Universidade do Minho
Escola de Ciências

Universidade do Minho

Licenciatura em Ciências da Computação

Trabalho Prático Programação Orientada aos Objetos

Implementação de uma aplicação para gestão de atividade física

Grupo xx

Trabalho realizado por:

João Pedro da Silva Barbosa (A100054)

Diogo Coelho da Silva (A100092)

Pedro Miguel Ramôa Oliveira (A97686)

Conteúdo

Índice de figuras

1.Introdução

No âmbito deste trabalho prático, foi proposta a implementação de uma aplicação que faça a gestão de atividades físicas. Com esta aplicação, deve ser possível a cada utilizador registar a realização de atividades e obter um resumo das estatísticas das realizações das mesmas.

Este projeto foi também pensado para respeitar as ideias deste paradigma da programação, a programação orientada aos objetos. Com base nisto, tivemos de respeitar os seus ideais que incluem a abstração, encapsulamento e herança de classes, entre outros conceitos base de programação orientada a objetos.

Este relatório tem como objetivo apresentar a nossa linha de pensamentos para a conceção e desenvolvimento de funcionalidades da aplicação proposta. Para isso vamos fazer uma pequena explicação da arquitetura de classes utilizada, descrevendo brevemente as mesmas e os seus métodos.

2.Arquitetura da aplicação

Diagrama de classes do projeto. Para uma melhor leitura do mesmo, recomenda-se que se abra o ficheiro .jpeg na pasta principal do projeto. No mesmo é possível dar zoom nas classes para permitir uma melhor visualização das mesmas. Este diagrama foi criado com recurso ao software starUML. As tabelas a verde representam subclasses das classes de tipo de atividade, que são as superclasses e estão representadas a amarelo. As tabelas a vermelho representam todas as superclasses principais do projeto, utilizador, atividade e planos de treino. E por fim, as tabelas a azul representam as classes *Main* e *Menu*.

