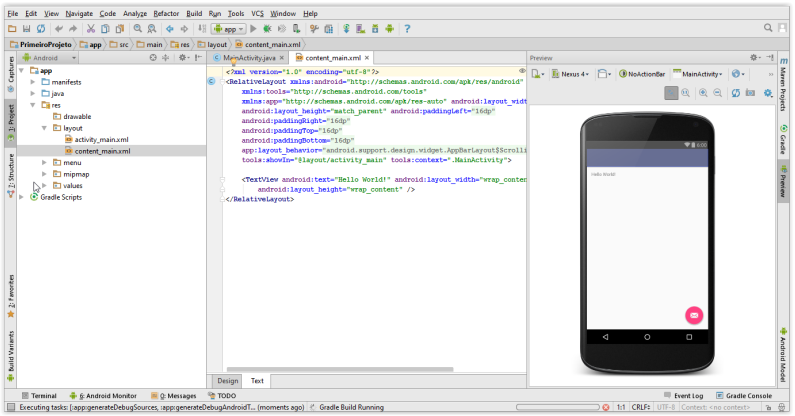

Figura 9. Visão do código XML da *Activity*

2.3. Executando o projeto

Para que seja possível testar a aplicação criada, é necessário possuir um dispositivo Android conectado ao computador (via USB) ou um emulador configurado. A IDE Android Studio fornece a opção de criar uma máquina virtual contendo o sistema operacional Android. É possível executar a aplicação pressionando *Shift + F10*, e uma pequena janela irá aparecer contendo o cenário da Figura 10.

Para a configuração de um emulador basta selecionar os '...' da parte inferior da tela. Esta opção nos levará à tela de gerenciamento de emulador (Figura 11). Escolha a opção *Create Virtual Device*.

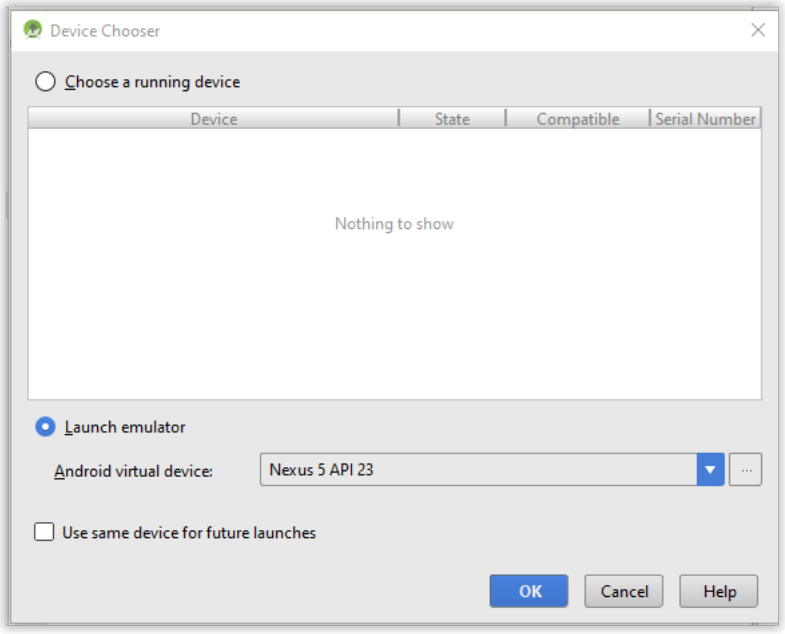

Figura 10. Tela *Device Chooser*

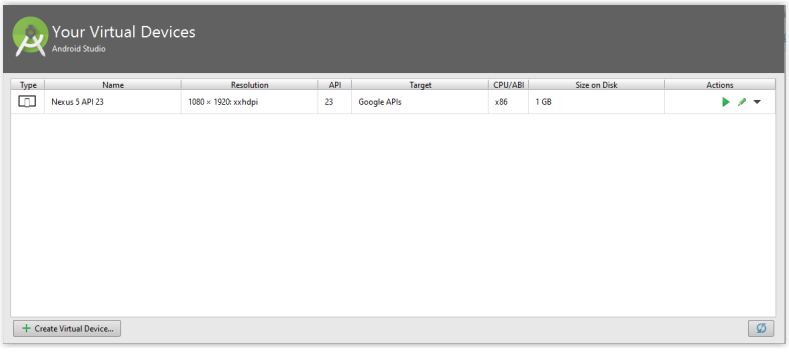

Figura 11. Listagem de emuladores

Agora é possível configurar o emulador com as características que tem em mente para a utilização do aplicativo. A Figura 12, mostra a seleção do *hardware* do nosso emulador. Clique em *Next*. Depois do *Hardware* selecionado, deve-se escolher a imagem do sistema que o emulador da IDE irá utilizar. As Figuras 13 e 14 mostram os últimos detalhes da configuração do emulador.

