



Escola de Engenharia  
Departamento de Informática

Licenciatura em Engenharia Informática

# Projecto Java - FitnessUM

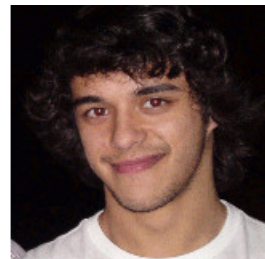
Programação Orientada aos Objectos



69303  
Bruno Pereira



66822  
Miguel Guimarães



69854  
João Mano

Braga, Junho de 2014

# Conteúdo

<b>1</b>	<b>Estrutura da aplicação</b>	<b>3</b>
1.1	Actividades . . . . .	3
1.1.1	Classe abstracta Activity . . . . .	4
1.1.2	Indoor,Outdoor e actividades desportivas . . . . .	4
1.1.3	Comparadores e Interfaces . . . . .	4
1.2	Utilizadores . . . . .	5
1.2.1	Classe abstracta Person . . . . .	6
1.2.2	Classes User e Admin . . . . .	6
1.2.3	Comparators . . . . .	7
1.2.4	Statistics . . . . .	7
1.3	Recordes Pessoais . . . . .	7
1.3.1	Classe abstracta Record . . . . .	7
1.3.2	DistancePerTime e TimePerDistance . . . . .	8
1.3.3	ListRecords . . . . .	8
1.3.4	Interfaces . . . . .	9
1.4	Eventos . . . . .	9
1.4.1	Classe abstracta <i>Event</i> . . . . .	9
1.4.2	Tipo de Evento . . . . .	10
1.4.3	Simulação . . . . .	10
1.4.4	Ranking . . . . .	11
1.5	Fórmulas . . . . .	11
1.5.1	Fórmula das Calorias . . . . .	11
1.5.2	Fórmula para a Simulação . . . . .	11
<b>2</b>	<b>Considerações finais</b>	<b>14</b>
2.1	Construtores . . . . .	14
2.2	Variáveis inutilizadas . . . . .	14
2.3	Actualização de estados . . . . .	14
<b>3</b>	<b>Anexos</b>	<b>15</b>
3.1	Imagens da aplicação . . . . .	15

## Lista de Figuras

1	Estrutura das actividades . . . . .	3
2	Comparador CompareActivity . . . . .	5
3	Exemplo de actividades que implementam as interfaces Distance, VerticalDistance e UserVs . . . . .	5
4	Estrutura das classes User e Admin . . . . .	6
5	Estrutura dos recordes . . . . .	8
6	Estrutura dos Eventos . . . . .	9
7	Menu login . . . . .	15
8	Menu registar na aplicação . . . . .	15
9	Menu principal do utilizador . . . . .	16
10	Menu adicionar actividade . . . . .	17
11	Menu ver recordes . . . . .	18
12	Menu ver actividades(estatísticas) . . . . .	19
13	Menu eventos do administrador . . . . .	20

# 1 Estrutura da aplicação

## 1.1 Actividades

Foram definidas as seguintes actividades desportivas para a nossa aplicação:

- Yoga
- Running
- Aerobics
- Skating
- Swimming
- Sailing
- IndoorCycling
- Walking
- Handball
- Tennis
- Basketball
- Skiing
- TableTennis
- Cycling
- Boxing
- MountainBiking
- Badminton
- Orienteering
- VolleyBallIndoor
- Snowboarding
- Football
- Polo
- VolleyBallBeach

Para a implementação destas actividades foi usada a seguinte estrutura:

