



Universidade do Minho  
Escola de Engenharia  
Licenciatura em Engenharia Informática

## Unidade Curricular de Programação Orientada aos Objetos

Ano Letivo de 2023/2024

### MakelItFit



Afonso Santos  
a104276



Hélder Gomes  
a104100



Pedro Pereira  
a104082

11 de maio de 2024

# Poo

Data da Receção	
Responsável	
Avaliação	
Observações	

## MakItFit

**Afonso Santos    Hélder Gomes    Pedro Pereira**  
a104276            a104100            a104082

11 de maio de 2024

## Resumo

O presente trabalho teve como objetivo desenvolver uma aplicação de gestão de atividades físicas e planos de treino para praticantes de exercício físico. A aplicação permite registar diferentes tipos de atividades físicas, como corrida, corrida no monte, flexões e musculação, além dos dados dos utilizadores e seus perfis (profissional, amador ou praticante ocasional).

Foram implementadas funcionalidades para criar e gerir utilizadores, atividades e planos de treino. É possível registar a realização de atividades pelos utilizadores, calculando automaticamente o consumo calórico com base em fórmulas parametrizadas conforme os perfis dos utilizadores. Adicionalmente, foram desenvolvidos mecanismos para avançar a data simulada, executar atividades previstas nos planos de treino e atualizar os registo dos utilizadores.

O programa permite ainda gerar planos de treino personalizados para cada utilizador, de acordo com os seus objetivos em termos de tipos de atividades, frequência semanal, consumo calórico mínimo e outras restrições, como a inclusão de atividades intensas (“Hard”). Foram implementadas funcionalidades para obter várias estatísticas, como o utilizador que mais calorias despendeu num determinado período, o tipo de atividade mais praticada, entre outras.

A aplicação encontra-se totalmente funcional, permitindo a gestão completa de utilizadores, atividades e planos de treino, assim como a obtenção de estatísticas relevantes. Todos os requisitos propostos foram cumpridos, adotando-se boas práticas de programação orientada a objetos, além de garantir a robustez e tratamento adequado de erros. A aplicação é capaz de armazenar e recuperar o seu estado num ficheiro, facilitando a sua utilização contínua.

**Área de Aplicação:** Desenvolvimento no âmbito da Programação Orientada aos Objetos.

**Palavras-Chave:** Fitness, gestão desportiva, arquitetura de aplicação, JAVA, design pattern, MVC, controlo de qualidade de código.

# Índice

<b>1. Introdução</b>	<b>1</b>
1.1. Organização de requisitos	1
1.2. Estratégia de desenvolvimento	2
1.3. Design patterns	4
<b>2. Diagrama de classes</b>	<b>6</b>
2.1. Modelação das Atividades	6
2.2. Modelação dos Utilizadores	9
2.3. Modelação dos Planos de Treino	10
2.4. Modelação das classes de utilitários	11
2.4.1. Utilitários	11
2.4.2. Tempo	12
2.4.3. Menu	12
2.4.4. Estatísticas	13
2.4.5. Exceções	13
2.5. Modelação do MVC	14
2.6. Modelação final	16
<b>3. Implementação</b>	<b>17</b>
3.1. Contextos base da aplicação	17
3.2. Saltos temporais no sistema	17
3.3. Realização de estatísticas sobre o estado do programa	17
3.4. Criação de atividades “Hard”	18
3.5. Gerar um plano de treino com objetivos	18
3.6. Salvaguarda do estado da aplicação	18
<b>4. Model-View-Controller</b>	<b>19</b>
4.1. Conceito do MVC	19
4.2. Implementação do modelo	19
<b>5. Utilização da aplicação</b>	<b>21</b>
5.1. Funcionamento geral	21
5.2. Início da aplicação	21
5.3. Modo MakelFitView	21
5.3.1. Interagir com utilizador	22
5.3.2. Interagir com atividades	22
5.3.3. Interagir com planos de treino	23
5.3.4. Simulação temporal	24
5.3.5. Guardar estado	24
5.3.6. Terminar aplicação	25
5.4. Modo utilizador	25
5.4.1. Autenticação	25
5.5. Modo administrador	25
5.5.1. Remover utilizador	26
5.5.2. Listar todos os utilizadores	26
5.5.3. Execução de estatísticas	27

<b>6. Conclusão</b>	<b>29</b>
<b>Bibliografia</b>	<b>30</b>
<b>Lista de Siglas e Acrónimos</b>	<b>31</b>
<b>Anexos</b>	<b>32</b>
Anexo 1: Logo da Universidade do Minho.	32
Anexo 2: Exemplo de representação gráfica do padrão Abstract Factory.	32
Anexo 3: Exemplo de representação conceitual do padrão Abstract Factory na aplicação.	33
Anexo 4: Exemplo de representação gráfica do padrão Facade.	33
Anexo 5: Exemplo de representação conceitual do padrão Facade na aplicação.	34
Anexo 6: Exemplo de representação gráfica do padrão Observer.	34
Anexo 7: Exemplo de representação conceitual do padrão Observer na aplicação.	35
Anexo 8: Representação da interface "ActivityInterface" no diagrama de classes.	36
Anexo 9: Representação da classe abstrata "Activity" no diagrama de classes.	36
Anexo 10: Representação da classe abstrata "Repetitions" no diagrama de classes.	36
Anexo 11: Representação da classe abstrata "Distance" no diagrama de classes.	37
Anexo 12: Representação da classe abstrata "RepetitionsWithWeights" no diagrama de classes.	37
Anexo 13: Representação da classe abstrata "DistanceWithAltimetry" no diagrama de classes.	37
Anexo 14: Representação da classe concreta "PushUp" no diagrama de classes.	38
Anexo 15: Representação da classe concreta "Running" no diagrama de classes.	38
Anexo 16: Representação da classe concreta "WeightSquat" no diagrama de classes.	38
Anexo 17: Representação da classe concreta "Trail" no diagrama de classes.	38
Anexo 18: Representação da interface "HardInterface" no diagrama de classes.	39
Anexo 19: Representação da interface "UserInterface" no diagrama de classes.	39
Anexo 20: Representação da classe abstrata "User" no diagrama de classes.	39
Anexo 21: Representação da classe concreta "Amateur" no diagrama de classes.	40
Anexo 22: Representação da classe concreta "Occasional" no diagrama de classes.	40
Anexo 23: Representação da classe concreta "Professional" no diagrama de classes.	40
Anexo 24: Representação da classe concreta "UserManager" no diagrama de classes.	41
Anexo 25: Representação da classe concreta "TrainingPlan" no diagrama de classes.	41
Anexo 26: Representação da classe concreta "TrainingPlanManager" no diagrama de classes.	41
Anexo 27: Representação da classe concreta "MakeItFitDate" no diagrama de classes.	42
Anexo 28: Representação da classe concreta "EmailValidator" no diagrama de classes.	42
Anexo 29: Representação da classe concreta "TimeManager" no diagrama de classes.	42
Anexo 30: Representação da classe concreta "Menu" no diagrama de classes.	42
Anexo 31: Representação da classe concreta "MenuItem" no diagrama de classes.	43
Anexo 32: Representação da interface "PreCondition" no diagrama de classes.	43
Anexo 33: Representação da interface "Handler" no diagrama de classes.	43
Anexo 34: Representação da classe concreta "QueriesManager" no diagrama de classes.	44
Anexo 35: Representação da classe concreta "ExistingEntityConflictException" no diagrama de classes.	44
44	
Anexo 36: Representação da classe concreta "InvalidTypeException" no diagrama de classes.	44
Anexo 37: Representação da classe concreta "EntityDoesNotExistException" no diagrama de classes.	45
Anexo 38: Representação da classe concreta "MakeItFit" no diagrama de classes.	45
Anexo 39: Representação da classe concreta "MakeItFitController" no diagrama de classes.	45
Anexo 40: Representação da classe concreta "AdminView" no diagrama de classes.	46
Anexo 41: Representação da classe abstrata "MakeItFitView" no diagrama de classes.	46
Anexo 42: Representação da classe concreta "UserView" no diagrama de classes.	47

## **Lista de Figuras**

Figura 1: Exemplo de representação gráfica do padrão Abstract Factory. [1]	4
Figura 2: Exemplo de representação conceitual do padrão Abstract Factory na aplicação.	4
Figura 3: Exemplo de representação gráfica do padrão Facade. [2]	4
Figura 4: Exemplo de representação conceitual do padrão Facade na aplicação.	4
Figura 5: Exemplo de representação gráfica do padrão Observer. [3]	5
Figura 6: Exemplo de representação conceitual do padrão Observer na aplicação.	5
Figura 7: Representação da interface “ActivityInterface” no diagrama de classes.	6
Figura 8: Representação da classe abstrata “Activity” no diagrama de classes.	7
Figura 9: Representação da classe abstrata “Repetitions” no diagrama de classes.	7
Figura 10: Representação da classe abstrata “Distance” no diagrama de classes.	7
Figura 11: Representação da classe abstrata “RepetitionsWithWeights” no diagrama de classes.	7
Figura 12: Representação da classe abstrata “DistanceWithAltimetry” no diagrama de classes.	8
Figura 13: Representação da classe concreta “PushUp” no diagrama de classes.	8
Figura 14: Representação da classe concreta “Running” no diagrama de classes.	8
Figura 15: Representação da classe concreta “WeightSquat” no diagrama de classes.	8
Figura 16: Representação da classe concreta “Trail” no diagrama de classes.	8
Figura 17: Representação da interface “Hard” no diagrama de classes.	9
Figura 18: Representação da interface “UserInterface” no diagrama de classes.	9
Figura 19: Representação da classe abstrata “User” no diagrama de classes.	9
Figura 20: Representação da classe concreta “Amateur” no diagrama de classes.	10
Figura 21: Representação da classe concreta “Occasional” no diagrama de classes.	10
Figura 22: Representação da classe concreta “Professional” no diagrama de classes.	10
Figura 23: Representação da classe concreta “UserManager” no diagrama de classes.	10
Figura 24: Representação da classe concreta “TrainingPlan” no diagrama de classes.	11
Figura 25: Representação da classe concreta “TrainingPlanManager” no diagrama de classes.	11
Figura 26: Representação da classe concreta “MakeItFitDate” no diagrama de classes.	12
Figura 27: Representação da classe concreta “EmailValidator” no diagrama de classes.	12
Figura 28: Representação da classe concreta “TimeManager” no diagrama de classes.	12
Figura 29: Representação da classe concreta “Menu” no diagrama de classes.	12
Figura 30: Representação da classe concreta “MenuItem” no diagrama de classes.	13
Figura 31: Representação da interface “PreCondition” no diagrama de classes.	13
Figura 32: Representação da interface “Handler” no diagrama de classes.	13
Figura 33: Representação da classe concreta “QueriesManager” no diagrama de classes.	13
Figura 34: Representação da classe concreta “ExistingEntityConflictException” no diagrama de classes.	13
Figura 35: Representação da classe concreta “InvalidTypeException” no diagrama de classes.	14
Figura 36: Representação da classe concreta “EntityDoesNotExistException” no diagrama de classes.	14
Figura 37: Representação da classe concreta “MakeItFit” no diagrama de classes.	14
Figura 38: Representação da classe concreta “MakeItFitController” no diagrama de classes.	15
Figura 39: Representação da classe concreta “AdminView” no diagrama de classes.	16
Figura 40: Representação da classe abstrata “MakeItFitView” no diagrama de classes.	16
Figura 41: Representação da classe concreta “UserView” no diagrama de classes.	16

# 1. Introdução

Este relatório tem como objetivo abordar o projeto prático da Unidade Curricular de Programação Orientada a Objetos (POO) do ano letivo 2023/2024. Serão discutidas as decisões tomadas, as funcionalidades implementadas pelo grupo, bem como o método de raciocínio adotado para o desenvolvimento do projeto final. O objetivo deste trabalho consistiu no desenvolvimento de uma aplicação abrangente para a gestão de atividades físicas e planos de treino.

O enunciado do projeto estabeleceu requisitos progressivos, desde funcionalidades básicas de criação e gestão de entidades até recursos avançados, como a geração automática de planos de treino de acordo com objetivos específicos dos utilizadores. Ao longo deste relatório, abordaremos as decisões de design, a arquitetura de classes adotada, bem como os desafios enfrentados e as soluções implementadas para atender a todos os requisitos propostos.

## 1.1. Organização de requisitos

Para dar início ao projeto, procedeu-se a uma análise detalhada dos requisitos apresentados no enunciado, que serviu de base para definir os três pilares fundamentais da aplicação:

- **Atividades:** Engloba o registo, classificação e gestão de uma variedade de atividades físicas, desde as que envolvem distância e altimetria até as que consistem em séries de repetições com ou sem pesos.
- **Utilizadores:** Inclui a criação e gestão de perfis de utilizadores, contemplando diferentes níveis de experiência e objetivos, como profissionais, amadores e praticantes ocasionais.
- **Planos de treino:** Abrange a elaboração de planos de treino personalizados para cada utilizador, levando em consideração os tipos de atividade, frequência, intensidade e objetivos pessoais.

