UNIVERSIDADE FEDERAL DE SANTA CATARINA

Cálculos Médicos: um aplicativo Android para cálculos de indicadores na área da saúde

Henio de Oliveira Bez Junior

Florianópolis - SC

2013 / 2

UNIVERSIDADE FEDERAL DE SANTA CATARINA

DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

SISTEMAS DE INFORMAÇÃO

Cálculos Médicos: um aplicativo Android para cálculos de indicadores na área da saúde

Henio de Oliveira Bez Junior

Trabalho de conclusão de curso
apresentado como parte dos requisitos
para obtenção do grau de bacharel em
Sistemas de Informação pela
Universidade Federal de Santa Catarina.

Florianópolis - SC

2013 / 2

Henio de Oliveira Bez Junior

Cálculos Médicos: um aplicativo Android para cálculos de indicadores na área da saúde

Trabalho de conclusão de curso apresentado como parte dos requisitos para obtenção do grau de bacharel em Sistemas de Informação pela Universidade Federal de Santa Catarina.

Banca Examinadora

_____

Orientador: Prof. Dr. Fernando Augusto da Silva Cruz

_____

Prof. Ms. José Eduardo De Lucca

_____

Prof. Dr. João Bosco Mangueira Sobral

Aos meus pais.

Agradecimentos

Não podendo ser diferente, agradeço imensamente aos meus pais por estarem presentes em todos os momentos da minha vida, sempre com muito apoio, carinho e amor. Vocês, papai e mamãe, são o suporte e a motivação para toda minha luta.

Também agradeço à minha namorada, por ocupar um espaço enorme em meu coração, dando-me amor e paz sempre que necessário, tornando-se uma pessoa imprescindível em minha vida.

Aos professores, mestres e doutores, sinto-me na obrigação de agradecer por todo o conhecimento repassado e por todas as experiências compartilhadas, fatos estes que me tornaram uma pessoa melhor e um profissional mais qualificado.

Por fim, agradeço aos amigos, que sempre estiveram presentes, por tornarem tudo mais especial.

Sumário

Lista de Figuras

Lista de Tabelas

Siglas

GPS	Global Positioning System
SMS	Short Message System
VM	Virtual Machine
Java ME	Java Micro Edition
OHA	Open Handset Alliance
API	Appication Programming Interface
HTML	Hiper Text Markup Language
C2DM	Cloud to Device Messaging
NFC	Near Field Communication
VoIP	Voice over Internet Protocol
GCM	Google Cloud Messaging
SGL	Scalable Games Language
SQL	Structured Query Language
SSL	Security Sockets Layer
SO	Sistema Operacional
URI	Uniform Resource Identifier
URL	Uniform Resource Location
IU	Interface com Usuário
XML	Extensible Markup Language
UML	Unified Modeling Language
IDE	Integrated Development Environment

SDK	Software Development Kit
ADT	Android Development Tools
LDL	Low Density Lipoprotein
IMC	Índice de Massa Corporal
INR	International Normalized Ratio
VO2máx	Volume de Oxigênio Máximo

Resumo

Fato consumado nos dias de hoje, o uso de dispositivos móveis, como smartphones e tablets, cresce gradualmente, tornando-se objeto quase que imensurável às pessoas. Acompanhando a popularidade destas tecnologias, cresce também o mercado de desenvolvimento de aplicativos e utilitários destinados a tais aparelhos.

Por baixo da usabilidade dos aplicativos, está presente a camada dos sistemas operacionais. Dentre as plataformas de softwares disponíveis para smartphones e tablets, o Android está presente na maioria dos dispositivos móveis.

A área da saúde possui grande importância a todos os cidadãos. Neste ramo, diariamente são realizados diversos cálculos para avaliação de pacientes.

O aplicativo desenvolvido, chamado *Cálculos Médicos*, automatiza estas pequenas atividades, oferecendo maior segurança e agilidade. Este aplicativo, disponível para a plataforma Android, apresenta uma série de cálculos, classificações e escalas com o objetivo de auxiliar profissionais das áreas de Educação Física, Fisioterapia e Medicina na realização de seus compromissos diários.

O software desenvolvido, além de possuir boa usabilidade, diminui o tempo de execução destas atividades, tornando possível que o profissional da saúde concentre-se em outras atividades.

Palavras Chave: Android, Smartphone, aplicativo, saúde.

Abstract

Nowadays, use of smartphones and tablets has a gradually increase, becoming very important to people. Following these technologies, the market of app development has increased too.

Underneath the app usability, there is an operational systems layer. Among the software platforms available for smartphones and tablets, Android is present in majority.

The health area is very important to all citizens. In this area, every day many calculations are carried out to evaluate patients.

The app developed, named *Cálculos Médicos*, automates small activities, providing greater security and speed. This app, available to Android, consists in calculations, classifications and scales to help professionals of physical education, physiotherapy and medicine in their daily activities.

The software developed, beyond to have good usability, decreases the time to realize these activities, making possible that the professionals of health area focus themselves in other operations.

Keywords. Android, Smartphone, app, health.

1. Introdução

1.1. Contextualização

A computação móvel, nos últimos tempos, teve um crescimento acelerado em todo o mundo, ocasionando uma explosão concomitante no desenvolvimento de softwares e aplicativos para este tipo de tecnologia. Smartphone ou tablet são instrumentos importantíssimos à grande maioria das pessoas, seja para uso pessoal – agenda de compromissos ou de contatos, dentre outros – ou para diversão – jogos e afins. Outra vertente que começa a conquistar mercado é a utilização destes pequenos instrumentos para fins profissionais, como, por exemplo, um arquiteto utilizando um tablet para exibição de projetos a seus clientes, um treinador de futebol exibindo vídeos e táticas a um ou mais atletas e, até mesmo, um médico fazendo cálculos e dando respostas rápidas a seus pacientes.

Por trás da usabilidade de um dispositivo móvel, há um sistema operacional responsável por proporcionar toda a base para que aplicativos sejam instalados da forma mais transparente possível, possibilitando utilização de forma rápida e segura. Existem situações onde há a necessidade de grande processamento de dados, de vultosos espaços para armazenamento e de placas de vídeos com grande poder de processamento. Nestes casos, as possibilidades de uso para smartphones, atualmente, são irrisórias. Contudo, há uma lacuna de encaixe perfeito para os dispositivos móveis, que são os casos de uso onde o tamanho da tela, a qualidade do vídeo e a potência de processamento não são fatores críticos. E é exatamente esta fatia estratégica

do mercado que estes dispositivos, juntamente com seus sistemas operacionais, estão explorando e ocupando.

Os sistemas operacionais que mais se destacam no mundo dos smartphones e tablets, no contexto atual, são Android, iOS, Windows Phone, Symbian OS e BlackBerry OS. Dentre estes, a plataforma Android domina o mercado atual, estando presente em mais da metade dos celulares de todo o mundo. Características que impulsionam a plataforma Android e a fazem ser muito bem aceita pelos usuários são os fatos de ser desenvolvida pelo Google, possuindo diversos serviços do Google embarcados, e de ser um sistema operacional de código aberto, proporcionando liberdade para desenvolvedores e empresas fabricantes. Segundo Lecheta (2010, p20) [1]

> O Android é a resposta do Google para ocupar esse espaço. Consiste em uma nova plataforma de desenvolvimento para aplicativos móveis, baseada em um sistema operacional Linux, com diversas aplicações já instaladas e, ainda, um ambiente de desenvolvimento bastante poderoso, ousado e flexível.

1.2. Justificativa

A venda de telefones celulares cresce a cada dia. No Brasil, a telefonia móvel já ultrapassou a marca de um celular por habitante (ANATEL, 2010) [2]. Estes números incluem celulares convencionais, utilizados basicamente para execução e recebimento de chamadas de voz e mensagens de texto (SMS), e celulares smartphones, usados para, além do uso convencional, acesso à internet (e-mail, online banking, sites variados), multimídia, câmera, GPS e

inúmeros aplicativos. Dentro deste mundo de quase duzentos milhões de aparelhos, a quantidade de smartphones também cresce de forma vultosa. Em 2012, as vendas desse tipo de dispositivo cresceram 78% em relação ao ano anterior, alcançando mais de quinze milhões de unidades vendidas em um ano (REUTERS, 2012) [3]. O crescimento no número de vendas deve continuar, visto que os baixos preços, aplicados por fabricantes e vendedores, devem se consolidar, favorecendo os consumidores.

Acompanhando o crescimento da venda dos smartphones, o sistema operacional Android também é o que mais cresce no mundo. Durante os primeiros quatro meses de 2013, Android teve um aumento de quase 80% nas vendas, comparado com o mesmo período do ano anterior. Com isso, contabilizando somente as vendas de 2013, o Android é parte integrante de 75% dos novos aparelhos (EXAME ABRIL, 2013) [4].

Mesmo com todo este domínio e importância, é uma tarefa não muito fácil encontrar materiais concisos e de qualidade referenciando o assunto. Facilmente, encontram-se tutoriais simples e técnicos, mas sem fundamentação teórica, ou livros extensos e cansativos abordando o tema Android.

O presente trabalho terá como incumbência ocupar este meio pouco explorado, preenchendo esta lacuna de forma satisfatória, oferecendo uma ferramenta preciosa para quem procura conhecer os principais pontos teóricos sobre o ambiente de software Android. Para cumprir esta expectativa, oferecer-se-ão tópicos importantes, que partirão desde o histórico da plataforma até as tendências atuais de mercado.

Além disso, notou-se outra importante carência no mercado atual. Reconhecidamente, a área da saúde é altamente valorizada e de suma importância para a população em geral, todavia há poucos aplicativos para a plataforma Android destinados a esta área. Partindo deste pressuposto, será desenvolvido um aplicativo que poderá ser comumente utilizado por profissionais da saúde para automatização de tarefas diárias, auxiliando-os de forma positiva no atendimento a pacientes.

1.3. Objetivos

Embasados pela contextualização e pela justificativa, onde estão listados problemas pontuais da tecnologia em estudo, são listados objetivo geral e objetivos específicos do presente trabalho.

1.3.1. Objetivo Geral

O objetivo geral é desenvolver um aplicativo com enfoque na área da saúde. Este aplicativo consistirá em um programa mestre, composto por diversas funcionalidades que auxiliarão os profissionais da saúde em suas atividades diárias.

1.3.2. Objetivos Específicos

Os objetivos específicos são:

- Realizar um estudo, listando e explicitando as principais características da plataforma Android, fazendo com que este sirva de referência e ferramenta de pesquisa para futuras obras e desenvolvimentos acerca do assunto;
- Adquirir conhecimento do estado da arte no que tange o tema Android;
- Realizar um estudo junto a especialistas de diferentes áreas da saúde, buscando quais indicadores da sua área de atuação se enquadram nos objetivos do presente trabalho, verificando a viabilidade de inseri-los no aplicativo que será desenvolvido;
- Fazer uma documentação UML do aplicativo que será desenvolvido;
- Desenvolver um aplicativo internacionalizado;
- Testar e avaliar o aplicativo desenvolvido.

2. Fundamentação Teórica

Este presente capítulo consiste no cumprimento do primeiro e segundo objetivos específicos do trabalho, que são adquirir conhecimento do estado da arte e elaborar um levantamento teórico sobre a plataforma Android, fornecendo insumo para que estudantes e profissionais da área tenham um material de qualidade para pesquisas acadêmicas e profissionais acerca do assunto.

2.1. O Android

Antes de se explicitar histórico e versões do Android, é de grande valia definir do que se trata o objeto de estudo, ou seja, explicar o que é o Android.

Hoje, um celular possui inúmeras funcionalidades além das duas primordiais, que são conversas telefônicas e envio e recebimento de mensagens de texto. Esta afirmação pode ser facilmente confirmada analisando o tamanho dos aparelhos desde a sua invenção, em meados do século anterior, até os dias atuais.

Nos primórdios, os celulares eram aparelhos de grandes dimensões, principalmente devido à tecnologia pouco avançada da época. Ao passar dos anos, com a progressão da tecnologia, os aparelhos foram diminuindo de tamanho, chegando ao ápice nos primeiros anos do atual século. Após este fenômeno, os aparelhos começaram a aumentar de tamanho novamente, chegando a telas com mais de sete polegadas.

Figura 1 – Evolução no Tamanho dos Aparelhos Celulares.

Esta evolução no tamanho dos aparelhos, observada na Figura 1, tem uma justificativa. No seu surgimento, os celulares convencionais eram utilizados somente para chamadas de voz, e anos depois, como nova funcionalidade, passaram a enviar e receber mensagens de texto. Possuindo somente estas duas utilidades, não havia motivo para os aparelhos serem grandes.

Então, acompanhando o avanço tecnológico, foram lançados aparelhos de pequeno porte, do tamanho aproximado da palma de uma mão. Entretanto a evolução continuou, a tecnologia avançou a passos largos, e foram atribuídas novas funcionalidades aos dispositivos.

Atualmente, chamadas de voz e SMS são apenas duas funcionalidades dos aparelhos celulares, acompanhadas de inúmeras outras, como verificador de e-mail, dispositivo de acesso à internet, GPS, agenda de compromissos, bloco de notas, online banking, câmera fotográfica, reprodutor de filmes, músicas e jogos, dentre outros. Como consequência deste acréscimo de funções, visando suportar todos estes utilitários e manter uma fácil usabilidade, os aparelhos aumentaram de tamanho.

Obviamente existe uma diferença entre celular e smartphone, todavia, no decorrer deste trabalho, ao se utilizar a palavra celular, esta deverá ser entendida como smartphone. Quando for necessário referenciar celulares que somente fazem chamadas de voz e trocam mensagens de texto, utilizar-se-á o termo *celular convencional*.

Nos dias de hoje, os celulares estão mais para computadores pessoais do que para telefones fixos. Para suportar essa gama de utilidades, faz-se necessária uma base bem consistente, com pilares fortes para suportar o peso das aplicações. Justamente neste ponto surge nosso objeto de estudo, a plataforma Android. Mais do que um sistema operacional, Android é um ambiente de software que possui um sistema operacional baseado em Linux. Além disso, Android engloba outras qualidades, como “uma rica Interface de Usuário, aplicativos de usuário, bibliotecas de código, frameworks de aplicativo e suporte a multimídia” (ABLESON, 2012, p. 4) [5].

Muitas vezes o Android é confundido com uma plataforma de hardware, contudo esta classificação é errônea, pois Android é somente software, um ambiente onde são executados aplicativos nativos e criados por desenvolvedores, indistintamente, rodando em uma máquina virtual, chamada de Dalvik, suportada por um sistema operacional baseado no kernel 2.6 do Linux. A utilização do kernel baseado em Linux para ser o coração do Android é uma característica muito positiva, pois todo gerenciamento de processos e memória é feito de forma transparente, isto é, desenvolvedores não têm a necessidade de se preocupar com estes controles. Além disso, esta característica permite a execução de inúmeros aplicativos de forma concorrente, um em primeiro e outros em segundo plano, possibilitando, por

exemplo, que um aplicativo seja executado enquanto se recebe uma chamada de voz [1].

Falando um pouco sobre o sistema operacional do Android, baseado no Linux, ele é incumbido de fazer todo o trabalho árduo, que normalmente é feito por qualquer outro sistema operacional, como gerenciamento de memória, processos e threads, além de alguns quesitos de segurança, redes e drivers (LECHETA, 2010, p. 23) [1]. O kernel Linux desempenha o papel de abstrair o hardware do aparelho, de torná-lo transparente. É exatamente no kernel que os drivers, como *wi-fi* e Bluetooth, são implementados.

Grande parte dos estudantes e profissionais amantes do Linux acredita que o Android foi o primeiro telefone de código-fonte aberto, todavia, esta informação não é verdadeira. O primeiro celular com Linux foi lançado no ano de 2003 pela Motorola e foi batizado de A760 (Figura 2). Este celular foi muito esperado, pois combinava um sistema operacional Linux com a linguagem de programação Java, sendo o primeiro aparelho da história a oferecer esta combinação.

Figura 2 - Motorola A760, com Linux e Java.

Apesar de não ter sido o primeiro telefone de código-aberto lançado, o Android foi o primeiro a explodir mundialmente, sendo sensação e batendo recordes de vendas, principalmente por ser desenvolvido pelo Google. O peso do Google em conjunto com outras empresas parceiras fortes, como Motorola e Samsung, tornou o Android uma febre em todo o mundo [5].

Uma das características que torna o Android tão popular é a utilização da linguagem de programação Java, característica esta que, por muitos, é considerada o fator crítico de sucesso da plataforma. Não é objetivo deste trabalho explorar a linguagem Java, mas serão citadas duas características positivas de sua utilização que tornam o Android tão atrativo. A primeira delas é o rico e poderoso ambiente de desenvolvimento oferecido, e a segunda é o grande número de desenvolvedores espalhados pelo mundo.

Um dos grandes problemas enfrentados pelo Google neste projeto é justamente causado pela utilização da linguagem de programação orientada a objetos Java, ocasionando, inclusive, em diversos processos por parte da Oracle, empresa detentora do licenciamento da máquina virtual Java (Java VM). Java é diferente das outras linguagens de programação, pois ao invés de ser compilada para um código nativo, Java é compilada para um *bytecode* para, em sequencia, ser executada por uma VM.

O Android não utiliza a Java VM, e sim a Dalvik VM, que é uma máquina virtual própria, otimizada para rodar seus aplicativos. Android não é uma plataforma Java ME, pois “do ponto de vista do desenvolvedor de aplicativos, o Android é um ambiente Java, mas o tempo de execução não é estritamente uma VM Java“ (ABLESON, 2012, p. 13) [5]. Portanto, apesar dos aplicativos

Android serem codificados em Java, o modo de execução ocorre na Dalvik VM, ao invés da máquina virtual licenciada pela Oracle.

2.2. Android e suas camadas

Android, reconhecidamente, é mais que um sistema operacional, sendo classificado por Lecheta como uma plataforma de software, mais precisamente uma pilha composta por cinco camadas [1]. Segundo Ableson (2012, p. 10), “sem o contexto de o Android ser uma plataforma projetada para ambientes móveis, seria fácil confundir o Android com um ambiente computacional geral”, pois, do ponto de vista arquitetural, estão presentes todas as camadas de uma plataforma computacional padrão [5].

Acima do hardware do dispositivo, composto por recursos fundamentais como localizador GPS, Bluetooth e câmera, dentre outros, está localizada a segunda camada da pilha Android, que é justamente o kernel Linux 2.6. Sobrepondo-se à segunda camada, estão as bibliotecas embarcadas, escritas em C e C++, utilizadas por inúmeros componentes do sistema, em conjunto com o Android *Runtime*, composto pela máquina virtual Dalvik.

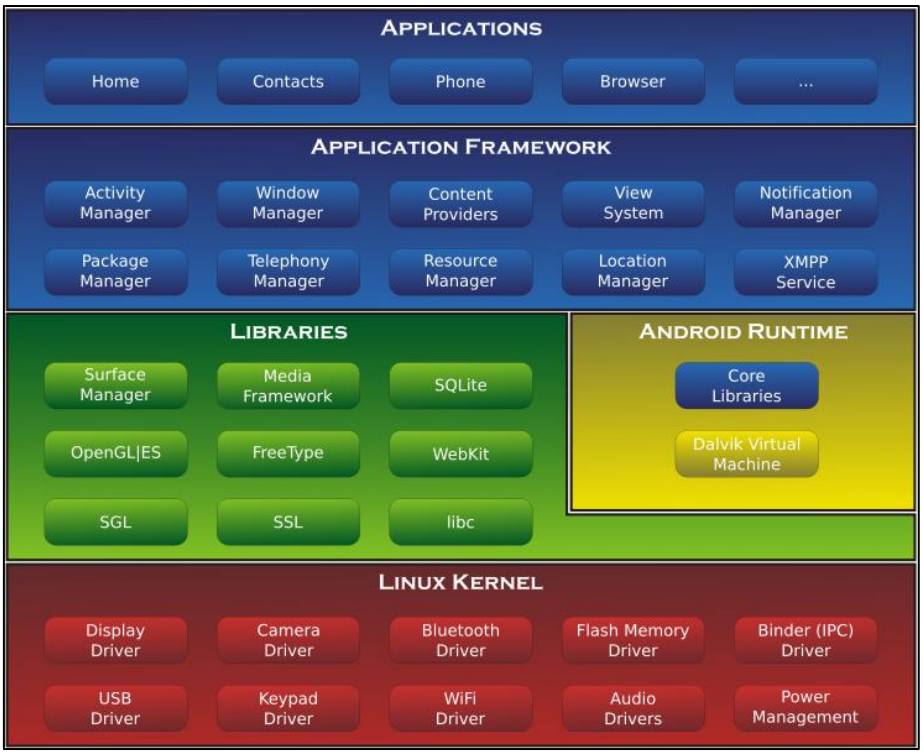

Figura 3 - Arquitetura do Android

Nas bibliotecas desta terceira camada, há importantes implementações, como suporte a bancos de dados, através do *SQLite,* e suporte gráfico com OpenGL ES, para animações em até três dimensões, e SGL, para animações em duas dimensões. Além disso, estão presentes bibliotecas para suporte a formatos de imagens, áudio e vídeo e suporte a comunicação segura, por meio da utilização do *Secure Sockets Layer* (SSL) [5].

A quarta camada é o framework de aplicações, onde estão contidos diversos gerenciadores. Esta camada é escrita na linguagem de programação Java, roda na Dalvik VM, e oferece o suporte necessário para criação de aplicativos, através das bibliotecas nativas. Exatamente neste ponto que são oferecidos gerenciadores para atividades e visualizações, Janelas, serviços baseados em localização, telefonia e toda a gama de recursos disponibilizados

pela plataforma Android [5]. Igualmente, é nesta camada que são oferecidos vários serviços para os aplicativos, com o intuito de reutilizar-se componentes.

Por fim, no topo da pilha da plataforma Android estão localizados os aplicativos dos usuários. Uma característica importante, que torna o Android ainda mais popular, é o fato da inexistência de diferenças entre aplicativos integrados e aplicativos desenvolvidos por terceiros, ou seja, a plataforma oferece um ambiente para desenvolvimento onde se podem utilizar todos os poderosos recursos do sistema.

2.2.1. Kernel Linux

O Android é construído sobre um kernel Linux (versão 2.6) modificado para oferecer recursos imprescindíveis à plataforma, como segurança, agilidade, possibilidade de comunicação em rede e gerenciamento de memória e processos. Em um ambiente dinâmico, faz-se necessária a abstração da camada mais profunda. Esta abstração é feita pelo Linux, separando o hardware das demais camadas de software. Com isso, mesmo havendo mudanças consideráveis no hardware, os níveis superiores não são atingidos, podendo permanecer inalterados [11].

Assim como as outras partes do Android, o kernel modificado também possui código-fonte aberto, podendo ser acessado através do repositório de dados públicos do Android. Neste repositório estão disponíveis versões oficiais e versões de testes ou em desenvolvimento. Devido ao rápido avanço da tecnologia, a troca de aparelhos celulares é algo comum, visto que são

lançados produtos novos a uma velocidade assustadora. O código aberto é um alicerce para que o sistema operacional acompanhe a evolução dos dispositivos, pois empresas e desenvolvedores trabalham e ajudam nesta rápida evolução.

2.2.2. Dalvik VM

Um dos principais problemas encontrados pelo Google no desenvolvimento e utilização do Android é o emprego do Java como linguagem de programação para codificar aplicativos, visto que há direitos autorais e de licenciamento da Oracle para a utilização da máquina virtual Java.

No Android, o ambiente de codificação é o Java, mas ao invés de utilizar a Java VM, da Oracle, utiliza-se uma máquina virtual própria, chamada Dalvik. Esta se difere da Java VM, pois não interpreta Java *bytecodes*, e sim arquivos com uma extensão própria, Dalvik *Executable* (*.dex)*. De acordo com Ableson (2012, p. 12) [5] os arquivos *.dex*

> São equivalentes lógicos dos códigos de byte Java, mas permitem ao Android executar seus aplicativos na sua própria VM, que é ao mesmo tempo livre das limitações de licenciamento da Oracle e uma plataforma aberta na qual o Google [...] pode melhorar conforme necessário.

Resumidamente, o processo de desenvolvimento e compilação ocorre de maneira simples e direta. As aplicações Android são desenvolvidas em Java, utilizando-se todos os recursos desta linguagem. Como uma Java VM padrão, os arquivos *.java* são compilados, gerando os arquivos bytecodes

.class correspondentes. Após isso, eis o que difere Java ME do Android, os arquivos *.class* são convertidos para extensões *.dex*, que correspondem à aplicação do Android compilada. Para se obter o arquivo final que será instalado nos dispositivos móveis, basta-se compactar os arquivos *Dalvik Executable* e outros arquivos necessários, como sons e imagens, em um único arquivo *Android Package File* (*.apk*) [1].

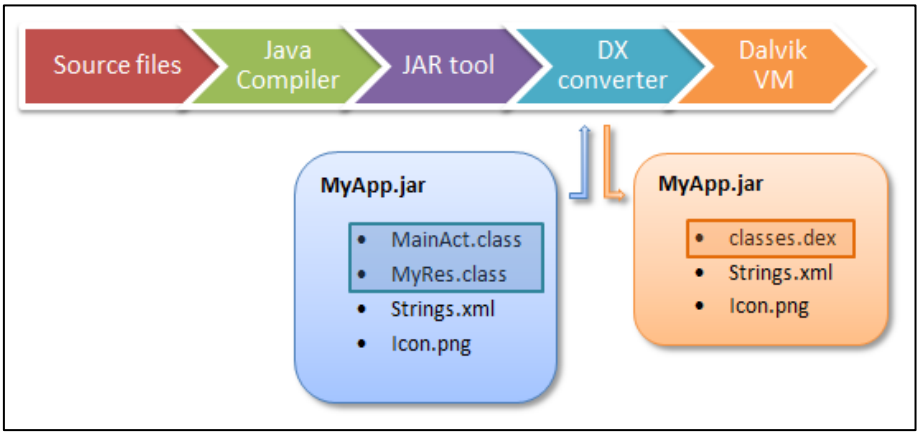

Figura 4 - Tempo de compilação Android.

Dalvik VM não é compatível com a Java VM, uma vez que aquela é uma máquina virtual com modificações necessárias para rodar em ambientes com pouco poder de processamento, ou seja, é uma máquina virtual otimizada e especializada para pequenos sistemas. Outros aspectos importantes que distinguem as duas máquinas virtuais é o fato de que Dalvik foi projetada para possibilitar a execução de várias instâncias concomitantemente, além de deixar tarefas secundárias aos cuidados do kernel Linux. Portanto, considerando estas perspectivas, pode-se afirmar que Android não é uma plataforma Java ME [11].

2.3. Intent e IntentFilter

Até então foram apresentados aspectos históricos, mercadológicos e estruturais da plataforma Android. A partir de agora, será explorada a vertente da concepção de aplicativos, que são desenvolvidos, como já citado anteriormente, utilizando a linguagem de programação Java. O ponto de partida será uma classe muito importante, a Intent.

Intent representa a intenção da aplicação em realizar uma tarefa, como, por exemplo, abrir um site na internet ou mostrar a lista de contatos registrados no telefone. Segundo Lecheta (2010, p.135) [1], a classe *android.content.Intent* "é o coração do Android. Um Intent está presente em todos os lugares e representa uma mensagem da aplicação para o sistema operacional, solicitando que algo seja realizado". Portanto, Intent refere-se à intenção de realizar uma operação, como, por exemplo, enviar e recuperar dados.

De acordo com Ableson (2012, p.14) [5], Intent "é uma declaração de necessidade. Ele é composto de diversas informações que descrevem a ação ou serviços desejados". Um Intent fornece uma facilidade para a realização de integração entre códigos de diferentes aplicações em tempo de execução, isto é, oferece embasamento para que aplicações que estão rodando em processos diferentes se comuniquem.

É justamente este atributo que torna o entendimento e uso do Intent algo muito importante aos desenvolvedores de aplicativos Android. Por exemplo, ao utilizar um Intent para abrir um navegador de internet, outra aplicação, nativa do Android, executará essa atividade, e o comando para tal execução é feito

através do Intent, gerenciado pelo sistema operacional. Portanto, aplicativos podem utilizar funcionalidades de outros através da criação e envio de um Intent, que requisitará o código alheio para manipular o Intent. Isto é usado, por exemplo, para buscar um contato no telefone [5].

A intenção de realizar determinada tarefa é enviada ao sistema operacional na forma de uma mensagem, chamada de *broadcast*. O Intent pode ser utilizado para inúmeros fins além de enviar uma mensagem ao SO, como abrir uma interface na aplicação, solicitar ligação, acessar determinado site, exibir localização no Google Maps e instalar um aplicativo [1].

Na prática, utilizam-se URIs como componentes de dados do Intent. Um Intent é criado passando como parâmetros a ação que se deseja executar e o URI correspondente. Por exemplo, para acessar o site da Universidade Federal de Santa Catarina, um parâmetro na criação do Intent será o URI obtido através do parse da URL *http://www.ufsc.br*. O outro parâmetro será a ação que se deseja executar, no caso, a ação que fará com que o navegador web seja aberto e o site informado seja acessado, que é a *ACTION_VIEW*. Depois de criado o Intent, basta chamar o método *startActivity*, da classe *Activity*, passando como parâmetro o Intent criado.

Um aspecto importante da plataforma de software Android é que não há diferenças entre criar um Intent que será respondido pela própria aplicação e um que será respondido pelo sistema operacional [1].

Um objeto Intent é formado por seis componentes: Action, Category, Component, Data, Extras e Type. Estes componentes devem ser utilizados para descrever a tarefa que deverá ser realizada, sendo que a própria

plataforma Android utilizará estas informações para definir qual classe executará a requisição. Dentre os principais tipos de componentes Android, que serão explicados no decorrer deste trabalho, três deles podem manipular requisições do Intent, são eles: Activity, BroadcastReceiver e Service. Esta configuração é feita no arquivo de configuração da aplicação Android, *AndroidManifest.xml*, utilizando a tag do elemento IntentFilter [5].

Abaixo seguem as principais ações utilizadas na instanciação de objetos da classe Intent.

Ação	Descrição	Valor da Constante
ACTION_MAIN	Indica o ponto de entrada da aplicação.	android.intent.action.MAIN
ACTION_VIEW	Exibe dados para o usuário.	android.intent.action.VIEW
ACTION_ATTACH_DATA	Indica que parte de algum dado deve ser anexado em outro lugar.	android.intent.action.ATTACH_DATA
ACTION_EDIT	Possibilita edição de dados fornecidos.	android.intent.action.EDIT
ACTION_PICK	Escolhe um item de um dado, retornando o que foi selecionado.	android.intent.action.PICK
ACTION_DIAL	Mostra ao usuário a interface com um número a ser discado, possibilitando início de uma chamada.	android.intent.action.DIAL
ACTION_CALL	Inicia uma chamada.	android.intent.action.CALL
ACTION_SEND	Envia algum dado para alguém.	android.intent.action.SEND
ACTION_SEND_TO	Envia algum dado para alguém específico.	android.intent.action.SENDTO
ACTION_ANSWER	Lida com uma chamada telefônica.	android.intent.action.ANSWER
ACTION_INSERT	Insere um item vazio.	android.intent.action.INSERT

ACTION_DELETE	Exclui um dado.	android.intent.action.DELETE
ACTION_RUN	Executa um dado.	android.intent.action.RUN
ACTION_SYNC	Faz uma sincronização de dados.	android.intent.action.SYNC
ACTION_PICK_ACTIVITY	Escolhe uma atividade dada uma intenção, retornando a classe selecionada.	android.intent.action.PICK_ACTIVITY
ACTION_SEARCH	Faz uma pesquisa.	android.intent.action.SEARCH
ACTION_WEB_SEARCH	Faz uma pesquisa na internet.	android.intent.action.WEB_SEARCH

Tabela 1 - Principais ações utilizadas em Intents

Como já dito, o compartilhamento de código é um dos aspectos que fazem o Intent ser tão importante para o Android. Aplicativos usam uns aos outros através da criação e manipulação de Intents. Uma vantagem que essa característica proporciona é a ocorrência da utilização de interfaces comuns para as mesmas atividades, facilitando o uso por parte do usuário, pois para uma mesma ação sempre será mostrada uma mesma interface. Essa vantagem é importante, principalmente para dispositivos móveis, onde a grande maioria dos usuários não são conhecedores profundos da tecnologia e não desejam aprender várias maneiras diferentes de executar a mesma atividade [5].

Abaixo seguem alguns exemplos da utilização do Intent, com suas respectivas ações, para enviar uma mensagem ao sistema operacional, que abrirá alguma tela nativa da plataforma.

Ação	Código Java
Abrir um site	Uri uri = Uri.parse(“http://www.ufsc.br”); Intent intent = new Intent(Intent.ACTION_VIEW, uri); startActivity(intent);

Fazer uma ligação	Uri uri = Uri.parse(“tel:99999999”); Intent intent = new Intent(Intent.ACTION_CALL, uri); startActivity(intent);
Solicitar exibição do mapa do Google Maps em uma coordenada específica	Uri uri = Uri.parse(“geo:-25.1234, -50.1234”); Intent intent = new Intent(Intent.ACTION_VIEW, uri); startActivity(intent);
Visualizar todos os contatos da agenda	Uri uri = Uri.parse(“content://contacts/people”); Intent intent = new Intent(Intent.ACTION_PICK, uri); startActivity(intent);

Tabela 2 - Exemplos de ações utilizando Intent

Resumidamente, segundo Lecheta (2010, p.159) [1], “é possível utilizar o método *startActivity(intent)* informando um Intent que pode conter informações para iniciar uma Activity da própria aplicação ou conter informações para iniciar uma Activity nativa do Android”. O Intent indica uma intenção de realizar uma ação, portanto esta ação deve ser enviada ao sistema operacional, em forma de mensagem, para que seja executada, ou pela aplicação local ou por uma aplicação que está rodando em outro processo.

Um exemplo que torna a compreensão do Intent bastante intuitiva é dado pelo mesmo Lecheta (2010, p.160) [1]. Após o dispositivo móvel receber uma mensagem SMS,

> O que acontece internamente antes de a mensagem aparecer na caixa de entrada é que um Intent com o conteúdo da mensagem é disparada ao sistema operacional para que as aplicações interessadas e com permissão possam interceptar a mensagem SMS. [...] a aplicação da caixa de entrada (inbox) é que faz isso por padrão.

Um Intent pode ser classificado como implícito ou explícito. A diferença básica entre os dois é que no explícito a classe que deve manipular o Intent é especificada através da passagem do nome desta classe como argumento na

criação do Intent. Já no Intent implícito, a escolha da classe que executará a requisição depende do IntentFilter e do sistema operacional Android. Quando um Intent é enviado ao sistema operacional, este analisa Activities, Services e BroadcastReceivers (estes três itens serão explicados no decorrer deste trabalho) que estão cadastrados e disponíveis, enviando a requisição para o contêiner que possui as melhores indicações para resolver o problema [5].

Portanto, três dos principais componentes das aplicações Android – Activity, Service e BroadcastReceiver – são chamados por mensagens enviadas ao sistema operacional, isto é, pelos Intents. Esta característica é considerada uma facilidade para, em tempo de execução, conectar componentes da mesma ou de diferentes aplicações. Para informar ao sistema quais os Intents implícitos um componente pode manipular, Activity, Service e BroadcastReceiver podem ter um ou mais IntentFilters. Cada filtro utilizado descreve a capacidade do componente em receber um conjunto de intenções para tratar. Vale ressaltar que, para um Intent explícito, os filtros não são verificados, pois a requisição é enviada diretamente a classe indicada no Intent [12].

Já foi definido que o Intent é uma declaração de necessidade. O IntentFilter é a declaração de que uma classe tem capacidade e disponibilidade para dar assistência àqueles em necessidade, definindo uma relação entre o Intent e o aplicativo. Um componente, quando possui determinado IntentFilter, está apto a executar chamadas para o respectivo Intent. O IntentFilter pode ser relacionado exclusivamente ao Intent, à ação ou a ambos, possuindo o campo categoria que ajuda a classificar a ação. IntentFilter é definido no principal arquivo de configuração do Android, *AndroidManifest.xml*, com a tag <*intent-*

filter>. Concisamente, o Intent é uma mensagem que tem uma intenção para realização de uma tarefa, enquanto o IntentFilter tem a responsabilidade de filtrar os Intents por ação e categoria.

O campo categoria nada mais é que uma String contendo informações adicionais sobre o tipo de componente que pode manipular o Intent. Um objeto Intent pode possuir quantas categorias forem necessárias, sendo que as principais são [12]:

- CATEGORY_BROWSABLE: usado para indicar que a Activity pode, de forma segura, mostrar dados referenciados por um link, como imagens;
- CATEGORY_GADGET: indica que uma Activity pode ser incorporada dentro de outra que hospeda *gadgets*;
- CATEGORY_HOME: usado para que a tela inicial do dispositivo seja mostrada ao usuário;
- CATEGORY_LAUCHER: indica que a Activity inicial do sistema, isto é, o *entry point*;
- CATEGORY_PREFERENCE: indica que a Activity é um painel de preferencias.

2.4. Principais Componentes do Android

No item anterior, foi citado que um aplicativo Android possui alguns componentes primordiais, que podem ser definidos como os principais blocos de construção de uma aplicação Android. Cada um desses componentes é um diferente ponto por onde o sistema pode entrar na aplicação, assim sendo, uma

aplicação pode não possuir os quatro componentes, mas deverá ter, pelo menos, um deles, que, neste caso, servirá como único ponto de entrada [12].

Existem quatro componentes principais em uma aplicação Android, que são: Activity, Service, BroadcastReceiver e ContentProvider. Cada um deles possui um objetivo diferente e um ciclo de vida distinto, que define como e quando o componente é criado e destruído.

2.4.1. Activity

Este é o componente mais utilizado e mais intuitivo, isto é, com maior facilidade de compreensão e entendimento. Cada Activity representa uma tela do sistema, ou seja, é uma interface de interação com o usuário. Esta não é uma regra geral, mas em quase todos os casos, há uma relacionamento um-para-um entre a quantidade de telas e o número de Activities do sistema. Segundo exemplo oficial na documentação do Android (2013) [13], uma aplicação de e-mail deve conter uma Activity para mostrar a lista de novos e-mails, outra Activity para composição de um e-mail e outra Activity para leitura dos e-mails, ou seja, uma Activity para cada interface de interação com o usuário.

Uma Activity é implementada como subclasse da *android.app.Activity*. Segundo Ableson (2013, p.18) [5], uma Activity possui várias funções, sendo que uma das principais é “exibir os elementos de IU, que são implementados como *Views* e são geralmente definidos em arquivos de layout XML”. Para mudar de tela, ou seja, para invocar outra Activity, utiliza-se o método *startActivity(intent)*, passando como argumento a instância de um Intent. Cada

Activity do sistema tem a incumbência de responder aos eventos iniciados na tela, além de definir qual *View* será responsável por desenhar a IU.

A interface fornecida por uma Activity é formada por componentes chamados *View*, que são, concisamente, o que o usuário do aplicativo vê e interage. Ao iniciar uma Activity, deve ser implementado o método *onCreate*, que será responsável pela composição da Activity. Este método é obrigatório e deve ser chamado somente uma vez. Dentro do *onCreate*, deve-se invocar outro, chamado *setContentView*, que receberá como argumento a *View* associada à Activity. Cada *View* é um objeto do tipo *android.view.View* [1].

*View*s são utilizadas para manipulação de layout das telas, fornecendo elementos de texto para títulos e feedback, botões e formas para entradas do usuário, além de desenhar imagens da tela do dispositivo. Views utilizam recursos gráficos, como Strings, cores e estilos, que são compilados pelo Android e disponibilizados para os aplicativos na forma de resources (Ableson. 2012, p.65) [5].

Neste ponto surge um novo conceito do Android, a classe *R.java*, que instancia atributos para referenciar resources individuais e faz a ligação entre referências binárias e código fonte do aplicativo. É importante ressaltar que há um relacionamento entre Activity, View e resource. De forma simples, uma Activity é composta por uma árvore hierárquica de elementos View, e estas utilizam resources na sua composição.

Exemplificando, as Views podem ser facilmente manipuladas, pois há inúmeros métodos na classe principal View, de onde todas as outras são filhas. Estes métodos possibilitam alterações na interface em geral e em todos os

componentes que forma esta interface, como definir layout, alterar margens, alinhamento, largura, altura, cores, dentre outros [5]. As Views são, segundo padrão MVC, definidas em arquivos XML, contudo, elementos da interface podem ser diretamente manipulados via código Java. A tabela 3 mostra alguns desses métodos.

Método	Descrição
setBackgroundColor(int color)	Altera a cor de fundo.
setClickable(boolean click)	Altera a possibilidade de o elemento ser clicado.
setFocusable(boolean focus)	Altera a possibilidade de o elemento poder ser focado.
setMinimumHeight(int minHeight)	Define a altura mínima do elemento.
setMinimumWidth(int minWidth)	Define a largura mínima do elemento.
setOnClickListener(OnClickListener click)	Configura um ouvinte para disparar no evento de clique.
setOnFocusChangeListener(OnFocus ChangeListener focus)	Configura um ouvinte para disparar quando ocorrer mudança de foco em um elemento.
setPadding(int left, int right, int top, int bottom)	Altera margens internas do componente.
setGravity(int gravity)	Configura alinhamento do componente.
setHeight(int height)	Altera a altura.
setWidth(int width)	Altera a largura.
setText(CharSequence text)	Altera o texto a ser exibido em um *TextView*

Tabela 3 – Manipulando Views via código Java

Fechando os parênteses onde foi explicado o componente View, o assunto volta a ser Activity, que, como já dito, é um componente da aplicação que provê uma interface onde o usuário pode interagir ordenando uma ação, como fazer uma ligação, tirar uma foto ou visualizar um mapa, sendo que uma Activity responde a eventos iniciados tanto pelo usuário quando pelo sistema operacional.

Uma aplicação nada mais é do que um conjunto de Activities ligadas umas às outras, possuindo um fraco acoplamento, uma vez que quando

iniciada uma nova Activity, passando como argumento o Intent, qualquer outra presente no sistema poderá executar a próxima ação, seja pertencente ao mesmo processo ou a outro. Como já mencionado, uma Activity pode iniciar outra, e cada vez que uma nova Activity é iniciada, a anterior é parada, sendo armazenada pelo sistema em uma estrutura de dados do tipo pilha, chamada *back stack*. Esta pilha utiliza o mecanismo FIFO (*first in, first out*), portanto quando uma Activity é finalizada, a próxima que será executada é a que está no topo da *back stack* [13].

Sempre que uma Activity possui seu estado alterado, é feita uma notificação e é chamado um método de *callback*, onde podem ser implementadas situações de retorno para que não ocorram ações indesejadas. Há vários métodos de *callback*, chamados quando o estado da Activity é alterado para criada, iniciada, executando, pausada, parada ou destruída. Estes estados serão vistos detalhadamente no próximo tópico, onde serão explicitadas todas as etapas do ciclo de vida de uma Activity.

2.4.1.1. Ciclo de Vida de uma Activity

Um aspecto imprescindível para os desenvolvedores da plataforma Android é o conhecimento detalhado do ciclo de vida de uma Activity. Segundo Lecheta (2010, p.94) [1], “o importante é entender que o sistema operacional cuida desse ciclo de vida, mas ao desenvolver aplicações é importante levar cada estado possível em consideração para desenvolver uma aplicação mais robusta”.

Por uma questão de arquitetura, o sistema operacional do Android, baseado no kernel Linux, tem total controle sobre a execução das Activities, portanto, se for necessário para o sistema o fechamento de uma tela (que corresponde a uma Activity), esta será encerrada. Neste caso, entendendo o ciclo de vida da Activity, o desenvolvedor poderá implementar ações para que não ocorram perdas de dados ou informações da aplicação.

Conforme mencionado anteriormente, as Activities em execução são armazenadas em uma pilha chamada *back stack*. Em determinado momento, o sistema operacional estará executando a Activity que está no topo da pilha, fazendo com que as demais executem em segundo plano ou fiquem em estado de pausa. Se o SO considerar necessário o encerramento da execução de uma ou mais Activities, as que estão em estado de pausa serão as primeiras a serem encerradas. Isto pode acontecer, por exemplo, quando o SO observa a necessidade da liberação de recursos ou memória [1].

Mais detalhadamente, pode-se dizer que, quando a plataforma verifica a necessidade de liberação de recursos, utiliza critérios para reduzir a quantidade de processos e Activities em execução. De acordo com Ableson [5], os critérios utilizados são:

- Os primeiros processos encerrados, em caso de necessidade, serão os processos rodando em vazio, isto é, sem nenhuma Activity associada;
- Os próximos processos a serem encerrados serão os que hospedam Activities não visíveis que estão rodando em segundo plano;

- Após isso, o sistema operacional, se ainda necessário, encerrará Activities visíveis rodando em segundo plano;
- Os processos mais importantes e, consequentemente, os últimos que serão encerrados, serão os processos que hospedam Activities rodando em primeiro plano, isto é, a Activity que está no topo da *back stack*. Este processo raramente é pausado ou parado.

Entendida a sequência de encerramento dos processos que carregam as Activities, será iniciado o estudo do ciclo de vida. O gerenciamento do ciclo de vida de uma Activity deve ser implementado por meio da utilização de métodos de callback, tornando a aplicação desenvolvida mais robusta e flexível. Resumidamente, os métodos de callback são aqueles invocados após a mudança de estado de uma Activity.

Uma Activity possui três estados fundamentais: executando, pausada e parada. A Activity estará no estado executando quando estiver com foco em si e em primeiro plano, ou seja, é o elemento presente no topo da *back stack*. Uma Activity pausada é aquela que ainda está ativa, mas sem o foco, podendo ser encerrada caso o sistema operacional necessite de liberação de recursos ou memória, conforme critérios apresentados anteriormente. Por fim, uma Activity estará no estado parada quando estiver em segundo plano, ou seja, quando houver uma outra Activity executando em primeiro plano. Uma Activity parada ainda é considerada ativa [13].

