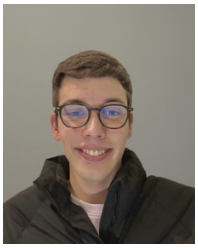
Programação Orientada aos Objetos

Ano letivo 2023/2024

Grupo 40



Diogo Neto A98197

Pedro Argainha A104351

Tomás Ogando A93174

1 – Introdução

O presente relatório tem como objetivo relatar o processo de desenvolvimento do projeto proposto, um software de gestão de uma aplicação fitness.

Este projeto foi desenvolvido no âmbito da unidade curricular, utilizando a linguagem de programação JAVA com a respetiva orientação aos objetos como principal objetivo.

2 – Classes

2.1 – Model

2.1.1 – Atividades

private String id;

private String nome;

private LocalDateTime dataRealização;

private long duracaoMin;

private double frequanciaCardiacaMedia;

private Utilizador utilizador;

A classe "Atividades" desempenha um papel fundamental no nosso sistema, armazenando informações relevantes sobre as atividades realizadas pelos utilizadores. Abaixo estão detalhados os atributos e métodos principais desta superclasse:

- **id**: Identificador único da atividade.
- **nome**: Nome da atividade.
- **dataRealização**: Data e hora em que a atividade foi realizada.
- **duracaoMin**: Duração da atividade em minutos.
- **frequanciaCardiacaMedia**: Frequência cardíaca média durante a atividade.
- **utilizador**: Referência ao utilizador que realizou a atividade.

Para além desta superclasse, decidimos criar 5 interfaces que visam a implementação de novas atividades de forma fácil, sendo elas:

- Repetições
- DistanciaAltimetria
- Distancia
- RepeticoesPesos
- Hard

As 4 primeiras atividades enumeradas permitem categorizar todos os desportos e implementar os métodos que o sistema impões, já a interface Hard, pode ser de implementação facultativa e permite ao sistema identificar atividades que exigem mais esforço por parte do utilizador de modo que sejam criados planos de treino coerentes.

2.1.1.1 – Alongamentos

private int repeticoes;

private int series;

private BodyPart parteCorpo;

A classe "Alongamentos" é uma subclasse da classe "Atividades", desempenhando um papel específico na representação de atividades do tipo alongamento no nosso sistema. Abaixo estão representados os atributos específicos desta subclasse, uma vez que esta herda os atributos to ponto

- **repeticoes**: Número de repetições realizadas durante o alongamento.
- **series**: Número de séries de alongamentos realizadas.
- **parteCorpo**: Parte do corpo, representada pelo enum "BodyPart".

Esta subclasse amplia a funcionalidade da classe "Atividades", permitindo a representação e armazenamento de informações específicas relacionadas às atividades de alongamento. Com estes atributos adicionais, podemos registar elementos específicos sobre a atividade, assim como implementar fórmulas de cálculo diferentes.

Implementa interface Distancia.

2.1.1.2 – CorridaTerreno

private double distancia;

private double altimetria;

private TipoTerreno tipoTerreno;

A classe "CorridaTerreno" é uma subclasse da classe "Atividade", projetada para representar atividades específicas de corrida em terreno variado no nosso sistema. Abaixo estão detalhados os atributos específicos desta subclasse:

- distancia: Distância percorrida durante a corrida.

- altimetria: Altitude alcançada durante a corrida.

- tipoTerreno: Tipo de terreno em que a corrida foi realizada, representado pelo enum "TipoTerreno".

Esta subclasse estende a funcionalidade da classe "Atividade", fornecendo campos adicionais para registar informações específicas relacionadas às atividades de corrida em terreno variado. Com atributos extra como a distancia percorrida, a altimetria do percurso e o tipo de terreno percorrido o sistema consegue fazer uma série de cálculos específicos para o cálculo de calorias gastas ou frequência cardíaca média durante a atividade.

Implementa interface DistanciaAltimetria.

2.1.1.3 – Remo

private double distancia;

private TipoTerreno tipoterreno;

A classe "Remo" é uma subclasse da classe "Atividade", projetada para representar atividades específicas de remo no nosso sistema. Abaixo estão detalhados os atributos específicos e métodos adicionais dessa subclasse:

- **distancia**: Distância percorrida durante o remo.

- **tipoTerreno**: Tipo de terreno em que o remo foi realizado, representado pelo enum "TipoTerreno", neste caso pode é apenas “Pista”

Esta subclasse estende a funcionalidade da classe "Atividade", fornecendo campos adicionais para registar informações específicas relacionadas às atividades de remo o que permite uma fórmula de cálculo de calorias ou frequência cardíaca média personalizada.

