



**Universidade do Minho**  
Licenciatura em Engenharia Informática

## **Unidade Curricular de Programação Orientada aos Objetos**

Ano Letivo de 2023/2024

### **Fitness App**



**- Guilherme Araújo de Oliveira (a95021)**



**- Mateus Lemos Martins (a100645)**



**- Francisco José Magalhães da Rocha Coelho (a104521)**

# POO

## **Introdução**

Este relatório descreve a elaboração de uma aplicação de gestão de atividades físicas e planos de treino para os utilizadores. A aplicação deve permitir registar utilizadores, atividades e planos de treino, bem como gerar estatísticas sobre os mesmos. Deve ainda permitir simular mudanças de data. O trabalho foi realizado no âmbito da unidade curricular Programação Orientada aos Objetos, desta forma o projeto deve seguir os princípios de POO e incluir uma camada de interação com o utilizador por meio de menus de texto.

# 1. Arquitetura e estrutura de classes

## 1.1 Classe Atividade

```
public abstract class Atividade implements Serializable{  
  
    private static int nextId = 0;  
  
    private int id;  
    private String nome;  
    private LocalDateTime dataHora;  
    private Duration duracao;  
    private Utilizador user;  
    private double calorias;  
}
```

Figura 1 - Atributos da classe Atividade

Esta é uma classe abstrata que representa genericamente uma atividade e possui os seguintes atributos:

- **nextId**: atributo inteiro estático utilizado para gerar um id único;
- **id**: inteiro que serve de identificador único da atividade;
- **nome**: String que representa o nome da atividade;
- **dataHora**: LocalDateTime que representa a hora de realização da atividade;
- **duracao**: Duration que representa a duração da atividade;
- **user**: utilizador que realiza a atividade;
- **calorias**: quantidade de calorias gastas na execução da atividade.

Nesta classe temos todos os métodos habituais, os **construtores** (vazio, parametrizado e de cópia), os métodos **get** e **set** para os atributos acima referidos, os métodos **toString**, **equals** e **clone**, o método abstrato **calculaCalorias** que deve ser implementado nas classes que se encontrem abaixo de Atividade na hierarquia e são implementados ainda três métodos necessários ao funcionamento da aplicação.

## 1.2 Classe Atividade\_D

```
public class Atividade_D extends Atividade{

    private static Map<Integer, String> tipos;

    static {
        tipos = new HashMap<Integer, String>();
        tipos.put(key:1, value:"Remo");
        tipos.put(key:2, value:"Corrida na Pista de Atletismo");
        tipos.put(key:3, value:"Patinagem");
    }

    private double distanciaPercorrida;
```

Figura 2 - Atributos da classe Atividade\_D

A classe Atividade\_D estende a classe Atividade e representa as atividades do tipo distância. Esta classe herda os atributos da classe pai aos quais acrescentamos os seguintes:

- **tipos:** HashMap onde estão contidos os nomes das modalidades que estão incluídas neste tipo de atividades (distância), permitindo desta forma acrescentar facilmente novas modalidades através de um **put** como podemos observar na figura 2;
- **distanciaPercorrida:** dado que se trata de uma atividade do tipo distância temos um atributo double que permite precisamente guardar essa mesma distância percorrida pelo utilizador.

Esta classe contém ainda todos os métodos habituais já referidos acima e ainda a implementação do método abstrato da classe Atividade (**calculaCalorias**).

## 1.3 Classe Atividade\_DA

```
public class Atividade_DA extends Atividade{

    private static Map<Integer, String> tipos;

    static {
        tipos = new HashMap<Integer, String>();
        tipos.put(key:1, value:"Corrida na Estrada");
        tipos.put(key:2, value:"Trail no Monte");
        tipos.put(key:3, value:"Bicicleta de Estrada");
        tipos.put(key:4, value:"Bicicleta de Montanha");
    }

    private double distanciaPercorrida;
    private double altimetria;
```

Figura 3 - Atributos da classe Atividade\_DA

Tal como a classe Atividade\_D, a classe Atividade\_DA também estende a classe Atividade e representa as atividades do tipo distância e altimetria. Esta contém os seguintes atributos.

