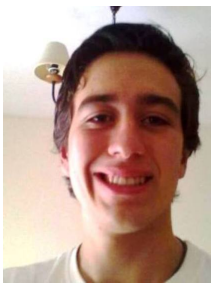
Universidade do Minho
Escola de Engenharia

Joana Arantes
(57810)

José Pereira
(67680)



Pedro Cunha
(67677)

Programação Orientada aos Objectos

Relatório: Projecto Fitness UM

Licenciatura em Engenharia Informática

Ano lectivo 2013/2014

Grupo 39

7 de Junho de 2014

Conteúdo

1 Introdução

No âmbito da unidade curricular de Programação Orientada aos Objectos, do 2º ano da Licenciatura em Engenharia Informática, foi elaborado este trabalho prático, de nome Fitness UM, utilizando a linguagem de programação Java, com a utilização do ambiente Eclipse (para facilitar o design da interface gráfica).

Este trabalho prático consiste numa aplicação que gere actividades desportivas de fitness, que permite ao seu utilizador registar as actividades desportivas realizadas e visualizar as actividades efectuadas pelos seus amigos.

Esta aplicação é semelhante às conhecidas redes sociais de fitness, como o Endomondo (`http://www.endomondo.com/`), o Strava (`http://www.strava.com/`) e o Runkeeper (`http://runkeeper.com/`).

2 Estrutura da aplicação

Em geral, a estrutura da aplicação está dividida pelas seguintes partes:

- **1 -** Parte de lógica aplicacional necessária à criaçao das actividades de fitness;
- **2 -** Parte de rede de utilizadores, registo e manutanção de utilizadores, seus amigos e actividades associados a eles;
- **3 -** Interface de utilização através da classe *FitnessUM*.

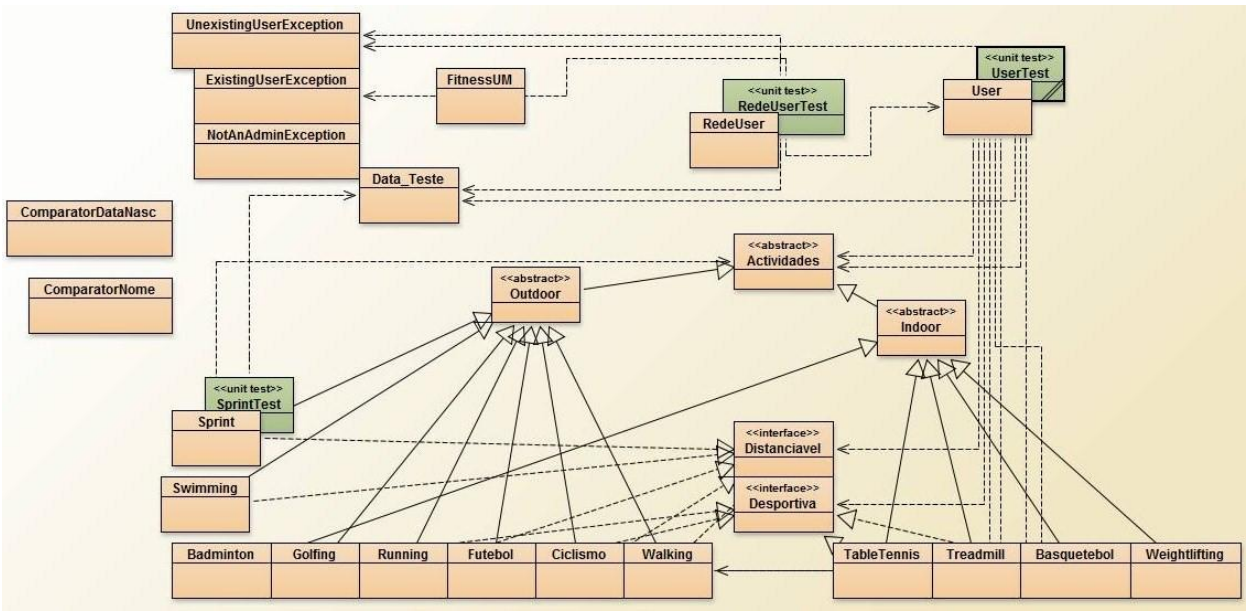


Figura 1: Estrutura da aplicação

2.1 Classes utilizadas

Como é necessário interligar conceitos e tarefas, através de envio de mensagens aos objectos criados, foi necessário criar diferentes tipos de classes.

