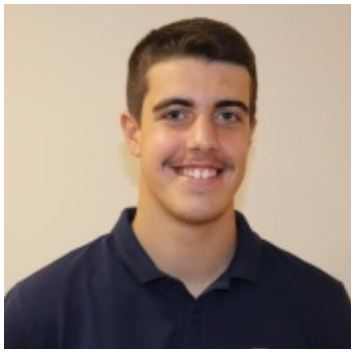
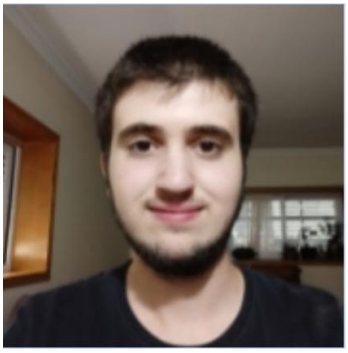
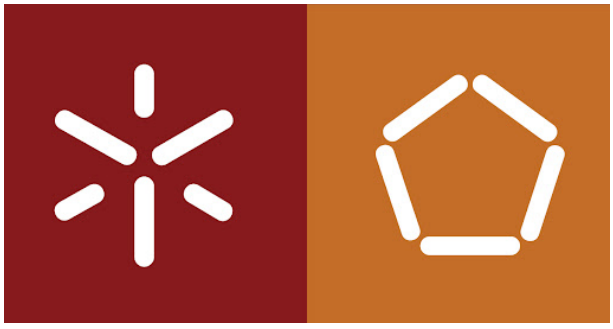
Relatório Programação Orientada aos Objetos

A100615 - João Miguel Mendes Moura
A100661 - Daniel Henrique Cracel Rodrigues
A100691 - Francisco Manuel Afonso
Maio 2024







Conteúdo

Introdução

O presente relatório tem como objetivo descrever o projeto realizado no contexto da Unidade Curricular de Programação Orientada aos Objetos que consiste no desenvolvimento de uma aplicação que faça a gestão de atividades e planos de treino havendo diversos tipos de atividades como por exemplo *trail*, levantamento de peso, remo e flexões. A aplicação também deve ter a noção de utilizadores, havendo uma forma de *login* e de registo.

Classes

No nosso projeto decidimos dividir em 13 classes de entre as quais a *Activity*, *ActivityInterface*, *Trail*, *Rowing*, *WeigthLifting*, *Pushups*, *Plan*, *PlanInterface*, *NewMenu*, *TextUI*, *User*, *UserInterface* e *APP*.

2.1 *User*

A classe *User* armazena informações necessárias para a caracterização da mesma.

- `private int` code;
- `private String` name;
- `private String` password;
- `private String` residence;
- `private String` email;
- `private double` ahr;
- `private TypeOfUser` typeofuser;

Estas são as informações que nós usamos para definir um User. No caso de TypeOfUser temos três tipos: *AMATEUR*, *PROFESSIONAL* e *OCCASIONAL*. No caso do *average heart rate* começa a zero em todos os utilizadores acabados de registar e conforme as atividades que vão realizando, o *average heart rate* vai aumentando.

2.2 *UserInterface*

A classe *UserInterface* armazena informações necessárias para a caracterização da mesma.

- `private List<User>` users;

Nesta classe é onde fazemos o *load*, o *save* e todas as funções que pedem informações ao utilizador para este introduzir no terminal.

2.3 *Activity*

A classe *Activity* armazena informações necessárias para a caracterização da mesma.

- `private String` name;
- `private double` time;
- `private Difficulty` difficulty;
- `private User` user;

Nesta classe definimos as nossas atividades com as suas características gerais a todas as atividades. Usamos também a classe *User* para definir quem é o utilizador que regista a atividade no sistema. O tipo *Difficulty* configura dois tipos de dificuldade: *NORMAL* e *HARD*.

2.4 *ActivityInterface*

A classe *ActivityInterface* armazena informações necessárias para a caracterização da mesma.

- `private List<Activity>` activities;

Nesta classe é onde fazemos o *load*, o *save* e todas as funções que pedem informações ao utilizador para este introduzir no terminal.

2.5 *Pushups*

A classe *Pushups* armazena informações necessárias para a caracterização da mesma.

- `private int` reps;

Nesta classe nós damos *extend* á classe *Activity*. Para definir a classe usamos *reps* para identificar o número de repetições de flexões.

2.6 *Rowing*

A classe *Rowing* armazena informações necessárias para a caracterização da mesma.

