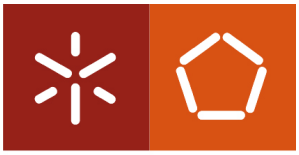

Universidade do Minho

LEI — Licenciatura de Engenharia Informática

UC8204P1 — Programação Orientada a Objectos

FitnessUM

Grupo 14

Rui Camposinhos - a72625, Rui Oliveira - a67661, André Santos - a61778



Braga, 1 de junho de 2014

Conteúdo

Capítulo 1

Introdução

O presente projecto enquadra-se na unidade curricular de Programação Orientada a Objectos do curso de Licenciatura em Engenharia Informática da Universidade do Minho. O projecto pretende implementar uma aplicação, designada *FitnessUM*, para registar e simular actividades desportivas. A aplicação foi desenvolvida na linguagem *java* e pretende simular um ambiente de rede social.

1.1 Objectivos

O presente projeto tem como objectivo principal consolidar e colocar em práctica os conceitos fundamentais transmitidos durante a UC de Programação Orientada a Objectos.

De acordo com o enunciado [1], os principais objectivos definidos para a aplicação *FitnessUM* são os seguintes:

Requisitos básicos

- Aceder à aplicação utilizando as credenciais (email e password);
- Visualizar a informação das actividades, do próprio e dos amigos, com possibilidade de aceder aos seus detalhes;
- Registar a informação de uma actividade;
- Consultar, por ordem cronológica, e remover actividades;
- Aceder às estatísticas de distância, tempo e calorias gastas;

Requisitos de valorização

- Definir tipos de recordes e registar os recordes pessoais de cada utilizador;
- Definir eventos e proceder à sua simulação a partir dos administratores da aplicação.

1.2 Organização do Relatório

O presente relatório serve de suporte à aplicação desenvolvida, apresentando uma descrição geral dos aspectos mais importantes. Com excepção do presente capítulo introdutório, o relatório é composto por dois capítulos iniciais e nucleares - Arquitectura e Estruturas de Dados - e por quatro capítulos complementares relacionados com funcionalidades particulares da aplicação - Consumo de Calorias, Forma dos Utilizadores, Estatísticas e Simulação de Eventos. No final são tecidas algumas conclusões, bem como apresentadas algumas imagens da aplicação em anexo.

Capítulo 2

Arquitectura e Descrição da Aplicação

A aplicação foi desenvolvida utilizando simultaneamente os IDEs *Eclipse* e *BlueJ*.

Foi utilizada uma arquitectura do tipo *Model–View–Controller* (MVC) - ver figura 2.1 -, permitindo uma interface gráfica com o utilizador.

De forma a manter uma estrutura de pastas partilhada entre os autores e um controlo de versões eficaz, foi utilizada a ferramenta open source *GIT* (http://git-scm.com/), com repositório privado no bitbucket (https://bitbucket.org/rui0liveiras94/fitnessum-poo).

2.1 Packages

A figura 2.2 apresenta os packages principais da aplicação, em correspondência com a arquitectura MVC descrita.

Na figura 2.3 descreve o package *model*, com os seus sub-packages principais - activity, user e

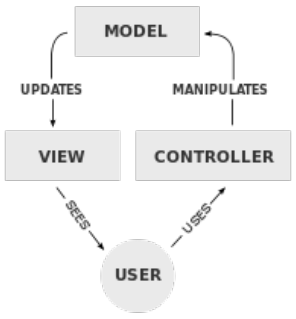

Figura 2.1: Diagrama com a relação típica dos componentes do MVC (ref.:http://en.wikipedia.org/wiki/Model-view-controller).

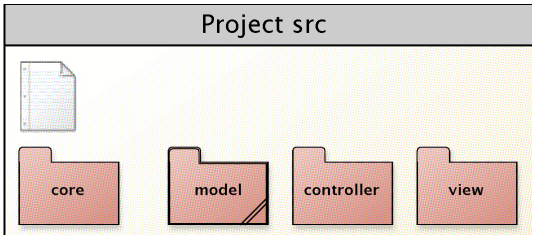

Figura 2.2: Pasta *source* com os vários *packages* desenvolvidos.

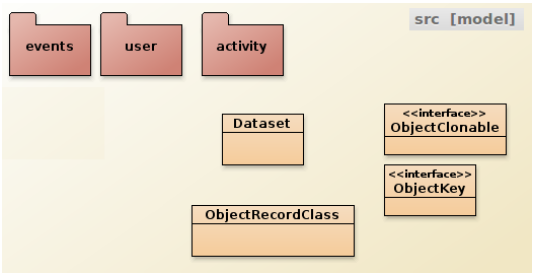

Figura 2.3: *Package model* (src/model).

events -, bem como *Dataset* para gestão dos dados e a classe para definição dos recordes (ver 3.3). A classe *Dataset* é o conjunto de toda a informação referente à aplicação criada, contendo um conjunto de utilizadores e outro de eventos. Esta informação é acedida através da classe *Manager*, descrita na secção 2.2.

O package *model*, descrito na figura 2.4, gere toda a interface gráfica com o utilizador.

O package *controller* encontra-se descrito na figura 2.5, possuindo todas as classes de controlo da aplicação, bem como a classe principal (Main) com o executável da aplicação.