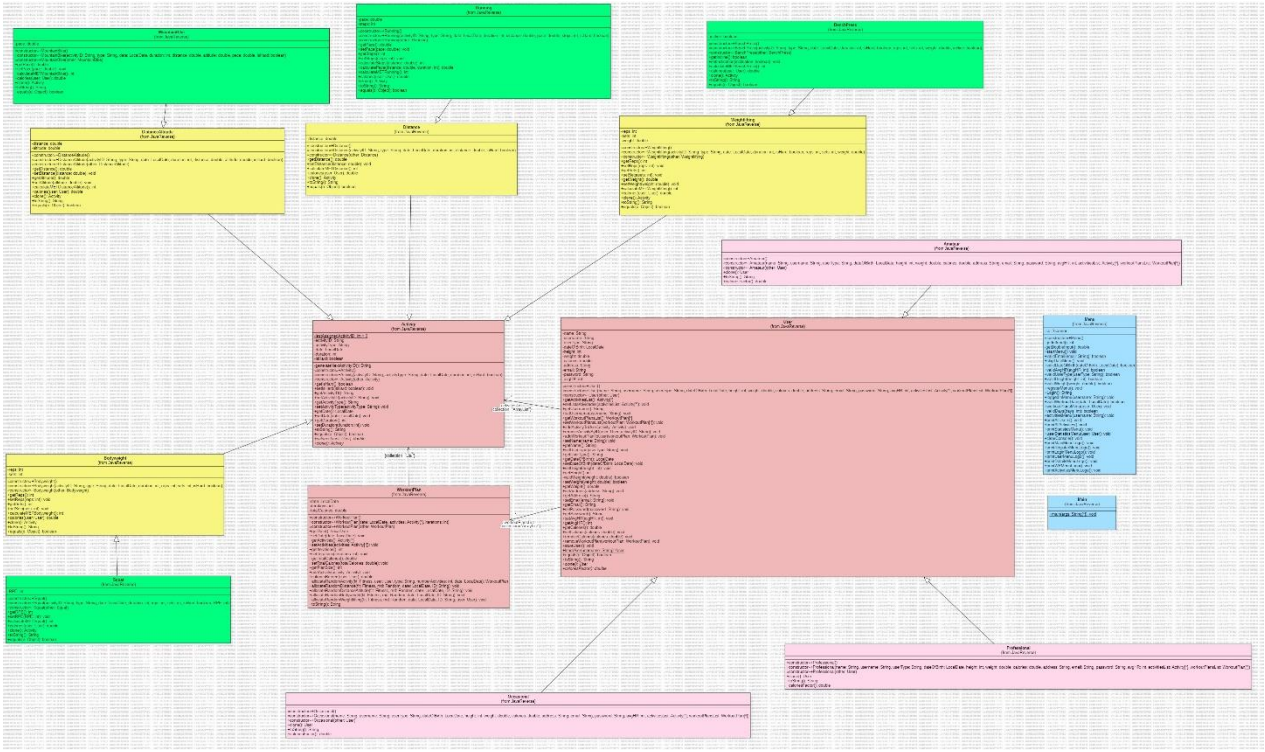

Figura 1- Diagrama de classes

2.1Classe *Activity*

Esta classe desempenha o papel de uma estrutura base para representar uma variedade de atividades físicas. Nela, estão encapsuladas informações comuns a todas as atividades, como identificação, tipo, data e duração e um identificador de dificuldade da atividade (*isHard*). Esta classe serve como superclasse para as classes abstratas *"Distance", "DistanceAltitude", "Weightlifting"* e *"Bodyweight",* partilhando variáveis de instância e implementação de métodos com a mesma assinatura. Esta abordagem permite promover uma estrutura modular e mais organizada para o código, facilitando a manutenção e expansão da aplicação caso pretendido. Ao utilizar herança e polimorfismo, as outras classes podem ser estendidas a partir desta classe para representar diferentes tipos de atividades, mantendo a estrutura, aumentando assim a consistente e a coesão dentro da aplicação. Isto contribui para os princípios do POO.

2.2Classe *Distance*

A classe mencionada serve como uma estrutura base e fundamental para a classe "*Running*", sendo a sua superclasse. Todos os exercícios relacionados com o tipo *"Distance"* são representados por esta classe. Ambas as classes partilham variáveis de instância, como por exemplo os atributos identificação, tipo, data e duração, o que facilita a herança e cria um nível mais elevado de abstração. No entanto, para a classe *"Running",* são adicionados os parâmetros de ritmo (*pace*) e passos (steps), que são específicos para esta atividade. Esta abordagem foi escolhida pelo grupo por proporcionar uma maior facilidade para a expansão da aplicação, permitindo a adição de novos exercícios deste tipo. Além disso, na classe "*Distance*", foi implementado o cálculo de calorias específico para atividades de corrida, levando em consideração fatores como a idade do utilizador. Este cálculo é realizado através do método *calories(User user),* que multiplica o *MET* (*Metabolic Equivalent of Task*) específico para a corrida, a duração da atividade, o peso do utilizador e o fator de calorias do utilizador. Esta implementação do cálculo de calorias na classe "*Distance*" reflete a preocupação do grupo em criar uma estrutura coesa e abrangente para representar diversas atividades físicas, mantendo os princípios da Programação Orientada a Objetos (POO) de reutilização de código, modularidade e hierarquia de classes para promover uma arquitetura de código robusta e escalável.

2.3 Classe *DistanceAltitude*

Em semelhança com a anterior, as três próximas classes foram implementadas com o mesmo objetivo da primeira, satisfazer os requisitos do paradigma de Programação Orientada aos Objetos (POO). A classe mencionada serve como uma estrutura base e fundamental para a classe "*MountainBike*", sendo a sua superclasse. Todos os exercícios relacionados com o tipo "*DistanceAltitude*" são representados pela classe. Assim como nas classes anteriores, ambas compartilham variáveis de instância, como identificação, tipo, data e duração. No entanto, para a classe "*MountainBike*", é adicionado o parâmetro de ritmo (*pace*), que é específico para este tipo de atividade. Além disso, o cálculo de calorias é diferente para o "*MountainBike*",

levando em consideração fatores específicos, como a idade do utilizador. Este cálculo é realizado através da função *calories(User user)* que multiplica o *MET (Metabolic Equivalent of Task)* específico para o *MountainBike*, a duração da atividade, o peso do utilizador, e o fator de calorias do utilizador. Esta abordagem foi escolhida pelo grupo por proporcionar uma maior facilidade para a expansão da aplicação, permitindo a adição de novos exercícios deste tipo, mantendo a consistência e a coesão do código. Estes princípios refletem os requisitos do paradigma de Programação Orientada a Objetos (POO), enfatizando a reutilização de código, modularidade e hierarquia de classes para uma arquitetura de código mais robusta e escalável.