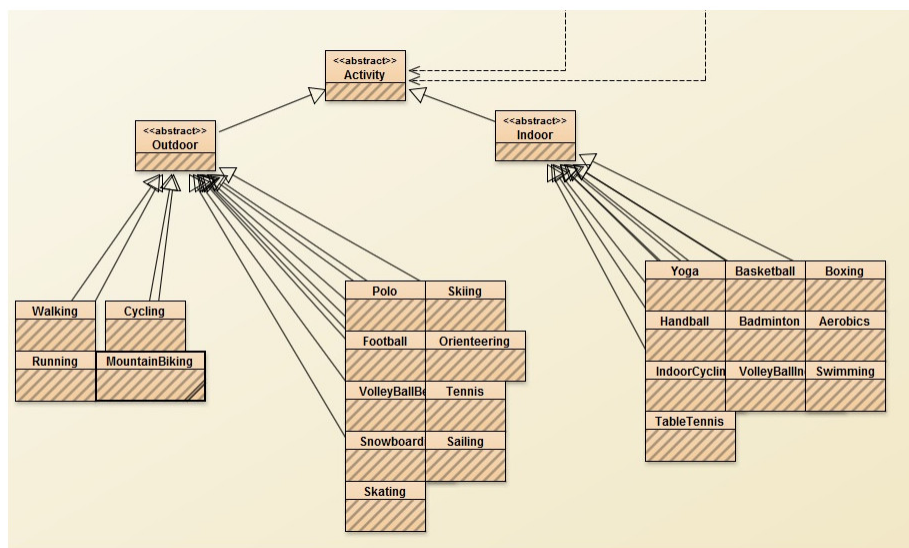


Figura 1: Estrutura das actividades

### 1.1.1 Classe abstracta Activity

Esta é a classe mais abstracta que contém o conceito de actividade. Contém variáveis comuns a todas as actividades:

- *String name*, nome da actividade criada.
- *GregorianCalendar date*, data de quando se realizou a actividade.
- *double timeSpent*, tempo gasto na actividade.
- *double calories*, campo preenchido pela aplicação de uma fórmula.

tal como os construtores, *getters* e *setters*.

### 1.1.2 Indoor,Outdoor e actividades desportivas

Todas as actividades desportivas tem um aspecto importante,o clima caso sejam praticadas ao ar livre.

Devido a este aspecto foram criadas duas classes abstractas,subclasses de *Activity*,para essa distinção.

- Outdoor,contém a variável: *String weather*
- Indoor

Todas as actividades desportivas são subclasses de *Indoor* ou *Outdoor* como exemplificado na figura 1.

### 1.1.3 Comparadores e Interfaces

Para organizar as actividades criaram-se dois tipo de comparadores,como exemplificado na figura 2

- CompareActivity- Compara a actividade pela data da realização da mesma.
- CompareActivityByTime- Compara a actividade pelo tempo gasto na realização desta.

Depois de uma análise às actividades desportivas, ficou claro que para certas actividades deviam-se registar distancias e que para outras registar pontuações,neste seguimento foram criadas as seguintes interfaces:

- UserVs-Interface de métodos relacionados com pontos(pontos próprios e pontos do adversário)
- Distance -Interface de métodos relacionados com actividades de distancia.
- VerticalDistance- Interface de métodos relacionados com actividades de distancia vertical

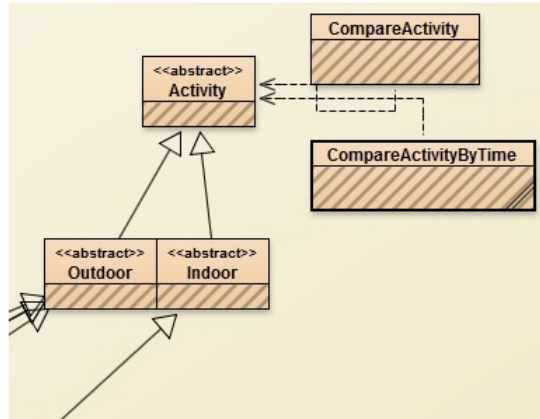


Figura 2: Comparador CompareActivity

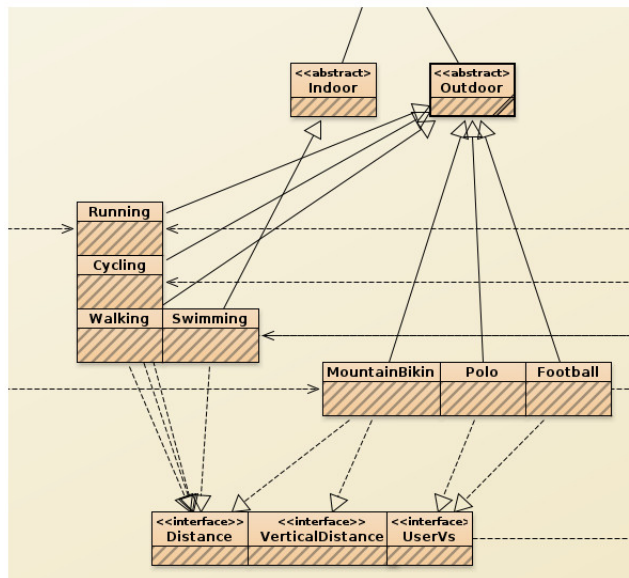


Figura 3: Exemplo de actividades que implementam as interfaces Distance, VerticalDistance e UserVs