- **tipos:** tem exatamente a mesma função do atributo tipos de Atividade\_D;
- **distanciaPercorrida:** também representa o mesmo que o atributo com o mesmo nome de Atividade\_D;
- **altimetria:** tal como o nome sugere este é o atributo no qual é guardada a altimetria ganha pelo utilizador na execução desta atividade.

Tal como a classe anterior esta também contém todos os métodos habituais e ainda o método **calculaCalorias**.

## 1.4 Classe Atividade\_R

```
public class Atividade_R extends Atividade{

    private static Map<Integer, String> tipos;

    static {
        tipos = new HashMap<Integer, String>();
        tipos.put(key:1, value:"Abdominais");
        tipos.put(key:2, value:"Alongamentos");
        tipos.put(key:3, value:"Flexões");
    }

    private int numeroRepeticoes;
```

Figura 4 - Atributos da classe Atividade\_R

Esta classe também é uma extensão da classe Atividade e representa as atividades com repetições, contendo os seguintes atributos:

- **tipos:** mantém as mesmas funções das classes anteriores;
- **numeroRepeticoes:** atributo onde são guardados o número de repetições realizadas.

Inclui os métodos habituais anteriormente referidos e o método **calculaCalorias**.

## 1.5 Classe Atividade\_RCP

```
public class Atividade_RCP extends Atividade{

    private static Map<Integer, String> tipos;

    static {
        tipos = new HashMap<Integer, String>();
        tipos.put(key:1, value:"Levantamento de Pesos");
        tipos.put(key:2, value:"Extensão de Pernas");
    }

    private int numeroRepeticoes;
    private double pesoLevantado;
```

Figura 5 - Atributos da classe Atividade\_RCP

Temos ainda a classe Atividade\_RCP, a última que estende Atividade, que representa as atividades com repetições com pesos e inclui os seguintes atributos:

- **tipos:** mantém as mesmas funções das classes anteriores;
- **numeroRepeticoes:** tem a mesma função da classe anterior;
- **pesoLevantado:** representa o peso utilizado pelo utilizador por repetição.

## 1.6 Classe Utilizador

```
public class Utilizador implements Serializable{

    private String username;
    private String nome;
    private String morada;
    private String email;
    private double frequenciaCardiacaMedia;
    private TipoUtilizador tipoUtilizador;
    private Map<Integer, Atividade> atividades;
```

Figura 6 - Atributos da classe Utilizador

Esta é a classe que representa um utilizador da aplicação através da seguinte lista de atributos:

- **username**: este é o identificador único do utilizador que será único;
- **nome**: representa o nome do utilizador;
- **morada**: representa a morada do utilizador;
- **email**: atributo correspondente ao email do utilizador;
- **frequenciaCardiacaMedia**: inteiro que representa a frequência cardíaca média do utilizador;
- **tipoUtilizador**: aqui estará representado o tipo de utilizador que pode ser um dos 3 tipos enumerados na classe TipoUtilizador;
- **atividades**: HashMap onde são guardadas todas as atividades que correspondem ao utilizador em questão.

Para além dos métodos habituais esta classe ainda implementa uma série de métodos necessários à execução do programa como por exemplo o método **fatorMultiplicativo** que devolve o fator a ser utilizado para o cálculo das calorias de uma atividade para o utilizador em questão.

## 1.7 Classe TipoUtilizador

```
public enum TipoUtilizador {  
    PROFISSIONAL,  
    AMADOR,  
    OCASIONAL  
}
```

Figura 7 - Classe TipoUtilizador

A classe TipoUtilizador consiste apenas na enumeração dos 3 tipos possíveis de utilizador, sendo eles:

- **Profissional**
- **Amador**
- **Ocasional**

Estes tipos terão influência no método como são calculadas as calorias gastas em cada atividade.