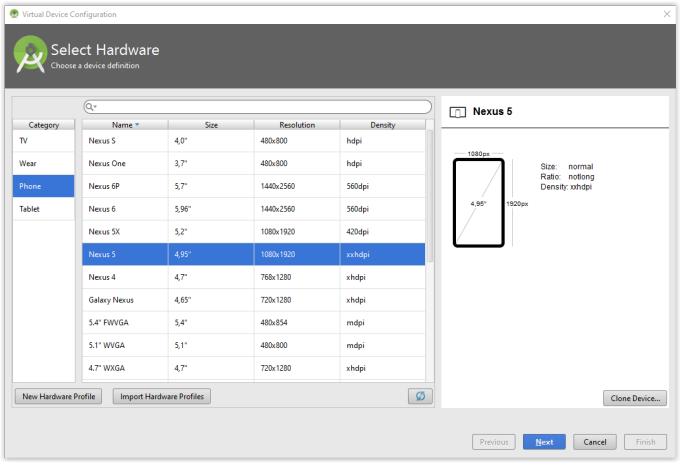

Figura 12. Tela *Select Hardware*

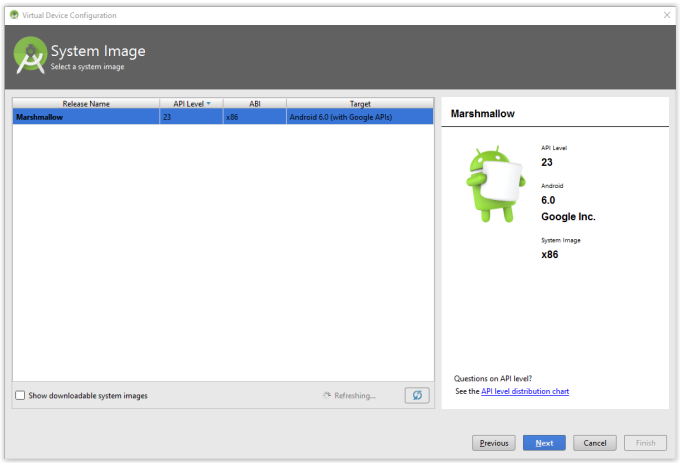

Figura 13. Tela *Select System Imagem*

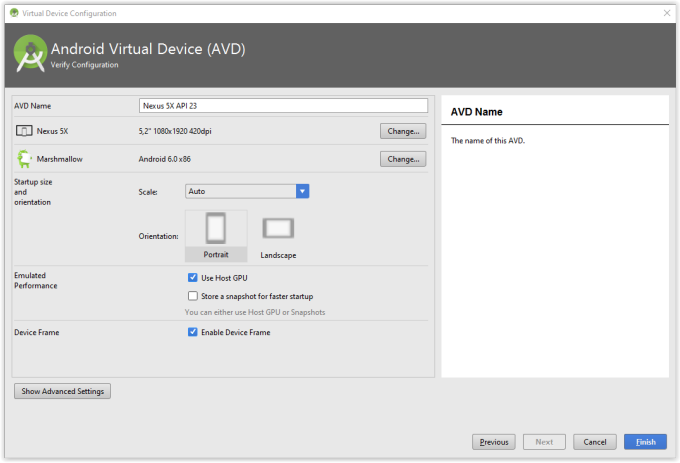

Figura 14. Informações gerais do emulador

Com o emulador criado, é possível selecioná-lo na janela que aparece após a execução da aplicação (Figura 10). O emulador rodando a aplicação criada terá o resultado representado na Figura 15.

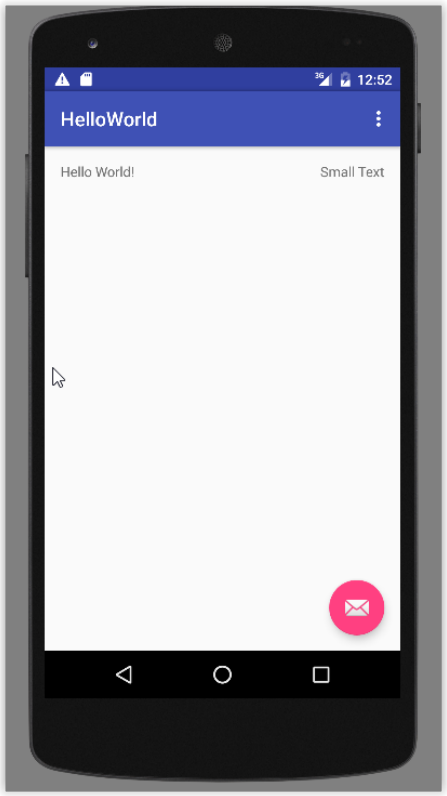

Figura 15. Execução do app no emulador

3. Sensores

Grande parte dos dispositivos que utilizam Android, trabalham com sensores que medem a movimentação, a orientação e diversas condições ambientais. Esses sensores podem

fornecer dados brutos com alta precisão e exatidão. Um sensor de um aparelho eletrônico representa, de maneira geral, uma resposta ao estímulo da natureza física, por exemplo, temperatura, luminosidade, velocidade, aceleração, entre outros. Como exemplo de um jogo que pode interpretar os gestos e movimentos do usuário (inclinação, vibração, rotação ou balanço). Da mesma forma, uma aplicação de clima pode usar sensor de umidade e temperatura de um dispositivo para calcular e comunicar o ponto de orvalho, ou uma aplicação de viagens pode utilizar o sensor de campo geomagnético e acelerômetro para relatar uma bússola. A plataforma Android suporta três grandes categorias de sensores [Developers 2016]:

- **Sensores de movimento:** medem forças de aceleração e rotação ao longo de três eixos. Essa categoria inclui acelerômetros, sensores de gravidade, giroscópios e sensores de rotação de vetor.
- **Sensores ambientais:** medem vários parâmetros ambientais, tais como temperatura ambiente e a pressão do ar, iluminação e umidade. Essa categoria inclui barômetros, fotômetros e termômetros.
- **Sensores de posição:** medem a posição física de um dispositivo. Essa categoria inclui sensores de orientação e magnetômetros.

A plataforma Android fornece dispositivos virtuais que fornecem dados de forma que a aplicação obtenha acesso ao conjunto de sensores físicos de um determinado dispositivo móvel. Não inclusos na lista dos dispositivos físicos, estão as câmeras, impressão digital, microfone e telas sensíveis ao toque.

Existem os sensores baseados em hardware (sensores base) e os sensores baseados em software (sensores compostos) que também são conhecidos por sensores virtuais ou sintéticos [Developers 2016]. Os sensores base utilizam componentes físicos e coletam dados por medição, tais como:

- Acelerômetro: é aplicado a um dispositivo nos três eixos e mede a força da aceleração em m/s^2, incluindo a força da gravidade (agitação e inclinação).
- Giroscópio: mede a taxa de rotação em rad/s em torno de cada um dos três eixos de um dispositivo (detecção de rotação).
- Luz: mede o nível de iluminação de um ambiente em lx (brilho da tela).
- Magnetômetros: mede o campo geomagnético do ambiente para todos os três eixos físico em microTesla (bússola).
- Pressão: mede a pressão do ar em hPa ou mbar (alterações na pressão do ar).
- Proximidade: mede a proximidade de um objeto em relação à tela de um dispositivo em cm (posição do telefone durante uma chamada).
- Umidade: mede a umidade relativa do ambiente em percentual (ponto de orvalho e umidade relativa).
- Temperatura: mede a temperatura do ambiente em graus Celsius (temperatura do ar).

Os sensores compostos imitam sensores de hardware e derivam de um ou mais sensores base. Alguns exemplos são: vetor de rotação que é composto por acelerômetro, magnetômetro e giroscópio; gravidade é composto por acelerômetro e giroscópio; detector de passos é composto por acelerômetro. Acelerômetros e magnetômetros são mais

comuns em *smartphones*, e barômetros ou termômetros são menos comuns. Um dispositivo pode ter mais de um sensor do mesmo tipo, como exemplo, dois sensores de gravidade, cada um tendo uma gama diferente.

Cada sensor Android tem uma forma de funcionar e gera diferentes tipos de dados. No caso dos sensores oficiais, existe uma biblioteca (*sensors.h*) que os define sob o prefixo *SENSOR_TYPE_*. Os sensores não oficiais, tem um tipo temporário definidos pelos fabricantes que pode ser promovido para oficial e compartilhado com outros desenvolvedores. Não pode existir mais de um sensor com o mesmo tipo, sendo que uma lista com todos os sensores disponíveis no dispositivo é apresentada pela *Hardware Abstraction Layer* (HAL).