Desses pilares, foram definidos os seguintes requisitos generalizados para uma primeira fase de planeamento:

### 1. Gestão de atividades e planos de treino:

- Desenvolver uma aplicação que permita gerir diferentes tipos de atividades físicas, como corrida, ciclismo, pilates, levantamento de pesos, entre outras.
- Permitir a criação de planos de treino para utilizadores, indicando as atividades a realizar e seus detalhes.

### 2. Registo e acompanhamento das atividades:

- Permitir que cada utilizador registe a realização de atividades, associando detalhes como data, duração, designação e outros, para acompanhamento posterior.
- Calcular o consumo calórico com base no perfil do utilizador e na atividade realizada.

### 3. Simulação de avanço do tempo:

- Permitir a simulação de avanço do tempo para realizar atividades agendadas automaticamente e atualizar os registos dos utilizadores.

### 4. Suporte a diversos tipos de atividades:

- Incluir uma variedade de atividades, desde aquelas que envolvem distância e altimetria até as que consistem em séries de repetições, com ou sem pesos.
- Registar o tempo gasto em cada atividade e calcular o consumo calórico conforme a fórmula estipulada.

**5. Classificação das atividades em níveis de dificuldade:**

- Diferenciar atividades consideradas “Hard” de outras, como trail no monte e ciclismo de montanha.

**6. Gestão de diferentes tipos de utilizadores:**

- Permitir a existência de diferentes tipos de utilizadores (profissionais, amadores, ocasionais), com fácil extensão para adicionar novos tipos através de código.
- Definir um fator multiplicativo para cada tipo de utilizador, a ser utilizado no cálculo das calorias.

**7. Registo detalhado de utilizadores:**

- Manter registos do código do utilizador, nome, morada, email, telemóvel, altura, peso, frequência cardíaca média e outros.
- Associar atividades realizadas aos utilizadores, sejam elas isoladas ou parte de um plano de treino.

**8. Criação e gestão de planos de treino:**

- Permitir a construção de planos de treino, especificando parâmetros para construção automático ou introdução manual para processos detalhados.
- Considerar se as atividades serão realizadas uma única vez ou em várias iterações.

**9. Geração automática de planos de treino personalizados:**

- Criar planos de treino específicos para cada utilizador, considerando seus objetivos em termos de tipo de exercício e consumo calórico esperado.

## 1.2. Estratégia de desenvolvimento

O grupo decidiu implementar o desenvolvimento do projeto seguindo a metodologia Scrum e utilizar a plataforma Trello como ferramenta para planejar, acompanhar e distribuir as tarefas entre os três membros do grupo. Dentro desta metodologia, as atividades são organizadas em sprints, com prazos definidos, permitindo o acompanhamento do progresso e a avaliação regular dos resultados.

Em relação ao código, como observável no bloco de código em baixo, optámos por um modelo padrão de programação em Java, aplicando boas práticas de produção de código, como convenções de nomenclatura, organização lógica e modularização de código. Isso assegura clareza e consistência, facilita a manutenção e maximiza a legibilidade. Também adotámos o padrão Javadoc para documentar as classes, métodos e parâmetros, oferecendo uma referência clara e estruturada para futuros desenvolvedores. Como ferramenta de compilação utilizamos Gradle 8.6.

```

/**
 * Creates a new training plan based on the provided specifications.
 *
 * @param userCode The code of the user for whom the training plan will be created.
 * @param startDate The start date of the new training plan.
 * @return The code of the created training plan.
 * @throws IllegalArgumentException If the provided arguments are not valid.
 */
public UUID createTrainingPlan(UUID userCode, MakeItFitDate startDate) throws
IllegalArgumentException {
    TrainingPlan trainingPlan = this.trainingPlanManager.createTrainingPlan(userCode,
startDate);
    this.trainingPlanManager.insertTrainingPlan(trainingPlan);
    return trainingPlan.getCode();
}

```

Para a realização de testes, recorremos à biblioteca JUnit 5. Esta biblioteca fornece ferramentas modernas e flexíveis para a criação de testes unitários, permitindo a definição de conjuntos de testes com verificações precisas para verificar o comportamento esperado das funcionalidades. Além disso, a JUnit 5 suporta a execução automatizada de testes, possibilitando a identificação precoce de problemas durante o desenvolvimento. Segue um exemplo descritivo de testes de inserção de um plano de treino:

```

/**
 * Test the method insertTrainingPlan with a valid training plan
 */
@Test
public void testInsertTrainingPlanAndGetTrainingPlan() {
    UUID code = UUID.randomUUID();
    TrainingPlan trainingPlan = new TrainingPlan(UUID.randomUUID(), MakeItFitDate.of(2024,
4, 4));

    TrainingPlanManager manager = new TrainingPlanManager();
    manager.insertTrainingPlan(trainingPlan);

    TrainingPlan retrievedPlan = manager.getTrainingPlan(code);

    assertNotNull(retrievedPlan);
    assertEquals(trainingPlan, retrievedPlan);
}

/**
 * Test the method insertTrainingPlan with a null training plan
 */
@Test
public void testInsertTrainingPlanWithNullTrainingPlan() {
    TrainingPlanManager manager = new TrainingPlanManager();
    assertThrows(IllegalArgumentException.class, () -> {
        manager.insertTrainingPlan(null);
    });
}

```

Na aplicação, pretendemos implementar a estratégia de agregação como parte do encapsulamento previsto em Programação Orientada aos Objetos (POO). A agregação é uma relação entre objetos em que um objeto contém ou possui referências a outros objetos, mas a vida desses objetos agregados não depende do objeto que os contém.

Esta estratégia permite criar estruturas mais complexas a partir de objetos mais simples, promovendo a reutilização de código e a flexibilidade no design do sistema. Ao encapsular funcionalidades específicas em diferentes classes, agregamos essas classes em entidades maiores para oferecer funcionalidades mais amplas.

Por exemplo, ao modelar componentes importantes da aplicação, como utilizadores, utilizamos a agregação para reunir características e comportamentos distintos em classes separadas. Em seguida, essas classes foram agregadas numa classe maior que as administra e as coordena.

A aplicação desta estratégia não só promove o encapsulamento, permitindo que cada classe se concentre nas suas responsabilidades específicas, como também facilita a manutenção e a evolução do código. Isto porque as mudanças podiam ser feitas de forma isolada, sem impactar outras partes da aplicação.

### 1.3. Design patterns

O grupo avançou com a projeção da aplicação ao estudar design patterns para identificar quais os padrões de desenvolvimento mais adequados aos requisitos. Um padrão que se destacou foi o **Abstract Factory**, que permite a criação de famílias de objetos relacionados ou dependentes sem definir explicitamente suas classes concretas. No contexto das atividades e dos utilizadores, este padrão é crucial para garantir a implementação correta de famílias de atividades e as suas interações, como ilustrado na Figura 1 e na Figura 2. O **Abstract Factory** possibilita a criação de objetos específicos para diferentes tipos de atividades, mantendo uma interface consistente e permitindo a extensão fácil do sistema para novos tipos de atividades e utilizadores.

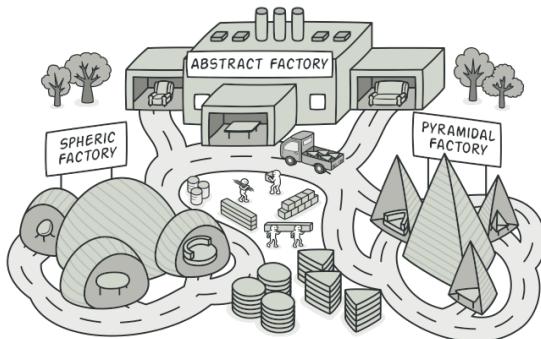


Figura 1: Exemplo de representação gráfica do padrão Abstract Factory. [1]

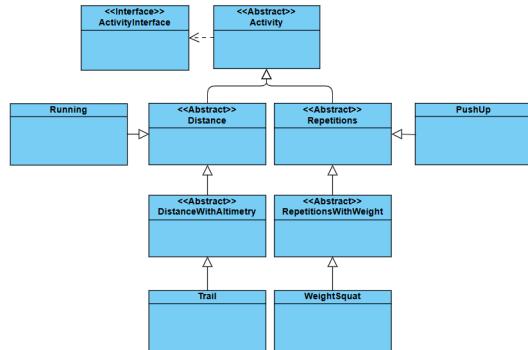


Figura 2: Exemplo de representação conceitual do padrão Abstract Factory na aplicação.

Durante a investigação sobre padrões de design, identificámos o padrão **Facade** como especialmente relevante. Este padrão permite criar uma única interface de acesso para um conjunto de funcionalidades relacionadas, centralizando as operações e ocultando a complexidade interna das classes subjacentes. Ao encapsular as interações com subsistemas, o **Facade** simplifica a utilização da aplicação e garante uma maior modularidade e coesão, ao mesmo tempo em que previne o acesso direto ao conhecimento interno das classes, proporcionando assim uma camada de abstração adicional para o utilizador. Observe-se na Figura 3 e Figura 4 as representações gráfica e conceitual, respetivamente.

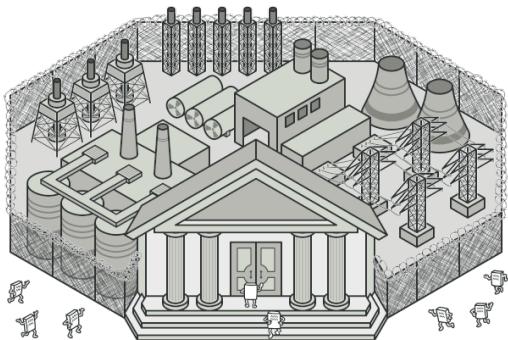


Figura 3: Exemplo de representação gráfica do padrão Facade. [2]

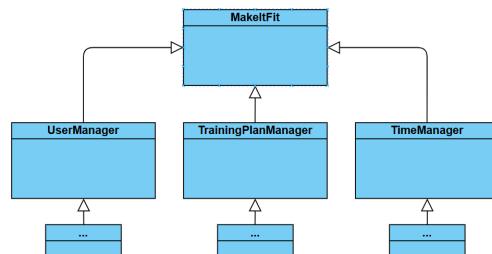


Figura 4: Exemplo de representação conceitual do padrão Facade na aplicação.

Finalmente surgiu o padrão **Observer** que é um padrão comportamental que estabelece uma dependência entre objetos, de forma que quando um objeto muda de estado, todos os objetos dependentes dele são notificados e atualizados automaticamente. Este padrão é especialmente útil em casos como o da mudança da data na aplicação, permitindo uma arquitetura mais flexível e desacoplada, como mostrado na Figura 5 e Figura 6.

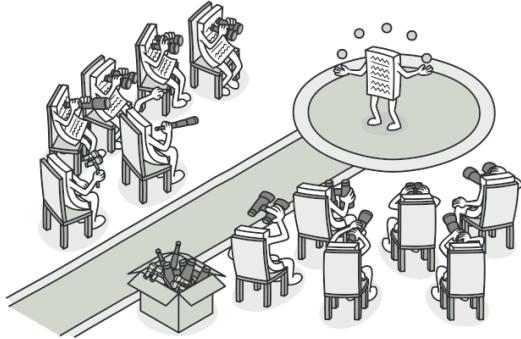


Figura 5: Exemplo de representação gráfica do padrão Observer. [3]

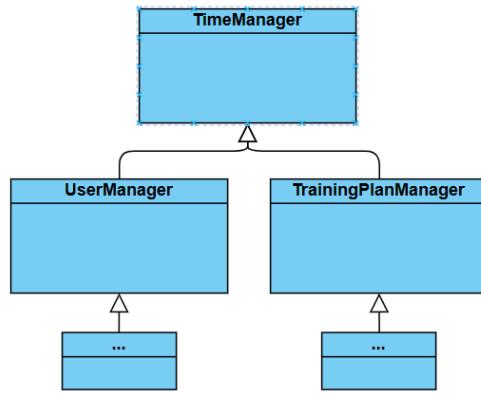


Figura 6: Exemplo de representação conceitual do padrão Observer na aplicação.

## 2. Diagrama de classes

Nesta secção do relatório, será apresentada a construção do sistema com base no diagrama de classes. A ferramenta “Visual Paradigm” [4] foi utilizada para modelar o sistema, empregando a notação UML (Unified Modeling Language) para a representação gráfica das classes e as suas relações.

O diagrama de classes é uma parte crucial do design do sistema, pois oferece uma visão estrutural dos componentes da aplicação. Descreve as classes que compõem o sistema, os seus atributos e métodos, além de ilustrar as relações entre as classes, como herança, associação e dependência.

Ao apresentar o diagrama de classes, esta secção fornecerá uma visão abrangente da estrutura do sistema, destacando as classes principais, as suas responsabilidades, atributos e métodos, bem como as interações entre elas. Isso garantirá uma base sólida para entender a arquitetura do sistema e como ele foi projetado para atender aos requisitos estabelecidos inicialmente.

### 2.1. Modelação das Atividades

Começou-se por modelar as atividades de acordo com o padrão Abstract Factory anteriormente apresentado. Nesse contexto, criou-se uma interface denominada “ActivityInterface”, que obriga à implementação de métodos essenciais ao bom funcionamento da aplicação, como calculateCaloricWaste().

