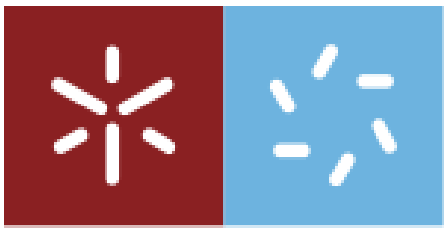

Universidade do Minho
Escola de Ciências



Programação Orientada aos Objetos

Trabalho Prático

Relatório de Desenvolvimento

André Oliveira Barbosa
A91684

Francisco António Borges Paulino
A91666

Luís Miguel da Silva Vila
A95675

12 de maio de 2024

Resumo

Área: Programação Orientada aos Objetos

No âmbito da disciplina de Programação Orientada aos Objetos foi-nos proposto construir uma aplicação que faça a gestão das actividades e planos de treino, de praticantes de actividades físicas.

Sendo o objetivo principal deste trabalho consolidar a aprendizagem da utilização da linguagem Java aplicada aos conceitos de Programação Orientada aos Objetos, entendemos que atingimos o proposto e que construímos um programa robusto capaz de fazer o pretendido.

Conteúdo

Lista de Figuras

Capítulo 1

Introdução

Este projeto consistiu no desenvolvimento de uma aplicação que faz a gestão das actividades e planos de treino de praticantes de actividades físicas. A aplicação permite que os utilizadores registem e criem atividades ou planos de treino realizados e consultar várias estatísticas sobre si mesmo. É também possível realizar saltos de tempo de forma a efetuar simulações. O projeto contém um ficheiro *output.txt* com um estado que pode ser carregado para o programa inicialmente utilizando a opção do menu *Carregar Ficheiro Inicial*.

Ao longo deste relatório vamos detalhar todas as classes e implementações relevantes.

Capítulo 2

Classes

2.1 Utilizador

```
1     private String nickname;
      private String password;
3     private String nomeUtilizador;
      private String emailUtilizador;
5     private String generoUtilizador;
      private LocalDate data_nascimento;
7     private double alturaUtilizador;
      private double pesoUtilizador;
9     private int frequencia_cardiaca_media;
      private double fator_multiplicativo;
11    private double total_calorias;
      private List<Atividade> historico_atividades;
13    private List<PlanoTreino> planos_treino;
```

Esta classe representa um utilizador com os atributos que consideramos coerentes e necessários para o bom funcionamento do programa. Dado que os utilizadores podem ser praticantes ocasionais, amadores ou profissionais, implementamos as seguintes subclasses:

2.1.1 Ocasional

```
      private int frequencia_pratica;
2     private String motivacao;
```

Esta subclasse de Utilizador representa um utilizador ocasional. O campo "motivação"dá a oportunidade deste registar uma frase que indique o seu propósito.
A fórmula de cálculo de fator para um utilizador ocasional depende da frequência (freq_p). Esta frequência representa uma estimativa do utilizador quanto ao número de atividades que faz por mês.

$$fator = (idade + peso + altura + freq\_p) \times 0.1$$

2.1.2 Amador

```
    private String nivelExp;
2   private String modalidade_favorita;
```

Esta subclasse de Utilizador representa um utilizador amador. Esta implementa o cálculo do fator que vai ser usado para o cálculo das calorias gastas após registar uma atividade. O campo "atividade favorita"permite que um utilizador amador registe a sua atividade favorita.
Um utilizador amador pode ser do tipo iniciante, intermédio ou avançado. Dependendo deste atributo, a sua fórmula de cálculo de fator é diferente.

- Para utilizadores amadores do tipo **Avançado**:

$$fator = (idade + peso + altura) \times 0.7$$

- Para utilizadores amadores do tipo **Intermédio**:

$$fator = (idade + peso + altura) \times 0.5$$

- Para utilizadores amadores do tipo **Iniciante**:

$$fator = (idade + peso + altura) \times 0.3$$

- Para outros tipos (debug):

$$fator = 6666$$

2.1.3 Profissional

```
    private String especialidade;
2   private double experiencia;
```

Esta subclasse de Utilizador representa um utilizador profissional. Este tem a possibilidade de especificar a sua especialidade e anos de experiência. A classe implementa o cálculo do fator que vai ser usado para o cálculo das calorias gastas após registar uma atividade.
Utilizadores profissionais têm diferentes fórmulas de cálculo de fator dependendo da sua idade e dos anos de experiência.