Quando há a transição da Activity de um estado para o outro, um método de callback é chamado. Os métodos de callback existentes estão na classe mãe Activity, e são herdados na implementação quando a Activity criada

estender *android.app.Activity*. De acordo com a necessidade, o desenvolvedor deve sobrescrever os métodos de callback para que sejam tomadas medidas necessárias quando o estado da Activity é alterado [13]. Por exemplo, se um usuário está manipulando algum aplicativo de inserção de dados e recebe uma ligação, este aplicativo será posto em segundo plano enquanto ocorre o atendimento. Neste caso, deve-se sobrescrever um ou mais métodos de callback para que, quando a ligação for finalizada, o aplicativo retorne ao estado que estava, sem que haja perda de dados.

Na Figura 5 é possível observar a classe *ExampleActivity*, que estende *Activity*, sobrescrevendo os principais métodos de callback existentes no ciclo de vida de uma Activity.

Abaixo seguem os métodos de callback existentes e uma pequena descrição de cada um deles:

- *onCreate()*
    - Chamado na criação da Activity. Este método deve ser invocado somente uma vez, de forma obrigatória. Dentro dele, sempre haverá a criação de uma View, sendo associada à Activity. Logo após a execução deste método, o *onStart()* é chamado automaticamente, dando início ao ciclo de vida visível da Actvity;
    - Próximo método chamado: *onStart().*

```
public class ExampleActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // The activity is being created.
    }
    @Override
    protected void onStart() {
        super.onStart();
        // The activity is about to become visible.
    }
    @Override
    protected void onResume() {
        super.onResume();
        // The activity has become visible (it is now "resumed").
    }
    @Override
    protected void onPause() {
        super.onPause();
        // Another activity is taking focus (this activity is about to be "paused").
    }
    @Override
    protected void onStop() {
        super.onStop();
        // The activity is no longer visible (it is now "stopped")
    }
    @Override
    protected void onDestroy() {
        super.onDestroy();
        // The activity is about to be destroyed.
    }
}
```

Figura 5 – Principais métodos de callback de uma Activity

- *onStart()*
    - Chamado quando a Activity já possui uma View e está em iminência de se tornar visível;
    - Próximo método chamado: *onResume()* ou *onStop().*
- *onRestart()*
    - Chamado quando a Activity está sendo reiniciada, ou seja, ainda está na pilha de Activities, todavia não estava em execução;
    - Próximo método chamado: *onStart()*.

- *onResume()*
    - Chamado quando a Activity recebe o foco de execução, ou seja, quando está rodando em primeiro plano. Este método é invocado quando a Activity está no topo da *back stack*, quando começa a interagir com o usuário. Este método é chamado sempre após o *onStart()*;
    - Próximo método chamado: *onPause()*.
- *onPause()*
    - Chamado quando outra Activity recebe o foco, ou seja, quando a Activity atual deixa de executar em primeiro plano, deixando, também, o topo da pilha de Activities. Neste método, devem-se salvar informações sobre o estado atual da aplicação, para que, quando retornar à execução, recupere o contexto sem perder dados;
    - Próximo método chamado: *onResume()* ou *onStop().*
- *onStop()*
    - Chamado quando a Activity está sendo encerrada, não estando mais visível ao usuário. Após este estado, a Activity pode ser reiniciada ou destruída;
    - Próximo método chamado: *onRestart()* ou *onDestroy().*
- *onDestroy()*
    - Este método é chamado quando uma Activity encerra sua execução, ou seja, quando é removida de forma irreversível da memória do sistema. Este método pode ser

chamado automaticamente pelo SO, visando à liberação de recursos e memória.

- Próximo método chamado: nenhum.

Além da existência destes métodos fundamentais, há outros menos importantes no ciclo de vida de uma Activity, como *onPostCreate, onPostResume* e *onSaveInstanceState*. Estes métodos, normalmente não são sobrescritos na implementação de uma Activity.

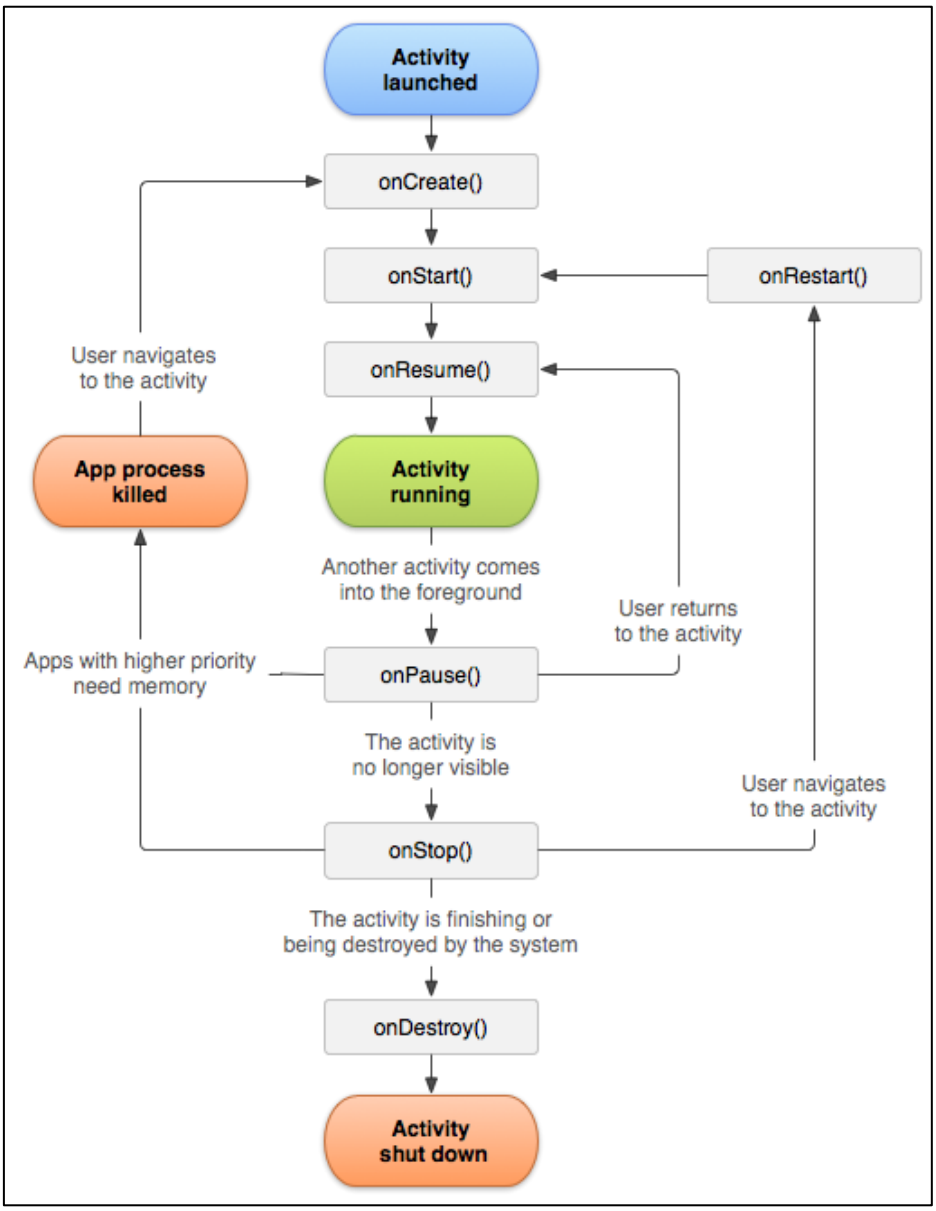

Figura 6 – Ciclo de vida completo de uma Activity

Observando a Figura 6, referente ao ciclo de vida completo de uma Activity, fica explícito que uma Activity nasce com a invocação do método *onCreate*. Após a criação da Activity, sempre será chamado o método de callback *onStart* que, de maneira obrigatória, chamará o *onResume*. Após a execução deste último método, a Activity estará com foco e executando em primeiro plano.

Quando uma Activity tiver sua execução interrompida, o primeiro método a ser chamado será o *onPause*. Este método de callback tem importância imensurável no ciclo de vida, pois é a ultima chance que o desenvolvedor terá para limpar dados ou salvar informações de estado. Seguindo as ideias de Ableson (2012, p.75) [5]

> Os processos que hospedam suas classes Activity não serão finalizados pela plataforma até que o método *onPause* tenha sido completado, mas eles podem finalizar depois disso. O sistema vai tentar rodar todos os métodos de ciclo de vida a cada vez, mas se os recursos diminuírem criticamente, os processos que estão hospedando Activities que além do método *onPause, podem ser finalizados em qualquer ponto.*

Após o método *onPause*, a Activity poderá seguir por dois caminhos distintos: retornar à execução ou encaminhar para encerramento. Se a Activity retornar à execução, será chamado o método *onResume*. Caso seja encaminhada para encerramento, a Activity pode seguir por outros dois caminhos. No primeiro deles, considerado o caminho natural da Activity, é chamado o método *onStop*. A partir deste estado, a Activity pode ser reiniciada, através do *onRestart*, pode ser finalizada pelo SO ou pode seguir para o

onDestroy. No segundo caminho, a Activity é finalizada pelo SO visando à liberação de recursos de processamento e memória.

2.4.2. Service

Assim como a Activity, Service é um dos principais componentes do Android. Um Service deve ser utilizado quando se deseja executar operações de longa duração em segundo plano, sem oferecer uma interface para interação com o usuário. Uma aplicação pode iniciar um Service, que será executado em segundo plano, mesmo que a aplicação que o iniciou já tenha sido finalizada [14], ou seja, depois que um Service foi iniciado, ele executará até ser parado explicitamente ou até ser interrompido pelo sistema operacional.

Para iniciar uma Activity, deve ser criada uma classe estendendo *android.app.Activity*. Um Service também deve ser criado como subclasse da classe mãe *android.app.Service*. Outra semelhança com as Activities, é que um Service também é iniciado chamando um método padrão, no caso, *startService(Intent)*, que também recebe uma intenção como argumento. “Criamos um Service de maneira semelhante a como criamos Activities e receptores de Broadcast: estendemos a classe básica, implementamos os métodos abstratos e redefinimos os métodos de ciclo de vida conforme necessário” (Ableson. 2012, p.120) [1]. Assim como Activities, Services também possuem métodos de ciclo de vida que devem ser utilizados para alocar e liberar recursos, de modo similar ao feito no controle do ciclo de vida das Activities.

Um Service pode ser iniciado de duas maneiras, *Started* e *Bound*. O estado *Started* indica que um Service foi iniciado por outro componente da aplicação, como, por exemplo, por uma Actvitiy, através do método *startService*. Quando iniciado, este tipo de Service trabalhará em segundo plano por tempo indeterminado, sem possuir vínculo algum com qualquer outro componente do sistema. Apesar de o Service possuir um consumo muito pequeno de memória e processamento, não deve ser criado um componente deste tipo com tempo de execução muito longo, pois pode comprometer a velocidade de execução de outras operações e, até mesmo, a bateria do aparelho.

A outra forma de iniciar um Service é o *Bound*, indicando que o Service está vinculado a outro componente. Este estado acontecerá quando outro componente da aplicação chamar o método *bindService*. Um Service no estado *Bound* oferece uma interface cliente servidor que permite a interação do componente, que chamou *bindService*, com o próprio Service. Um *Bound* Service estará em execução somente enquanto o componente que a criou estiver vinculado a ela, ou seja, quando o vínculo for quebrado, o Service terminará sua execução [14].

Um Service pode ser criado para trabalhar nos dois modos citados acima, ou seja, pode ser iniciado para rodar indefinitivamente e também pode ser vinculado. Independentemente de como o Service é iniciado, qualquer componente da aplicação poderá utilizá-lo, até mesmo um componente de outra aplicação. Para evitar o uso do Service por uma aplicação externa, este pode ser declarado como de uso privado, no arquivo manifesto (arquivo de configuração presente em todos os aplicativos Android).

Para criar um Service, deve-se criar uma subclasse da classe *android.app.Service* e sobrescrever alguns métodos de callback que lidam com aspectos chave do ciclo de vida. Se um componente inicia um Service chamando *startService*, que, consequentemente, invocará o método *onStartComand*, o Service será no tipo não vinculado, executando por tempo indeterminado, finalizando somente se parar a si mesmo, com *stopSelf*, ou se outro componente chamar *stopService*. Em contrapartida, se um componente chamar *bindService* para iniciar o Service, então este será vinculado ao componente, executando somente enquanto este vínculo estiver ativo [14].

O sistema Android poderá encerrar a execução de um Service do mesmo modo que faz com as Activities, utilizando os mesmos critérios já citados. Este encerramento forçado ocorrerá quando o sistema observar a necessidade de liberar recursos e de memória para execução de processamentos mais importantes.

2.4.2.1. Ciclo de Vida de um Service

Como existem dois tipos de Services, vinculado a componente e não vinculado a componente, existem também dois ciclos de vida, um para cada tipo. Conforme observado na Figura 7, os dois ciclos de vida são semelhantes, possuindo discrepâncias somente no período em que estão ativos (cor bege na figura abaixo).

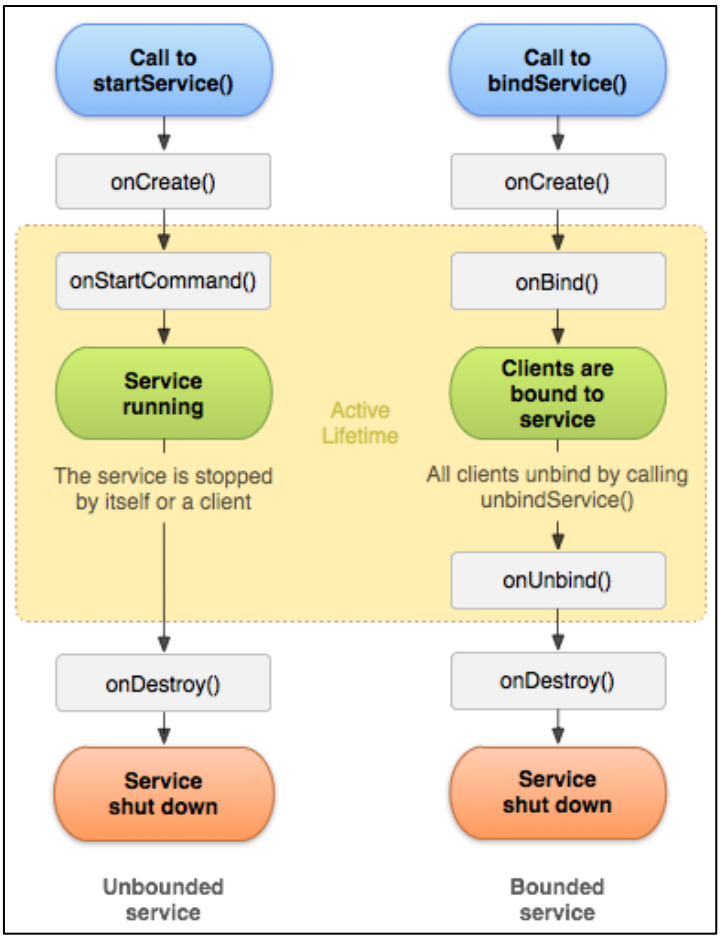

Figura 7 - Diferentes ciclos de vida de um Service

No lado esquerdo da Figura 7, pode-se observar o ciclo de vida de um Service iniciado com *startService*, isto é, de um Service *Started*, não vinculado a outro componente. Após executado este método de iniciação do Service, é chamado o *onCreate*, onde o Service é propriamente criado. Após isso, automaticamente é invocado o método *onStartComand*, onde ocorre o início da execução do Service. Ao final da execução, seja por terminar sua tarefa ou por requisição de um cliente, o método *onDestroy* é chamado para finalizar o componente.

Já o lado direito da Figura 7 representa o ciclo de vida de um Service vinculado a outro componente da aplicação, isto é, um Service iniciado por

meio da invocação do método *bindService*. Este tipo de Service entra no seu estado ativo depois de chamado o método *onBind*. Neste caso, ocorre uma conexão com outro componente, que pode ser considerado um cliente do Service. Para finalizar esta conexão, o cliente deve chamar o método *unbindService*, que precede o método incumbido de realmente finalizar o componente, chamado de *onDestroy* [14]. É importante ressaltar que o método *onStart* não é utilizado quando um Service é somente vinculado. Com isso, ao ser desvinculado, o sistema operacional Android tem a liberdade de finalizar este Service, que acontece com a chamada do *onDestroy*.

2.4.3. BroadcastReceiver

Como já explicitado anteriormente, as aplicações Android possuem quatro componentes principais: Activity, Service, BroadcastReceiver e ContentProvider. Uma aplicação não tem a necessidade de possuir os quatro componentes, mas terá, de forma obrigatória, pelo menos um deles.

Uma Activity está diretamente ligada a uma interface gráfica. Um Service tem a incumbência de rodar em segundo plano, não possuindo uma ligação direta obrigatória com uma interface. Um BroadcastReceiver, assim como um Service, não é interligado de forma impreterível a uma IU e roda em segundo plano, todavia, estes dois componentes diferem-se pois, enquanto Services são utilizados para operações longas, BroadcastReceivers devem ser usados para operações curtas. Ableson (2012, p.20) [5] diz que

> Se o BroadcastReceiver precisar de mais do que uma quantia trivial de execução de código, é recomendado que o código inicie uma requisição para um Service completar a função requisitada, porque o componente do aplicativo Service é projetado para operações de longo prazo, enquanto o BroadcastReceiver é feito para responder a diversos gatilhos.

Resumidamente, BroadcastReceiver deve ser utilizado quando se deseja que a aplicação responda um evento lançado pelo próprio sistema operacional, isto é, deve receber um Intent e fazer o processamento necessário em segundo plano. Exemplos de eventos globais, onde um BroadcastReceiver pode ser implementado para responder, são recebimento de mensagens de texto e de chamadas de voz.

BroadcastReceiver também pode ser utilizado para troca de mensagens entre aplicativos do Android, sem interferir na atividade foco do usuário, pois este componente “não utiliza interface gráfica e não se comunica diretamente com o usuário, pelo contrário, ele é executado em segundo plano sem que o usuário perceba” (Lecheta. 2010, p.290) [1].

A forma mais comum para se registar um *receiver* é no arquivo *AndroidManifest.xml*, de forma estática, por meio da utilização da tag *receiver*, juntamente com *intent-filter*. Desta forma, se um evento para o qual o *receiver* foi registrado acontecer, o método *onReceive* é chamado pela plataforma Android, sendo executado sem que o usuário tome conhecimento [15]. Um *receiver* também pode ser registrado em tempo de execução, através da execução do método *Context.registerReceiver*. Esta forma de registrar um BroadcastReceiver deve ser utilizada quando se deseja que ele esteja

disponível somente enquanto a Activity estiver executando, ou seja, quando determinada IU estiver em primeiro plano.

A Figura 8 apresenta o exemplo de registro de um BroadcastReceiver, que tem como IntentFilter a ação *android.intent.action.PHONE_STATE*. Isto quer dizer que este *receiver* será notificado somente quando o estado do telefone for mudado, como, por exemplo, quando o dispositivo receber uma chamada de voz.

```
<receiver android:name="MyPhoneReceiver" >
    <intent-filter>
        <action android:name="android.intent.action.PHONE_STATE" >
        </action>
    </intent-filter>
</receiver>
```

Figura 8 - Registro estático de um BroadcastReceiver

Assim como os componentes vistos anteriormente, para instanciar um BroadcastReceiver deve-se estender a classe mãe *android.content.BroadcastReceiver*. Após isso, basta sobrescrever o método *onReceive*, onde se pode implementar o comportamento desejado para quando este *receiver* for executado.

2.4.4. ContentProvider

Por fim, o último dos principais componentes de aplicações Android é o ContentProvider, provedor de conteúdo, um mecanismo para publicação, leitura e escrita de dados. Em outras palavras, ContentProvider gerencia o acesso a um conjunto estruturado de dados, encapsulando e oferecendo mecanismos para definição da segurança destes dados. Este componente é

um padrão de interface que conecta dados de um processo com o código em execução em outro processo [16].

Normalmente um aplicativo faz a manipulação de dados, sem a necessidade de que outra aplicação os consuma. Todavia, quando há a necessidade de uma aplicação deixar os dados que manipula disponíveis para outra aplicação, deve utilizar o ContentProvider. Da mesma forma, quando um aplicativo necessita acessar dados de outro, deve executar esta operação acessando o ContentProvider deste outro aplicativo, sendo que este acesso pode ocorrer tanto para leitura quanto para escrita de dados. O ContentProvider é utilizado tanto para troca de dados entre componentes de uma mesma aplicação quanto entre componentes de aplicações diferentes [5].

ContentProvider nada mais é do que um provedor de conteúdos, utilizado para compartilhamento de dados e informações entre aplicações. Com a utilização da classe *android.content.ContentProvider,* as informações geradas por um aplicativo, e armazenadas em banco de dados SQLite ou arquivos ou qualquer outra forma de armazenamento, ficarão públicas para serem acessadas por outras aplicações. Neste ponto, um exemplo clássico é a possibilidade de “ler os contatos cadastrados na agenda do celular usando uma API padronizada” (Lecheta. 2010, p.412) [1]. Existem provedores de conteúdos nativos no Android, como, por exemplo, consultar contatos da agenda e visualizar arquivos de imagens e vídeos presentes no dispositivo.

Quando se deseja acessar dados em um provedor de conteúdos, deve-se utilizar uma instância da classe *ContentResolver*, que fará o papel de um cliente. O objeto *resolver* comunica-se com o objeto *provider*, que é uma

instância da classe ContentProvider. O *provider* recebe a requisição de dados do cliente, executa a ação solicitada e retorna com os resultados. Em outras palavras, uma aplicação acessa dados de um ContentProvider com um objeto *ContentResolver*, sendo que esta classe possui métodos para as funções básicas de persistência de dados, conhecidas como as operações CRUD (*create, retrieve, update* e *delete*).

Um ContentProvider, concisamente, é responsável por gerenciar acesso a um repositório central de dados, independente da forma de armazenamento. Em conjunto, ContentProvider e *ContentResolver* oferecem consistência, padrão de interface para dados, comunicação entre processos diferentes e segurança no acesso a dados [16].

Para acessar dados em um ContentProvider, deve-se chamar *ContentResolver.query()*, utilizando-se um URI *content* como argumento na chamada do método *query*. Este URI identifica o dado em um *provider*, devendo ser instanciado no código Java como uma String final, pública e estática. Um URI *content* é composto pelo nome simbólico do provedor (*authority*) e pelo nome que aponta para uma tabela (*path*), que no caso, é um caminho ou diretório. Portanto, o URI *content* será formada pela estrutura que segue: “content://authority/path”.

Além do URI *content,* o método *query* ainda possui os argumentos *projection, selection, selectionArgs e sortOrder*. Em comparação com uma SQL query, URI representa a cláusula FROM, *projection* representam as colunas inseridas na cláusula FROM, *selection* é equivalente ao WHERE e *sortOrder* equivale ao ORDER BY. *SelectionArgs* não tem um equivalente exato em uma

consulta SQL pura. Portanto, acessar ContentProvider é similar a usar SQL [16].

Um aspecto importante no uso dos URIs é que existem constantes no sistema, evitando a necessidade de decorar cada String URI. A tabela 4 mostra alguns exemplos de URI nativas do sistema Android, juntamente com suas constantes.

URI	Constante
content://com.android.contacts/contacts	ContactsContract.Contacts.CONTENT_URI
content://com.android.contacts/contacts/1	---
content://media/internal/images/media	MediaStore.Images.Media.INTERNAL_CONTENT_URI
content://media/external/images/media	MediaStore.Images.Media.EXTERNAL_CONTENT_URI

Tabela 4 - URI nativas do Android e suas constantes

O primeiro URI listado acima retorna todos os contatos cadastrados na agenda do celular, o terceiro retorna todas as imagens disponíveis no celular e o quarto retorna todas as imagens disponíveis no cartão de memória externo. Já o segundo URI é responsável por retornar apenas o contato com identificador igual a um, sendo que esta operação não possui uma constante associada [1].

O método *ContentResolver.query()* sempre retorna um objeto da classe Cursor, que contem o conjunto de dados resultante de uma consulta à base de dados. A classe Cursor oferece acesso aleatório de escrita e leitura para o conjunto de resultados retornado por uma query. Através da utilização dos métodos da classe *android.database.Cursor*, pode-se interagir sobre as linhas do conjunto de resultados, determinar o tipo de dado de cada coluna, obter os

dados de uma coluna e verificar outras propriedades dos resultados. Se uma consulta não retornar nenhum dado, o *provider* retornará um objeto vazio da classe cursor, isto é, com *Cursor.getCount()* igual a zero [16].

ContentProviders, assim como Activities, Services e BroadcastReceivers, devem ser definidos no arquivo de configuração *AndroidManifest.xml*. Um ContentProvider é declarado neste arquivo por meio da tag *provider*, que deve possuir a classe que implementa o provedor e associar uma autoridade particular a esta classe.

Outra característica importante no uso de ContentProvider é a especificação de permissões de acesso, isto é, o *provider* de uma aplicação pode especificar permissões que outras aplicações deverão ter para acessar o conjunto de dados em questão. Esta funcionalidade é utilizada para garantir que o usuário saiba que dados uma aplicação tentará acessar. Estas permissões serão visíveis aos usuários finais quando estão instalando um aplicativo no seu dispositivo, pois deverão ter consentimento do usuário.

Por padrão, o Android proíbe que uma aplicação acesse dados da outra. Assim sendo, se não houverem permissões especificadas para um provider, uma aplicação não terá acesso aos dados de outra aplicação. Por outro lado, os componentes de uma aplicação sempre terão acesso liberado aos dados dos outros componentes da mesma aplicação, independentemente das permissões de acesso especificadas no AndroidManifest.xml [16]. As permissões deverão ser listadas no arquivo de configuração da aplicação, com a utilização da tag *uses-permission*.

2.5. O arquivo AndroidManifest.xml

AndroidManifest.xml é um arquivo de configuração que deve estar presente em todas as aplicações Android. Este arquivo deve possuir exatamente este nome e deve estar localizado no diretório *root* da aplicação, tendo a incumbência de apresentar informações essenciais acerca da aplicação à plataforma Android, explicitando ao sistema o que deve ser feito antes do código Java ser executado [17].

Como já mencionado algumas vezes nesse trabalho, um aplicativo Android deve possuir ao menos um dos quatro componentes principais, que são Activity, Service, BroadcastReceiver e ContentProvider. Estes componentes utilizam *IntentFilters* para listar *Intents* que estão aptos a executar. Esta ligação entre componente, *IntentFilter* e *Intent* deve estar presente justamente no arquivo manifesto, para que o sistema operacional Android saiba para qual aplicação ou componente direcionar a execução de um *Intent* [5].

O arquivo manifesto deve conter, dentre outras informações, o nome do pacote Java para o aplicativo, que representa o identificador único da aplicação, a descrição dos componentes da aplicação juntamente com seus *Intents*, conforme citados no parágrafo acima, qual processo irá hospedar cada um dos componentes da aplicação, quais permissões a aplicação deverá possuir para acessar partes protegidas da API e para interagir com outras aplicações, quais permissões outras aplicações deverão ter para interagir com os componentes desta aplicação e a versão mínima da API Android necessária para rodar esta aplicação [17].

Na Figura 9 segue um esqueleto do arquivo AndroidManifest.xml, contendo todos os elementos possíveis. A maioria dos elementos presentes não é obrigatória.

No tópico “Principais Elementos”, serão detalhados os principais elementos que compõem o manifesto da aplicação, presentes na Figura 9.

Um aspecto importante que deve ser configurado dentro do arquivo manifesto é o ponto de partida da aplicação, isto é, a primeira Activity que será exibida ao iniciar-se o aplicativo. Para isto, é necessário declarar um elemento *intent-filter*, dentro de um elemento Activity com <action android:name=”android.intent.action.MAIN”/> e <category android:name=”android.intent.category.LAUCHER”/>. Com isso, quando clicado no ícone para iniciar a aplicação, a Activity que possuir o elemento *intent-filter* com ação e categoria acima, será automaticamente exibida na tela.

```
<?xml version="1.0" encoding="utf-8"?>

<manifest>

    <uses-permission />
    <permission />
    <permission-tree />
    <permission-group />
    <instrumentation />
    <uses-sdk />
    <uses-configuration />
    <uses-feature />
    <supports-screens />
    <compatible-screens />
    <supports-gl-texture />

    <application>

        <activity>
            <intent-filter>
                <action />
                <category />
                <data />
            </intent-filter>
            <meta-data />
        </activity>

        <activity-alias>
            <intent-filter> . . . </intent-filter>
            <meta-data />
        </activity-alias>

        <service>
            <intent-filter> . . . </intent-filter>
            <meta-data/>
        </service>

        <receiver>
            <intent-filter> . . . </intent-filter>
            <meta-data />
        </receiver>

        <provider>
            <grant-uri-permission />
            <meta-data />
            <path-permission />
        </provider>

        <uses-library />

    </application>

</manifest>
```

Figura 9 - Estrutura do AndroidManifest.xml

2.5.1. Principais Elementos

O primeiro elemento do manifesto é justamente o *manifest*. Este é o elemento raiz do arquivo e deve possuir, obrigatoriamente, o elemento *application* e os atributos *xmlns:android* e *package*. O atributo *xmlns:android* define o Android namespace, que deve ser, por padrão, "http://schemas.android.com/apk/res/android". Já o atributo *package* representa o nome completo do aplicativo, isto é, o pacote principal do projeto, devendo ser único e podendo conter apenas letras, maiúsculas e minúsculas, números e underscores ("_"), devendo, ainda, iniciar com uma letra. Sintaxe e atributos deste elemento podem ser visualizados na Figura 10.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          package="string"
          android:sharedUserId="string"
          android:sharedUserLabel="string resource"
          android:versionCode="integer"
          android:versionName="string"
          android:installLocation=["auto" | "internalOnly" | "preferExternal"] >
    . . .
</manifest>
```

Figura 10 - Sintaxe e atributos do elemento *manifest*

O elemento *uses-permission* solicita uma permissão que a aplicação deve possuir para funcionar corretamente, como, por exemplo, utilizar a câmera do dispositivo. Estas permissões devem ser concedidas pelo usuário no momento da instalação do aplicativo. O único atributo deste elemento é o *android:name*, que representa o nome da permissão, definida pela aplicação através da utilização do elemento *permission.*

Conforme citado acima, o elemento *permission* é responsável por declarar as permissões de segurança que podem ser usadas para limitar o

acesso a componentes ou recursos específicos, da própria aplicação ou de outras. A Figura 11 exibe a sintaxe e os principais atributos deste elemento.

```
<permission android:description="string resource"
            android:icon="drawable resource"
            android:label="string resource"
            android:name="string"
            android:permissionGroup="string"
            android:protectionLevel=["normal" | "dangerous" |
                                     "signature" | "signatureOrSystem"] />
```

Figura 11 – Sintaxe e atributos do elemento p*ermission*

O elemento *application* é responsável pela declaração da aplicação, contendo sub elementos para cada um dos componentes da aplicação – Activity, Service, BroadcastReceiver e ContentProvider. Este elemento contém inúmeros atributos, como *android:allowTaskReparenting, android.debuggable, android:description, android:enabled, android:icon, android:label, android:permission, android: process, android.theme*, dentre outros.

O primeiro dos principais componentes do Android é declarado através do elemento ac*tivity*. Todas as Activities da aplicação deverão estar mapeadas no manifesto por meio da utilização deste elemento, caso contrário não serão vistas pelo sistema operacional e nunca serão executadas. Este elemento do AndroidManifest.xml deve estar inserido dentro do *application*, podendo conter dois sub elementos, *intent-filter* e *meta-data*. Abaixo segue sintaxe do elemento, juntamente com os principais atributos configuráveis.

```
<activity android:allowTaskReparenting=["true" | "false"]
          android:alwaysRetainTaskState=["true" | "false"]
          android:clearTaskOnLaunch=["true" | "false"]
          android:configChanges=["mcc", "mnc", "locale",
                                 "touchscreen", "keyboard", "keyboardHidden",
                                 "navigation", "screenLayout", "fontScale", "uiMode
                                 "orientation", "screenSize", "smallestScreenSize"]
          android:enabled=["true" | "false"]
          android:excludeFromRecents=["true" | "false"]
          android:exported=["true" | "false"]
          android:finishOnTaskLaunch=["true" | "false"]
          android:hardwareAccelerated=["true" | "false"]
          android:icon="drawable resource"
          android:label="string resource"
          android:launchMode=["multiple" | "singleTop" |
                              "singleTask" | "singleInstance"]
          android:multiprocess=["true" | "false"]
          android:name="string"
          android:noHistory=["true" | "false"]
          android:parentActivityName="string"
          android:permission="string"
          android:process="string"
          android:screenOrientation=["unspecified" | "user" | "behind" |
                                     "landscape" | "portrait" |
                                     "reverseLandscape" | "reversePortrait" |
                                     "sensorLandscape" | "sensorPortrait" |
                                     "sensor" | "fullSensor" | "nosensor"]
          android:stateNotNeeded=["true" | "false"]
          android:taskAffinity="string"
          android:theme="resource or theme"
          android:uiOptions=["none" | "splitActionBarWhenNarrow"]
          android:windowSoftInputMode=["stateUnspecified",
                                       "stateUnchanged", "stateHidden",
                                       "stateAlwaysHidden", "stateVisible",
                                       "stateAlwaysVisible", "adjustUnspecified",
                                       "adjustResize", "adjustPan"] >
    . . .
</activity>
```

Figura 12 – Sintaxe e atributos do elemento *activity*

Através do elemento *service,* devem ser declaradas todas as subclasses da classe *android.app.Service* presentes na aplicação. Assim como acontece com as Activities, todos os Services devem estar contidos no AndroidManifest.xml, pois os não declarados estarão invisíveis ao sistema, não sendo executados. Este elemento deve estar dentro do elemento *application* e pode conter elementos do tipo *intent-filter* e *meta-data*. Os atributos possíveis dentro do elemento *service* são *android:enabled, android:exported,*

android:icon, android:isolatedProcess, android:label, android:name, android:permission e *android:process*.

O terceiro dos principais componentes do Android que também deve estar presente no manifesto da aplicação é o BroadcastReceiver. Este é representado pelo elemento *receiver* e possui como atributos configuráveis *android:enable, android:exported, android:icon, android:label, android:name, android:permission* e *android:process.*

Por fim, o quarto e último componente principal do Android, ContentProvider, é declarado no manifesto através da utilização do elemento *provider*. Este, assim como os outros três componentes, deve ser inserido dentro do elemento *application*. Todos os providers da aplicação devem estar explicitados no manifesto, e devem ser declarados somente os ContentProviders que são partes da aplicação, ou seja, ContentProviders de outras aplicações que serão usados não necessitam ser declarados. Na Figura 13, estão listados sintaxe e principais atributos do *provider*.