- `private double` distance;

Nesta classe nós damos *extend* á classe *Activity*. Para definir a classe usamos *distance* para identificar a distância percorrida a fazer remo.

2.7 *Trail*

A classe *Trail* armazena informações necessárias para a caracterização da mesma.

- `private double` distance;
- `private double` altimeter;

Nesta classe nós damos *extend* á classe *Activity*. Para definir a classe usamos *distance* e *altimeter* para identificar a distância percorrida a fazer *Trail* e para identificar o ganho de elevação.

2.8 *Weightlifting*

A classe *Weightlifting* armazena informações necessárias para a caracterização da mesma.

- `private int` reps;
- `private int` weight;

Nesta classe nós damos *extend* á classe *Activity*. Para definir a classe usamos *reps* para identificar o numero de repetições e usamos *weight* para identificar o peso.

2.9 *Plan*

A classe *Plan* armazena informações necessárias para a caracterização da mesma.

- `private List<Activity>` activities;
- `private LocalDateTime` date;
- `private int` timesPerWeek;

Estas são as características da nossa classe que trata dos planos de . Usamos uma lista de atividades para caso seja um plano de treino com mais do que uma atividade. Usamos também a data em que foi registado o plano de treino e temos também o número de vezes que o utilizador faz esse plano de treino por semana.

2.10 *PlanInterface*

A classe *PlanInterface* armazena informações necessárias para a caracterização da mesma.

- `private List<Plan>` plans;

Nesta classe é onde fazemos o *load*, o *save* e todas as funções que pedem informações ao utilizador para este introduzir no terminal.

2.11 *TextUI*

A classe *TextUI* armazena informações necessárias para a caracterização da mesma.

- `private UserInterface` userInterface;
- `private ActivityInterface` activityInterface;
- `private PlanInterface` planInterface;
- `private String` usersFile;
- `private String` activitiesFile;
- `private String` planFile;

Nesta classe é onde tratamos dos textos e dos menus que vão aparecer no terminal ao utilizador. Nesta *class* usamos as funções *load* para guardar a informação contida em cada ficheiro.

2.12 *NewMenu*

A classe *NewMenu* é a classe que trata dos menus e da maneira como estes são apresentados no terminal. Nesta classe usamos como base a classe fornecida pelos professores.

Funcionalidades

Quanto ao funcionamento do programa é nos pretendido criar uma simulação de compra e venda de artigos. Quanto ás suas funcionalidades o projeto tem:

3.1 *Load*

Esta funcionalidade trata de carregar toda a informação contida nos nossos ficheiros *activities.dat*, *plan.dat* e *users.dat* para uma lista.

3.2 Save

Com a funcionalidade *save*, como o nome indica, damos *save* de todas as alterações feitas na lista e guardamos a lista de volta ao ficheiro.

3.3 Registar Utilizador

Nesta funcionalidade nós registamos um utilizador, pedindo ao utilizador que nos indique o seu nome, a sua palavra-passe, a sua morada, *email* e que nos diga qual é o tipo de utilizador que é. No fim é criado o utilizador registando-o na lista e posteriormente no ficheiro.

3.4 *Login*

Nesta funcionalidade pedimos ao utilizador que nos indique o seu nome e a sua palavra-passe. Com isto verificamos se esse utilizador existe e se a palavra-passe corresponde. Tendo o *login* feito passamos para o menu em que o utilizador escolhe o que quer fazer.

3.5 Adicionar Atividade

Aqui o utilizador regista uma atividade no programa. Pedimos ao utilizador que indique, o nome da atividade e o tempo que esta demorou. Depois pergunta qual é o tipo de atividade (*Pushups, Rowing, Trail* ou *Weightlifting*) havendo depois perguntas especificas para cada atividade diferente, por exemplo para os *Pushups* é perguntado o número de repetições mas para o *Trail*, é pedido a distância e o ganho de elevação.

3.6 Fazer Atividade

Nesta opção o utilizador indica qual é a atividade que pretende registar que realizou, dizendo o nome da atividade. O programa de pois indica qual foi o gasto de calorias feito ao longo da atividade.

3.7 Adicionar Plano de Treino

Nesta funcionalidade o utilizador regista um plano de treino dizendo quais são as atividades que pertencem ao plano e quantas vezes por semana é realizado o plano.

3.8 *Prints*

Há três opções de *print* (*Print Activities, Print User* e *Print Plan*) que mostra o que está registado nos ficheiros.