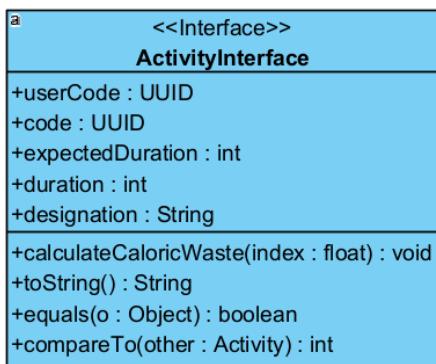


Figura 7: Representação da interface “ActivityInterface” no diagrama de classes.

Em seguida, estabeleceu-se a classe abstrata “Activity”, uma vez que o método mencionado anteriormente deve ser implementado de forma específica por cada tipo de atividade. Assim, ele é declarado como *abstract* na classe “Activity”, tornando-a também abstrata.

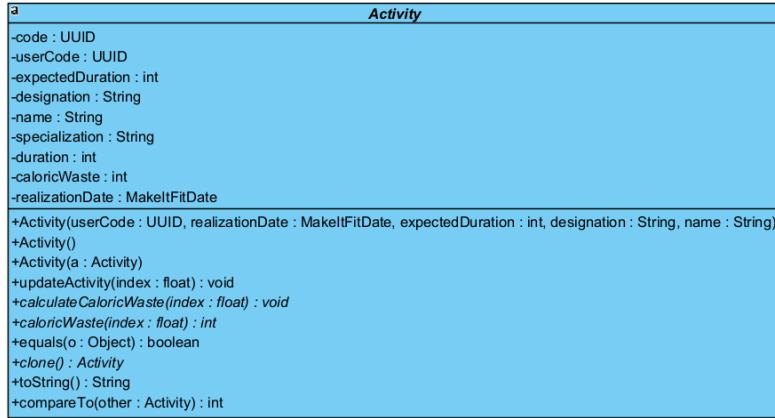


Figura 8: Representação da classe abstrata “Activity” no diagrama de classes.

A seguir, foram criadas subclasses de “Activity” que representam os tipos previstos nos requisitos: “Distance” (*extends “Activity”*) e “Repetitions” (*extends “Activity”*). Além disso, existem mais duas classes derivadas destas: “DistanceWithAltimetry” (*extends “Distance”*) e “RepetitionsWithWeight” (*extends “Repetitions”*).

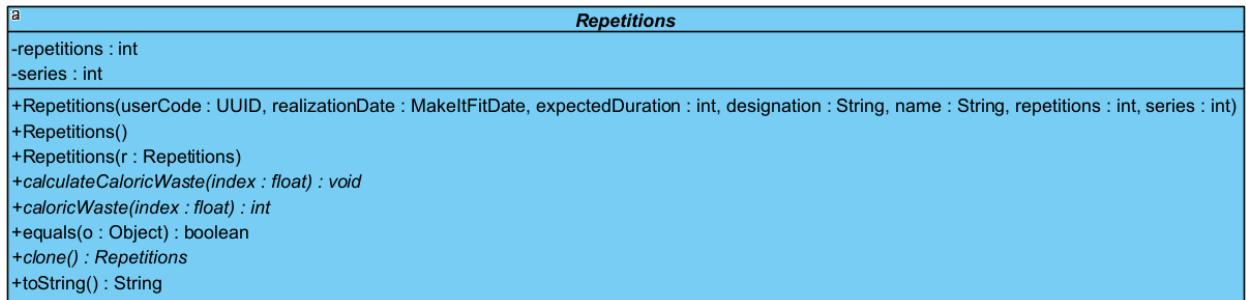


Figura 9: Representação da classe abstrata “Repetitions” no diagrama de classes.

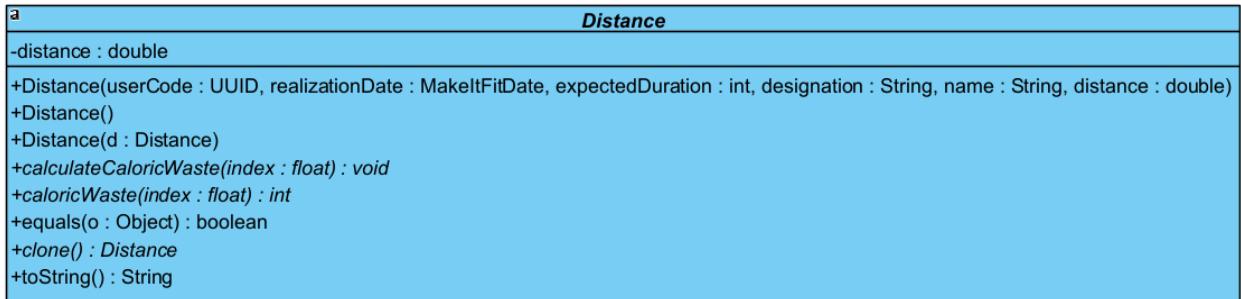


Figura 10: Representação da classe abstrata “Distance” no diagrama de classes.

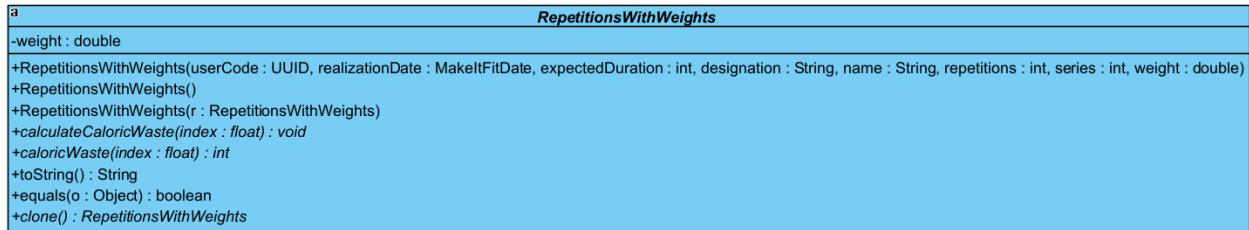


Figura 11: Representação da classe abstrata “RepetitionsWithWeights” no diagrama de classes.

```

a DistanceWithAltimetry
-elevationGain : double
-elevationLoss : double
+DistanceWithAltimetry(userCode : UUID, realizationDate : MakeltFitDate, expectedDuration : int, designation : String, name : String, distance : double, elevationGain : double, elevationLoss : double)
+DistanceWithAltimetry()
+DistanceWithAltimetry(a : DistanceWithAltimetry)
+calculateCaloricWaste(index : float) : void
+caloricWaste(index : float) : int
>equals(o : Object) : boolean
+clone() : DistanceWithAltimetry
+toString() : String

```

Figura 12: Representação da classe abstrata “DistanceWithAltimetry” no diagrama de classes.

Todas estas classes são abstratas, pois não implementam o método calculateCaloricWaste() diretamente, passando essa responsabilidade para as suas subclasses, como “PushUp” (*extends “Repetitions”*), “WeightSquat” (*extends “RepetitionsWithWeight”*), “Running” (*extends “Distance”*) e “Trail” (*extends “DistanceWithAltimetry”*). Estas últimas implementam o método conforme necessário, herdando características e métodos das suas superclasses e cumprindo a obrigatoriedade de implementar os métodos específicos.

```

a PushUp
+PushUp(userCode : UUID, realizationDate : MakeltFitDate, expectedDuration : int, designation : String, name : String, repetitions : int, series : int)
+PushUp()
+PushUp(p : PushUp)
+calculateCaloricWaste(index : float) : void
+caloricWaste(index : float) : int
+toString() : String
>equals(o : Object) : boolean
+clone() : PushUp

```

Figura 13: Representação da classe concreta “PushUp” no diagrama de classes.

```

a Running
-averageSpeed : double
+Running(userCode : UUID, realizationDate : MakeltFitDate, expectedDuration : int, designation : String, name : String, distance : double, speed : double)
+Running()
+Running(r : Running)
+getSpeed() : double
+setSpeed(speed : double) : void
+calculateCaloricWaste(index : float) : void
+caloricWaste(index : float) : int
+toString() : String
>equals(o : Object) : boolean
+clone() : Running

```

Figura 14: Representação da classe concreta “Running” no diagrama de classes.

```

a WeightSquat
+WeightSquat(userCode : UUID, realizationDate : MakeltFitDate, expectedDuration : int, designation : String, name : String, repetitions : int, series : int, weight : double)
+WeightSquat()
+WeightSquat(w : WeightSquat)
+calculateCaloricWaste(index : float) : void
+caloricWaste(index : float) : int
+toString() : String
>equals(o : Object) : boolean
+clone() : WeightSquat

```

Figura 15: Representação da classe concreta “WeightSquat” no diagrama de classes.

```

a Trail
+TRAIL_TYPE_EASY : int = 0
+TRAIL_TYPE_MEDIUM : int = 1
+TRAIL_TYPE_HARD : int = 2
-trailType : int
+Trail(userCode : UUID, realizationDate : MakeltFitDate, expectedDuration : int, designation : String, name : String, distance : double, elevationGain : double, elevationLoss : double, trailType : int)
+Trail()
+Trail(trail : Trail)
+calculateCaloricWaste(index : float) : void
+caloricWaste(index : float) : int
+toString() : String
>equals(obj : Object) : boolean
+clone() : Trail

```

Figura 16: Representação da classe concreta “Trail” no diagrama de classes.

Adicionalmente, define-se ainda a interface “Hard” para servir como identificador e colocar nas atividades o fator de serem difíceis ou não, permitindo facilidade de caracterização das atividades.

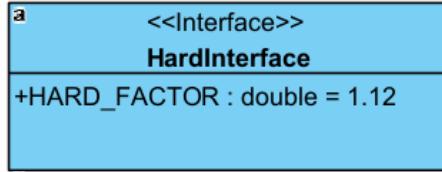


Figura 17: Representação da interface “Hard” no diagrama de classes.

## 2.2. Modelação dos Utilizadores

Na modelação dos utilizadores, foi adotado o padrão Abstract Factory, criando uma estrutura consistente para a gestão de diferentes tipos de utilizadores na aplicação. O ponto de partida foi a definição da interface “UserInterface”, que impõe a implementação de métodos essenciais para o correto funcionamento da aplicação.

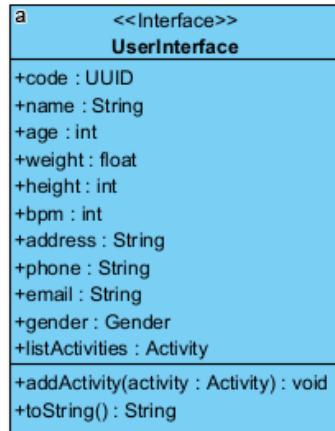


Figura 18: Representação da interface “UserInterface” no diagrama de classes.

Em seguida, estabeleceu-se a classe abstrata “User”, responsável por implementar métodos comuns a todos os tipos de utilizadores e servir como base para as subclasses específicas. A classe “User” não contém métodos abstratos que devem ser implementados pelas subclasses mas previne que não hajam instâncias de “User” e sim dos seu tipos, de acordo com suas particularidades.

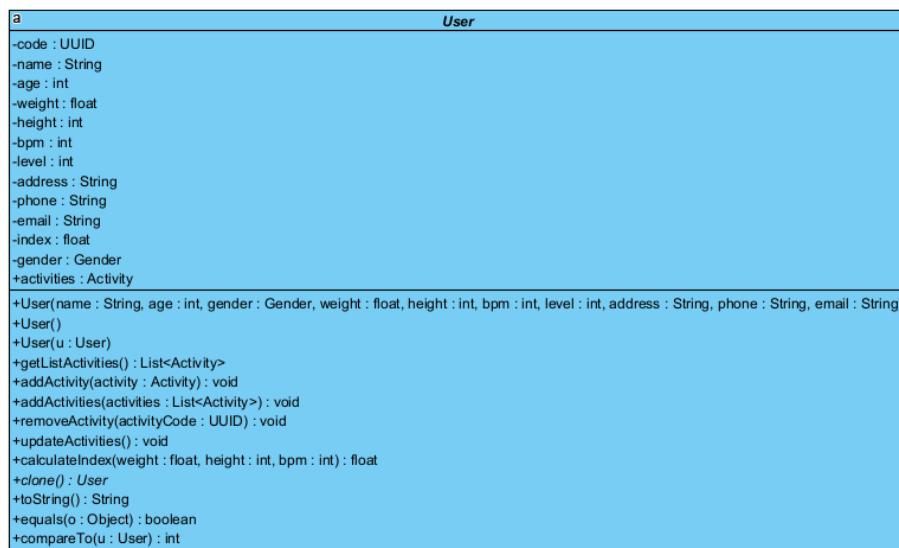


Figura 19: Representação da classe abstrata “User” no diagrama de classes.