## 1.8 Classe PlanoDeTreino

```
public class PlanoDeTreino implements Serializable{  
  
    private static int nextId = 0;  
  
    private Integer id;  
    private String nome;  
    private LocalDate dataInicio;  
    private LocalDate dataFim;  
    private Map<Integer, Atividade> atividades;  
    private double objetivoCalorias;  
    private double objetivoDistancia;  
    private int frequenciaSemanal;  
    private Utilizador user;
```

Figura 8 - Atributos da classe PlanoDeTreino

Esta classe representa um plano de treino que é essencialmente um conjunto de atividades contidas num período que correspondem ao mesmo utilizador, podendo ter várias repetições da mesma, por exemplo, podemos escolher 3 séries de abdominais, que mantêm as mesmas informações de calorias, repetições, etc., mas altera a hora consoante a duração, por exemplo, se a primeira série tiver início às 19:00H e a atividade tiver a duração de 5M a segunda série terá início às 19:05H e assim sucessivamente. A classe tem os seguintes atributos:

- **nextId**: tem as mesmas funções do atributo com o mesmo nome da classe Atividade;
- **id**: identificador único do plano;
- **nome**: nome do plano;



- **dataInicio**: data de início do plano de treino;
- **dataFim**: data de finalização do plano de treino;
- **atividades**: HashMap onde estão contidas as atividades presentes no plano em questão sendo as chaves o id de cada atividade;
- **objetivoCalorias**: atributo que representa o objetivo de calorias a perder com a execução do plano;
- **objetivoDistancia**: atributo que representa o objetivo de distância a percorrer durante a execução do plano;
- **frequenciaSemanal**: número de vezes que o plano deve ser executado numa semana;
- **user**: Utilizador que realiza o plano de treino.

A classe inclui os métodos habituais e ainda dois métodos, um para adicionar atividades ao plano e outro para remover atividades do plano.

## 1.9 Classe TerminalManipulation

```
public class TerminalManipulation implements Serializable {
    private transient Scanner scanner;
    private transient DateTimeFormatter formatter = DateTimeFormatter.ofPattern(pattern:"dd-MM-yyyy HH:mm", Locale.ENGLISH);
    private transient DateTimeFormatter formatter2 = DateTimeFormatter.ofPattern(pattern:"dd-MM-yyyy", Locale.ENGLISH);
}
```

Figura 9 - Classe TerminalManipulation

Esta classe tem como objetivo auxiliar o funcionamento da aplicação. Para isto contém os seguintes atributos:

- **scanner**: atributo do tipo Scanner utilizado para ler inputs do utilizador;
- **formatter**: atributo do tipo DateTimeFormatter utilizado para formatar os tipos de data e hora;
- **formatter2**: tem a mesma função do anterior, mas apenas para datas sem hora.

Aqui estão contidos uma série de métodos que são utilizados para a construção de menus, controlo de inputs do utilizador, etc.

## 1.10 Classe Stats

```
public class Stats implements Serializable{  
  
    public Stats(){  
    }  
}
```

Figura 10 - Classe Stats

A classe Stats trata-se de uma classe que não contém atributos, mas que contribui para o funcionamento de uma parte da aplicação, a parte das estatísticas. Nesta classe estão contidos vários métodos cujo objetivo é fazer o cálculo de certas e determinadas estatísticas.

## 1.11 Classe Fitness\_APP

```
public class Fitness_APP implements Serializable{  
  
    private Map<String, Utilizador> utilizadores;  
    private Map<Integer, Atividade> atividades;  
    private Map<Integer, PlanoDeTreino> planos;  
    private ChronoLocalDateTime<?> data;  
    private transient Scanner scanner;  
}
```

Figura 11 - Atributos da classe Fitness\_APP

Esta classe trata-se do cérebro da nossa aplicação, é nela que se encontra o armazenamento dos utilizadores, atividades, planos de treino, data, etc., para isto a classe contém os seguintes atributos:

- **utilizadores:** HashMap que armazena todos os utilizadores e cuja key é o respetivo username;
- **atividades:** HashMap que armazena todas as atividades e cuja key é o respetivo id;
- **planos:** HashMap que armazena todos os planos de treino e cuja key é o respetivo id;
- **data:** atributo onde é armazenada a data do programa que por definição é a data atual, mas que pode ser alterada pelo utilizador;
- **scanner:** é utilizado para ler os inputs do utilizador.