Por meio do *framework* Sensor é possível realizar a identificação de sensores e a capacidade dos sensores. Identificar sensores e capacidades de sensores em tempo de execução é útil se a aplicação possui características que dependem de tipos ou capacidades de sensores específicos. Por exemplo, quando se deseja identificar todos os sensores que estão presentes em um dispositivo e desativar os recursos da aplicação que dependem de sensores que não estão presentes. Da mesma forma, pode-se desejar identificar todos os sensores de um determinado tipo para escolher a implementação do sensor que tem o melhor desempenho para a aplicação.

O sensor de orientação é um exemplo utilizado a todo momento nos dispositivos móveis, quando, por exemplo, altera-se o modo de visualização da tela para o formato retrato (*portrait*) ou paisagem (*landscape*). A coleção de sensores disponíveis para a utilização varia de acordo com o aparelho e também com a versão do Android. Quanto mais novo o aparelho, maior será a gama de sensores disponíveis. É possível acompanhar quais são os sensores disponíveis através do site oficial[2] de desenvolvimento Android. A verificação de sensores no dispositivo pode ser facilitada com o aplicativo *Sensor Box*[3].

Para acessar um sensor basta chamá-lo utilizando o método `getDefaultSensor(type)` da classe *SensorManager*. É necessário passar como parâmetro uma das constantes referentes ao sensor que será utilizado. Para obter uma instância da classe *SensorManager* é feita uma chamada ao método `getSystemService(service_name)` da classe *Context*. Nesse caso deve-se passar como parâmetro `Context.SENSOR_SERVICE`. Após declarar essa instância é necessário adicionar um *listener* para a notificação sobre alterações nos valores do sensor em questão.

O controle e a manipulação dos valores que o sensor fornece é necessário que a classe que utilizará o sensor implemente dois métodos: *(i)* `onAccuracyChanged()` e *(ii)* `onSensorChanged()`. O primeiro método está relacionado às mudanças de precisão do sensor, Esse método recebe como parâmetro um objeto do tipo Sensor e outro do tipo de calibragem (definido na Tabela 1 de constantes inteiras).

Na próxima seção um dos sensores citados anteriormente, o sensor de localização.

---

[2]Site oficial disponível em: `http://developer.android.com/intl/pt-br/guide/topics/sensors/sensors_overview.html`

[3]Disponível em: `https://play.google.com/store/apps/details?id=imoblife.androidsensorbox`

Tipo de calibragem	Informação do sensor
SENSOR_STATUS_ACCURACY_LOW	Baixa precisão
SENSOR_STATUS_ACCURACY_MEDIUM	Certo nível de determinação
SENSOR_STATUS_ACCURACY_HIGH	Máximo de precisão
SENSOR_STATUS_UNRELIABLE	Não é confiável

Tabela 1. Calibragem dos sensores

3.1. GPS

3.1.1. Chave de acesso

Para utilizar o sensor de localização, a primeira configuração a ser realizada é ativar o serviço de mapas. Esta operação será executada por meio da opção *APIs ativadas* e então habilitando o serviço *Google Maps Android API*. O próximo passo consiste em criar a chave de acesso para o serviço Google Maps. A criação de chaves é feita no *Google Developers Console*[4]. Deve-se então obter o *SHA-1 fingerprint* do certificado que foi utilizado para compilar o projeto. Por padrão um certificado de *debug* é criado atuomaticamente. Para localizá-lo basta acessar o arquivo em `C:/Users/usuario/.android/debug.keystore` caso o sistema operacional seja Windows e, caso o sistema for Linux o arquivo poderá ser encontrado em `/home/usuario/.android/debug.keystore`. Após gerar o caminho do certificado de *debug*, deve-se copiar e colar em um *prompt* de comando para extrair o *SHA-1 fingerprint*.

```
C:\keytool -list -v
    -keystore "C:\Users\usuario\.android\debug.keystore"
    -alias androiddebugkey -storepass android
    -keypass android
```

Apenas o valor da chave *SHA1* é importante neste momento. Deve-se copiar o valor, voltar para a página do *Google Developers* e acessar a página *APIs & auth>Credentials*, então pressionar o botão *Criar credenciais>Chave de API>Chave de Android*. Na janela que aparecer, deve-se preencher as três informações (nome para a chave, nome do pacote e o valor do *SHA-1*). Este procedimento irá resultar em uma chave de acesso ao *Google API Android.* Em seguida será exibida uma tela com a *Key* criada.

3.1.2. Configuração do Projeto

É também necessário configurar o projeto, pois a API de mapas depende do *Google Play Services*. Para tal deve-se acessar o arquivo `/build.gradle` e inserir as seguintes linhas:

```
1 ...
2 compile 'com.google.android.gms:play-services-maps:8.3.0'
3 compile 'com.google.android.gms:play-services-location:8.3.0'
```

[4]*Google Developers Console* disponível em: `https://console.developers.google.com`

Perceba que o `build.gradle` é responsável pela configuração de *build global*, nesse caso com o comando `compile` ele se encarrega de baixar as dependências do projeto, sem precisar importar bibliotecas para dentro de nosso *workspace*.

No arquivo de configuração do sistema `AndroidManifest.xml` deve-se estabelecer as permissões de acesso aos sensores:

```
...
<uses-permission
   android:name="android.permission.INTERNET" />
<uses-permission
   android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission
   android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission
   android:name="android.permission.READ_EXTERNAL_STORAGE" />

<!-- GPS por hardware -->
<uses-permission
   android:name="android.permission.ACCESS_FINE_LOCATION" />

<!-- Triangulacao de antenas -->
<uses-permission
   android:name="android.permission.ACCESS_COARSE_LOCATION" />

<!-- Dependencia do Mapas v2 -->
<uses-feature
   android:glEsVersion="0x00020000" android:required="true" />
...
```

Note que na última linha onde foram tratadas as dependências do *Mapas v2*, utilizou-se como parâmetro na *uses-feature* o valor `glEsVersion="0x00020000"`. O mesmo faz referência a versão do OpenGL ES que a aplicação utilizará, o valor presente no manifesto indica a versão 2.0 da plataforma.

```
...
<application
    <!-- Google Play Services -->
    <meta-data
        android:name="com.google.android.gms.version"
        android:value="@integer/google_play_services_version" />

    <!-- Chave de acesso criada anteriormente -->
    <meta-data android:name="com.google.android.maps.v2.API_KEY"
        android:value="@string/API_KEY" />
</application>
```

Pelo fato de se ter utilizado a chave `@string API_KEY` no arquivo de manifesto, deve-se então criar um arquivo de internacionalização `/res/values/strings_config.xml`:

```
<resources>
    <string name="app_name">GPS</string>
    <string name="action_settings">Settings</string>
    <string name="API_KEY">
```

```
5         Caracteres que definem a API Key gerada
6     </string>
7 </resources>
```