O package *core*, descrito na figura 2.6, foi criado para alojar algumas funcionalidades mais específicas da aplicação. A sua separação dos packages principais deveu-se unicamente a uma opção de organização e gestão do código, assim como para facilitar o desenvolvimento em paralelo por parte dos diferentes autores. Neste package inclui-se o código desenvolvido para o cálculo de calorias, cálculo da forma, cálculo de estatísticas e métodos de simulação. Todos os métodos foram implementados como `static`.

2.2 Encapsulamento: Manager

A figura 2.7 apresenta o diagrama de classes relativo ao *Manager*.

A classe Manager foi criada para facilitar o encapsulamento de tipos `Map` ou `Set`, e realiza as operações básicas de gestão (na adição faz um clone, na pesquisa retornar um clone).

Quando se instancia a classe *Manager*, passa-se a estrutura que pretendemos encapsular, a partir daí esta estrutura nunca mais pode ser acedida. Assim, caso seja necessária aceder a algum método especifico da estrutura, antes de esta ser instanciada deverá ser guardada.

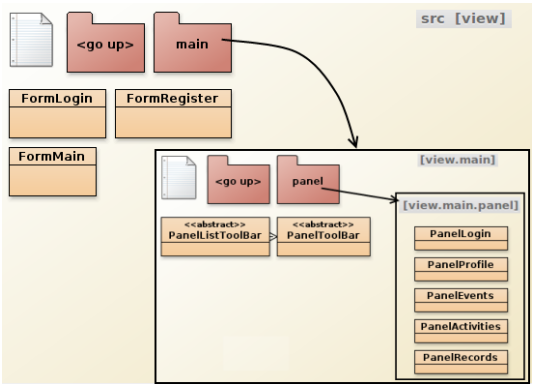

Figura 2.4: *Package view* (src/view).

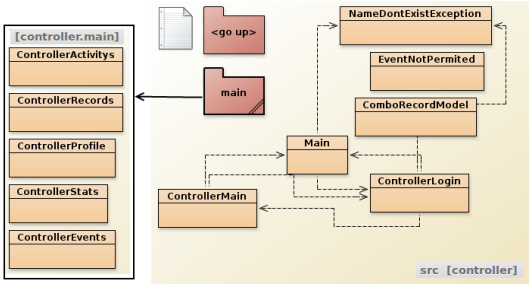

Figura 2.5: *Package controller* (src/controller).

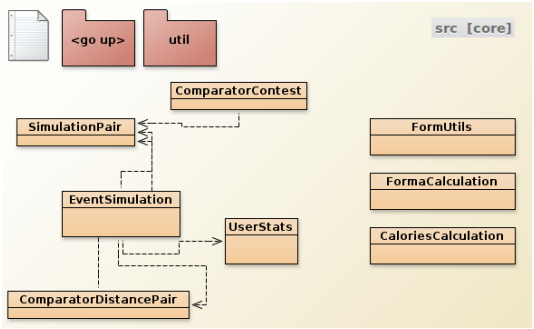

Figura 2.6: *Package core* (src/core).

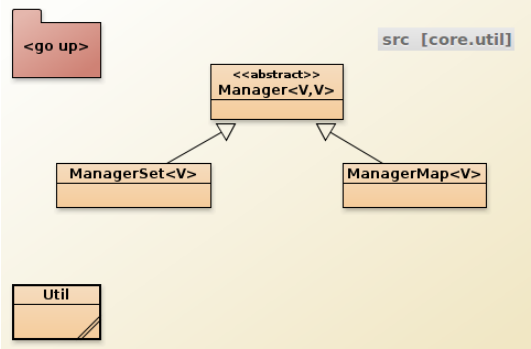

Figura 2.7: *Package* util (src/core/util) com o *Manager* de colecções.

Desta forma torna-se suficiente retornar o *Manager* para que o exterior consiga aceder de forma encapsulada à informação.

Para se poder utilizar um *Manager* de `Map` os objetos devem implementar a interface *ObjectKey*. O valor retornado por essa função será a chave do objecto no `Map`.

2.3 Diagramas de Classes

2.3.1 Actividades

Na figura 2.8 apresentam-se todas as actividades físicas implementadas na aplicação, bem como as constantes especificas de cada clasee `MET` e `Intensidade`, que serão descritas adiante. As actividades foram todas classificadas em um de cinco tipos: Distance, Altimetry, Skill, Ludic e Contest. As estruturas de dados de cada classe encontram-se descritas no capítulo 3.

As classes de actividades físicas para além de funcionarem como um modelo de classificação, obedecem a uma hierarquia, onde são herdados atributos e "obrigações" das classes hierárquicamente superiores. A figura 2.9 apresenta o diagrama de classes de actividades, obtido através do *BlueJ*. A hierarquia de classes definida implica a herança de alguns métodos abstractos fundamentais para a aplicação. São exemplo disso os método `getIntensidade()` e o `getMET()`. O primeiro devolve o número de minutos diários recomendados para a actividade, para que o utilizador atinga uma forma ideal. O segundo, serve de base ao cálculo das calorias dessa actividade, devolvendo uma constante específica de cada actividade (ver secção 4.2).