## 1.2 Utilizadores

Para distinguir utilizadores regulares de administradores criou-se a estrutura exemplificada na figura 4

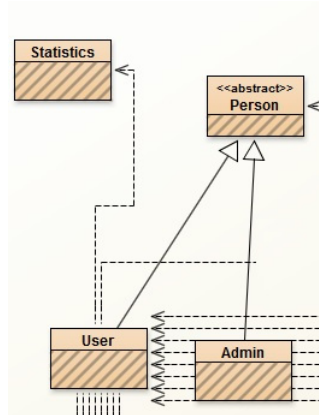


Figura 4: Estrutura das classes User e Admin

### 1.2.1 Classe abstracta Person

Classe geral para todo tipo de utilizador. As suas variáveis são:

- *String email;*
- *String password;*
- *String name;*
- *char gender;*
- *GregorianCalendar dateOfBirth;*

### 1.2.2 Classes User e Admin

As subclasses de Person referem-se a dois possíveis tipos de utilizador, utilizador normal ou utilizador com privilégios de administrador.

A classe Admin não tem métodos ou variáveis adicionais, visto que este tipo de utilizador apenas opera sobre a base de dados da aplicação.

A classe User adiciona as seguintes variáveis:

- *int height;*
- *double weight;*
- *String favoriteActivity;*
- *TreeSet<Activity> userActivities* - Actividades realizadas pelo utilizador;

- *TreeSet<String> friendsList* - Lista dos amigos do utilizador;
- *TreeMap<String, ListRecords> records* - Lista dos seus recordes pessoais;
- *TreeSet<String messageFriend* - Lista de pedidos de amizade;

Respectivos métodos *getters* e *setters*, construtores e métodos auxiliares para a gestão de amigos/pedidos de amizade, recordes pessoais, das suas actividades e estatísticas relevantes. Ainda contém funções auxiliares para a simulação de eventos.

### 1.2.3 Comparators

O tipo *Person* tem apenas um comparator:

- *ComparePersonByName* - que ordena por ordem alfabética do seu nome.

### 1.2.4 Statistics

A classe *Statistics* é usada para mostrar ao utilizador dados relevantes das suas actividades, estes podem ser discriminados por um dado mês ou por um ano. As suas variáveis são:

- *double timeSpend;*
- *double calories;*
- *double distance;*

contém os respectivos métodos *getters* e *setters* e construtores.

## 1.3 Recordes Pessoais

Para registar os recordes chegou-se a estrutura da fig 5:

Como se pode verificar na figura 5, apenas as seguintes actividades contêm recordes:

- Running
- Cycling
- Walking
- MountainBiking
- Swimming

### 1.3.1 Classe abstracta Record

Esta classe representa todos os recordes que o utilizador pode bater. Contém apenas uma variável:

- *String name*-Nome do tipo de recorde a bater(ex: 1km,10 miles,Cooper...)

métodos construtores, *getName()* e *isEmpty()* que verifica se esse recorde existe ou não.



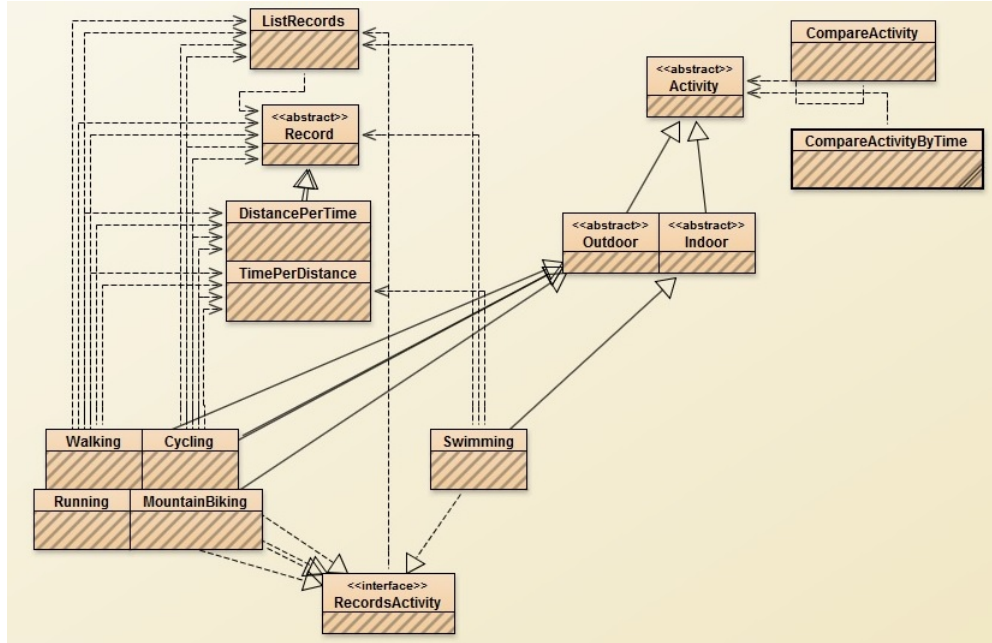


Figura 5: Estrutura dos recordes

### 1.3.2 DistancePerTime e TimePerDistance

Estas classes simbolizam os dois diferentes tipos de recordes.