3.1.3. Obter localização atual

Para exibir o mapa deve-se inserir no layout da `Activity` o mapa com a classe `SupportMapFragment`. Adicione no arquivo `content_main.xml` o seguinte código:

```
1  ...
2      <LinearLayout
3          android:layout_width="match_parent"
4          android:layout_height="match_parent"
5          android:orientation="vertical">
6          <TextView
7              android:id="@+id/text"
8              android:layout_width="wrap_content"
9              android:layout_height="wrap_content"
10             android:text="Ultima localizaÃ§Ã£o pela API."
11             android:paddingBottom="50px"/>
12         <fragment
13             android:id="@+id/map"
14             android:layout_width="match_parent"
15             android:layout_height="match_parent"
16             class="com.google.android.gms.maps.SupportMapFragment" />
17     </LinearLayout>
18     ...
```

Deste modo, ao executar a aplicação, o mapa apenas mostrará a localização padrão. Para se obter a localização atual, se criará um botão para que o mesmo acione a ação de se obter o atual posicionamento. No arquivo `/res/menu/menu_main.xml`:

```
1 <menu xmlns:android="http://schemas.android.com/apk/res/android"
2     xmlns:app="http://schemas.android.com/apk/res-auto"
3     xmlns:tools="http://schemas.android.com/tools"
4     tools:context=".MainActivity">
5     <item android:id="@+id/action_my_location"
6         android:title="@string/action_my_location"
7         android:icon="@drawable/ic_media_play"
8         app:showAsAction="always" />
9 </menu>
```

Agora se dirigindo aos arquivos responsáveis pela lógica da nossa aplicação, são neles que devem ser inseridos as diretivas de conexão, ação dos itens do menu, checagem de permissões, entre outros detalhes. Vamos criar a classe `PermissionUtils.java`:

```
1 ...
2     public static boolean validate(Activity activity,
3                     int requestCode,
4                     String... permissions) {
5         List<String> list = new ArrayList<String>();
```

```
        for (String permission : permissions) {
            if (!checkPermission(activity, permission)) {
                list.add(permission);
            }
        }
        if (list.isEmpty()) {
            return true;
        }

        String[] newPermissions = new String[list.size()];
        list.toArray(newPermissions);

        ActivityCompat.requestPermissions(activity,
                        newPermissions,
                        1);

        return false;
    }
}
...
```

Será utilizado o método `validate()` que possui como objetivo validar as permissões da aplicação que são passadas por parâmetro pela variável `permission`. Através da mesma, é criado uma lista do tipo `String`, onde posteriormente ela será percorrida para se obter e checar todas as permissões.

Em toda `Activity` deve-se controlar o fluxo de conexões e de estados que a aplicação terá, neste caso serão seis os métodos que terão este controle: `onCreate()`, `onConnected()`, `onConnectionSuspended()`, `onConnectionFailed()`, `onStop()` e `onStart()`. Agora ao arquivo principal do exemplo. Arquivo `MainActivity.java`:

```
...
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);

        //Fragment do mapa
        mapFragment = (SupportMapFragment)
         getSupportFragmentManager().findFragmentById(R.id.map);
        mapFragment.getMapAsync(this);

        //Configura o objeto GoogleApiClient
        mGoogleApiClient = new GoogleApiClient.Builder(this)
                .addConnectionCallbacks(this)
                .addOnConnectionFailedListener(this)
                .addApi(LocationServices.API)
                .build();
    }

    @Override
    public void onMapReady(GoogleMap map) {
```

```
24          Log.d(TAG, "onMapReady: " + map);
25          this.map = map;
26          map.setMapType(GoogleMap.MAP_TYPE_NORMAL);
27
28      }
29
30      @Override
31      public void onConnected(Bundle bundle) {
32          toast("Conectado ao Google P. Services");
33      }
34
35      @Override
36      public void onConnectionSuspended(int i) {
37          toast("Conexao interrompida");
38
39      }
40
41      @Override
42      public void onConnectionFailed
43        (ConnectionResult connectionResult) {
44          toast("Erro ao conect : " + connectionResult);
45      }
46
47      // Metodo da Activity
48      // Paro conexao mGoogleApiClient
49      @Override
50      protected void onStop() {
51          mGoogleApiClient.disconnect();
52          super.onStop();
53      }
54
55      // Metodo da Activity
56      // Inicia conexao mGoogleApiClient
57      @Override
58      protected void onStart() {
59          super.onStart();
60          mGoogleApiClient.connect();
61      }
62  }
63  ...
```

Por fim, no mesmo arquivo, cria-se outro dois métodos: `onOptionsItemSelected()` e `setMapLocation`. O primeiro método é responsável pela `String` contendo as permissões necessárias para a aplicação, após a validação das permissões pelo método `PermissionsUtils.validade()` ele chama o método `setMapLocation`, o mesmo contém os comandos que vão controlar o zoom da câmera do mapa, qual posição ele irá marcar, até mesmo qual o desenho da marcação será utilizado.

```
1  ...
2    @Override
3      //Aciona a atualizacao do mapa
4      public boolean onOptionsItemSelected(MenuItem item) {
5          SplashActivity sa = new SplashActivity();
6          switch (item.getItemId()) {
```

```
            case R.id.action_my_location:
                String permissions[] = new String[]{
                        Manifest.permission.WRITE_EXTERNAL_STORAGE,
                        Manifest.permission.READ_EXTERNAL_STORAGE,
                        Manifest.permission.ACCESS_COARSE_LOCATION,
                        Manifest.permission.ACCESS_FINE_LOCATION,

                };

                boolean ok = PermissionsUtils.validate(this,
                        0,
                        permissions);
                if (ok){
                    Location l =
                    LocationServices.FusedLocationApi
                     .getLastLocation(mGoogleApiClient);
                    Log.d(TAG, "lastlocation: " + l);

                    setMapLocation(l);
                    return true;
                }
        }
        return super.onOptionsItemSelected(item);
    }
    // Atualiza local do mapa
    private void setMapLocation(Location l) {
        if (map != null && l != null) {
            LatLng latLng = new LatLng(l.getLatitude(),
                    .getLongitude());
            CameraUpdate update = CameraUpdateFactory
                    .newLatLngZoom(latLng, 15);
            map.animateCamera(update);
            Log.d(TAG, "setMapLocation: " + l);
            TextView text = (TextView) findViewById(R.id.text);
            text.setText(String.format("Lat/Lnt %f%f, provider: %s",
               l.getLatitude(),
               l.getLongitude(),
               l.getProvider()));
            CircleOptions circle =
               new CircleOptions().center(latLng);
            circle.fillColor(Color.RED);
            circle.radius(25); //metros
            map.clear();
            map.addCircle(circle);
        }
    }
```

Ao executar o aplicativo se terá algo parecido com a imagem 16, que também mostra o efeito após clicar no item do menu que aciona a ação que se obtém a localização atual.