2.3.2 Eventos

Para além das actividades, também os eventos obedecem a uma hierarquia. As estruturas de dados de cada classe de evento encontram-se descritas no capítulo 3. De forma semelhante às actividades, embora mais simplificada, definiram dois tipos de eventos: Distance e Contest. A figura 2.10 descreve essas dependências.

Designação (EN) (ref. Endomondo)	Designação (PT)	METS	Intens. (min.)	Activity type				
				Distance	Distance+ Altimetry	Skill	Ludic	Contest
Skiing, downhill	Ski downhill	5,3	119	x				
Rowing	Remo	5,8	109	x				
Sailing	Vela	3,0	210	x				
Windsurfing	Windsurfing	3,0	210	x				
Fitness walking	Caminhada	3,5	180	x				
Orienteering	Corrida de Orientação	9,0	70	x				
Swimming	Natação	5,8	109	x				
Kayaking	Caiaque	5,0	126	x				
Running	Corrida	7,0	90		x			
Cycling, sport	Ciclismo	7,5	84		x			
Mountain biking	BTT	8,5	74		x			
Skiing, cross country	Ski TT	7,0	90		x			
Hiking	Caminhada montanha	6,0	105		x			
Skating	Skate	7,0	90			x		
Snowboarding	Snowboard	5,3	119			x		
Golfing	Golfe	4,8	131			x		
Riding	Hipismo	5,5	115			x		
Gymnastics	Ginástica (olímpica)	3,8	166			x		
Surfing	Surf	3,0	210			x		
Aerobics	Aeróbica	7,3	86				x	
Dancing	Dança	7,8	81				x	
Pilates	Pilates	3,0	210				x	
Scuba diving	Mergulho	7,0	90				x	
Yoga	Ioga	4,0	158				x	
Gymn training	Ginásio (vários)	5,0	126				x	
Climbing	Escalada	8,0	79				x	
Badminton	Badminton	5,5	115					x
Basketball	Basquetebol	6,5	97					x
Boxing	Boxe	12,8	49					x
Fencing	Esgrima	6,0	105					x
Football, rugby	Râguebi	6,3	100					x
Football, soccer	Futebol	7,0	90					x
Handball	Andebol	8,0	79					x
Hockey	Hoquei	7,8	81					x
Polo	Polo	8,0	79					x
Squash	Squash	7,3	86					x
Table tennis	Ténis de mesa	4,0	158					x
Tennis	Ténis	7,3	86					x
Volleyball, beach	Voleibol de praia	8,0	79					x
Volleyball, indoor	Voleibol	4,0	158					x
Martial arts	Artes marciais	10,3	61					x

Figura 2.8: Lista de actividades implementadas na aplicação.

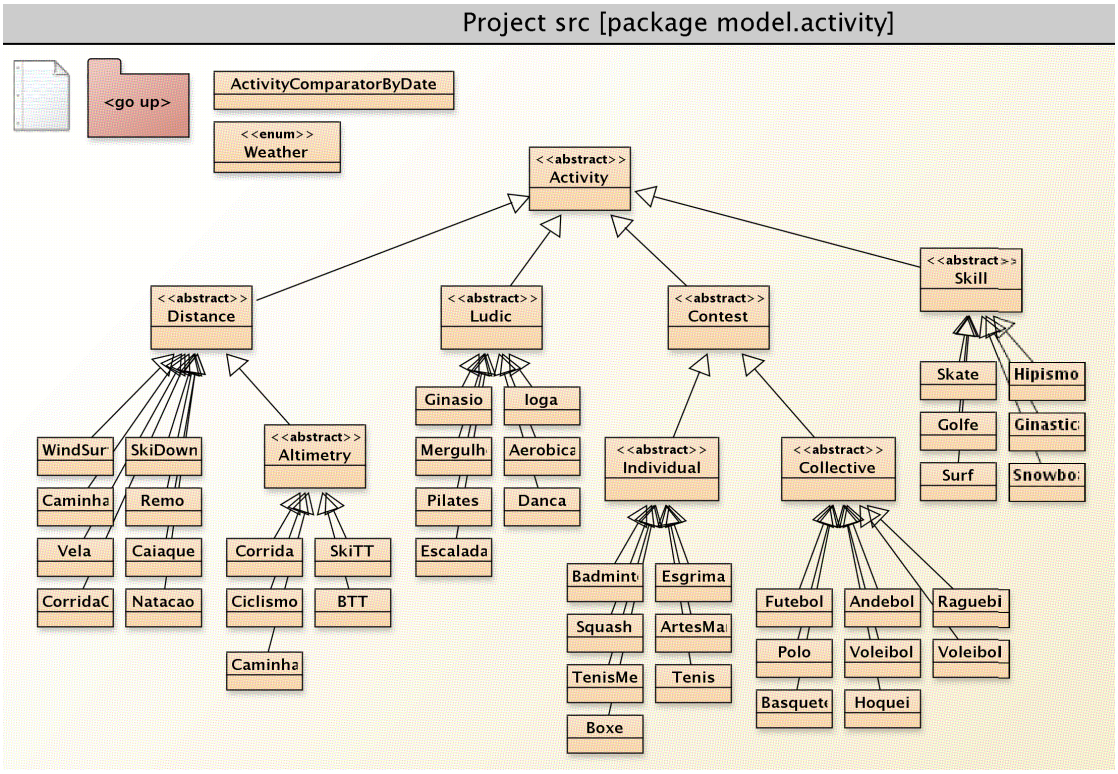

Figura 2.9: Diagrama com dependências da super classe *Activity* (src/model/activity).