DistancePerTime é um recorde em que o objectivo é fazer a maior distância para um dado tempo. As suas variáveis são:

- *double recordTime* - Tempo do recorde;
- *double distance* - Distância registada;

Enquanto que TimePerDistance representa um recorde de menor tempo para uma certa distância. As suas variáveis são:

- *double recordDistance* - Distância do recorde;
- *double time* - Tempo registado;

Estas duas classes têm os mesmos métodos, no entanto os métodos *update* e *setStatistics*, estão implementados de maneiras diferentes, tendo em conta que em DistancePerTime, quanto maior a distância melhor é o recorde, e no caso do TimePerDistance, o melhor recorde é o de menor tempo.

### 1.3.3 ListRecords

Classe que agrupa todos os recordes de uma actividade. Tem como variáveis:

- *String name* - Aqui o nome simboliza o tipo de actividade (Ex: Running, Walking...);

- *ArrayList<Record> recs* - Lista dos recordes;

Tem implementado métodos construtores, *getters*, *setters* e ainda um método *updateList()* que aplica a função *update()* a todos os objectos *Record* da lista. (Substitui na lista original caso recorde da segunda lista seja melhor).

### 1.3.4 Interfaces

Nesta fase, visto que nem todas as actividades desportivas implementarem recordes, chegou-se então à conclusão que estas actividades precisam sempre de devolver a lista de recordes registados, então implementou-se a seguinte interface:

- *RecordsActivity*;

Que contém o seguinte método:

- *getListRecords*;

## 1.4 Eventos

A figura 6 representa a nossa estrutura para os eventos.

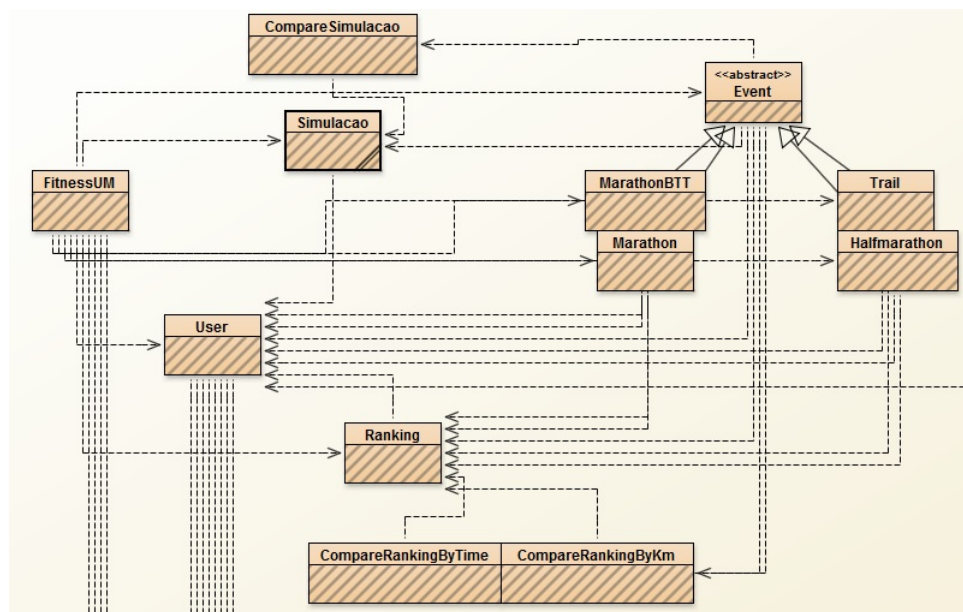


Figura 6: Estrutura dos Eventos

### 1.4.1 Classe abstracta *Event*

Classe com o conceito mais abstracto de Evento, contém as variáveis:

- *String name* - Nome do evento;