A partir dessa base, foram criadas três subclasses que representam os tipos de utilizadores previstos nos requisitos: “Amateur” (*extends “User”*), “Occasional” (*extends “User”*) e “Professional” (*extends “User”*). Cada uma dessas subclasses determina certas características que os distinguem entre si.

a	<b>Amateur</b>
+Amateur(name : String, age : int, gender : Gender, weight : float, height : int, bpm : int, level : int, address : String, phone : String, email : String)	
+Amateur(amateur : Amateur)	
+clone() : Amateur	
+toString() : String	

Figura 20: Representação da classe concreta “Amateur” no diagrama de classes.

a	<b>Occasional</b>
-frequency : int	
+Occasional(name : String, age : int, gender : Gender, weight : float, height : int, bpm : int, level : int, address : String, phone : String, email : String, frequency : int)	
+Occasional(o : Occasional)	
+clone() : Occasional	
+toString() : String	

Figura 21: Representação da classe concreta “Occasional” no diagrama de classes.

a	<b>Professional</b>
-specialization : String	
-frequency : int	
+Professional(name : String, age : int, gender : Gender, weight : float, height : int, bpm : int, level : int, address : String, phone : String, email : String, frequency : int)	
+Professional(p : Professional)	
+updateSpecialization() : void	
+clone() : Professional	
+toString() : String	

Figura 22: Representação da classe concreta “Professional” no diagrama de classes.

Além disso, a aplicação possui um gestor de utilizadores, denominado “UserManager”. Este gestor é responsável pela administração de todos os utilizadores na aplicação, mantendo um *Map* que armazena os diferentes utilizadores registados, permitindo a fácil recuperação e gestão dos mesmos.

a	<b>UserManager</b>
-usersByCode : Map<UUID, User>	
-usersByEmail : Map<String, User>	
+UserManager()	
+createUser(name : String, age : int, gender : Gender, weight : float, height : int, bpm : int, level : int, address : String, phone : String, email : String, frequency : int, type : String) : User	
+insertUser(user : User) : void	
+removeUserByCode(code : UUID) : void	
+removeUserByEmail(email : String) : void	
+existsUserWithEmail(email : String) : boolean	
+getUserByCode(code : UUID) : User	
+getUserByEmail(email : String) : User	
+updateUser(user : User) : void	
+getAllUsers() : List<User>	
+getActivitiesFromUser(email : String) : List<Activity>	
+addActivityToUser(email : String, activity : Activity) : void	
+removeActivityFromUser(email : String, activityCode : UUID) : void	
+addActivitiesToUser(userCode : UUID, activities : List<Activity>) : void	
+updateSystem() : void	

Figura 23: Representação da classe concreta “UserManager” no diagrama de classes.

## 2.3. Modelação dos Planos de Treino

Na modelação dos planos de treino, foi adotada uma estrutura simples e eficaz para a gestão e implementação dos mesmos na aplicação. A classe concreta “TrainingPlan” representa um plano de treino específico e contém os dados e métodos necessários para definir e manipular um plano de treino, como a lista de atividades, data de início, e código do utilizador que tem o plano de treino.

a	TrainingPlan
<<Property>>	-userCode : UUID
<<Property>>	-code : UUID
<<Property>>	-activities : MyTuple<Integer, Activity>
<<Property>>	-startDate : MakeltFitDate
+TrainingPlan	(userCode : UUID, startDate : MakeltFitDate)
+TrainingPlan	()
+TrainingPlan	(tp : TrainingPlan)
+addActivity	(repetitions : int, activity : Activity) : void
+removeActivity	(code : UUID) : void
+updateActivities	(currentDate : MakeltFitDate, index : float) : void
+extractActivities	(currentDate : MakeltFitDate) : List<Activity>
+toString	() : String
+equals	(o : Object) : boolean
+compareTo	(other : TrainingPlan) : int

Figura 24: Representação da classe concreta “TrainingPlan” no diagrama de classes.

Além disso, a aplicação deverá incluir um gestor específico para planos de treino, denominado “TrainingPlanManager”. Este gestor é responsável pela criação, armazenamento e gestão dos planos de treino na aplicação. Mantém um *Map* que armazena os diferentes planos de treino. Também lida com a geração de planos de treino personalizados com base nas preferências e características do utilizador. Isso inclui a seleção das atividades mais adequadas, a programação de sessões de treino e a definição de metas alcançáveis para cada utilizador.

a	TrainingPlanManager
-trainingPlans	: Map<UUID, TrainingPlan>
+TrainingPlanManager	()
+createTrainingPlan	(userCode : UUID, startDate : MakeltFitDate) : TrainingPlan
+constructTrainingPlanByObjectives	(trainingPlan : TrainingPlan, index : float, hardActivities : boolean, maxActivitiesPerDay : int, maxDifferentActivities : int, weeklyRecurrence : int, minimumCaloricWaste : int) : TrainingPlan
+insertTrainingPlan	(trainingPlan : TrainingPlan) : void
+removeTrainingPlan	(code : UUID) : void
+getTrainingPlan	(code : UUID) : TrainingPlan
+updateTrainingPlan	(trainingPlan : TrainingPlan) : void
+getAllTrainingPlans	() : List<TrainingPlan>
+addActivity	(code : UUID, repetitions : int, activity : Activity) : void
+removeActivity	(code : UUID, activity : UUID) : void
+getTrainingPlansFromUser	(userCode : UUID) : List<TrainingPlan>
+updateActivities	(currentDate : MakeltFitDate, index : float) : void
+extractActivities	(currentDate : MakeltFitDate, userCode : UUID) : List<Activity>

Figura 25: Representação da classe concreta “TrainingPlanManager” no diagrama de classes.

## 2.4. Modelação das classes de utilitários

### 2.4.1. Utilitários

Começamos por criar a classe “MakeltFitDate” para acrescentar uma camada de abstração à gestão do tempo na nossa aplicação, desta forma garantimos, que ainda sofrendo alterações externas podemos manter a funcionalidade íntegra.

a	MakeItFitDate
-date : LocalDate	
+MakeItFitDate()	
+of(year : int, month : int, dayOfMonth : int) : MakeItFitDate	
-MakeItFitDate(date : LocalDate)	
+fromString(date : String) : MakeItFitDate	
+getDayOfWeek() : int	
+getDayOfMonth() : int	
+getMonth() : int	
+getYear() : int	
+isBeforeOrSame(date : MakeItFitDate) : boolean	
+isBefore(date : MakeItFitDate) : boolean	
+isAfterOrSame(date : MakeItFitDate) : boolean	
+isAfter(date : MakeItFitDate) : boolean	
+plusDays(days : int) : MakeItFitDate	
+equals(o : Object) : boolean	
+distance(date : MakeItFitDate) : int	
+toString() : String	
+compareTo(object : MakeItFitDate) : int	
+clone(object : MakeItFitDate) : int	

Figura 26: Representação da classe concreta “MakeItFitDate” no diagrama de classes.

De seguida, com a possibilidade de interação com o utilizador, surge a necessidade de validar os endereços de email providenciados.

a	EmailValidator
-EMAIL_PATTERN : String = "^[a-zA-Z0-9.+ -]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}\$"	
+isValidEmail(emailString : String) : boolean	

Figura 27: Representação da classe concreta “EmailValidator” no diagrama de classes.

#### 2.4.2. Tempo

Para uma melhor modularização e gestão do tempo na aplicação, previmos a criação de uma classe específica para o efeito.

a	TimeManager
-currentDate : MakeItFitDate	
+TimeManager()	
+TimeManager(currentDate : MakeItFitDate)	
+advanceTime(days : int) : MakeItFitDate	

Figura 28: Representação da classe concreta “TimeManager” no diagrama de classes.

#### 2.4.3. Menu

Na unidade curricular foi-nos incentivado o uso de menus modulares que permitissem incutir versatilidade e conforto ao projeto, do qual surgiu a sugestão de criar um “Menu”, que controla o comportamento dos “MenuItem”, este que por si só, se associa a duas interfaces para otimização de utilização, a “PreCondition” e “Handler”.

a	Menu
-keepRunning : boolean	
-items : MenuItem	
+Menu(items : List<MenuItem>)	
+display() : void	
+getUserChoice() : int	
+executeSelectedOption() : void	
+run() : void	

Figura 29: Representação da classe concreta “Menu” no diagrama de classes.

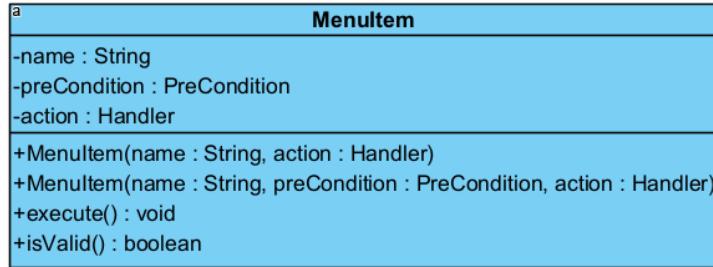


Figura 30: Representação da classe concreta “MenuItem” no diagrama de classes.

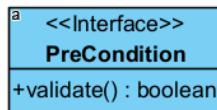


Figura 31: Representação da interface “PreCondition” no diagrama de classes.

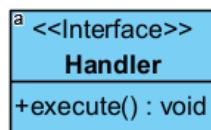


Figura 32: Representação da interface “Handler” no diagrama de classes.

#### 2.4.4. Estatísticas

Construção de um gestor para as estatísticas pedidas no enunciado do projeto.

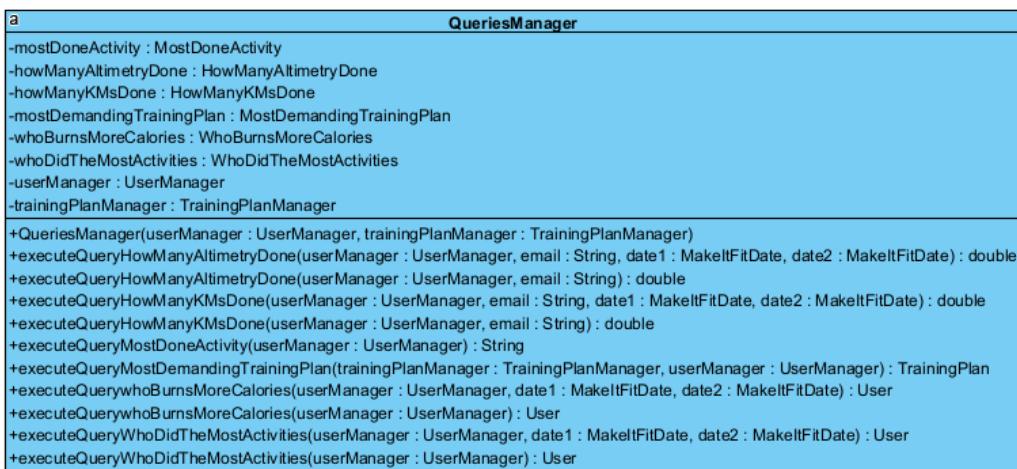


Figura 33: Representação da classe concreta “QueriesManager” no diagrama de classes.

#### 2.4.5. Exceções

Com o seguimento do projeto surgiu o objetivo de especificar algumas exceções já presentes na tecnologia e, como tal, dedicámos algumas classes à sua definição e implementação.

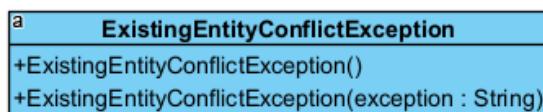


Figura 34: Representação da classe concreta “ExistingEntityConflictException” no diagrama de classes.

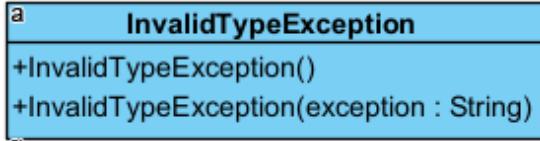


Figura 35: Representação da classe concreta “InvalidTypeException” no diagrama de classes.

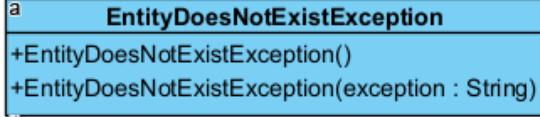


Figura 36: Representação da classe concreta “EntityDoesNotExistException” no diagrama de classes.

## 2.5. Modelação do MVC

Na modelação do MVC (Model-View-Controller), estruturámos a nossa aplicação com base nos três componentes principais: Modelo, Controlador e Vistas. O modelo representa a lógica de dados da aplicação, enquanto o controlador gere as interações entre o utilizador e o sistema, e as vistas apresentam os dados ao utilizador.

O nosso modelo, denominado “MakeltFit”, é responsável por encapsular as funcionalidades principais da aplicação, incluindo a gestão de utilizadores, atividades e planos de treino. O modelo também é responsável pela manipulação de dados e pela aplicação das regras previstas.