```
<provider android:authorities="list"
          android:enabled=["true" | "false"]
          android:exported=["true" | "false"]
          android:grantUriPermissions=["true" | "false"]
          android:icon="drawable resource"
          android:initOrder="integer"
          android:label="string resource"
          android:multiprocess=["true" | "false"]
          android:name="string"
          android:permission="string"
          android:process="string"
          android:readPermission="string"
          android:syncable=["true" | "false"]
          android:writePermission="string" >
    . . .
</provider>
```

Figura 13 - Sintaxe e atributos do elemento *provider*

3. Planejamento do Aplicativo

Para cumprimento do objetivo principal deste trabalho, foi desenvolvido um aplicativo para a plataforma Android, chamado *Cálculos Médicos.*

A área da saúde é, indiscutivelmente, de importância imensurável a todos os cidadãos. Feito um estudo de mercado, observou-se que a plataforma Android carece de aplicativos nesta área. Atividades diárias, de grande importância, são executadas inúmeras vezes por profissionais da área, como cálculos rápidos, classificações e escalas. O aplicativo *Cálculos Médicos* foi desenvolvido, justamente, para dar suporte a estes profissionais, visando uma maior segurança e facilidade na execução destas atividades.

Este capítulo apresentará o processo de desenvolvimento da aplicação, desde o levantamento e análise de requisitos à codificação. Também serão explicitados configuração do ambiente de desenvolvimento, tecnologias utilizadas, pesquisa de campo e especificação por meio de diagramas, utilizando a linguagem de modelagem UML.

3.1. Metodologia de Desenvolvimento

3.1.1. Configuração do Ambiente

Para possibilitar o desenvolvimento de aplicações para a plataforma Android, há algumas ferramentas e tecnologias que devem ser utilizadas. Dentre estas ferramentas, deve-se escolher, primeiramente, o IDE - Integrated Development

Environment – de desenvolvimento, ou seja, qual ferramenta o programador utilizará como base para escrever as linhas de código. O IDE escolhida para desenvolvimento do aplicativo *Cálculos Médicos* foi o Eclipse, versão *June*. Neste ponto, vale ressaltar que o Eclipse é baixado na forma de um arquivo compactado, não se fazendo necessária uma instalação, isto é, depois de terminado o download, é necessário somente descompactar o arquivo, e ele estará pronto para uso.

Outra ferramenta utilizada neste trabalho foi o Android SDK (*Software Development Kit*), que é o ambiente de desenvolvimento destinado a quem deseja programar aplicações para o sistema operacional Android. Esta ferramenta pode ser baixada, de forma gratuita, no site oficial do Android. O Android SDK disponibiliza funcionalidades primordiais ao desenvolvedor, como emulador, *debugger,* bibliotecas, documentação e tutoriais.

Por fim, para integração do IDE Eclipse com o Android SDK, foi utilizado o plug-in Android para o Eclipse, chamado de *Android Development Tools* – ADT. Segundo documentação oficial do Android, ADT é um plug-in utilizado para fornecer um ambiente poderoso e integrado para desenvolvimento de aplicativos Android. ADT aperfeiçoa as funcionalidades do Eclipse para tornar mais rápida a configuração do ambiente de desenvolvimento de aplicações Android, além de facilitar a criação de novos projetos e de interfaces gráficas, dentre outros [19].

No Eclipse, deve-se discriminar o SDK que está sendo utilizado. Isto deve ser feito na opção *Preferences*, item presente na aba *Window.* Com a janela de preferencias aberta, deve-se clicar na opção *Android,* posicionado na

árvore à esquerda da tela e selecionar o local de instalação do SDK, conforme Imagem 14.

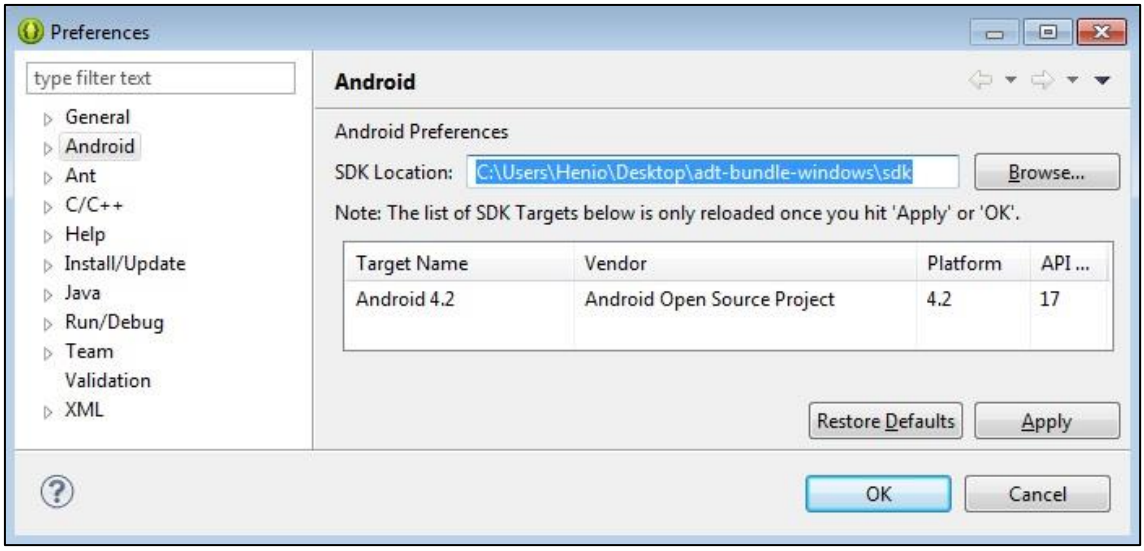

Figura 14 - Configurando a localização do Android SDK no Eclipse

Para o perfeito funcionamento do ambiente de desenvolvimento Android, devem ser feitas, além das configurações básicas presentes em qualquer ambiente de desenvolvimento Java, as configurações citadas acima. Vale ressaltar que há arquivos compactados prontos para o desenvolvimento Android disponíveis na internet. Estes arquivos possuem tanto o IDE Eclipse, como o SDK Android e o plug-in ADT. Portanto, feito o download deste arquivo, basta descompactá-lo e executar o Eclipse para iniciar o desenvolvimento.

3.1.2. Levantamento de Requisitos

Uma etapa imprescindível no processo de desenvolvimento de software é o levantamento e a análise de requisitos. Segundo Pressman (2011, p.151), “a análise de requisitos resulta na especificação de características operacionais

do software, indica a interface do software com outros elementos do sistema e estabelece restrições que o software deve atender" [18].

Evidenciando a importância desta etapa no processo de desenvolvimento de software, pode-se afirmar que, após sua execução, analistas e programadores terão explicitadas as funcionalidades que o sistema deverá disponibilizar ao usuário, isto é, os profissionais terão o entendimento do que se espera do software. O documento de levantamento e análise de requisitos deve ser validado e autorizado pelo cliente, pois qualquer alteração refletirá no valor final do software e em um possível não cumprimento de prazos. Portanto, se o cliente desejar alterar os requisitos do software, custos e prazos devem ser recalculados, de acordo com estas mudanças.

Com o intuito de fazer um aplicativo que contivesse funcionalidades condizentes com as atividades diárias de um profissional da área da saúde, fez-se necessária a execução de uma pesquisa de campo para levantamento de requisitos.

Para tal feito, foram entrevistados profissionais das áreas de Nutrição, Fisioterapia, Fonoaudiologia, Educação Física, Medicina e Odontologia. Nas entrevistas foi feita uma apresentação da proposta do trabalho, explicitando detalhes sobre o funcionamento geral do aplicativo a ser desenvolvido. Após isto, os profissionais foram questionados se, dentro da sua profissão, havia um ou mais procedimentos que poderiam ser automatizados de acordo com o modelo de funcionamento que se esperava para o aplicativo.

Dentre os contatados, retornaram positivamente os profissionais das áreas de Fisioterapia, Educação Física e Medicina, resultando nos cálculos discriminados na sucessão.

- Anion Gap:
    - **Descrição**:
        - Também conhecido como intervalo aniônico, representa a diferença entre cátions (Sódio) e ânions (Bicarbonato e Cloro) presentes no sangue.
    - **Fórmula para cálculo**:
        - Anion Gap = Sódio – (Cloro + Bicarbonato).
    - **Unidades de medida utilizadas:**
        - Anion Gap = mmol/L;
        - Nível de Sódio = mmol/L;
        - Nível de Cloro = mmol/L;
        - Nível de Bicarbonato = mmol/L.
    - **Resultado:**
        - Anion Gap calculado em mmol/L.
- Classificação de Forrest:
    - **Descrição**:
        - Classificação endoscópica de úlcera. Classifica os pacientes em sangramento ativo (em jato ou escorrendo), em sangramento recente (coto vascular visível, coágulo recente ou fundo hematínico) ou em sangramento inexistente.
    - **Fórmula para cálculo**:

        - Por ser uma classificação, não possui fórmula para cálculo.
    - **Resultado:**
        - De acordo com o tipo de lesão, exibe o tipo de sangramento, o nível de frequência dos estigmas e o nível de risco de ressangramento.
- Classificação de Killip:
    - **Descrição**:
        - Utilizado em indivíduos com infarto agudo de miocárdio para verificar as chances de falecimento nos trinta dias subsequentes ao ataque cardíaco.
    - **Fórmula para cálculo**:
        - Por ser uma classificação, não possui fórmula para cálculo.
    - **Resultado:**
        - De acordo com a classe, exibe o nível de insuficiência cardíaca e a taxa de mortalidade.
- Critério Ranson:
    - **Descrição**:
        - Utiliza onze parâmetros para estimar o prognóstico de pacientes com pancreatite aguda, sendo que cinco parâmetros devem ser analisados na admissão do paciente, e os restantes após 48 horas da admissão.
    - **Fórmula para cálculo**:
        - Um parâmetro presente representa um ponto no critério Ranson. Após a análise dos onze parâmetros, o paciente é classificado de acordo com o total de parâmetros

presentes, informando as chances de mortalidade e de uma possível ocorrência de pancreatite grave.

- **Resultado:**
  - Exibe a porcentagem de mortalidade e a chance de pancreatite grave.

- Escala de Apgar:
  - **Descrição**:
    - Avalia o nível de asfixia de um recém-nascido através da observação de cinco sinais objetivos do bebê no primeiro, no quinto e no décimo minuto após o nascimento. Os cinco sinais objetivos são frequência cardíaca, respiração, tônus muscular, cor da pele e irritabilidade reflexiva.
  - **Fórmula para cálculo**:
    - Cada um dos cinco sinais objetivos tem três opções, sendo que as opções representam ou nenhum, ou um ou dois pontos na escala de Apgar. Ao final da observação dos cinco sinais, somam-se os pontos, obtendo-se o resultado final da escala.
  - **Resultado:**
    - Classifica-se o recém-nascido em sem asfixia, com asfixia leve, com asfixia moderada ou com asfixia grave.
- Escala de Ashworth Modificada:
  - **Descrição**:
    - Graduação do tônus muscular de um paciente entre seis graus possíveis (0, 1, 1+, 2, 3 e 4).

- **Fórmula para cálculo**:
  - Por ser uma classificação, não possui fórmula para cálculo.
- **Resultado:**
  - Exibe a descrição da graduação na qual o paciente foi classificado.

- Escala de Child Pugh:
  - **Descrição**:
    - Utilizada para avaliar o prognóstico de doença hepática crônica. São analisados cinco critérios: bilirrubina total, albumina, INR, ascite e encefalopatia.
  - **Fórmula para cálculo**:
    - Cada um dos critérios é pontuado em um, dois ou três. Ao final da escolha de todos os critérios, somam-se os pontos, obtendo-se a escala Child Pugh.
  - **Resultado:**
    - Exibe a classe do paciente, a porcentagem de sobrevida em um e em dois anos.
- Escala Glasgow de Coma:
  - **Descrição**:
    - Escala neurológica para verificar o nível de consciência de uma pessoa após traumatismo craniano. Utiliza resultados de teste de abertura ocular, de resposta verbal e de resposta motora.
  - **Fórmula para cálculo**:

        - Teste de abertura ocular é pontuado de um a quatro (ausente; em resposta à dor; em comando verbal; espontaneamente). Teste de resposta verbal é pontuado de um a cinco (emudecido; sons incompreensíveis; palavras desconexas; confuso, desorientado; conversa normalmente). Teste de reposta motora é pontuado de um a seis (não se movimenta; extensão hipertônica; flexão hipertônica; flexão inespecífica; localiza estímulos dolorosos; obedece comandos). Ao final da escolha das opções, somam-se os pontos, obtendo-se a escala de Glasgow.
    - **Resultado:**
        - Exibe o tipo de coma, o tipo de trauma e se o paciente necessita de intubação imediata.

- Escala MELD:
    - **Descrição**:
        - É um sistema de pontuação utilizado para avaliar a gravidade da doença hepática crônica. É uma escala numérica que exibe o risco de morte de um paciente enquanto espera por um transplante de fígado.
    - **Fórmula para cálculo**:
        - MELD = 3,78 * Ln(Bilirrubina) + 11,2 * Ln(INR) + 9,57 * Ln(Creatinina) + 6,43.
    - **Unidades de medida utilizadas:**
        - Bilirrubina = mg/dL;

        - Creatinina = mg/dL.
    - **Resultado:**
        - Exibe o valor da escala MELD e a porcentagem do risco de morte nos próximos três meses de vida do paciente.
- Escala Ramsay:
    - **Descrição**:
        - Utilizada para, entre seis graus existentes, classificar o nível sedação de pacientes.
    - **Fórmula para cálculo**:
        - Por ser uma classificação, não possui fórmula para cálculo.
    - **Resultado:**
        - Exibe a descrição do grau de sedação selecionado.
- Fórmula de Cockroft-Gault:
    - **Descrição:**
        - Utilizado para medir a estimativa de filtração glomerular, isto é, a depuração plasmática de creatinina estimada.
    - **Fórmula para cálculo:**
        - Estimativa = ((140 - idade) * peso) / (Creatinina * 72).
        - Se o paciente for do sexo feminino, deve-se utilizar um fator de correção, multiplicando o valor total da estimativa por 0,85.
    - **Unidades de medida utilizadas:**
        - Estimativa = mL/min;
        - Idade = anos;
        - Peso = quilos;

        - Creatinina = mg/dL.
    - **Resultado:**
        - Estimativa em mL/min.

- Fórmula de Parkland:
    - **Descrição:**
        - Utilizado para calcular a hidratação necessária em pacientes que sofreram queimaduras.
    - **Fórmula para cálculo:**
        - Soro requerido = 0,004 * peso * % área corporal queimada.
    - **Unidades de medida utilizadas:**
        - Soro requerido = litros;
        - Peso = quilos.
    - **Resultado:**
        - Exibe a quantidade total de litros de soro requeridos nas próximas vinte e quatro horas, discriminando a quantidade de soro que deve ser utilizada nas primeiras oito horas e a quantidade de soro que deve ser utilizada nas últimas dezesseis horas.
- Índice de Massa Corporal (IMC):
    - **Descrição:**
        - Utilizado para medir o grau de obesidade de uma pessoa, sabendo se ela está em seu peso ideal ou não.
    - **Fórmula para cálculo:**
        - IMC = peso / altura²
    - **Unidades de medida utilizadas:**

        - Peso = quilos;
        - Altura = metros.
    - **Resultado:**
        - Exibe o valor do IMC e a classificação do paciente, informando se ele está abaixo do peso ideal, no peso ideal, acima do peso, obeso, obeso severo ou obeso mórbido.
- LDL – Cálculo do Colesterol:
    - **Descrição:**
        - Utilizado para medir o valor do LDL (*Low Density Lipoprotein*), conhecido como colesterol “ruim” ou “mau”.
    - **Fórmula para cálculo:**
        - LDL = Colesterol Total – HDL – (Triglicerídeos / 5)
    - **Unidades de medida utilizadas:**
        - LDL = mg/dL;
        - Colesterol Total = mg/dL;
        - HDL = mg/dL;
        - Triglicerídeos = mg/dL.
    - **Resultado:**
        - Exibe o valor do LDL em mg/dL e classifica o paciente de acordo com seu nível de colesterol.
- Margem de Gordura Corporal:
    - **Descrição:**
        - Utiliza sexo e idade da pessoa para classifica-la de acordo com seu percentual de gordura corporal.
    - **Fórmula para cálculo:**

        - Por ser uma classificação, não possui fórmula para cálculo.
    - **Unidades de medida utilizadas:**
        - Idade: anos.
    - **Resultado:**
        - Classifica a pessoa em com pouca gordura corporal, saudável, com alto teor de gordura ou obesa.
- Massa Óssea Estimada:
    - **Descrição:**
        - Utiliza o sexo e o peso da pessoa para exibir sua massa óssea estimada.
    - **Fórmula para cálculo:**
        - Por ser uma classificação, não possui fórmula para cálculo.
    - **Unidades de medida utilizadas:**
        - Peso: quilos.
    - **Resultado:**
        - Massa de massa óssea estimada em quilos.
- Risco de Pneumonia CURB-65:
    - **Descrição:**
        - Utiliza cinco critérios (confusão, nível de ureia, respiração, pressão sanguínea e idade) para determinar o risco de morte, nos próximos trinta dias, de uma pessoa com pneumonia.
    - **Fórmula para cálculo:**
        - Cada critério selecionado como presente soma um ponto no total. De acordo com o total, classifica o paciente.

- **Resultado:**
    - Exibe o risco de morte do paciente nos próximos trinta dias.

- Teste de Cooper (Teste dos 12 Minutos):
    - **Descrição:**
        - Calcula o volume de oxigênio máximo de um indivíduo, que representa o volume máximo de oxigênio que o corpo consegue transportar e utilizar durante a execução de um exercício físico.
    - **Fórmula para cálculo:**
        - VO2 máximo = (distância percorrida – 504,9) / 44,73
    - **Unidades de medida utilizadas:**
        - VO2 máximo = L/min
        - Idade: anos;
        - Distância percorrida: metros.
    - **Resultado:**
        - Exibe o VO2 máximo calculado em L/min e o resultado do teste de Cooper do paciente.
- Zona Aeróbica de Treinamento:
    - **Descrição:**
        - Lista as zonas alto de treinamento aeróbico de um indivíduo, a partir de sua idade.
    - **Fórmula para cálculo:**
        - FcMax (Frequência Cardíaca Máxima) = 220 – idade;
        - Zona de Esforço Máximo = entre 90% e 100% da FcMax;

- Zona de Limiar Aeróbico = entre 80% e 90% da FcMax;
- Zona Aeróbica = entre 70% e 80% da FcMax;
- Zona de Controle de Peso = entre 60% e 70% da FcMax;
- Zona de Atividade Moderada = entre 50% e 60% da FcMax.

- **Unidades de medida utilizadas:**
  - FcMax e Zonas = bpm (batimentos por minuto);
  - Idade = anos.
- **Resultado:**
  - Exibe a frequência cardíaca máxima e todas as zonas de treinamento de uma pessoa.

3.2. Especificação UML

Após a etapa de levantamento e análise de requisitos, outra que se destaca no ciclo de vida de um software é o projeto, isto é, especificação e modelagem. Análise e projeto de software juntos equivalem, mais especificamente, ao planejamento do software. As demais etapas que completam o ciclo de vida do software são implementação, testes e manutenção.

A modelagem do aplicativo *Cálculos Médicos* foi feita utilizando UML, uma linguagem de modelagem desenvolvida com o objetivo de unificar diversas linguagens de especificação já existentes. “A linguagem UML é composta por treze diagramas, classificados em duas categorias: os diagramas estruturais e os diagramas de comportamento” (SILVA, 2007, p.84). Para a

modelagem de um software ser considerada completa, ela não deve ter, obrigatoriamente, todos os treze diagramas, mas alguns são essenciais, como diagrama de classes, diagrama de casos de uso e diagrama de atividades.

3.2.1. Diagrama de Casos de Uso

O diagrama de casos de uso, composto por casos de uso, atores e relacionamentos, modela o conjunto de funcionalidades de um software em alto nível de abstração. O principal objetivo deste tipo de diagrama é listar, de forma superficial, as funcionalidades do sistema, isto é, exibir o que o sistema oferece ao usuário. Analisando o diagrama de casos de uso exibido na Figura 15, que representa o aplicativo *Cálculos Médicos*, é possível saber o que o que o sistema faz, mas sem o detalhamento de sua estrutura interna. Este tipo de visão é chamado de *visão caixa-preta,* justamente pela característica de não mostrar o que há no interior do sistema [20].

O aplicativo desenvolvido possui duas funcionalidades básicas, visualizar informações sobre o sistema e visualizar os cálculos disponíveis. De acordo com o diagrama modelado, o caso de uso *Visualizar Cálculos* inclui outro caso de uso, *Escolher Categoria*. A inclusão (<<*include*>>) é uma associação muito importante no diagrama de casos de uso. De acordo com Silva (2007, p.109) [20], a “associação de inclusão estabelece que parte do comportamento inerente a um caso de uso está definida em outro caso de uso, isto é, que um caso de uso contém o comportamento definido em outro caso de uso”.

Por fim, após escolher a categoria, o usuário terá em suas mãos os cálculos disponíveis no sistema, podendo calcular o que for necessário. Como cada cálculo corresponde a uma funcionalidade inerente ao sistema, cada um deles deve ser representado por um caso de uso, conforme o diagrama que segue.

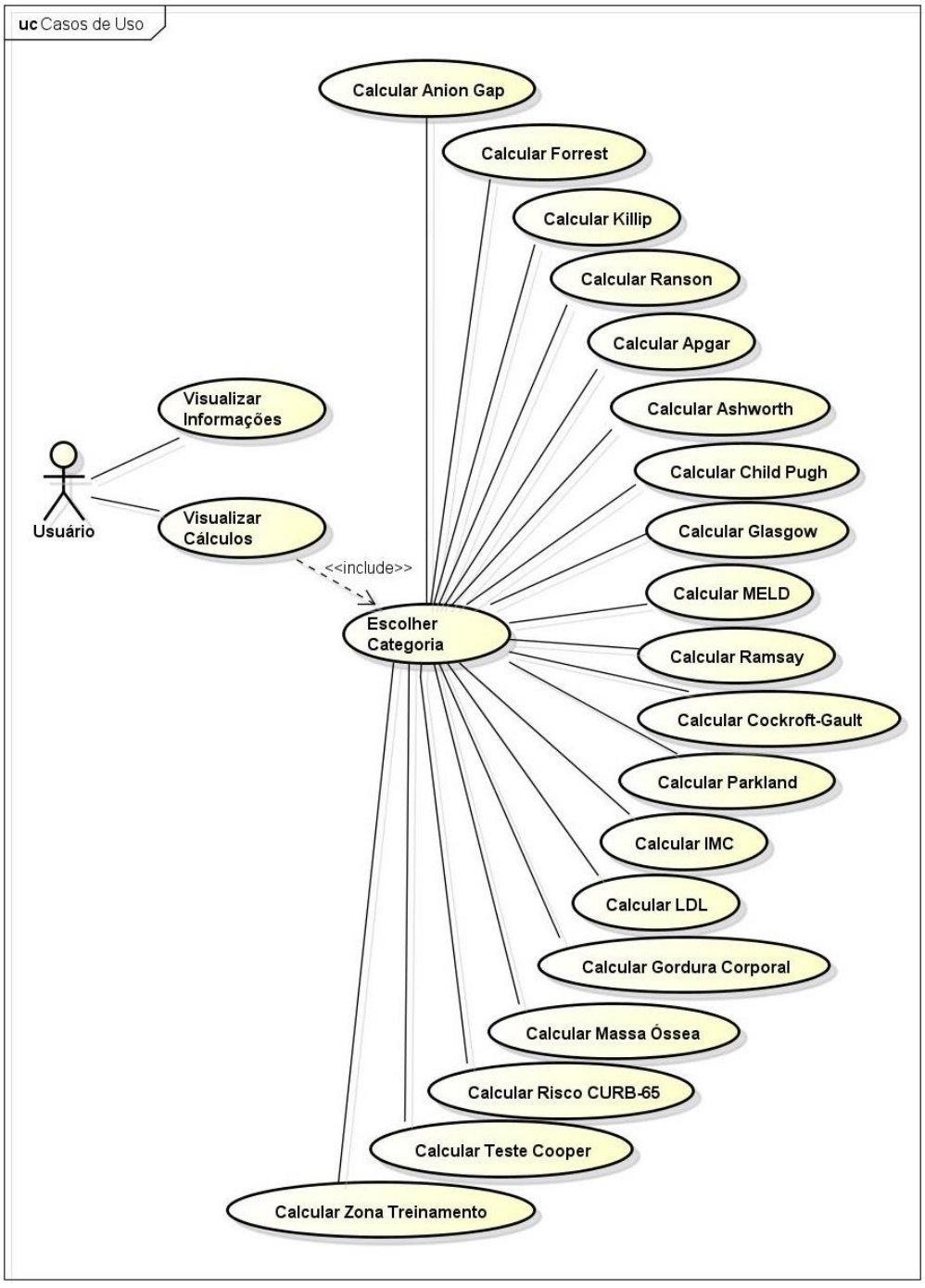

Figura 15 - Diagrama de Casos de Uso do Aplicativo

3.2.2. Diagrama de Classes

Segundo Silva (2007, p.43) [20], a modelagem com diagrama de classes

> Corresponde ao resultado do esforço mais primitivo de geração de modelos gráficos que descrevem um programa orientado a objetos, isto é, as classes com sua estrutura e os relacionamentos entre classes. É a alternativa mais rudimentar para descrever os elementos de um programa orientado a objetos de forma mais legível que aquela presenta na implementação.

Um diagrama de classes é composto por classes, sendo que a representação de cada classe possui três objetos principais. O primeiro deles é o identificador, o nome da classe. O segundo é o conjunto de atributos, onde são exibidos todos os atributos da classe, juntamente com suas visibilidades (*public, private* ou *protected*) e seus tipos, que podem ser primitivos (*String, Integer, boolean*, dentre outros) ou não. O terceiro e último objeto de uma classe é o conjunto de métodos, que exibe o identificador do método, sua visibilidade, seu tipo de retorno e seus parâmetros e tipos.

Outro aspecto importante no diagrama de classes é o relacionamento entre as classes, que pode ser do tipo herança (presente na modelagem do aplicativo *Cálculos Médicos*), agregação, composição, associação, realização e dependência.

Todos os cálculos disponíveis no aplicativo desenvolvido são Activities, filhas de uma classe genérica chamada *ActivityGeral*, sendo que a esta também é filha de uma outra, a *Activity* do Android. Neste ponto, vale ressaltar que toda tela de um aplicativo Android representa uma Activity, por este motivo, todo cálculo é uma classe que estende, de forma indireta, a classe

android.app.Activity. Portanto, estas classes relacionam-se entre si através da herança, “uma relação de especialização entre duas classes, em que uma delas corresponde a um conceito mais genérico e a outra, a um conceito mais específico” (Silva. 2007, p.49) [20].

Quando detectado um relacionamento de herança, a classe mais genérica, que foi estendida, é chamada de superclasse, enquanto a classe que a estende é chamada de subclasse. No presente caso, discriminado por meio dos diagramas de classe abaixo, a classe *ActivityGeral* representa a superclasse, sendo estendida por todas as outras, que são as subclasses.

Por questão de didática, foram modelados cinco diagramas de classe, visto que há muitas classes no sistema e ficaria ilegível representa-las todas no mesmo diagrama. Outro detalhe importante é que a superclasse *ActivityGeral* repete-se em todos os diagramas. Isto é feito propositalmente, para explicitar todos os relacionamentos de herança presentes entre as classes do software.

A Figura 16 representa o primeiro diagrama de classes. Neste, estão presentes a superclasse *ActivityGeral* e as subclasses *CalculaAnionGap, CalculaApgar* e *CalculaAshworth*. No diagrama, observa-se claramente que estas três classes estendem aquela. De forma simplória, isto representa que, por exemplo, a classe *CalculaAnionGap*, além dos atributos *campoNivelSodio, campoNivelCloro* e *campoNivelBicarbonato*, também possui os atributos *popupDescricao, labelResultado* e *valorFinal.* O mesmo raciocínio emprega-se para os métodos, ou seja, além dos métodos *onCreate, validaDados, calcularAnionGap* e *limpaTela*, esta classe possui os métodos *onResume,*

esconderTecladoVirtual, exibeResultados, escondeResultados e *mostraPopupDescricao*.

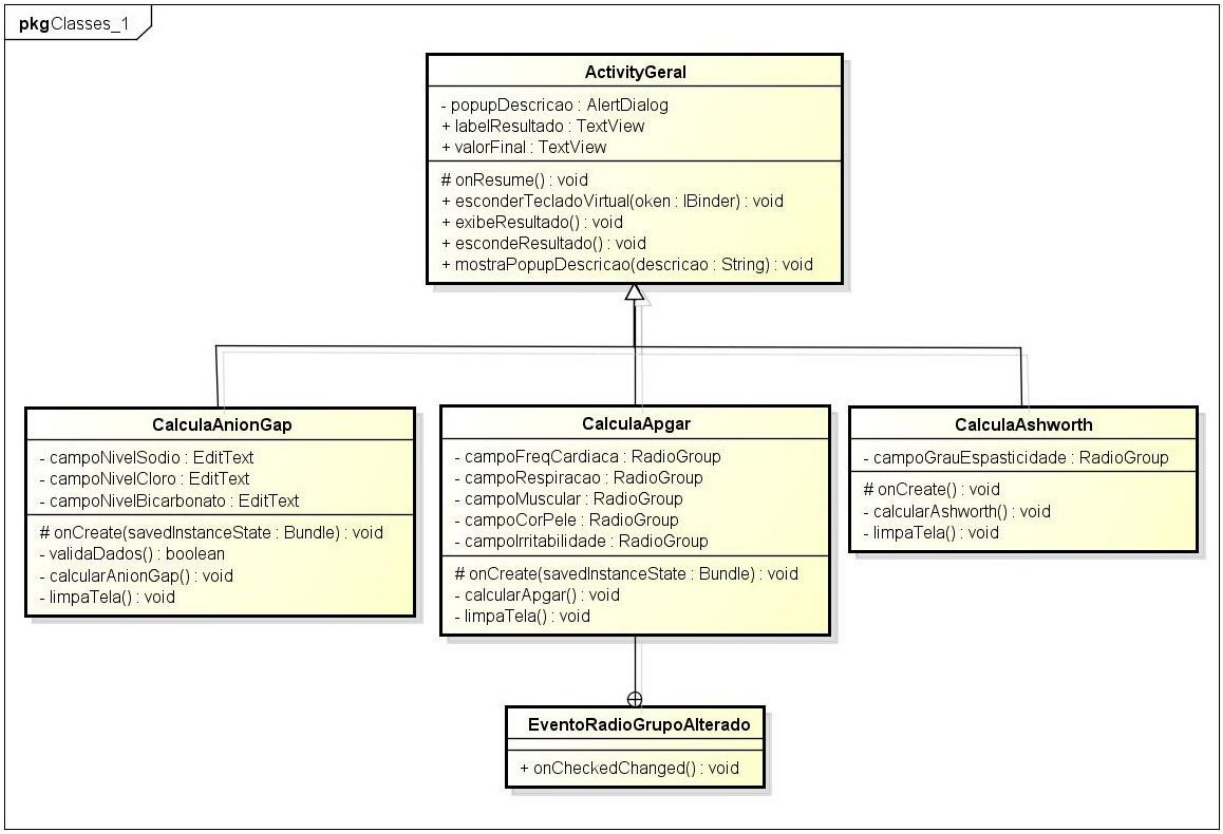

Figura 16 - Diagrama de Classes 1

No relacionamento de herança presente, pode-se dizer que todo objeto da classe *CalculaAnionGap* (ou *CalculaApgar*, ou *CalculaAshworth*) também é objeto da *ActivityGeral*, mas a recíproca não é verdadeira, ou seja, um objeto da classe mais genérica, *ActivityGeral*, não necessariamente será um objeto da classe mais específica.

Ainda no diagrama de classes da Figura 16, há a classe *EventoRadioGrupoAlterado*, ligada à classe *CalculaApgar*. Pode-se observar que estas classes ligam-se por uma reta, onde, em uma das pontas, há um

sinal de adição (+) dentro de um círculo. Esta simbologia representa que a classe *EventoRadioGrupoAlterado* é uma classe interna à classe *CalculaApgar*.

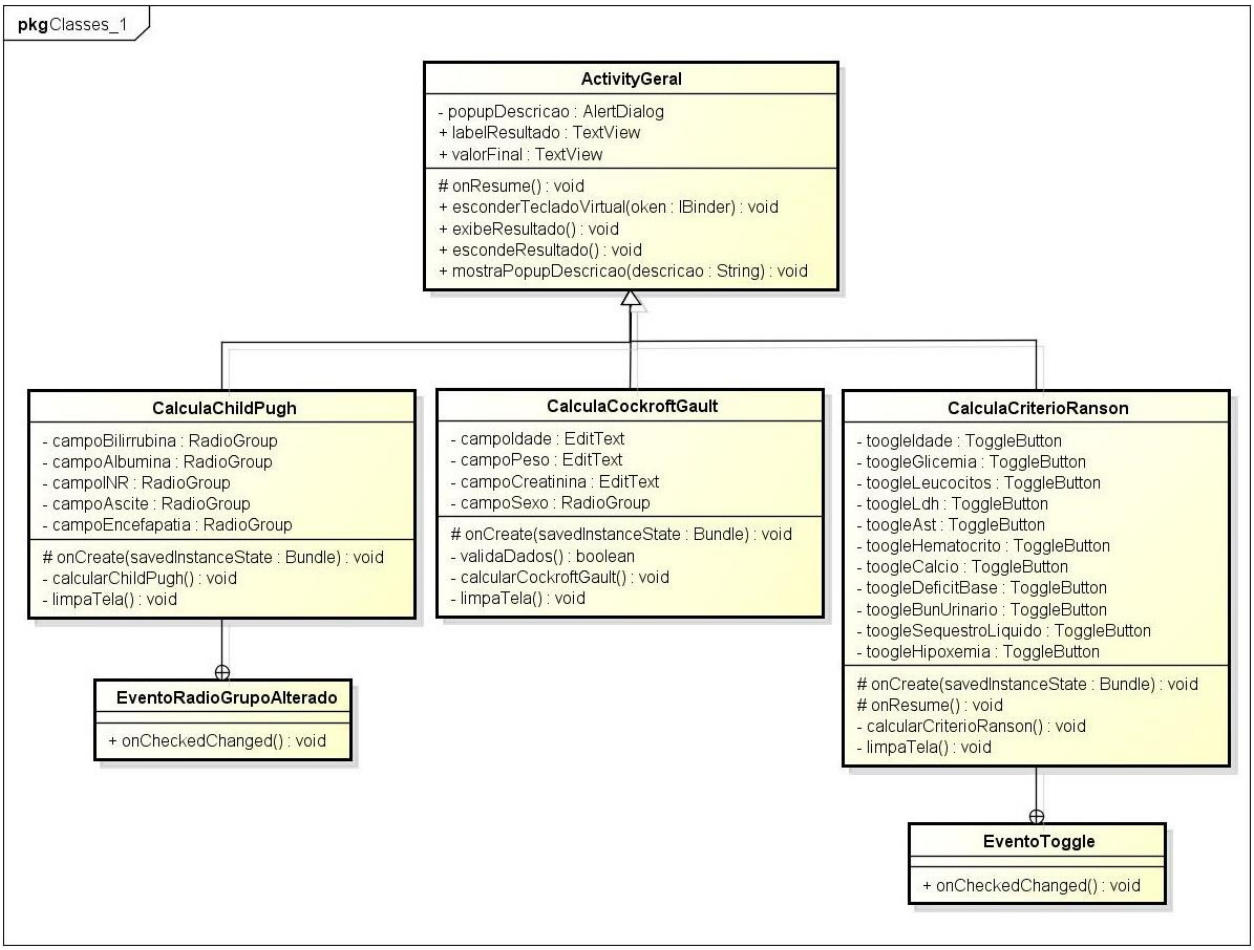

Figura 17 - Diagrama de Classes 2

Como dito, o diagrama da Figura 17, assim como os que virão na sequência, segue o mesmo raciocínio explicitado anteriormente. Neste, há duas classes internas, a *EventoRadioGrupoAlterado,* interna à *CalculaChildPugh*, e a *EventoToggle*, interna à *CalculaCritérioRanson*. Observa-se que a classe *EventoRadioGrupoAlterado* possui o mesmo nome da classe interna presente na Figura 16, todavia, por serem internas, representam classes diferentes, executando ações diferentes.

Um aspecto importante para se observar nos diagramas de classe é a visibilidade de atributos e métodos, representada pelos sinais positivo (+), negativo (-) e sustenido (#). O sinal positivo, presente antes do atributo ou método, indica que este é do tipo *public*, isto é, pode ser acessado por qualquer classe do sistema. Já o sinal negativo representa um atributo ou método *private*, que pode ser acessado somente dentro da própria classe. Por fim, o sustenido representa a visibilidade *protected*, indicando que o atributo ou método poderá ser acessado somente pela classe presente ou por suas subclasses.

Abaixo seguem os demais diagramas de classe modelados, onde estão representados todos os cálculos do aplicativo *Cálculos Médicos.*

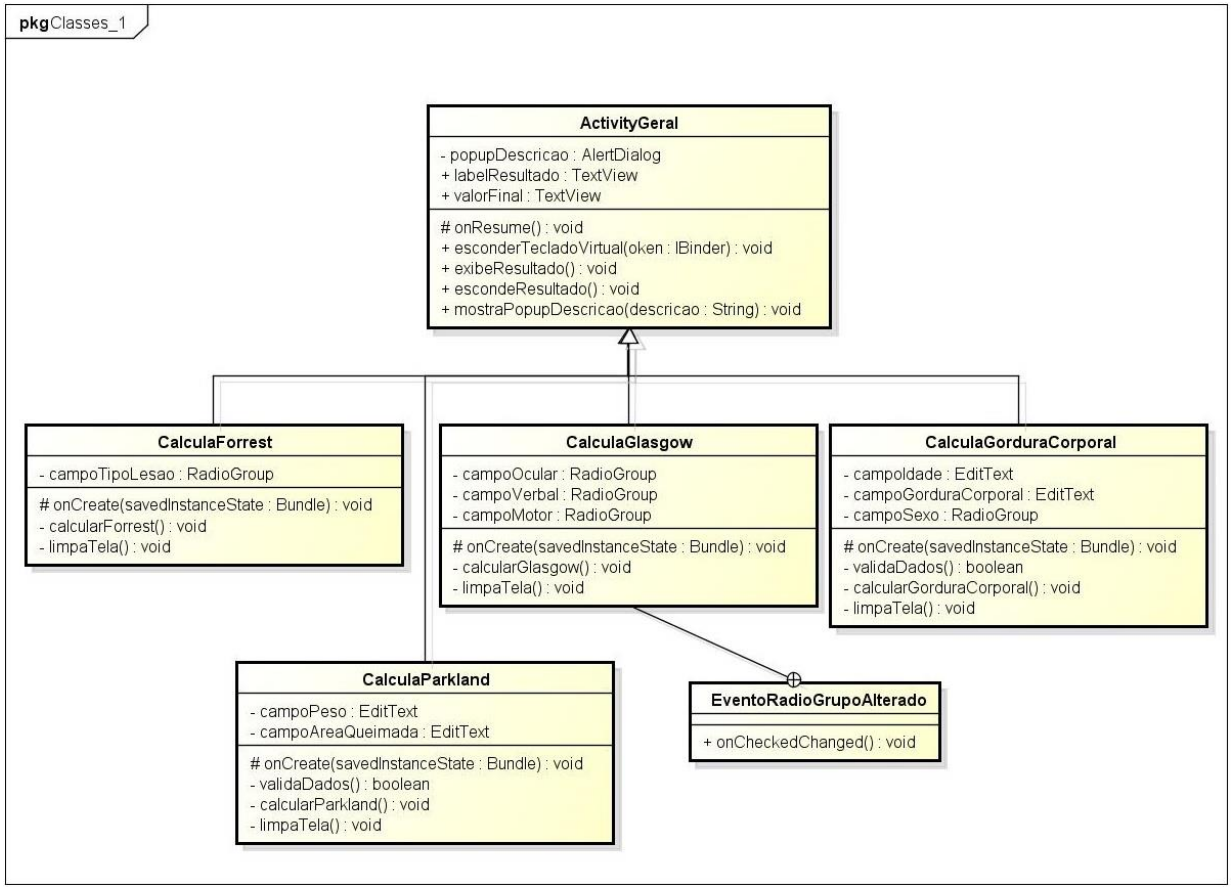

Figura 18 - Diagrama de Classes 3

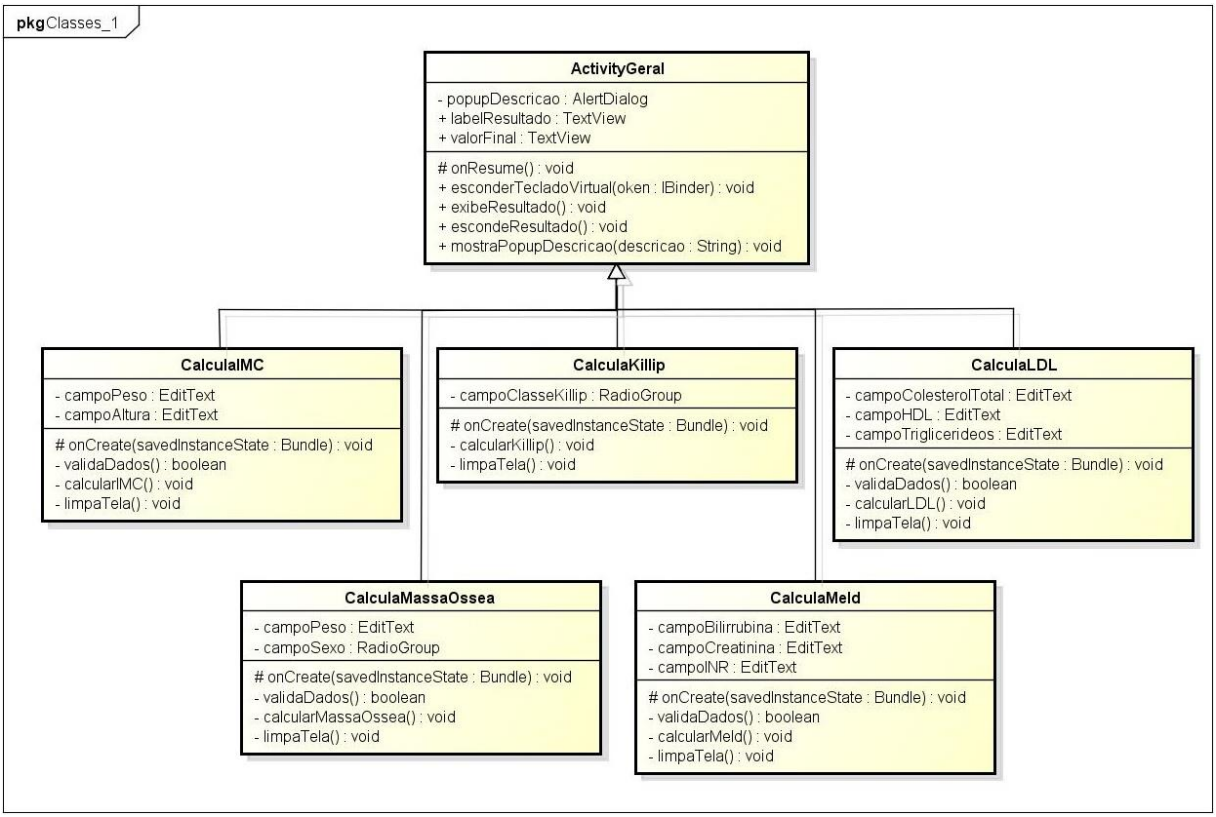

Figura 19 - Diagrama de Classes 4

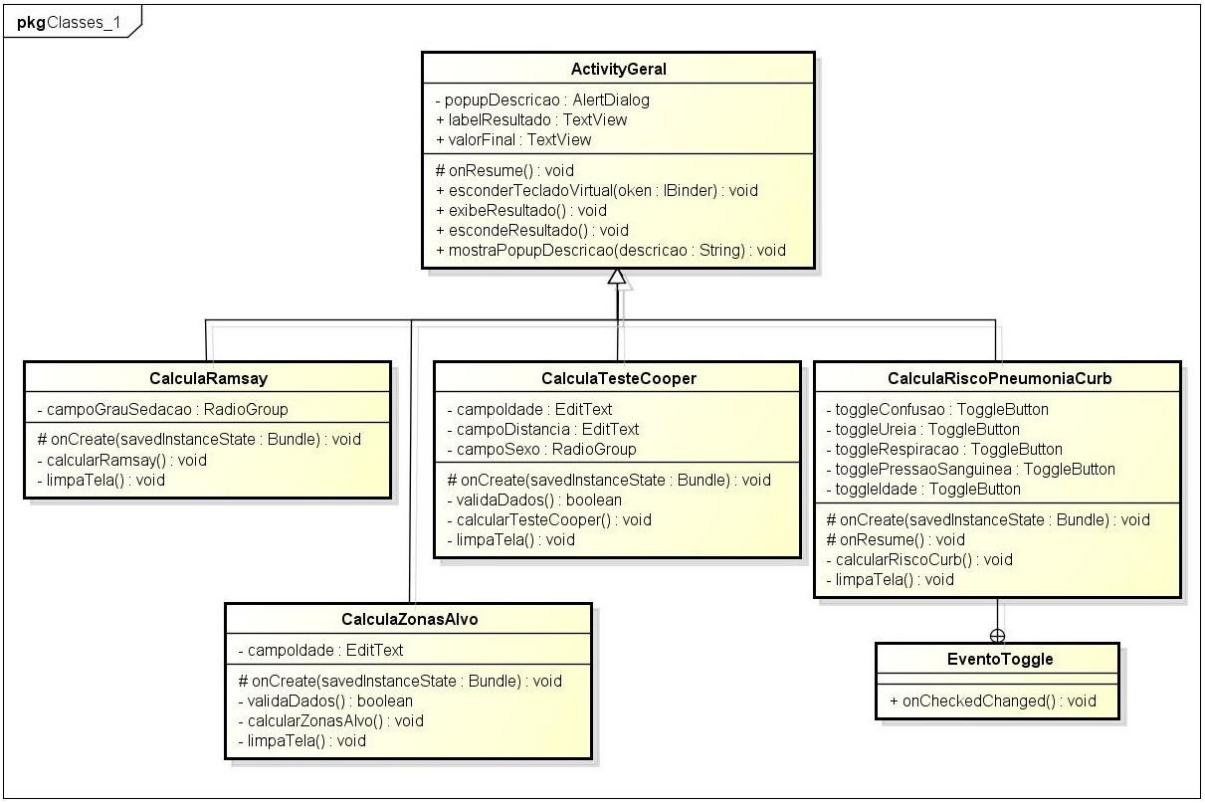

Figura 20 - Diagrama de Classes 5

3.2.3. Diagrama de Atividades

Outro diagrama imensurável na modelagem de um software é o diagrama de atividades, “voltado a detalhar o comportamento de um programa, isto é, descrevê-lo quando em execução” (Silva. 2007, p.155) [20]. A utilidade do diagrama de atividades na modelagem de um software é o detalhamento do que o programa faz, podendo descrever o comportamento do sistema como um todo ou detalhar os casos de uso.

Enquanto o diagrama de classes modela o sistema de forma estática, o de atividades modela o sistema de forma dinâmica, em tempo de execução. Este tipo de diagrama não tem o intuito de discriminar quem executa as atividades, e sim, exibir o que é feito. Conforme Silva, (2007, p.183) [20]

> Em função do conjunto de elementos de modelagem do diagrama de atividades, é possível produzir modelagens com ênfases bastante distintas – como com ou sem fluxo de objetos – e modelando contextos bastante diversos, como algoritmo de método, caso de uso ou um processo de negócio.

Todavia, apesar desta diversidade, no presente trabalho será utilizado o diagrama de atividades para detalhar o comportamento dos casos de uso do aplicativo *Cálculos Médico*. Os casos de uso que merecem refinamento por meio do diagrama de atividades são os referentes aos cálculos em si, como Anion Gap, Forrest, Killip, Ranson, Apgar, dentre outros.

Visualizando o diagrama de casos de uso, exposto na seção 3.2.1, observa-se que, para chegar à execução de qualquer uma das funcionalidades referente aos cálculos, executar-se-ão, obrigatoriamente, os casos de uso

Visualizar Cálculos e *Escolher Categoria*. Portanto, o diagrama de atividades abaixo representa, justamente, a execução destas duas funcionalidades.

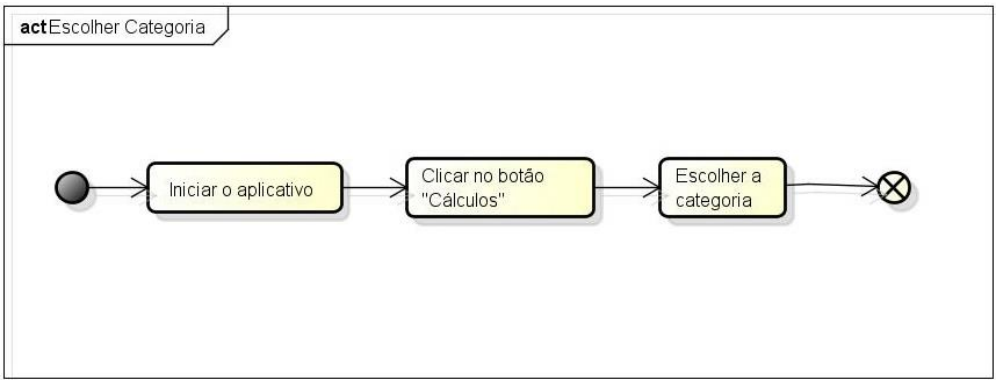

Figura 21 – Diagrama de atividades "Escolher Categoria"

Analisando a Imagem 21, o fluxo de execução inicia-se no círculo preenchido em preto, sendo que a única ação possível a ser tomada é "Iniciar o aplicativo". Após executada esta ação, o fluxo segue em caminho único, onde o usuário deverá clicar no botão "Cálculos", e, após o sistema listar as categorias disponíveis, o usuário deverá "Escolher a categoria" que deseja, finalizando o fluxo de execução acima.

Já na Figura 22, exibida abaixo, apresenta-se o diagrama de atividades "Calcular Anion Gap". Este diagrama representa a ação desde o inicio da execução do sistema até a exibição do resultado final da funcionalidade. Como o caso de uso "Escolher categoria" está incluso no "Calcular Anion Gap", suas ações não precisam ser modeladas novamente. Para evitar este retrabalho, basta adicionar este diagrama, na forma de atividade, no novo diagrama. Uma atividade é representada dentro de outro diagrama de atividades conforme "Escolher categoria" está abaixo, um retângulo com um ícone na forma de forcado.

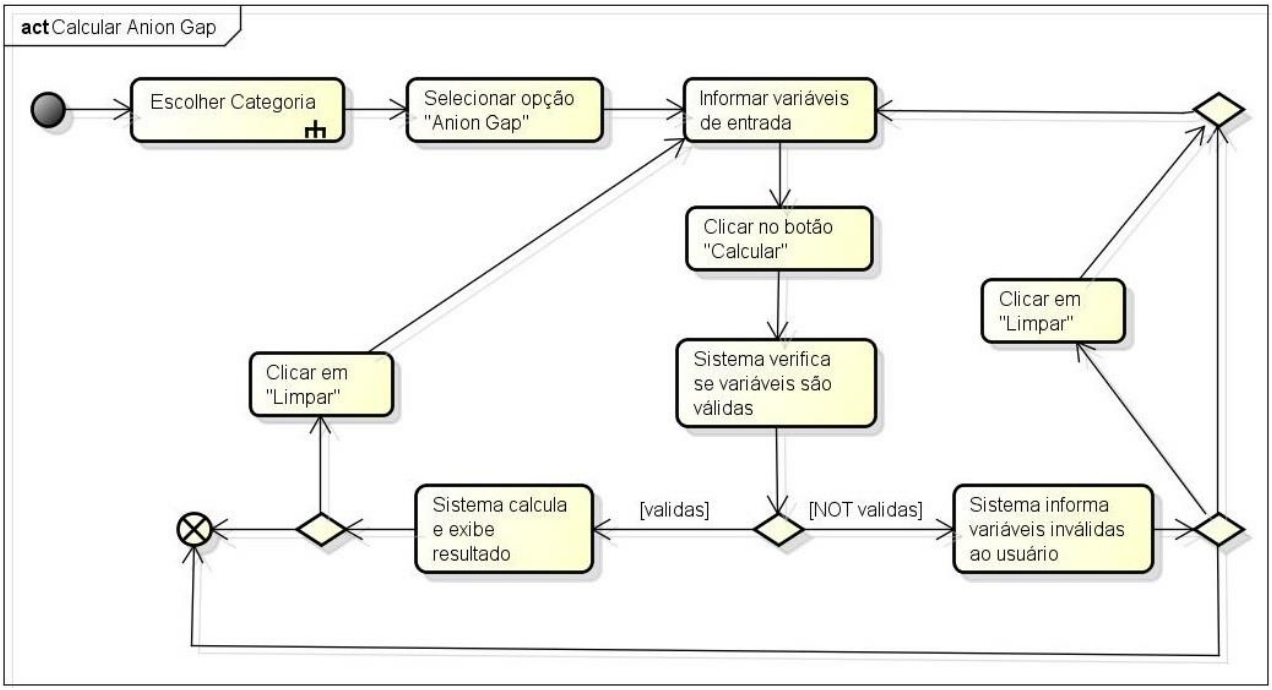

Figura 22 – Diagrama de atividades "Calcular Anion Gap"

Analisando o diagrama de atividades exposto acima, que refina o caso de uso “Calcular Anion Gap”, é possível observar que após a execução das ações da atividade “Escolher Categoria”, o usuário deverá selecionar a opção *Anion Gap* na lista de cálculos disponível. Após esta ação, o usuário informará as variáveis de entrada e clicará no botão “Calcular”. Com isso, o sistema verificará se as variáveis foram informadas corretamente. Se sim, o sistema exibe o resultado final e o usuário terá duas opções para seguir: finalizar a execução ou clicar em “Limpar” e fazer um novo cálculo. Por outra via, se as variáveis não foram informadas corretamente, o sistema informará a irregularidade ao usuário, que terá três caminhos distintos para seguir. O primeiro, e mais simples, é finalizar a execução. O segundo caminho é clicar em “Limpar” e informar todas as variáveis novamente. O terceiro e último caminho é arrumar a(s) variável(eis) errada e clicar em “Calcular”, seguindo o fluxo já detalhado.

Portanto, no diagrama de atividades acima é possível verificar todo o fluxo de execução necessário para o sistema executar a funcionalidade de calcular o valor do *Anion Gap*, todavia, não é discriminado qual classe ou método do sistema será responsável por executar esta ou aquela ação, isto é, o diagrama de atividades exibe o que deve ser feito, mas não por quem deve ser feito.

O cálculo do *Anion Gap* segue, resumidamente, a seguinte lógica: o usuário escolhe o cálculo e insere as variáveis de entrada, o sistema valida e, se correto, exibe o resultado, se incorreto, informa os erros. Além deste cálculo, outras nove funcionalidades do sistema seguem este mesmo raciocínio: *Escala Meld, Fórmula Cockroft-gault, Fórmula Parkland, IMC (Índice Massa Corporal), LDL (Colesterol), Margem de Gordura Corporal, Massa Óssea Estimada, Teste Cooper (12 minutos)* e *Zona Aeróbica Treinamento*. Assim sendo, o diagrama de atividades para refinar estes casos de uso é muito similar ao do “Calcular Anion Gap”, sendo que a única modificação necessária é escolher a opção correta na ação “Selecionar opção *nome_do_cálculo*”.

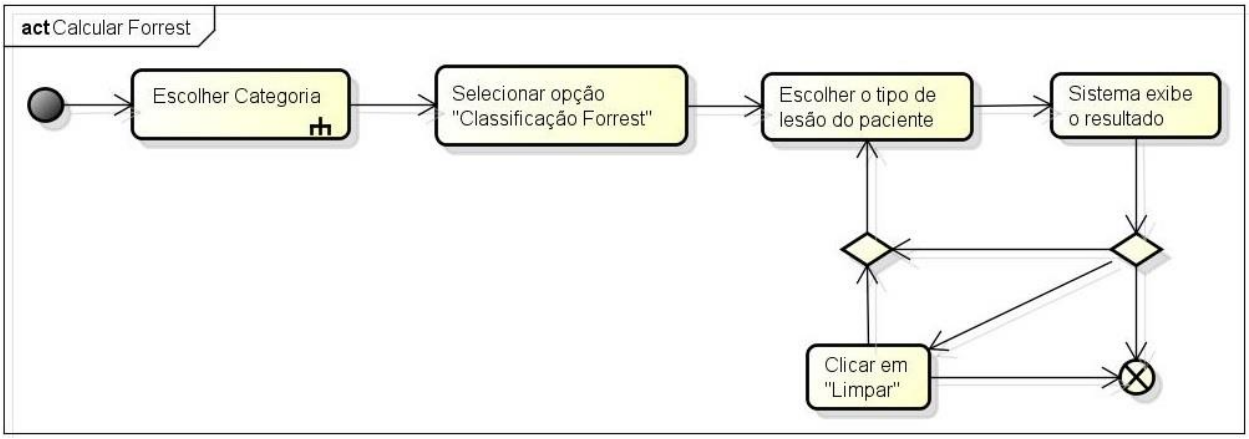

Figura 23 - Diagrama de atividades "Calcular Forrest"

Assim como todos os demais, o diagrama de atividades “Calcular Forrest” também engloba as ações modeladas na atividade “Escolher Categoria”. Após estas ações, o usuário deverá selecionar a opção “Classificação Forrest” e escolher o tipo de lesão do paciente. Com isso, o sistema estará apto a informar o resultado, e o usuário terá que escolher entre encerrar a execução, clicar em “Limpar” ou, simplesmente, escolher outro tipo de lesão e analisar o novo resultado. É possível observar no diagrama que, após clicar em “Limpar”, o usuário poderá encerrar a execução ou escolher outro tipo de lesão do paciente.

Os diagramas a seguir, “Calcular Killip”, “Calcular Ashworth” e “Calcular Ramsay”, presentes nas Figuras 24, 25 e 26, respectivamente, seguem a mesma lógica e mesmo fluxo de execução do “Calcular Forrest”, explicado no parágrafo anterior.

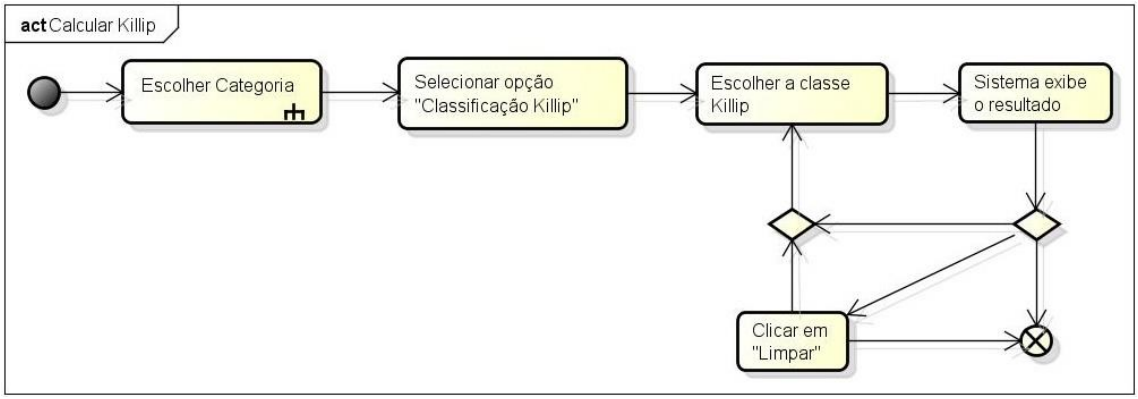

Figura 24 - Diagrama de atividades "Calcular Killip"

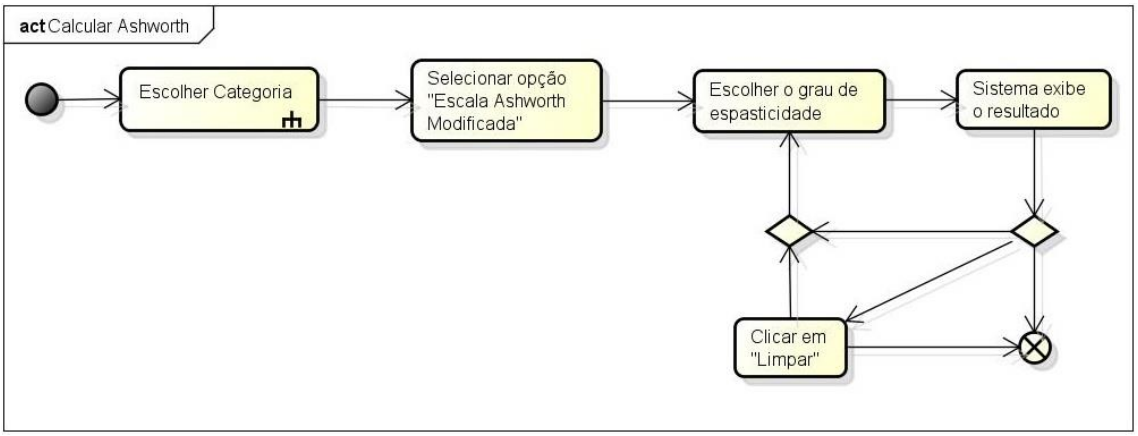

Figura 25 - Diagrama de atividades "Calcular Ashworth"

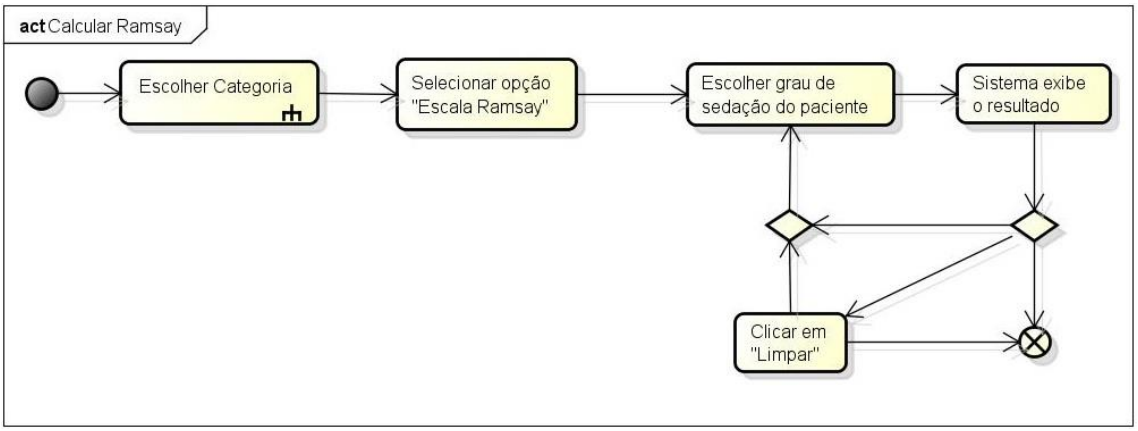

Figura 26 - Diagrama de atividades "Calcular Ramsay"

O próximo grupo de cálculos semelhantes será representado pelo diagrama de atividades “Calcular Apgar”. Todos os cálculos são iniciados da mesma forma, por meio da execução das ações presentes na atividade “Escolher Categoria”, seguido da ação para selecionar o cálculo, que no caso é “Escala Apgar”. Após isso, a interface para inserção de valores será exibida ao usuário e ele deverá selecionar um item. Os cálculos desse grupo possuem alguns critérios, onde, em cada um deles, deve ser selecionado um item.

O sistema verificará se há um item selecionado para cada critério do cálculo e, se não houver, exibirá uma mensagem de erro e deixará livre para o

usuário selecionar o item do próximo critério. Este ciclo prossegue até todos os critérios estiverem com uma resposta selecionada. Neste ponto, o sistema exibirá o resultado final ao usuário.

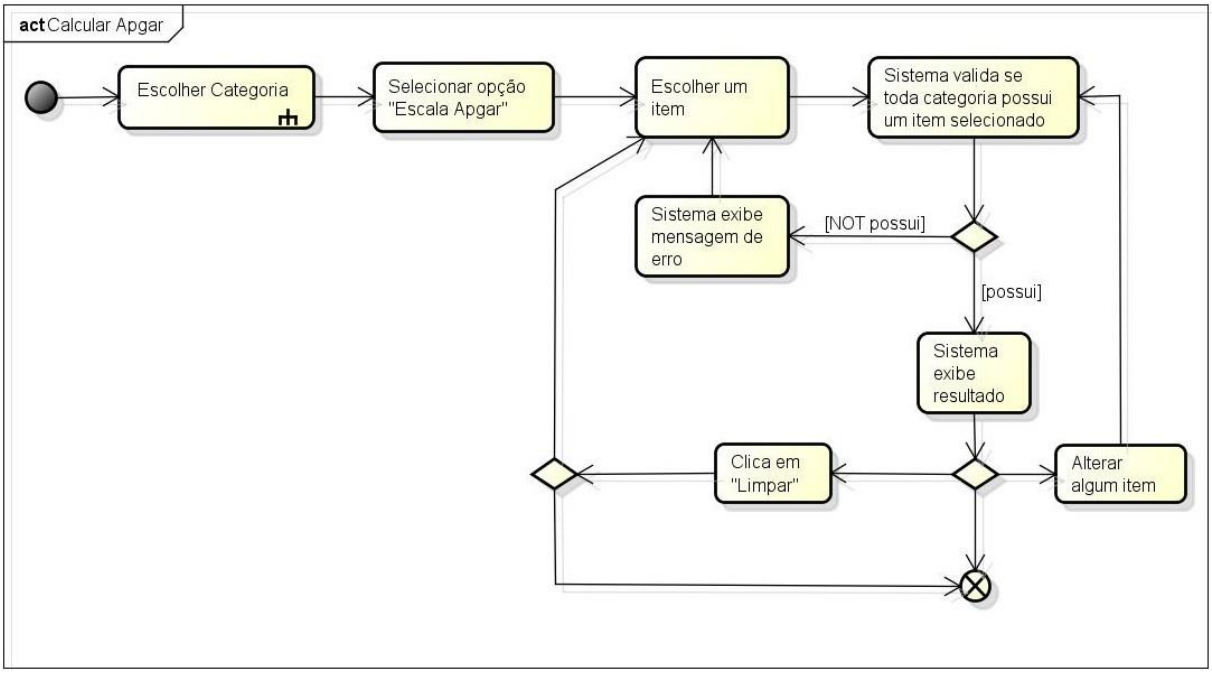

Figura 27 - Diagrama de atividades "Calcula Apgar"

Após exibição do resultado final, o usuário poderá seguir três caminhos distintos: encerrar a aplicação, alterar o item de alguma categoria ou clicar em “Limpar”. Se alterado algum item, o sistema recalculará a *Escala Apgar* e fornecerá um novo resultado. Se clicado em “Limpar”, o usuário poderá encerrar a execução ou inserir novas variáveis para um novo cálculo.

O refinamento dos casos de uso “Calcular Child Pugh” e “Calcular Glasgow” seguem exatamente a mesma lógica e fluxo de execução do diagrama presente na Figura 27, com a única diferença de que, ao invés de selecionar o cálculo *Escala Apgar*, deverão ser selecionados ou *Escala Child Pugh* ou *Escala Glasgow de Coma*.

Por fim, o último grupo de cálculos, no quesito similaridade, engloba *Critério Ranson* e *Risco Pneumonia CURB-65*. Após escolher a categoria e a opção de cálculo, o usuário deverá selecionar “Sim” ou “Não” para cada uma das variáveis de entrada. Quando alterada uma variável, o sistema calcula e exibe o resultado. Após isso, o usuário poderá escolher entre encerrar a aplicação, alterar o valor de alguma variável de entrada ou clicar em “Limpar”. Se alterada alguma variável, o sistema recalcula o resultado. Se clicado em “Limpar”, o usuário poderá encerrar a aplicação ou voltar a inserir valores para as variáveis de entrada. Este fluxo está explicitado no diagrama de atividades da Figura 28.

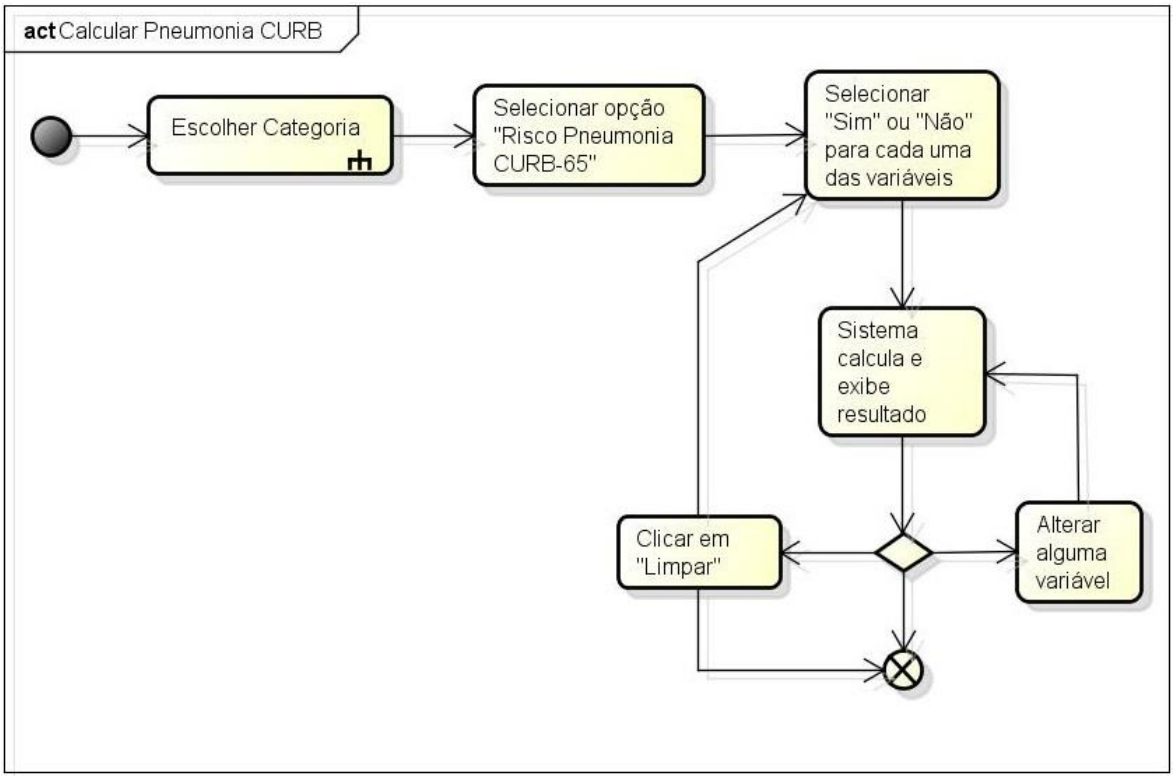

Figura 28 - Diagrama de atividades "Calcular Pneumonia CURB"

4. Desenvolvimento do Aplicativo

Depois de concluídas as etapas de levantamento, análise e especificação de requisitos e de modelagem do software, inicia-se a etapa de implementação, que nada mais é do que escrever o código. Como já mencionado na fundamentação teórica e na configuração do ambiente, utilizar-se-á o eclipse como IDE de desenvolvimento e Java como linguagem de programação.

4.1.1. Estrutura de Pastas e Arquivos (Package Explorer)

Nesta primeira etapa, será explicitada a estrutura de pastas e arquivos do projeto *Cálculos Médicos*. Esta estrutura, também conhecida como *Package Explorer*, deve ser bem estudada e configurada, pois tende a facilitar o desenvolvimento e, futuramente, a manutenção do software.

Dentro da raiz do projeto, há duas pastas principais, que são *src* e *res*. Ainda na raiz, está presente o *AndroidManifest.xml*, que é o arquivo de configuração obrigatório a qualquer projeto Android, cujo objetivo é listar permissões necessárias, além de todas os componentes do sistema e outras configurações importantes.

Na Figura 29, além das já citadas, visualiza-se a pasta *gen (Generated Java Files)*. Dentro desta pasta ficará o arquivo R.java, gerado automaticamente na compilação do projeto. Além disso, há a imagem *ic_launcher-web.png*, que é o ícone do aplicativo.

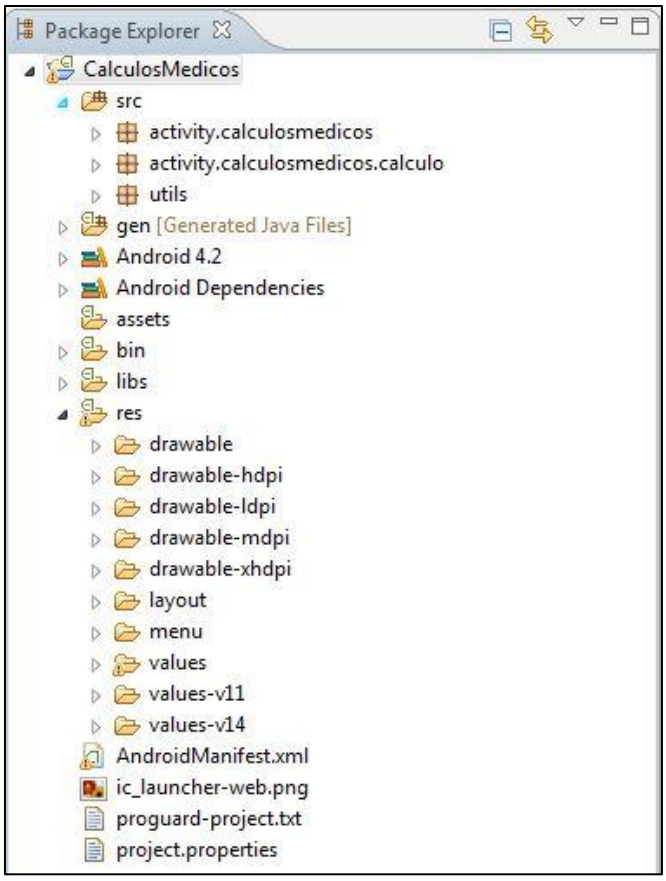

Figura 29 - Pastas e arquivos do aplicativo *Cálculos Médicos*

Dentro da pasta *src* estão localizadas todas as classes *.java* do sistema. Como já visto, em um aplicativo Android, toda interface com o usuário será, obrigatoriamente, representada por uma *Activity*, ou seja, toda interface com o usuário será uma classe Java que estende a *Activity* do Android. A Figura 30 lista, exatamente, estas *Activities*, juntamente com as demais classes codificadas para funcionamento do sistema.

A pasta *src* contém três subpastas, conhecidas como *package*, que são: *activity.calculosmedicos*, *activity.calculosmedicos.calculo* e *utils*. Dentro do primeiro *package*, *activity.calculosmedicos*, estão presentes *Activities* de apoio ao sistema, isto é, partes integrantes do sistema, mas que não representam cálculos. Já no *package activity.calculosmedicos.calculo* estão todas as

Activities que representam os cálculos disponíveis no aplicativo. Por fim, no último *package*, utils, estão duas classes Java que não são filhas da *android.app.Activity*.

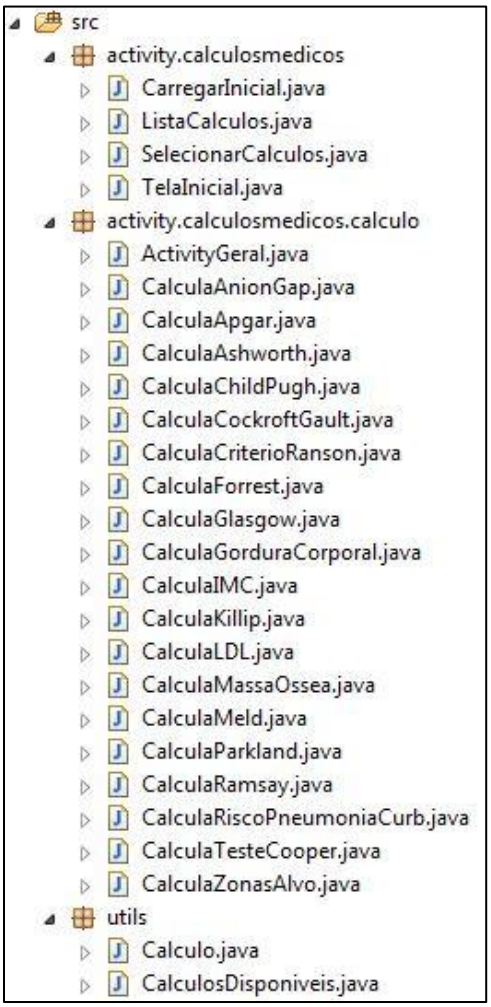

Figura 30 - Classes Java do aplicativo

Na Figura 29, observa-se uma série de subpastas dentro da pasta *res*. Dentre todas, destacam-se *drawable*, *layout* e *values*. Dentro da pasta *drawable* estão localizadas as imagens utilizadas no sistema, com exceção do ícone do aplicativo, que fica na raiz do projeto. Na pasta *layout* estão todos os arquivos *.xml*, um para cada *Activity*, ou seja, cada interface possui um arquivo

XML para configurações de layout, como inclusão de componentes, botões, caixas de texto, rótulos, espaçamentos, cores, etc.

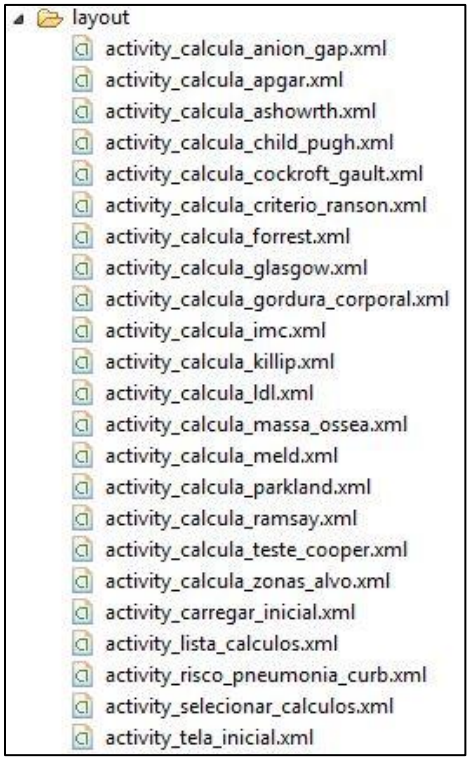

Figura 31 - Arquivos XML para layout das interfaces

Já na pasta *values*, está presente o arquivo *strings.xml*, também padrão em qualquer aplicativo Android. Este arquivo contém todos os Strings utilizados no sistema, tendo como função primordial a internacionalização do software.

4.1.2. Processo de Internacionalização do Aplicativo

Um aspecto bastante importante no desenvolvimento de um software, devido à globalização e à facilidade do sistema ser comercializado em outros países, é a sua internacionalização. Com a interação de mercados, a internacionalização tornou-se uma grande preocupação para empresas e desenvolvedores (De

Lucca, 2004.) [21]. Ainda segundo De Lucca (2012, p.2012), para um software ser considerado internacionalizado

> Todo o texto do programa visível ao usuário deve poder ser traduzido, de forma a adaptar-se ao novo local de distribuição. Mais do que isso, o texto do programa deve ser *culturalmente neutro*, ou seja, não deve utilizar expressões que fazem parte de uma determinada cultura específica (como gírias ou expressões idiomáticas, por exemplo).

A internacionalização no desenvolvimento Android é uma tarefa bastante simples, graças à existência dos arquivos *R.java* e *strings.xml*.

No arquivo *strings.xml* ficam todas traduções do aplicativo, no formato chave valor. Por exemplo, o nome do aplicativo é armazenado da seguinte maneira: “<string name="app_name">Cálculos Médicos</string>”. Como outro exemplo, o nome do cálculo *Risco de Pneumonia CURB-65* é armazenado da mesma forma: “<string name="risco_curb">Risco de Pneumonia CURB-65</string>”.

Quando o projeto é compilado, cria-se automaticamente o arquivo *R.java.* Este arquivo jamais deve ser alterado manualmente, pois, sempre que o projeto for recompilado, ele será excluído e criado novamente, fazendo com que todas as alterações feitas sejam perdidas. A classe *R* contém classes estáticas internas, que fazem referências às figuras, aos componentes de layout, aos arquivos de layout e às Strings do projeto.

Portanto, para acessar qualquer String do sistema, basta utilizar a referência que é criada na classe *R*, sendo que este recurso pode ser utilizado

em qualquer classe ou arquivo do projeto, pois a classe *R* é acessível em todo o código.

Em uma classe Java, para acessar uma String internacionalizada no arquivo *strings.xml*, basta invocar o método *getString()* da classe contexto, também disponível em todo código, passando como parâmetro a referência da String na classe *R*, como segue: “getString(R.string.app_name)”. O mesmo acontece para o nome do cálculo *Risco de Pneumonia CURB-65* e para qualquer outra String do sistema. No caso deste cálculo, o acesso à String ficaria da seguinte maneira: “getString(R.string.risco_curb)”.

Em um arquivo XML, o acesso às Strings também é feito de forma simples e direta. Por exemplo, no componente do tipo *TextView,* que representa o rótulo “Frequência Cardíaca” presente no cálculo de Apgar, deve ser informado o valor do seu conteúdo, ou seja, o texto que será apresentado na tela. Esta codificação é feita da forma a seguir: “android:text="@string/label_freq_cardiaca"”. Com isso, ao exibir a interface deste arquivo XML, será recuperado o valor da String com chave *label_freq_cardiaca* no arquivo *strings.xml*, apresentando-o ao usuário.

Portanto, para disponibilizar o aplicativo desenvolvido em outros idiomas, bastar criar outro arquivo *strings.xml* com as mesmas chaves, todavia com os valores das Strings no idioma requerido.

Logicamente, internacionalizar um software não é somente alterar o idioma dos seus Strings, existem várias outras questões a serem consideradas, mas não serão abordadas.

Além dos Strings, os arquivos e componentes de layout e imagens também podem ser acessado da mesma maneira, por meio da utilização de referências da *R.java*, todavia utilizando a classe interna correta. Exemplificando, para referenciar uma imagem, deve-se utilizar o padrão *R.drawable.nome_da_imagem* e para referenciar um layout, deve-se utilizar *R.layout.nome_do_layout*.

Como já discriminado, toda *Activity* possui um arquivo de layout. A ligação entre ambos é feita no método *onCreate()* da *Activity,* através da invocação do layout por meio da utilização da classe *R*. Portanto, de forma obrigatória, todo método de criação de uma atividade deve chamar o método para informar qual seu layout, sendo que este requisito é alcançado por meio do código “setContentView(R.layout.nome_do_layout)”.

4.1.3. Interfaces e Lógicas de Funcionamento

Quando iniciado o sistema, isto é, quando clicado no ícone do aplicativo (ver Figura 32) em um dispositivo Android, será exibida a tela inicial do *Cálculos Médicos* ao usuário.

Para que o aplicativo saiba qual deve ser a primeira *Activity* a ser executada, deve-se configurar esta opção no *AndroidManifest.xml*. Como explicitado anteriormente, este arquivo de configuração contém todas as *Activities* do sistema, sendo que somente uma delas contém o parâmetro indicando que o aplicativo deverá começar sua execução iniciando esta atividade.

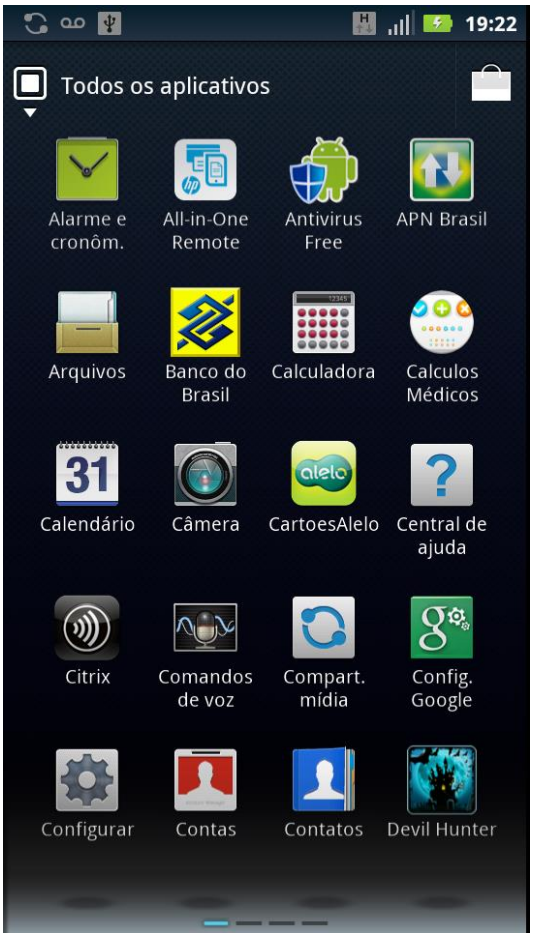

Figura 32 - Aplicativos Android, incluindo *Cálculos Médicos*

No aplicativo desenvolvido, a *Activity* inicial do sistema é a *CarregarInicial.java*, cujo XML é *activity_carregar_inicial.xml*. Para indicar que o sistema deve iniciar executando esta *Activity*, utiliza-se um *intent-filter* dentro da tag *activity* no arquivo manifesto da aplicação. O *intent-filter* da atividade inicial possui duas tags, *action*, com valor *android.intent.action.MAIN*, e *category*, com valor *android.intent.category.LAUNCHER*. Com isso, a aplicação saberá que, ao ser iniciada, deverá começar executando esta *Activity*.

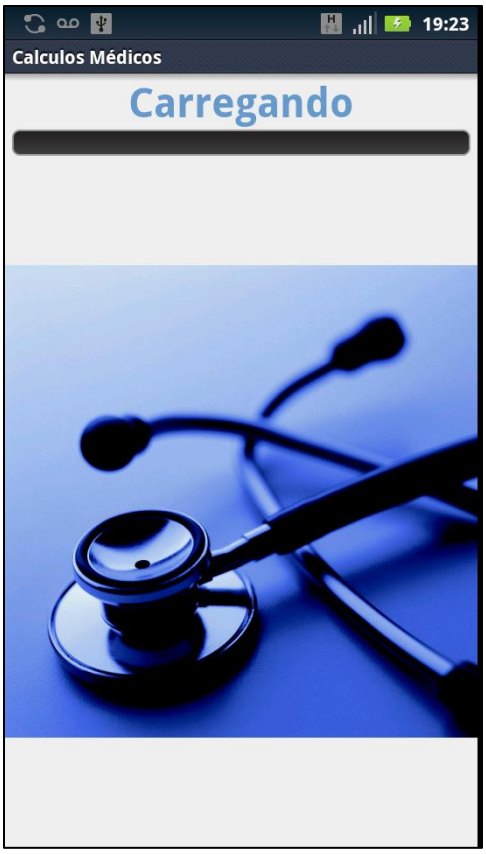

Figura 33 - Carregando: a Activity inicial do aplicativo

Portanto, depois de clicado no ícone *Cálculos Médicos*, será exibida ao usuário a interface da Figura 33. Esta interface possui uma imagem decorativa e uma barra de status indicando que a aplicação está sendo iniciada. O layout desta interface é codificado no arquivo *activity_carregar_inicial.xml*, e a lógica de programação está na classe Java *CarregarInicial*. Lembrando que, por ser uma interface, é consequentemente uma *Activity*, devendo estender a classe *android.app.Activity*.

Quando a barra de status chega ao final, esta *Activity* é encerrada e é iniciada uma nova, representando a tela inicial do sistema (*TelaInicial.java*). A tela inicial é uma interface simples, que possui somente dois botões: *Informações* e *Cálculos*. O botão *Informações* levará o usuário a uma nova interface, com informações a respeito do produto e do desenvolvedor. Já o

botão *Cálculos*, exibirá as categorias de cálculos disponíveis no sistema. Como ocorre com todas as *Activities*, a tela inicial possui um arquivo XML de layout, chamado *activity_tela_inicial.xml*.

A interface de informações, conforme pode ser observado na Figura 34, apresenta informações básicas sobre o que é o aplicativo e sobre quem o desenvolveu. Além disso, disponibiliza um e-mail para contato, possibilitando o envio de dúvidas ou sugestões ao desenvolvedor.

Figura 34 - Interface de informações do aplicativo

De acordo com o que foi apresentado no levantamento dos requisitos, o aplicativo possui três categorias de cálculos: educação física, fisioterapia e medicina. A *Activity SelecionarCalculos.java* exibe estas opções de escolha ao

usuário, além da opção de listar todos os cálculos disponíveis, independente da categoria na qual ele se insere, conforme Figura 35.

Figura 35 - Interface para selecionar a categoria dos cálculos

Neste ponto, torna-se importante explicitar como estão codificados os cálculos. Uma das classes do *package utils* na pasta *src* é a Calculo.java. Esta classe nada mais é que um *enum* listando todos os cálculos disponíveis no sistema, juntamente com seu nome e categoria a qual pertence. Por exemplo, o cálculo *Anion Gap* pertence à categoria de medicina, sendo listado no *enum Calculo* como "ANION_GAP("Anion Gap", "medicina")". Todos os cálculos disponíveis possuem uma entrada nesta classe. Alguns cálculos, como *Escala Glasgow*, são pertencentes a mais de uma categoria, sendo listados da forma que segue: "ESCALA_GLASGOW("Escala Glasgow de Coma","medicina,fisioterapia")". Neste caso, como *Escala Glasgow* pertence a