3.2. Acelerômetro

No atual contexto, refere-se ao sensor de aceleração, como `TYPE_ACCELEROMETER`, que simplesmente informa a força da aceleração (metros por segundo ao quadrado) apli-

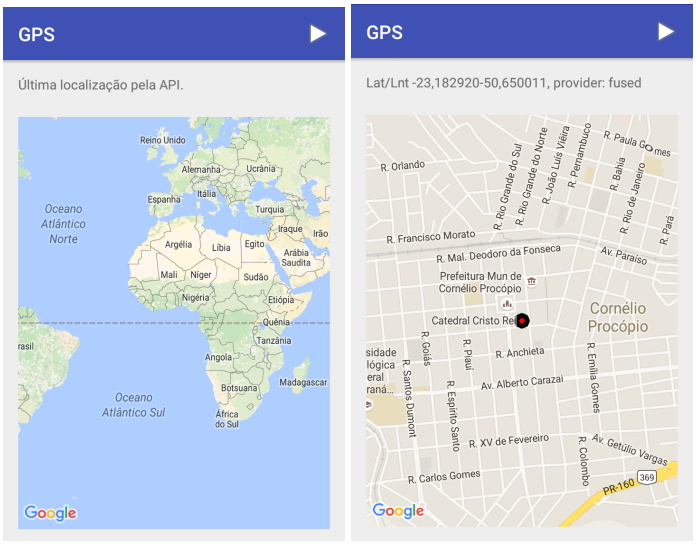

Figura 16. Execução do exemplo de GPS

cada no dispositivo nos eixos (x,y e z), incluindo a força da gravidade. O valor do sensor varia entre 0 e aproximadamente 9,81. A Figura 17 mostra como o sistema de coordenadas se aplica no sensor, ainda com base na mesma imagem, é possível assumir que o valor do eixo Y é aproximadamente 9,81. Deixando o celular de ponta cabeça teria-se -9,81. A mesma abordagem se aplica para os outros eixos, de maneira que no eixo X, ao girar o celular para a direita teríamos 9,81 e para esquerda -9,81. No caso do eixo Z, ele fica positivo quando o celular está sobre uma superfície, onde não há forças nos outros eixos.

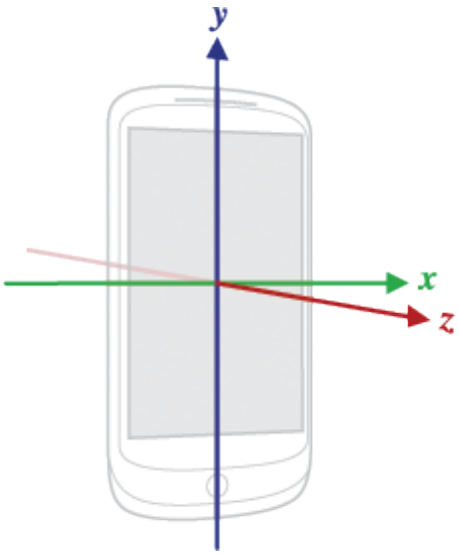

Figura 17. Sistema de coordenada utilizada no sensor

Com funcionamento do sistema de coordenadas no sensor em mente se criará um pequeno exemplo que testa os conceitos abordados. Crie uma `Activity` **sensor acelerometro** e adicione o seguinte layout que terá como função exibir os valores de todos os eixos.

```
1    <LinearLayout
```

```
        android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        android:orientation="vertical">
        <TextView
            android:id="@+id/tX"
            android:text="@string/x"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />
        <TextView
            android:id="@+id/tY"
            android:text="@string/y"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />
        <TextView
            android:id="@+id/tZ"
            android:text="@string/z"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />
    </LinearLayout>
```

Agora precisa-se criar o arquivo responsável pela lógica do exemplo o `AcelerometroActivity.java`, o qual defina-se da seguinte forma:

```
...
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setContentView(
                R.layout.activity_activity_sensor_acelerometro);
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);
        FloatingActionButton fab =
                (FloatingActionButton) findViewById(R.id.fab);
        fab.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                Snackbar.make(view,
                        "Replace with your own action",
                        Snackbar.LENGTH_LONG)
                        .setAction("Action", null).show();
            }
        });

        sensorManager = (SensorManager)
                getSystemService(SENSOR_SERVICE);
        sensor = sensorManager.getDefaultSensor(TIPO_SENSOR);

    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater()
                .inflate(
                        R.menu.menu_activity_sensor_acelerometro,
                        menu);
```

```
        return true;
    }
...
```

Ressalta-se o método `onCreate()`, pelo mesmo criar uma instância do sensor necessário para a aplicação.

```
...
// Retomo a activity e aloco novamente o listener
    @Override
    protected void onResume(){
        super.onResume();
        if (sensor != null){
            sensorManager.registerListener(this,
                    sensor,
                    SensorManager.SENSOR_DELAY_NORMAL);
        }
    }
// Deixa activity, desaloca-se memoria
    @Override
    protected void onPause(){
        super.onPause();
        sensorManager.unregisterListener(this);
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        int id = item.getItemId();
        if (id == R.id.action_settings) {
            return true;
        }
        return super.onOptionsItemSelected(item);
    }

    // Mudancas no sensor, refletem novos valores
    @Override
    public void onSensorChanged(SensorEvent event) {
        float sensorX = -event.values[0];
        float sensorY = event.values[1];
        float sensorZ = event.values[2];

        TextView tx = (TextView) findViewById(R.id.tX);
        TextView ty = (TextView) findViewById(R.id.tY);
        TextView tz = (TextView) findViewById(R.id.tZ);

        if (tx != null) {
            tx.setText("X: " + sensorX);
            ty.setText("Y: " + sensorY);
            tz.setText("Z: " + sensorZ);
        }
    }
```

Os métodos `onPause()`, `onResume()` tem como papel controlar os estados da aplicação, afim de poupar memória e consequentemente bateria do celular. A principal funcionalidade do exemplo, encontra-se no método `onSensorChanged()`, cuja

função é atualizar os valores que são obtidos constantemente pelo celular que muda de posição. Para se testar o acelerômetro criado, trave a orientação no arquivo de manifesto. `AndroidManifest.xml`:

```
<activity
    ...
    android:name=".activity_sensor_acelerometro"
    android:screenOrientation="portrait"
    ...
    />
```

A Figura 18 mostra o resultado do aplicativo após ser compilado e executado no *smartphone*.

