Universidade do Minho
Escola de Engenharia

Programação Orientada a objetos

Licenciatura em Engenharia Informática

Ano Letivo de 2023/2024

João Ricardo Ribeiro Rodrigues (a100598)

POO

Introdução

Este projeto surge no âmbito da unidade curricular de Programação Orientada a Objetos da Licenciatura em Engenharia Informática da Universidade do Minho. O trabalho consistiu na implementação de uma aplicação de gestão de treino. Esta aplicação, além de permitir o registo e acompanhamento das atividades físicas realizadas pelos utilizadores, também é capaz de gerar planos de treino personalizados com base nos objetivos individuais de cada utilizador.

A aplicação permite aos utilizadores registar as suas atividades físicas e visualizar o progresso do seu treino.

O projeto foi desenvolvido utilizando os princípios da Programação Orientada a Objetos, o que permitiu uma estrutura de modularizada e encapsulada. O que será particularmente útil na medida em que facilita a evolução da aplicação, uma vez que o acrescento de novas funcionalidades é bastante mais simples.

Ao longo deste documento será feita uma breve explicação da arquitetura adotada, bem como o funcionamento do programa.

As atividades

As atividades são implementadas através da classe Atividade e de todas as suas classes filhas. A classe Atividade possui a estrutura base de uma atividade, com as informações gerais a todas as atividades, como apresentado na figura abaixo:

```
public abstract class Atividade implements Serializable {
    9 usages
    protected String nome;
    13 usages
    protected double tempoDispendido;
    17 usages
    protected double frequenciaCardiacaMedia;
    13 usages
    protected boolean isHard;
    16 usages
    protected double consumoCalorias;
```

Figura 1 - Variáveis da classe Atividade

A classe implementa os getters e setters para as variáveis e a declaração abstrata da função *setConsumoCalorias*, cuja definição se encontra nas classes ilhas.

```
public abstract void setConsumoCalorias(Utilizador utilizador);
```

Figura 2 - Função setConsumoCalorias

De forma a definir os diferentes tipos de atividades que existirão no programa, criamos classes filhas da classe Atividade, cada classe filha representando uma atividade. As classes filhas têm os seus atributos próprios e os atributos herdados da classe pai, assim como os seus getters e setters.

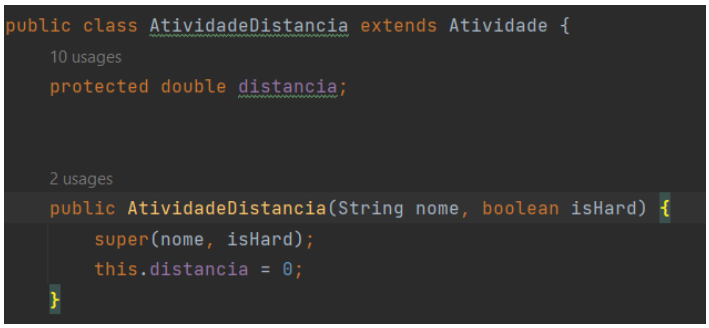


Figura 3 - Exemplo da classe filha AtividadeDistancia

Adicionalmente, cada uma possui uma definição da função *setConsumoCalorias* diferetnte, especifica para as caracteristicas de cada atividade e assim facilmente customizavel para ela.

Figura 4 - Exemplo de uma definição da funçã setConsumoCalorias

Em casos de uma atividade partilhar características de outra, como é o caso da classe AtividadeDistanciaAltimetria, a outra classe pode ser aproveitada usando herança, como podemos ver na imagem abaixo:

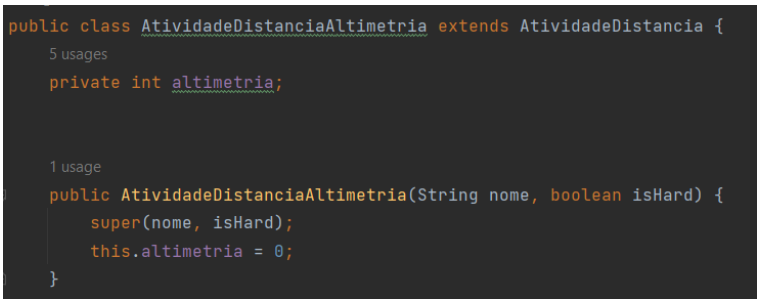


Figura 5 - Classe que herda uma filha da classe Atividades

Desta forma não só o processo de adicionar uma atividade suportada no código como a sua customização se tornam mais simples, podendo reutilizar conteudo de outras classes.