duas categorias, ela estará presente na lista de cálculos quando o usuário pressionar o botão “Fisioterapia” ou “Medicina” e, logicamente, “Todos”.

A outra classe presente no *package utils* é a *CalculosDisponiveis.java*. Esta classe possui quatro métodos: *getCalculosDisponiveis(), getCalculosEducaocaoFisica(), getCalculosFisioterapia()* e *getCalculosMedicina()*. Se, na interface de categorias (Figura 35), o usuário clicar no botão para listar todos os cálculos, será chamado o método *getCalculosDisponiveis()*, que retornará todos os cálculos disponíveis no sistema. Se o usuário clicar no botão *Educação Física*, será chamado o método *getCalculosEducaocaoFisica()*, que listará todos os cálculos que pertencem à categoria de educação física. O mesmo ocorre para os botões *Fisioterapia* e *Medicina*.

Independentemente da categoria de cálculo escolhida, será exibida a *Activity ListarCalculos.java*. Está é a única interface de listagem do aplicativo, que será alimentada de acordo com a categoria escolhida pelo usuário. O layout desta interface é montado no XML *activity_lista_calculos*, sendo a lista um componente do tipo *ListView*. Na Figura 36, segue a lista de cálculos após clique no botão *Todos*. É importante destacar que não aparecem todos os cálculos na interface do dispositivo, tendo o usuário que percorrer a lista, para cima ou para baixo, para visualizar os demais cálculos existentes.

Enfim, neste ponto, o usuário terá em mãos a lista dos cálculos disponíveis, de acordo com a categoria escolhida. Quando clicado em algum dos itens da lista, o sistema iniciará a *Activity* referente ao cálculo selecionado. Por exemplo, se o usuário clicar em *Escala Apgar*, será chamado o método

para iniciar a atividade *CalculaApgar.java*. Este chamado ocorre da seguinte forma: “startActivity(new Intent(getApplication(), CalculaApgar.class));”. Todas as *Activities* são iniciadas desta maneira, alterando somente o nome da classe.

Figura 36 - Interface exibindo todos os cálculos disponíveis no sistema

Como todas as *Activities* de cálculos possuem características e comportamentos comuns, criou-se uma superclasse, chamada *ActivityGeral*, que estende *android.app.Activity* e deve ser estendida por todas as classes Java que representam cálculos. A *ActivityGeral* possui alguns métodos básicos, que são utilizados por todas as *Activities* que as entendem. Os métodos são *onResume()*, que esconde o teclado virtual quando a *Activity* é carregada, *escondeTecladoVirtual()*, que, quando invocado, também esconde o teclado virtual, *exibeResulatdo()*, *escondeResultado()* e *mostraPopupDescricao()*.

Todas as interfaces de cálculo possuem um ícone de informação (), que, quando clicado, abre um *popup* que contém a descrição e a fórmula utilizada para o cálculo em questão. O método *mostraPopupDescricao()* da *ActivityGeral* é o responsável por exibir este *popup*, que é um objeto do tipo *AlertDialog.Builder*.

Na lista da Figura 36, caso o usuário clique em *Anion Gap*, o aplicativo iniciará a *Activity* chamada *CalculaAnionGap*. Como já explicitado, esta *Activity* estende a *ActivityGeral*, fato este que pode ser observado nos diagramas de classes disponibilizados na modelagem do software. A classe *CalculaAnionGap.java* possui métodos que se apresentam em quase todas as outras *Activities* do sistema, como *onCreate(), validaDados(), calculaAnionGap()* e *limpaTela()*.

No método *onCreate()* são instanciados os componentes presentes na interface do sistema e implementados eventos nos botões e no ícone de informações. Todos os botões do aplicativo são do tipo *android.widget.Button*, e possuem como evento um *View.OnClickListener()*, inserido no método *setOnClickListener()*.

O método *validaDados()*, como o próprio nome indica, é responsável por validar os dados de entrada do usuário. Este método não está presente em todas as atividades, pois há alguns cálculos onde, na verdade, não são feitos cálculos, apenas classificações ou exibições de escalas. Nestes casos, não há nada a ser validado, visto que o usuário deverá escolher uma das opções disponíveis e o sistema exibe o valor correspondente como resultado.

Toda *Activity* possui um método do tipo *calculaNomeDaActivity()*, onde são implementadas as lógicas dos cálculos, opções de classificação e criação e execução das fórmulas. A classe *CalculaAnionGap* possui o método principal *calculaAnionGap()*, a classe *CalculaRamsay* possui o método *calcularRamsay()*, a classe *CalculaZonasAlvo* possui o método *calcularZonasAlvo()*, e assim por diante.

Por fim, o método *limpaTela()*, como o próprio nome explicita, é responsável por limpar as entradas dos usuários e também os resultados já exibidos, deixando a interface pronta para um novo cálculo. A Figura 37 apresenta a interface do cálculo de *Anion Gap*, sem entradas do usuário.

Figura 37 - Interface do cálculo de Anion Gap

No tópico de levantamento de requisitos, foram detalhados os cálculos disponíveis no aplicativo. Conforme visto na interface da Figura 37, e no detalhamento do *Anion Gap* nos requisitos do sistema, há três campos para inserção de dados por parte do usuário, que são nível de sódio, nível de cloro e nível de bicarbonato, todos os três devendo ser inseridos na unidade de medida *mmol/L*.

Depois de inseridos valores de entrada, o usuário deverá clicar no botão *Calcular*, fazendo com que o sistema execute o método *calcularAnionGap()* e mostre o resultado na interface, conforme exibido na Figura 38. Todavia, antes de fazer o cálculo, são validados os dados inseridos pelo usuário. Caso haja erro, ou algum dado não tenha sido inserido, a interface exibe a irregularidade.

Figura 38 - Interface com resultado do Anion Gap

O resultado do *Anion Gap* não possui nenhuma interpretação, ou seja, é exibido apenas o resultado numérico, que será avaliado pelo profissional. Há alguns cálculos, mostrados na sequência, que além de um resultado numérico possuem uma classificação escrita, auxiliando ainda mais o profissional da saúde.

Caso seja clicado no botão *Limpar*, será invocado o método *limpaTela()* da classe *CalculaAnionGap*, que ao ser executado, removerá os valores de entrada do usuário – nível de sódio, nível de cloro e nível de bicarbonato – e os rótulos de resultado, deixando a interface limpa para inserção de novos dados. Se clicado no botão *Voltar*, o sistema volta à tela anterior, exibindo novamente a lista de cálculos.

Por fim, há o ícone de informações na interface do *Anion Gap*. Este botão, ao ser selecionado, abrirá um *popup* com a descrição do cálculo em execução. Se houver uma fórmula esta também é exibida, consoante à Imagem 39.

Há vários outros cálculos do sistema que seguem este mesmo padrão do *Anion Gap*, ou seja, o usuário insere valores numéricos, o sistema valida estes valores e exibe o resultado. Um exemplo disto é o a *Escala MELD*, onde o usuário deve inserir os valores de bilirrubina, creatinina e INR. Na Figura 40, o usuário não inseriu um valor para INR, mas clicou em calcular, fazendo com que o sistema exiba um erro de validação.

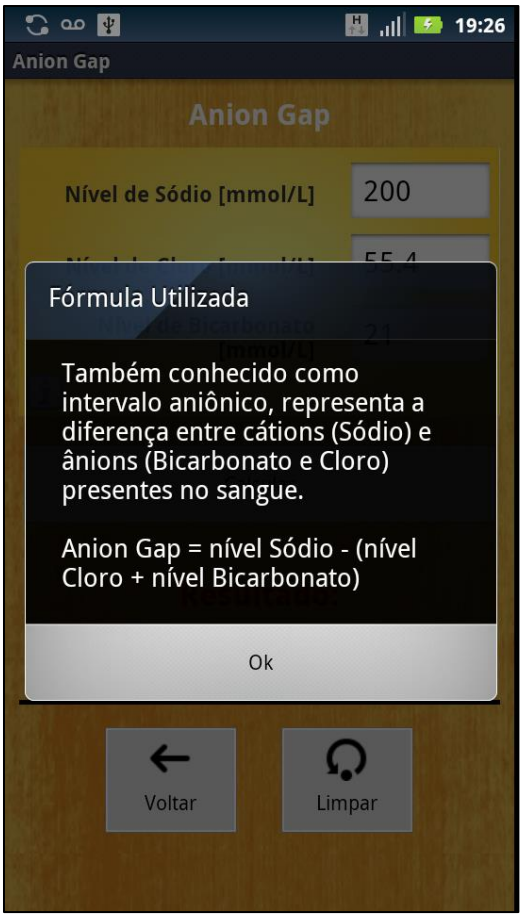

Figura 39 - Interface exibindo a descrição do Anion Gap

O usuário, defrontando-se com a mensagem de erro “Campo Obrigatório”, terá as opções de simplesmente inserir o valor da variável faltante, ou clicar em limpar e inserir todos os valores novamente. De qualquer forma, após a inserção do dado irregular, o usuário clicará em calcular, fazendo com que o sistema novamente valide os dados. Se, nesta segunda tentativa, os dados forem considerados válidos, será exibido o resultado final na interface do calculador da *Escala MELD*.

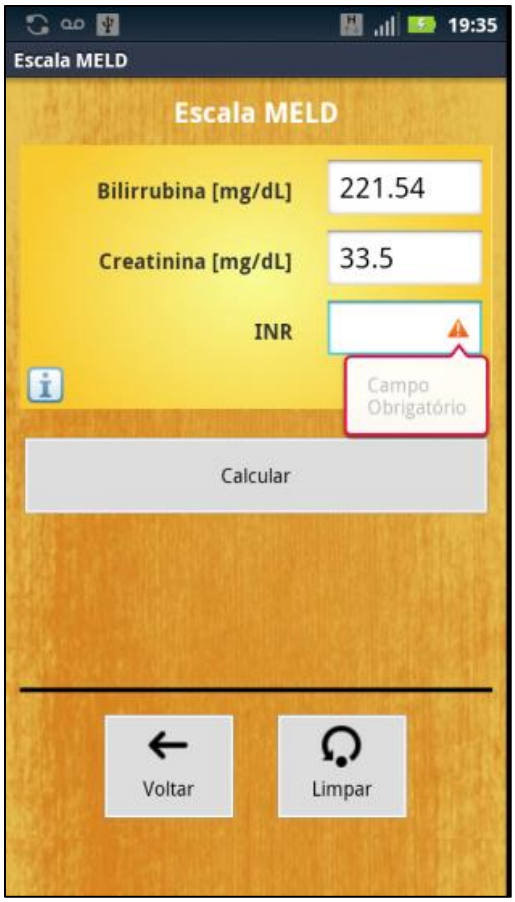

Figura 40 - Interface da Escala MELD com erro de validação

Como dito anteriormente, o resultado do *Anion Gap* exibe somente um resultado numérico, sem interpretação. Em contrapartida, o resultado da *Escala MELD*, além do valor final da escala, exibe também uma informação importante ao profissional que está manuseando o aplicativo. Como explicitado no levantamento e análise de requisitos, além do valor da escala, o resultado oferece a porcentagem do risco de mortalidade do paciente nos próximos três meses. Este resultado poder ser visualizado na Figura 41.

As outras interfaces de cálculo que seguem este mesmo padrão são, além de *Anion Gap* e *Escala MELD*, *Fórmula de Cockroft-Gault, Fórmula de Parkland, IMC (Índice Massa Corporal), LDL (Colesterol), Margem de Gordura*

Corporal, Massa Óssea Estimada, Teste de Cooper (12 minutos) e *Zona Aeróbica Treinamento*.

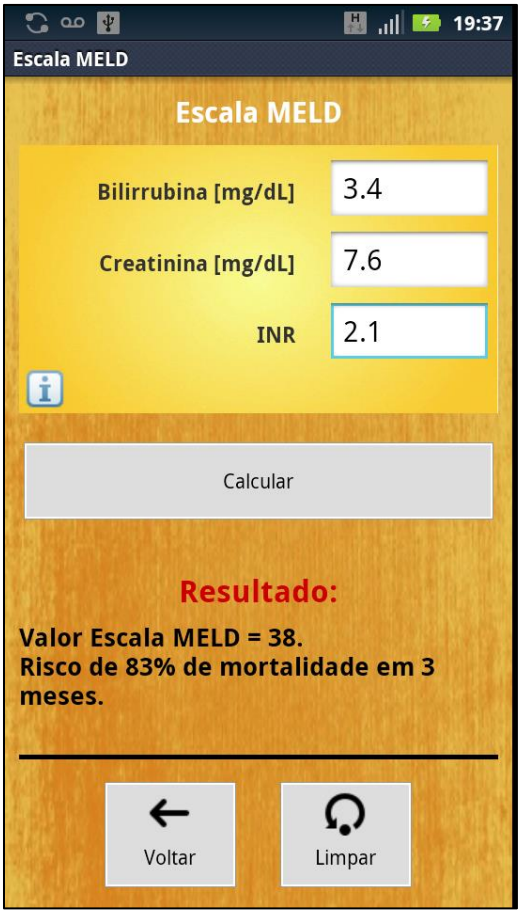

Figura 41 - Interface com resultado da Escala MELD

Outro grupo de cálculos onde interfaces de inserção e validação de dados se assemelham é formado por *Critério de Ranson* e *Risco de Pneumonia CURB-65*. Nestes dois tipos de cálculos, há uma série de variáveis onde o usuário entrará com os valores *Sim* ou *Não*. Este valor indica que a variável não está presente, aquele significa que a variável está presente. Por exemplo, no *Risco de Pneumonia CURB-65*, há uma variável descrita como *Idade >= 65 anos*. Se o paciente que está sendo analisado possuir mais de sessenta e cinco anos, esta variável deve ser configurada como *Sim*, caso contrário, como *Não*.

Neste tipo de interface não há nenhuma validação, pois não há como o usuário deixar de inserir ou inserir valor errado, visto que a interface não permite. Analisando a tela é possível observar também que não há o botão calcular. Neste caso, foi adicionado um evento de alteração de estado em cada um dos campos *Sim* ou *Não*, ou seja, sempre que uma variável de resposta tiver seu valor alterado, o risco de pneumonia, ou critério de Ranson, será calculado e exibido, não havendo necessidade de um botão *Calcular*.

Nestas duas *Activities*, cada uma das variáveis de entrada (*Sim* ou *Não*) é um objeto do tipo *ToogleButton*. Para inserção do evento de alteração de estado, foi invocado o método *setOnCheckedChangeListener()*, passando como parâmetro um objeto da classe *OnCheckedChangeListener*, que, na sua implementação, chama o método *calculaRiscoCurb()* ou, no caso do critério de Ranson, *calculaCriterioRanson()*. Com isso, sempre que alterada uma variável, o método para fazer o cálculo em questão é executado, exibindo o novo resultado ao usuário.

Como todos os outros cálculos, os desse grupo também possuem o ícone de informações em sua interface. Na Figura 42, é possível observar o *Risco de Pneumonia CURB-65*, com variáveis alteradas e seu resultado de momento, e também o *popup* com descrição a respeito deste cálculo.

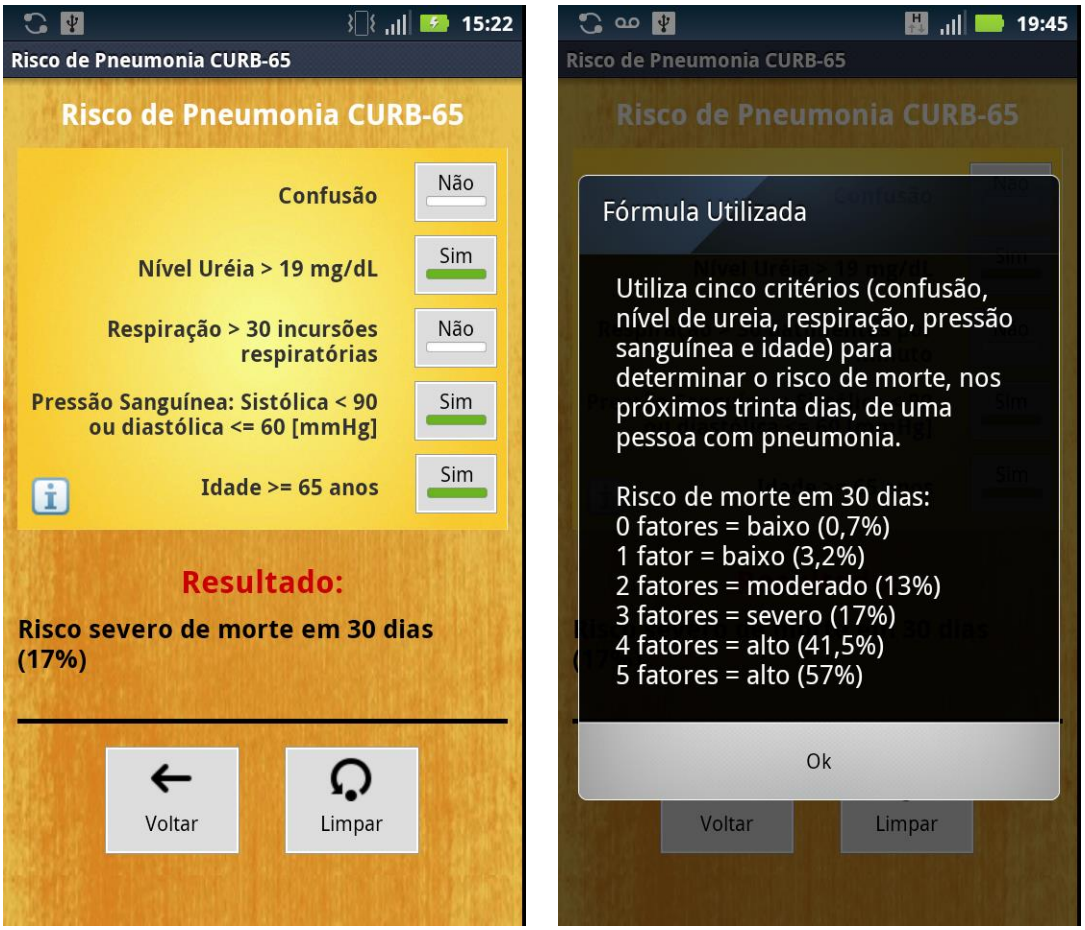

Figura 42 – Interface Risco de Pneumoria CURB-65: resultado e descrição

O grupo formado por *Classificação de Forrest, Classificação de Killip, Escala de Ashworth Modificada* e *Escala Ramsay* também possui características semelhantes entre seus integrantes. Os cálculos deste grupo têm como característica possuir um grupo de opções, sendo possível a seleção de somente uma, com o resultado sendo exibido automaticamente, novamente sem a necessidade de pressionar o botão *Calcular*.

A *Classificação de Forrest* é um dos exemplos deste tipo de interface. Neste caso, o usuário terá que escolher o tipo de lesão do paciente, dentre seis possíveis. Ao selecionar um deles, o resultado é exibido na tela, ao selecionar outro, o novo resultado é exibido na tela, não tendo a necessidade do clique no

botão de “Limpar”, apesar de este estar disponível. Conforme detalhado nos requisitos do sistema e visualizado na Figura 43, a *Classificação de Forrest* exibe o nível de sangramento, a frequência dos estigmas e o risco de ressangramento do paciente avaliado.

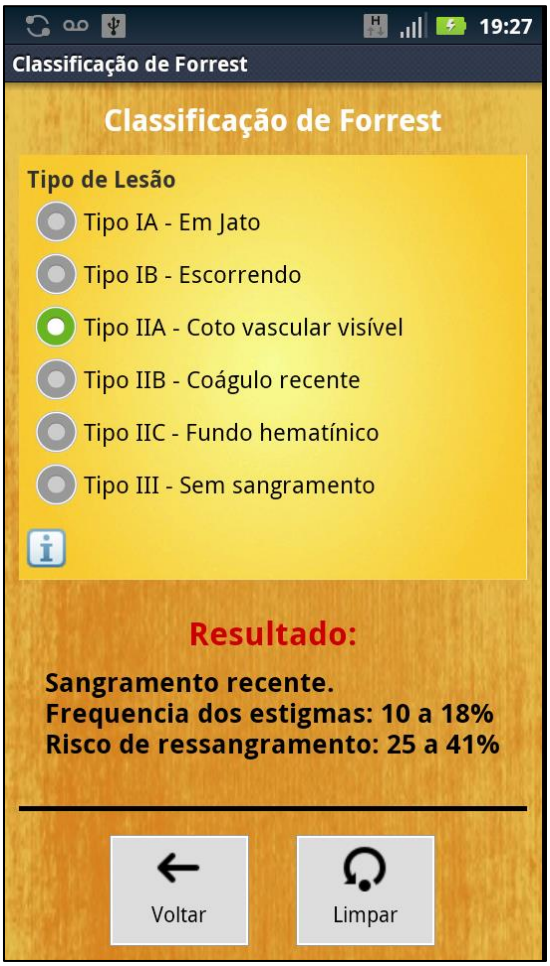

Figura 43 - Interface resultado Classificação de Forrest

Por fim, o último grupo de cálculos é constituído por *Escala de Apgar, Classificação Child Pugh* e *Escala de Coma Glasgow*. Os cálculos deste grupo possuem dois ou mais critérios, onde, para cada um deles, deve ser selecionado somente um item como resposta, sendo que o resultado final é calculado baseado no somatório do valor correspondente a cada uma das respostas selecionadas.

Utilizando como exemplo a *Escala de Apgar*, este cálculo possui as variáveis frequência cardíaca, respiração, tônus muscular, cor da pele e irritabilidade reflexiva. Cada um destas variáveis possui três opções de escolha na sua resposta. Por exemplo, o tônus muscular deve ser classificado em flácido, flexão de pernas e braços ou movimento ativo / boa flexão e a frequência cardíaca deve ser classificada em ausente, menor que cem batimentos por minuto ou maior que cem batimentos por minuto.

Neste grupo, novamente não há o botão calcular, todavia se faz necessária uma validação para verificar se o usuário escolheu um item para cada critério analisado. Portanto, sempre que o usuário seleciona algum item de algum critério é disparado um evento que verifica se todas as variáveis possuem uma resposta. Se a validação for positiva, o resultado é exibido na tela, caso contrário, é exibida uma mensagem de alerta informando ao usuário que deve ser selecionado um item para cada critério, conforme Figura 44.

Ainda na Figura 44, referente ao cálculo da *Escala de Apgar*, é possível observar que não foi inserida resposta para o critério irritabilidade reflexiva. Por este motivo, no lugar do resultado é exibido o alerta indicando a irregularidade. Assim que selecionada uma opção para a variável faltante, a mensagem de alerta será retirada da tela e, no seu lugar, o resultado do cálculo será exibido.

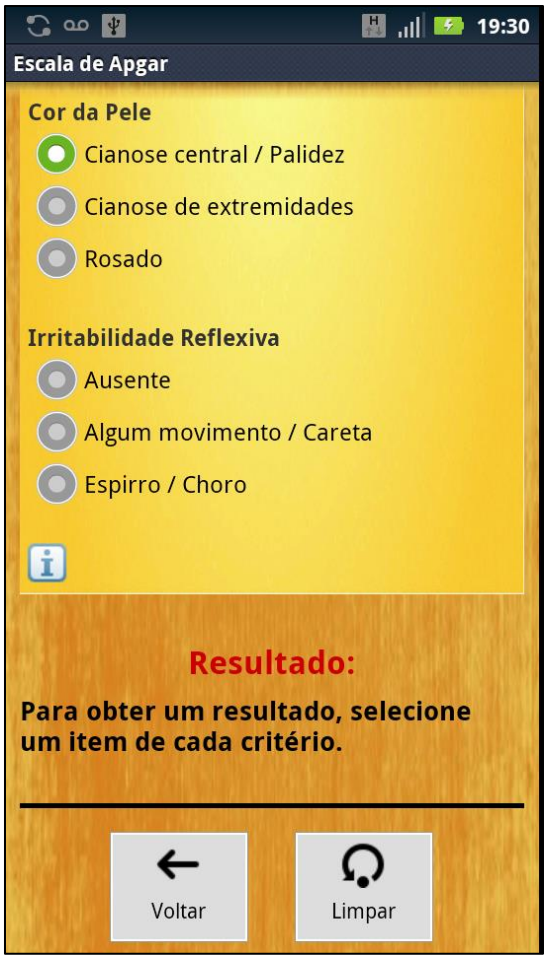

Figura 44 - Interface validação da Escala de Apgar

5. Avaliação dos Resultados

Finalizado o desenvolvimento, fez-se uma pesquisa de satisfação com o objetivo de verificar a aceitação do aplicativo entre os profissionais das áreas de Educação Física, Fisioterapia e Medicina.

Para realização desta atividade, aplicou-se um questionário aos profissionais. Este questionário era composto por seis perguntas, sendo que cada uma delas deveria ser classificada entre grau zero, o pior valor (discordo totalmente, ruim, não) e grau cinco, o melhor valor (concordo totalmente, bom, sim). Os questionamentos foram:

- Classifique a usabilidade do aplicativo;
- Gostaria que o aplicativo disponibilizasse mais funcionalidades (cálculos, classificações e escalas)?
- Você ajudaria na pesquisa por mais funcionalidades?
- Você utilizaria o aplicativo no seu ambiente de trabalho?
- Gostaria que o aplicativo fosse disponibilizado em outros sistemas operacionais (Windows Phone, iOS)?
- Classifique sua satisfação ao conhecer e utilizar o aplicativo.

A tabela 5 exibe as respostas enviadas pelos catorze (14) profissionais entrevistados. Analisando os resultados, é possível observar uma avaliação muito positiva, visto que a grande maioria das perguntas possuem respostas entre os graus quatro e cinco, que são os mais positivos.

Dentre as perguntas, a única que obteve respostas classificadas com grau um foi o item que questiona se os usuários gostariam que o aplicativo

fosse disponibilizado para outros sistemas operacionais. Pode-se concluir que os avaliados que assinalaram não como resposta (resposta um), são usuários do Android que não pretendem migrar para outra plataforma, portanto, não perceberam a necessidade deste desenvolvimento. Já os que assinalaram esta resposta com grau cinco, ou são usuários de outros sistemas operacionais, ou usuários do Android gostariam desta disponibilização para que, em um futuro, possam migrar de plataforma e continuar utilizando o aplicativo.

Tópicos	Quantidade de Respostas				
	1	2	3	4	5
Usabilidade do aplicativo	-	-	-	1	13
Gostaria de mais funcionalidades	-	-	-	-	14
Ajudaria na pesquisa	-	-	1	4	9
Utilizaria o app no trabalho	-	-	-	3	11
Gostaria de utilizar em outros SOs	9	-	-	-	5
Satisfação do usuário	-	-	-	2	12

Tabela 5 - Avaliação do Aplicativo pelos Profissionais

Observando as respostas dos outros questionamentos, ficou bastante evidente a aceitação do aplicativo desenvolvido dentre os profissionais da área da saúde. Um aspecto que chama a atenção positivamente é que, entre os profissionais que avaliaram o *Cálculos Médicos*, onze deles, ou 79%, responderam que utilizariam o aplicativo diariamente no seu ambiente de trabalho, enquanto os outros 21% provavelmente fariam este uso.

Além destas respostas, foi disponibilizado um campo para comentários, onde grande parte dos profissionais relatou uma considerável diminuição no tempo de execução destas atividades, além do aumento da segurança, uma vez que esta automatização minimizou a possibilidade de erro humano na realização dos cálculos.

6. Conclusões e Trabalhos Futuros

O primeiro objetivo específico deste trabalho era oferecer uma fundamentação teórica para que estudantes e profissionais de tecnologia da informação pudessem utilizá-lo como base para futuros estudos ou desenvolvimentos. Os capítulos iniciais foram confeccionados justamente para cumprir este objetivo, oferecendo embasamentos importantes acerca da tecnologia estudada.

Quando se estuda a plataforma Android, alguns tópicos são considerados importantes e de grande importância para o seu entendimento, como *Intent, Activity, Service, BroadcastReceiver, ContentProvider, Dalvik VM* e *AndroidManifest.xml.* Estes elementos foram devidamente discriminados e explicitados, fazendo com que o leitor, dispensando uma pequena quantia de tempo, obtenha um entendimento considerável sobre o sistema operacional Android.

Realizada esta primeira etapa, o segundo escopo principal do trabalho tratava do desenvolvimento de um aplicativo para a área da saúde. Foram realizadas várias etapas para cumprimento desta funcionalidade, como análise e levantamento de requisitos, montagem do ambiente de desenvolvimento, modelagem e, por fim, codificação do software.

É importante ressaltar que, um estudante ou profissional que já tenha algum conhecimento na linguagem de programação Java, terá capacidades para, por meio da leitura deste trabalho, montar o ambiente de desenvolvimento para aplicativos Android e iniciar implementações nesta nova tecnologia.

Durante o desenvolvimento, destacou-se a internacionalização do aplicativo. Esta prática vem crescendo dia após dia, principalmente como consequência direta do processo de globalização e do advento da rede mundial de computadores. A internacionalização foi desenvolvida de forma simples e direta, devido ao suporte claro e conciso oferecido pela plataforma Android.

Além disso, foram focadas etapas imprescindíveis no ciclo de vida de um software. A fase de modelagem foi desenvolvida utilizando a linguagem de especificação UML, por meio da utilização dos diagramas de casos de uso, de classes e de atividades. Estes três diagramas, apesar de não formarem uma modelagem completa, oferecem suporte para uma boa codificação.

Cumprindo o objetivo principal deste trabalho, o aplicativo desenvolvido, nomeado de *Cálculos Médicos*, oferece cálculos, escalas e classificações muito utilizadas nas áreas de Medicina, Educação Física e Fisioterapia. O aplicativo é uma ferramenta poderosa que poderá auxiliar de forma considerável profissionais destas áreas.

Avaliado por profissionais das três áreas abordadas no projeto, o aplicativo *Cálculos Médicos* obteve grande aceitação, tanto pelas funcionalidades oferecidas quanto pela usabilidade. Os profissionais, após conhecimento e uso da ferramenta, relataram a facilidade com que os cálculos passaram a ser feitos, diminuindo consideravelmente o tempo necessário para realização destas tarefas. Estes profissionais também avaliaram um aumento na segurança, uma vez que o aplicativo minimiza a possibilidade de falha humana na execução dos cálculos.

Outro aspecto que demonstra a aceitação do aplicativo dentre os profissionais é o fato de que quase 80% dos avaliadores confirmaram que utilizariam ou irão utilizar o aplicativo durante a execução de suas atividades profissionais, fato este que comprova a importância e a utilidade deste trabalho no futuro.

Por fim, o crescimento acentuado da tecnologia Android, juntamente com a importância que a área da saúde representa para cada cidadão, deram ênfase e inspiração para que o presente trabalho fosse desenvolvido de forma satisfatória, resultando em um produto benéfico para toda a sociedade.

A principal sugestão para trabalho futuro é, utilizando como base o aplicativo desenvolvido, fazer o levantamento de requisitos, a modelagem e o desenvolvimento de novos cálculos, escalas e classificações. Seria de grande valia a disponibilização de novas funcionalidades, tanto nas áreas já exploradas como em novas áreas, citando como exemplo Odontologia, Nutrição, Fonoaudiologia, Enfermagem, Farmácia, dentre outras.

Na avaliação feita, 100% dos profissionais responderam que gostariam que o aplicativo disponibilizasse mais funcionalidades, sendo que destes, grande parte estaria disposta a auxiliar na pesquisa destes novos cálculos.

Para mais, seria interessante a elaboração de um manual em formato tutorial, explicando detalhadamente como funciona a montagem do ambiente de desenvolvimento Android, além da inicialização por meio de um aplicativo modelo, onde seriam introduzidos pontos específicos do desenvolvimento de aplicativos para esta plataforma.

7. Referências Bibliográficas

[1] LECHETA, Ricardo R. **Google Android: aprenda a criar aplicações para dispositivos móveis com o Android SDK**. 2 ed. São Paulo: Novatec Editora, 2010.

[2] ANATEL, “Brasil ultrapassa um celular por habitante”. Disponível em <http://www.anatel.gov.br/Portal/exibirPortalNoticias.do?acao=carregaNoticia&codigo=21613>. Acesso em 21 de Maio de 2013.

[3] REUTERS BRASIL, “Vendas de smartphones no Brasil crescem 78% em 2012--estudo”. Disponível em <http://br.reuters.com/article/internetNews/idBRSPE92D04O20130314>. Acesso em 21 de Maio de 2013.

[4] EXAME ABRIL, “Android tem crescimento de 80% nas vendas”. Disponível em <http://exame.abril.com.br/tecnologia/android/noticias/android-tem-crescimento-de-79-5-nas-vendas>. Acesso em 21 de Maio de 2013.

[5] ABLESON, W. Frank. **Android em ação.** 3 ed. Rio de Janeiro: Elsevier, 2012.

[6] OLHAR DIGITAL, “Windows Phone é o sistema móvel que mais cresce”. Disponível em <http://olhardigital.uol.com.br/produtos/mobilidade/noticias/windows-phone-e-o-sistema-movel-que-mais-cresce,-diz-idc>. Acesso em 20 de Maio de 2013.

[7] SUPERINTERESSANTE, “Conheça a história do Android, o sistema operacional mobile do Google”. Disponível em

<http://super.abril.com.br/galerias-fotos/conheca-historia-android-sistema-operacional-mobile-google-688822.shtml#0>. Acesso em 30 de Maio de 2013.

[8] MOBILIDADE, “Dossiê Android: 3 anos de vida, e muitas datas para serem lembradas (História do Android)”. Disponível em <http://mobilidade.fm/geral/2010/11/historia-do-android/>. Acesso em 30 de Maio de 2013.

[9] REDMOND PIE, “Android Version History Guide: v1.0 To v4.1 Jelly Bean”. Disponível em <http://www.redmondpie.com/android-version-history-guide-v1.0-to-v4.1-jelly-bean-infographic/>. Acesso em 30 de Maio de 2013.

[10] ANDROID, “Android 4.2: A new flavor of Jelly Bean”. Disponível em <http://www.android.com/whatsnew/>. Acesso em 31 de Maio de 2013.

[11] BRAHLER, Stefan. “Analysis of the Android Architecture”. Disponível em <http://os.ibds.kit.edu/downloads/sa_2010_braehler-stefan_android-architecture.pdf>. Acesso em 31 de Maio de 2013.

[12] ANDROID, “App components”. Disponível em <http://developer.android.com/guide/components/index.html>. Acesso em 09 de Junho de 2013.

[13] ANDROID, “API guides: Activities”. Disponível em <http://developer.android.com/guide/components/activities.html>. Acesso em 09 de Junho de 2013.

[14] ANDROID, “API guides: Services”. Disponível em <http://developer.android.com/guide/components/services.html>. Acesso em 12 de Junho de 2013.

[15] VOGEL, Lars. “Android BroadcastReceiver Tutorial”. Disponível em <http://www.vogella.com/articles/AndroidBroadcastReceiver/article.html>. Acesso em 26 de Junto de 2013.

[16] ANDROID, “API guides: Content Providers”. Disponível em <http://developer.android.com/guide/topics/providers/content-providers.html>. Acesso em 26 de Junho de 2013.

[17] ANDROID, “The AndroidManifest.xml File”. Disponível em <http://developer.android.com/guide/topics/manifest/manifest-intro.html>. Acesso em 30 de Junho de 2013.

[18] PRESSMAN, Roger S. **Engenharia de Software: Uma Abordagem Profissional**. 7 ed. São Paulo: Editora Bookman, 2011.

[19] ANDROID, “ADT Plugin”. Disponível em <http://developer.android.com/tools/sdk/eclipse-adt.html>. Acesso em 01 de Outubro de 2013.

[20] SILVA, Ricardo Pereira. **UML: Modelagem Orientada a Objetos**. 1 ed. Florianópolis: Visual Books, 2007.

[21] DE LUCCA, José Eduardo; PRUDÊNCIO, Achilles Colombo; VALOIS, Djali Avelino. **Introdução à Internacionalização e à Localização de Software**. Florianópolis: UFSC, 2004.

Apêndice A – Artigo

Cálculos Médicos: um aplicativo Android para cálculos de indicadores na área da saúde

Henio de Oliveira Bez Junior

Departamento de Informática e Estatística. Universidade Federal de Santa Catarina (UFSC)

Caixa Postal 476 – 88.040-900 – Florianópolis – SC - Brasil

heniobezjr@gmail.com

Abstract. This paper specifies the main topics of author´s TCC. Details the aspects that motivated the author to do this work, beyond the main objective and how works and was specified the Android app. This app, named “Cálculos Médicos”, has many features, like calculation of indicators, classifications and scales, every features to health area, with the objective to help professionals of physical education, physiotherapy and medicine in their daily activities.

Keywords. Android, app, health.

Resumo. Este artigo detalha os principais tópicos do trabalho de conclusão de curso do autor. São explicitados os aspectos que motivaram a elaboração do trabalho, além do detalhamento do objetivo geral e de como funciona e foi desenvolvido o aplicativo Android. Este aplicativo, chamado “Cálculos Médicos”, engloba um conjunto de funcionalidades, como cálculos de indicadores, classificações e escalas, todos para a área da saúde, com o intuito de auxiliar profissionais de Educação Física, Fisioterapia e Medicina nas suas atividades diárias.

Palavras-chave. Android, aplicativo, saúde.

1. Introdução e Motivação

Fato consumado nos dias de hoje, o uso de dispositivos móveis, como smartphones e tablets, cresce gradualmente, tornando-se objeto quase que imensurável às pessoas.

Acompanhando este crescimento, crescem também os sistemas operacionais móveis, principalmente o Android. Durante os primeiros quatro meses de 2013, Android teve um aumento de quase 80% nas vendas, comparado com o mesmo período do ano anterior. Com isso, contabilizando somente as vendas de 2013, o Android é parte integrante de 75% dos novos aparelhos [1].

O crescimento na quantidade de aplicações disponíveis também cresce de forma exponencial. Segundo dados da Wikipédia [2], em Julho de 2011 havia aproximadamente 250 mil aplicativos disponíveis aos usuários. Em dois anos este número cresceu assustadoramente, ultrapassando a marca de um milhão de aplicativos.

A área da saúde é altamente valorizada e de suma importância para a população em geral, todavia se percebeu uma carência em aplicativos para cálculos de indicadores destinados a esta área. Partindo deste pressuposto, foi desenvolvido um aplicativo que poderá ser comumente utilizado por profissionais da saúde para automatização de tarefas diárias, auxiliando-os de forma positiva no atendimento a pacientes.

2. Plataforma Android

Mais do que um sistema operacional, Android é um ambiente de software que possui um sistema operacional baseado em Linux. Além disso, Android engloba outras qualidades, como uma rica Interface de Usuário, aplicativos de usuário, bibliotecas de código, frameworks de aplicativo e suporte a multimídia [3].

Muitas vezes o Android é confundido com uma plataforma de hardware, contudo esta classificação é errônea, pois Android é somente software, um ambiente onde são executados aplicativos nativos e criados por desenvolvedores, indistintamente, rodando em uma máquina virtual, chamada de Dalvik, suportada por um sistema operacional baseado no kernel 2.6 do Linux.

A utilização do kernel baseado em Linux para ser o coração do Android é uma característica muito positiva, pois todo gerenciamento de processos e memória é feito de forma transparente, isto é, desenvolvedores não têm a necessidade de se preocupar com estes controles. Além disso, esta característica permite a execução de inúmeros aplicativos de forma concorrente, um em primeiro e outros em segundo plano, possibilitando, por exemplo, que um aplicativo seja executado enquanto se recebe uma chamada de voz [4].

Uma das características que torna o Android tão popular é a utilização da linguagem de programação Java, característica esta que, por muitos, é considerada o fator crítico de sucesso da plataforma. Além desta, as duas outras características responsáveis pela grande popularização do Android são utilização do kernel Linux e atualização constante de versões, possuindo média de duas atualizações por ano, número alto para sistemas operacionais.

Sobre o processo de desenvolvimento e compilação de aplicativos, este ocorre de maneira simples e direta. As aplicações Android são desenvolvidas em Java, utilizando-se todos os recursos desta linguagem. Como uma Java VM padrão, os arquivos *.java* são compilados, gerando os arquivos bytecodes *.class* correspondentes. Após isso, os arquivos *.class* são convertidos para extensões *.dex*, que correspondem à aplicação do Android compilada. Para se obter o arquivo final que será instalado nos dispositivos móveis, basta-se compactar os arquivos *Dalvik Executable* e outros arquivos necessários, como sons e imagens, em um único arquivo *Android Package File* (*.apk*) [4].

3. Aplicativo *Cálculos Médicos*

O objetivo principal do trabalho de conclusão de curso [5] foi desenvolver um aplicativo com enfoque na área da saúde. O aplicativo consiste em um conjunto de funcionalidades, possuindo cálculos de indicadores, classificações e escalas, com a incumbência de auxiliar profissionais de algumas áreas da saúde nas suas atividades diárias. Estas atividades, de grande importância, são executadas inúmeras vezes. O aplicativo *Cálculos Médicos* foi desenvolvido, justamente, para dar suporte a estes profissionais, visando uma maior segurança e facilidade na execução das suas tarefas.

Para desenvolvimento do aplicativo, na etapa de levantamento de requisitos, foram entrevistados profissionais de diversas áreas de atuação. Nas entrevistas fez-se uma apresentação da proposta do trabalho, explicitando detalhes sobre o funcionamento geral do aplicativo a ser desenvolvido. Após isto, os profissionais foram questionados se, dentro da sua profissão, havia um ou mais procedimentos que poderiam ser automatizados de acordo com o modelo de funcionamento que se esperava para o aplicativo.

Dentre os contatados, retornaram positivamente os profissionais das áreas de Educação Física, Fisioterapia e Medicina. Com isso, fez-se uma coleta de informações a respeito das funcionalidades que seriam incorporadas no aplicativo.

As funcionalidades incluídas foram: Anion Gap, Classificação de Forrest, Classificação de Killip, Critério Ranson, Escala de Apgar, Escala de Ashworth Modificada, Escala de Child Pugh, Escala Glasgow de Coma, Escala MELD, Escala Ramsay, Fórmula de Cockroft-Gault, Fórmula de Parkland, Índice de Massa Corporal (IMC), LDL – Cálculo do Colesterol, Margem de Gordura Corporal, Massa Óssea Estimada, Risco de Pneumonia CURB-65, Teste de Cooper (Teste dos 12 Minutos) e Zona Aeróbica de Treinamento.

3.1. Funcionamento do Aplicativo

Ao acessar a aplicação, o usuário poderá visualizar informações acerca do aplicativo ou selecionar a opção de cálculos. Selecionando esta última opção, deverá ser escolhida a categoria desejada, que poderá ser Educação Física, Fisioterapia ou Medicina. Dependendo da área selecionada, serão exibidos os cálculos correspondentes.

Se, na interface de categorias (Figura 1), o usuário clicar no botão *Educação Física*, serão listados os cálculos que pertencem à área de Educação Física. Caso seja clicado no botão *Fisioterapia,* serão exibidos os cálculos da área de Fisioterapia e, se clicado no botão *Medicina*, o usuário visualizará as funcionalidades referentes à área de Medicina.

O usuário também terá a opção de listar todos os cálculos do sistema. Feita esta escolha, serão exibidos na tela todos os cálculos, classificações e escalas presentes no aplicativo. Na Figura 1, observa-se a interface onde o usuário selecionará a categoria desejada.

Figura 45 - Interface para o usuário selecionar a categoria

Neste ponto, o usuário terá em mãos a lista dos cálculos disponíveis, de acordo com a categoria escolhida. A Figura 2 exibe uma lista com todos os

cálculos do sistema. Esta é uma lista de “correr”, onde o usuário deslizará a tela para baixo ou para cima, percorrendo os cálculos presentes na lista.

Figura 46 - Lista exibindo cálculos presentes no aplicativo

Nesta lista, o usuário deverá clicar na funcionalidade desejada. Com isso, será exibida a interface para a inserção de dados. Por exemplo, na Figura 3, é exibido o cálculo da *Escala Meld*. Na parte esquerda da imagem, a interface limpa, na direita, com inserção de dados e exibição de resultados.

Figura 47 - Escala Meld limpa e com inserção de dados e resultados

Ainda na Figura 3, observa-se um ícone de informações. Todas as interfaces de cálculo possuem este ícone (), que, quando clicado, abre um *popup* que contém a descrição e a fórmula utilizada para o cálculo em questão. Na Figura 4, lado direito, é possível visualizar as informações referentes à *Escala Meld*. Também na figura 4, imagem da esquerda, observa-se atuação do validador. Todas os cálculos possuem validadores que impossibilitam a inserção de valores errados por parte do usuário.

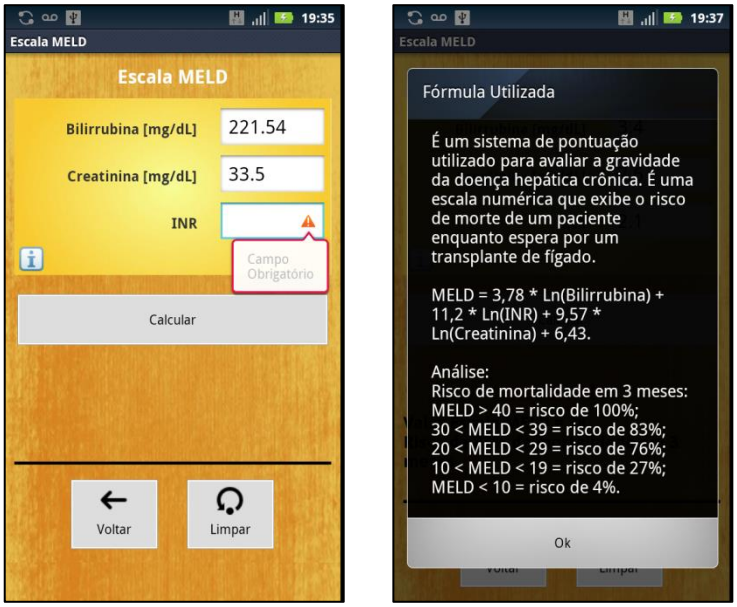

Figura 48 - Escala Meld: validador e informações

4. Conclusões e Trabalhos Futuros

Finalizado o desenvolvimento, fez-se uma pesquisa de satisfação com o objetivo de verificar a aceitação do aplicativo entre os profissionais das áreas de Educação Física, Fisioterapia e Medicina.

Analisando o retorno dos profissionais, observou-se respostas bastante positivas. Houve uma grande aceitação, refletida na resposta de que quase 80% dos avaliadores com certeza utilizariam o aplicativo no seu ambiente de trabalho.

Portanto, o aplicativo *Cálculos Médicos* obteve grande reconhecimento, tanto pelas funcionalidades oferecidas quanto pela usabilidade. Os profissionais, após conhecimento e uso da ferramenta, relataram a facilidade

com que os cálculos passaram a ser feitos, diminuindo consideravelmente o tempo necessário para realização destas tarefas. Estes profissionais também avaliaram um aumento na segurança, uma vez que o aplicativo impossibilita falha humana na execução dos cálculos.

A principal sugestão para trabalho futuro é, utilizando como base o aplicativo desenvolvido, fazer o levantamento de requisitos, a modelagem e o desenvolvimento de novos cálculos, escalas e classificações. Seria de grande valia a disponibilização de novas funcionalidades, tanto nas áreas já exploradas como em novas áreas, citando como exemplo Odontologia, Nutrição, Fonoaudiologia, Enfermagem, Farmácia, dentre outras.

5. Referências Bibliográficas

[1] EXAME ABRIL, “Android tem crescimento de 80% nas vendas”. Disponível em <http://exame.abril.com.br/tecnologia/android/noticias/android-tem-crescimento-de-79-5-nas-vendas>. Acesso em 21 de Maio de 2013.

[2] WIKIPEDIA, “Google Play”. Disponivel em <http://pt.wikipedia.org/wiki/Google_Play>. Acesso em 05 de Novembro de 2013.

[3] ABLESON, W. Frank. **Android em ação.** 3 ed. Rio de Janeiro: Elsevier, 2012.

[4] LECHETA, Ricardo R. **Google Android: aprenda a criar aplicações para dispositivos móveis com o Android SDK**. 2 ed. São Paulo: Novatec Editora, 2010.

[5] JUNIOR, Henio de Oliveira Bez. (2013), **Cálculos Médicos: um aplicativo Android para cálculos de indicadores na área da saúde**. Universidade Federal de Santa Catarina (UFSC).

Apêndice B – Histórico do Android

Tudo começou no ano de 2003 com a fundação da Android Inc., uma empresa de tecnologia totalmente independente, localizada nos Estados Unidos, estado da Califórnia, que desenvolvia sistemas operacionais para celulares. O principal fundador desta organização foi Andy Rubin. Apenas dois anos após sua fundação, em 2005, a empresa foi comprada pelo Google, com o intuito de desenvolver um sistema operacional para plataformas móveis baseada em Linux [7]. A partir desse momento, espalharam-se notícias pelas principais redes do mundo de que o Google estaria entrando no mercado de dispositivos móveis.

Em 2007, acabando com todos os rumores, o Google apresentou o Android ao mundo. Juntamente com o lançamento do Android como plataforma, foi anunciada a criação da OHA, acrônimo de *Open Handset Alliance*, um conjunto de trinta e quatro empresas com o objetivo de enraizar o código aberto na telefonia móvel. Dentro desta aliança de organizações destacam-se, além do Google, empresas como HTC, Intel, LG, Motorola, Samsung, Sprint Nextel, Telefonica e T-Mobile [8].

Por fim, após um emaranhado de notícias, em Outubro de 2008, foi anunciado o primeiro aparelho celular com Android. Segundo Lecheta (2010, p27) [1]

> O T-Mobile G1 desenvolvido pela HTC foi o primeiro celular lançado com a plataforma do Android e, como esperado, causou um grande agito no mercado. A notícia de seu lançamento causou um grande impacto e superou as

> expectativas de vendas da HTC, e mesmo antes de seu lançamento já havia sido esgotado todo o estoque para os pedidos de pré-venda.

Além de tela sensível ao toque e teclado Qwerty, o HTC G1 possuía recursos exclusivamente disponibilizados pelo Android, funcionalidades consideradas bastante avançadas para a época, como janela de notificações, central para aquisição de aplicativos e integração com o Gmail.

A primeira versão comercial do sistema operacional, o Android 1.0, foi lançado juntamente com o primeiro aparelho, em 2008. A primeira atualização, Android 1.1, foi disponibilizada já no início de 2009. Esta versão já possuía algumas funcionalidades inclusas, hoje consideradas básicas, como alarme, navegador, calculadora, câmera, mapas, e-mail, mensagens e músicas [8].

A partir da terceira versão do Android, em Abril de 2009, todas foram batizadas com nomes de sobremesas ou bolos, em inglês, seguindo uma ordem alfabética. Desde então, foram lançadas nove versões do sistema operacional, com média de uma nova atualização a cada quatro meses, aproximadamente. Abaixo, seguem características e novidades de cada uma das versões da plataforma Android.

Como já citado acima, a terceira versão do Android foi lançada em Abril de 2009, com base no kernel Linux 2.6.27. Batizada de *Cupcake*, teve como principais novidades a correção automática de textos e suporte a *widgets*, que são pequenas aplicações ou programas com o objetivo de permitir fácil acesso a funções frequentemente utilizadas. Além disso, surgiram ou foram aperfeiçoadas outras funcionalidades, como Bluetooth, gravação e reprodução de vídeos, possibilidade de copiar e colar textos em navegadores web, trocas

animadas de telas, carregamento de vídeos no YouTube e de fotos no Picasa [9]. Abaixo segue tela inicial do Android 1.5, com ícones de alguns aplicativos e *widget* de um relógio.

Figura 49 - Tela Inicial Android 1.5.

Em Setembro do mesmo ano foi lançada a versão 1.6 do Android, nomeada de *Donut*, em português, “rosquinha”. As principais novidades desta produção foram busca on-line automática no Google, a partir da página inicial do aparelho e atualização do consumo da bateria, que passou a ser mostrada em forma de porcentagem de uso por aplicativo, isto é, quanto cada aplicativo em execução estava consumindo de bateria em um determinado momento. Outras novidades marcantes foram integração de câmera com galeria de fotos e vídeos, suporte à resolução de tela WVGA, gravador de voz e novas APIs.

Apenas um mês depois, em Outubro de 2009, foi disponibilizada a versão 2.0 da plataforma, com codinome *Éclair*, tendo como característica mais marcante a possibilidade de um usuário associar múltiplas contas Google a um único aparelho. Para mais, várias outras utilidades foram oferecidas ou

atualizadas, como suporte ao *Bluetooth* 2.1, busca rápida por mensagens de texto já existentes, papéis de parede e proteções de tela animados, novas funcionalidades na câmera, suporte ao HTML5 nos navegadores, otimizações no hardware, busca rápida por contatos e suporte a múltiplos tamanhos e resoluções de tela.

Já em 2010, *Froyo*, a versão 2.2 do Android, foi lançada. Os principais surgimentos foram introdução da tecnologia Adobe Flash nos navegadores, possibilidade de transformar o aparelho em um roteador *wi-fi* usando 3G e criação do C2DM, um serviço que possibilita envio de mensagens de um computador para um celular, previamente cadastrado, por meio da utilização da infraestrutura de nuvens do Google. Esta versão também foi marcada por outras novidades, como atalhos dedicados na tela inicial do aparelho, reconhecimento de comandos de voz para busca de contatos e de informações na internet, opção de salvar aplicativos em cartões de memória e teclados de múltiplos idiomas [9].

Ainda em 2010, todavia em Dezembro, o Android 2.3 foi lançado, sendo nomeado *Gingerbread* e tendo como base o Linux Kernel 2.6.35. A principal novidade desta nova versão foi o NFC – *Near Field Communication*, uma tecnologia que permite trocar dados entre dispositivos Android compatíveis. Outras novas *features* desta versão foram melhor gerenciamento sobre aplicativos, possibilidade de acesso a várias câmeras, novos efeitos de áudio, novo gerenciador de download, tela multitoque e suporte ao VoIP (voz sobre IP), permitindo uso de aplicativos para chamadas de voz, como, por exemplo, Skype e Google Talk.

A versão 3.0, chamada de *Honeycomb*, do português favo de mel, foi considerada, por muitos, um divisor de águas. Lançada em Fevereiro de 2011, foram feitas reformulações drásticas no sistema operacional, visando uma maior compatibilidade com tablets, visto que a popularização destes novos dispositivos estava crescendo dia após dia. Esta atualização, baseado no kernel Linux 2.6.36, foi a primeira feita exclusivamente para tablets, tendo como principais novidades uma repaginada completa nas interfaces da plataforma.

Figura 50 – Os doces do Android

Em Outubro de 2011 foi lançada a versão 4.0 do Android, *Ice Cream Sandwich*. As principais alterações disponibilizadas nesta release foram aperfeiçoamento no compartilhamento de arquivos, reconhecimento facial para desbloqueio do dispositivo, calendário unificado, navegador Google Chrome, widgets redimensionáveis e análise detalhada sobre uso de dados. Após a versão anterior, destinada somente para tablets, o Google tinha a incumbência de desenvolver um sistema operacional compatível concomitantemente para ambos os aparelhos, smartphones e tablets, pois manter duas versões de um sistema operacional não seria viável. *Ice Cream Sandwich* veio justamente para consolidar a compatibilidade da plataforma com dispositivos de tamanho de telas variados [9].

Por fim, já em 2012, no mês de Julho, foi disponibilizada a versão 4.1 da plataforma Android, nomeada *Jelly Bean*. As principais novidades foram busca inteligente por aplicativos, otimização no toque na tela, suporte a diversas linguagens, widgets para bloqueio de tela e suporte a múltiplos usuários. Outra característica importante foi a substituição da tecnologia C2DM, lançada na versão 2.2, por outra mais segura e veloz, denominada *Google Cloud Messaging* (GCM). Foi lançada, também em 2012, uma melhoria para a versão 4.1, denominada como um novo sabor para *Jelly Bean*, que oferece pequenas alterações em relação à anterior. Nesta versão, 4.2, disponibilizaram-se apenas aprimoramentos na interface, processamento e segurança do sistema operacional [10].

Em Julho de 2013, foi lançada uma nova atualização do Android, destinada principalmente aos aparelhos da linha *Nexus*. Esta é a versão 4.3 do sistema operacional Android.

Abaixo, segue tabela com o histórico das versões da plataforma Android.

Versão	Nome	Lançamento
1.0	---	Outubro de 2008
1.5	Cupcake	Abril de 2009
1.6	Donut	Setembro de 2009
2.0	Éclair	Outubro de 2009
2.2	Froyo	Maio de 2010
2.3	Gingerbread	Dezembro de 2010
3.0	Honeycomb	Fevereiro de 2011
4.0	Ice Cream Sandwich	Outubro de 2011
4.1	Jelly Bean	Julho de 2012
4.2	Jelly Bean (a new flavor)	Dezembro de 2012
4.3	Jelly Bean	Julho de 2013

Tabela 6 - Histórico das Versões do Android

Apêndice C – Código Fonte