- Para utilizadores profissionais com **idade igual ou inferior a 50 anos**:

$$fator = (idade + peso + altura + AnosExp) \times 1.6$$

- Para utilizadores profissionais com **idade superior a 50 anos**:

$$fator = (idade + peso + altura + AnosExp) \times 1.2$$

2.2 Atividade

```
    private int codigo;
2   private String nome;
    private String descricao;
4   private int duracao;
    private LocalDate data_realizada;
6   private int freq_cardiaca_atividade;
    private double calorias_gastas_atividade;
8   private boolean isHard;
```

Atividade é uma superclasse abstrata onde temos os identificadores gerais. Consideramos esta uma boa abordagem pois as suas subclasses partilham vários atributos. Cada uma das subclasses de Atividade seguintes têm a sua própria fórmula de cálculo para as calorias gastas. Estas fórmulas foram implementadas de forma a se assemelharem o mais possível com um cenário real.

2.2.1 Patinagem

Com os atributos distância e percurso, esta subclasse contém também um método para calcular as calorias gastas na sua realização.

$$calorias = \left(0.035 \times fator \times (freq\_med + 0.12 \times freq\_ativ) \times \frac{tempo}{60.0}\right) + 0.75 \times dist$$

2.2.2 Remo

Com os atributos distância e percurso, esta subclasse contém também um método para calcular as calorias gastas na sua realização.

$$calorias = \left(0.035 \times fator \times (freq\_med + 0.12 \times freq\_ativ) \times \frac{tempo}{60.0}\right) + 0.75 \times dist$$

2.2.3 Corrida

Com os atributos distância, altimetria e percurso, esta subclasse contém também um método para calcular as calorias gastas na sua realização.

$$calorias = \left(0.035 \times fator \times (freq\_med + 0.12 \times freq\_ativ) \times \frac{tempo}{60.0}\right) + (0.75 \times dist + 0.25 \times alt)$$

2.2.4 Bicicleta

Com os atributos distância, altimetria e percurso, esta subclasse contém também um método para calcular as calorias gastas na sua realização.

$$calorias = \left(\left(0.035 \times fator \times (freq\_med + 0.12 \times freq\_ativ) \times \frac{tempo}{60.0}\right) + (0.75 \times dist + 0.25 \times alt)\right)$$

2.2.5 Abdominais

Com o atributo repetições, esta subclasse contém também um método para calcular as calorias gastas na sua realização.

$$calorias = \left(\left(0.035 \times fator \times (freq\_med + 0.12 \times freq\_ativ) \times \frac{tempo}{60.0}\right) + (0.75 \times rep)\right)$$

2.2.6 Alongamentos

Com o atributo repetições, esta subclasse contém também um método para calcular as calorias gastas na sua realização.

$$calorias = \left(0.035 \times fator \times (freq\_med + 0.12 \times freq\_ativ) \times \frac{tempo}{60.0}\right) + (0.75 \times rep)$$

2.2.7 ExtensaoPernas

Com os atributos repetições e peso, esta subclasse contém também um método para calcular as calorias gastas na sua realização.

$$calorias = \left(0.035 \times fator \times (freq\_med + 0.12 \times freq\_ativ) \times \frac{tempo}{60.0}\right) + (0.75 \times rep + 0.25 \times pes)$$

2.2.8 LevantaPesos

Com os atributos repetições e peso, esta subclasse contém também um método para calcular as calorias gastas na sua realização.

$$calorias = \left(0.035 \times fator \times (freq\_med + 0.12 \times freq\_ativ) \times \frac{tempo}{60.0}\right) + (0.75 \times rep + 0.25 \times pes)$$

2.2.9 Flexoes

Com o atributo repetições, esta subclasse contém também um método para calcular as calorias gastas na sua realização.

$$calorias = \left(0.035 \times fator \times (freq\_med + 0.12 \times freq\_ativ) \times \frac{tempo}{60.0}\right) + (0.75 \times rep)$$

2.3 PlanoTreino

```
1     private String nome;
      private LocalDate data;
3     private Map<Integer, Atividade> atividades;
      private int n_iteracoes;
```

Esta classe é usada para gerir e armazenar planos de treino. As suas principais funções são adicionar e configurar atividades num plano de treinos.