As classes estão separadas da seguinte maneira:

2.1.1 *Classe Abstract Actividades*

A definição do que é uma actividade é uma parte neste projecto extremamente importante para o bom funcionamento do mesmo. Assim, decidimos declarar esta classe como abstracta, de forma a poder implementar, posteriormente, vários tipos de actividades diferentes. Existem apenas alguns métodos que são obrigatórios serem subclasses desta classe, como é o caso do método de cálculo de calorias queimadas, o método *toString* e o método *clone*.

Cada actividade terá obrigatóriamente uma duração (definimos essa duração em segundos, para facilitar conversões) e uma data de realização (definida como um *GregorianCalendar*.

2.1.2 *Classe Abstract Indoor*

Para além das variáveis de instância da sua *superclasse(Actividades)*, terá o nome de um estabelecimento, como por exemplo, o nome do ginásio em que se realizou certa actividade. Sendo abstracta, esta classe não é obrigada a implementar os métodos deixados como abstractos na sua superclasse.

2.1.3 *Classe Abstract Outdoor*

Para além das variáveis de instância da sua *superclasse(Actividades)*, terá um inteiro para a temperatura, e uma string para a informação do estado do tempo. Sendo abstracta, esta classe não é obrigada a implementar os métodos deixados como abstractos na sua superclasse.

2.1.4 As actividades

De forma a exemplificar, e como requerido pelo enunciado do projecto, as classes *Badminton*, *Basquetebol*, *TableTennis*, *Treadmill*, *Weightlifting*, *Ciclismo*, *Futebol*, *Golfing*, *Running*, *Sprint*, *Swimming* e *Walking*, representem as actividades que decidimos implementar, a partir das mais de 30 actividades diferentes apresentadas no enunciado do projecto.

2.1.5 Os utilizadores

Para auxiliar a ordenar os elementos da rede de utilizadores, foram criadas duas classes chamadas de *Comparator*, uma por nome, outra por data de nascimento, usando o método de ordenação natural, de *Strings* e de *GregorianCalendars*, respectivamente.

2.1.6 As interfaces

Para "etiquetar"as actividades, obrigando-as a ter implementados certos métodos, foram criadas duas interfaces *Distanciavel* e *Desportiva*, em que a interface *Distanciavel* obriga as classes que a quiserem implementar de aplicar o método *getDistancia()*. De forma análoga, todas as classes que quiserem implementar a interface *Desportiva*, terão de aplicar os métodos *getScored()* e *getConceded()*. Isto trará vantagens porque diminuirá drasticamente o código, pois, por exemplo, o predicado *(Futebol f instanceof Desportiva)* será verdadeira.

2.1.7 *Classe User*

Esta classe define concretamente um utliizador de uma rede social. Para o nosso caso, um utilizador define-se pelo seu nome, e-mail, password, desporto favorito, data de nascimento (*GregorianCalendar*), idade, altura (em centímetros), peso e género ('M' ou 'F').

2.1.8 *Classe RedeUser*

Esta classe será aquela que fará de base de dados da nossa rede social. Guardará os utilizadores (num par (E-mail,Utilizador)) e fará várias acções sobre um utilizador (actualizar a informação, por exemplo). Como se trata de uma base de dados que poderá crescer muito, as operações de leitura e escrita de ficheiro de objectos serão necessárias nesta classe.

2.2 Interface de utilização

A aplicação Fitness UM possui uma interface gráfica que utiliza o *cardLayout*, que facilita a navegação, melhora a apresentação e ajudou-nos a organizar melhor os dados.