2.4 Classe *WeightLifting*

A classe mencionada serve como uma estrutura base e fundamental para a classe "*BenchPress*", sendo a sua superclasse. Todos os exercícios relacionados com o tipo "*WeightLifting*" são representados por esta classe. Ambas as classes partilham variáveis de instância, como identificação, tipo, data e duração, o que facilita a herança e cria um nível mais elevado de abstração. Além disso, para a classe "*BenchPress*", é adicionado o parâmetro de inclinação (*inclination*), que representa a inclinação do banco durante o exercício. Essa abordagem foi escolhida pelo grupo por proporcionar uma maior facilidade e flexibilidade para a expansão da aplicação, permitindo a adição de novos exercícios deste tipo. Também foi implementado o cálculo de calorias específico para a atividade de "*BenchPress*". Esse cálculo é realizado através da função *calories(User user),* que leva em consideração o fator de calorias do utilizador. Essa integração do cálculo de calorias na classe "*BenchPress*" reflete a preocupação do grupo em criar uma estrutura coesa e abrangente para representar diversos exercícios de levantamento de peso, mantendo os princípios da Programação Orientada a Objetos (POO) de reutilização de código, modularidade e hierarquia de classes para promover uma arquitetura de código robusta e escalável.

2.5 Classe *BodyWeight*

A classe mencionada serve como uma estrutura base e fundamental para a classe "*Squat*", sendo a sua superclasse. Todos os exercícios relacionados com o tipo "*BodyWeight*" são representados por esta classe. Ambas as classes partilham variáveis de instância, como identificação, tipo, data, duração, repetições (reps) e séries (*sets*), o que facilita a herança e cria um nível mais elevado de abstração. Além disso, para a classe "*Squat*", é adicionado o parâmetro *RPE (Rate of Perceived Exertion*), que representa a percepção subjetiva do esforço de cada utilizador durante o exercício. Essa abordagem foi escolhida pelo grupo por proporcionar uma maior facilidade e flexibilidade para a expansão da aplicação, permitindo a adição de novos exercícios deste tipo. Também foi implementado o cálculo de calorias específico para a atividade de "*Squat*". Esse cálculo é realizado através da função *calories(User user),* que leva em consideração o fator de calorias do utilizador. Esta integração do cálculo de calorias na classe "*Squat*" reflete a preocupação do grupo em criar uma estrutura coesa e abrangente para representar diversos exercícios de peso corporal, mantendo os

princípios da Programação Orientada a Objetos (POO) de reutilização de código, modularidade e hierarquia de classes para promover uma arquitetura de código robusta e escalável.

2.6 Classe Utilizador

Esta classe abstrata representa todos os utilizadores que interagem com o programa. Esta classe foi escolhida para superclasse de "*Professional*", "*Amateur*" e "*Occasional*" devido a possuírem variáveis de instância em comum. Estas último servem para representar um tipo de utilizador em específico. Cada tipo de utilizador conta com métodos de cálculos diferentes para refletir na sua condição.

Da forma que a classe está implementada é relativamente fácil adicionar outro tipo de utilizador à aplicação.

2.7 Classe *Fitness*

A classe "*Fitness*" é responsável por fazer a gestão de dados da aplicação relacionados com as atividades dos utilizadores. Esta classe foi criada para encapsular os dados relativos às atividades físicas dos utilizadores, incluindo as variáveis de instância e métodos.

2.8 Classe Menu

A classe "Menu" é responsável por fornecer uma interface interativa com o utilizador, em modo texto.

2.9 Classe *WorkoutPlan*

Esta classe é responsável por criar planos de treino para um utilizador. Os planos de treino são gerados de acordo com um input de dias por parte do utilizador da aplicação.

3.Funcionalidades

Nas próximas imagens, vão ser representadas todas as funcionalidades implementadas na nossa aplicação.

Figura 2- Menu principal

Menu principal da aplicação com opções de registar utilizadores e fazer login com um utilizador já criado. Não há utilizadores repetidos. As opções 4 e 5 servem para fazer *debug*. Nos melhores dos casos, e numa aplicação de mundo real, estas opções não estariam disponíveis para os utilizadores "comuns", apenas para o *admin* da aplicação.

Figura 3-Menu de registo de um utilizador

A interface de registo de um utilizador pede ao utilizador da aplicação que indique os campos necessários, havendo um novo pedido se o utilizador introduzir um valor invalido no campo.

Figura 4- Menu de login de utilizador

O menu de registo do utilizador pede ao utilizador que introduza os seus dados de login. Se os dados estiverem errados, são pedidos novos dados ao utilizador.