Aqui estão contidos diversos métodos, utilizados na execução do programa, como, por exemplo, métodos para efetuar o registo dos objetos (utilizadores, atividades e planos), para os remover, para os listar, etc.

Temos ainda os métodos relacionados com as estatísticas que utilizam os métodos presentes na classe Stats.

## 1.12 Classe Fitness\_APP\_Controller

```
public class Fitness_APP_Controller implements Serializable{  
    private Fitness_APP app;  
    private TerminalManipulation tm;  
    private transient Scanner scanner;  
}
```

Figura 12 - Classe Fitness\_APP\_Controller

Esta classe tem como objetivo tratar dos menus da aplicação, utilizando para isso:

- **app**: objeto da classe Fitness\_APP onde serão guardados os principais dados da aplicação;
- **tm**: instância de TerminalManipulation para melhor manipulação do terminal;
- **scanner**: utilizado para ler os inputs do utilizador.

A classe contém os métodos de criação dos menus que por sua vez utilizam os métodos da **app** de registo, remoção, etc.

## 1.13 Classe Main

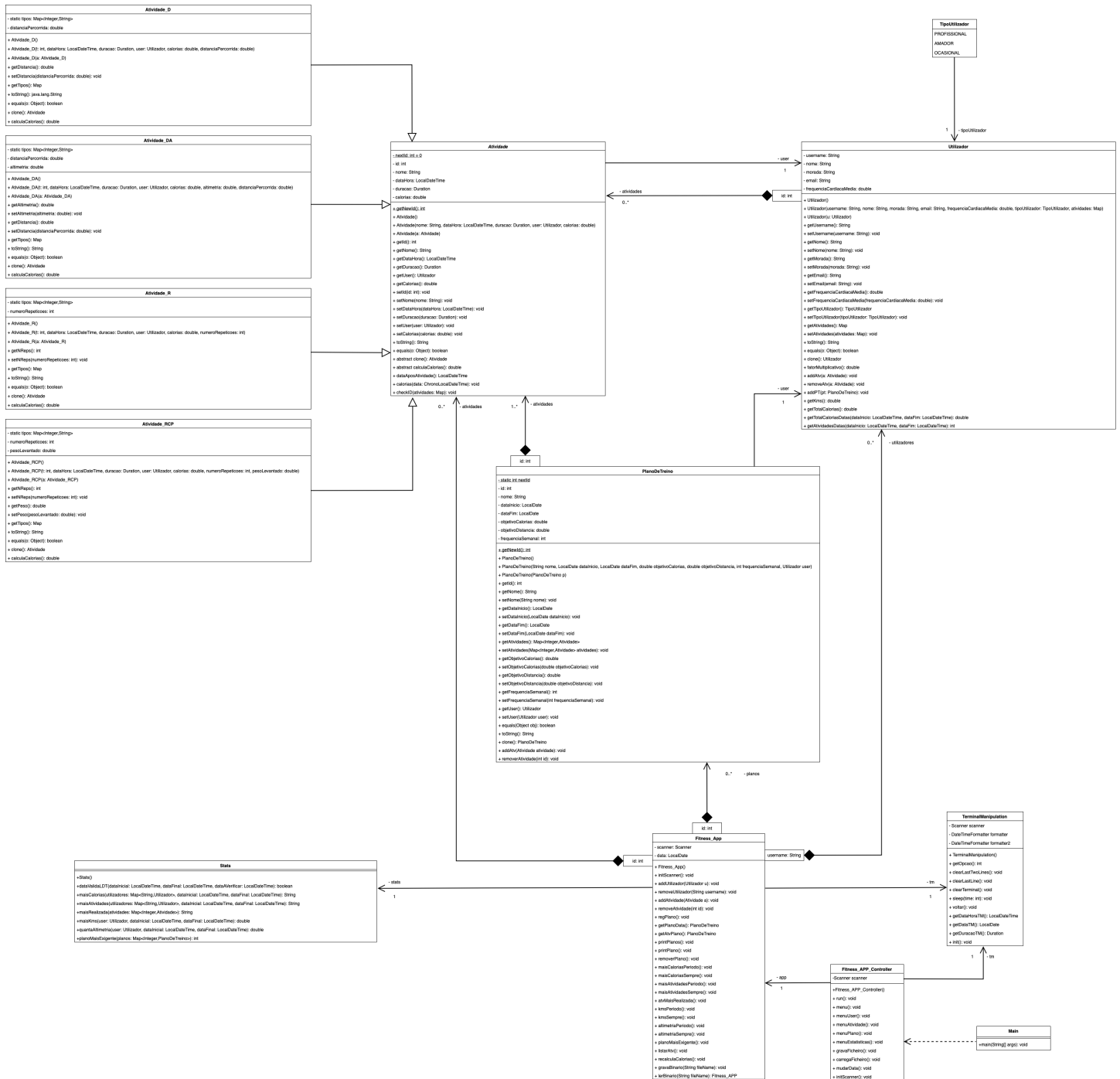
```
public class Main {  
    Run | Debug  
    public static void main(String[] args) throws FileNotFoundException, IOException, ClassNotFoundException {  
        Fitness_APP_Controller fitness = new Fitness_APP_Controller();  
        fitness.run();  
    }  
}
```