2.4 Controller

Esta classe conecta a lógica do projeto com a interação com o utilizador. A função desta classe é receber o input do utilizador, trabalhando-o de forma a, posteriormente, invocar a classe View para fornecer o output.

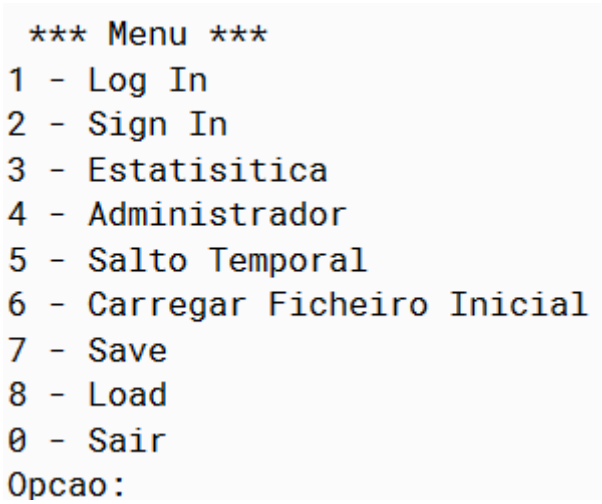

Figura 2.1: Menu inicial. Permite aceder às várias funionalidades do programa

2.5 Main

Classe responsável por inicializar o programa. Invoca o método run do controller.

2.6 Menu

Esta classe, que comunica unicamente com o Controller, implementa um menu em modo texto.

2.7 FitnessApp

```
    private Map<String, Utilizador> utilizadores;
2   private Map<Integer, Atividade> atividades;
    private Map<String,PlanoTreino> planos_treino;
4   private LocalDate dia_atual;
```

Para além de ser a classe central do programa, esta classe permite que se efetue saltos de tempo.

2.8 View

De uma forma geral, esta classe gere a forma como o output e apresentado ao utilizador.

2.9 Parser

O Parser e uma classe responsável por gerir os dados recebidos. Esta tem como objetivo ler todo o texto de um ficheiro, e organizar os seus elementos.

2.10 Exceções

Criamos as seguintes classes que representam exceções, de modo a facilitar a compreensão quando um erro ocorre.

2.10.1 UtilizadorNaoExisteException

2.10.2 UtilizadorExisteException

2.10.3 LinhaIncorretaException

2.10.4 PlanoTreinoNaoExisteException

2.10.5 PlanoTreinoExisteException

2.10.6 AtividadeExisteException

2.10.7 AtividadeNaoExisteException

Capítulo 3

Estrutura do Projeto

O nosso projeto segue a estrutura *Model View Controller* (MVC), estando por isso organizado em três camadas:

- A camada de dados (o modelo) é composta pela classe FitnessApp
- A camada de interação com o utilizador (a vista, ou apresentação) é composta unicamente pela classe View.
- A camada de controlo do fluxo do programa (o controlador) é composta pela classe Controller.

Todo o projeto baseia-se na ideia de encapsulamento.

Capítulo 4

Diagrama de Classes

De seguida, apresentamos um Diagrama de Classes com a arquitectura de classes que suporta o programa desenvolvido.