Os utilizadores

De forma a implementar os utilizadores foi criada uma classe Utilizadores. Nesta são guardadas todas as informações de um utilizador, assim como as atividades que realizou e os planos de treino que possui, como podemos visualizar na figura abaixo:

Figura 6 - Variáveis da classe Utilizador

A classe possui todas as funções de getters e setters para as variáveis da classe, assim como funções para adicionar atividades e planos de treino aos respetivos ArrayLists, nomeadamente as funções *addAtividades* e *addPlanosDeTreino*.

O plano de treino

Um plano de treino é constituido por uma lista de atividades a realizar e uma data para o efetuar. A classe PlanoDeTreino é a nossa estrutura para criar planos de treino, que posteriormente serão adicionados à lista de planos de treino de um utilizador.

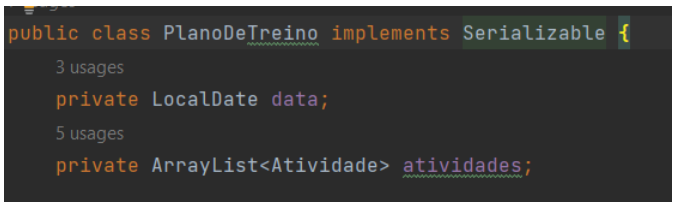


Figura 7 - Variáveis da classe PlanoDeTreino

As atividades a realizar são guardadas num ArrayList e para guardar a data utilizamos uma variável do tipo LocalDate, já preparada para lidar com datas.

Store

De forma a ter um local onde as informações resultantes da execução do programa são guardadas foi criada uma classe Store, que será responsável por guardar as informações, carregar as informações a partir de um arquivo aquando do inicio da execução do programa e guardar as informações no arquivo no fim da execução.

As estruturas utilizadas foram as seguintes:

```java
public class Store implements Serializable {
    7 usages
    private Map <Integer, Utilizador> storeUtilizadores;
    6 usages
    private Map <String, Atividade> storeAtividades;
    3 usages
    private int user;
```

Figura 8 - Variáveis da classe Store

A variável *storeUtilizadores* é onde são guardados todos os utilizadores criados, usando o código do utilizador como chave do mapa para ser fácil de aceder ao utilizador pretendido.

A variável *storeAtividades* é onde são guardados todas as atividades registadas no programa, usando o nome da atividade como chave do mapa para tornar o acesso fácil.

A variável *user* contém o código do utilizador que está loggado no momento. O conteudo desta é atualizado sempre que um login bem sucedido for realizado (exceto quando é feito login com o administrador, cujo código é 0).

Para além das habituais funções da classe, nomeadamente construtores, getters e setters, esta classe possui duas funções muito importantes para o funcionamento do programa, as funções *salvarStore* e *carregarStore*. Estas são responsáveis por salvar o estado do programa em um arquivo aquando do termino da sua execução e por carregar o estado do programa presente no arquivo, respetivamente.

```java
public static void salvarStore(Store store, String filename) {
    try (ObjectOutputStream outputStream = new ObjectOutputStream(new FileOutputStream(filename))) {
        outputStream.writeObject(store);
        System.out.println("Dados salvos com sucesso.");
    } catch (IOException e) {
        e.printStackTrace();
    }
}

1 usage
public static Store carregarStore(String filename) {
    Store store = null;
    try (ObjectInputStream inputStream = new ObjectInputStream(new FileInputStream(filename))) {
        store = (Store) inputStream.readObject();
        System.out.println("Dados carregados com sucesso.");
    } catch (IOException | ClassNotFoundException e) {
        e.printStackTrace();
    }
    return store;
}
```