Figura 13 - Classe Main

A classe Main consiste apenas na criação de uma instância de Fitness\_APP\_Controller e na execução do método run() da mesma, que dá início à execução da aplicação.

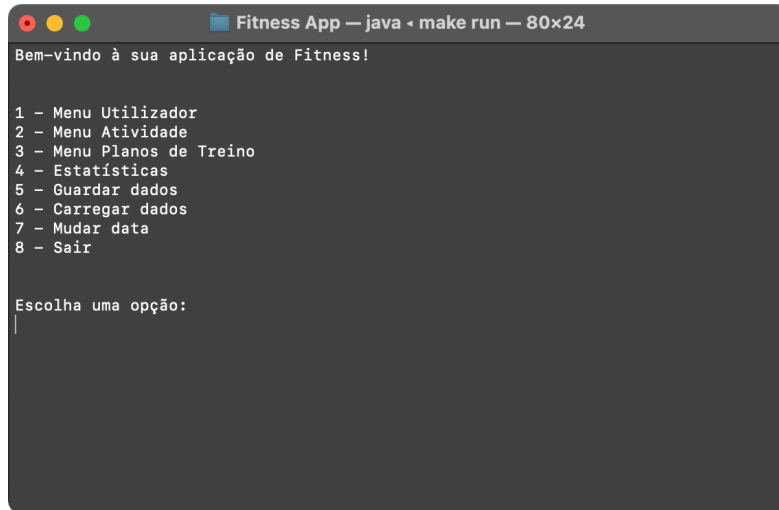
## 2. Diagrama de Classes

De seguida podemos observar o nosso diagrama de classes onde estão representadas todas as classes implementadas. Em caso de dificuldade de visualização a imagem encontra-se também disponível na pasta do trabalho.