Figura 5 -Mensagem de erro de login

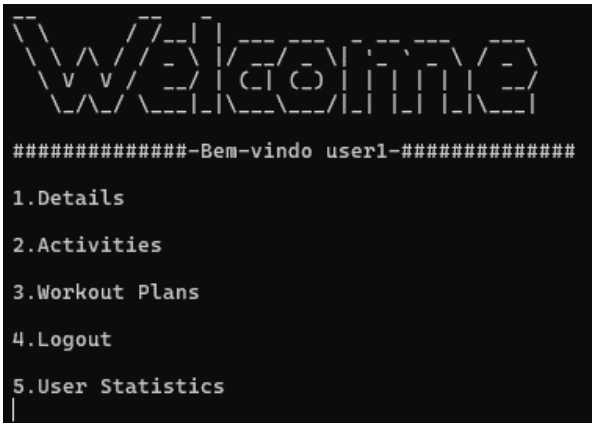

Figura 6 - Menu do utilizador

Este menu é apresentado ao utilizador após a tentativa certa de login por parte do utilizador. Aqui ele pode escolher visualizar os detalhes da sua conta, ir para o menu de atividades e de planos de treino, dar logout e ver algumas das suas estatísticas.

Figura 7 - Interface dos detalhes do utilizador

Esta interface é apresentada quando o utilizador pede os detalhes do user. São listadas informações básicas sobre o utilizador, assim como uma soma das calorias que o utilizador gastou em realizar todas as atividades e as respetivas atividades executadas.

Figura 8-Menu das atividades

Neste menu é possível executar atividades guardadas, ou seja, simular a execução de uma atividade por parte de um utilizador e dar log aos detalhas da atividade, apagar uma atividade existente do mapa de atividades, listar todas as atividades disponíveis e adicionar atividades ao mapa de atividades.

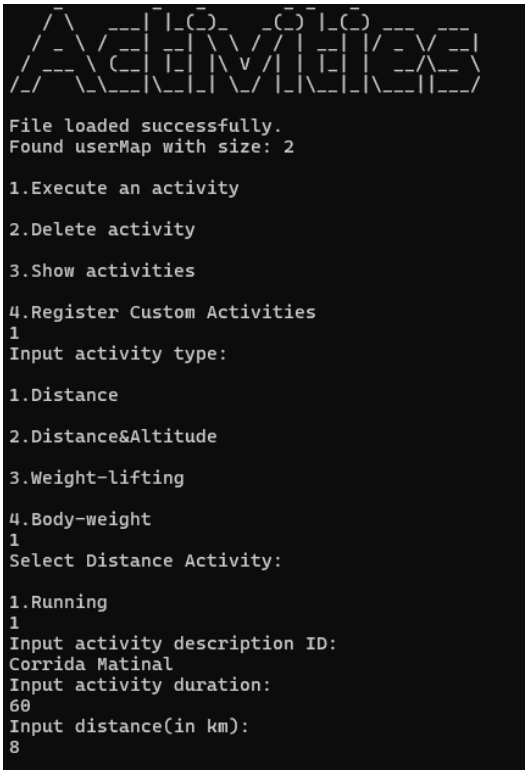

Figura 9 -Execução de uma atividade

Neste menu é possível executar uma atividade. Ela pede ao utilizador os dados da atividade e guarda os dados no registo de atividades do utilizador.

Figura 10 - Interface de adicionar atividades

Neste menu é possível adicionar atividades a lista de atividades da aplicação. As atividades adicionadas estendem da atividade de tipo base e dos seus parâmetros. Por exemplo se adicionarmos. Neste caso criamos uma atividade "Caminha", que é estendida a partir do tipo de atividade "*Distance*" e terá os mesmos parâmetros que a anterior.

Figura 11-Menu planos de treinos

Neste menu é possível adicionar planos de treino a um utilizador.