- *String tipoActivity* - Tipo de actividade (Running, Walking, ...);
- *String location* - Onde se realiza a prova;
- *int maxParticipants* - Número máximo de participantes;
- *int participants* - Número actual de participantes ;
- *GregorianCalendar deadline* - Data limite de inscrição;
- *GregorianCalendar date* - Data de realização;
- *double duration* - Duração da prova;
- *TreeSet<User> participantsList* - Lista de participantes;
- *TreeSet<Ranking> ranking* - Classificação dos que acabaram a prova;
- *TreeSet<Ranking> desistentes* - Participantes que desistiram da prova;
- *TreeSet<Simulacao> simula* - Informação relevante para simular cada concorrente;

respectivos *getters* e *setters* e os vários construtores. Ainda tem métodos auxiliares para, adicionar um *User*, *Ranking* (desistente ou não) e *Simulacao* aos respectivos *Sets* e para mostrar a classificação geral do evento.

### 1.4.2 Tipo de Evento

Subclasses de Evento (*Marathon*, *HalfMarathon*, *MarathonBTT* e *Trail*), todas estas contem mais uma variável *distance*, que nos casos de *Marathon* e *HalfMarathon* são variáveis *final*, porque este tipo de eventos tem distâncias específicas. Não tem métodos auxiliares para além de *getDistance()*.

### 1.4.3 Simulação

Para guardar dados relevantes à simulação de cada utilizador para um evento, foi criada a classe *Simulacao*. A simulação de cada evento é feita actualizando os dados desta classe a cada km.

- *double tempoGeral* - Tempo acumulado do utilizado na realização da prova.
- *double tempoMedio* - Tempo médio por km.
- *int kmDesiste* - Número de km que o utilizador aguenta durante a prova.
- *User u* - Utilizador associado á simulação.

esta classe, para além dos métodos construtores e *getters* e *setters*, contém apenas um método *actualiza*, que simula a passagem de uma distância (passada como argumento), usando o tempo médio por km e aplicando um factor aleatório (usando *Math.random()*).

#### 1.4.4 Ranking

Cada evento, para organizar a sua classificação, utiliza duas colecções de objectos da classe *Ranking*. Uma delas, usada para organizar todos os participantes, que concluíram a prova, por ordem de chegada, a outra onde estão os aqueles que não terminaram, organizados por número de quilómetros realizados.

Esta classe usa as seguintes variáveis:

- *double time* - Tempo de realizado no evento;
- *int km* - Número de quilómetros realizados;
- *User athlete* - Utilizador;

Das variáveis *time* e *km*, apenas uma irá ter algum valor para cada utilizador, visto que esta classe é usada para ordenar classificações finais, cada pessoa tem ou um tempo de conclusão do evento ou o número do quilómetro em que desistiu. *Ranking* contém os métodos *getters* e *setters* relevantes, construtores, e para além dos métodos essenciais, foram implementados dois métodos *toString* alternativos, para os dois casos.

### 1.5 Fórmulas

Em certos momentos do trabalho surgiu a necessidade de codificar fórmulas.

Tal aconteceu para calcular as calorias gastas em cada actividade e para a simulação dos eventos.

#### 1.5.1 Fórmula das Calorias

**MET(Metabolic Equivalent of Task)**- É uma medida fisiológica que expressa o custo energético de cada actividade física.

Sabendo o que MET's representa, e retirando essa medida, para cada actividade ,pelo seguinte quadro: [http://www.cdof.com.br/MET\\_compendium.pdf](http://www.cdof.com.br/MET_compendium.pdf)

Criou-se a seguinte fórmula das calorias para cada actividade:

$$\text{Calorias} = \text{mets} * \text{pesoDoUtilizador} * (\text{tempoGasto}(\text{min})/60)$$

#### 1.5.2 Fórmula para a Simulação

Para inferir um valor médio de minutos/km foi seguido o seguinte raciocínio:

1. Calculou-se um tempo médio em função do evento.
2. Contou o número de actividades praticadas do tipo do evento(ex: evento-Marathon, tipo do evento-Running).

3. Aplicou-se a fórmula idealizada.

Para calcular o tempo médio em função do evento, fez-se uma distinção entre MarathonBTT e os outros eventos.

Para o evento **MarathonBTT** percorreu-se todas as actividades do tipo "MountainBiking" e para cada uma delas:

- Calculou-se um factor, de 0 a 1, em função do parâmetro VerticalDistance (quanto maior, maior é o factor).
- Somou-se a distancia com o acumulado da mesma.
- Somou-se o acumulado do tempo com o tempo gasto/ distancia feita.