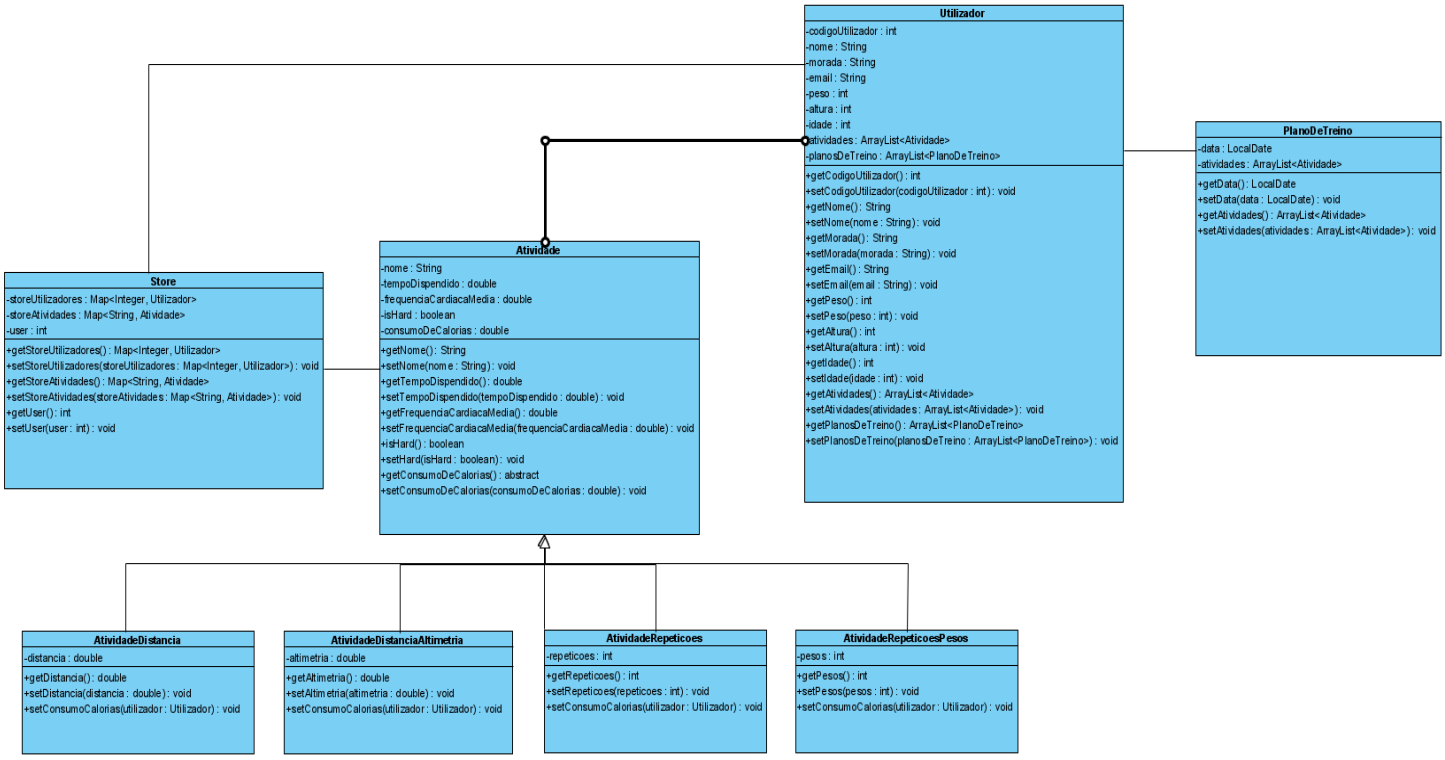
Figura 9 - Funções salvarStore e carregarStore

Menus

As restantes classes do projeto são classes cuja função é exibir um menu e tratar a opção inserida, tais como:

- **Login:** efetua o login no programa com um utilizador ou com o administrador
- **MenuAdministrador:** apresenta o menu do administrador, onde podem ser adicionados utilizadores e atividades ao programa
- **MenuPrincipal:** apresenta o menu do utilizador, onde podem ser registadas as atividades realizadas pelo user e adicionar planos de treino ao utilizador

Diagrama de classe



Descrição da aplicação desenvolvida

Passando a analisar o resultado final, irá ser apresentado o fluxo de execução do programa nas próximas figuras.