Implementa interface Repeticoes.

2.1.1.4 – Pesos

private double peso;

private int repetições;

private int series;

A classe "Pesos" é uma extensão da classe "Atividade" e tem a finalidade de representar atividades específicas de levantamento de pesos dentro do nosso sistema. Abaixo estão descritos os atributos exclusivos desta classe:

- **Peso**: Representa o peso levantado durante a atividade

- **Repetições**: Indica o número de repetições realizadas durante a atividade.

- **Séries**: Refere-se ao número de séries de levantamento de pesos executadas.

Essa subclasse foi criada para estender as funcionalidades da classe "Atividade", possibilitando o armazenamento de informações específicas relacionadas aos treinos com pesos. Com esses atributos adicionais, conseguimos registar detalhes importantes sobre cada sessão de treino, incluindo o peso utilizado, quantidade de repetições e número de séries realizadas. Isso proporciona uma visão detalhada do progresso e desempenho dos utilizadores nas atividades de musculação, assim como permite fórmulas personalizadas.

Implementa interface RepeticoesPesos.

2.1.1.5 – Btt

Private Int dificuldade;

Private double distancia;

Private double altimetria;

Private TipoTerreno tipoTerreno;

Private Boolean hard;

A classe "BTT" é uma subclasse da classe "Atividade" e foi projetada para representar atividades específicas de ciclismo todo-o-terreno (BTT) dentro do nosso sistema. Abaixo estão detalhados os atributos exclusivos desta classe:

- **Dificuldade**: Representa o nível de dificuldade da atividade, variando de 1 a 5.
- **Distância**: Indica a distância percorrida durante a atividade de BTT.
- **Altimetria**: Refere-se à variação de altitude durante a atividade.
- **Tipo de Terreno**: Descreve o tipo de terreno em que a atividade foi realizada.
- **Hard**: Indica se a atividade é considerada de dificuldade elevada

Essa classe foi criada para proporcionar uma representação precisa e detalhada das atividades de BTT realizadas pelos utilizadores do sistema. Com esses atributos específicos, podemos registrar e analisar informações relevantes para os praticantes de ciclismo todo-o-terreno, incluindo a dificuldade do percurso e as características do terreno percorrido.

Implementa as interfaces DistanciaAltimetria e Hard.

2.1.2 – Utilizadores

private String id;

private String email;

private String password;

private String nome;

private Genero genero;

private double altura;

private double peso;

private LocalDate dataNascimento;

private double frequenciaCardiacaMediaRepouso;

private double fatorMultiplicativoCalorias;

private Map<String, Atividade> atividades;

private Map<String, PlanoTreino> planodeTreino

A classe "Utilizadores" desempenha um papel crucial no sistema, representando os utilizadores que utilizam a aplicação fitness. Abaixo estão detalhados os principais atributos e métodos desta classe:

- **id**:Identificador único do utilizador.

- **email**: email do utilizador

- **password**: palavra passe do utilizador

- **nome**: Nome completo do utilizador.

- **genero**: Género do utilizador, representado por um enum "Genero".

- **altura**: Altura do utilizador em metros.

- **peso**: Peso do utilizador em quilogramas.

- **dataNascimento**: Data de nascimento do utilizador.

- **frequenciaCardiacaMediaRepouso**: Frequência cardíaca média do utilizador em repouso.

- **fatorMultiplicativoCalorias**: Um valor calculado a partir do género, idade, peso, altura e frequência cardíaca média do utilizador, utilizado para calcular as calorias gastas durante as atividades.

- **atividades**: Um mapa que armazena as atividades **realizadas** pelo utilizador, utilizando o identificador único como chave.

- **planodeTreino**: Um mapa que armazena os planos de treino atribuídos ao utilizador, utilizando o identificador único como chave.

Essa estrutura permite uma gestão abrangente dos utilizadores, incluindo informações pessoais, histórico de atividades e planos de treino atribuídos.

Esta super classe contém 3 subclasses:

- Utilizador Profissional
- Utilizador Ocasional
- Utilizador Amador

Que não adicionam nenhum atributo ao utilizador porém são extremamente uteis para definir fórmulas personalizadas para cada tipo de utilizador, de forma a obter um cálculo de frequência cardíaca média em repouso e fator multiplicativo de calorias adequado ao estilo de vida do utilizador.