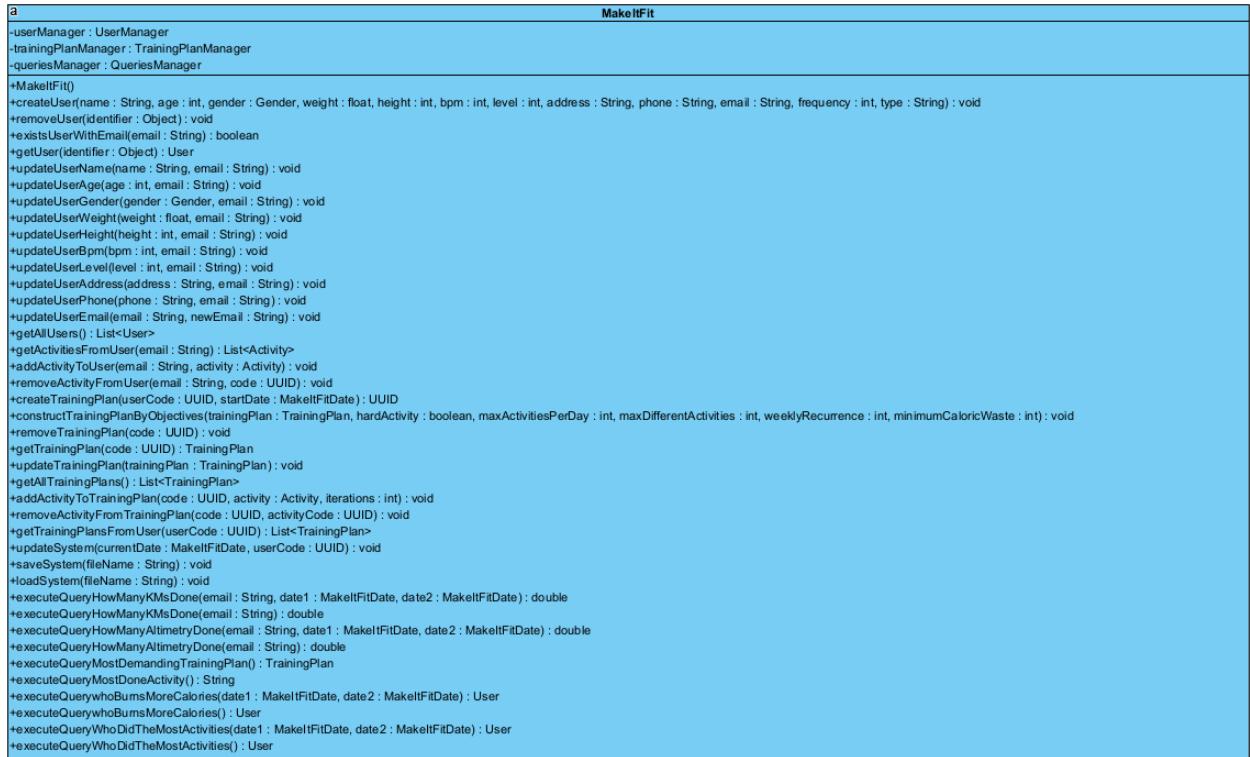


Figura 37: Representação da classe concreta “MakeltFit” no diagrama de classes.

O controlador, denominado “MakeltFitController”, atua como intermediário entre o modelo e as vistas. Ele processa as entradas do utilizador, recebidas através das vistas, e coordena as respostas adequadas, atualizando o modelo conforme necessário e transmitindo os dados atualizados para as vistas.

```

a                                     MakeItFitController
+email : String
-name : String
-trainingPlan : UUID
-makeItFit : MakeItFit
-timeManager : TimeManager
+MakeItFitController()
+login(email : String) : void
+createUser(name : String, age : int, gender : Gender, weight : float, height : int, bpm : int, level : int, address : String, phone : String, frequency : int, type : String) : void
+removeUser() : void
+getUserDetails() : String
+updateName(name : String) : void
+updateAge(age : int) : void
+updateGender(gender : Gender) : void
+updateWeight(weight : float) : void
+updateHeight(height : int) : void
+updateBpm(bpm : int) : void
+updateLevel(level : int) : void
+updateAddress(address : String) : void
+updatePhone(phone : String) : void
+updateEmail(email : String) : void
+getAllUsers() : String
+getActivities() : String
+addActivityToUser(date : MakeItFitDate, duration : int, designation : String, name : String, repetitions : int, series : int) : void
+addActivityToUser(date : MakeItFitDate, duration : int, designation : String, name : String, distance : double, speed : double) : void
+addActivityToUser(date : MakeItFitDate, duration : int, designation : String, name : String, distance : double, elevationGain : double, elevationLoss : double, trailType : int) : void
+addActivityToUser(date : MakeItFitDate, duration : int, designation : String, name : String, repetitions : int, series : int, weight : double) : void
+removeActivityFromUser(activity : UUID) : void
+updateSystemDate(days : int) : void
+createTrainingPlan(startDate : MakeItFitDate) : void
+removeTrainingPlan() : void
+constructTrainingPlanByObjectives(hardActivity : boolean, maxActivitiesPerDay : int, maxDifferentActivities : int, weeklyRecurrence : int, minimumCaloricWaste : int) : void
+addActivityToTrainingPlan(date : MakeItFitDate, duration : int, designation : String, name : String, repetitions : int, series : int, iterations : int) : void
+addActivityToTrainingPlan(date : MakeItFitDate, duration : int, designation : String, name : String, distance : double, speed : double, iterations : int) : void
+addActivityToTrainingPlan(date : MakeItFitDate, duration : int, designation : String, name : String, distance : double, elevationGain : double, elevationLoss : double, trailType : int, iterations : int) : void
+addActivityToTrainingPlan(date : MakeItFitDate, duration : int, designation : String, name : String, repetitions : int, series : int, weight : double, iterations : int) : void
+removeActivityFromTrainingPlan(activity : UUID) : void
+getTrainingPlansFromUser() : String
+getTrainingPlans() : String
+saveSystem(fileName : String) : void
+loadSystem(fileName : String) : void
+executeQueryHowManyKMsDone(date1 : MakeItFitDate, date2 : MakeItFitDate) : double
+executeQueryHowManyKMsDone() : double
+executeQueryHowManyAltmetryDone(date1 : MakeItFitDate, date2 : MakeItFitDate) : double
+executeQueryHowManyAltmetryDone() : double
+executeQueryMostDemandTrainingPlan() : String
+executeQueryMostDoneActivity() : String
+executeQueryWhoBurnsMoreCalories(date1 : MakeItFitDate, date2 : MakeItFitDate) : String
+executeQueryWhoBurnsMoreCalories() : String
+executeQueryWhoDidTheMostActivities(date1 : MakeItFitDate, date2 : MakeItFitDate) : String
+executeQueryWhoDidTheMostActivities() : String

```

Figura 38: Representação da classe concreta “MakeItFitController” no diagrama de classes.

Quanto às vistas, criámos duas subclasses da “MakeItFitView”: “UserView” e “AdminView”. A “UserView” é voltada para a interação com utilizadores comuns, fornecendo uma interface para realizar atividades, visualizar planos de treino e acompanhar o progresso. A “AdminView” é destinada aos administradores da aplicação, permitindo-lhes gerir utilizadores, atividades e planos de treino de uma maneira mais abrangente.

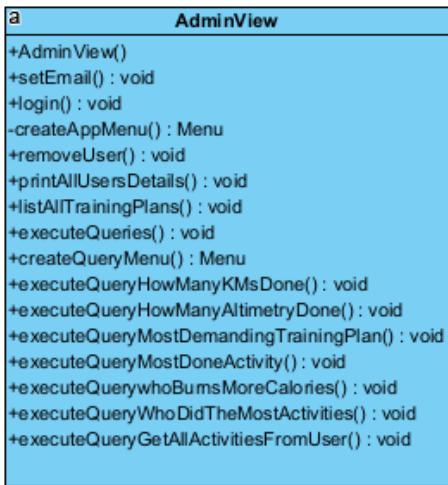


Figura 39: Representação da classe concreta “Admin-View” no diagrama de classes.

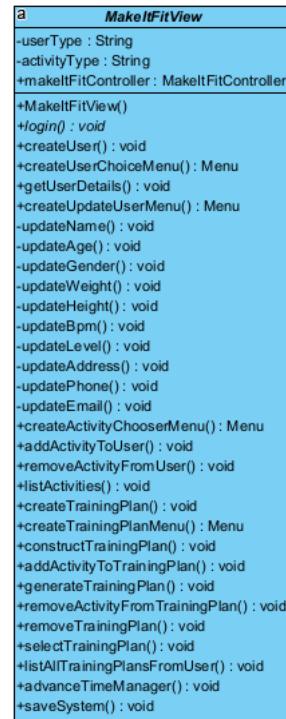


Figura 40: Representação da classe abstrata “Makel-FitView” no diagrama de classes.

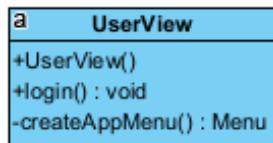


Figura 41: Representação da classe concreta “User-View” no diagrama de classes.

## 2.6. Modelação final

Finalmente, foram estabelecidas todas as relações entre as várias representações apresentadas, algumas que, com o avançar do projeto, sofreram pequenas alterações. Como o diagrama de classes final não é apresentável no espaço reservado para este ficheiro, está disponibilizado como anexo externo na mesma diretoria do relatório atual.

## 3. Implementação

Nesta secção do relatório, será apresentada a implementação, onde é representado o culminar de esforços meticolosos e de um planeamento estratégico. Este relatório oferece uma visão detalhada dos processos, estratégias e resultados alcançados durante a fase de execução. Vamos explorar os objetivos definidos, as soluções desenvolvidas para ultrapassar desafios, proporcionando uma avaliação abrangente do sucesso da implementação.

### 3.1. Contextos base da aplicação

Utilizando o padrão Abstract Factory, implementamos as entidades fundamentais da nossa aplicação, as atividades, os utilizadores e, por último, os planos de treino. Optamos por organizar por packages, de forma a manter a estrutura clara e organizada. As atividades estão agrupadas num package separado, com diferentes implementações e tipos de atividades, desta forma é possível garantir que a responsabilidade de calcular as calorias fica a cargo de cada subclasse, permitindo assim personalização no cálculo calórico de cada atividade.

Da mesma forma, os utilizadores seguem uma estrutura semelhante, com um package para os diferentes tipos de utilizadores, facilitando a necessidade de ter obrigatoriamente de um certo tipo de utilizador. De forma adicional, os utilizadores com o tipo profissional adquirem uma variável de instância exclusiva, denominada especialização, que indica o tipo de atividade mais praticada pelo utilizador, que é atualizada automaticamente de acordo com a atividade mais realizada pelo utilizador.

Quanto aos planos de treino, estão integralmente contidos dentro de um único package, facilitando sua gestão e manutenção.

### 3.2. Saltos temporais no sistema

De forma a simular um contexto real de uma aplicação, é possível replicar saltos temporais usando o padrão Observer. Isso permite atualizar dinamicamente todas as informações à medida que o tempo avança. Assim, as calorias gastas em atividades passadas e a inclusão de novas atividades num plano de treino para um utilizador são automaticamente atualizadas à medida que o tempo avança. Com o objetivo de garantir uma implementação o mais modular possível, optamos por abstrair a classe Date, criando a `MakelFitDate`, uma classe personalizada que nos permite ajustar facilmente como o tempo é tratado na nossa aplicação.

### 3.3. Realização de estatísticas sobre o estado do programa

O `QueriesManager` é a classe responsável na implementação da resolução de estatísticas da aplicação. Esta classe acessa a informação dos utilizadores e dos planos de treinos. Assim, sempre que ocorrer uma alteração temporal ou forem adicionadas novas informações, os resultados serão recalculados.

Todas as estatísticas pedidas foram implementadas:

1. Qual é o utilizador que mais calorias despendeu num período ou desde sempre?
2. Qual o utilizador que mais atividades realizou num período ou desde sempre?

3. Qual o tipo de atividade mais realizada?
4. Quantos quilómetros é que um utilizador realizou num período ou desde sempre?
5. Quantos metros de altimetria é que um utilizar totalizou num período ou desde sempre?
6. Qual o plano de treino mais exigente em função do dispêndio de calorias proposto?
7. Listar as atividades de um utilizador.

A resolução de todas as estatísticas segue um procedimento uniforme. Ao recebermos a entrada de dados, o primeiro passo é verificar a sua validade. Após validar os dados, procedemos a uma análise abrangente, percorrendo todas as informações relevantes. É importante destacar que este processo é iterativo. Por fim, ao atualizarmos ou adicionarmos novos dados, reavaliarmos e recalcularmos as estatísticas pertinentes. Esta abordagem garante que os resultados sejam sempre atualizados e precisos.

### **3.4. Criação de atividades “Hard”**

As atividades podem ser consideradas “Hard”, ou seja, com uma dificuldade acrescida, por esse motivo e de forma a respeitar a modularidade decidimos implementar uma interface, denominada “HardInterface”. Posteriormente as atividades que sejam consideradas dessa categoria apenas necessitam de implementar a interface descrita.

### **3.5. Gerar um plano de treino com objetivos**

Como requisito final, o utilizador pode criar um plano de treino baseado em objetivos, entre eles, se deseja atividades “Hard”, o número máximo de atividades por dia, o número máximo de atividades distintas, a recorrência semanal e o principal, o número mínimo de calorias que se pretende obter com o plano de treino em criação. Respeitando os objetivos anteriormente descritos, e restrições impostas por nós, como número máximo de repetições por exercício, número mínimo e máximo dos atributos das atividades entre outras, a nossa aplicação gera um plano de treino personalizado ao utilizador.

No decorrer do desenvolvimento, surgiu a necessidade de escolher atividades para adicionar ao plano de treino de forma aleatória, dado isto, e não conseguindo alcançar uma melhor solução, optámos por construir uma classe separada para que crie facilidade de resolução do problema caso se venha a evidenciar o expediente.