De seguida encontram-se algumas fotos da interface gráfica:

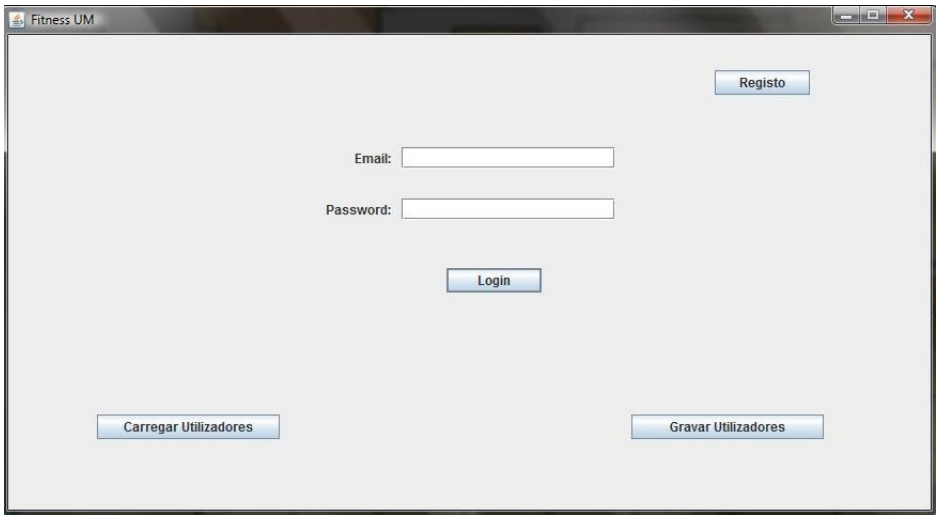


Figura 2: Interface gráfica - Menu 1

Figura 3: Interface gráfica - Menu 2

3 Capacidade de extensão

Durante a realização do projecto, tivemos sempre o cuidado de colocar as classes e os métodos o mais genérico possível, utilizando o encapsulamento de dados e a hierarquia de classes.

Os melhores exemplos disso são as classes *Actividades*, *Indoor* e *Outdoor*, que são facilmente extensíveis. Ao criar a classe de uma nova actividade, no cabeçalho da classe basta apenas pôr *"extends Actividades"*, sendo este um dos requisitos básicos do projecto.

4 Decisões mais importantes

Na escolha das estruturas de dados, para a representação dos dados, ao longo do projecto utilizamos vários tipos de estruturas: *ArrayList's* quando apenas necessitamos de percorrer sequencialmente os dados, *HashMap's* quando necessitamos de aceder o mais rápido possível a dados de forma não organizada, e *TreeSet's* quando necessitamos de ordenar grandes quantidades de informação (como é no caso de ordenar os utilizadores por nome ou data de nascimento). A escolha dos *ArrayList's* foi bastante fácil. Apesar de ser uma estrutura de dados básica que não aparenta ter nenhuma vantagem, existe uma redução do overhead de código "escondido"em relação a um *HashMap*, por exemplo, tornando o funcionamento do programa um pouco mais rápido. Porque motivo haveríamos de criar uma estrutura tão complexa como um *HashMap* para guardar as actividades, amigos e pedidos de amizade?

Para realizar a rede de utilizadores usamos um *HashMap* para conseguirmos tempos "tendencialmente constantes"no acesso aos utilizadores da rede.

5 Conclusão

Com o desenvolvimento deste projecto, conseguimos assimilar melhor o paradigma da Programação Orientada aos Objectos, o conceito de objecto e os seus atributos, hierarquia de classes e reutilização de código. Aprendemos também a usar a API da linguagem de programação JAVA, a percorrer a sua documentação e a realizar eficientemente um projecto de média dimensão.

Concluímos que a utilização de uma linguagem de Programação Orientada aos Objectos é útil e tem vantagens em relação ao uso de outro tipo de linguagens de outro paradigma, no sentido da organização do código e divisão do trabalho entre elementos de uma equipa. Também por limitações de tempo e deslocação, não foi possível implementar a funcionalidade de criação de eventos, por parte dos Admins. Por fim, queremos agradecer à equipa docente por nos ter dado a oportunidade de aprender mais uma tecnologia que é hoje imensamente importante no mercado de trabalho.