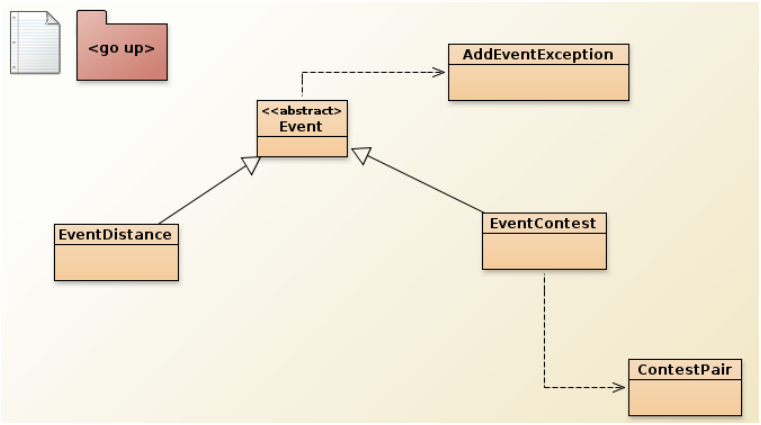

Figura 2.10: Diagrama com dependências da super classe *Event* (src/model/events).

2.4 Funcionalidades da Aplicação

A aplicação desenvolvida apresenta um conjunto considerável de funcionalidades, sendo de destacar as seguintes:

- Interface gráfica com o utilizador;
- Registo de novos utilizadores com credenciais;
- Possibilidade de registar actividade em 41 modalidades;
- Amizades entre utilizadores (dependendo da aceitação de convites);
- Apresentação de índice evolutivo de forma em função das actividades registadas;
- Cálculo das calorias dispendidas em função do género e características físicas de cada índividuo, por intermédio da frequência cardíaca ou tipo de actividade;
- Apresentação de estatísticas de actividades em termos de duração, calorias e distância percorrida;
- Criação e simulação de eventos de distância ou do tipo torneio;
- Possibilidade de extensibilidade para novas actividades e eventos.

Capítulo 3

Estruturas de Dados Principais

3.1 User class

As variáveis de instância da classe `User` são apresentadas no excerto de código 3.1. Na figura 3.1 apresenta-se o package respetivo. Para as permissões foi usada uma enumeração correspondente aos diferentes tipos de utilizador. Estas permissões são usadas para controlar as funcionalidades a que o utilizador tem acesso na aplicação. Para os recordes foi usado um `Map` de uma classe correspondente a uma modalidade para o conjunto de recordes dessa modalidade. Um recorde é constituído pelo seu tipo, dado por um inteiro e por uma "Actividade modelo", que representa o recorde. Para os amigos e lista de amigos foi utilizado o `Manager`, que é instanciado numa `HashSet` de `String`. Para o conjunto de actividades realizadas, foi também utilizado o Manager instanciado numa árvore (TreeSet), ordenado por meio de um `Comparator`, que compara as actividades por ordem crescente da sua data de realização.

3.2 Activity class

Na superclasse abstracta `Activity` foram incorporadas todas as variáveis de instância e métodos gerais das actividades, designadamente a data, duração, meteorologia e frequência cardíaca média

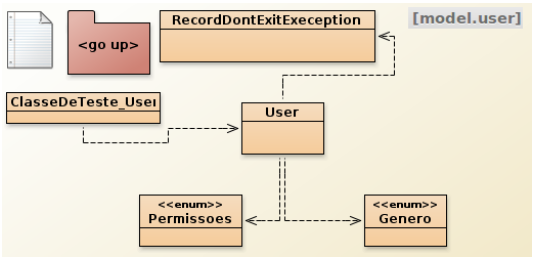

Figura 3.1: *Package user* (src/model/user).

Código 3.1: Variáveis de instância da classe User (src/model/user).

```
1  private String email;
2  private String nome;
3  private String password;
4  private Genero genero;
5  private int altura; /*cm*/
6  private int peso; /*kg*/
7  private GregorianCalendar dataNascimento;
8  private Activity desportoFavorito;
9  private Permissoes permissoes;
10 private Map<Class<?>, HashMap<Integer, Activity>> recordes; /*ver documentacao*/
11 private int fcr; /*frequencia cardiaca em repouso - para calculo das calorias*/
12 private Manager<String> amigos; /*emails amigos:HashSet com chaves para rede social*/
13 private Manager<String> convitesAmigos; /*emails de amigos: convites - HashSet*/
14 private Manager<Activity> actividadesUser; /*Actividades do User:TreeSet -Comp.ByDate*/
```

durante a actividade, caso seja fornecida pelo utilizador. Para as condições meteorológicas foi usada uma enumeração para cobrir as situações mais frequentes em termos de clima. Num nível hierárquico abaixo da superclasse surgem as subclasses `Distance`, `Altimetry`, `Contest`, `Skill` e `Ludic`.