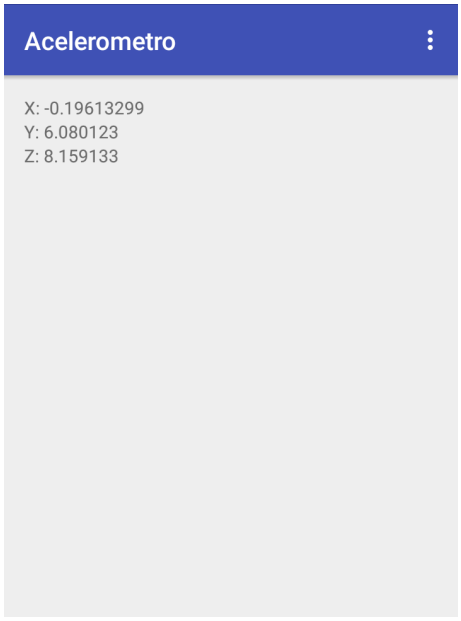

Figura 18. Tela do aplicativo Acelerometro

Vale ressaltar que para este exemplo travou-se a orientação da tela como retrato, caso a orientação da tela mude, os eixos x, y e z também irão e o controle do mesmo deve ser implementado.

4. Trabalhando com a Câmera

Para se tirar fotos com o Android, existem duas opções. A primeira consiste em trabalhar com a classe `android.hardware.Camera`, controlando a API de baixo nível (implementação de alguns métodos, rotação de tela, controle de estado , etc.). A opção mais interessante é utilizar uma `intent` para a aplicação nativa do Android.

De forma simples, uma `Intent` representa uma ”Mensagem”, um pedido que é encaminhado para o sistema operacional. O sistema irá responder de acordo com a mensagem recebida, neste caso, abrirá a Câmera.

Vamos ao código. Primeiro, altera-se o arquivo `activity_main.xml` de forma com que se possa clicar em um botão para acionar a câmera e em seguida, visualizar a prévia da foto registrada.

```
...
    <!-- Button para acionar a cam -->
    <ImageButton android:id="@+id/btAbrirCamera"
        android:layout_width="60dp"
        android:layout_height="60dp"
        android:src="@android:drawable/ic_menu_camera"
        android:text="Abrir a Camera" />
    <!-- ImageView que receberÃ¡ a foto -->
    <ImageView android:id="@+id/imagem"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:layout_gravity="center"/>
...
```

Com a interação pronta, cria-se agora o código `MainActivity.java`, responsável por disparar a intent para abrir a câmera.

```
public class MainActivity extends AppCompatActivity {
    private ImageView imgView;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        imgView = (ImageView) findViewById(R.id.imagem);
        ImageButton b =
                (ImageButton) findViewById(R.id.btAbrirCamera);
        b.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent i =
                    new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
                startActivityForResult(i, 0);
            }
        });
    }

    @Override
    protected void onActivityResult(int requestCode,
                                    int resultCode,
                                    Intent data){
        super.onActivityResult(requestCode,
                resultCode,
                data);
        if (data != null){
            Bundle bundle = data.getExtras();
            if (bundle != null){
                Bitmap bitmap = (Bitmap) bundle.get("data");
                imgView.setImageBitmap(bitmap);
            }
        }
    }
}
```

Agora basta declarar a permissão para utilizar a câmera no arquivo de manifesto `AndroidManifest.xml`.

```
<manifest ...>
    <uses-permission android:name="android.permission.CAMERA" />
    <application ... />
</manifest>
```

4.1. Obter arquivo de foto

Na situação atual, o exemplo apenas tira a foto e mostra uma prévia em baixa qualidade. Para que se possa obter a foto original com toda a resolução, é necessário passar um parâmetro para a intent da câmera, que deve ser o caminho para salvar o arquivo.

Será necessário utilizar duas classse[5]: `SDCardUtils` e `ImageResizeUtils`. A classe `SDCardUtils` traz consigo os métodos necessários para se manusear arquivos no geral e classe `ImageResizeUtils` trata de redimencionar imagens de uma forma eficiente. O arquivo `MainActivity.java` trabalhará com alguns métodos dessas classes.

```
public class MainActivity extends AppCompatActivity {
    private ImageView imgView;
    // Caminho para salvar o arquivo
    private File file;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        imgView = (ImageView) findViewById(R.id.imagem);
        ImageButton b =
                (ImageButton) findViewById(R.id.btAbrirCamera);
        b.setOnClickListener(new View.OnClickListener() {

            @Override
            public void onClick(View v) {
                // Cria o caminho do arquivo no SD card
                file = SDCardUtils.getPrivateFile(getBaseContext(),
                        "foto.jpg",
                        Environment.DIRECTORY_PICTURES);
                // Chama intent informando o arquivo para salvar foto
                Intent i =
                    new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
                i.putExtra(MediaStore.EXTRA_OUTPUT,
                        Uri.fromFile(file));
                startActivityForResult(i, 0);
            }
        });
        if (savedInstanceState != null){
            // Caso a tela tenha girado, recupera o estado
            file =
                (File) savedInstanceState.getSerializable("file");
```

[5]Classes disponíveis em: `https://github.com/livroandroid/AndroidUtils/tree/master/app/src/main/java/livroandroid/lib/utils`

```
33              showImage(file);
34          }
35      }
36
37      @Override
38      protected  void onSaveInstanceState(Bundle outState) {
39          super.onSaveInstanceState(outState);
40          // Salvar o estado caso gire a tela
41          outState.putSerializable("file", file);
42      }
43  ...
```

O método `SDCardUtils.getPrivateFile(...)` na linha 18, cria um arquivo *foto.jpg* no SD card, e esse arquivo foi passado por intent para a câmera. A linha 29 recupera o arquivo logo depois de recriar a `acitivity` e a linha 38 salva o arquivo no `Bundle` antes de destruir a `acitivity`. Ambas as linhas são referentes ao ciclo de vida da `acitivity`. O usuário pode girar o aplicativo móvel entre horizontal e vertical, levando isto em conta, é necessário salvar o estado da foto, pois a mudança de orientação da tela mata e recria a `activity`.

```
1   ...
2       @Override
3       protected void onActivityResult(int requestCode,
4                                       int resultCode,
5                                       Intent data){
6           super.onActivityResult(requestCode,
7                   resultCode,
8                   data);
9           if (data != null){
10              Bundle bundle = data.getExtras();
11              if (bundle != null){
12                  Bitmap bitmap = (Bitmap) bundle.get("data");
13                  imgView.setImageBitmap(bitmap);
14              }
15          }
16
17          if (resultCode == RESULT_OK && file != null){
18              // Recebe o resultado da intent da Cam
19              showImage(file);
20          }
21      }
22
23      // Atualiza a imagem na tela
24      private void showImage(File file){
25          if (file != null && file.exists()){
26              Log.d("foto", file.getAbsolutePath());
27              int w = imgView.getWidth();
28              int h = imgView.getHeight();
29              // Redimensionar foto antes de mostra-la no ImageView
30              Bitmap bitmap =
31                  ImageResizeUtils.getResizedImage(Uri.fromFile(file),
32                      w, h, false);
33              Toast.makeText(this, "w/h: " +
34                      bitmap.getWidth() +
35                      "/" +
```

```
36                          bitmap.getHeight(),
37                          Toast.LENGTH_SHORT).show();
38                  imgView.setImageBitmap(bitmap);
39              }
40          }
41  ...
```

A linha 17 é o trecho de código reponsável por receber o resultado da intent da câmera, e nesse caso, basta abrir a foto do arquivo. Dentro do método `showImage()` na linha 30, percebe-se o código que faz o redimensionamento da foto antes de mostrá-la no `ImageView`.