### **3.6. Salvaguarda do estado da aplicação**

Para garantir a continuidade e conveniência para o utilizador, implementámos uma funcionalidade de salvaguarda do estado da aplicação, utilizando técnicas de serialização das instâncias e escrita em binário nos ficheiros. Isto permite que o utilizador retome a sua atividade, mesmo após fechar e reabrir a aplicação. Todos os dados relacionados com os planos de treino, preferências do utilizador e histórico de atividades são armazenados de forma segura e eficiente através desta abordagem. Com esta funcionalidade, proporcionamos uma experiência contínua e sem preocupações para todos os nossos utilizadores.

## 4. Model-View-Controller

Nesta secção, abordamos o conceito de MVC (Model-View-Controller) e a sua aplicação na nossa aplicação. O padrão MVC é uma abordagem amplamente reconhecida que separa a lógica de dados, a apresentação e o controlo de uma aplicação. Esta organização modular não apenas facilita a manutenção e escalabilidade do sistema, mas também melhora a clareza e flexibilidade do código. Aqui, explicamos como o padrão MVC foi implementado na nossa aplicação, destacando as vantagens e desafios associados a essa estrutura.

### 4.1. Conceito do MVC

O padrão de design MVC (Model-View-Controller) é um dos mais utilizados na programação orientada aos objetos para a organização de aplicações, principalmente em casos com interfaces de utilizador. O padrão separa a lógica do programa em três componentes distintos: o Modelo (Model), a Vista (View) e o Controlador (Controller), proporcionando uma estrutura clara e modular para o desenvolvimento de aplicações.

**Modelo (Model):** O modelo representa a camada de dados da aplicação e é responsável pela lógica de dados. Ele define a estrutura dos dados e mantém seu estado, além de gerir operações como armazenamento, recuperação e manipulação dos dados. O modelo não interage diretamente com a vista, o que permite uma maior independência entre as camadas.

**Vista (View):** A vista é responsável pela apresentação dos dados ao utilizador. Ela recebe informações do controlador e exibe os dados do modelo de acordo com os requisitos da interface de utilizador. As vistas devem ser flexíveis para se adaptarem a diferentes estilos de apresentação e dispositivos.

**Controlador (Controller):** O controlador atua como intermediário entre o modelo e a vista. Ele recebe entradas do utilizador através da vista e processa essas entradas, manipulando o modelo de acordo com as ações do utilizador. Em seguida, o controlador atualiza a vista com os dados do modelo atualizados.

Ao utilizar o padrão MVC, conseguimos separar as responsabilidades de cada componente, o que torna o código mais organizado e facilita a manutenção e a escalabilidade da aplicação. Além disso, esta separação permite a reutilização de código, já que as mudanças numa camada não necessariamente impactam as outras.

### 4.2. Implementação do modelo

De acordo com a modelagem do diagrama de classes e a correta implementação do padrão Facade, conseguimos avançar com a implementação do MVC. Centralizando todos os métodos funcionais numa única classe chamada “MakeltFit”, passámos a disponibilizar esses métodos no nosso controlador, o “MakeltFit-Controller”. Este controlador inclui algumas variáveis de instância, como o email e o nome, para aprimorar a distinção entre as informações passadas para as vistas e a manipulação de dados.

Desta forma, a vista não tem acesso direto aos dados, pois, a cada solicitação, o email é associado diretamente aos métodos que posteriormente acionam métodos do modelo (“MakeltFit”). Adicionalmente, as vistas armazenam variáveis de instância, como o tipo de atividade ou utilizador atualmente selecionado, para otimizar o processamento das informações.

Todos os métodos na vista (“MakeltFitView”) são responsáveis por interagir com o utilizador e por enviar as informações corretas para o controlador (“MakeltFitController”), que, por sua vez, as filtra, processa e encaminha ao modelo (“MakeltFit”). Além disso, os menus foram adicionados para oferecer uma orientação sequencial na aplicação e apresentar de forma consistente as opções disponíveis ao utilizador para uso da plataforma.

## 5. Utilização da aplicação

Nesta secção, abordamos a forma de utilização da nossa aplicação, ajudando no processo desde como iniciar, até às funcionalidades presentes e como elas operam.

### 5.1. Funcionamento geral

A aplicação segue uma abordagem sequencial através de menus de linha de comando (CLI), que nos permite navegar pelas restantes funcionalidades da aplicação. Esta estrutura de menus simplifica a interação do utilizador com o sistema, fornecendo uma experiência intuitiva e direcionada. Ao utilizar os menus, os utilizadores podem facilmente explorar e aceder às diversas funcionalidades disponíveis na aplicação. Além disso, a natureza sequencial dos menus promove uma navegação fluida e organizada, facilitando a utilização e compreensão do sistema.

### 5.2. Início da aplicação

Para iniciar a aplicação, é essencial verificar se todas as dependências estão instaladas, incluindo o Gradle, JUnit5 e JavaDoc. Posteriormente, procede-se à compilação da aplicação utilizando o comando específico, conforme indicado no ficheiro README.md que acompanha o projeto. Uma vez compilada com sucesso, a aplicação está pronta para ser executada através do segundo comando fornecido no mesmo ficheiro. Esta abordagem garante que a aplicação seja executada de forma correta e sem problemas relacionados com dependências ausentes ou compilação inadequada.

Comandos para compilar e executar a aplicação, respetivamente:

```
$ ./gradlew build      $ ./gradlew run --console=plain
```

Ao iniciar o programa, somos apresentados com o seguinte menu: A primeira opção permite-nos inicializar a aplicação no modo utilizador ou no modo admin e se queremos começar com um estado já existente ou recomeçar a aplicação, sendo que as diferenças entre estes modos serão desenvolvidas posteriormente. A segunda opção explica as identidades da aplicação e as suas principais características. A terceira opção apresenta os autores do programa e, por último, temos a opção de sair do mesmo.

```
1. Init the application  
2. Help  
3. Authors  
4. Exit  
Select the option:
```

### 5.3. Modo MakeItFitView

Este modo implementa métodos partilhados por ambas as views, user view e admin view.

### 5.3.1. Interagir com utilizador

Para interagir com o utilizador, temos duas opções disponíveis: visualizar as informações do utilizador ou atualizar as suas informações pessoais.

Opções existentes:

1. User Details
2. Update User info

Exemplo de visualizar as informações através do modo utilizador:

```
[Teste] Getting user details ...
== (User details) ==
Code:
c528ab5a-1c87-44b9-8ac9-49fa354cc0b5
Name: Teste
Age: 20
Gender: Male
Weight: 70.00 kg
Height: 180 cm
Bpm: 110
Level: 6
Address: Rua Teste Teste
Phone: 999999999
Email: teste@mail.com
Activities: []
Frequency: 3
=====
```

Exemplo de uma atualização das informações através do modo utilizador:

1. Update Name
2. Update Age
3. Update Gender
4. Update Weight
5. Update Height
6. Update BPM
7. Update Level
8. Update Address
9. Update Phone
10. Update Email
11. Exit

Select the option:

### 5.3.2. Interagir com atividades

Com atividades temos 3 opções de manipulação como adicionar, remover uma atividade e listar as várias atividades que um utilizador tem.

Opções existentes:

3. Add activity
4. Remove activity
5. List activities

Exemplo de como adicionar uma atividade através do modo utilizador:

```
[Teste] Adding activity to user ...
[Teste] Please select the activity (PushUp | WeightSquat | Running | Trail): Trail
[APP] Please enter the following information:
[APP] Date (dd/mm/aaaa): 20/06/2024
[APP] Duration (minutes): 60
[APP] Designation: pedalar
[APP] Distance (meters): 5000
[APP] Elevation gain (meters): 200
[APP] Elevation loss (meters): 100
[APP] Trail type ([0] Easy, [1] Medium, [2] Hard): 1
[Teste] Activity added to user successfully.
```

Exemplo de como listar uma atividade através do modo utilizador:

```
[Teste] Listing activities ...
[Activity:
 Trail, Code: 43eacc53-d486-486c-b2e8-954a4ffb2cbf, Designation: pedalar, Expected
Duration: 60 minutes, Realization Date: 20/06/2024, Caloric Waste: 0 calories, Distance:
5000.0 meters, Elevation Gain: 200.0 meters, Elevation Loss: 100.0 meters, Trail Type: 1]
```

Exemplo de como remover uma atividade através do modo utilizador:

```
[Teste] Removing activity from user ...
[Teste] Please insert the code of the activity: 43eacc53-d486-486c-b2e8-954a4ffb2cbf
[Teste] Activity removed successfully.
```

### 5.3.3. Interagir com planos de treino

Para manipular os planos de treino, dispomos de seis opções: criar um plano de treino, onde é possível gerar ou adicionar várias atividades; remover um plano de treino, com a opção de gerar ou adicionar diversas atividades; adicionar e remover uma atividade específica de um plano de treino; e listar os planos de treino disponíveis.

Opções existentes:

6. Create training plan
7. Remove training plan
8. Add activity to a training plan
9. Remove activity from a training plan
10. List all training plans

Exemplo de uma geração de treino através do modo utilizador:

```
[teste] Creating training plan ...
[teste] Please enter the start date (dd/mm/aaaa): 13/05/2024
[teste] Training plan created successfully.
1. Construct plan
2. Generate plan
3. Exit
Select the option: 2
[teste] Generating training plan ...
[teste] Do you want to add hard activities? (y/n): n
[teste] Please insert the maximum number of activities per day (max 3): 1
[teste] Please insert the maximum number of different activities: 2
[teste] Please insert the weekly recurrence expected (max 7): 2
[teste] Please insert the minimum number of calories to be consumed: 999
[teste] Training plan generated successfully.
```

Exemplo de uma listagem de planos de treinos através do modo utilizador:

```
[teste] Listing all training plans ...
[ == (Training plan details) ==
Training Plan: a2592c28-a639-46e3-8e39-bff4a6bd1dc2
    User Code: dc2a6915-d579-4365-8a36-fe6edb28f7a
    Start Date: 13/05/2024
    Activities(Iterations / Activity):
        [(3, Activity: WeightSquat, Code: ad5b6e1d-e583-4203-bee4-9c73547bc8b9,
Designation: WeightSquat created automatically, Expected Duration: 14 minutes, Realization
Date: 13/05/2024, Caloric Waste: 0 calories, Repetitions: 19, Series: 4, Weight: 52.0
Kg)]]
```

Exemplo de como remover um plano de treino através do modo utilizador:

```
[teste] Removing training plan ...
[teste] Please insert the code of the training plan: a2592c28-a639-46e3-8e39-bff4a6bd1dc2
[teste] Training plan removed successfully.
```

As opções de adicionar uma tarefa e remover uma tarefa são muito semelhantes o que foi referido Secção 5.3.2 com a particularidade de primeiramente é necessário fornecer o código do plano de treino.<sup>11</sup>

#### 5.3.4. Simulação temporal

É possível simular a progressão e o retrocesso do tempo dentro da nossa aplicação. Quando avançamos no tempo, o número de atividades realizadas por um utilizador aumenta, refletindo-se também na atualização das estatísticas do programa.

Opção em questão: Demonstraçao do avanço com o tempo através do modo utilizador:

11. Advance time  
[teste] Advancing time ...  
[teste] Please enter the number of days to advance: 10  
[teste] Time advanced successfully.

Um utilizador que ainda não começou o seu plano de treino antes do avanço de 10 dias.

```
[teste] Getting user details ...  
== (User details) ==  
Code: f9e0837d-00b6-4d37-9647-5c1772a9bd6e  
Name: teste  
Age: 20  
Gender: Male  
Weight: 80.00 kg  
Height: 170 cm  
Bpm: 110  
Level: 6  
Address: rua teste  
Phone: 999888777  
Email: teste@mail.com  
Activities: []  
=====
```

Depois do avanço dos 10 dias:

```
[teste] Getting user details ...  
== (User details) ==  
Code: f9e0837d-00b6-4d37-9647-5c1772a9bd6e  
Name: teste  
Age: 20  
Gender: Male  
Weight: 80.00 kg  
Height: 170 cm  
Bpm: 110  
Level: 6  
Address: rua teste  
Phone: 999888777  
Email: teste@mail.com  
Activities: [Activity: PushUp, Code: 0b0a1cdb-c6b6-4d0a-8c92-adf6241d3456,  
Designation: PushUp created automatically, Expected Duration: 25 minutes, Realization  
Date: 15/05/2024, Caloric Waste: 155 calories, Repetitions: 17, Series: 3, , Activity:  
WeightSquat, Code: 3cbd8278-9e4f-4ea9-8b1e-550aa9603e3c, Designation: WeightSquat created  
automatically, Expected Duration: 37 minutes, Realization Date: 16/05/2024, Caloric Waste:  
1640 calories, Repetitions: 11, Series: 1, Weight: 49.0 Kg  
]  
=====
```

#### 5.3.5. Guardar estado

Após a utilização da aplicação, é sempre possível guardar o estado atual, incluindo todas as informações e registos realizados durante a sessão. Esta informação é armazenada de forma a permitir que, numa próxima utilização da aplicação, o utilizador possa retomar exatamente de onde parou, permitindo-lhe manter o seu progresso e registos ao longo do tempo, mesmo em diferentes sessões de utilização da aplicação.