A subclasse `Distance` especializa a superclasse com duas variáveis de instância relativas a actividades que envolvam distâncias, a saber a distância percorrida e a velocidade atingida no percurso. A subclasse `Altimetry` descende da subclasse `Distance`, acrescentando atributos às actividades que além de terem distância têm associada altura, nomeadamente a distância que foi subida e descida, bem como as altitudes máxima e mínima atingidas. A subclasse `Contest` descende da superclasse e representa as actividades em que esteja envolvido algum tipo de disputa/jogo (de equipa ou individual). As suas variáveis de instância representam os pontos obtidos pela equipa adversária e pela própria equipa. A subclasse `Skill` representa as actividades em que a performance é avaliada em termos de uma pontuação, através de uma exibição individual. As variáveis de especialização são a pontuação final obtida e o máximo de pontuação obtida para um determinado movimento. A subclasse `Ludic` representa actividades mais simples e com um carácter mais lúdico. Esta classe não apresenta nenhum especialização relativamente à superclasse. Para além dos métodos abstractos impostos pela superclasse, são também definidos nesta classe os tipos de recordes específicos. Todas as variáveis da superclasse e subclasses referidas são apresentadas no excerto de código 3.2.

3.3 Mecanismo para Recordes

Os recordes das Actividades foram organizados da seguinte forma: todas as actividades implementam `ObjectRecord`, e em cada actividade tem uma enumeração de `Records` e outra com os `EnumAttr`. Os `Records` e os `EnumAttr` devem ser acedidos através dos métodos da interface `ObjectRecord`.

Quando uma actividade extende a outra (exemplo: `Altimetry` extende a `Distance`), os métodos de acesso aos `Records` do objecto funcionam da seguinte forma:

- Se o recorde estiver definido naquela Actividade encontrou e pára;
- Caso contrario pesquisa no nível seguinte

Código 3.2: Variáveis de instância da superclasse Activity e subclasses respectivas (src/model/activity).

```
1  /***Super Classe***/
2  private GregorianCalendar mDate;
3  private long mDuration; /*activity duration [ms]*/
4  private Weather mWeather;
5  private int mHearthRate;/*heart rate [1/min] - for calorie burn calculation*/
6
7  /**Especializacoes Classe Distance**/
8  private int mDistance;
9  private int mMaxSpeed;
10
11 /**Especializacoes Classe Contest**/
12 private int mPointRival;
13 private int mPointTeam;
14 private Contest.Result mResult;
15
16 /**Especializacoes Classe Skill**/
17 private int mPoints;
18 private int mMaxTrick;
```

3.3.1 Record interface

A interface `Record` representa um tipo de recorde e é caracterizado por:

- Um Nome e um ID.
- Um atributo principal, que é atributo que irá ser medido.
- Um atributo fixo (opcional), que funciona como constante para criar recordes semelhantes, mas com um valor diferente (exemplo: ïecorde de tempo em X metros"; X pode é visto como um atributo fixo).

Esta interface também oferece a funcionalidade de verificar, quando necessário, se um valor de um atributo é semelhante ao valor do atributo fixo.

3.3.2 EnumAttr interface

A interface `EnumAttr` representa um atributo, é caracterizado por um Nome e um ID. Esta interface permite aceder aos atributos de forma genérica.

3.3.3 ObjectRecord interface

A interface `ObjectRecord` representa uma entidade constituída por vários recordes. Assim, um objecto que implemente `ObjectRecord` terá as seguintes funcionalidades:

- Porta de acesso para os recordes através do seu ID.
- Porta de acesso para os atributos através do seu ID.
- Ser comparado com outro `ObjectRecord`, para um determinado `Record`, e saber qual é o melhor.
- Representar este `ObjectRecord` para um determinado `Record` sobe a forma de String.

3.4 Event class

As variáveis de instância da superclasse `Event` são apresentadas no excerto de código 3.3. Um evento pode ser caracterizado pelo seu nome, actividade associada, tipo de recorde associado, data de início e término, um valor correspondente a um pré-requisito necessário que um utilizador possa aderir ao evento, o número máximo de utilizadores que o evento suporta e o número de utilizadores actualmente inscritos. Os eventos são também classificados em dois tipos: de distância, ou do tipo torneio (jogos). As subclasses que traduzem essa classificação são a subclasse `EventDistance` e `EventContest`.

Código 3.3: Variáveis de instância da superclasse Event e subclasses respectivas (src/model/event).

```
/***Super Classe***/
private String mName;
private Activity mActivity;/*ATENTION: event weather inside activity*/
private int mRecordType;
private GregorianCalendar mEventDate;
private GregorianCalendar mEndDate;/*deadline for joining*/
private long mPreRequisite;
private int mMaxNumUsers;
private ManagerSet<String> mUsers; /*HashSet ok User keys(emails)*/

/**Especializacoes Classe EventDistance**/
private int mEventDistance;/*An event is about a Activity with its record type, like:
    Run 10000m*/

/**Especializacoes Classe Contest**/
private int mNum;/*Legs Between Users (1 Leg = 2 games)*/
private int mPointsVic;
private int mPointsDraw;
private int mPointsLoss;
```

Capítulo 4

Cálculo do Consumo de Calorias por Actividade

O procedimento de cálculo utilizado para estimar a quantidade de calorias dispendidas por actividade seguiu uma filosofia semelhante à utilizada na rede social *Endomondo* (www.endomondo.com). Na figura 4.1 apresenta-se um diagrama explicativo do mesmo, onde se evidenciam duas vias: uma baseada na frequência cardíaca e outra no tipo de actividade.