## 3. Descrição da aplicação

### 3.1 Menu Inicial

A screenshot of a terminal window titled "Fitness App — java ◀ make run — 80x24". The window displays a welcome message "Bem-vindo à sua aplicação de Fitness!" followed by a numbered list of eight options: 1 - Menu Utilizador, 2 - Menu Atividade, 3 - Menu Planos de Treino, 4 - Estatísticas, 5 - Guardar dados, 6 - Carregar dados, 7 - Mudar data, and 8 - Sair. Below the list, it prompts "Escolha uma opção:" with a cursor on the next line.

```
Fitness App — java ◀ make run — 80x24
Bem-vindo à sua aplicação de Fitness!

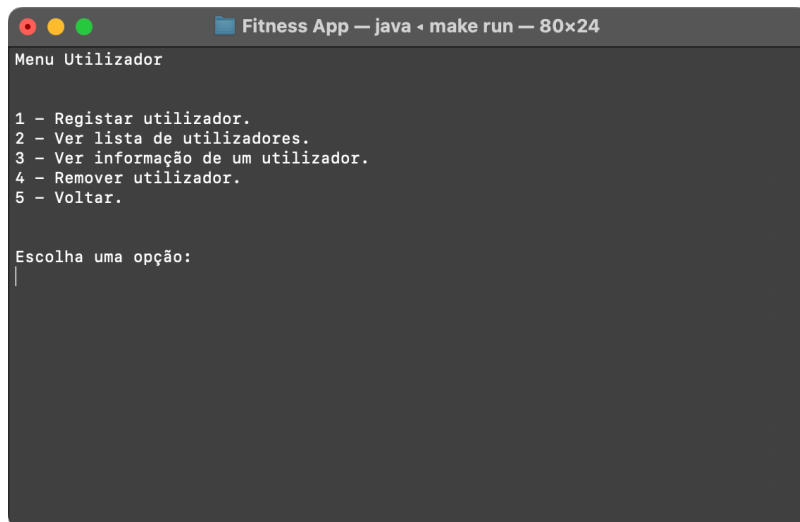
1 - Menu Utilizador
2 - Menu Atividade
3 - Menu Planos de Treino
4 - Estatísticas
5 - Guardar dados
6 - Carregar dados
7 - Mudar data
8 - Sair

Escolha uma opção:
|
```

Figura 14 - Menu Inicial

Quando é executada a aplicação é iniciado o menu inicial que consiste na enumeração dos restantes menus aos quais o utilizador pode aceder.

### 3.2 Menu Utilizador

A screenshot of a terminal window titled "Fitness App — java ◀ make run — 80x24". The window displays the title "Menu Utilizador" followed by a numbered list of five options: 1 - Registrar utilizador., 2 - Ver lista de utilizadores., 3 - Ver informação de um utilizador., 4 - Remover utilizador., and 5 - Voltar.. Below the list, it prompts "Escolha uma opção:" with a cursor on the next line.

```
Fitness App — java ◀ make run — 80x24
Menu Utilizador

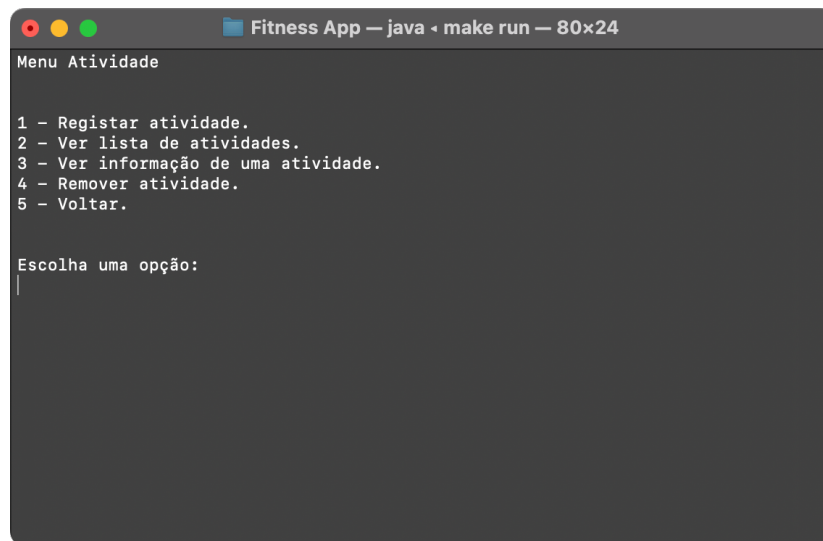
1 - Registrar utilizador.
2 - Ver lista de utilizadores.
3 - Ver informação de um utilizador.
4 - Remover utilizador.
5 - Voltar.

Escolha uma opção:
|
```

Figura 15 - Menu Utilizador

Este é o menu onde estão disponíveis as funcionalidades relacionadas diretamente com o utilizador.

