

UNIVERSIDADE DO MINHO

ESCOLA DE ENGENHARIA



Programação Orientada aos Objetos

Licenciatura em Engenharia Informática

Aplicação de gestão de atividades e planos de
treino de praticantes de atividades físicas



Figure 1: Carlos Fernandes 100890

1 Introdução

Este relatório retrata o desenvolvimento de uma aplicação de gestão de atividades e planos de treino de praticantes de atividades físicas. Aplicação esta desenvolvida na linguagem de programação Java, de forma a aplicar os conceitos da Programação Orientada aos Objetos abordados ao longo do semestre. Na generalidade, a aplicação permite adicionar novos utilizadores, atividades e planos de treino, assim como simular a execução das atividades, obtendo indicadores das respetivas e permitindo visualizar dados estatísticos provenientes dessa simulação.

2 Arquitetura e estrutura de classes

2.1 Classe Atividade

```
public class Atividade implements Serializable {  
    private String id;  
    private String nome;  
    private String dificuldade;  
    private int tempo;  
    private double frequenciaCardiacaMedia;  
    private double caloriasConsumidas;  
    private LocalDateTime data;
```

Figure 2.1: Atributos da classe Atividade

Esta classe representa uma atividade genérica cujos atributos são:

- id - Identificador único da Atividade
- nome - Nome da atividade (Existência explicada na seção 2.1.2)
- dificuldade - Indica se a atividade é do tipo *Hard*
- tempo - Tempo de execução da atividade
- frequenciaCardiacaMedia - Indica a frequência média do utilizador durante a atividade
- caloriasConsumidas - Indica o total de calorias consumidas na atividade
- data - Indica a data de realização da atividade

2.1.1 Classes Distancia, DistanciaAltimetria, Repeticoes e RepeticoesPesos

Para diferenciar os tipos de atividades foram criadas subclasses para cada um, desta forma estas herdam os atributos já existentes na *SuperClass* Atividade e acrescentam novos parâmetros conforme as necessidades de cada categoria, por exemplo para as atividades de distância e altimetria são criados esses mesmos atributos próprios. Para além disso a fórmula de cálculo de calorias varia conforme o tipo de atividade. Inicialmente, a ideia era adaptar a fórmula de Harris-Benedict, mas após o confronto com dificuldades no processo, foi decidido criar uma fórmula do zero, ajustando os valores no processo para obter resultados realistas.

2.1.2 Classes referentes a atividades específicas

Seguindo o mesmo processo supracitado, foram criadas as classes das atividades específicas como Corrida_estrada, Trail_monte, etc. Sendo que, ao fazer a chamada do respetivo construtor, é especificado se a atividade em questão é do tipo *Hard*, assim como é definido

```
public class DistanciaAltimetria extends Atividade {
    private double distancia;
    private double altimetria;
}
```

Figure 2.2: Atributos da classe DistanciaAltimetria

o nome da atividade, para facilitar o desenvolvimento dos métodos estatísticos, assim como para evitar criar métodos *toString* para cada atividade específica.

```
public class Bicicleta_montanha extends DistanciaAltimetria {
    public Bicicleta_montanha(DistanciaAltimetria atividade) {
        super(atividade);
        this.setNome("Bicicleta_montanha");
        this.setDificuldade("Hard");
    }
}
```

Figure 2.3: Construtor da classe Bicicleta_Montanha

2.2 Classe PlanoTreino

2.2.1 Classe AtividadePlano

Esta classe representa a atividade que é inserida no plano de treino, contendo, então, a atividade base e o número de iterações da mesma.

```
public class AtividadePlano implements Serializable {
    protected Atividade atividadeBase;
    private int iteracoes;
}
```

Figure 2.4: Atributos da classe AtividadePlano

2.2.2 Classe PlanoTreino

Esta classe representa um plano de treino e tem como atributos o ID único, a data de realização e por fim, o conjunto de atividades em forma de *Map*, onde a chave é o ID da atividade.

```
public class PlanoTreino implements Serializable {
    private String id;
    private LocalDate dataRealizacao;
    private Map<String, AtividadePlano> atividades;
}
```

Figure 2.5: Atributos da classe PlanoTreino

2.3 Classe Utilizador

2.3.1 Classe Utilizador

Esta classe representa um utilizador, sendo que cada um tem um código único, o nome, a morada, o email, a frequência cardíaca média (em repouso) e o fator multiplicativo base, sendo este dependente do tipo de utilizador (Praticante ocasional, Amador e Profissional), sendo que este é utilizado juntamente com a frequência do utilizador como fator multiplicativo no cálculo de calorias. Para além disso, contém o conjunto de atividades isoladas, realizadas e planos de treino do utilizador, sendo estes *Maps* cujas *keys* são os IDs das atividades e o ID dos planos de treino, respetivamente.

```
public class Utilizador implements Serializable { 3 inheritors

    private int codigo;
    private String nome;
    private String morada;
    private String email;
    private double frequenciaCardiacaMedia;
    private double fatorMultiplicativoBase;
    private Map<String, Atividade> atividadesIsoladas;
    private Map<String, Atividade> atividadesRealizadas;
    private Map<String, PlanoTreino> planosTreino;
```