2.1.3 – Fitness

private String nomeGinasio;

private LocalDateTime data;

private Map<String, Utilizador> utilizadoresMap; // id -> utilizador

private List<Atividade> atividades;

private List<PlanoTreino> planosTreino;

private List<Comparator<Utilizador>> comparatorsEstatisticasUser;

A classe "Fitness" desempenha um papel central no sistema, representando o ginásio ou centro fitness que utiliza a aplicação.

Apesar de apresentar alguns métodos que controlam as classes do model, esta classe tem características próprias, assim como os seus métodos de cálculo.

Abaixo estão detalhados os principais atributos desta classe:

- **nomeGinasio**: O nome do ginásio ou centro de fitness.

- **data**: Data em que o ginásio se encontra

- **utilizadoresMap**: Um mapa que armazena os utilizadores registados no ginásio, utilizando o identificador único como chave.

- **atividades**: Uma lista que armazena as atividades pré-definidas oferecidas pelo ginásio.

- **planosTreino**: Uma lista que armazena os planos de treino pré-definidos disponíveis no ginásio.

- **comparatorsEstatisticasUser**: Uma lista de comparadores utilizados para estatísticas de utilizadores.

Essa estrutura permite a gestão abrangente do ginásio na aplicação, incluindo a gestão de utilizadores, atividades e planos de treino oferecidos.

2.2 – Controller

Private Fitness f;

Classe que implementa o Controller, responsável pela interligação entre a View e o Model do nosso projeto.

- **Fitness** fitness - presente no model

2.3 – View

private Scanner sc

private Controller c

Classe que implementa a View, interface gr´afica, auxiliando o utilizador na manipulação do estado do programa.

- **sc** Scanner de interação com o utilizador

- **c** Controller

3 – Funcionamento do programa

Tal como nos foi proposto e aconselhado, dividimos o programa em 3 partes, respeitando assim o MVC (Model, View e Controller). A nossa classe Model é a classe Fitness. Nela, encontramos todo o tipo de relações entre classes relativas ao backoffice do programa, sua API, queries e interações entre as outras diversas classes existentes no pacote Model.

View está localizada na diretoria View, contendo todo o código relativo à interface, ou seja, parte visual da aplicação. Contém de igual forma os mecanismos de interação com o utilizador de forma que este possa manipular e trabalhar com o programa. Mais à frente no relatório falaremos melhor do componente View.

Por fim, o Controller trata de fazer a ponte entre o backend e o frontend do nosso programa. Recebe o input obtido da interação com o utilizador, auxiliado pela View, efetuando o pretendido através de pedidos feitos à API do Model.

3.1 – View

A nossa view dispõe de uma série de menus que permitem visualizar estatísticas, criar atividades e planos de treino interactivamente, assim como navegar por menus administrativos que permitem recolher informações relativas à data da simulação, logs da mesma e até guardar a simulação atualmente em curso.

3.2 – Simulação

Outra componente fundamental do nosso programa consiste na simulação da aplicação fitness, desta forma, após a criação manual de algumas atividades/planos pré-definidos ou ao carregar um objeto de estado, podemos realizar uma simulação dando uma data ou o número de dias que queremos avançar em relação à data atual do ginásio. Após efetuar os cálculos do dispêndio de calorias e atualizar os maps, é possível consultar o resultado das queries referidas no enunciado.

4 – Conclusão

Acreditamos ter respeitado as regras no que toca à Programação Orientada a Objetos, garantindo o encapsulamento necessário entre as classes e tirando proveito da modularidade e do polimorfismo, uma vez que, como pedido no enunciado, o programa foi projetado para que novos tipos de atividades sejam criados sem muito esforço no que toca à compreensão do código uma vez que as interfaces facilitam a implementação desta nova atividade.

O nosso código, particularmente no pacote Model, foi projetado para utilizar o máximo de iteradores internos possível, uma vez que, nos poupam linhas de código e até na criação de novos métodos. Um exemplo seriam as queries implementadas para estatísticas de utilizadores pois, estas utilizam o mesmo método e os mesmos iteradores internos, apenas muda o comparador passado como parâmetro para método, o que facilita bastante a criação de novas queries para utilizadores.

Em suma, o desenvolvimento deste projeto solidificou e ajudou a consolidar os conhecimentos adquiridos nas aulas teóricas, tendo sido importante para ter uma percepção empírica das implicações do desenvolvimento num ambiente orientado aos objetos.

5 – Diagrama de classes