No fim calcula-se o tempo médio total em função do tempo médio calculado com o factor médio e distancia média realizadas nas actividades.

Para os restantes eventos, a ideia foi a mesma, apenas não se usou o factor.

Tendo o tempo médio calculado ( $tm$ ) a fórmula para o calculo do tempo médio final é a seguinte:

$$\text{tempo} = tm + (1 * tabWeather(weather)) + (1 * tabTemp(temperatura)) - (n^o/100) + (age/100)$$

tabWeather — > método que devolve um factor (de 0 a 1) em função dos seguintes climas:

- Sol
- Sol intenso
- Sol intenso com ventos fortes
- Chuva
- Chuva com ventos fortes
- Chuva intensa
- Chuva intensa com ventos fortes
- Trovoada
- Trovoada com ventos fortes
- Nublado

tabTemp- >método que devolve um factor (de 0 a 1) em função da temperatura, em graus celsius.

nº - > número de actividades praticadas do tipo do evento.

Em **cada km** da simulação o usa-se este tempo calculado e multiplica-se por  $(Math.random() + 0.5)$ , factor aleatório que aumenta ou diminui o tempo no km do participante.

Para a possibilidade de um participante do evento **desistir num certo Km** foi usada a seguinte estratégia:

1. Calculou-se a idade do participante.
2. Calculou-se uma probabilidade.
3. Calculou-se o possível km em que desiste.

A probabilidade foi calculada em função da idade:

- age < 15- factor =  $Math.random() + 0.1$ ;
- age < 20- factor =  $Math.random() + 0.5$ ;
- age < 25- factor =  $Math.random() + 0.7$ ;
- age < 30- factor =  $Math.random() + 0.85$ ;
- age < 35- factor =  $Math.random() + 0.9$ ;
- age < 40- factor =  $Math.random() + 0.6$ ;
- age < 45- factor =  $Math.random() + 0.5$ ;
- age < 50- factor =  $Math.random() + 0.4$ ;
- age < 55- factor =  $Math.random() + 0.3$ ;
- age < 60- factor =  $Math.random() + 0.2$ ;
- age < 65- factor =  $Math.random() + 0.1$ ;
- age >= 65- factor =  $Math.random() + 0.05$ ;

Finalmente multiplica-se este factor pela totalidade de km do evento, e se este km for menor que a distancia total do evento,então será o km de desistência do participante.

## 2 Considerações finais

### 2.1 Construtores

Determinadas classes desta aplicação contém construtores que não estão a ser utilizados por nós, foram sendo adicionados enquanto criávamos estas classes, no entanto, mesmo não sendo utilizados, podem ter utilidade futura fora desta aplicação.

### 2.2 Variáveis inutilizadas

Na classe *Event* a variável *duration* nunca é usada, pois, na criação desta classe pensávamos que íamos precisar da duração do evento, como não demos conta da sua inutilidade até estarmos num estado mais avançado do projecto, decidimos simplesmente ignorar esta variável em vez de a remover. O utilizador nunca terá conhecimento da existência deste problema, pois nunca é pedido esta informação e apenas usamos um construtor que a põe a zero.

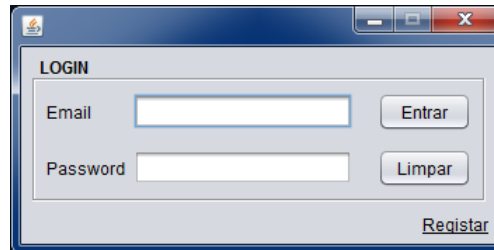
### 2.3 Actualização de estados

Durante os testes à funcionalidade da FitnessUM, deparamos com o problema de que, ao ser removida uma actividade, a sua informação continuava a ter relevância para as estatísticas e também para os recordes. Para melhorar a nossa aplicação foi mudada a implementação das estatísticas para apenas mostrar as actividades que ainda estão na aplicação, em vez de serem actualizadas na adição de uma nova actividade.

Para os recordes, no entanto, não foi possível implementar uma funcionalidade equivalente por motivos de tempo, e assim os recordes mostrados ao utilizador são os de todas as suas actividades realizadas, mesmo as que foram removidas.