Para a utilização dos métodos criados é necessário adicionar uma nova permissão no arquivo de manifesto.

```
1  ...
2  <uses-permission
3      android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
4  ...
```

A Figura 19 mostra como ficou o aplicativo produzido neste exemplo. Do lado esquerdo, encontra-se o estado inicial da aplicação, onde a foto ainda não foi tirada com a câmera do dispositivo móvel. Do lado direito, se tem o resultado depois da foto ser tirada.

Caso o redimensionamento da foto não fosse feito o Android poderia apresentar erros de memória (*OutOfMemory*). Por isso, antes de mostrar o bitmap no `ImageView` é feito o redimensionamento para que a mesma fique somente com o tamanho necessário para ser exibida no `ImageView`. Graças a este controle é possível poupar recursos e memória.

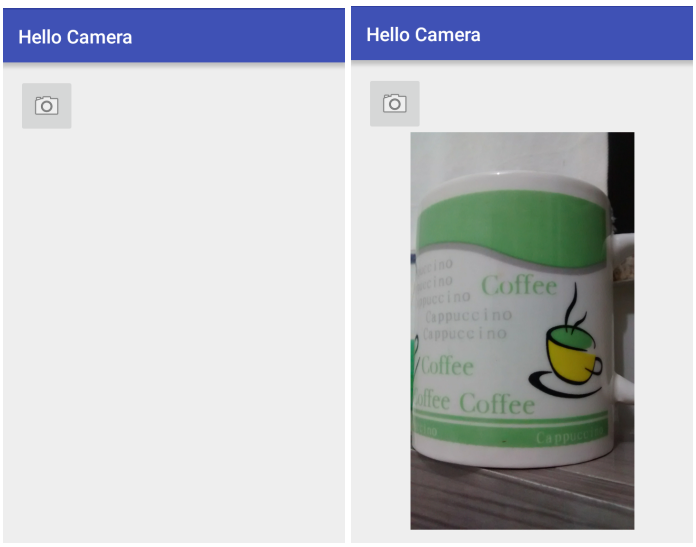

Figura 19. Execução do exemplo Câmera

5. Contador de passos

Ao invés de utilizar o sensor de acelerômetro para calcular a quantidade de passos andados, ou um sensor mais específico como o sensor `STEP_COUNTER`, será utilizada a

plataforma *Google Fit*. Esta plataforma permite salvar e ler os dados dos sensores por meio da nuvem do Google, sendo que ainda é possível centralizar as informações obtidas pelos sensores em sua página oficial[6]. Além disso, a API *Google Fit* é composta por outras subAPIs: `SensorsApi`, `RecordingAPi`, `SesssionApi` e `HistoryApi`.

Como o *Google Fit* faz parte do *Google Play Services*, é necessário conectar aos servidores do Google. A configuração necessária é similar ao que foi utilizado no exemplo do GPS. A forma para habilitar a *Fitness* API pode ser encontrada na página do *console do Google Developers*[7]. Deve-se acessar a opção *Criar credenciais*, selecionar *ID do cliente OAuth* e marcar a opção *Android*. Na tela que aparecerá deverá ser informado um nome qualquer para o ID desta chave, nome do pacote de seu projeto e o certificado SHA-1 (a seção de GPS mostra como obter este certificado). Além de criar a credencial, é necessário ativar a *Fitness* API; para tal, basta pesquisar por `fit`, selecionar a API em questão e ativá-la. Para poder utilizar a API *Fit* também é necessário declarar a dependência do *Google Play Services* no arquivo `app/build.gradle`:

```
...
compile 'com.google.android.gms:play-services:9.0.2'
```

No arquivo `AndroidManifest.xml`:

```
...
<uses-permission android:name="android.permission.INTERNET" />
<application ...
    <!-- Google Play Services -->
    <meta-data
    android:name="com.google.android.gms.version"
        android:value="@integer/google_play_services_version" />
        ...
</application>
```

Neste exemplo, a interface consiste em apenas dois TextView, um com o título `Contador de passos` e o outro será responsável por receber a quantidade de passos que está sendo dado. Segue o arquivo `activity_main.xml`:

```
...
<TextView
    android:id="@+id/textView1"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:gravity="center_horizontal"
    android:text="Contador de passos"
    android:textSize="25sp" />
<TextView
    android:id="@+id/textView2"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:gravity="center_horizontal"
    android:text="---"
    android:textSize="25sp" />
```

[6] *Google Fit* disponível em: `https://fit.google.com/`

[7] *Console do Google Developers* disponíel em: `https://console.developers.google.com/apis/library`

```
...
```

No arquivo `MainActivity.java` foi feito toda a lógica do programa, o método onCreate tem como principal objetivo configurar o objeto `GoogleApiClient` de acordo com as dependências do projeto.

```
@Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        text = (TextView) findViewById(R.id.textView2);

        getSupportActionBar().setDisplayHomeAsUpEnabled(true);

        if (savedInstanceState != null) {
            authInProgress = savedInstanceState.getBoolean(AUTH_PENDING
                );
        }

        mGoogleApiClient = new GoogleApiClient.Builder(this)
                .addApi(Fitness.SENSORS_API)
                .addScope(new Scope(Scopes.FITNESS_LOCATION_READ))
                .addScope(new Scope(Scopes.FITNESS_ACTIVITY_READ))
                .addConnectionCallbacks(this)
                .addOnConnectionFailedListener(this)
                .build();
    }
```