```
/**
 * enum que representa todos os cálculos disponíveis no sistema.
 * Cada cálculo possui um nome e uma categoria.
 *
 * @author Henio
 * @since 2013
 */
public enum Calculo {

	ANION_GAP("Anion Gap","medicina"),
	CLASSIF_FORREST("Classificação Forrest","medicina"),
	CLASSIF_KILLIP("Classificação Killip","medicina"),
	CRITERIO_RANSON("Critério Ranson","medicina"),
	ESCALA_APGAR("Escala Apgar","medicina"),
	ESCALA_ASHWORTH("Escala Ashworth Modificada","fisioterapia"),
	ESCALA_CHILD("Escala Child Pugh","medicina"),
	ESCALA_GLASGOW("Escala Glasgow de coma","medicina,fisioterapia"),
	ESCALA_MELD("Escala Meld","medicina"),
	ESCALA_RAMSAY("Escala Ramsay","fisioterapia"),
	FORMULA_COCKROFT("Fórmula Cockroft-gault","medicina"),
	FORMULA_PARKLAND("Fórmula Parkland","medicina"),
	IMC("IMC (Índice Massa Corporal)","educacaofisica"),
	LDL("LDL (Colesterol)","medicina"),
	MARGEM_GORDURA("Margem Gordura Corporal","educacaofisica"),
	MASSA_OSSEA("Massa Óssea Estimada","educacaofisica"),
	RISCO_CURB("Risco Pneumonia CURB-65","medicina"),
	TESTE_COOPER("Teste Cooper (12 minutos)","educacaofisica"),
	ZONA_AEROBICA("Zona Aeróbica Treinamento","educacaofisica");

	private String nome;
	private String categoria;