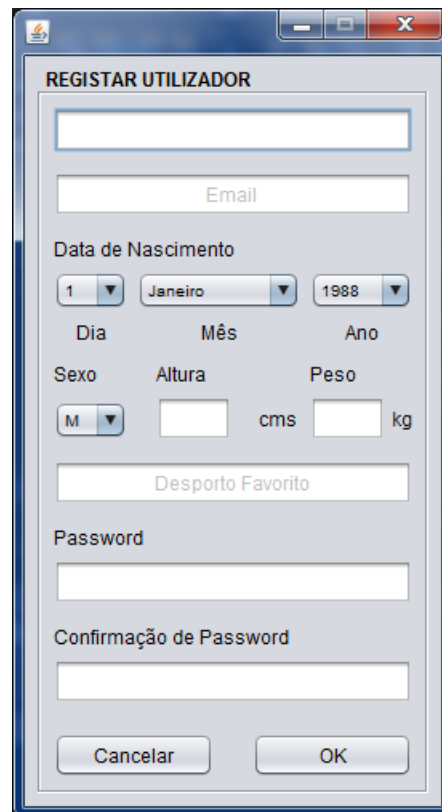
## 3 Anexos

### 3.1 Imagens da aplicação



A screenshot of a Windows-style application window titled "LOGIN". The window has a blue title bar with standard minimize, maximize, and close buttons. The main content area is light gray and contains two text input fields: "Email" and "Password". To the right of the "Email" field is a button labeled "Entrar". To the right of the "Password" field is a button labeled "Limpar". At the bottom right of the window, there is a link labeled "Registar" with a red underline.

Figura 7: Menu login



A screenshot of a Windows-style application window titled "REGISTAR UTILIZADOR". The window has a blue title bar with standard minimize, maximize, and close buttons. The main content area is light gray and contains several form fields and buttons. At the top is a large empty text input field. Below it is a text input field labeled "Email". Underneath is a section for "Data de Nascimento" (Date of Birth) with three dropdown menus for "Dia" (Day), "Mês" (Month), and "Ano" (Year). The "Dia" dropdown shows "1", "Mês" shows "Janeiro", and "Ano" shows "1988". Below this are labels for "Sexo", "Altura", and "Peso". The "Sexo" dropdown shows "M". The "Altura" field is followed by "cms" and the "Peso" field is followed by "kg". Below these is a text input field labeled "Desporto Favorito". At the bottom are two text input fields labeled "Password" and "Confirmação de Password". At the very bottom are two buttons: "Cancelar" and "OK".

Figura 8: Menu registar na aplicação



The screenshot shows a web application window titled "user1 - Utilizador". At the top, there are four buttons: "+ ATIVIDADE", "Ver pedidos de amizade", "+ AMIGO", and "LOGOUT". Below these is a navigation bar with tabs: "Home", "Ver Atividades", "Recordes", "Amigos", and "Eventos". The main content area is divided into two sections. The first section, "DADOS PESSOAIS", contains fields for "Nome" (user1), "Email" (1), "Peso" (70.0), "Altura" (170), "Data de Nascimento" (15/Novembro/1991), and "Desporto Favorito" (Futebol). The second section, "ESTATÍSTICAS", displays three metrics: "Tempo gasto" (279.0), "Distância percorrida" (118.0), and "Calorias queimadas" (2278). To the right of these metrics are radio buttons for "Mensais" (selected) and "Anuais". At the bottom of the statistics section are navigation buttons: "< Mês", "Junho (2014)", and "Mês >".