### 3.3 Menu Atividades



```
Fitness App — java ◀ make run — 80x24
Menu Atividade

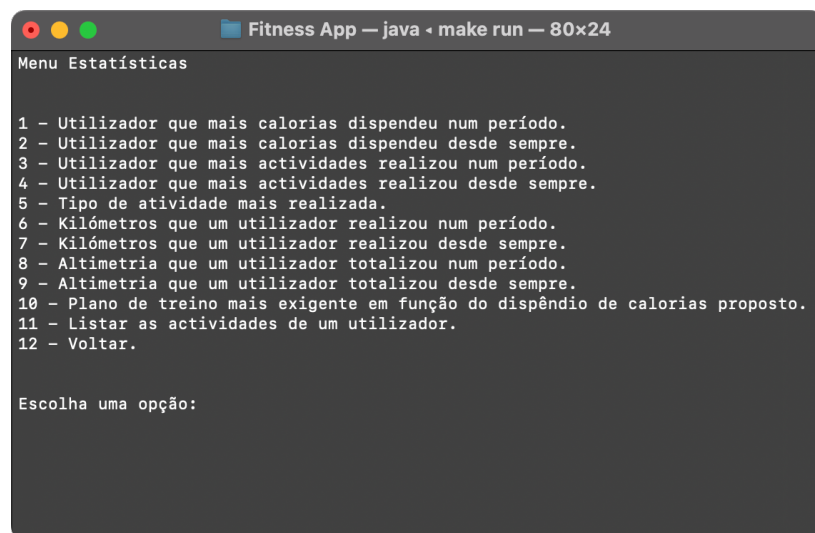
1 - Registrar atividade.
2 - Ver lista de atividades.
3 - Ver informação de uma atividade.
4 - Remover atividade.
5 - Voltar.

Escolha uma opção:
|
```

Figura 16 - Menu Atividades

Este é o menu onde estão disponíveis as funcionalidades relacionadas diretamente com as atividades.

### 3.4 Menu Estatísticas



```
Fitness App — java ◀ make run — 80x24
Menu Estatísticas

1 - Utilizador que mais calorias dispendeu num período.
2 - Utilizador que mais calorias dispendeu desde sempre.
3 - Utilizador que mais actividades realizou num período.
4 - Utilizador que mais actividades realizou desde sempre.
5 - Tipo de atividade mais realizada.
6 - Kilómetros que um utilizador realizou num período.
7 - Kilómetros que um utilizador realizou desde sempre.
8 - Altimetria que um utilizador totalizou num período.
9 - Altimetria que um utilizador totalizou desde sempre.
10 - Plano de treino mais exigente em função do dispêndio de calorias proposto.
11 - Listar as actividades de um utilizador.
12 - Voltar.

Escolha uma opção:
```

Figura 17 - Menu Estatísticas

Menu onde são enumeradas todas as estatísticas que estão disponíveis para visualização na aplicação.

### 3.5 Menu Guardar Dados

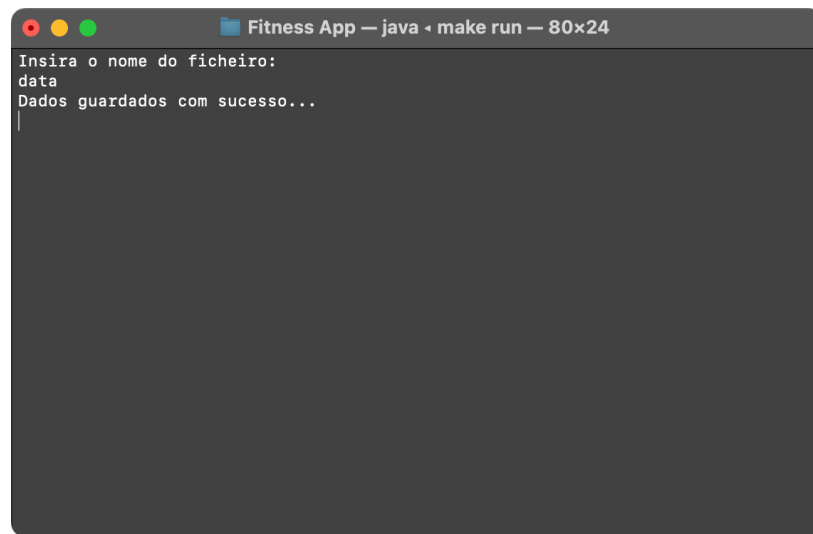


Figura 18 – Execução do Menu Guardar Dados

Este é o menu onde é possível efetuar a gravação do estado atual da aplicação, ou seja, todos os utilizadores, atividades e planos de treino existentes. A gravação é feita para um ficheiro binário que pode ser posteriormente carregado para a aplicação.