Figure 2.6: Atributos da classe Utilizador

2.3.2 Classes PraticanteOcasional, Amador e Profissional

Seguindo a mesma linha de pensamento das atividades, foram definidas subclasses para cada tipo de utilizador sendo que cada atribui um fator multiplicativo base diferente.

```
public class Amador extends Utilizador { Carlos
    public Amador(Utilizador utilizador) { Carlos
        super(utilizador);
        double fatorAmador = 0.4;
        this.setFatorMultiplicativoBase(fatorAmador);
    }
}
```

Figure 2.7: Construtor da classe Amador

2.4 Classe Recorde

Esta classe foi criada para representar um recorde relativo a um tipo de atividade, sendo que tem como atributos o utilizador detentor o recorde e a atividade executada pelo mesmo.

```
public class Recorde implements Serializable {
    private Utilizador user;
    private Atividade activity;
```

Figure 2.8: Atributos da classe Recorde

2.5 Classe Gestor

Esta classe agrega todas a informações, assim como, possui os métodos mais importantes, aqueles que permitem simular a execução das atividades e obter dados específicos sobre os mesmos. Sendo assim, os atributos da classe são os *Maps*, que armazenam os utilizadores (código único -> Utilizador), recordes (nomeAtividade -> Recorde), atividades (id -> Atividade) e planos de treino (id -> PlanoTreino).

```
public class Gestor implements Serializable {  🧑 Carlos
    private Map<Integer, Utilizador> utilizadores;
    private Map<String, Recorde> recordes;
    private Map<String, Atividade> atividades;
    private Map<String, PlanoTreino> planosTreino;
```

Figure 2.9: Atributos da classe Gestor

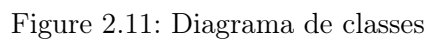
2.6 Classe TextUI

A classe TextUI tem a responsabilidade de implementar a componente visual através da utilização da classe NewMenu, permitindo ao utilizador comunicar os comandos pretendidos à classe Gestor.

```
public TextUI() {  🧑 Carlos
    this.gestor = new Gestor();
}
```

Figure 2.10: Contrutor da classe TextUI

(disponível na pasta do trabalho para melhor visualização)



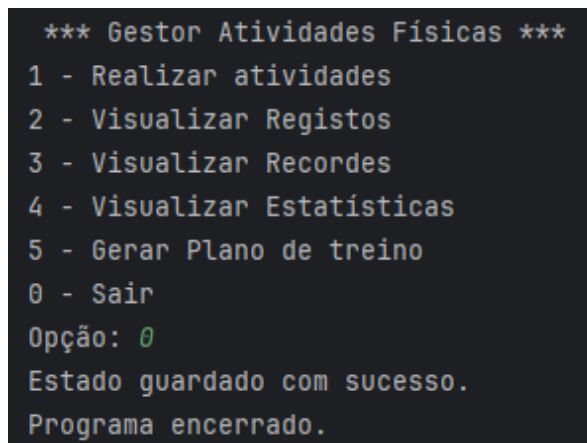
3 Descrição de funcionalidades

3.1 Criar estado inicial

Para criar o estado inicial do programa foi usado o programa *CriarEstadoInicial*, que cria atividades e planos de treino, que são adicionados aos utilizadores, que por sua vez são registados no Gestor. Este estado inicial é guardado em formato binário num ficheiro .dat.

3.2 Carregar estado e Guardar estado

No arranque do programa é carregado o estado guardado no ficheiro .dat disponível. Quando o programa é terminado, este guarda o estado atual num ficheiro do mesmo tipo.



```
*** Gestor Atividades Físicas ***
1 - Realizar atividades
2 - Visualizar Registos
3 - Visualizar Recordes
4 - Visualizar Estatísticas
5 - Gerar Plano de treino
0 - Sair
Opção: 0
Estado guardado com sucesso.
Programa encerrado.
```

Figure 3.1: Carregar e guardar estado

3.3 Menu Principal

Como vemos na figura 3.1, o menu principal apresenta as seguintes opções:

- Realizar atividades: Permite ao utilizador realizar saltos no tempo, simulando a realização das atividades compreendidas no intervalo. Sendo que é feito o cálculo das calorias das atividades e atualizado o registo das mesmas.
- Visualizar Registos: Permite obter as informações de todos os utilizadores.
- Visualizar Recordes: Permite obter os recordes de calorias consumidas de todos os tipos de atividades.
- Visualizar Estatísticas: Remete para o menu que contém todas as operações relativas a dados estatísticos.
- Gerar Plano de treino: Permite ao utilizador gerar um plano de treino conforme as suas necessidades.