Opção em questão: Demonstraçao de como guardar o estado através do modo utilizador:

```
12. Save system state
```

```
[teste] Saving system state ...
[teste] Please enter the file name: savefile
[teste] System state saved successfully.
```

### 5.3.6. Terminar aplicação

Assim como em qualquer aplicação convencional, também é disponibilizada uma opção para sair do programa. Ao selecionar esta opção, o utilizador é encaminhado de volta ao menu inicial da aplicação. referido na Secção 5.2

Opção em questão:

Demonstração de como guardar o estado através do modo utilizador:

```
13. Exit
```

```
[teste] Saving system state ...
[teste] Please enter the file name: savefile
[teste] System state saved successfully.
```

## 5.4. Modo utilizador

### 5.4.1. Autenticação

Para um utilizador autenticar-se, é necessário introduzir o seu email. Caso o utilizador não exista, será necessário registar-se, fornecendo alguns dados como tipo de atleta, nome, idade, peso, altura, entre outros. Se o utilizador já existir, apenas será realizado o login.

Exemplo de registar um utilizador:

```
[APP] Please enter your email: teste@mail.com
[APP] The email teste@mail.com does not exist.
[APP] Would you like to create a new user? (y/n): y
[APP] Creating user ...
[APP] Please enter the type of user ( Amateur | Occasional | Professional): Occasional
[APP] Please enter the following information:
[APP] Name: Teste
[APP] Age: 20
[APP] Gender ( Male | Female | Other): Male
[APP] Weight (Kg): 70
[APP] Height (cm): 180
[APP] Bpm: 110
[APP] Level: 6
[APP] Address: Rua Teste Teste
[APP] Phone: 999999999
[APP] Frequency: 3
[Teste] User created successfully.
```

Exemplo de fazer login de um utilizador:

```
[APP] Please enter your email: teste@mail.com
[teste] Logged in successfully.
```

## 5.5. Modo administrador

Agora no modo administrador, estão disponíveis opções adicionais para uma gestão mais eficaz da aplicação. Estas opções foram concebidas para proporcionar ao administrador um maior controlo sobre as funcionalidades e dados da aplicação, permitindo uma gestão mais personalizada e adaptada às necessidades específicas do sistema.

### **5.5.1. Remover utilizador**

O administrador tem a possibilidade de remover um utilizador, sendo necessário fornecer o email associado a esse utilizador.

Opção em questão:

3. Remove user

Demonstração de como guarda um estado:

```
[APP] Removing user ...
[APP] Enter the user's email: teste@mail.com
[APP] User removed successfully.
```

### **5.5.2. Listar todos os utilizadores**

No modo administrador, existe a possibilidade de listar todos os utilizadores registados na aplicação. Esta funcionalidade permite ao administrador ter uma visão geral de todos os utilizadores do sistema, facilitando a gestão e o acompanhamento das contas de utilizador. Além disso, através desta opção, o administrador pode aceder rapidamente às informações de cada utilizador, se necessário, para realizar operações específicas ou identificar padrões de utilização.

```

== (All users) ==
[    == (User details) ==
Code: 3257a805-c949-44c1-b969-cad54a1ccc1a
Name: teste
Age: 20
Gender: Male
Weight: 70.00 kg
Height: 170 cm
Bpm: 100
Level: 4
Address: Rua teste
Phone: 999888777
Email: teste@mail.com
Activities: []
=====
    == (User details) ==
Code: 4321714d-9a23-4603-a984-a1898a03a8d2
Name: teste2
Age: 21
Gender: Female
Weight: 60.00 kg
Height: 160 cm
Bpm: 105
Level: 2
Address: Travessa Teste Teste
Phone: 666555444
Email: teste2@mail.com
Activities: []
Frequency: 3
=====
    == (User details) ==
Code: c57cd8e7-307c-49a2-91ef-8e09fbc22c29
Name: teste3
Age: 22
Gender: Other
Weight: 175.00 kg
Height: 200 cm
Bpm: 150
Level: 7
Address: Avenida teste
Phone: 333222111
Email: teste3@mail.com
Activities: []
Specialization: No specialization
Frequency: 3
=====
]
=====
```

### 5.5.3. Execução de estatísticas

O administrador tem a possibilidade de realizar estatísticas

#### 3. Execute queries

Assim sendo mesmo pode obter um maior conhecimento sobre a sua aplicação. A análise estatística permite ao administrador extrair informações valiosas, sobre o desempenho, utilização e tendências da aplicação. Essa compreensão mais profunda da aplicação pode ajudar o administrador a tomar decisões infor-

madas e aprimorar continuamente a experiência do utilizador, bem como otimizar o funcionamento geral da aplicação.

Opção em questão:

```
[teste3] Executing queries ...
[APP] Please select the query to execute:
1. How many kms has a user has run in a
period or ever since?
2. How many meters of altimetry has a user
totaled in a period or ever since?
3. What is the most demanding training
plan depending on the proposed calorie
expenditure?
4. what type of activity is most carried
out?
5. Which user has burned the most calories
over a period of time or ever?
6. Which user has carried out the most
activities in a period or ever since?
7. List all activities from a user.
8. Exit
```

Demonstração de da estatística “How many kms has a user has run in a period or ever since?”:

```
[teste3] Executing query ...
[APP] Enter the user's email:
teste@mail.com
[APP] Do you choose a period of time? (y/
n): y
[APP] Enter the start date (dd/mm/aaaa):
11/05/2024
[APP] Enter the end date (dd/mm/aaaa):
20/05/2024
Result: 9.76 kilometers
```

## **6. Conclusão**

Em suma, foi desenvolvida uma aplicação sólida e abrangente para a gestão de atividades físicas e planos de treino. Esta permite o registo e acompanhamento de diversos tipos de atividades, desde corrida até exercícios de musculação, calculando o dispêndio calórico com base no perfil do utilizador. Inclui funcionalidades essenciais, como a criação de utilizadores e atividades, o registo de execuções, a geração automática de planos personalizados e a obtenção de estatísticas detalhadas.

Foram implementados mecanismos para simular o avanço do tempo, permitindo a execução automática de atividades agendadas e a atualização dos recordes dos utilizadores. A aplicação também contempla a noção de atividades intensas (“Hard”), com restrições específicas na geração de planos de treino. Todos os requisitos propostos foram cumpridos, com a adoção de boas práticas de programação orientada a objetos, tratamento robusto de erros e persistência do estado da aplicação em ficheiro. O resultado final é uma ferramenta completa e versátil para a gestão e acompanhamento de rotinas de atividades físicas.

## Bibliografia

- [1] «Abstrat Factory Design Pattern». Acedido: 4 de abril de 2024. [Em linha]. Disponível em: <https://refactoring.guru/design-patterns/abstract-factory>
- [2] «Facade Design Pattern». Acedido: 4 de abril de 2024. [Em linha]. Disponível em: <https://refactoring.guru/design-patterns/facade>
- [3] «Observer Design Pattern». Acedido: 5 de abril de 2024. [Em linha]. Disponível em: <https://refactoring.guru/design-patterns/observer>
- [4] «Visual Paradigm Documentation». Acedido: 16 de abril de 2024. [Em linha]. Disponível em: <https://www.visual-paradigm.com/support/documents/>

## **Lista de Siglas e Acrónimos**

**POO** Programação Orientada aos Objetos

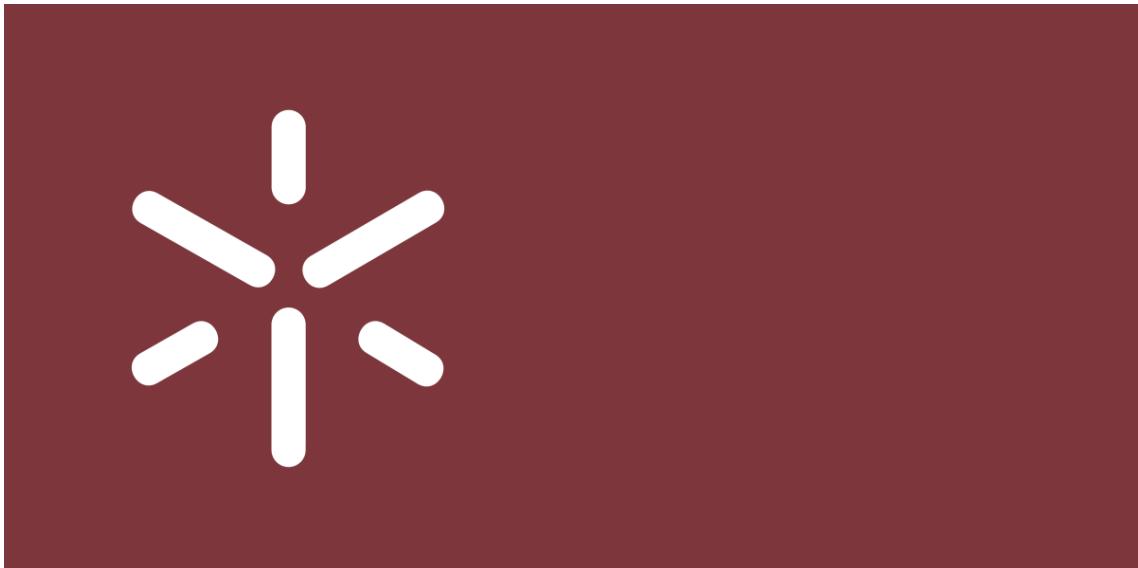
**MVC** Model-View-Controller

**UML** Unified Modeling Language

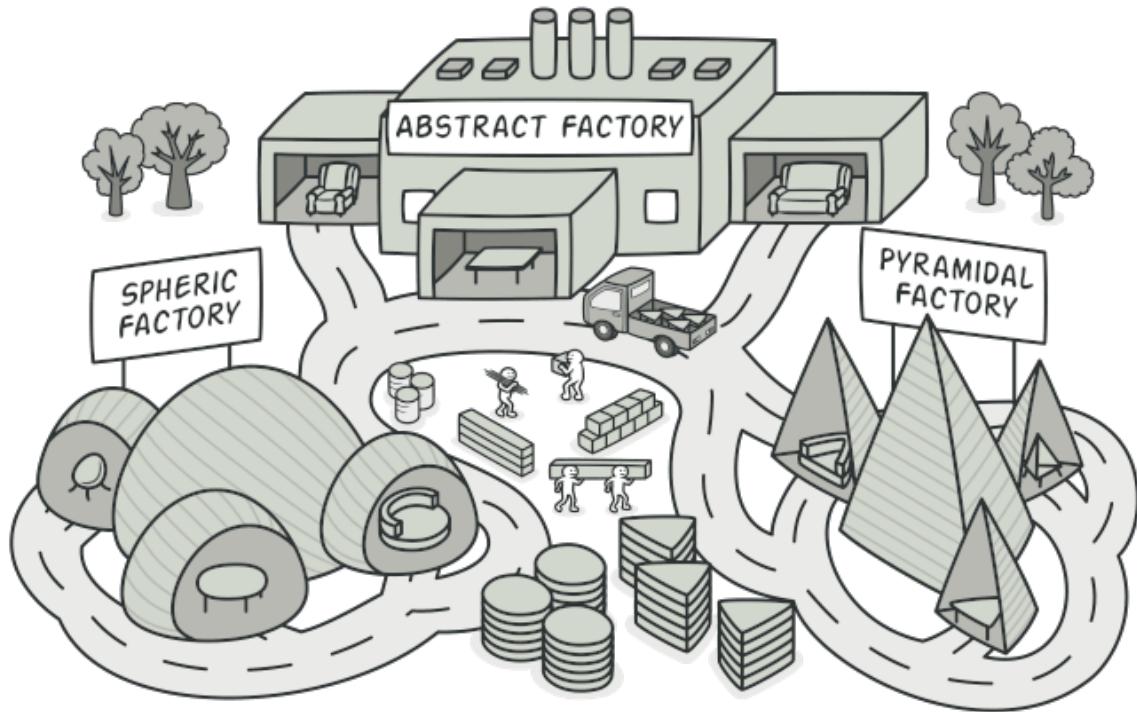
**CLI** Command Line Interface

## **Anexos**

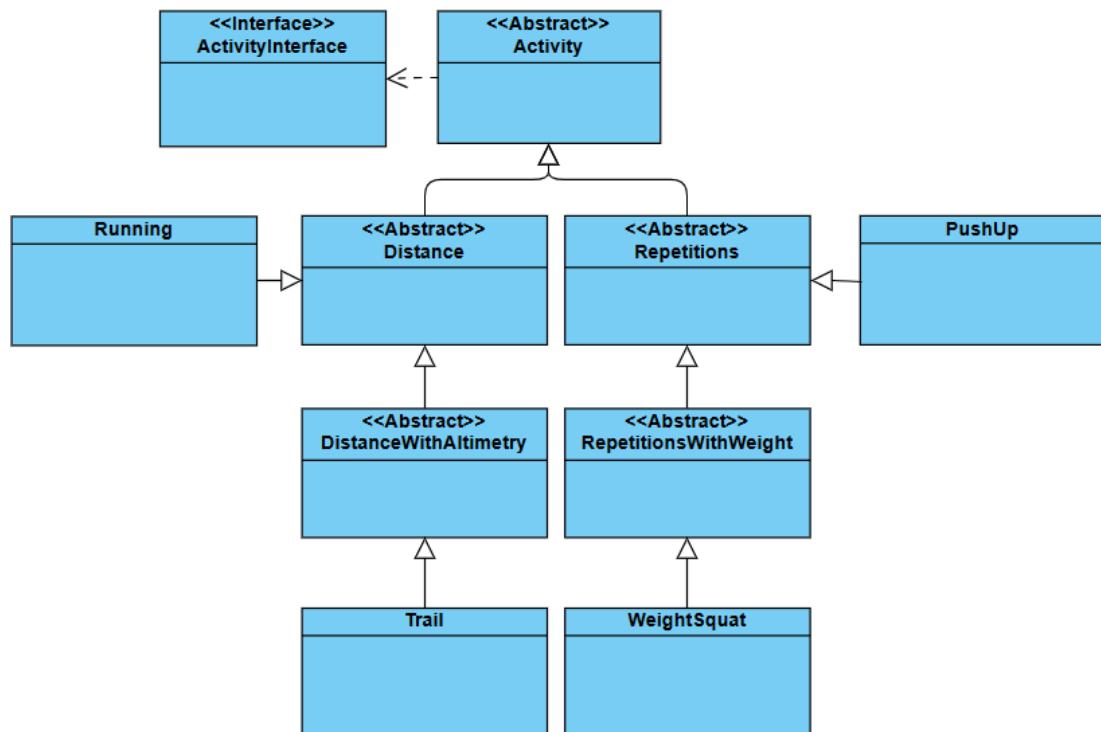
**Anexo 1: Logo da Universidade do Minho.**