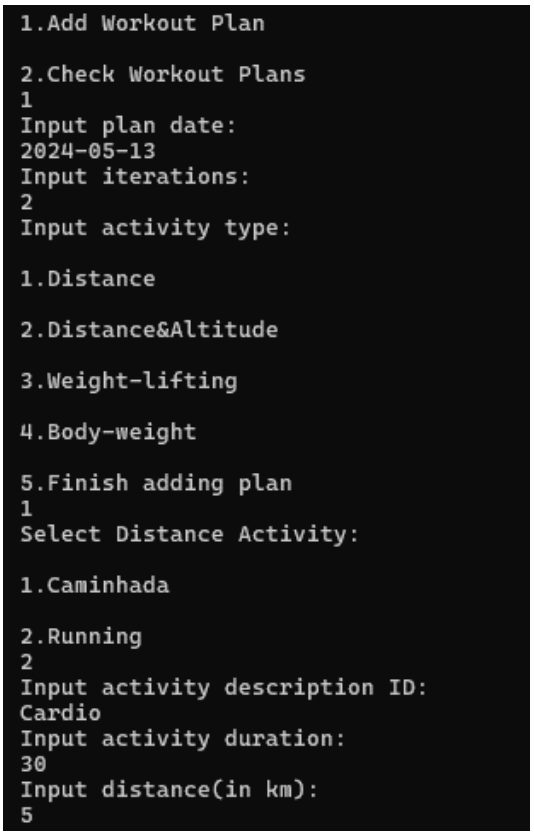

Figura 12- Adicionar um plano de treinos

Quando um utilizador adiciona um plano de treino, este vai ficar agendado, sendo executado com recurso ao avanço no tempo.

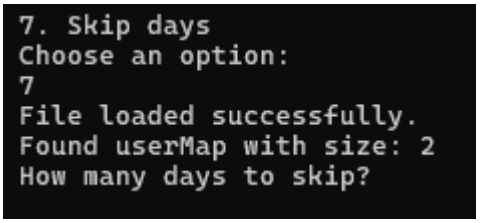

Figura 13-Avançar data

Neste menu é possível avançar a data. Todas as atividades e planos de treinos dos utilizadores são simuladas, e os seus dados guardados nos registos dos utilizadores.

4.Explicação da arquitetura e algumas decisões

Ao longo da implementação do projeto, o grupo decidiu tomar as seguintes decisões:

1. Cada utilizador criado vai gerar um novo ficheiro. ser (*serialized object*), como o nome do utilizador, para guardar as informações de forma persistente e, assim, ser possível carregar os dados ao iniciar a aplicação.

2. Os dados das atividades e planos de treino gerados vão ser guardados num ficheiro *data.ser* para serem carregados ao iniciar a aplicação.
3. É possível adicionar atividades "*custom*", que nada mais nada menos, são instâncias com o nome modificado das classes de tipo selecionadas.
4. Todas as informações relacionadas com as atividades (ex.: calorias gastas), são guardadas em cada utilizador, após a simulação de atividades agendadas.
5. As atividades agendadas, pertencem aos planos de treino gerado.
6. A nossa noção de realizar um exercício e ser possível dar log as informações do mesmo é a nossa opção 1, ou seja, adicionar um exercício. A noção de adicionar uma atividade é de criar uma atividade *custom*.
7. O programa começa com 4 atividades base, uma por cada tipo de atividade.
8. Devido a forma como o nosso projeto está organizado, é possível adicionar atividades base, ou sejam outros tipos de atividades e também outros tipos de utilizadores com bastante facilidade.

5.Implementação da noção hard

O grupo decidiu implementar a noção de *hard activity*, quando uma atividade satisfaz certos requisitos. Estes requisitos diferem de classe para classe, mas quando atingidos, ativam uma *flag* que coloca o *boolean isHard* em *true*. Assim, e para todas as atividades temos a noção *hard* implementada.

6. Conclusão

Em resumo, a realização e desenvolvimento desta aplicação foi uma tarefa desafiadora, em que foram aplicados os conceitos lecionados ao longo do semestre na UC. Durante o processo de implementação da aplicação, o grupo tentou garantir a implementação de uma arquitetura sólida e flexível, que satisfaça os conceitos da programação orientada aos objetos.

O grupo optou por adotar uma abordagem baseada em herança e encapsulamento, conseguindo criar uma hierarquia de classes bem estruturada, permitindo uma maior facilidade para a expansão da aplicação e manutenção da mesma. A utilização de classes abstratas proporcionou uma organização mais eficiente dos dados e comportamentos comuns, enquanto as subclasses permitiram uma representação mais precisa e específica.

Destacamos também o sucesso em implementar uma interface *user friendly* que proporciona ao utilizador uma maior facilidade em interagir com a aplicação.

Destacamos como parte mais difícil na implementação deste projeto, a simulação de avanço de tempo e a capacidade de gerar planos de treino aleatórios. Embora implementados com sucesso, foi sem dúvida uma parte que fez perder muito tempo na sua implementação.

Para concluir, o grupo está satisfeito com o resultado deste projeto, pois assegura todos os requisitos propostos no enunciado, cumprindo todos os requisitos do paradigma de programação estudado na UC.