Os dados básicos necessários para o cálculo das calorias dispendidas estão relacionados com o individuo e são:

- Género;
- Características físicas — peso, altura;
- Idade;
- Frequência cardíaca em repouso (no caso da via de cálculo baseada na frequência cardíaca).

Os dados complementares são:

- Duração da actividade;
- O tipo de actividade (no caso da via de cálculo baseada no tipo de actividade);
- Frequência cardíaca média durante a actividade (no caso da via de cálculo baseada na frequência cardíaca).

4.1 Cálculo Baseado na Frequência Cardíaca

As formulações expostas na presente secção tiveram por base a aplicação web "Heart Rate Based Calorie Burn Calculator", da página http://www.shapesense.com/fitness-exercise/calculators/. Por uma questão de completitude apresentam-se todas as referências originais. No cálculo pela via da frequência cardíaca um dos parâmetros fundamentais é o $VO2_{max}$, que é a capacidade máxima do corpo de um indivíduo em transportar e fazer uso de oxigênio durante um exercício físico incremental. Este parâmetro pode ser estimado de acordo com [5], com base

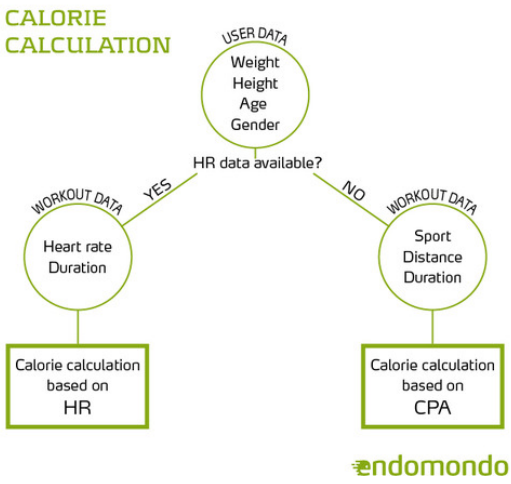

Figura 4.1: Diagrama com o procedimento de cálculo adoptado para a estimativa do consumo de calorias por actividade (ref. [3]).

na frequência cardíaca máxima (MHR) e em repouso (RHR) — equação 4.2. A frequência cardíaca máxima pode ser estimada com base na idade de acordo com [6] — equação 4.1. As calorias brutas dispendidas (GCB) foram estimadas de acordo com [7] — equação 4.3. Por fim, para se determinar o número de calorias efectivas (NCB) calculou-se a taxa metabólica basal (BMR) e a taxa de actividade ($RMRCB$) A estimativa das calorias efectivas foi realizada de acordo com [4] — equações 4.4, 4.5 e 4.6.

$$MHR = 208 - 0.7 \times Idade \quad [1/\text{min}] \tag{4.1}$$

$$VO2_{max} = 15.3 \times \frac{MHR}{RMR} \quad [\text{ml}/(\text{kg.min})] \tag{4.2}$$

Homens: (4.3)

$$GCB = \frac{-55.0969 + 0.6309 \times HR + 0.1988 \times W + 0.2017 \times A}{4.184} \times 60 \times T \quad [\text{kcal}]$$

Mulheres:

$$GCB = \frac{-20.4022 + 0.4472 \times HR + 0.1263 \times W + 0.074 \times A}{4.184} \times 60 \times T \quad [\text{kcal}]$$

onde:

HR = frequência cardíaca $[1/\text{min}]$
W = Peso [kg]
A = Idade [anos]
T = Duração exercício [h]

$$NCB = GCB - RMRCB \quad [\text{kcal}] \tag{4.4}$$

Tabela 4.1: Valores típicos do índice MET (ref. http://www.my-calorie-counter.com/mets_calculation.asp)

METS	Activity
1	sitting quietly and watching television
2	walking, less than 2.0 mph, level ground, strolling, very slow
3	loading /unloading a car
4	bicycling, $<$ 10 mph, leisure, to work or for pleasure
(...)	(...)
11	running, 6.7 mph
12	fire fighter, general

$$RMRCB = \frac{BMR \times 1.1}{24} \times T \quad \text{[kcal]} \tag{4.5}$$

Homens: (4.6)

$$BMR = (13.75 \times W) + (5 \times H) - (6.76 \times A) + 66 \quad \text{[kcal/24h]}$$

Mulheres:

$$BMR = (9.56 \times W) + (1.85 \times H) - 4.68 \times A) + 655 \quad \text{[kcal/24h]}$$

onde:

H = altura [cm]

restantes referências na eq. 4.3

4.2 Cálculo Baseado no Tipo de Actividade

Na ausência de dados relativos à frequência cardíaca do individuo, o cálculo do número de calorias dispendidas (CB) é efectuado com base no índice MET ("The Metabolic Equivalent of Task"). Este índice é uma medida fisiológica do custo energético de um dado exercício físico. Com base nesta abordagem mais simplificada, as calorias dispendidas podem ser calculadas de acordo com a equação 4.7.

$$CB = MET \times W \times T \quad \text{[kcal]} \tag{4.7}$$

Os índices MET foram definidos de acordo com o "Compendium of physical activities" [2], da autoria do "Healthy Lifestyles Research Center", "School of Nutrition and Health Promotion" da Arizona State University https://sites.google.com/site/compendiumofphysicalactivities/. Na tabela 4.1 apresentam-se alguns valores típicos dos índices MET. De forma a adaptar os valores dos índices MET a cada índividuo (peso, altura e idade), foram aplicadas correcções de acordo com [4] — valor corrigido $CMET$ (eq. 4.8), para mais detalhes ver https://sites.google.com/site/compendiumofphysicalactivities/corrected-mets.

$$CMET = MET \times \frac{3.5 \text{ ml/kg/min}}{RMRCB[\text{ ml/kg/min}]} \tag{4.8}$$

Capítulo 5

Cálculo da Forma dos Utilizadores

A forma é um valor de 0 a 1. (Pode ser mudado alterando as variáveis `MAX_FORMA` e `MIN_FORMA`).

O cálculo da forma assume um número de dias nos quais todas as actividades feitas nesse intervalo têm influência para a forma (variável `DIAS_RELEVANTES`). Se `DIAS_RELEVANTES=24`, significa eu só as actividades nos últimos 24 dias têm influência para a forma. Isto serve para simular o facto de só as actividades mais recentes deverem ter influência na forma actual e quanto mais recentes as actividades, maior a influência na forma.

A cada dia é atribuído um "peso", o quanto esse dia vai valer para o cálculo final da forma. No entanto, não é feita uma distribuição equitativa dos pesos pelos dias. Dias mais distantes no tempo têm menos influência e portanto, menos peso. Essa "menos influência" é dada por uma taxa (variável TAXA) que representa o decréscimo na forma resultante de ficar 1 dia sem fazer nenhuma actividade. Se `TAXA=0.05`, significa que cada dia que se ande para trás, tem -5% de influência no cálculo da forma. A contribuição para a forma de cada dia é calculado multiplicando o peso desse dia pelo quociente entre o número de minutos que o utilizador fez de uma actividade e o número de minutos recomendado para essa actividade (número de minutos recomendado para 1 actividade = `Intensidade`). Somadas todas as contribuições de cada dia, tem-se a forma final.

Capítulo 6

Estatísticas dos Utilizadores

As estatísticas sobre as actividades dos utilizadores, para além de um requisito da aplicação, foi também útil para para a simulação de eventos (ver capítulo 7). Para esses dois propósitos foi criada uma classe `UserStats`, contendo os métodos de estatísticas. O excerto de código 6.1 apresenta os cabeçalhos dos métodos desenvolvidos.

Os três primeiros métodos recebem um utilizador e duas datas passadas como argumento, e devolvem informação sobre as actividades desse utilizador entre o período decorrido entre as duas datas. O método `getDistanceStats` devolve a distância que um utilizador percorreu no total. Para o cálculo, são tidas em conta todas as actividades subclasse de `Distance`. O método `getCaloriesStats` devolve as calorias gastas. Uma vez que a todas as actividades está associado um gasto de calorias, para o cálculo são tidas em conta todas as actividades do utilizador no período considerado. O método `getWorkoutStats` devolve o tempo total gasto em actividades. Tal como no método anterior, também este tem em conta todas as actividades do utilizador no período entre as datas passadas como argumento.

Para as actividades em que a performance possa ser avaliada por uma pontuação (`Contest` e `Skill`) foi criado um método adicional (quarto método do código 6.1). Este método recebe um utilizador e uma classe. A classe passada como argumento corresponde à modalidade da actividade que se pretende considerar. É devolvido um inteiro com o total de pontos conseguidos pelo utilizador no último ano e na actividade especificada.

Código 6.1: Métodos para cálculo de estatísticas (src/core/UserStats).

```
1 static public int getDistanceStats(User usr, GregorianCalendar startDate,
      GregorianCalendar endDate);
2
3 static public int getCaloriesStats(User usr, GregorianCalendar startDate,
      GregorianCalendar endDate);
4
5 /** @return time in hours **/
6 static public double getWorkoutStats(User usr, GregorianCalendar startDate,
```

Capítulo 7

Simulação de Eventos

7.1 Eventos do Tipo Corrida (Distance)

Para a simulação de eventos de corrida foi definido um método que tem em conta os seguintes factores:

- Recorde pessoal do utilizador para a distância da corrida;
- Parâmetro aleatório, variável entre 0.5 e 1.5 e que pode traduzir um ganho ou perda em relação ao recorde;
- Parâmetro de forma, tal como discutido no capítulo 5;
- Parâmetro relativo às condições meteorológicas, que pode traduzir um aumento do tempo do recorde de duas vezes (condições más) ou não afectar o tempo do recorde (em condições ideais);
- O método permite simular uma desistência, que foi definida com probabilidade de 1% para a idade de 20 (agravando para idades superiores, ou melhorando para idades inferiores).