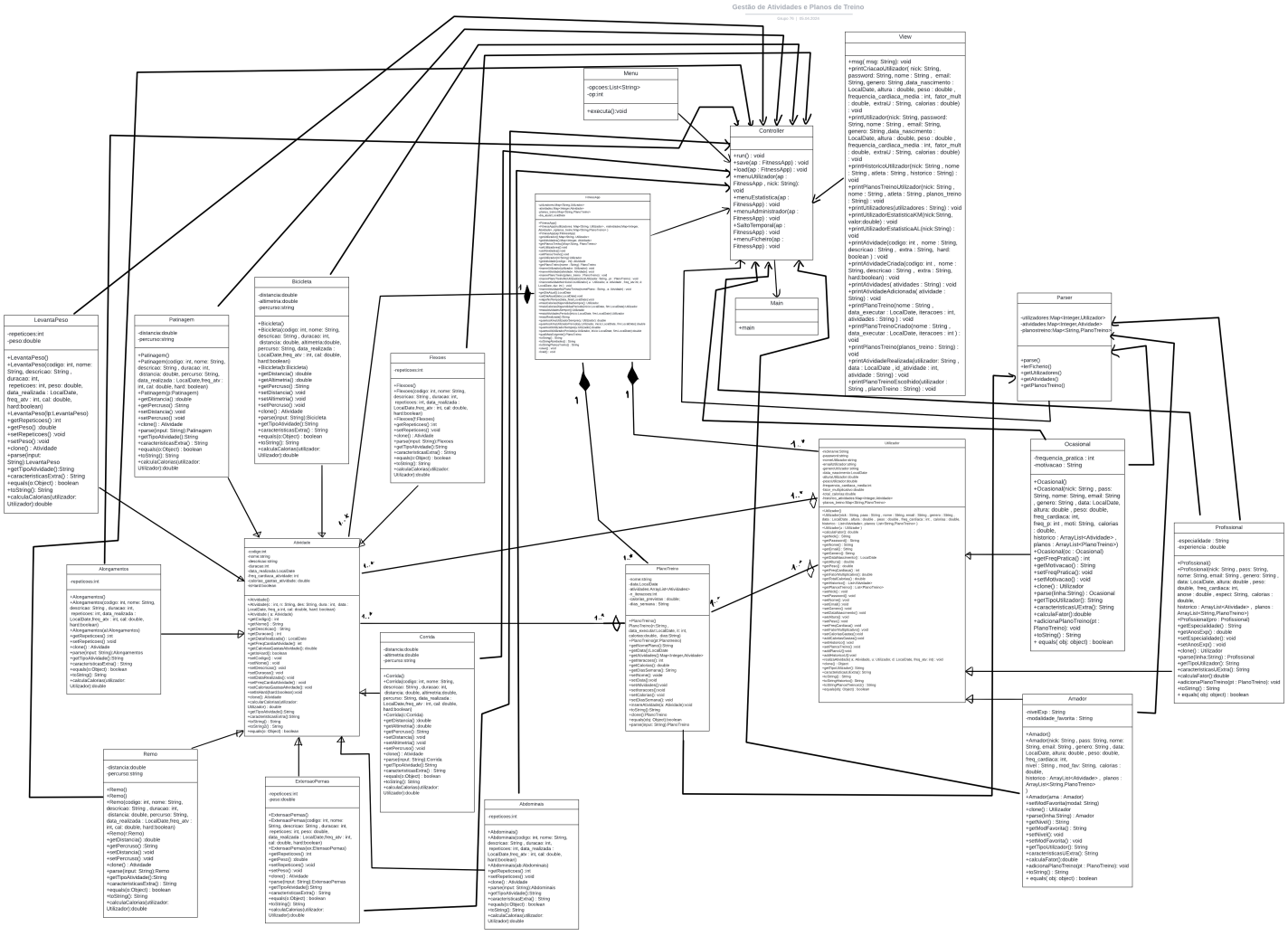

Figura 4.1: Diagrama de Classes UML do programa

Capítulo 5

Conclusão

Tendo finalizado o trabalho prático, propomo-nos agora a fazer uma análise crítica de todo este projeto de forma global tendo em conta todos os aspetos positivos e as dificuldades encontradas. Finalmente, propomo-nos a analisar criticamente o programa construído.

No que diz respeito aos aspetos positivos, realçamos o facto de termos concluído, na nossa ótica, o projeto com sucesso e de o programa final ser de algum modo eficiente, e capaz de ser futuramente ser expandido. Para além disso consideramos que a nossa organização durante o desenvolvimento foi bastante positiva dado que fizemos um diagrama UML inicial que nos permitiu ter uma visão completa do que pretendíamos para o projeto. As reuniões com o docente permitiram também esclarecer várias dúvidas e colmatar obstáculos que nos foram aparecendo.

Consideramos que as dificuldades principais surgiram na interpretação do enunciado, no desenvolvimento das estatísticas e na capacidade de expansão do programa. Para além disso, encontrar uma fórmula para o cálculo de calorias que devolvesse valores aproximados à realidade foi um desafio.

Em suma, consideramos que o balanço final do projeto foi positivo pois conseguimos ultrapassar boa parte das nossas dificuldades e elaborar um programa eficiente. Consideramos também que este projeto permitiu-nos consolidar conhecimento tanto na linguagem Java como no paradigma da programação orientada aos objetos.