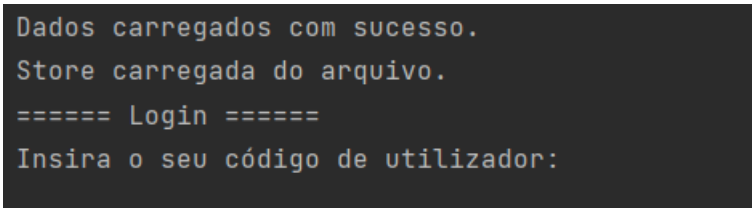


Figura 10 - Login no programa

Quando o programa é iniciado, é apresentado o login, onde é pedido que insira o seu código de utilizador e a informação de se foram carregados dados do arquivo.

Caso o codigo inserido seja 0, o programa assume que é o administrador que está a fazer login e direciona-o para o menu de administrador.

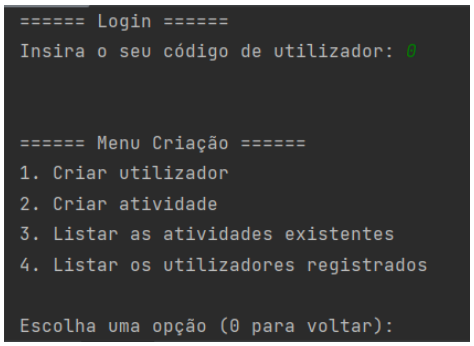


Figura 11 - Menu de Administrador

Caso seja diferente de 0, e exista um utilizador com esse código, ele é direcionado para o menu de utilizador e a variavel *user* na store é atualizada para o codigo inserido.

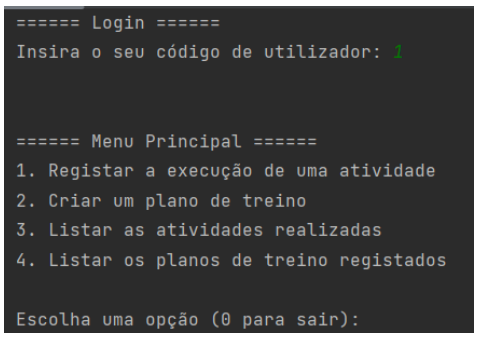


Figura 12 - Menu de Utilizador

No caso de termos acedido ao menu de administrador, e termos inserido a opção 1, é nos pedido que introduzamos as informações do utilizador a criar.

```
Escolha uma opção (0 para voltar): 1
Insira o nome do utilizador: joao
Insira a morada do utilizador: Rua das Flores
Insira o email do utilizador: joao@gmail.com
Insira a peso do utilizador: 80
Insira a altura do utilizador (cm): 185
Insira a idade do utilizador: 21
```

Figura 13 - Menu Administrador opção 1

Caso a opção inserida seja a opção 2 será nos pedido que introduzamos as informações da atividade a criar.

```
Escolha uma opção (0 para voltar): 2
Insira o nome da atividade: corrida
O nivel de dificuldade é Hard (true ou false): false
Insira o tipo da atividade: distancia
```

Figura 14 - Menu Administrador opção 2

Caso a opção inserida seja a 3 ou 4, serão listados todos as atividades ou utilizadores que existem na store, respetivamente.