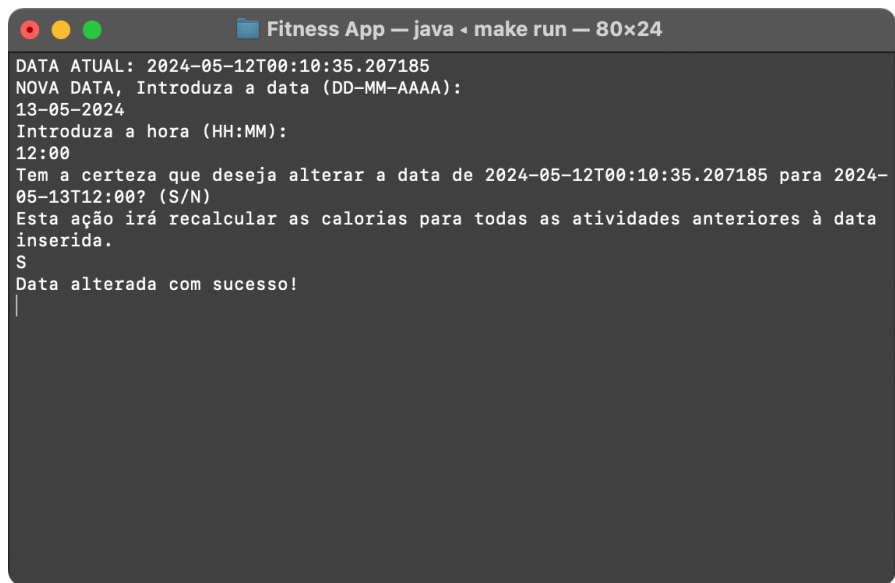
### 3.6 Menu Carregar Dados



Figura 19 – Execução do Menu Carregar Dados

Este menu é um pouco o inverso do menu interior, neste é possível efetuar a leitura de um ficheiro binário para carregar um estado para a aplicação em execução, ou seja, carregar todos os utilizadores, atividades e planos de treino gravados no ficheiro escolhido.

### 3.7 Menu Mudar Data



```
DATA ATUAL: 2024-05-12T00:10:35.207185
NOVA DATA, Introduza a data (DD-MM-AAAA):
13-05-2024
Introduza a hora (HH:MM):
12:00
Tem a certeza que deseja alterar a data de 2024-05-12T00:10:35.207185 para 2024-
05-13T12:00? (S/N)
Esta ação irá recalcular as calorias para todas as atividades anteriores à data
inserida.
S
Data alterada com sucesso!
```

Figura 20 - Execução do Menu Mudar Data

Por fim temos o menu para a mudança de data. O programa permite que seja alterada a data e hora do mesmo para poder ser feita uma simulação de cálculo de calorias e, por sua vez, das estatísticas.



## 4. Conclusão

O nosso trabalho apresenta uma aplicação robusta de gestão de atividades físicas e planos de treino, desenvolvida com base em princípios de Programação Orientada a Objetos (POO). Com uma arquitetura bem estruturada e uma interface intuitiva, a aplicação oferece aos utilizadores uma maneira eficaz de registar atividades, acompanhar o seu progresso e gerar estatísticas relevantes. Temos noção que ficaram alguns objetivos por cumprir como a introdução da noção de atividades “hard” e a geração automática de planos de treino, que seriam alguns aspetos a serem implementados numa fase seguinte do desenvolvimento do programa. Mas apesar disto pensamos que resultou num bom produto final, que teria espaço para evolução numa fase seguinte. Em suma, achamos que apesar de não terem sido implementados todos os objetivos foi concluída a parte fulcral do projeto.