Multiplicando todos os factores enunciados obtém-se um valor aleatório para cada etapa. Este método foi designado `getSimulationDistance` e é parcialmente apresentado no ínicio do excerto de código 7.1.

A simulação completa processa-se de acordo com as seguintes fases de cálculo:

1. Geração de resultados aleatórios para todas as etapas da corrida (usualmente 1 etapa = 1km), guardando os resultados de cada utilizador num `ArrayList`;
2. Este resultados são guardados num `Map`, em que a chave são os emails dos utilizadores e os valores os resultados referidos (penúltimo método no código 7.1);
3. Com os resultados indexados por chave de utilizador e em `ArrayList`, é possível definir a classificação em cada etapa, somando os tempos acumulados até à etapa pretendida e construindo um `TreeSet`, ordenado por tempo (último método no código 7.1).

Código 7.1: Métodos para a simulação de corridas (src/core/EventSimulation.java).

```
/** simulacao de uma etapa de uma corrida de um User */
static private long getSimulationDistance(User u, Distance act,
                                          int recordType, int stages) {
    (...)

    /*probabilida de desistir proporcional a faixa etaria, 1% para 20 anos*/
    if(Math.random()*20 < 0.01*u.getIdade())
        result = Long.MAX_VALUE/2; /*desistiu => duracao muito elevada*/
    else{
        /* result = [0.5-1.5]*prsBest*[10-1]*[2-1] */
        result = (long) (rndFact * prsBest * fitnessFact * weatherFact);
    }
    return result;
}

/** simulacao completa para todos os Users **/
static public Map<String,ArrayList<Long>> getAllResults(List<User> users, Distance act,
                                          int recordType, int stages);

/** treeSet para resultados ordenados por etapa(km) **/
static public TreeSet<SimulationPair> getStageClassification(Map<String,ArrayList<Long
    >> results, int stage);
```

7.2 Eventos do Tipo Torneio (Contest)

Na simulação de um torneio é necessário proceder às seguintes fases de cálculo:

1. Gerar jogos entre os utilizadores, em função do número de mãos definidas;
2. Simular os diferentes jogos;
3. Criar um `Map<String,Integer>` com os pontos acumulados por cada utilizador ao longo do torneio;
4. Gerar a tabela classificativa final, extraindo os resultados do `Map` e construindo um `TreeSet` ordenado por pontos.

O método de simulação de jogos é apresentado no código 7.2 e tem em conta os pontos acumulados na modalidade no último ano, bem como a forma actual de cada utilizador.

Código 7.2: Métodos para a simulação de uma disputa entre dois utilizadores (src/-core/EventSimulation.java).

```
/**
* Simula uma disputa entre dois utilizadores
* @param u1 utilizador 1
* @param u2 utilizador 2
* @param category categoria da actividade
* @return <0 se user 1 vence; =0 se empatam; ou >0 se user 2 vence
*/
static public int getSimulationContest(User u1, User u2, Class<? extends Activity>
    category){
  int user1Pts = UserStats.getPtsFromLastYear(u1, category);
  int user2Pts = UserStats.getPtsFromLastYear(u2, category);
  double user1Fitness = u1.getForma();
  double user2Fitness = u2.getForma();
  double rnd1 = 1 + (Math.random() - 0.50);
  double rnd2 = 1 + (Math.random() - 0.50);
  return (int) (rnd2*user2Pts*user2Fitness - rnd1*user1Pts*user1Fitness);
}
```

Capítulo 8

Conclusões

(...)

Bibliografia

[1] Ribeiro AN. Projeto prático de programação orientada aos objectos, lei e lcc. 2014.

[2] Ainsworth BE; Haskell WL; Herrmann SD; Meckes N; Bassett Jr DR; Tudor-Locke C; Greer JL; Vezina J; Whitt-Glover MC; Leon AS. 2011 compendium of physical activities: a second update of codes and met values. *Medicine and Science in Sports and Exercise*, 43(8):1575–1581, 2011.

[3] Endomondo. Calories - calculation method. http://gsfn.us/t/4183d. [Acedido em maio de 2014, publicado em maio de 2013].

[4] Harris J.A.; Benedict F.G. A biometric study of human basal metabolism. *Proceedings of the National Academy of Sciences USA*, 4(12):370–373, 1918.

[5] Uth N.; Sørensen H.; Overgaard K.; Pedersen PK. Estimation of vo2max from the ratio between hrmax and hrrest–the heart rate ratio method. *European Journal of Applied Physiology*, 91(1):111 – 5, 2004.

[6] Tanaka H.; Monahan K. D.; Seals D. R. Age-predicted maximal heart rate revisited. *Journal of the American College of Cardiology*, 37(1):153 – 156, 2001.

[7] Keytel L.R.; Goedecke J.H.; Noakes T.D.; Hiiloskorpi H.; Laukkanen R.; van der Merwe L.; Lambert E.V. Prediction of energy expenditure from heart rate monitoring during submaximal exercise. *Journal of Sports Sciences*, 23(3):289 – 97, 2005.

Apêndice A

Demo da aplicação