No caso de termos acedido ao menu de utilizador, e termos inserido a opção 1, é nos pedido que introduzamos as informações da atividade a registar a execução.

```
Escolha uma opção (0 para sair): 1
Insira o nome da atividade: corrida
Indique a duração (em minutos): 20
Indique a frequência cardiaca média: 80
Insira a distância percorrida: 10
A execução da atividade foi registada com sucesso.
```

Figura 15 - Menu Utilizador opção 1

Caso a opção inserida seja a opção 2 será nos pedido que introduzamos as informações do plano de treino a criar.

```
Escolha uma opção (0 para sair): 2
Indique a data para a realização: 2024-05-15

Indique as atividades a realizar (0 para terminar): corrida

Indique a duração (em minutos): 20
Indique a frequência cardiaca média: 100
Insira a distância percorrida: 15

Indique as atividades a realizar (0 para terminar): 0
```

Figura 16 - Menu Utilizador opção 2

Caso a opção inserida seja a 3 ou 4, serão listados todos as atividades registadas ou todos os planos de treino criados que existem no utilizador, respetivamente.

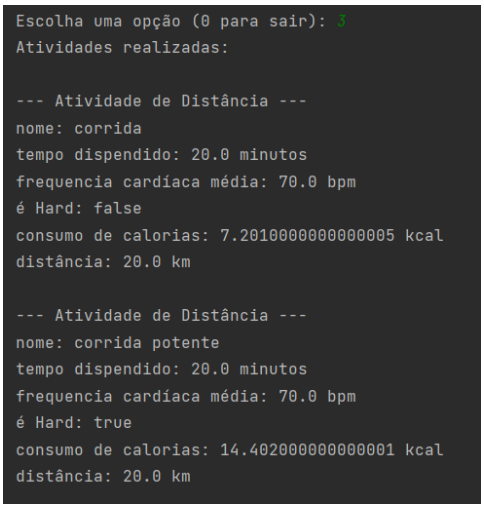


Figura 17 - Menu Utilizador opção 3

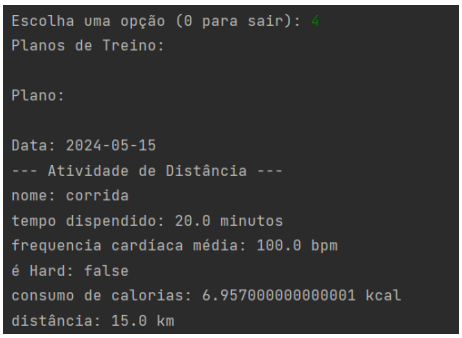
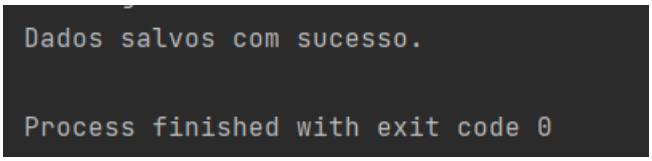


Figura 18 - Menu Utilizador opção 4

No fim da execução do programa é guardado no arquivo o estado atual da store, como podemos ver na figura abaixo:



Conclusão

O trabalho realizado inclui todas as funcionalidades fundamentais para a gestão das entidades para além de listar as atividades e planos de treino de um utilizador.

A noção de *Hard* foi também implementada, sendo ela uma variável da classe Atividade que depois influencia a forma como o cálculo do consumo de calorias é realizado.

De um modo geral, os princípios da programação orientada a objetos foram respeitados, com o intuito de tornar o código modular e de fácil manutenção. A adição de novos tipos

de atividades é simples e direta, podendo fazer uso de propriedades de outras atividades, tal como a sua definição e calculo do consumo de calorias.

Com uma gestão de tempo mais eficiente, gostaria de ter implementado todas as estatísticas pedidas, a passagem do tempo e a possibilidade de gerar um plano de treino de acordo com os objetivos do utilizador. No entanto, devido ao abandono do projeto por parte das minhas colegas, que nunca mostraram interesse em colaborar neste projeto, mesmo após várias tentativas de contacto, vi-me obrigado a concentrar o meu tempo nas funcionalidades ao meu alcance e em dar o meu melhor nas mesmas.

Espero com este trabalho, apesar de não ter todas as funcionalidades implementadas, ter demonstrado o meu conhecimento da disciplina obtido das aulas.

Devido à minha situação de grupo e à elevada saturação do calendário escolar, tive de fazer algumas submissões após a hora limite que espero que possam ser contabilizadas, visto terem sido alterações menores mas uteis para o projeto e para a apresentção (nomeadamente alguns erros ortográficos).