	Calculo(String nome, String categoria){
		this.nome = nome;
		this.categoria = categoria;
	}

	public String getNome() {
		return nome;
	}

	public String getCategoria() {
		return categoria;
	}
}

/**
 * Classe utilizada como controle para manipular os cálculos disponíveis na aplicação
 *
 * @author Henio
 * @since 2013
 */
public class CalculosDisponiveis {

	/**
	 * Retorna uma lista com o nome de todos os cálculos disponíveis na aplicação.
	 *
	 * @return calculosDisponiveis
	 */
	public static List<String> getCalculosDisponiveis() {
		List<String> calculosDisponiveis = new ArrayList<String>();

		Calculo[] calculos = Calculo.class.getEnumConstants();
		for (Calculo calculo : calculos) {
```

```
                        calculosDisponiveis.add(calculo.getNome());
                }

                return calculosDisponiveis;
        }

        /**
         * Retorna uma lista com o nome dos cálculos da categoria educacaofisica disponíveis na aplicação.
         *
         * @return calculosEducacaoFisica
         */
        public static List<String> getCalculosEducacaoFisica() {
                List<String> calculosEducacaoFisica = new ArrayList<String>();

                Calculo[] calculos = Calculo.class.getEnumConstants();
                for (Calculo calculo : calculos) {
                        if(calculo.getCategoria().contains("educacaofisica")){
                                calculosEducacaoFisica.add(calculo.getNome());
                        }
                }

                return calculosEducacaoFisica;
        }

        /**
         * Retorna uma lista com o nome dos cálculos da categoria fisioterapia disponíveis na aplicação.
         *
         * @return calculosFisioterapia
         */
        public static List<String> getCalculosFisioterapia() {
                List<String> calculosFisioterapia = new ArrayList<String>();

                Calculo[] calculos = Calculo.class.getEnumConstants();
                for (Calculo calculo : calculos) {
                        if(calculo.getCategoria().contains("fisioterapia")){
                                calculosFisioterapia.add(calculo.getNome());
                        }
                }

                return calculosFisioterapia;
        }

        /**
         * Retorna uma lista com o nome dos cálculos da categoria medicina disponíveis na aplicação.
         *
         * @return calculosMedicina
         */
        public static List<String> getCalculosMedicina() {
                List<String> calculosMedicina = new ArrayList<String>();

                Calculo[] calculos = Calculo.class.getEnumConstants();
                for (Calculo calculo : calculos) {
                        if(calculo.getCategoria().contains("medicina")){
                                calculosMedicina.add(calculo.getNome());
                        }
                }

                return calculosMedicina;
        }
}

/**
 * Activity que representa a interface exibida ao abrir a aplicação, antes de exibir a tela inicial.
 *
 * @author HenioJR
 * @since 2013
 *
```

```
 */
public class CarregarInicial extends Activity {

	private ProgressBar barraProgresso;
	private int statusBarraProgresso;

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

		statusBarraProgresso = 0;
		barraProgresso = (ProgressBar) findViewById(R.id.progressBar);
		barraProgresso.setProgress(0);
		barraProgresso.setMax(100);

		Handler handler = new Handler();
		handler.postDelayed(new CarregarInicialHandler(), 1000);
	}

	/**
	 * Classe responsável por iniciar a Activity principal do aplicativo.
	 */
	class CarregarInicialHandler implements Runnable{
		@Override
		public void run() {
			while(statusBarraProgresso < 100){
				try {
					Thread.sleep(75);
					statusBarraProgresso += 5;
					barraProgresso.setProgress(statusBarraProgresso);

				} catch (InterruptedException e) {
					e.printStackTrace();
				}
			}
			startActivity(new Intent(getApplication(), TelaInicial.class));
			CarregarInicial.this.finish();
		}
	}
}

public class InformacoesApp extends Activity {

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

		Button botaoVoltar = (Button) findViewById(R.id.botaoVoltarInfo);
		botaoVoltar.setOnClickListener(new View.OnClickListener() {
			@Override
			public void onClick(View v) {
				startActivity(new Intent(getApplication(), TelaInicial.class));
			}
		});
	}
}

public class ListaCalculos extends Activity {

	static String categoria;

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

```
		ListView listaCalculos = (ListView) findViewById(R.id.listaCalculos);

		ArrayAdapter<String> adapter = null;

		if("educacaofisica".equals(categoria)){
			adapter = new ArrayAdapter<String>(this, android.R.layout.simple_list_item_1,
CalculosDisponiveis.getCalculosEducacaoFisica());
		} else if("fisioterapia".equals(categoria)){
			adapter = new ArrayAdapter<String>(this, android.R.layout.simple_list_item_1,
CalculosDisponiveis.getCalculosFisioterapia());
		} else if("medicina".equals(categoria)){
			adapter = new ArrayAdapter<String>(this, android.R.layout.simple_list_item_1,
CalculosDisponiveis.getCalculosMedicina());
		} else {
			// todos
			adapter = new ArrayAdapter<String>(this, android.R.layout.simple_list_item_1,
CalculosDisponiveis.getCalculosDisponiveis());
		}

		listaCalculos.setAdapter(adapter);
		listaCalculos.setOnItemClickListener(new AdapterView.OnItemClickListener() {
			@Override
			public void onItemClick(AdapterView<?> parent, final View view, int position, long id)
{
				final String calculoSelecionado = (String) parent.getItemAtPosition(position);
			ListaCalculos.this.selecionaActivityCorreta(calculoSelecionado);
			}
		});

		Button botaoVoltar = (Button) findViewById(R.id.botaoVoltarCategorias);
		botaoVoltar.setOnClickListener(new OnClickListener() {
			@Override
			public void onClick(View v) {
				startActivity(new Intent(getApplication(), SelecionarCalculos.class));
			}
		});
	}

	public static void setCategoria(String categoria) {
		ListaCalculos.categoria = categoria;
	}

	/**
	 * Verifica qual item da lista de cálculos foi selecionado e inicia a Activity correta, criando um Intent
explícito para a Activity.
	 *
	 * @param calculoSelecionado
	 */
	public void selecionaActivityCorreta(String calculoSelecionado){
		if(Calculo.ANION_GAP.getNome().equals(calculoSelecionado)){
			startActivity(new Intent(getApplication(), CalculaAnionGap.class));
		}
		else if (Calculo.RISCO_CURB.getNome().equals(calculoSelecionado)){
			startActivity(new Intent(getApplication(), CalculaRiscoPneumoniaCurb.class));
		}
		else if (Calculo.FORMULA_PARKLAND.getNome().equals(calculoSelecionado)){
			startActivity(new Intent(getApplication(), CalculaParkland.class));
		}
		else if (Calculo.ESCALA_MELD.getNome().equals(calculoSelecionado)){
			startActivity(new Intent(getApplication(), CalculaMeld.class));
		}
		else if (Calculo.FORMULA_COCKROFT.getNome().equals(calculoSelecionado)){
			startActivity(new Intent(getApplication(), CalculaCockroftGault.class));
		}
		else if (Calculo.ESCALA_GLASGOW.getNome().equals(calculoSelecionado)){
			startActivity(new Intent(getApplication(), CalculaGlasgow.class));
		}
```

```
                else if (Calculo.ESCALA_APGAR.getNome().equals(calculoSelecionado)){
                        startActivity(new Intent(getApplication(), CalculaApgar.class));
                }
                else if (Calculo.ESCALA_CHILD.getNome().equals(calculoSelecionado)){
                        startActivity(new Intent(getApplication(), CalculaChildPugh.class));
                }
                else if (Calculo.IMC.getNome().equals(calculoSelecionado)){
                        startActivity(new Intent(getApplication(), CalculaIMC.class));
                }
                else if (Calculo.LDL.getNome().equals(calculoSelecionado)){
                        startActivity(new Intent(getApplication(), CalculaLDL.class));
                }
                else if (Calculo.ESCALA_RAMSAY.getNome().equals(calculoSelecionado)){
                        startActivity(new Intent(getApplication(), CalculaRamsay.class));
                }
                else if (Calculo.ESCALA_ASHWORTH.getNome().equals(calculoSelecionado)){
                        startActivity(new Intent(getApplication(), CalculaAshworth.class));
                }
                else if (Calculo.CLASSIF_FORREST.getNome().equals(calculoSelecionado)){
                        startActivity(new Intent(getApplication(), CalculaForrest.class));
                }
                else if (Calculo.CLASSIF_KILLIP.getNome().equals(calculoSelecionado)){
                        startActivity(new Intent(getApplication(), CalculaKillip.class));
                }
                else if (Calculo.CRITERIO_RANSON.getNome().equals(calculoSelecionado)){
                        startActivity(new Intent(getApplication(), CalculaCriterioRanson.class));
                }
                else if (Calculo.ZONA_AEROBICA.getNome().equals(calculoSelecionado)){
                        startActivity(new Intent(getApplication(), CalculaZonasAlvo.class));
                }
                else if (Calculo.TESTE_COOPER.getNome().equals(calculoSelecionado)){
                        startActivity(new Intent(getApplication(), CalculaTesteCooper.class));
                }
                else if (Calculo.MARGEM_GORDURA.getNome().equals(calculoSelecionado)){
                        startActivity(new Intent(getApplication(), CalculaGorduraCorporal.class));
                }
                else if (Calculo.MASSA_OSSEA.getNome().equals(calculoSelecionado)){
                        startActivity(new Intent(getApplication(), CalculaMassaOssea.class));
                }
        }
}

public class SelecionarCalculos extends Activity {

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

                Button botaoTodos = (Button) findViewById(R.id.botaoTodosCalculos);
                botaoTodos.setOnClickListener(new OnClickListener() {
                        @Override
                        public void onClick(View v) {
                                ListaCalculos.setCategoria("todos");
                                startActivity(new Intent(getApplication(), ListaCalculos.class));
                        }
                });

                Button botaoCategoriaEducacaoFisica = (Button)
findViewById(R.id.botaoCategoriaEducacaoFisica);
                botaoCategoriaEducacaoFisica.setOnClickListener(new OnClickListener() {
                        @Override
                        public void onClick(View v) {
                                ListaCalculos.setCategoria("educacaofisica");
                                startActivity(new Intent(getApplication(), ListaCalculos.class));
                        }
                });
```

```
		Button botaoCategoriaFisioterapia = (Button) findViewById(R.id.botaoCategoriaFisioterapia);
		botaoCategoriaFisioterapia.setOnClickListener(new OnClickListener() {
			@Override
			public void onClick(View v) {
				ListaCalculos.setCategoria("fisioterapia");
				startActivity(new Intent(getApplication(), ListaCalculos.class));
			}
		});

		Button botaoCategoriaMedicina = (Button) findViewById(R.id.botaoCategoriaMedicina);
		botaoCategoriaMedicina.setOnClickListener(new OnClickListener() {
			@Override
			public void onClick(View v) {
				ListaCalculos.setCategoria("medicina");
				startActivity(new Intent(getApplication(), ListaCalculos.class));
			}
		});

		Button botaoVoltar = (Button) findViewById(R.id.botaoVoltarCategorias);
		botaoVoltar.setOnClickListener(new OnClickListener() {
			@Override
			public void onClick(View v) {
				startActivity(new Intent(getApplication(), TelaInicial.class));
			}
		});
	}
}

/**
 * Tela inicial do sistema, que exibirá uma lista com todos os cálculos possíveis.
 *
 * @author Henio
 * @since 2013
 */
public class TelaInicial extends Activity {

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

		final Button botaoInformacoes = (Button) findViewById(R.id.botaoInformacoes);
		botaoInformacoes.setOnClickListener(new OnClickListener() {
			@Override
			public void onClick(View v) {
				startActivity(new Intent(getApplication(), InformacoesApp.class));
			}
		});

		final Button botaoCalculos = (Button) findViewById(R.id.botaoCalculos);
		botaoCalculos.setOnClickListener(new OnClickListener() {
			@Override
			public void onClick(View v) {
				startActivity(new Intent(getApplication(), SelecionarCalculos.class));
			}
		});
	}
}

public class ActivityGeral extends Activity {

	private AlertDialog popupDescricao;
	public TextView labelResultado, valorFinal;

	@Override
	protected void onResume(){
```

```
		super.onResume();
		/*
		 * Esconde teclado virtual. Aqui, não funciona chamando o método esconderTecladoVirtual
		 */

	getWindow().setSoftInputMode(WindowManager.LayoutParams.SOFT_INPUT_STATE_ALWAYS_HID
DEN);
	}

	/**
	 * Método responsável por esconder o teclado virtual da tela
	 *
	 * @param activity
	 * @param token
	 */
	public void esconderTecladoVirtual(IBinder token){
		InputMethodManager imm = (InputMethodManager)
this.getSystemService(Context.INPUT_METHOD_SERVICE);
		imm.hideSoftInputFromWindow(token, 0);
	}

	public void exibeResultado(){
		this.labelResultado.setVisibility(View.VISIBLE);
		this.valorFinal.setVisibility(View.VISIBLE);
	}

	public void escondeResultado(){
		this.labelResultado.setVisibility(View.INVISIBLE);
		this.valorFinal.setVisibility(View.INVISIBLE);
	}

	/**
	 * Método responsável por criar e exibir um popup com a descrição do cálculo em questão.
	 *
	 * @param descricao
	 */
	public void mostraPopupDescricao(String descricao){
		AlertDialog.Builder builder = new AlertDialog.Builder(this);
		builder.setTitle(getString(R.string.descricao));
		builder.setMessage(descricao);
		builder.setPositiveButton("Ok", new DialogInterface.OnClickListener() {
			@Override
			public void onClick(DialogInterface dialog, int which) {
				dialog.dismiss();
			}
		});
		this.popupDescricao = builder.create();
		this.popupDescricao.show();
	}
}

public class CalculaAnionGap extends ActivityGeral {

	private EditText campoNivelSodio, campoNivelCloro, campoNivelBicarbonato;

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

		System.out.println("Criando Activity: CalculaAnionGap");

		this.campoNivelSodio = (EditText) findViewById(R.id.campoNivelSodio);
		this.campoNivelCloro = (EditText) findViewById(R.id.campoNivelCloro);
		this.campoNivelBicarbonato = (EditText) findViewById(R.id.campoNivelBicarbonato);
		this.labelResultado = (TextView) findViewById(R.id.labelResultado);
		this.valorFinal = (TextView) findViewById(R.id.valorFinal);
```

```
		Button botaoCalcular = (Button) findViewById(R.id.botaoCalcular);
		botaoCalcular.setOnClickListener(new View.OnClickListener() {
			@Override
			public void onClick(View v) {

	CalculaAnionGap.this.esconderTecladoVirtual(CalculaAnionGap.this.getCurrentFocus().getWindowToken
());
				if(CalculaAnionGap.this.validaDados()){
					CalculaAnionGap.this.calcularAnionGap();
				}
			}
		});

		Button botaoVoltar = (Button) findViewById(R.id.botaoVoltar);
		botaoVoltar.setOnClickListener(new View.OnClickListener() {
			@Override
			public void onClick(View v) {
				startActivity(new Intent(getApplication(), ListaCalculos.class));
			}
		});

		Button botaoLimpar = (Button) findViewById(R.id.botaoLimpar);
		botaoLimpar.setOnClickListener(new View.OnClickListener() {
			@Override
			public void onClick(View v) {
				CalculaAnionGap.this.limpaTela();
			}
		});

		ImageButton botaoInfo = (ImageButton) findViewById(R.id.botaoInfo);
		botaoInfo.setOnClickListener(new View.OnClickListener() {
			@Override
			public void onClick(View v) {
				String formula = getString(R.string.anion_gap_formula);
				CalculaAnionGap.this.mostraPopupDescricao(formula);
			}
		});
	}


	/**
	 * Valida os dados inseridos pelo usuário. Se alguma variável não foi inserida destaca o campo em
vermelho e exibe alerta ao usuário
	 */
	private boolean validaDados(){
		boolean dadosValidos = true;
		if(this.campoNivelSodio.getText() == null ||
"".equals(this.campoNivelSodio.getText().toString())){
			this.campoNivelSodio.setError(getString(R.string.campo_obrigatorio));
			dadosValidos = false;
		}
		if(this.campoNivelCloro.getText() == null ||
"".equals(this.campoNivelCloro.getText().toString())){
			this.campoNivelCloro.setError(getString(R.string.campo_obrigatorio));
			dadosValidos = false;
		}
		if(this.campoNivelBicarbonato.getText() == null ||
"".equals(this.campoNivelBicarbonato.getText().toString())){
			this.campoNivelBicarbonato.setError(getString(R.string.campo_obrigatorio));
			dadosValidos = false;
		}
		return dadosValidos;
	}

	/**
	 * Captura os valores inseridos pelo usuário, calcula o valor do Anion Gap e exibe na tela
```

```
	 */
	private void calcularAnionGap(){

		System.out.println("Dados válidos. Calculando AnionGap");

		Double nivelSodio = Double.valueOf(this.campoNivelSodio.getText().toString());
		Double nivelCloro = Double.valueOf(this.campoNivelCloro.getText().toString());
		Double nivelBicarbonato = Double.valueOf(this.campoNivelBicarbonato.getText().toString());

		Double resultadoCalculado = ( nivelSodio - (nivelCloro + nivelBicarbonato));
		String resultadoArredondado = new DecimalFormat("0.000").format(resultadoCalculado);

		String resultado = getString(R.string.anion_gap) + " = " + resultadoArredondado + " " +
getString(R.string.mmol_l);

		this.valorFinal.setText(resultado);
		this.exibeResultado();
	}

	/**
	 * Limpa a tela para um novo cálculo
	 */
	private void limpaTela(){
		this.campoNivelSodio.setText("");
		this.campoNivelCloro.setText("");
		this.campoNivelBicarbonato.setText("");
		this.escondeResultado();
		this.campoNivelSodio.requestFocus();
	}
}

public class CalculaApgar extends ActivityGeral {

	private RadioGroup campoFreqCardiaca, campoRespiracao, campoMuscular, campoCorPele,
campoIrritabilidade;

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

		System.out.println("Criando Activity: CalculaApgar");

		this.campoFreqCardiaca = (RadioGroup) findViewById(R.id.radioGroupFreqCard);
		this.campoRespiracao = (RadioGroup) findViewById(R.id.radioGroupRespiracao);
		this.campoMuscular = (RadioGroup) findViewById(R.id.radioGroupMuscular);
		this.campoCorPele = (RadioGroup) findViewById(R.id.radioGroupCor);
		this.campoIrritabilidade = (RadioGroup) findViewById(R.id.radioGroupIrritabilidade);
		this.labelResultado = (TextView) findViewById(R.id.labelResultado);
		this.valorFinal = (TextView) findViewById(R.id.valorFinal);

		EventoRadioGrupoAlterado eventoRadio = new EventoRadioGrupoAlterado();
		this.campoFreqCardiaca.setOnCheckedChangeListener(eventoRadio);
		this.campoRespiracao.setOnCheckedChangeListener(eventoRadio);
		this.campoMuscular.setOnCheckedChangeListener(eventoRadio);
		this.campoCorPele.setOnCheckedChangeListener(eventoRadio);
		this.campoIrritabilidade.setOnCheckedChangeListener(eventoRadio);

		Button botaoVoltar = (Button) findViewById(R.id.botaoVoltar);
		botaoVoltar.setOnClickListener(new View.OnClickListener() {
			@Override
			public void onClick(View v) {
				startActivity(new Intent(getApplication(), ListaCalculos.class));
			}
		});

		Button botaoLimpar = (Button) findViewById(R.id.botaoLimpar);
```

```
		botaoLimpar.setOnClickListener(new View.OnClickListener() {
			@Override
			public void onClick(View v) {
				CalculaApgar.this.limpaTela();
			}
		});

		ImageButton botaoInfo = (ImageButton) findViewById(R.id.botaoInfo);
		botaoInfo.setOnClickListener(new View.OnClickListener() {
			@Override
			public void onClick(View v) {
				CalculaApgar.this.mostraPopupDescricao(getString(R.string.apgar_formula));
			}
		});
	}

	private void calcularApgar(){
		Integer resultado = 0;
		boolean exibirResultado = true;

		/*
		 * Recupera resultado variável frequência cardíaca
		 */
		switch (this.campoFreqCardiaca.getCheckedRadioButtonId()) {
		case R.id.campoFreqCard1:
			break;
		case R.id.campoFreqCard2:
			resultado += 1;
			break;
		case R.id.campoFreqCard3:
			resultado += 2;
			break;
		default:
			exibirResultado = false;
		}

		/*
		 * Recupera resultado variável respiração
		 */
		switch (this.campoRespiracao.getCheckedRadioButtonId()) {
		case R.id.campoRespiracao1:
			break;
		case R.id.campoRespiracao2:
			resultado += 1;
			break;
		case R.id.campoRespiracao3:
			resultado += 2;
			break;
		default:
			exibirResultado = false;
		}

		/*
		 * Recupera resultado variável tônus muscular
		 */
		switch (this.campoMuscular.getCheckedRadioButtonId()) {
		case R.id.campoMuscular1:
			break;
		case R.id.campoMuscular2:
			resultado += 1;
			break;
		case R.id.campoMuscular3:
			resultado += 2;
			break;
		default:
			exibirResultado = false;
		}
```

```
		/*
		 * Recupera resultado variável cor da pele
		 */
		switch (this.campoCorPele.getCheckedRadioButtonId()) {
		case R.id.campoCorPele1:
			break;
		case R.id.campoCorPele2:
			resultado += 1;
			break;
		case R.id.campoCorPele3:
			resultado += 2;
			break;
		default:
			exibirResultado = false;
		}

		/*
		 * Recupera resultado variável irritabilidade reflexa
		 */
		switch (this.campoIrritabilidade.getCheckedRadioButtonId()) {
		case R.id.campoIrritabilidade1:
			break;
		case R.id.campoIrritabilidade2:
			resultado += 1;
			break;
		case R.id.campoIrritabilidade3:
			resultado += 2;
			break;
		default:
			exibirResultado = false;
		}

		String parecerFinal = "";

		if(exibirResultado){
			if(resultado > 7){
				parecerFinal = getString(R.string.escala_apgar) + ": " + resultado + ".\n" +
getString(R.string.result_bebe_sem_asfixia);
			} else if(resultado > 4){
				parecerFinal = getString(R.string.escala_apgar) + ": " + resultado + ".\n" +
getString(R.string.result_bebe_asfixia_leve);
			} else if(resultado > 2){
				parecerFinal = getString(R.string.escala_apgar) + ": " + resultado + ".\n" +
getString(R.string.result_bebe_asfixia_moderada);
			} else {
				parecerFinal = getString(R.string.escala_apgar) + ": " + resultado + ".\n" +
getString(R.string.result_bebe_asfixia_grave);
			}
		} else {
			parecerFinal = getString(R.string.selecione_item_opcao);
		}

		this.valorFinal.setText(parecerFinal);
		this.exibeResultado();
	}

	private void limpaTela(){
		this.campoCorPele.clearCheck();
		this.campoFreqCardiaca.clearCheck();
		this.campoIrritabilidade.clearCheck();
		this.campoMuscular.clearCheck();
		this.campoRespiracao.clearCheck();
		this.escondeResultado();
	}

	/**
```

```
	 * Classe interna que representa o evento para calcular a escala de Apgar. Este evento é disparado quando o
RadioGroup
	 * é alterado.
	 */
	private class EventoRadioGrupoAlterado implements RadioGroup.OnCheckedChangeListener{
		@Override
		public void onCheckedChanged(RadioGroup arg0, int arg1) {
			CalculaApgar.this.calcularApgar();
		}
	}
}

public class CalculaAshworth extends ActivityGeral {

	private RadioGroup campoGrauEspasticidade;

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

		System.out.println("Criando Activity: CalculaAshworth");

		this.campoGrauEspasticidade = (RadioGroup)
findViewById(R.id.radioGroupGrauEspasticidade);
		this.labelResultado = (TextView) findViewById(R.id.labelResultado);
		this.valorFinal = (TextView) findViewById(R.id.valorFinal);

		this.campoGrauEspasticidade.setOnCheckedChangeListener(new
RadioGroup.OnCheckedChangeListener() {
			@Override
			public void onCheckedChanged(RadioGroup group, int checkedId) {
				CalculaAshworth.this.calcularAshworth();
			}
		});

		Button botaoVoltar = (Button) findViewById(R.id.botaoVoltar);
		botaoVoltar.setOnClickListener(new View.OnClickListener() {
			@Override
			public void onClick(View v) {
				startActivity(new Intent(getApplication(), ListaCalculos.class));
			}
		});

		Button botaoLimpar = (Button) findViewById(R.id.botaoLimpar);
		botaoLimpar.setOnClickListener(new View.OnClickListener() {
			@Override
			public void onClick(View v) {
				CalculaAshworth.this.limpaTela();
			}
		});

		ImageButton botaoInfo = (ImageButton) findViewById(R.id.botaoInfo);
		botaoInfo.setOnClickListener(new View.OnClickListener() {
			@Override
			public void onClick(View v) {

	CalculaAshworth.this.mostraPopupDescricao(getString(R.string.ashworth_formula));
			}
		});
	}

	private void calcularAshworth(){
		String resultado = "";
		boolean exibirResultado = true;

		switch (this.campoGrauEspasticidade.getCheckedRadioButtonId()) {
```

```
                case R.id.campoGrauEspasticidade0:
                        resultado = getString(R.string.result_ashworth_1);
                        break;
                case R.id.campoGrauEspasticidade1:
                        resultado = getString(R.string.result_ashworth_2);
                        break;
                case R.id.campoGrauEspasticidade1plus:
                        resultado = getString(R.string.result_ashworth_3);
                        break;
                case R.id.campoGrauEspasticidade2:
                        resultado = getString(R.string.result_ashworth_4);
                        break;
                case R.id.campoGrauEspasticidade3:
                        resultado = getString(R.string.result_ashworth_5);
                        break;
                case R.id.campoGrauEspasticidade4:
                        resultado = getString(R.string.result_ashworth_6);
                        break;
                default:
                        exibirResultado = false;
                }

                if(exibirResultado){
                        valorFinal.setText(resultado);
                        this.exibeResultado();
                } else {
                        this.escondeResultado();
                }
        }

        private void limpaTela(){
                this.campoGrauEspasticidade.clearCheck();
                this.escondeResultado();
        }
}

public class CalculaChildPugh extends ActivityGeral {

        private RadioGroup campoBilirrubina, campoAlbumina, campoINR, campoAscite, campoEncefapatia;

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

                System.out.println("Criando Activity: CalculaChildPugh");

                this.campoBilirrubina = (RadioGroup) findViewById(R.id.radioGroupBilirrubinaTotal);
                this.campoAlbumina = (RadioGroup) findViewById(R.id.radioGroupAlbumina);
                this.campoINR = (RadioGroup) findViewById(R.id.radioGroupINR);
                this.campoAscite = (RadioGroup) findViewById(R.id.radioGroupAscite);
                this.campoEncefapatia = (RadioGroup) findViewById(R.id.radioGroupEncefalopatia);
                this.labelResultado = (TextView) findViewById(R.id.labelResultado);
                this.valorFinal = (TextView) findViewById(R.id.valorFinal);

                EventoRadioGrupoAlterado eventoRadio = new EventoRadioGrupoAlterado();
                this.campoBilirrubina.setOnCheckedChangeListener(eventoRadio);
                this.campoAlbumina.setOnCheckedChangeListener(eventoRadio);
                this.campoINR.setOnCheckedChangeListener(eventoRadio);
                this.campoAscite.setOnCheckedChangeListener(eventoRadio);
                this.campoEncefapatia.setOnCheckedChangeListener(eventoRadio);

                Button botaoVoltar = (Button) findViewById(R.id.botaoVoltar);
                botaoVoltar.setOnClickListener(new View.OnClickListener() {
                        @Override
                        public void onClick(View v) {
                                startActivity(new Intent(getApplication(), ListaCalculos.class));
```

```
		}
	});

	Button botaoLimpar = (Button) findViewById(R.id.botaoLimpar);
	botaoLimpar.setOnClickListener(new View.OnClickListener() {
		@Override
		public void onClick(View v) {
			CalculaChildPugh.this.limpaTela();
		}
	});

	ImageButton botaoInfo = (ImageButton) findViewById(R.id.botaoInfo);
	botaoInfo.setOnClickListener(new View.OnClickListener() {
		@Override
		public void onClick(View v) {

CalculaChildPugh.this.mostraPopupDescricao(getString(R.string.child_pugh_formula));
		}
	});
}

private void calcularChildPugh(){
	Integer resultado = 0;
	boolean exibirResultado = true;

	/*
	 * Recupera resultado variável bilirrubina total
	 */
	switch (this.campoBilirrubina.getCheckedRadioButtonId()) {
	case R.id.campoBilirrubinaTotal1:
		resultado += 1;
		break;
	case R.id.campoBilirrubinaTotal2:
		resultado += 2;
		break;
	case R.id.campoBilirrubinaTotal3:
		resultado += 3;
		break;
	default:
		exibirResultado = false;
	}

	/*
	 * Recupera resultado variável albumina
	 */
	switch (this.campoAlbumina.getCheckedRadioButtonId()) {
	case R.id.campoAlbumina1:
		resultado += 1;
		break;
	case R.id.campoAlbumina2:
		resultado += 2;
		break;
	case R.id.campoAlbumina3:
		resultado += 3;
		break;
	default:
		exibirResultado = false;
	}

	/*
	 * Recupera resultado variável INR
	 */
	switch (this.campoINR.getCheckedRadioButtonId()) {
	case R.id.campoINRchild1:
		resultado += 1;
		break;
	case R.id.campoINRchild2:
```

```
                        resultado += 2;
                        break;
                case R.id.campoINRchild3:
                        resultado += 3;
                        break;
                default:
                        exibirResultado = false;
                }

                /*
                 * Recupera resultado variável ascite
                 */
                switch (this.campoAscite.getCheckedRadioButtonId()) {
                case R.id.campoAscite1:
                        resultado += 1;
                        break;
                case R.id.campoAscite2:
                        resultado += 2;
                        break;
                case R.id.campoAscite3:
                        resultado += 3;
                        break;
                default:
                        exibirResultado = false;
                }

                /*
                 * Recupera resultado variável encefalopatia
                 */
                switch (this.campoEncefapatia.getCheckedRadioButtonId()) {
                case R.id.campoEncefalopatia1:
                        resultado += 1;
                        break;
                case R.id.campoEncefalopatia2:
                        resultado += 2;
                        break;
                case R.id.campoEncefalopatia3:
                        resultado += 3;
                        break;
                default:
                        exibirResultado = false;
                }

                String parecerFinal = "";

                if(exibirResultado){
                        if(resultado > 9){
                                parecerFinal = getString(R.string.classificacao_child_pugh) + ": " + resultado
+ getString(R.string.result_child_1);
                        } else if(resultado > 6){
                                parecerFinal = getString(R.string.classificacao_child_pugh) + ": " + resultado
+ getString(R.string.result_child_2);
                        } else {
                                parecerFinal = getString(R.string.classificacao_child_pugh) + ": " + resultado
+ getString(R.string.result_child_3);
                        }
                } else {
                        parecerFinal = getString(R.string.selecione_item_opcao);
                }

                this.valorFinal.setText(parecerFinal);
                this.exibeResultado();
        }

        private void limpaTela(){
                this.campoBilirrubina.clearCheck();
                this.campoAlbumina.clearCheck();
```

```
		this.campoINR.clearCheck();
		this.campoAscite.clearCheck();
		this.campoEncefapatia.clearCheck();
		this.escondeResultado();
	}

	/**
	 * Classe interna que representa o evento para calcular a classificação de Child Pugh. Este evento é
disparado quando o RadioGroup
	 * é alterado.
	 */
	private class EventoRadioGrupoAlterado implements RadioGroup.OnCheckedChangeListener{
		@Override
		public void onCheckedChanged(RadioGroup arg0, int arg1) {
			CalculaChildPugh.this.calcularChildPugh();
		}
	}
}

public class CalculaCockroftGault extends ActivityGeral {

	private EditText campoIdade, campoPeso, campoCreatinina;
	private RadioGroup campoSexo;

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

		System.out.println("Criando Activity: CalculaCockroftGault");

		this.campoIdade = (EditText) findViewById(R.id.campoIdadeGault);
		this.campoPeso = (EditText) findViewById(R.id.campoPesoGault);
		this.campoCreatinina = (EditText) findViewById(R.id.campoCreatininaGault);
		this.campoSexo = (RadioGroup) findViewById(R.id.campoSexoGault);
		this.labelResultado = (TextView) findViewById(R.id.labelResultado);
		this.valorFinal = (TextView) findViewById(R.id.valorFinal);

		Button botaoCalcular = (Button) findViewById(R.id.botaoCalcular);
		botaoCalcular.setOnClickListener(new View.OnClickListener() {
			@Override
			public void onClick(View v) {

	CalculaCockroftGault.this.esconderTecladoVirtual(CalculaCockroftGault.this.getCurrentFocus().getWindo
wToken());
				if(CalculaCockroftGault.this.validaDados()){
					CalculaCockroftGault.this.calcularCockroftGault();
				}
			}
		});

		Button botaoVoltar = (Button) findViewById(R.id.botaoVoltar);
		botaoVoltar.setOnClickListener(new View.OnClickListener() {
			@Override
			public void onClick(View v) {
				startActivity(new Intent(getApplication(), ListaCalculos.class));
			}
		});

		Button botaoLimpar = (Button) findViewById(R.id.botaoLimpar);
		botaoLimpar.setOnClickListener(new View.OnClickListener() {
			@Override
			public void onClick(View v) {
				CalculaCockroftGault.this.limpaTela();
			}
		});
```

```
		ImageButton botaoInfo = (ImageButton) findViewById(R.id.botaoInfo);
		botaoInfo.setOnClickListener(new View.OnClickListener() {
			@Override
			public void onClick(View v) {

	CalculaCockroftGault.this.mostraPopupDescricao(getString(R.string.cockroft_gault_formula));
			}
		});
	}

	/**
	 * Valida os dados inseridos pelo usuário. Se alguma variável não foi inserida destaca o campo em
vermelho e exibe alerta ao usuário
	 */
	private boolean validaDados(){
		boolean dadosValidos = true;
		if(this.campoIdade.getText() == null || "".equals(this.campoIdade.getText().toString())){
			this.campoIdade.setError(getString(R.string.campo_obrigatorio));
			dadosValidos = false;
		}
		if(this.campoPeso.getText() == null || "".equals(this.campoPeso.getText().toString())){
			this.campoPeso.setError(getString(R.string.campo_obrigatorio));
			dadosValidos = false;
		}
		if(this.campoCreatinina.getText() == null || "".equals(this.campoCreatinina.getText().toString())){
			this.campoCreatinina.setError(getString(R.string.campo_obrigatorio));
			dadosValidos = false;
		}
		return dadosValidos;
	}

	/**
	 * Captura os valores inseridos pelo usuário, calcula o valor do Anion Gap e exibe na tela
	 */
	private void calcularCockroftGault(){

		System.out.println("Dados válidos. Calculando Escala Meld");

		String resultadoArredondado = "";
		Double creatinina = Double.valueOf(this.campoCreatinina.getText().toString());
		if(creatinina != 0){
			Integer idade = Integer.valueOf(this.campoIdade.getText().toString());
			Double peso = Double.valueOf(this.campoPeso.getText().toString());
			boolean sexoMasculino = true;

			if(this.campoSexo.getCheckedRadioButtonId() == R.id.sexoFemGault){
				sexoMasculino = false; //feminino
			}

			Double resultadoCalculado = ((140 - idade) * peso) / (creatinina * 72);

			if(!sexoMasculino){
				resultadoCalculado *= 0.85;
			}

			resultadoArredondado = new DecimalFormat("0.00").format(resultadoCalculado);
		}else{
			resultadoArredondado = "0";
		}

		String resultado = getString(R.string.estimativa) + " = " + resultadoArredondado + " " +
getString(R.string.mL_min) + ".";

		this.valorFinal.setText(resultado);
		this.exibeResultado();
	}
```

```
	/**
	 * Limpa a tela para um novo cálculo
	 */
	private void limpaTela(){
		this.campoIdade.setText("");
		this.campoPeso.setText("");
		this.campoCreatinina.setText("");
		this.campoSexo.check(R.id.sexoMascGault);
		this.escondeResultado();
		this.campoIdade.requestFocus();
	}
}

public class CalculaCriterioRanson extends ActivityGeral {

	private ToggleButton toogleIdade, toogleGlicemia, toogleLeucocitos, toogleLdh, toogleAst,
toogleHematocrito, toogleCalcio,
		toogleDeficitBase, toogleBunUrinario, toogleSequestroLiquido, toogleHipoxemia;

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

		System.out.println("Criando Activity: CalculaCriterioRanson");

		this.toogleIdade = (ToggleButton) findViewById(R.id.toggleIdadeRanson);
		this.toogleGlicemia = (ToggleButton) findViewById(R.id.toggleGlicemiaRanson);
		this.toogleLeucocitos = (ToggleButton) findViewById(R.id.toggleLeucocitosRanson);
		this.toogleLdh = (ToggleButton) findViewById(R.id.toggleLdhSericaRanson);
		this.toogleAst = (ToggleButton) findViewById(R.id.toggleAstSericaRanson);
		this.toogleHematocrito = (ToggleButton) findViewById(R.id.toggleQuedaHematocritoRanson);
		this.toogleCalcio = (ToggleButton) findViewById(R.id.toggleCalcioSericoRanson);
		this.toogleDeficitBase = (ToggleButton) findViewById(R.id.toggleDeficitBaseRanson);
		this.toogleBunUrinario = (ToggleButton) findViewById(R.id.toggleBunUrinarioRanson);
		this.toogleSequestroLiquido = (ToggleButton)
findViewById(R.id.toggleSequestroLiquidoRanson);
		this.toogleHipoxemia = (ToggleButton) findViewById(R.id.toggleHipoxemiaRanson);
		this.labelResultado = (TextView) findViewById(R.id.labelResultado);
		this.valorFinal = (TextView) findViewById(R.id.valorFinal);

		EventoToggle eventoToggle = new EventoToggle();

		this.toogleIdade.setOnCheckedChangeListener(eventoToggle);
		this.toogleGlicemia.setOnCheckedChangeListener(eventoToggle);
		this.toogleLeucocitos.setOnCheckedChangeListener(eventoToggle);
		this.toogleLdh.setOnCheckedChangeListener(eventoToggle);
		this.toogleAst.setOnCheckedChangeListener(eventoToggle);
		this.toogleHematocrito.setOnCheckedChangeListener(eventoToggle);
		this.toogleCalcio.setOnCheckedChangeListener(eventoToggle);
		this.toogleDeficitBase.setOnCheckedChangeListener(eventoToggle);
		this.toogleBunUrinario.setOnCheckedChangeListener(eventoToggle);
		this.toogleSequestroLiquido.setOnCheckedChangeListener(eventoToggle);
		this.toogleHipoxemia.setOnCheckedChangeListener(eventoToggle);

		Button botaoVoltar = (Button) findViewById(R.id.botaoVoltar);
		botaoVoltar.setOnClickListener(new View.OnClickListener() {
			@Override
			public void onClick(View v) {
				startActivity(new Intent(getApplication(), ListaCalculos.class));
			}
		});

		Button botaoLimpar = (Button) findViewById(R.id.botaoLimpar);
		botaoLimpar.setOnClickListener(new View.OnClickListener() {
			@Override
			public void onClick(View v) {
```

```
                        CalculaCriterioRanson.this.limpaTela();
                    }
                });

                ImageButton botaoInfo = (ImageButton) findViewById(R.id.botaoInfo);
                botaoInfo.setOnClickListener(new View.OnClickListener() {
                    @Override
                    public void onClick(View v) {

        CalculaCriterioRanson.this.mostraPopupDescricao(getString(R.string.ranson_formula));
                    }
                });
            }

            @Override
            protected void onResume(){
                super.onResume();
                this.calcularCriterioRanson();
            }

            private void calcularCriterioRanson(){
                System.out.println("Calculando CriterioRanson");

                int idade = (this.toogleIdade.isChecked() ? 1 : 0);
                int glicemia = (this.toogleGlicemia.isChecked() ? 1 : 0);
                int leucocitos = (this.toogleLeucocitos.isChecked() ? 1 : 0);
                int ldh = (this.toogleLdh.isChecked() ? 1 : 0);
                int ast = (this.toogleAst.isChecked() ? 1 : 0);
                int hematocrito = (this.toogleHematocrito.isChecked() ? 1 : 0);
                int calcio = (this.toogleCalcio.isChecked() ? 1 : 0);
                int deficitBase = (this.toogleDeficitBase.isChecked() ? 1 : 0);
                int bunUrinario = (this.toogleBunUrinario.isChecked() ? 1 : 0);
                int sequestroLiquido = (this.toogleSequestroLiquido.isChecked() ? 1 : 0);
                int hipoxemia = (this.toogleHipoxemia.isChecked() ? 1 : 0);

                int total = idade + glicemia + leucocitos + ldh + ast + hematocrito + calcio + deficitBase +
bunUrinario + sequestroLiquido + hipoxemia;

                StringBuilder resultado = new StringBuilder();

                if(total != 0){
                    if(total < 3){
                        resultado.append(getString(R.string.result_quantidade_criterios));
                        resultado.append(": ");
                        resultado.append(total);
                        resultado.append(".\n");
                        resultado.append(getString(R.string.result_ranson_1));
                    } else {
                        if(total < 5){
                            resultado.append(getString(R.string.result_quantidade_criterios));
                            resultado.append(": ");
                            resultado.append(total);
                            resultado.append(".\n");
                            resultado.append(getString(R.string.result_ranson_2));
                        } else if(total < 7){
                            resultado.append(getString(R.string.result_quantidade_criterios));
                            resultado.append(": ");
                            resultado.append(total);
                            resultado.append(".\n");
                            resultado.append(getString(R.string.result_ranson_3));
                        } else {
                            resultado.append(getString(R.string.result_quantidade_criterios));
                            resultado.append(": ");
                            resultado.append(total);
                            resultado.append(".\n");
                            resultado.append(getString(R.string.result_ranson_4));
                        }
```

```
                                        resultado.append(getString(R.string.result_pancreatite_grave_provavel));
                                }
                                this.valorFinal.setText(resultado.toString());
                                this.exibeResultado();
                        } else {
                                this.escondeResultado();
                        }
                }

                /**
                 * Limpa a tela para um novo cálculo
                 */
                private void limpaTela(){
                        this.toogleIdade.setChecked(false);
                        this.toogleGlicemia.setChecked(false);
                        this.toogleLeucocitos.setChecked(false);
                        this.toogleLdh.setChecked(false);
                        this.toogleAst.setChecked(false);
                        this.toogleHematocrito.setChecked(false);
                        this.toogleCalcio.setChecked(false);
                        this.toogleDeficitBase.setChecked(false);
                        this.toogleBunUrinario.setChecked(false);
                        this.toogleSequestroLiquido.setChecked(false);
                        this.toogleHipoxemia.setChecked(false);
                        this.escondeResultado();
                        this.calcularCriterioRanson();
                }

                /**
                 * Classe interna que representa o evento para calcular o criterio de Ranson. Este evento é disparado quando
o ToggleButton
                 * é clicado.
                 */
                private class EventoToggle implements CompoundButton.OnCheckedChangeListener{
                        @Override
                        public void onCheckedChanged(CompoundButton buttonView,
                                        boolean isChecked) {
                                CalculaCriterioRanson.this.calcularCriterioRanson();

                        }
                }
}

public class CalculaForrest extends ActivityGeral {

        private RadioGroup campoTipoLesao;

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

                System.out.println("Criando Activity: CalculaForrest");

                this.campoTipoLesao= (RadioGroup) findViewById(R.id.radioGroupTipoLesaoForrest);
                this.labelResultado = (TextView) findViewById(R.id.labelResultado);
                this.valorFinal = (TextView) findViewById(R.id.valorFinal);

                this.campoTipoLesao.setOnCheckedChangeListener(new
RadioGroup.OnCheckedChangeListener() {
                        @Override
                        public void onCheckedChanged(RadioGroup group, int checkedId) {
                                CalculaForrest.this.calcularForrest();
                        }
                });

                Button botaoVoltar = (Button) findViewById(R.id.botaoVoltar);
```

```
		botaoVoltar.setOnClickListener(new View.OnClickListener() {
			@Override
			public void onClick(View v) {
				startActivity(new Intent(getApplication(), ListaCalculos.class));
			}
		});

		Button botaoLimpar = (Button) findViewById(R.id.botaoLimpar);
		botaoLimpar.setOnClickListener(new View.OnClickListener() {
			@Override
			public void onClick(View v) {
				CalculaForrest.this.limpaTela();
			}
		});

		ImageButton botaoInfo = (ImageButton) findViewById(R.id.botaoInfo);
		botaoInfo.setOnClickListener(new View.OnClickListener() {
			@Override
			public void onClick(View v) {

	CalculaForrest.this.mostraPopupDescricao(getString(R.string.forrest_formula));
			}
		});
	}

	private void calcularForrest(){
		String resultado = "";
		boolean exibirResultado = true;

		switch (this.campoTipoLesao.getCheckedRadioButtonId()) {
		case R.id.campoTipoLesaoForrest1A:
			resultado = getString(R.string.result_forrest_1);
			break;
		case R.id.campoTipoLesaoForrest1B:
			resultado = getString(R.string.result_forrest_2);
			break;
		case R.id.campoTipoLesaoForrest2A:
			resultado = getString(R.string.result_forrest_3);
			break;
		case R.id.campoTipoLesaoForrest2B:
			resultado = getString(R.string.result_forrest_4);
			break;
		case R.id.campoTipoLesaoForrest2C:
			resultado = getString(R.string.result_forrest_5);
			break;
		case R.id.campoTipoLesaoForrest3:
			resultado = getString(R.string.result_forrest_6);
			break;
		default:
			exibirResultado = false;
		}

		if(exibirResultado){
			valorFinal.setText(resultado);
			this.exibeResultado();
		}else{
			this.escondeResultado();
		}
	}

	private void limpaTela(){
		this.campoTipoLesao.clearCheck();
		this.escondeResultado();
	}
}

public class CalculaGlasgow extends ActivityGeral {
```

```
private RadioGroup campoOcular, campoVerbal, campoMotor;

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

	System.out.println("Criando Activity: CalculaGlasgow");

	this.campoOcular = (RadioGroup) findViewById(R.id.radioGroupOcular);
	this.campoVerbal = (RadioGroup) findViewById(R.id.radioGroupVerbal);
	this.campoMotor = (RadioGroup) findViewById(R.id.radioGroupMotor);
	this.labelResultado = (TextView) findViewById(R.id.labelResultado);
	this.valorFinal = (TextView) findViewById(R.id.valorFinal);

	EventoRadioGrupoAlterado eventoRadio = new EventoRadioGrupoAlterado();
	this.campoOcular.setOnCheckedChangeListener(eventoRadio);
	this.campoVerbal.setOnCheckedChangeListener(eventoRadio);
	this.campoMotor.setOnCheckedChangeListener(eventoRadio);

	Button botaoVoltar = (Button) findViewById(R.id.botaoVoltar);
	botaoVoltar.setOnClickListener(new View.OnClickListener() {
		@Override
		public void onClick(View v) {
			startActivity(new Intent(getApplication(), ListaCalculos.class));
		}
	});

	Button botaoLimpar = (Button) findViewById(R.id.botaoLimpar);
	botaoLimpar.setOnClickListener(new View.OnClickListener() {
		@Override
		public void onClick(View v) {
			CalculaGlasgow.this.limpaTela();
		}
	});

	ImageButton botaoInfo = (ImageButton) findViewById(R.id.botaoInfo);
	botaoInfo.setOnClickListener(new View.OnClickListener() {
		@Override
		public void onClick(View v) {

CalculaGlasgow.this.mostraPopupDescricao(getString(R.string.glasgow_formula));
		}
	});

}

private void calcularGlasgow(){
	Integer resultado = 0;
	boolean exibirResultado = true;

	/*
	 * Recupera resultado variável ocular
	 */
	switch (this.campoOcular.getCheckedRadioButtonId()) {
	case R.id.campoOcular1:
		resultado += 1;
		break;
	case R.id.campoOcular2:
		resultado += 2;
		break;
	case R.id.campoOcular3:
		resultado += 3;
		break;
	case R.id.campoOcular4:
		resultado += 4;
```

```
		break;
default:
		exibirResultado = false;
}

/*
 * Recupera resultado variável Verbal
 */
switch (this.campoVerbal.getCheckedRadioButtonId()) {
case R.id.campoVerbal1:
		resultado += 1;
		break;
case R.id.campoVerbal2:
		resultado += 2;
		break;
case R.id.campoVerbal3:
		resultado += 3;
		break;
case R.id.campoVerbal4:
		resultado += 4;
		break;
case R.id.campoVerbal5:
		resultado += 5;
		break;
default:
		exibirResultado = false;
}

/*
 * Recupera resultado variável Motor
 */
switch (this.campoMotor.getCheckedRadioButtonId()) {
case R.id.campoMotor1:
		resultado += 1;
		break;
case R.id.campoMotor2:
		resultado += 2;
		break;
case R.id.campoMotor3:
		resultado += 3;
		break;
case R.id.campoMotor4:
		resultado += 4;
		break;
case R.id.campoMotor5:
		resultado += 5;
		break;
case R.id.campoMotor6:
		resultado += 6;
		break;
default:
		exibirResultado = false;
}

StringBuilder parecerFinal = new StringBuilder();

if(exibirResultado){
		parecerFinal.append(getString(R.string.escala_glasgow));
		parecerFinal.append(": ");

		if(resultado == 15){
			parecerFinal.append(resultado);
			parecerFinal.append(getString(R.string.result_glasgow_1));
		} else if(resultado > 10){
			parecerFinal.append(resultado);
			parecerFinal.append(getString(R.string.result_glasgow_2));
		} else if(resultado > 6){
```

```
                                        parecerFinal.append(resultado);
                                        parecerFinal.append(getString(R.string.result_glasgow_3));
                                } else if(resultado > 3){
                                        parecerFinal.append(resultado);
                                        parecerFinal.append(getString(R.string.result_glasgow_4));
                                } else {
                                        parecerFinal.append(resultado);
                                        parecerFinal.append(getString(R.string.result_glasgow_5));
                                }

                                if(resultado < 9){
                                        parecerFinal.append(getString(R.string.result_glasgow_6));
                                }
                        } else {
                                parecerFinal.append(getString(R.string.selecione_item_opcao));
                        }

                        this.valorFinal.setText(parecerFinal.toString());
                        this.exibeResultado();
                }

                private void limpaTela(){
                        this.campoVerbal.clearCheck();
                        this.campoOcular.clearCheck();
                        this.campoMotor.clearCheck();
                        this.escondeResultado();
                }

                /**
                 * Classe interna que representa o evento para calcular a escala de coma de Glasgow. Este evento é
disparado quando o RadioGroup
                 * é alterado.
                 */
                private class EventoRadioGrupoAlterado implements RadioGroup.OnCheckedChangeListener{
                        @Override
                        public void onCheckedChanged(RadioGroup arg0, int arg1) {
                                CalculaGlasgow.this.calcularGlasgow();
                        }
                }
}

public class CalculaGorduraCorporal extends ActivityGeral {

        private EditText campoIdade, campoGorduraCorporal;
        private RadioGroup campoSexo;

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

                System.out.println("Criando Activity: CalculaGorduraCorporal");

                this.campoIdade = (EditText) findViewById(R.id.campoIdadeGorduraCorporal);
                this.campoGorduraCorporal = (EditText) findViewById(R.id.campoGorduraCorporal);
                this.campoSexo = (RadioGroup) findViewById(R.id.campoSexoGorduraCorporal);
                this.labelResultado = (TextView) findViewById(R.id.labelResultado);
                this.valorFinal = (TextView) findViewById(R.id.valorFinal);

                Button botaoCalcular = (Button) findViewById(R.id.botaoCalcular);
                botaoCalcular.setOnClickListener(new View.OnClickListener() {
                        @Override
                        public void onClick(View v) {

        CalculaGorduraCorporal.this.esconderTecladoVirtual(CalculaGorduraCorporal.this.getCurrentFocus().get
WindowToken());
                                if(CalculaGorduraCorporal.this.validaDados()){
```

```
                                        CalculaGorduraCorporal.this.calcularGorduraCorporal();
                                }
                        }
                });

                Button botaoVoltar = (Button) findViewById(R.id.botaoVoltar);
                botaoVoltar.setOnClickListener(new View.OnClickListener() {
                        @Override
                        public void onClick(View v) {
                                startActivity(new Intent(getApplication(), ListaCalculos.class));
                        }
                });

                Button botaoLimpar = (Button) findViewById(R.id.botaoLimpar);
                botaoLimpar.setOnClickListener(new View.OnClickListener() {
                        @Override
                        public void onClick(View v) {
                                CalculaGorduraCorporal.this.limpaTela();
                        }
                });

                ImageButton botaoInfo = (ImageButton) findViewById(R.id.botaoInfo);
                botaoInfo.setOnClickListener(new View.OnClickListener() {
                        @Override
                        public void onClick(View v) {

        CalculaGorduraCorporal.this.mostraPopupDescricao(getString(R.string.gordura_corporal_formula));
                        }
                });
        }

        /**
         * Valida os dados inseridos pelo usuário. Se alguma variável não foi inserida destaca o campo em
vermelho e exibe alerta ao usuário
         */
        private boolean validaDados(){
                boolean dadosValidos = true;
                if(this.campoIdade.getText() == null || "".equals(this.campoIdade.getText().toString())){
                        this.campoIdade.setError(getString(R.string.campo_obrigatorio));
                        dadosValidos = false;
                }
                if(this.campoGorduraCorporal.getText() == null ||
"".equals(this.campoGorduraCorporal.getText().toString())){
                        this.campoGorduraCorporal.setError(getString(R.string.campo_obrigatorio));
                        dadosValidos = false;
                }
                return dadosValidos;
        }

        /**
         * Captura os valores inseridos pelo usuário, calcula a gordura corporal e exibe na tela
         */
        private void calcularGorduraCorporal(){

                System.out.println("Dados válidos. Calculando Margem de Gordura Corporal");

                Integer idade = Integer.valueOf(this.campoIdade.getText().toString());
                Integer gorduraCorporal = Integer.valueOf(this.campoGorduraCorporal.getText().toString());
                boolean sexoMasculino = true;

                if(this.campoSexo.getCheckedRadioButtonId() == R.id.sexoFemGorduraCorporal){
                        sexoMasculino = false; //feminino
                }

                String resultado = "";
                if(sexoMasculino){
                        if(idade > 17 && idade < 40){
```

```
                    if(gorduraCorporal < 9){
                        resultado = getString(R.string.restul_gordura_1);
                    } else if (gorduraCorporal < 21){
                        resultado = getString(R.string.restul_gordura_2);
                    } else if (gorduraCorporal < 26){
                        resultado = getString(R.string.restul_gordura_3);
                    } else {
                        resultado = getString(R.string.restul_gordura_4);
                    }
                } else if (idade < 60){
                    if(gorduraCorporal < 12){
                        resultado = getString(R.string.restul_gordura_5);
                    } else if (gorduraCorporal < 23){
                        resultado = getString(R.string.restul_gordura_6);
                    } else if (gorduraCorporal < 29){
                        resultado = getString(R.string.restul_gordura_7);
                    } else {
                        resultado = getString(R.string.restul_gordura_8);
                    }
                } else if (idade < 100){
                    if(gorduraCorporal < 14){
                        resultado = getString(R.string.restul_gordura_9);
                    } else if (gorduraCorporal < 26){
                        resultado = getString(R.string.restul_gordura_10);
                    } else if (gorduraCorporal < 31){
                        resultado = getString(R.string.restul_gordura_11);
                    } else {
                        resultado = getString(R.string.restul_gordura_12);
                    }
                }
            } else {
                if(idade > 17 && idade < 40){
                    if(gorduraCorporal < 22){
                        resultado = getString(R.string.restul_gordura_13);
                    } else if (gorduraCorporal < 34){
                        resultado = getString(R.string.restul_gordura_14);
                    } else if (gorduraCorporal < 40){
                        resultado = getString(R.string.restul_gordura_15);
                    } else {
                        resultado = getString(R.string.restul_gordura_16);
                    }
                } else if (idade < 60){
                    if(gorduraCorporal < 24){
                        resultado = getString(R.string.restul_gordura_17);
                    } else if (gorduraCorporal < 35){
                        resultado = getString(R.string.restul_gordura_18);
                    } else if (gorduraCorporal < 41){
                        resultado = getString(R.string.restul_gordura_19);
                    } else {
                        resultado = getString(R.string.restul_gordura_20);
                    }
                } else if (idade < 100){
                    if(gorduraCorporal < 25){
                        resultado = getString(R.string.restul_gordura_21);
                    } else if (gorduraCorporal < 37){
                        resultado = getString(R.string.restul_gordura_22);
                    } else if (gorduraCorporal < 43){
                        resultado = getString(R.string.restul_gordura_23);
                    } else {
                        resultado = getString(R.string.restul_gordura_24);
                    }
                }
            }

            if("".equals(resultado)){
                resultado = getString(R.string.restul_gordura_25);
            }
```

```
                this.valorFinal.setText(resultado);
                this.exibeResultado();
        }

        /**
         * Limpa a tela para um novo cálculo
         */
        private void limpaTela(){
                this.campoIdade.setText("");
                this.campoGorduraCorporal.setText("");
                this.campoSexo.check(R.id.sexoMascGorduraCorporal);
                this.escondeResultado();
                this.campoIdade.requestFocus();
        }
}

public class CalculaIMC extends ActivityGeral {

        private EditText campoPeso, campoAltura;

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

                System.out.println("Criando Activity: CalculaIMC");

                this.campoPeso = (EditText) findViewById(R.id.campoPesoIMC);
                this.campoAltura = (EditText) findViewById(R.id.campoAlturaIMC);
                this.labelResultado = (TextView) findViewById(R.id.labelResultado);
                this.valorFinal = (TextView) findViewById(R.id.valorFinal);

                Button botaoCalcular = (Button) findViewById(R.id.botaoCalcular);
                botaoCalcular.setOnClickListener(new View.OnClickListener() {
                        @Override
                        public void onClick(View v) {

        CalculaIMC.this.esconderTecladoVirtual(CalculaIMC.this.getCurrentFocus().getWindowToken());
                                if(CalculaIMC.this.validaDados()){
                                        CalculaIMC.this.calcularIMC();
                                }
                        }
                });

                Button botaoVoltar = (Button) findViewById(R.id.botaoVoltar);
                botaoVoltar.setOnClickListener(new View.OnClickListener() {
                        @Override
                        public void onClick(View v) {
                                startActivity(new Intent(getApplication(), ListaCalculos.class));
                        }
                });

                Button botaoLimpar = (Button) findViewById(R.id.botaoLimpar);
                botaoLimpar.setOnClickListener(new View.OnClickListener() {
                        @Override
                        public void onClick(View v) {
                                CalculaIMC.this.limpaTela();
                        }
                });

                ImageButton botaoInfo = (ImageButton) findViewById(R.id.botaoInfo);
                botaoInfo.setOnClickListener(new View.OnClickListener() {
                        @Override
                        public void onClick(View v) {
                                CalculaIMC.this.mostraPopupDescricao(getString(R.string.imc_formula));
                        }
```

```
                });
        }

        /**
         * Valida os dados inseridos pelo usuário. Se alguma variável não foi inserida destaca o campo em
vermelho e exibe alerta ao usuário
         */
        private boolean validaDados(){
                boolean dadosValidos = true;
                if(this.campoPeso.getText() == null || "".equals(this.campoPeso.getText().toString())){
                        this.campoPeso.setError(getString(R.string.campo_obrigatorio));
                        dadosValidos = false;
                }
                if(this.campoAltura.getText() == null || "".equals(this.campoAltura.getText().toString())){
                        this.campoAltura.setError(getString(R.string.campo_obrigatorio));
                        dadosValidos = false;
                }
                return dadosValidos;
        }

        /**
         * Captura os valores inseridos pelo usuário, calcula o valor do IMC e exibe na tela
         */
        private void calcularIMC(){
                System.out.println("Dados válidos. Calculando IMC");

                Double peso = Double.valueOf(this.campoPeso.getText().toString());
                Double altura = Double.valueOf(this.campoAltura.getText().toString());

                Double resultadoCalculado = ( peso / (altura * altura));
                String resultadoArredondado = new DecimalFormat("0.00").format(resultadoCalculado);

                StringBuilder resultado = new StringBuilder();
                resultado.append(getString(R.string.imc_sigla));
                resultado.append(" = ");
                resultado.append(resultadoArredondado);
                resultado.append(".\n");

                if(resultadoCalculado < 18.5){
                        resultado.append(getString(R.string.result_imc_1));
                } else if(resultadoCalculado < 25){
                        resultado.append(getString(R.string.result_imc_2));
                } else if(resultadoCalculado < 30){
                        resultado.append(getString(R.string.result_imc_3));
                } else if(resultadoCalculado < 35){
                        resultado.append(getString(R.string.result_imc_4));
                } else if(resultadoCalculado < 40){
                        resultado.append(getString(R.string.result_imc_5));
                } else{
                        resultado.append(getString(R.string.result_imc_6));
                }

                this.valorFinal.setText(resultado);
                this.exibeResultado();
        }

        /**
         * Limpa a tela para um novo cálculo
         */
        private void limpaTela(){
                this.campoPeso.setText("");
                this.campoAltura.setText("");
                this.escondeResultado();
                this.campoPeso.requestFocus();
        }
}
```

```
public class CalculaKillip extends ActivityGeral {

	private RadioGroup campoClasseKillip;

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

		System.out.println("Criando Activity: CalculaKillip");

		this.campoClasseKillip= (RadioGroup) findViewById(R.id.radioGroupClasseKillip);
		this.labelResultado = (TextView) findViewById(R.id.labelResultado);
		this.valorFinal = (TextView) findViewById(R.id.valorFinal);

		this.campoClasseKillip.setOnCheckedChangeListener(new
RadioGroup.OnCheckedChangeListener() {
			@Override
			public void onCheckedChanged(RadioGroup group, int checkedId) {
				CalculaKillip.this.calcularKillip();
			}
		});

		Button botaoVoltar = (Button) findViewById(R.id.botaoVoltar);
		botaoVoltar.setOnClickListener(new View.OnClickListener() {
			@Override
			public void onClick(View v) {
				startActivity(new Intent(getApplication(), ListaCalculos.class));
			}
		});

		Button botaoLimpar = (Button) findViewById(R.id.botaoLimpar);
		botaoLimpar.setOnClickListener(new View.OnClickListener() {
			@Override
			public void onClick(View v) {
				CalculaKillip.this.limpaTela();
			}
		});

		ImageButton botaoInfo = (ImageButton) findViewById(R.id.botaoInfo);
		botaoInfo.setOnClickListener(new View.OnClickListener() {
			@Override
			public void onClick(View v) {
				CalculaKillip.this.mostraPopupDescricao(getString(R.string.killip_formula));
			}
		});
	}

	private void calcularKillip(){
		String resultado = "";
		boolean exibirResultado = true;

		switch (this.campoClasseKillip.getCheckedRadioButtonId()) {
		case R.id.campoKillip1:
			resultado = getString(R.string.result_killip_1);
			break;
		case R.id.campoKillip2:
			resultado = getString(R.string.result_killip_2);
			break;
		case R.id.campoKillip3:
			resultado = getString(R.string.result_killip_3);
			break;
		case R.id.campoKillip4:
			resultado = getString(R.string.result_killip_4);
			break;
		default:
			exibirResultado = false;
```

```
		}

		if(exibirResultado){
			valorFinal.setText(resultado);
			this.exibeResultado();
		}else{
			this.escondeResultado();
		}
	}


	private void limpaTela(){
		this.campoClasseKillip.clearCheck();
		this.escondeResultado();
	}
}

public class CalculaLDL extends ActivityGeral {

	private EditText campoColesterolTotal, campoHDL, campoTriglicerideos;

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

		System.out.println("Criando Activity: CalculaLDL");

		this.campoColesterolTotal = (EditText) findViewById(R.id.campoColesterolTotal);
		this.campoHDL = (EditText) findViewById(R.id.campoHDL);
		this.campoTriglicerideos = (EditText) findViewById(R.id.campoTriglicerideos);
		this.labelResultado = (TextView) findViewById(R.id.labelResultado);
		this.valorFinal = (TextView) findViewById(R.id.valorFinal);

		Button botaoCalcular = (Button) findViewById(R.id.botaoCalcular);
		botaoCalcular.setOnClickListener(new View.OnClickListener() {
			@Override
			public void onClick(View v) {

	CalculaLDL.this.esconderTecladoVirtual(CalculaLDL.this.getCurrentFocus().getWindowToken());
				if(CalculaLDL.this.validaDados()){
					CalculaLDL.this.calcularLDL();
				}
			}
		});

		Button botaoVoltar = (Button) findViewById(R.id.botaoVoltar);
		botaoVoltar.setOnClickListener(new View.OnClickListener() {
			@Override
			public void onClick(View v) {
				startActivity(new Intent(getApplication(), ListaCalculos.class));
			}
		});

		Button botaoLimpar = (Button) findViewById(R.id.botaoLimpar);
		botaoLimpar.setOnClickListener(new View.OnClickListener() {
			@Override
			public void onClick(View v) {
				CalculaLDL.this.limpaTela();
			}
		});

		ImageButton botaoInfo = (ImageButton) findViewById(R.id.botaoInfo);
		botaoInfo.setOnClickListener(new View.OnClickListener() {
			@Override
			public void onClick(View v) {
				CalculaLDL.this.mostraPopupDescricao(getString(R.string.ldl_formula));
```

```
                }
            });
        }

        /**
         * Valida os dados inseridos pelo usuário. Se alguma variável não foi inserida destaca o campo em
vermelho e exibe alerta ao usuário
         */
        private boolean validaDados(){
            boolean dadosValidos = true;
            if(this.campoColesterolTotal.getText() == null ||
"".equals(this.campoColesterolTotal.getText().toString())){
                this.campoColesterolTotal.setError(getString(R.string.campo_obrigatorio));
                dadosValidos = false;
            }
            if(this.campoHDL.getText() == null || "".equals(this.campoHDL.getText().toString())){
                this.campoHDL.setError(getString(R.string.campo_obrigatorio));
                dadosValidos = false;
            }
            if(this.campoTriglicerideos.getText() == null ||
"".equals(this.campoTriglicerideos.getText().toString())){
                this.campoTriglicerideos.setError(getString(R.string.campo_obrigatorio));
                dadosValidos = false;
            }else{
                Integer triglicerideoInformado =
Integer.valueOf(this.campoTriglicerideos.getText().toString());
                if(triglicerideoInformado > 400){
                    this.campoTriglicerideos.setError(getString(R.string.erro_triglicerideos));
                    dadosValidos = false;
                }
            }
            return dadosValidos;
        }

        /**
         * Captura os valores inseridos pelo usuário, calcula o valor do LDL e exibe na tela
         */
        private void calcularLDL(){
            System.out.println("Dados válidos. Calculando LDL");

            Double colesterolTotal = Double.valueOf(this.campoColesterolTotal.getText().toString());
            Double hdl = Double.valueOf(this.campoHDL.getText().toString());
            Double triglicerideos = Double.valueOf(this.campoTriglicerideos.getText().toString());

            Double resultadoCalculado = ( colesterolTotal - hdl - (triglicerideos / 5));
            String resultadoArredondado = new DecimalFormat("0.00").format(resultadoCalculado);

            StringBuilder resultado = new StringBuilder();
            resultado.append(getString(R.string.ldl_sigla));
            resultado.append(" = ");
            resultado.append(resultadoArredondado);
            resultado.append(" ");
            resultado.append(getString(R.string.mg_dl));
            resultado.append(".\n");

            //http://blogbiotecnica.ind.br/blog/2012/02/ldl-colesterol/
            if(resultadoCalculado < 100){
                resultado.append(getString(R.string.result_ldl_1));
            } else if(resultadoCalculado < 130){
                resultado.append(getString(R.string.result_ldl_2));
            } else if(resultadoCalculado < 160){
                resultado.append(getString(R.string.result_ldl_3));
            } else if(resultadoCalculado < 190){
                resultado.append(getString(R.string.result_ldl_4));
            } else{
                resultado.append(getString(R.string.result_ldl_5));
            }
```

```
                this.valorFinal.setText(resultado);
                this.exibeResultado();
        }

        /**
         * Limpa a tela para um novo cálculo
         */
        private void limpaTela(){
                this.campoColesterolTotal.setText("");
                this.campoHDL.setText("");
                this.campoTriglicerideos.setText("");
                this.escondeResultado();
                this.campoColesterolTotal.requestFocus();
        }
}

public class CalculaMassaOssea extends ActivityGeral {

        private EditText campoPeso;
        private RadioGroup campoSexo;

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

                System.out.println("Criando Activity: CalculaMassaOssea");

                this.campoPeso = (EditText) findViewById(R.id.campoPesoMassaOssea);
                this.campoSexo = (RadioGroup) findViewById(R.id.campoSexoMassaOssea);
                this.labelResultado = (TextView) findViewById(R.id.labelResultado);
                this.valorFinal = (TextView) findViewById(R.id.valorFinal);

                Button botaoCalcular = (Button) findViewById(R.id.botaoCalcular);
                botaoCalcular.setOnClickListener(new View.OnClickListener() {
                        @Override
                        public void onClick(View v) {

        CalculaMassaOssea.this.esconderTecladoVirtual(CalculaMassaOssea.this.getCurrentFocus().getWindowTo
ken());
                                if(CalculaMassaOssea.this.validaDados()){
                                        CalculaMassaOssea.this.calcularMassaOssea();
                                }
                        }
                });

                Button botaoVoltar = (Button) findViewById(R.id.botaoVoltar);
                botaoVoltar.setOnClickListener(new View.OnClickListener() {
                        @Override
                        public void onClick(View v) {
                                startActivity(new Intent(getApplication(), ListaCalculos.class));
                        }
                });

                Button botaoLimpar = (Button) findViewById(R.id.botaoLimpar);
                botaoLimpar.setOnClickListener(new View.OnClickListener() {
                        @Override
                        public void onClick(View v) {
                                CalculaMassaOssea.this.limpaTela();
                        }
                });

                ImageButton botaoInfo = (ImageButton) findViewById(R.id.botaoInfo);
                botaoInfo.setOnClickListener(new View.OnClickListener() {
                        @Override
                        public void onClick(View v) {
```

```
CalculaMassaOssea.this.mostraPopupDescricao(getString(R.string.massa_ossea_formula));
                }
        });
}

/**
 * Valida os dados inseridos pelo usuário. Se alguma variável não foi inserida destaca o campo em
vermelho e exibe alerta ao usuário
 */
private boolean validaDados(){
        boolean dadosValidos = true;
        if(this.campoPeso.getText() == null || "".equals(this.campoPeso.getText().toString())){
                this.campoPeso.setError(getString(R.string.campo_obrigatorio));
                dadosValidos = false;
        }
        return dadosValidos;
}

/**
 * Captura os valores inseridos pelo usuário, calcula a massa óssea e exibe na tela
 */
private void calcularMassaOssea(){

        System.out.println("Dados válidos. Calculando Média de Massa Óssea Estimada");

        Double peso = Double.valueOf(this.campoPeso.getText().toString());
        boolean sexoMasculino = true;

        if(this.campoSexo.getCheckedRadioButtonId() == R.id.sexoFemMassaOssea){
                sexoMasculino = false; //feminino
        }

        StringBuilder resultado = new StringBuilder();
        resultado.append(getString(R.string.media_massa_ossea_estimada));
        resultado.append(": ");

        if(sexoMasculino){
                if(peso < 65){
                        resultado.append(getString(R.string.result_massa_ossea_1));
                } else if (peso < 95){
                        resultado.append(getString(R.string.result_massa_ossea_2));
                } else {
                        resultado.append(getString(R.string.result_massa_ossea_3));
                }
        } else {
                if(peso < 55){
                        resultado.append(getString(R.string.result_massa_ossea_4));
                } else if (peso < 75){
                        resultado.append(getString(R.string.result_massa_ossea_5));
                } else {
                        resultado.append(getString(R.string.result_massa_ossea_6));
                }
        }

        this.valorFinal.setText(resultado);
        this.exibeResultado();
}

/**
 * Limpa a tela para um novo cálculo
 */
private void limpaTela(){
        this.campoPeso.setText("");
        this.campoSexo.check(R.id.sexoMascMassaOssea);
        this.escondeResultado();
        this.campoPeso.requestFocus();
```

```
	}
}

public class CalculaMeld extends ActivityGeral {

	private EditText campoBilirrubina, campoCreatinina, campoINR;

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

		System.out.println("Criando Activity: CalculaMeld");

		this.campoBilirrubina = (EditText) findViewById(R.id.campoBilirrubina);
		this.campoCreatinina = (EditText) findViewById(R.id.campoCreatinina);
		this.campoINR = (EditText) findViewById(R.id.campoINR);
		this.labelResultado = (TextView) findViewById(R.id.labelResultado);
		this.valorFinal = (TextView) findViewById(R.id.valorFinal);

		Button botaoCalcular = (Button) findViewById(R.id.botaoCalcular);
		botaoCalcular.setOnClickListener(new View.OnClickListener() {
			@Override
			public void onClick(View v) {

	CalculaMeld.this.esconderTecladoVirtual(CalculaMeld.this.getCurrentFocus().getWindowToken());
				if(CalculaMeld.this.validaDados()){
					CalculaMeld.this.calcularMeld();
				}
			}
		});

		Button botaoVoltar = (Button) findViewById(R.id.botaoVoltar);
		botaoVoltar.setOnClickListener(new View.OnClickListener() {
			@Override
			public void onClick(View v) {
				startActivity(new Intent(getApplication(), ListaCalculos.class));
			}
		});

		Button botaoLimpar = (Button) findViewById(R.id.botaoLimpar);
		botaoLimpar.setOnClickListener(new View.OnClickListener() {
			@Override
			public void onClick(View v) {
				CalculaMeld.this.limpaTela();
			}
		});

		ImageButton botaoInfo = (ImageButton) findViewById(R.id.botaoInfo);
		botaoInfo.setOnClickListener(new View.OnClickListener() {
			@Override
			public void onClick(View v) {
				CalculaMeld.this.mostraPopupDescricao(getString(R.string.meld_formula));
			}
		});
	}

	/**
	 * Valida os dados inseridos pelo usuário. Se alguma variável não foi inserida destaca o campo em
vermelho e exibe alerta ao usuário
	 */
	private boolean validaDados(){
		boolean dadosValidos = true;
		if(this.campoBilirrubina.getText() == null ||
"".equals(this.campoBilirrubina.getText().toString())){
			this.campoBilirrubina.setError(getString(R.string.campo_obrigatorio));
			dadosValidos = false;
```

```
                }
                if(this.campoCreatinina.getText() == null || "".equals(this.campoCreatinina.getText().toString())){
                        this.campoCreatinina.setError(getString(R.string.campo_obrigatorio));
                        dadosValidos = false;
                }
                if(this.campoINR.getText() == null || "".equals(this.campoINR.getText().toString())){
                        this.campoINR.setError(getString(R.string.campo_obrigatorio));
                        dadosValidos = false;
                }
                return dadosValidos;
        }

        /**
         * Captura os valores inseridos pelo usuário, calcula o valor do Anion Gap e exibe na tela
         */
        private void calcularMeld(){

                System.out.println("Dados válidos. Calculando Escala Meld");

                Double bilirrubina = Double.valueOf(this.campoBilirrubina.getText().toString());
                Double creatinina = Double.valueOf(this.campoCreatinina.getText().toString());
                Double inr = Double.valueOf(this.campoINR.getText().toString());

                Double resultadoCalculado = 3.78 * Math.log(bilirrubina) + 11.2 * Math.log(inr) + 9.57 *
Math.log(creatinina) + 6.43;

                if(resultadoCalculado.equals(Double.NEGATIVE_INFINITY) ||
resultadoCalculado.equals(Double.POSITIVE_INFINITY)){
                        resultadoCalculado = 0.0;
                }

                Double resultadoArredondado = Math.floor(resultadoCalculado);
                String resultadoFormatado = new DecimalFormat("0").format(resultadoArredondado);

                StringBuilder resultado = new StringBuilder();
                resultado.append(getString(R.string.valor_escala_meld));
                resultado.append(" = ");
                resultado.append(resultadoFormatado);
                resultado.append(".\n");

                if(resultadoArredondado > 40){
                        resultado.append(getString(R.string.result_meld_1));
                } else if (resultadoArredondado > 30){
                        resultado.append(getString(R.string.result_meld_2));
                } else if (resultadoArredondado > 20){
                        resultado.append(getString(R.string.result_meld_3));
                } else if (resultadoArredondado > 10){
                        resultado.append(getString(R.string.result_meld_4));
                } else {
                        resultado.append(getString(R.string.result_meld_5));
                }

                this.valorFinal.setText(resultado);
                this.exibeResultado();
        }

        /**
         * Limpa a tela para um novo cálculo
         */
        private void limpaTela(){
                this.campoBilirrubina.setText("");
                this.campoCreatinina.setText("");
                this.campoINR.setText("");
                this.escondeResultado();
                this.campoBilirrubina.requestFocus();
        }
}
```

```
public class CalculaParkland extends ActivityGeral {

	private EditText campoPeso, campoAreaQueimada;

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

		System.out.println("Criando Activity: CalculaParkland");

		this.campoPeso = (EditText) findViewById(R.id.campoPeso);
		this.campoAreaQueimada = (EditText) findViewById(R.id.campoAreaQueimada);
		this.labelResultado = (TextView) findViewById(R.id.labelResultado);
		this.valorFinal = (TextView) findViewById(R.id.valorFinal);

		Button botaoCalcular = (Button) findViewById(R.id.botaoCalcular);
		botaoCalcular.setOnClickListener(new View.OnClickListener() {
			@Override
			public void onClick(View arg0) {

	CalculaParkland.this.esconderTecladoVirtual(CalculaParkland.this.getCurrentFocus().getWindowToken());
				if(CalculaParkland.this.validaDados()){
					CalculaParkland.this.calcularParkland();
				}
			}
		});

		Button botaoVoltar = (Button) findViewById(R.id.botaoVoltar);
		botaoVoltar.setOnClickListener(new View.OnClickListener() {
			@Override
			public void onClick(View v) {
				startActivity(new Intent(getApplication(), ListaCalculos.class));
			}
		});

		Button botaoLimpar = (Button) findViewById(R.id.botaoLimpar);
		botaoLimpar.setOnClickListener(new View.OnClickListener() {
			@Override
			public void onClick(View v) {
				CalculaParkland.this.limpaTela();
			}
		});

		ImageButton botaoInfo = (ImageButton) findViewById(R.id.botaoInfo);
		botaoInfo.setOnClickListener(new View.OnClickListener() {
			@Override
			public void onClick(View v) {

	CalculaParkland.this.mostraPopupDescricao(getString(R.string.parkland_formula));
			}
		});
	}

	/**
	 * Valida os dados inseridos pelo usuário. Se alguma variável não foi inserida destaca o campo em
vermelho e exibe alerta ao usuário
	 */
	private boolean validaDados(){
		boolean dadosValidos = true;
		if(this.campoPeso.getText() == null || "".equals(this.campoPeso.getText().toString())){
			this.campoPeso.setError(getString(R.string.campo_obrigatorio));
			dadosValidos = false;
		}
		if(this.campoAreaQueimada.getText() == null ||
"".equals(this.campoAreaQueimada.getText().toString())){
```

```
                this.campoAreaQueimada.setError(getString(R.string.campo_obrigatorio));
                dadosValidos = false;
            } else{
                Integer area = Integer.valueOf(this.campoAreaQueimada.getText().toString());
                if(area < 0 || area > 100){
                    this.campoAreaQueimada.setError(getString(R.string.valor_entre_0_e_100));
                    dadosValidos = false;
                }
            }
            return dadosValidos;
        }

        private void calcularParkland(){
            System.out.println("Dados válidos. Calculando AnionGap");

            Double peso = Double.valueOf(this.campoPeso.getText().toString());
            Double areaQueimada = Double.valueOf(this.campoAreaQueimada.getText().toString());

            Double resultadoTotal = 0.004 * peso * areaQueimada;

            String resultadoTotalArredondado = new DecimalFormat("0.00").format(resultadoTotal);
            String resultadoPorDoisIntervalos = new DecimalFormat("0.00").format(resultadoTotal/2);

            StringBuilder resultado = new StringBuilder();
            resultado.append(getString(R.string.requerido));
            resultado.append(" ");
            resultado.append(resultadoTotalArredondado);
            resultado.append(" ");
            resultado.append(getString(R.string.litros_soro_24_horas));
            resultado.append(" ");
            resultado.append(resultadoPorDoisIntervalos);
            resultado.append(" ");
            resultado.append(getString(R.string.litros_8_horas));
            resultado.append(" ");
            resultado.append(resultadoPorDoisIntervalos);
            resultado.append(" ");
            resultado.append(getString(R.string.litros_16_horas));

            this.valorFinal.setText(resultado);
            this.exibeResultado();
        }

        /**
         * Limpa a tela para um novo cálculo
         */
        private void limpaTela(){
            this.campoPeso.setText("");
            this.campoAreaQueimada.setText("");
            this.escondeResultado();
            this.campoPeso.requestFocus();
        }
}

public class CalculaRamsay extends ActivityGeral {

        private RadioGroup campoGrauSedacao;

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

            System.out.println("Criando Activity: CalculaRamsay");

            this.campoGrauSedacao= (RadioGroup) findViewById(R.id.radioGroupGrauSedacaoPaciente);
            this.labelResultado = (TextView) findViewById(R.id.labelResultado);
            this.valorFinal = (TextView) findViewById(R.id.valorFinal);
```

```
		this.campoGrauSedacao.setOnCheckedChangeListener(new
RadioGroup.OnCheckedChangeListener() {
			@Override
			public void onCheckedChanged(RadioGroup group, int checkedId) {
				CalculaRamsay.this.calcularRamsay();
			}
		});

		Button botaoVoltar = (Button) findViewById(R.id.botaoVoltar);
		botaoVoltar.setOnClickListener(new View.OnClickListener() {
			@Override
			public void onClick(View v) {
				startActivity(new Intent(getApplication(), ListaCalculos.class));
			}
		});

		Button botaoLimpar = (Button) findViewById(R.id.botaoLimpar);
		botaoLimpar.setOnClickListener(new View.OnClickListener() {
			@Override
			public void onClick(View v) {
				CalculaRamsay.this.limpaTela();
			}
		});

		ImageButton botaoInfo = (ImageButton) findViewById(R.id.botaoInfo);
		botaoInfo.setOnClickListener(new View.OnClickListener() {
			@Override
			public void onClick(View v) {

	CalculaRamsay.this.mostraPopupDescricao(getString(R.string.formula_ramsay));
			}
		});
	}

	private void calcularRamsay(){
		String resultado = "";
		boolean exibirResultado = true;

		switch (this.campoGrauSedacao.getCheckedRadioButtonId()) {
		case R.id.campoGrauSedacao1:
			resultado = getString(R.string.result_ramsay_1);
			break;
		case R.id.campoGrauSedacao2:
			resultado = getString(R.string.result_ramsay_2);
			break;
		case R.id.campoGrauSedacao3:
			resultado = getString(R.string.result_ramsay_3);
			break;
		case R.id.campoGrauSedacao4:
			resultado = getString(R.string.result_ramsay_4);
			break;
		case R.id.campoGrauSedacao5:
			resultado = getString(R.string.result_ramsay_5);
			break;
		case R.id.campoGrauSedacao6:
			resultado = getString(R.string.result_ramsay_6);
			break;
		default:
			exibirResultado = false;
		}

		if(exibirResultado){
			valorFinal.setText(resultado);
			this.exibeResultado();
		} else {
			this.escondeResultado();
```

```
		}
	}

	private void limpaTela(){
		this.campoGrauSedacao.clearCheck();
		this.escondeResultado();
	}
}

public class CalculaRiscoPneumoniaCurb extends ActivityGeral {

	private ToggleButton toggleConfusao, toggleUreia, toggleRespiracao, togglePressaoSanguinea,
toggleIdade;

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

		System.out.println("Criando Activity: CalculaRiscoPneumoniaCurb");

		this.toggleConfusao = (ToggleButton) findViewById(R.id.toggleConfusao);
		this.toggleUreia = (ToggleButton) findViewById(R.id.toggleUreia);
		this.toggleRespiracao = (ToggleButton) findViewById(R.id.toggleRespiracao);
		this.togglePressaoSanguinea = (ToggleButton) findViewById(R.id.togglePressaoSanguinea);
		this.toggleIdade = (ToggleButton) findViewById(R.id.toggleIdade);
		this.labelResultado = (TextView) findViewById(R.id.labelResultado);
		this.valorFinal = (TextView) findViewById(R.id.valorFinal);

		EventoToggle eventoToggle = new EventoToggle();

		this.toggleConfusao.setOnCheckedChangeListener(eventoToggle);
		this.toggleUreia.setOnCheckedChangeListener(eventoToggle);
		this.toggleRespiracao.setOnCheckedChangeListener(eventoToggle);
		this.togglePressaoSanguinea.setOnCheckedChangeListener(eventoToggle);
		this.toggleIdade.setOnCheckedChangeListener(eventoToggle);

		Button botaoVoltar = (Button) findViewById(R.id.botaoVoltar);
		botaoVoltar.setOnClickListener(new View.OnClickListener() {
			@Override
			public void onClick(View v) {
				startActivity(new Intent(getApplication(), ListaCalculos.class));
			}
		});

		Button botaoLimpar = (Button) findViewById(R.id.botaoLimpar);
		botaoLimpar.setOnClickListener(new View.OnClickListener() {
			@Override
			public void onClick(View v) {
				CalculaRiscoPneumoniaCurb.this.limpaTela();
			}
		});

		ImageButton botaoInfo = (ImageButton) findViewById(R.id.botaoInfo);
		botaoInfo.setOnClickListener(new View.OnClickListener() {
			@Override
			public void onClick(View v) {

	CalculaRiscoPneumoniaCurb.this.mostraPopupDescricao(getString(R.string.formula_curb));
			}
		});
	}

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