Os métodos `onStart()`, `onDestroy()`, `onConnected()`, `onConnectionSuspended()`, `onConnectionFailed()` e `onActivityResult()` controlam a conexão com o Google Play Services, informando na tela com o método `toast` o status da mesma.

```
 @Override
    protected void onStart() {
        super.onStart();
        // Conecta no Google Play Services
        if (!mGoogleApiClient.isConnecting() && !mGoogleApiClient.
            isConnected()) {
            toast("mGoogleApiClient.connect()");
            mGoogleApiClient.connect();
        }
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        // Desconecta ao sair
        mGoogleApiClient.disconnect();
    }

    @Override
    public void onConnected(Bundle bundle) {
        toast("Conectado no Google Play Services!");
```

```
21
22          startPedometer();
23      }
24
25      @Override
26      public void onConnectionSuspended(int cause) {
27          toast("Connection interrompida.");
28      }
29
30      @Override
31      protected void onSaveInstanceState(Bundle outState) {
32          super.onSaveInstanceState(outState);
33          outState.putBoolean(AUTH_PENDING, authInProgress);
34      }
35
36      private void toast(String s) {
37          Toast.makeText(getBaseContext(), s, Toast.LENGTH_SHORT).show();
38      }
39
40      @Override
41      protected void onActivityResult(int requestCode, int resultCode,
            Intent data) {
42          if (requestCode == REQUEST_OAUTH) {
43              authInProgress = false;
44              if (resultCode == RESULT_OK) {
45                  // Depois que o user autorizou faz login no Google Play
                        Services
46                  if (!mGoogleApiClient.isConnecting() && !
                        mGoogleApiClient.isConnected()) {
47                      mGoogleApiClient.connect();
48                  }
49              }
50          }
51      }
52
53      @Override
54      public void onConnectionFailed(ConnectionResult result) {
55          if (!result.hasResolution()) {
56              // Se for algum erro de configuracao ou servico mostra
                    alerta
57              GooglePlayServicesUtil.getErrorDialog(result.getErrorCode()
                    , this, 0).show();
58              return;
59          }
60          // Caso contrario pode ser porque o user nao autorizou o acesso
                .
61          if (!authInProgress) {
62              try {
63                  Log.i(TAG, "Attempting to resolve failed connection");
64                  authInProgress = true;
65                  result.startResolutionForResult(MainActivity.this,
66                          REQUEST_OAUTH);
67              } catch (IntentSender.SendIntentException e) {
68                  Log.e(TAG,
69                          "Exception while starting resolution activity",
                                e);
```

```
70                 }
71             }
72         }
```

O método `startPedometer()` é responsável pelo cálculo incremental dos passos feitos e também insere essa informação na segunda `textView` criada anteriormente.

```
1  ...
2      private void startPedometer() {
3          // Listener do Fitness API que conta os passos
4          OnDataPointListener listener = new OnDataPointListener() {
5              @Override
6              public void onDataPoint(DataPoint dataPoint) {
7                  for (Field field : dataPoint.getDataType().getFields())
                       {
8                      if (dataPoint.getDataType().equals(DataType.
                           TYPE_STEP_COUNT_DELTA)) {
9                          Value val = dataPoint.getValue(field);
10                         Log.d("livroandroid", "Valor Pedometro: " + val
                               );
11                         qtdePassos += val.asInt();
12                         runOnUiThread(new Runnable() {
13                             @Override
14                             public void run() {
15                                 text.setText("Passos: " + qtdePassos);
16                             }
17                         });
18                     }
19                 }
20             }
21         };
22 ...
```

Na Figura 20 é apresentado um exemplo do contador de passos em funcionamento.

Observação

Os exemplos usados neste texto podem ser encontrados em `https://github.com/davibernardos/Sensores_REA`.

Agradecimentos

Henrique N. Silva agradece o financiamento fornecido pelo edital 015/2015 - PROGRAD. Andre T. Endo é parcialmente financiado pela CNPq/Brasil processo número 445958/2014-6.

Referências

Ableson, F., Sen, R., King, C., and Ortiz, C. E. (2011). *Android in Action*. Manning Publications Co., Greenwich, CT, USA.

Android.com (2016). Disponível em: http://www.android.com.

Apple.com (2016). Disponível em: http://www.apple.com.

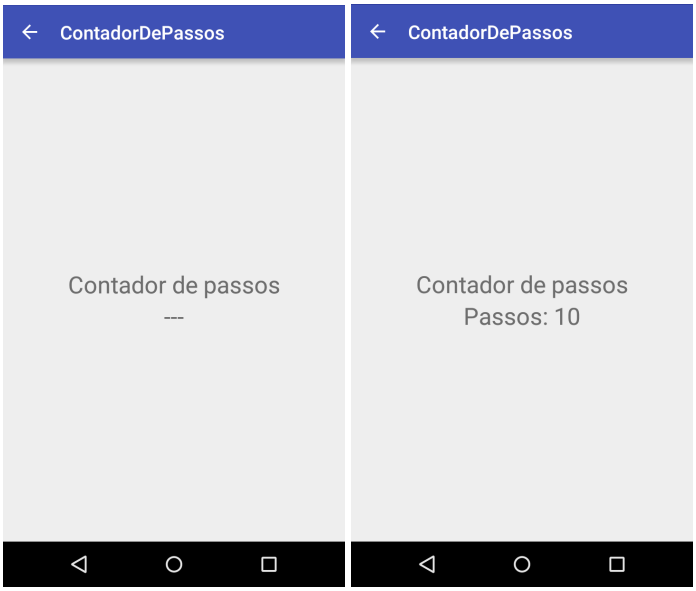

Figura 20. Exemplo do contador de passos

Bhattacharya, P., Ulanova, L., Neamtiu, I., and Koduru, S. C. (2013). An empirical analysis of bug reports and bug fixing in open source android apps. In *Proceedings of the 2013 17th European Conference on Software Maintenance and Reengineering*, CSMR '13, pages 133–143, Washington, DC, USA. IEEE Computer Society.

Developers, A. (2016). Disponível em: http://developer.android.com.

Joorabchi, M., Mesbah, A., and Kruchten, P. (2013). Real challenges in mobile app development. In *Empirical Software Engineering and Measurement, 2013 ACM / IEEE International Symposium on*, pages 15–24, Baltimore, MD. IEEE.

Lecheta, R. (2013). *Google Android: aprenda a criar aplicações com dispositivos móveis com o Android SDK*. Novatec, São Paulo, 3 edition.

Mednieks, Z., Dornin, L., Meike, G. B., and Nakamura, M. (2012). *Programming Android: Java Programming for the New Generation of Mobile Devices*. O'Reilly Media, Sebastopol, 2 edition.

Monteiro, J. (2012). *Google Android - Crie Aplicaçoes para Celulares e: Tablets*. Casa do Codigo, São Paulo, SP, Brasil.

Muccini, H., Francesco, A. D., and Esposito, P. (2012). Software Testing of Mobile Applications: Challenges and Future Research Directions. *7th IEEEACM International Workshop on Automation of Software Test AST 2012 ICSE 2012*, pages 29–35.

Syer, M., Nagappan, M., Adams, B., and Hassan, A. (2015). Studying the relationship between source code quality and mobile platform dependence. *Software Quality Journal*, 23(3):485–508.

Windows.com (2016). Disponível em: http://www.windowsphone.com.

Yang, Z. and Yang, M. (2012). Leakminer: Detect information leakage on android with static taint analysis. In *Proceedings of the 2012 Third World Congress on Software*

Engineering, WCSE '12, pages 101–104, Washington, DC, USA. IEEE Computer Society.