Encapsulamento

Sendo o encapsulamento uma parte fundamental na programação orientada a objetos, não podíamos deixar de parte a necessidade de ter um projeto encapsulado. Na nossa aplicação, o encapsulamento baseia-se em não da *get* diretamente mas sim dando *get* de um *clone*.

```
public Activity clone() {
    return new Activity(this);
}
```

Figura 4.1: Exemplo de *clone*

Diagrama de Classes

Para melhor perceber a estrutura do nosso projeto, realizamos um diagrama de classes com a ajuda do programa *Visual Paradigm*.

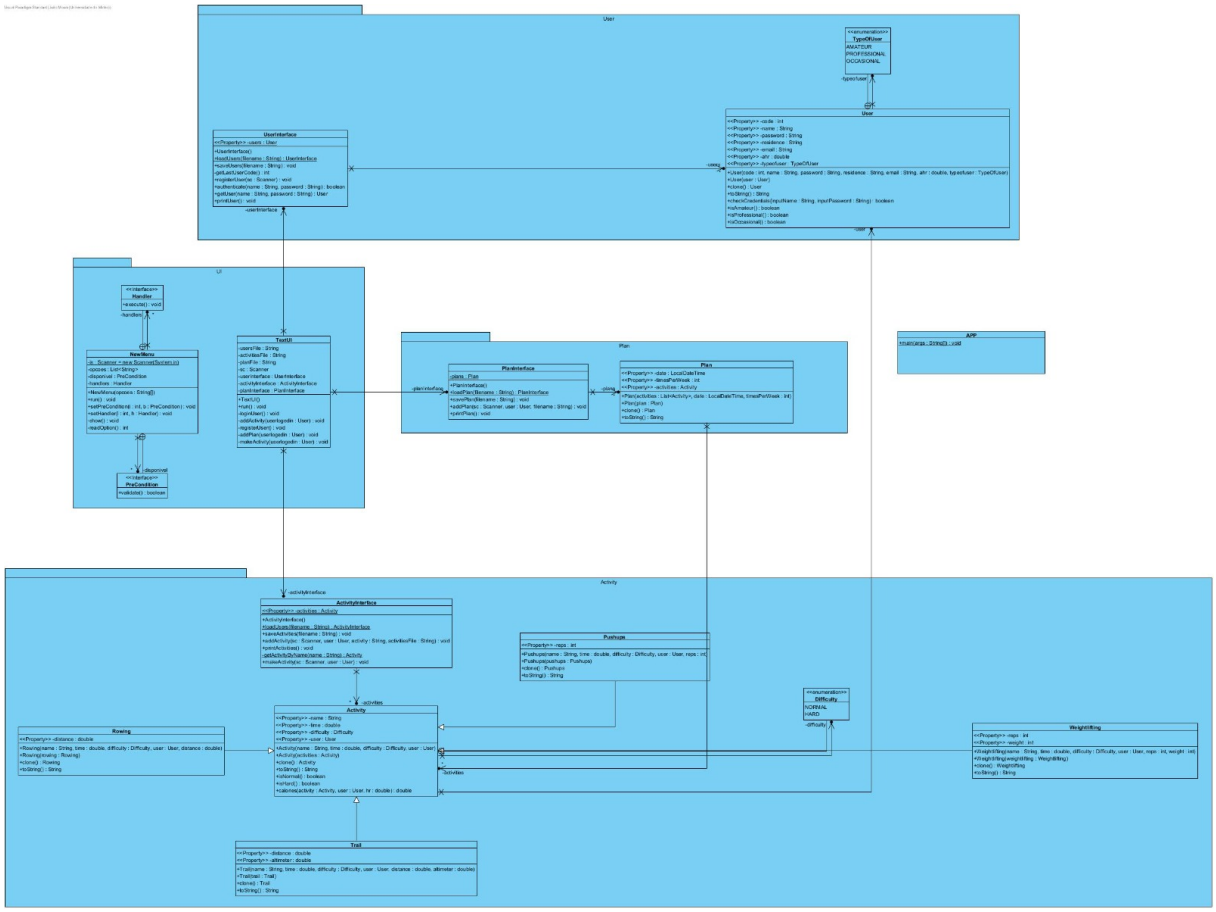


Figura 5.1: Diagrama de classes

Conclusão

Em resumo, sentimos que conseguimos aplicar bem os conceitos de programação orientada aos objetos, cumprindo sempre com o encapsulamento e com a modularização correta. Consideramos que temos um programa sólido e robusto.

6.1 Trabalho Futuro

- Implementar as estatísticas, mostrando os *records* de cada atividade.
- Criar a função que recomenda certos planos de treino tendo em conta o objetivo do utilizador