```
	}

	private void calcularRiscoCurb(){
		System.out.println("Calculando RiscoPneumoniaCurb");

		int confusao = (this.toggleConfusao.isChecked() ? 1 : 0);
		int ureia = (this.toggleUreia.isChecked() ? 1 : 0);
		int respiracao = (this.toggleRespiracao.isChecked() ? 1 : 0);
		int pressaoSanguinea = (this.togglePressaoSanguinea.isChecked() ? 1 : 0);
		int idade = (this.toggleIdade.isChecked() ? 1 : 0);

		int total = confusao + ureia + respiracao + pressaoSanguinea + idade;
		String resultado = "";

		switch (total) {
		case 0:
			resultado = getString(R.string.result_curb_1);
			break;
		case 1:
			resultado = getString(R.string.result_curb_2);
			break;
		case 2:
			resultado = getString(R.string.result_curb_3);
			break;
		case 3:
			resultado = getString(R.string.result_curb_4);
			break;
		case 4:
			resultado = getString(R.string.result_curb_5);
			break;
		default:
			resultado = getString(R.string.result_curb_6);
			break;
		}

		this.valorFinal.setText(resultado);
		this.exibeResultado();
	}

	/**
	 * Limpa a tela para um novo cálculo
	 */
	private void limpaTela(){
		this.toggleConfusao.setChecked(false);
		this.toggleUreia.setChecked(false);
		this.toggleRespiracao.setChecked(false);
		this.togglePressaoSanguinea.setChecked(false);
		this.toggleIdade.setChecked(false);
		this.calcularRiscoCurb();
	}

	/**
	 * Classe interna que representa o evento para calcular o risco de pneumonia CURB-65. Este evento é
disparado quando o ToggleButton
	 * é clicado.
	 */
	private class EventoToggle implements CompoundButton.OnCheckedChangeListener{
		@Override
		public void onCheckedChanged(CompoundButton buttonView,
				boolean isChecked) {
			CalculaRiscoPneumoniaCurb.this.calcularRiscoCurb();

		}
	}
}

public class CalculaTesteCooper extends ActivityGeral {
```

```
	private EditText campoIdade, campoDistancia;
	private RadioGroup campoSexo;

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

		System.out.println("Criando Activity: CalculaTesteCooper");

		this.campoIdade = (EditText) findViewById(R.id.campoIdadeCooper);
		this.campoDistancia = (EditText) findViewById(R.id.campoDistanciaCooper);
		this.campoSexo = (RadioGroup) findViewById(R.id.campoSexoCooper);
		this.labelResultado = (TextView) findViewById(R.id.labelResultado);
		this.valorFinal = (TextView) findViewById(R.id.valorFinal);

		Button botaoCalcular = (Button) findViewById(R.id.botaoCalcular);
		botaoCalcular.setOnClickListener(new View.OnClickListener() {
			@Override
			public void onClick(View v) {

	CalculaTesteCooper.this.esconderTecladoVirtual(CalculaTesteCooper.this.getCurrentFocus().getWindowT
oken());
				if(CalculaTesteCooper.this.validaDados()){
					CalculaTesteCooper.this.calcularTesteCooper();
				}
			}
		});

		Button botaoVoltar = (Button) findViewById(R.id.botaoVoltar);
		botaoVoltar.setOnClickListener(new View.OnClickListener() {
			@Override
			public void onClick(View v) {
				startActivity(new Intent(getApplication(), ListaCalculos.class));
			}
		});

		Button botaoLimpar = (Button) findViewById(R.id.botaoLimpar);
		botaoLimpar.setOnClickListener(new View.OnClickListener() {
			@Override
			public void onClick(View v) {
				CalculaTesteCooper.this.limpaTela();
			}
		});

		ImageButton botaoInfo = (ImageButton) findViewById(R.id.botaoInfo);
		botaoInfo.setOnClickListener(new View.OnClickListener() {
			@Override
			public void onClick(View v) {

	CalculaTesteCooper.this.mostraPopupDescricao(getString(R.string.formula_cooper));
			}
		});
	}

	/**
	 * Valida os dados inseridos pelo usuário. Se alguma variável não foi inserida destaca o campo em
vermelho e exibe alerta ao usuário
	 */
	private boolean validaDados(){
		boolean dadosValidos = true;
		if(this.campoIdade.getText() == null || "".equals(this.campoIdade.getText().toString())){
			this.campoIdade.setError(getString(R.string.campo_obrigatorio));
			dadosValidos = false;
		}
		if(this.campoDistancia.getText() == null || "".equals(this.campoDistancia.getText().toString())){
```

```
                this.campoDistancia.setError(getString(R.string.campo_obrigatorio));
                dadosValidos = false;
        }
        return dadosValidos;
}

/**
 * Captura os valores inseridos pelo usuário, calcula a capacidade aerobica (VO2) e exibe na tela
 */
private void calcularTesteCooper(){

        System.out.println("Dados válidos. Calculando Teste de Cooper");

        Integer idade = Integer.valueOf(this.campoIdade.getText().toString());
        Integer distancia = Integer.valueOf(this.campoDistancia.getText().toString());
        boolean sexoMasculino = true;

        if(this.campoSexo.getCheckedRadioButtonId() == R.id.sexoFemCooper){
                sexoMasculino = false; //feminino
        }