user1 - Utilizador

+ ATIVIDADE Ver pedidos de amizade + AMIGO LOGOUT

Home Ver Atividades Recordes Amigos Eventos

**DADOS PESSOAIS**

Nome user1 Peso 70.0

Email 1 Altura 170

Data de Nascimento Desporto Favorito

15 Novembro 1991 Futebol

**ESTATÍSTICAS**

Tempo gasto 279.0 Mensais

Distância percorrida 118.0 Anuais

Calorias queimadas 2278

< Mês Junho (2014) Mês >

Figura 9: Menu principal do utilizador

**NOVA ATIVIDADE**

Nome

Desporto Running

Data

Hora

Duração

Clima

Distância

KMs

Vertical

KMs

Pontuação

0 Pessoal

0 Adversário

Figura 10: Menu adicionar actividade

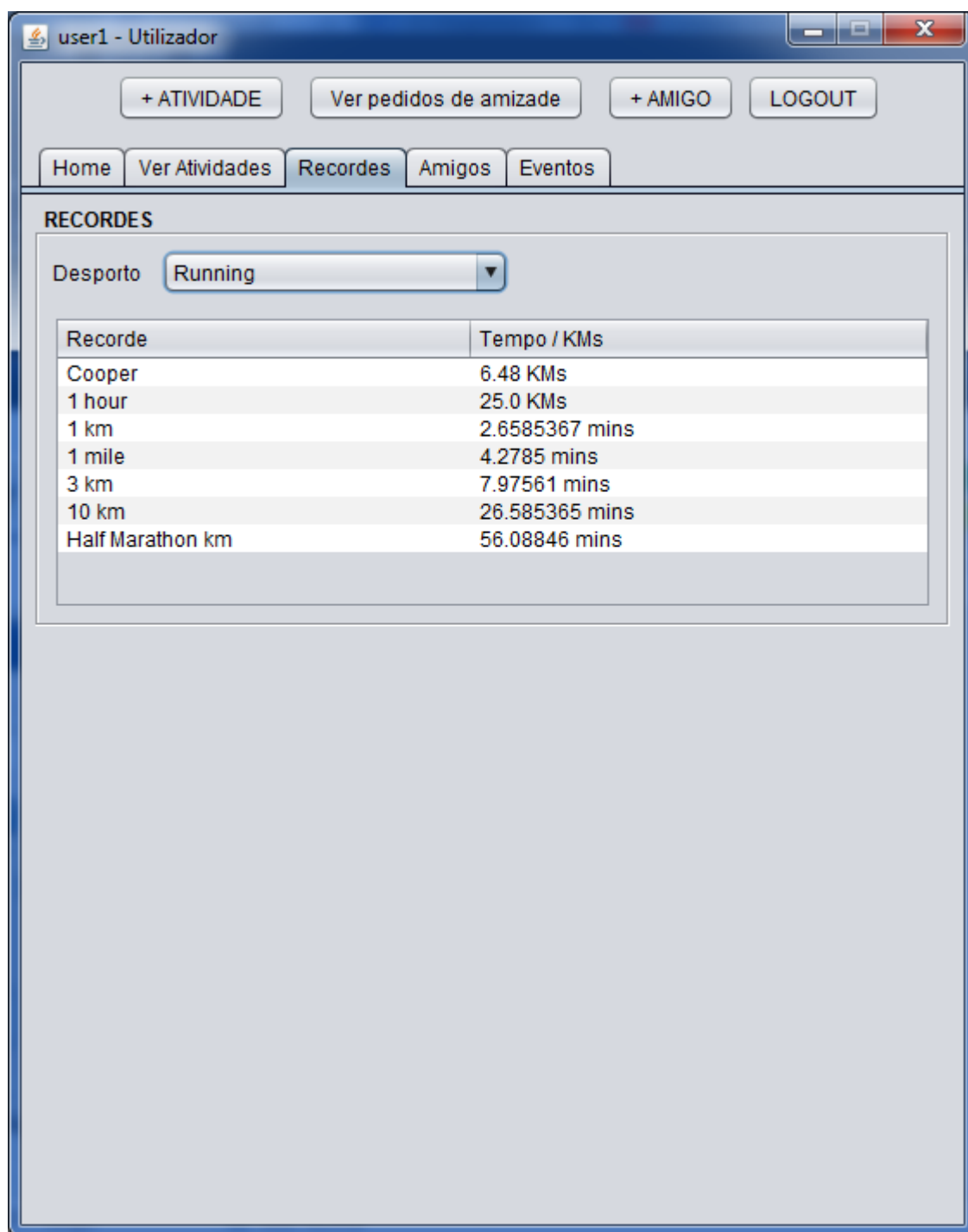


Figura 11: Menu ver recordes

user1 - Utilizador

+ ATIVIDADE Ver pedidos de amizade + AMIGO LOGOUT

Home Ver Atividades Recordes Amigos Eventos

### ATIVIDADE

Nome  ☒ 10 Últimas  
☐ Todas

Desporto

Data     
Dia Mês Ano

Hora  h  m

Duração   Calorias   
Horas Mins

Clima

Distância  KMs  
Vertical  KMs

Pontuação  Pessoal  
 Adversário

REMOVER

<< < 1/3 > >>

Figura 12: Menu ver actividades(estatísticas)

user1 - Utilizador

+ ATIVIDADE   Ver pedidos de amizade   + AMIGO   LOGOUT

Home   Ver Atividades   Recordes   Amigos   Eventos

### ATIVIDADE

Nome:  ☒ 10 Últimas ☐ Todas

Desporto:

Data:     
Dia   Mês   Ano

Hora:  h  m

Duração:   Calorias:   
Horas   Mins

**Clima**

**Distância**

KMs

Vertical

KMs

**Pontuação**

Pessoal

Adversário

<<   <   1/3   >   >>

Figura 13: Menu eventos do administrador