3.4 Menu de estatísticas

```
*** Gestor Atividades Físicas ***
1 - Visualizar utilizador com mais atividades
2 - Visualizar utilizador com mais calorias gastas
3 - Visualizar a atividade mais realizada
4 - Visualizar Kms percorridos por utilizador
5 - Visualizar metros de altimetria totalizados por utilizador
6 - Visualizar plano de treino mais exigente para calorias propostas
7 - Visualizar lista de atividades de um utilizador
0 - Sair
```

Figure 3.2: Menu de Estatísticas

Como vemos na figura 3.2, o menu apresenta as seguintes opções:

- Visualizar utilizador com mais atividades: Permite ver qual o utilizador com mais atividades realizadas num período ou desde sempre.
- Visualizar utilizador com mais calorias gastas: Permite ver qual o utilizador com mais calorias consumidas num período ou desde sempre.
- Visualizar a atividade mais realizada: Permite ver qual o tipo de atividade mais realizada.
- Visualizar Kms percorridos por utilizador: Indicando o código do utilizador, permite ver os Kms percorrido por este num período ou desde sempre.
- Visualizar metros de altimetria totalizados por utilizador: Indicando o código do utilizador, permite ver os metros de altimetria percorridos por este desde sempre ou num período.
- Visualizar plano de treino mais exigente para calorias propostas: Indicando o número de calorias pretendidas, devolve o plano de treino mais exigente dentro desse limite. Sendo que o plano pode ser um que ainda não tenha sido executado.
- Visualizar lista de atividades de um utilizador: Permite, indicando o código de utilizador, ver as atividades do mesmo, sendo que as atividades isoladas já executadas, aparecem apenas nas atividades realizadas. Por outro lado, os planos de treino mantêm-se registados no conjunto do utilizador, mesmo que já executados.

3.5 Noção de atividade *Hard*

Como foi referido na secção 2.1.2, a atividade é definida como *Hard* no seu construtor, esta categorização afeta o cálculo da frequência média durante a atividade, sendo que para uma atividade normal tem como valor 140, enquanto para uma atividade *Hard* tem valor 190.

```
public double calcularFrequenciaAtividade(Utilizador utilizador) { // Carlos
    double frequenciaBase;
    if (Objects.equals(this.getDificuldade(), b: "Hard")) {
        frequenciaBase = 190;
    } else {
        frequenciaBase = 140;
    }

    return (utilizador.getFrequenciaCardiacaMedia() + frequenciaBase) / 2;
}
```

Figure 3.3: Método para cálculo de frequência durante a atividade

3.6 Gerar plano de treino

O programa permite gerar um plano de treino conforme o tipo de atividades requisitado pelo cliente (Distancia, DistanciaAltimetria, Repeticoes e RepeticoesPesos). Para além disso, o utilizador indica a recorrência semanal dos tipos de atividades, o número máximo de atividades por dia (sempre menor que 3) e o consumo de calorias mínimo pretendido. No entanto, não foi implementado o fator de não acrescentar atividades *Hard* caso nesse dia ou no dia anterior tivessem sido realizadas atividades desse tipo. Outra debilidade deste gerador, é o facto do número de iterações das atividades criadas ser sempre 1.

```
Digite tipo de plano que pretende (Distancia, DistanciaAltimetria, Repeticoes ou RepeticoesPesos:
Distancia
Digite a data de realização (YYYY-MM-DD):
2024-05-11
Digite o número máximo de atividades por dia (máximo 3):
4
Por favor, digite um número até 3.
Digite o número máximo de atividades por dia (máximo 3):
2
Digite a recorrência semanal das atividades:
2
Digite o consumo calórico mínimo:
800
Digite o código do utilizador:
0
Plano de treino criado com sucesso!
```

Figure 3.4: Gerar Plano

4 Conclusão

Com a realização deste trabalho foi possível aplicar os conceitos abordados na UC de Programação Orientada aos Objetos, como encapsulamento, herança, polimorfismo e outros. A maior dificuldade encontrada foi a funcionalidade de gerar um plano de treino conforme os objetivos do utilizador, sendo que esta foi a implementação menos bem conseguida neste projeto. Em suma, o tema do trabalho foi bastante interessante, uma vez que obrigou a criação de várias entidades e o estabelecimento de relações entre as mesmas, permitindo uma boa aprendizagem da linguagem Java e do paradigma de POO.