        StringBuilder resultadoVO2 = new StringBuilder();
        resultadoVO2.append(getString(R.string.vo2_max));
        resultadoVO2.append(" = ");
        resultadoVO2.append(new DecimalFormat("0.00").format((distancia - 504.9)/44.73));
        resultadoVO2.append(" ");
        resultadoVO2.append(getString(R.string.l_min));

        StringBuilder resultadoTeste = new StringBuilder();
        resultadoTeste.append(getString(R.string.resultado_cooper));
        resultadoTeste.append(": ");

        if(sexoMasculino){
                if(idade < 30){
                        if(distancia < 1600) resultadoTeste.append(getString(R.string.muito_ruim));
                        else if(distancia < 2200) resultadoTeste.append(getString(R.string.ruim));
                        else if(distancia < 2400) resultadoTeste.append(getString(R.string.regular));
                        else if(distancia < 2800) resultadoTeste.append(getString(R.string.bom));
                        else resultadoTeste.append(getString(R.string.otimo));
                } else if(idade < 40){
                        if(distancia < 1500) resultadoTeste.append(getString(R.string.muito_ruim));
                        else if(distancia < 2000) resultadoTeste.append(getString(R.string.ruim));
                        else if(distancia < 2300) resultadoTeste.append(getString(R.string.regular));
                        else if(distancia < 2700) resultadoTeste.append(getString(R.string.bom));
                        else resultadoTeste.append(getString(R.string.otimo));
                } else if(idade < 50){
                        if(distancia < 1400) resultadoTeste.append(getString(R.string.muito_ruim));
                        else if(distancia < 1700) resultadoTeste.append(getString(R.string.ruim));
                        else if(distancia < 2100)resultadoTeste.append(getString(R.string.regular));
                        else if(distancia < 2500) resultadoTeste.append(getString(R.string.bom));
                        else resultadoTeste.append(getString(R.string.otimo));
                } else {
                        if(distancia < 1300) resultadoTeste.append(getString(R.string.muito_ruim));
                        else if(distancia < 1600)resultadoTeste.append(getString(R.string.ruim));
                        else if(distancia < 2000) resultadoTeste.append(getString(R.string.regular));
                        else if(distancia < 2400) resultadoTeste.append(getString(R.string.bom));
                        else resultadoTeste.append(getString(R.string.otimo));
                }
        } else{
                if(idade < 30){
                        if(distancia < 1500) resultadoTeste.append(getString(R.string.muito_ruim));
                        else if(distancia < 1800) resultadoTeste.append(getString(R.string.ruim));
                        else if(distancia < 2200) resultadoTeste.append(getString(R.string.regular));
                        else if(distancia < 2700) resultadoTeste.append(getString(R.string.bom));
                        else resultadoTeste.append(getString(R.string.otimo));
                } else if(idade < 40){
                        if(distancia < 1400) resultadoTeste.append(getString(R.string.muito_ruim));
```

```
                                        else if(distancia < 1700) resultadoTeste.append(getString(R.string.ruim));
                                        else if(distancia < 2000) resultadoTeste.append(getString(R.string.regular));
                                        else if(distancia < 2500) resultadoTeste.append(getString(R.string.bom));
                                        else resultadoTeste.append(getString(R.string.otimo));
                                } else if(idade < 50){
                                        if(distancia < 1200) resultadoTeste.append(getString(R.string.muito_ruim));
                                        else if(distancia < 1500) resultadoTeste.append(getString(R.string.ruim));
                                        else if(distancia < 1900) resultadoTeste.append(getString(R.string.regular));
                                        else if(distancia < 2300) resultadoTeste.append(getString(R.string.bom));
                                        else resultadoTeste.append(getString(R.string.otimo));
                                } else {
                                        if(distancia < 1100) resultadoTeste.append(getString(R.string.muito_ruim));
                                        else if(distancia < 1400) resultadoTeste.append(getString(R.string.ruim));
                                        else if(distancia < 1700) resultadoTeste.append(getString(R.string.regular));
                                        else if(distancia < 2200) resultadoTeste.append(getString(R.string.bom));
                                        else resultadoTeste.append(getString(R.string.otimo));
                                }
                        }

                        String resultado = resultadoVO2.toString() + "\n" + resultadoTeste.toString();
                        this.valorFinal.setText(resultado);
                        this.exibeResultado();
                }

                /**
                 * Limpa a tela para um novo cálculo
                 */
                private void limpaTela(){
                        this.campoIdade.setText("");
                        this.campoDistancia.setText("");
                        this.campoSexo.check(R.id.sexoMascCooper);
                        this.escondeResultado();
                        this.campoIdade.requestFocus();
                }
}

public class CalculaZonasAlvo extends ActivityGeral {

        private EditText campoIdade;

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

                System.out.println("Criando Activity: CalculaZonasAlvo");

                this.campoIdade = (EditText) findViewById(R.id.campoIdadeZonaAlvo);
                this.labelResultado = (TextView) findViewById(R.id.labelResultado);
                this.valorFinal = (TextView) findViewById(R.id.valorFinal);

                Button botaoCalcular = (Button) findViewById(R.id.botaoCalcular);
                botaoCalcular.setOnClickListener(new View.OnClickListener() {
                        @Override
                        public void onClick(View v) {

        CalculaZonasAlvo.this.esconderTecladoVirtual(CalculaZonasAlvo.this.getCurrentFocus().getWindowToke
n());
                                if(CalculaZonasAlvo.this.validaDados()){
                                        CalculaZonasAlvo.this.calcularZonasAlvo();
                                }
                        }
                });

                Button botaoVoltar = (Button) findViewById(R.id.botaoVoltar);
                botaoVoltar.setOnClickListener(new View.OnClickListener() {
                        @Override
```

```
                public void onClick(View v) {
                        startActivity(new Intent(getApplication(), ListaCalculos.class));
                }
        });

        Button botaoLimpar = (Button) findViewById(R.id.botaoLimpar);
        botaoLimpar.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View v) {
                        CalculaZonasAlvo.this.limpaTela();
                }
        });

        ImageButton botaoInfo = (ImageButton) findViewById(R.id.botaoInfo);
        botaoInfo.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View v) {

CalculaZonasAlvo.this.mostraPopupDescricao(getString(R.string.formula_zona));
                }
        });
}

/**
 * Valida os dados inseridos pelo usuário. Se alguma variável não foi inserida destaca o campo em
vermelho e exibe alerta ao usuário
 */
private boolean validaDados(){
        boolean dadosValidos = true;
        if(this.campoIdade.getText() == null || "".equals(this.campoIdade.getText().toString())){
                this.campoIdade.setError(getString(R.string.campo_obrigatorio));
                dadosValidos = false;
        }
        return dadosValidos;
}

/**
 * Captura os valores inseridos pelo usuário, calcula as zonas alvo e exibe na tela
 */
private void calcularZonasAlvo(){

        System.out.println("Dados válidos. Calculando ZonasAlvo");

        Integer idade = Integer.valueOf(this.campoIdade.getText().toString());

        Integer frequenciaMaxima = 220 - idade;
        Double zonaAtividadeModeradaMinima = (frequenciaMaxima * 0.5);
        Double zonaAtividadeModeradaMaxima = frequenciaMaxima * 0.6;
        Double zonaControlePesoMinima = frequenciaMaxima * 0.6;
        Double zonaControlePesoMaxima = frequenciaMaxima * 0.7;
        Double zonaAerobicaMinima = frequenciaMaxima * 0.7;
        Double zonaAerobicaMaxima = frequenciaMaxima * 0.8;
        Double zonaLimiarAnaerobicoMinimo = frequenciaMaxima * 0.8;
        Double zonaLimiarAnaerobicoMaximo = frequenciaMaxima * 0.9;
        Double zonaEsforcoMaximoMinimo = frequenciaMaxima * 0.9;
        Integer zonaEsforcoMaximoMaximo = frequenciaMaxima;

        StringBuilder resultado = new StringBuilder();
        resultado.append(getString(R.string.result_zona_alvo));
        resultado.append(getString(R.string.freq_cardiaca_max));
        resultado.append(": ");
        resultado.append(frequenciaMaxima);
        resultado.append(".\n");

        resultado.append(getString(R.string.esforco_max));
        resultado.append(": ");
        resultado.append(getString(R.string.entre));
```

```
                resultado.append(" ");
                resultado.append(zonaEsforcoMaximoMinimo.intValue());
                resultado.append(" ");
                resultado.append(getString(R.string.e));
                resultado.append(" ");
                resultado.append(zonaEsforcoMaximoMaximo.intValue());
                resultado.append(".\n");

                resultado.append(getString(R.string.limiar_aerobico));
                resultado.append(": ");
                resultado.append(getString(R.string.entre));
                resultado.append(" ");
                resultado.append(zonaLimiarAnaerobicoMinimo.intValue());
                resultado.append(" ");
                resultado.append(getString(R.string.e));
                resultado.append(" ");
                resultado.append(zonaLimiarAnaerobicoMaximo.intValue());
                resultado.append(".\n");

                resultado.append(getString(R.string.aerobica));
                resultado.append(": ");
                resultado.append(getString(R.string.entre));
                resultado.append(" ");
                resultado.append(zonaAerobicaMinima.intValue());
                resultado.append(" ");
                resultado.append(getString(R.string.e));
                resultado.append(" ");
                resultado.append(zonaAerobicaMaxima.intValue());
                resultado.append(".\n");

                resultado.append(getString(R.string.controle_peso));
                resultado.append(": ");
                resultado.append(getString(R.string.entre));
                resultado.append(" ");
                resultado.append(zonaControlePesoMinima.intValue());
                resultado.append(" ");
                resultado.append(getString(R.string.e));
                resultado.append(" ");
                resultado.append(zonaControlePesoMaxima.intValue());
                resultado.append(".\n");

                resultado.append(getString(R.string.atividade_moderada));
                resultado.append(": ");
                resultado.append(getString(R.string.entre));
                resultado.append(" ");
                resultado.append(zonaAtividadeModeradaMinima.intValue());
                resultado.append(" ");
                resultado.append(getString(R.string.e));
                resultado.append(" ");
                resultado.append(zonaAtividadeModeradaMaxima.intValue());
                resultado.append(".\n");


                this.valorFinal.setText(resultado.toString());
                this.exibeResultado();
        }

        /**
         * Limpa a tela para um novo cálculo
         */
        private void limpaTela(){
                this.campoIdade.setText("");
                this.escondeResultado();
                this.campoIdade.requestFocus();
        }
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="activity.calculosmedicos" >
    <uses-sdk
        android:minSdkVersion="10"
        android:targetSdkVersion="16" />
    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="activity.calculosmedicos.CarregarInicial"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
            android:name="activity.calculosmedicos.TelaInicial"
            android:label="@string/title_activity_tela_inicial" >
        </activity>
        <activity android:name="activity.calculosmedicos.ListaCalculos" >
        </activity>
        <activity
            android:name="activity.calculosmedicos.calculo.CalculaAnionGap"
            android:label="@string/anion_gap" >
        </activity>
        <activity android:name="activity.calculosmedicos.calculo.ActivityGeral" >
        </activity>
        <activity
            android:name="activity.calculosmedicos.calculo.CalculaRiscoPneumoniaCurb"
            android:label="@string/risco_curb" >
        </activity>
        <activity
            android:name="activity.calculosmedicos.calculo.CalculaParkland"
            android:label="@string/formula_parkland" >
        </activity>
        <activity
            android:name="activity.calculosmedicos.calculo.CalculaMeld"
            android:label="@string/escala_meld" >
        </activity>
        <activity
            android:name="activity.calculosmedicos.calculo.CalculaCockroftGault"
            android:label="@string/cockroft_gault" >
        </activity>
        <activity
            android:name="activity.calculosmedicos.calculo.CalculaGlasgow"
            android:label="@string/escala_glasgow" >
        </activity>
        <activity
            android:name="activity.calculosmedicos.calculo.CalculaApgar"
            android:label="@string/escala_apgar" >
        </activity>
        <activity
            android:name="activity.calculosmedicos.calculo.CalculaChildPugh"
            android:label="@string/classificacao_child_pugh" >
        </activity>
        <activity
            android:name="activity.calculosmedicos.calculo.CalculaIMC"
            android:label="@string/IMC" >
        </activity>
        <activity
            android:name="activity.calculosmedicos.calculo.CalculaLDL"
            android:label="@string/LDL" >
```

```
        </activity>
        <activity
            android:name="activity.calculosmedicos.calculo.CalculaRamsay"
            android:label="@string/escala_ramsay" >
        </activity>
        <activity
            android:name="activity.calculosmedicos.calculo.CalculaAshworth"
            android:label="@string/escala_ashworth" >
        </activity>
        <activity
            android:name="activity.calculosmedicos.calculo.CalculaForrest"
            android:label="@string/classificacao_forrest" >
        </activity>
        <activity
            android:name="activity.calculosmedicos.calculo.CalculaKillip"
            android:label="@string/classificacao_killip" >
        </activity>
        <activity
            android:name="activity.calculosmedicos.calculo.CalculaCriterioRanson"
            android:label="@string/criterio_ranson" >
        </activity>
        <activity
            android:name="activity.calculosmedicos.calculo.CalculaZonasAlvo"
            android:label="@string/zona_aerobica" >
        </activity>
        <activity
            android:name="activity.calculosmedicos.calculo.CalculaTesteCooper"
            android:label="@string/teste_cooper" >
        </activity>
        <activity
            android:name="activity.calculosmedicos.calculo.CalculaGorduraCorporal"
            android:label="@string/gordura_corporal" >
        </activity>
        <activity
            android:name="activity.calculosmedicos.calculo.CalculaMassaOssea"
            android:label="@string/massa_ossea" >
        </activity>
        <activity
            android:name="activity.calculosmedicos.SelecionarCalculos"
            android:label="@string/selecionar_categoria" >
        </activity>
        <activity
            android:name="activity.calculosmedicos.InformacoesApp"
            android:label="@string/informacoes" >
        </activity>
    </application>
</manifest>

<?xml version="1.0" encoding="utf-8"?>
<resources>

    <string name="app_name">Cálculos Médicos</string>
    <string name="calcular">Calcular</string>
    <string name="calculos">Cálculos</string>
    <string name="calculos_disponiveis">Cálculos Disponíveis</string>
    <string name="carregando">Carregando</string>
    <string name="formula">Fórmula</string>
    <string name="informacoes">Informações</string>
    <string name="limpar">Limpar</string>
    <string name="menu_settings">Settings</string>
    <string name="nao_calculado">Não Calculado</string>
    <string name="resultado">Resultado:</string>
    <string name="title_activity_tela_inicial">Bem Vindo!</string>
    <string name="voltar">Voltar</string>
    <string name="finalizar">Finalizar</string>
    <string name="campo_obrigatorio">Campo Obrigatório</string>
    <string name="selecione_item_opcao">Para obter um resultado, selecione um item de cada critério.</string>
```

```
    <string name="categoria_todos">Todos</string>
    <string name="categoria_educacao_fisica">Educação Física</string>
    <string name="categoria_fisioterapia">Fisioterapia</string>
    <string name="categoria_medicina">Medicina</string>
    <string name="selecionar_categoria">Selecione a categoria</string>
    <string name="descricao">Descrição</string>

    <!-- Informações -->
    <string name="info_1">Este aplicativo foi desenvolvido com o intuito de auxiliar profissionais da área da saúde.</string>
    <string name="info_2">Para cumprir este objetivo, disponibilizamos ferramentas para cálculos, classificações e escalas, separados por categorias de atuação.</string>
    <string name="desenvolvido_por">Desenvolvido por @HenioJR</string>
    <string name="contato">Dúvidas ou sugestões, enviar e-mail para heniobezjr@gmail.com</string>

    <!-- Unidades de medida -->
    <string name="mmol_l">mmol/L</string>
    <string name="mL_min">mL/min</string>
    <string name="mg_dl">mg/dL</string>
    <string name="l_min">l/min</string>

    <!-- Lista de Activities -->
    <string name="anion_gap">Anion Gap</string>
    <string name="risco_curb">Risco de Pneumonia CURB-65</string>
    <string name="formula_parkland">Fórmula de Parkland</string>
    <string name="escala_meld">Escala MELD</string>
    <string name="cockroft_gault">Fórmula de Cockroft-Gault</string>
    <string name="escala_glasgow">Escala de Coma Glasgow</string>
    <string name="escala_apgar">Escala de Apgar</string>
    <string name="classificacao_child_pugh">Classificação Child Pugh</string>
    <string name="IMC">IMC (Índice Massa Corporal)</string>
    <string name="LDL">LDL (Colesterol)</string>
    <string name="escala_ramsay">Escala de Ramsay</string>
    <string name="escala_ashworth">Escala Ashworth Modificada</string>
    <string name="classificacao_forrest">Classificação de Forrest</string>
    <string name="classificacao_killip">Classificação de Killip</string>
    <string name="criterio_ranson">Critério de Ranson</string>
    <string name="zona_aerobica">Zona Aeróbica Treinamento</string>
    <string name="teste_cooper">Teste de Cooper (12 minutos)</string>
    <string name="gordura_corporal">Margem de Gordura Corporal</string>
    <string name="massa_ossea">Massa Óssea Estimada</string>

    <!-- Activity CalculaAnionGap -->
    <string name="label_nivel_sodio">Nível de Sódio [mmol/L]</string>
    <string name="label_nivel_cloro">Nível de Cloro [mmol/L]</string>
    <string name="label_nivel_bicarbonato">Nível de Bicarbonato [mmol/L]</string>
    <string name="anion_gap_formula">Também conhecido como intervalo aniônico, representa a diferença entre cátions (Sódio) e ânions (Bicarbonato e Cloro) presentes no sangue.\n\nAnion Gap = nível Sódio - (nível Cloro + nível Bicarbonato)</string>

    <!-- Activity CalculaApgar -->
    <string name="label_freq_cardiaca">Frequência Cardíaca</string>
    <string name="label_ausente">Ausente</string>
    <string name="label_menor_cem">&amp;lt; 100/minuto</string>
    <string name="label_maior_cem">&amp;gt; 100/minuto</string>
    <string name="label_respiracao">Respiração</string>
    <string name="label_irregular_brad">Irregular / Bradipnéia</string>
    <string name="label_forte_choro">Forte / Choro</string>
    <string name="label_tonus_muscular">Tônus Muscular</string>
    <string name="label_flacido">Flácido</string>
    <string name="label_flexao_pernas_bracos">Flexão de pernas e braços</string>
    <string name="label_movimento_ativo">Movimento ativo / Boa flexão</string>
    <string name="label_cor_pele">Cor da Pele</string>
    <string name="label_cianose_central">Cianose central / Palidez</string>
    <string name="label_cianose_extremidades">Cianose de extremidades</string>
    <string name="label_rosado">Rosado</string>
    <string name="label_irritabilidade_reflexiva">Irritabilidade Reflexiva</string>
```

```
    <string name="label_algum_movimento">Algum movimento / Careta</string>
    <string name="label_espirro_choro">Espirro / Choro</string>
    <string name="apgar_formula">Avalia o nível de asfixia de um recém-nascido através da observação de cinco sinais objetivos do bebê no primeiro, no quinto e no décimo minuto após o nascimento. Os cinco sinais objetivos são frequência cardíaca, respiração, tônus muscular, cor da pele e irritabilidade reflexiva.\n\nEscala de Apgar:\n\nPontuação / Resultado\n8-10 = Sem asfixia\n5-7 = Asfixia leve\n3-4 = Asfixia moderada\n0-2 = Asfixia grave</string>
    <string name="result_bebe_sem_asfixia">Recém-nascido sem asfixia.</string>
    <string name="result_bebe_asfixia_leve">Recém-nascido com asfixia leve.</string>
    <string name="result_bebe_asfixia_moderada">Recém-nascido com asfixia moderada.</string>
    <string name="result_bebe_asfixia_grave">Recém-nascido com asfixia grave.</string>

    <!-- Activity CalculaRiscoPneumoniaCurb -->
    <string name="sim">Sim</string>
    <string name="nao">Não</string>
    <string name="label_confusao">Confusão</string>
    <string name="label_nivel_ureia">Nível Uréia &gt; 19 mg/dL</string>
    <string name="label_freq_respiratoria">Respiração &gt; 30 incursões respiratórias</string>
    <string name="label_pressao_sanguinea">Pressão Sanguínea: Sistólica &lt; 90 ou diastólica &lt;= 60 [mmHg]</string>
    <string name="label_idade_menor_65">Idade &gt;= 65 anos</string>
    <string name="formula_curb" formatted="false">Utiliza cinco critérios (confusão, nível de ureia, respiração, pressão sanguínea e idade) para determinar o risco de morte, nos próximos trinta dias, de uma pessoa com pneumonia.\n\nRisco de morte em 30 dias:\n0 fatores = baixo (0,7%)\n1 fator = baixo (3,2%)\n2 fatores = moderado (13%)\n3 fatores = severo (17%)\n4 fatores = alto (41,5%)\n"5 fatores = alto (57%)</string>
    <string name="result_curb_1">Baixo risco de morte em 30 dias (0,7%)</string>
    <string name="result_curb_2">Baixo risco de morte em 30 dias (3,2%)</string>
    <string name="result_curb_3">Risco moderado de morte em 30 dias (13%)</string>
    <string name="result_curb_4">Risco severo de morte em 30 dias (17%)</string>
    <string name="result_curb_5">Alto risco de morte em 30 dias (41,5%)</string>
    <string name="result_curb_6">Alto risco de morte em 30 dias (57%)</string>

    <!-- Activity CalculaParkland -->
    <string name="label_peso">Peso [kg]</string>
    <string name="label_area_queimada">Área corporal queimada [%]</string>
    <string name="parkland_formula">Utilizado para calcular a hidratação necessária em pacientes que sofreram queimaduras.\n\nSoro requerido [litros] = 0,004 x peso x % área corporal queimada</string>
    <string name="valor_entre_0_e_100">Valor entre 0 e 100</string>
    <string name="requerido">Requerido</string>
    <string name="litros_soro_24_horas">litros de soro em 24 horas, sendo</string>
    <string name="litros_8_horas">litros nas primeiras 8 horas e</string>
    <string name="litros_16_horas">litros nas próximas 16 horas.</string>

    <!-- Activity CalculaMeld -->
    <string name="label_bilirrubina">Bilirrubina [mg/dL]</string>
    <string name="label_creatinina">Creatinina [mg/dL]</string>
    <string name="label_INR">INR</string>
    <string name="meld_formula" formatted="false">É um sistema de pontuação utilizado para avaliar a gravidade da doença hepática crônica. É uma escala numérica que exibe o risco de morte de um paciente enquanto espera por um transplante de fígado.\n\nMELD = 3,78 * Ln(Bilirrubina) + 11,2 * Ln(INR) + 9,57 * Ln(Creatinina) + 6,43.\n\nAnálise: \nRisco de mortalidade em 3 meses: \nMELD &gt; 40 = risco de 100%;\n30 &lt; MELD &lt; 39 = risco de 83%;\n20 &lt; MELD &lt; 29 = risco de 76%;\n10 &lt; MELD &lt; 19 = risco de 27%;\nMELD &lt; 10 = risco de 4%.</string>
    <string name="valor_escala_meld">Valor Escala MELD</string>
    <string name="result_meld_1">Risco de 100% de mortalidade em 3 meses.</string>
    <string name="result_meld_2">Risco de 83% de mortalidade em 3 meses.</string>
    <string name="result_meld_3">Risco de 76% de mortalidade em 3 meses.</string>
    <string name="result_meld_4">Risco de 27% de mortalidade em 3 meses.</string>
    <string name="result_meld_5">Risco de 4% de mortalidade em 3 meses.</string>

    <!-- Activity CalculaCockroftGault -->
    <string name="label_idade">Idade [anos]</string>
    <string name="label_sexo">Sexo</string>
    <string name="label_masculino">Masc</string>
    <string name="label_feminino">Fem</string>
    <string name="cockroft_gault_formula">Utilizado para medir a estimativa de filtração glomerular, isto é, a depuração plasmática de creatinina estimada.\n\nEstimativa = ((140 - idade) * peso) / (creatinina * 72)</string>
```

```
    <string name="estimativa">Estimativa</string>

    <!-- Activity CalculaGlasgow -->
    <string name="label_teste_ocular">Teste de Abertura Ocular</string>
    <string name="label_teste_verbal">Teste de Resposta Verbal</string>
    <string name="label_teste_motor">Teste de Resposta Motora</string>
    <string name="label_ocular_1">Ausente</string>
    <string name="label_ocular_2">Em resposta à dor</string>
    <string name="label_ocular_3">Em comando verbal</string>
    <string name="label_ocular_4">Espontaneamente</string>
    <string name="label_verbal_1">Emudecido</string>
    <string name="label_verbal_2">Sons incompreensíveis</string>
    <string name="label_verbal_3">Palavras desconexas</string>
    <string name="label_verbal_4">Confuso, desorientado</string>
    <string name="label_verbal_5">Conversa normalmente</string>
    <string name="label_motor_1">Não se movimenta</string>
    <string name="label_motor_2">Extensão hipertônica</string>
    <string name="label_motor_3">Flexão hipertônica</string>
    <string name="label_motor_4">Flexão inespecífica</string>
    <string name="label_motor_5">Localiza estímulos dolorosos</string>
    <string name="label_motor_6">Obedece comandos</string>
    <string name="glasgow_formula">Escala neurológica para verificar o nível de consciência de uma pessoa após traumatismo craniano. Utiliza resultados de teste de abertura ocular, de resposta verbal e de resposta motora.\n\nEscala de Coma Glasgow:\n\nPontuação / Resultado\n15 = Normalidade\n11-14 = Coma superficial\n7-10 = Coma intermediário\n4-6 = Coma profundo\n3 = Estado vegetativo</string>
    <string name="result_glasgow_1">.\nNormalidade (Trauma leve).</string>
    <string name="result_glasgow_2">.\nComa superficial (Trauma moderado).</string>
    <string name="result_glasgow_3">.\nComa intermediário (Trauma grave).</string>
    <string name="result_glasgow_4">.\nComa profundo (Trauma grave).</string>
    <string name="result_glasgow_5">.\nComa profundo (Estado vegetativo).</string>
    <string name="result_glasgow_6">\nNecessidade de intubação imediata.</string>

    <!-- Activity CalculaChildPugh -->
    <string name="label_bilirrubina_total">Bilirrubina Total</string>
    <string name="label_campo_bilirrubina_1">&lt; 2 mg/dL</string>
    <string name="label_campo_bilirrubina_2"> entre 2 e 3 mg/dL</string>
    <string name="label_campo_bilirrubina_3">&gt; 3 mg/dL</string>
    <string name="label_albumina">Albumina</string>
    <string name="label_campo_albumina_1">&gt; 3,5 g/dL</string>
    <string name="label_campo_albumina_2">entre 2,8 e 3,5 g/dL</string>
    <string name="label_campo_albumina_3">&lt; 2,8 g/dL</string>
    <string name="label_campo_INR_1">&lt; 1,7</string>
    <string name="label_campo_INR_2">entre 1,7 e 2,2</string>
    <string name="label_campo_INR_3">&gt; 2,2</string>
    <string name="label_ascite">Ascite</string>
    <string name="label_nenhuma">Nenhuma</string>
    <string name="label_leve">Leve / Controlada</string>
    <string name="label_grave">Grave / Não controlada</string>
    <string name="label_encefalopatia">Encefalopatia</string>
    <string name="child_pugh_formula">Utilizada para avaliar o prognóstico de doença hepática crônica. São analisados cinco critérios: bilirrubina total, albumina, INR, ascite e encefalopatia.\n\nClassificação Child Pugh:\n\nPontuação / Resultado\n10-15 = Classe C\n7-9 = Classe B\n5-6 = Classe A</string>
    <string name="result_child_1" formatted="false">\nClasse C.\nSobrevida em um ano: 45%.\nSobrevida em dois anos: 35%.</string>
    <string name="result_child_2" formatted="false">\nClasse B.\nSobrevida em um ano: 81%.\nSobrevida em dois anos: 57%.</string>
    <string name="result_child_3" formatted="false">\nClasse A.\nSobrevida em um ano: 100%.\nSobrevida em dois anos: 85%.</string>

    <!-- Activity CalculaIMC -->
    <string name="label_altura">Altura [m]</string>
    <string name="imc_formula">Utilizado para medir o grau de obesidade de uma pessoa, sabendo se ela está em seu peso ideal ou não.\n\nIMC = peso / altura²\n\nResultado / Situação\nIMC &lt; 18,5 = Abaixo do peso ideal\n18,5 &lt; IMC &lt; 24,9 = Peso ideal\n25,0 &lt; IMC &lt; 29,9 = Acima do peso\n30,0 &lt; IMC &lt; 34,9 = Obesidade\n35,0 &lt; IMC &lt; 39,9 = Obesidade severa\nIMC &gt; 40,0 = Obesidade mórbida</string>
    <string name="imc_sigla">IMC</string>
    <string name="result_imc_1">Peso abaixo do ideal.</string>
```

```
    <string name="result_imc_2">Peso ideal, adequado à altura.</string>
    <string name="result_imc_3">Sobrepeso, predisposição à obesidade.</string>
    <string name="result_imc_4">Obesidade grau I.</string>
    <string name="result_imc_5">Obesidade grau II, severa.</string>
    <string name="result_imc_6">Obesidade grau III, mórbida.</string>

    <!-- Activity CalculaLDL -->
    <string name="label_colesterol_total">Colesterol Total [mg/dL]</string>
    <string name="label_hdl">HDL [mg/dL]</string>
    <string name="label_triglicerideos">Triglicerídeos [mg/dL]</string>
    <string name="ldl_formula">Utilizado para medir o valor do LDL (Low Density Lipoprotein), conhecido como colesterol “ruim” ou “mau”.\n\nLDL = colesterol total - HDL - (triglicerídeos / 5)</string>
    <string name="erro_triglicerideos">Deve ser &amp;lt; 400 mg/dL</string>
    <string name="ldl_sigla">LDL</string>
    <string name="result_ldl_1">Nível ótimo de colesterol LDL.</string>
    <string name="result_ldl_2">Nível bom de colesterol LDL.</string>
    <string name="result_ldl_3">Nível limítrofe alto de colesterol LDL.</string>
    <string name="result_ldl_4">Nível alto de colesterol LDL.</string>
    <string name="result_ldl_5">Nível multo alto de colesterol LDL.</string>

    <!-- Activity CalculaRamsay -->
    <string name="label_grau_sedacao_paciente">Grau de Sedação do Paciente</string>
    <string name="label_grau_sedacao_1">Grau 1</string>
    <string name="label_grau_sedacao_2">Grau 2</string>
    <string name="label_grau_sedacao_3">Grau 3</string>
    <string name="label_grau_sedacao_4">Grau 4</string>
    <string name="label_grau_sedacao_5">Grau 5</string>
    <string name="label_grau_sedacao_6">Grau 6</string>
    <string name="formula_ramsay">Utilizada para, entre seis graus existentes, classificar o nível sedação de pacientes.\n\nEscala de Ramsay:\n\nGrau / Resultado\n1 = Paciente ansioso\n2 = Paciente cooperativo\n3 = Paciente sonolento\n4 = Paciente dormindo, respostas rápidas\n5 = Paciente dormindo, respostas lentas\n6 = Paciente dormindo, sem respostas</string>
    <string name="result_ramsay_1">Paciente agitado, ansioso, colabora e atende aos comandos.</string>
    <string name="result_ramsay_2">Paciente cooperativo, tranquilo, colabora e atende aos comandos.</string>
    <string name="result_ramsay_3">Paciente sonolento, atende aos comandos.</string>
    <string name="result_ramsay_4">Paciente dormindo, responde rapidamente ao estímulo glabelar ou ao estímulo sonoro vigoroso.</string>
    <string name="result_ramsay_5">Paciente dormindo, responde lentamente ao estímulo glabelar ou ao estímulo sonoro vigoroso.</string>
    <string name="result_ramsay_6">Paciente dormindo, sem resposta.</string>

    <!-- Activity CalculaAshworth -->
    <string name="label_grau_espasticidade">Grau de Espasticidade</string>
    <string name="label_grau_espasticidade_0">Grau 0</string>
    <string name="label_grau_espasticidade_1">Grau 1</string>
    <string name="label_grau_espasticidade_1_plus">Grau 1+</string>
    <string name="label_grau_espasticidade_2">Grau 2</string>
    <string name="label_grau_espasticidade_3">Grau 3</string>
    <string name="label_grau_espasticidade_4">Grau 4</string>
    <string name="ashworth_formula">Graduação do tônus muscular de um paciente entre seis graus possíveis (0, 1, 1+, 2, 3 e 4).\n\nEscala de Ashworth:\n\nGrau / Resultado\n0 = Tônus normal\n1 = Leve aumento no tônus, resistência ao final\n1+ = Leve Aumento no tônus, resistência no meio\n2 = Marcante aumento no tônus\n3 = Considerável aumento no tônus\n4 = Rigidez</string>
    <string name="result_ashworth_1">Normal, sem aumento de tônus muscular.</string>
    <string name="result_ashworth_2">Leve aumento no tônus muscular, com mínima resistência ao final da amplitude do movimento articular.</string>
    <string name="result_ashworth_3">Leve aumento no tônus muscular, com mínima resistência em menos da metade da amplitude do movimento articular restante.</string>
    <string name="result_ashworth_4">Marcante aumento no tônus muscular durante a maior parte da amplitude do movimento articular, todavia com fácil movimento.</string>
    <string name="result_ashworth_5">Considerável aumento no tônus muscular com dificuldade de movimento.</string>
    <string name="result_ashworth_6">Segmento afetado rígido em flexão ou extensão.</string>

    <!-- Activity CalculaForrest -->
    <string name="label_tipo_lesao">Tipo de Lesão</string>
    <string name="label_tipo_forrest_1A">Tipo IA - Em Jato</string>
```

```
    <string name="label_tipo_forrest_1B">Tipo IB - Escorrendo</string>
    <string name="label_tipo_forrest_2A">Tipo IIA - Coto vascular visível</string>
    <string name="label_tipo_forrest_2B">Tipo IIB - Coágulo recente</string>
    <string name="label_tipo_forrest_2C">Tipo IIC - Fundo hematínico</string>
    <string name="label_tipo_forrest_3">Tipo III - Sem sangramento</string>
    <string name="forrest_formula" formatted="false">Classificação endoscópica de úlcera. Classifica os pacientes em sangramento ativo (em jato ou escorrendo), em sangramento recente (coto vascular visível, coágulo recente ou fundo hematínico) ou em sangramento inexistente.\n\nClassificação Forrest:\n\n"Tipo Lesão / Risco Ressangramento\nIA = >90%\nIB = 30 a 51%\nIIA = 25 a 41%\nIIB = 10 a 20%\nIIC = 0 a 30%\nIII = 0 a 2%</string>
    <string name="result_forrest_1" formatted="false">Sangramento ativo.\nFrequencia dos estigmas: 8 a 15%\nRisco de ressangramento: >90%</string>
    <string name="result_forrest_2" formatted="false">Sangramento ativo.\nFrequencia dos estigmas: 26 a 55%\nRisco de ressangramento: 30 a 51%</string>
    <string name="result_forrest_3" formatted="false">Sangramento recente.\nFrequencia dos estigmas: 10 a 18%\nRisco de ressangramento: 25 a 41%</string>
    <string name="result_forrest_4" formatted="false">Sangramento recente.\nFrequencia dos estigmas: 10 a 20%\nRisco de ressangramento: 10 a 20%</string>
    <string name="result_forrest_5" formatted="false">Sangramento recente.\nFrequencia dos estigmas: 12%\nRisco de ressangramento: 0 a 30%</string>
    <string name="result_forrest_6" formatted="false">Sangramento inexistente.\nFrequencia dos estigmas: 36%\nRisco de ressangramento: 0 a 2%</string>

    <!-- Activity CalculaKillip -->
    <string name="label_classe">Classe</string>
    <string name="label_classe_killip_1">Killip I</string>
    <string name="label_classe_killip_2">Killip II</string>
    <string name="label_classe_killip_3">Killip III</string>
    <string name="label_classe_killip_4">Killip IV</string>
    <string name="killip_formula" formatted="false">Utilizado em indivíduos com infarto agudo de miocárdio para verificar as chances de falecimento nos trinta dias subsequentes ao ataque cardíaco.\n\nClassificação Killip:\n\nClasse / Taxa Mortalidade\nKillip I = 6%\nKillip II = 17%\nKillip III = 38%\nKillip IV = 81%</string>
    <string name="result_killip_1">Sem sinais de insuficiência cardíaca.\nTaxa de mortalidade: 6%</string>
    <string name="result_killip_2">Insuficiência cardíaca leve ou moderada.\nTaxa de mortalidade: 17%</string>
    <string name="result_killip_3">Edema pulmonar.\nTaxa de mortalidade: 38%</string>
    <string name="result_killip_4">Choque cardiogênico.\nTaxa de mortalidade: 81%</string>

    <!-- Activity CalculaCriterionRanson -->
    <string name="label_admissao_paciente">Na Admissão do Paciente</string>
    <string name="label_idade_maior_55">Idade &gt; 55 anos</string>
    <string name="label_glicemia_maior_200">Glicemia &gt; 200 mg/dL</string>
    <string name="label_leucocitos_maior_16000">Número de leucócitos &gt; 16.000/mm³</string>
    <string name="label_ldh_maior_350">Desidrogenase láctica (LDH sérica) &gt; 350 IU/L</string>
    <string name="label_ast_maior_250">Aspartato aminotransferase (TGO/AST) &gt; 250 IU/L</string>
    <string name="label_48_horas_apos_admissao">48 Horas Após Admissão do Paciente</string>
    <string name="label_queda_hematocrito">Queda hematócrito &gt; 10%</string>
    <string name="label_calcio_serico">Cálcio sérico &lt; 8 mg/dL</string>
    <string name="label_deficit_base">Défict de base &gt; 4 mEq/L</string>
    <string name="label_aumento_bun_urinario">Aumento do BUN urinário &gt; 5 mg/dL</string>
    <string name="label_sequestro_liquido">Sequestro de líquido &gt; 6 L</string>
    <string name="label_hipoxemia">Hipoxemia (PO2) &lt; 60 mmHg</string>
    <string name="ranson_formula" formatted="false">Utiliza onze parâmetros para estimar o prognóstico de pacientes com pancreatite aguda, sendo que cinco parâmetros devem ser analisados na admissão do paciente, e os restantes após 48 horas da admissão.\n\nCritério de Ranson:\nNúmero Critérios / Mortalidade\n1 = 0,9%\n"2 = 0,9%\n3 = 16%\n4 = 16%\n5 = 40%\n6 = 40%\n7 = 100%\n8 = 100%\n9 = 100%\n10 = 100%\n11 = 100%</string>
    <string name="result_quantidade_criterios">Quantidade de critérios</string>
    <string name="result_ranson_1">Mortalidade: 0,9%.\nPancreatite grave: improvável.</string>
    <string name="result_ranson_2">Mortalidade: 16%.\n</string>
    <string name="result_ranson_3">Mortalidade: 40%.\n</string>
    <string name="result_ranson_4">Mortalidade: 100%.\n</string>
    <string name="result_pancreatite_grave_provavel">Pancreatite grave: provável.</string>

    <!-- Activity CalculaCooper -->
    <string name="label_distancia">Distância [metros]</string>
    <string name="formula_cooper">Calcula o volume de oxigênio máximo de um indivíduo, que representa o volume máximo de oxigênio que o corpo consegue transportar e utilizar durante a execução de um exercício físico.\n\nVO2máx = (distância - 504,9)/44,73</string>
```

```
    <string name="vo2_max">VO2máx</string>
    <string name="resultado_cooper">Resultado do Cooper</string>
    <string name="muito_ruim">muito ruim.</string>
    <string name="ruim">ruim.</string>
    <string name="regular">regular.</string>
    <string name="bom">bom.</string>
    <string name="otimo">ótimo.</string>

    <!-- Activity CalculaGorduraCorporal -->
    <string name="label_gordura_corporal">Gordura Corporal [%]</string>
    <string name="gordura_corporal_formula">Margem de Gordura Corporal\n\nLeva em consideração o sexo e a idade da pessoa, classificando-a em quatro categorias:\n* Com pouca gordura;\n* Saudável;\n* Alto teor de gordura;\n* Obesa.</string>
    <string name="restul_gordura_1" formatted="false">Entre 0% e 8% de gordura corporal.\nPessoa com pouca gordura.</string>
    <string name="restul_gordura_2" formatted="false">Entre 8% e 20% de gordura corporal.\nPessoa saudável.</string>
    <string name="restul_gordura_3" formatted="false">Entre 20% e 25% de gordura corporal.\nPessoa com alto teor de gordura.</string>
    <string name="restul_gordura_4" formatted="false">Mais que 25% de gordura corporal.\nPessoa obesa.</string>
    <string name="restul_gordura_5" formatted="false">Entre 0% e 11% de gordura corporal.\nPessoa com pouca gordura.</string>
    <string name="restul_gordura_6" formatted="false">Entre 11% e 22% de gordura corporal.\nPessoa saudável.</string>
    <string name="restul_gordura_7" formatted="false">Entre 22% e 28% de gordura corporal.\nPessoa com alto teor de gordura.</string>
    <string name="restul_gordura_8" formatted="false">Mais que 28% de gordura corporal.\nPessoa obesa.</string>
    <string name="restul_gordura_9" formatted="false">Entre 0% e 13% de gordura corporal.\nPessoa com pouca gordura.</string>
    <string name="restul_gordura_10" formatted="false">Entre 13% e 25% de gordura corporal.\nPessoa saudável.</string>
    <string name="restul_gordura_11" formatted="false">Entre 25% e 30% de gordura corporal.\nPessoa com alto teor de gordura.</string>
    <string name="restul_gordura_12" formatted="false">Mais que 30% de gordura corporal.\nPessoa obesa.</string>
    <string name="restul_gordura_13" formatted="false">Entre 0% e 21% de gordura corporal.\nPessoa com pouca gordura.</string>
    <string name="restul_gordura_14" formatted="false">Entre 21% e 33% de gordura corporal.\nPessoa saudável.</string>
    <string name="restul_gordura_15" formatted="false">Entre 33% e 39% de gordura corporal.\nPessoa com alto teor de gordura.</string>
    <string name="restul_gordura_16" formatted="false">Mais que 39% de gordura corporal.\nPessoa obesa.</string>
    <string name="restul_gordura_17" formatted="false">Entre 0% e 23% de gordura corporal.\nPessoa com pouca gordura.</string>
    <string name="restul_gordura_18" formatted="false">Entre 23% e 34% de gordura corporal.\nPessoa saudável.</string>
    <string name="restul_gordura_19" formatted="false">Entre 34% e 40% de gordura corporal.\nPessoa com alto teor de gordura.</string>
    <string name="restul_gordura_20" formatted="false">Mais que 40% de gordura corporal.\nPessoa obesa.</string>
    <string name="restul_gordura_21" formatted="false">Entre 0% e 24% de gordura corporal.\nPessoa com pouca gordura.</string>
    <string name="restul_gordura_22" formatted="false">Entre 24% e 36% de gordura corporal.\nPessoa saudável.</string>
    <string name="restul_gordura_23" formatted="false">Entre 36% e 42% de gordura corporal.\nPessoa com alto teor de gordura.</string>
    <string name="restul_gordura_24" formatted="false">Mais que 42% de gordura corporal.\nPessoa obesa.</string>
    <string name="restul_gordura_25" formatted="false">Não há resultado disponível para a idade informada.\nInforme uma idade entre 18 e 99 anos</string>

    <!-- Activity CalculaMassaOssea -->
    <string name="massa_ossea_formula">Utiliza o sexo e o peso da pessoa para exibir sua massa óssea estimada.\n\nNível de Massa Óssea:\n\nSexo Feminino\n\nPeso / Massa Óssea\n &amp;lt;50 kg = 1,95 kg\n50 - 75 kg = 2,40 kg\n&amp;gt; 75 kg = 2,95 kg\n\nSexo Masculino\n\nPeso / Massa Óssea\n&amp;lt; 65 kg = 2,66 kg\n65 - 95 kg = 3,29 kg\n&amp;gt; 95 kg = 3,69 kg</string>
    <string name="media_massa_ossea_estimada">Média de massa óssea estimada</string>
    <string name="result_massa_ossea_1">2,66 kg.</string>
    <string name="result_massa_ossea_2">3,29 kg.</string>
    <string name="result_massa_ossea_3">3,69 kg.</string>
```

```
    <string name="result_massa_ossea_4">1,95 kg.</string>
    <string name="result_massa_ossea_5">2,40 kg.</string>
    <string name="result_massa_ossea_6">2,95 kg.</string>

    <!-- Activity CalculaZonasAlvo -->
    <string name="formula_zona" formatted="false">Lista as zonas alto de treinamento aeróbico de um indivíduo, a
partir de sua idade.\n\nFcMax = 220 - idade.\n\nZona Alvo / bpn\nEsforço máximo = 90-100% FcMax\nLimiar
anaeróbico = 80-90% FcMax\nAeróbica = 70-80% FcMax\nControle peso = 60-70% FcMax\nAtivid. moderada =
50-60% FcMax</string>
    <string name="result_zona_alvo">Zonas alvo de treinamento aeróbico em bpm (batimentos por
minuto).\n\n</string>
    <string name="freq_cardiaca_max">Frequência Cardíaca Máxima</string>
    <string name="esforco_max">"Esforço máximo</string>
    <string name="limiar_aerobico">Limiar anaeróbico</string>
    <string name="aerobica">Aeróbica</string>
    <string name="controle_peso">Controle de peso</string>
    <string name="atividade_moderada">Atividade moderada</string>
    <string name="entre">entre</string>
    <string name="e">e</string>
    <string name="hello_world">Hello world!</string>
    <string name="title_activity_informacoes_app">InformacoesApp</string>
</resources>
```