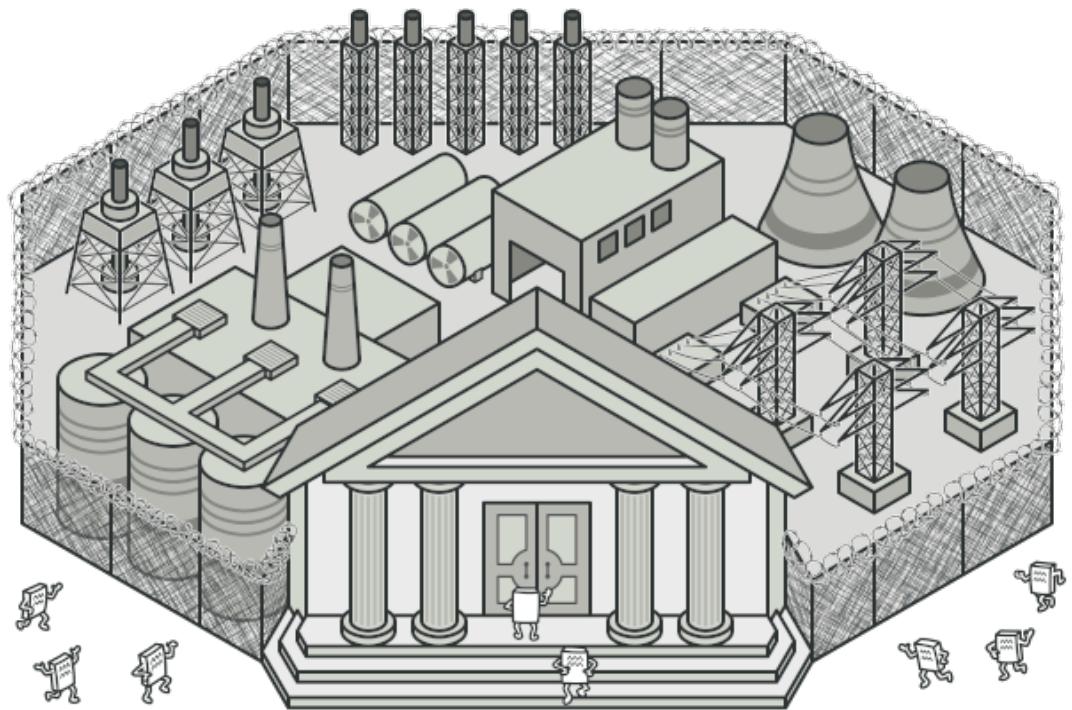
**Anexo 2: Exemplo de representação gráfica do padrão Abstract Factory.**



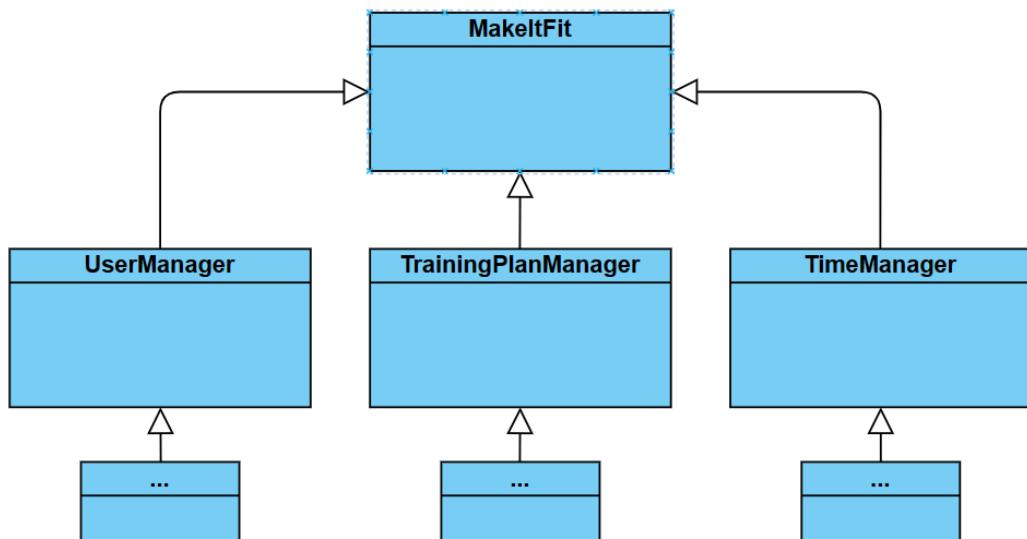
**Anexo 3: Exemplo de representação conceitual do padrão Abstract Factory na aplicação.**



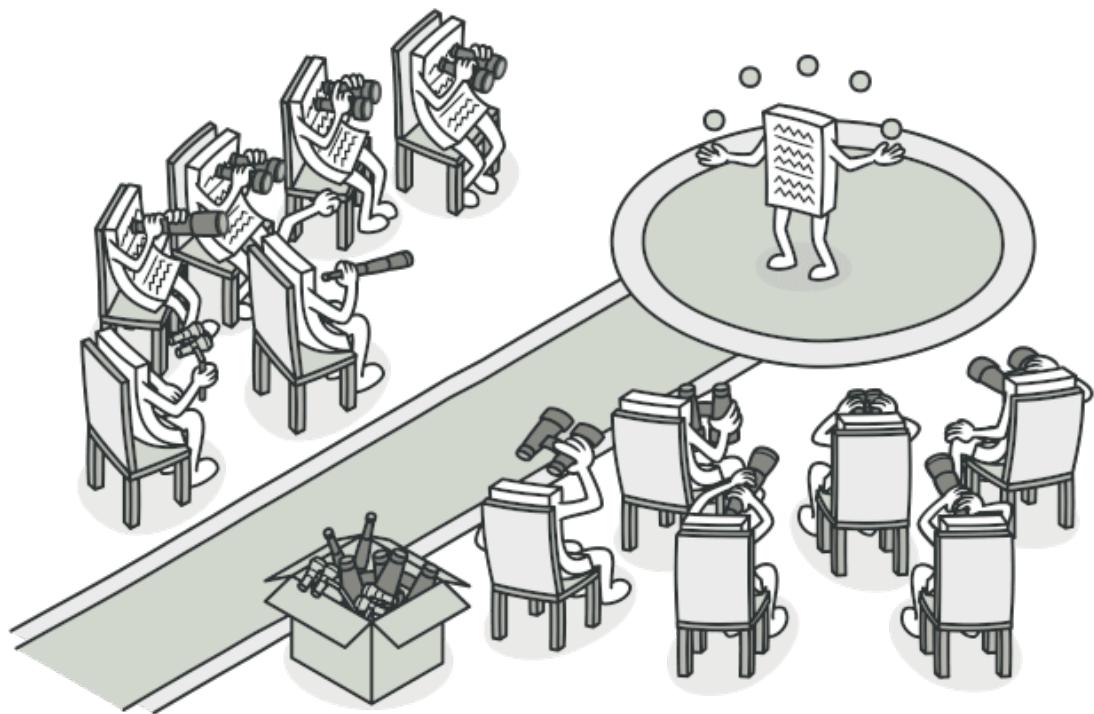
**Anexo 4: Exemplo de representação gráfica do padrão Facade.**



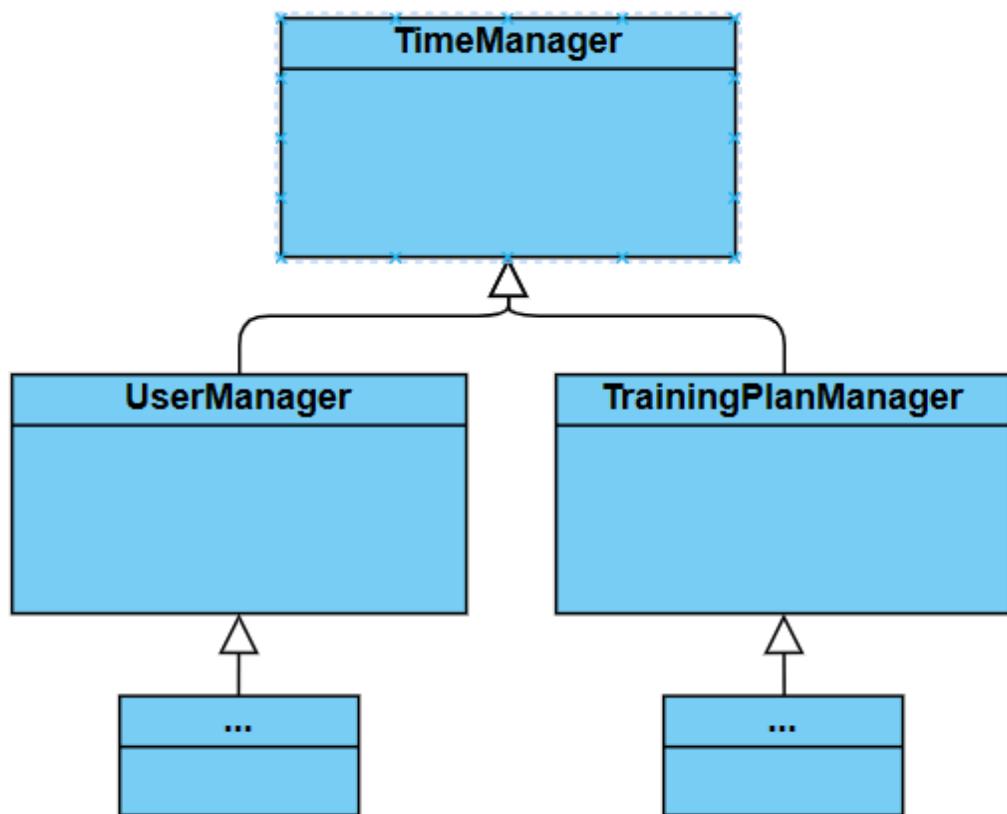
**Anexo 5: Exemplo de representação conceitual do padrão Facade na aplicação.**



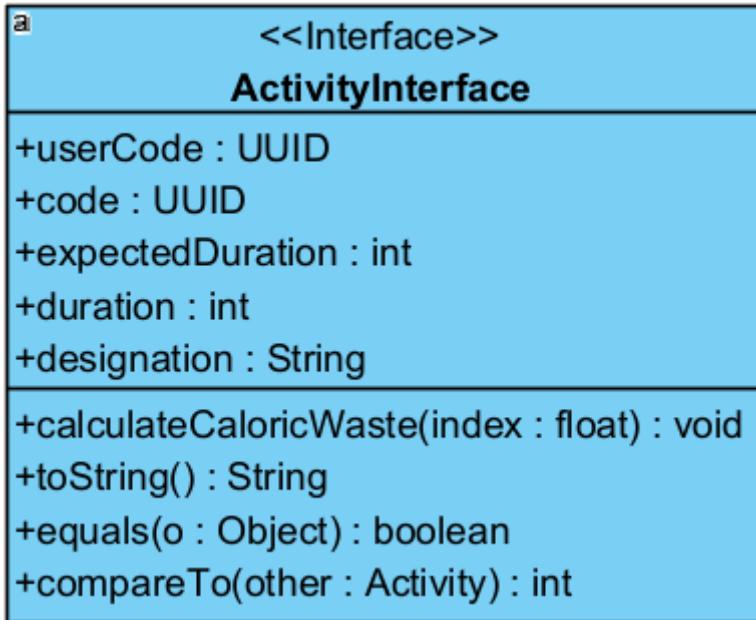
**Anexo 6: Exemplo de representação gráfica do padrão Observer.**



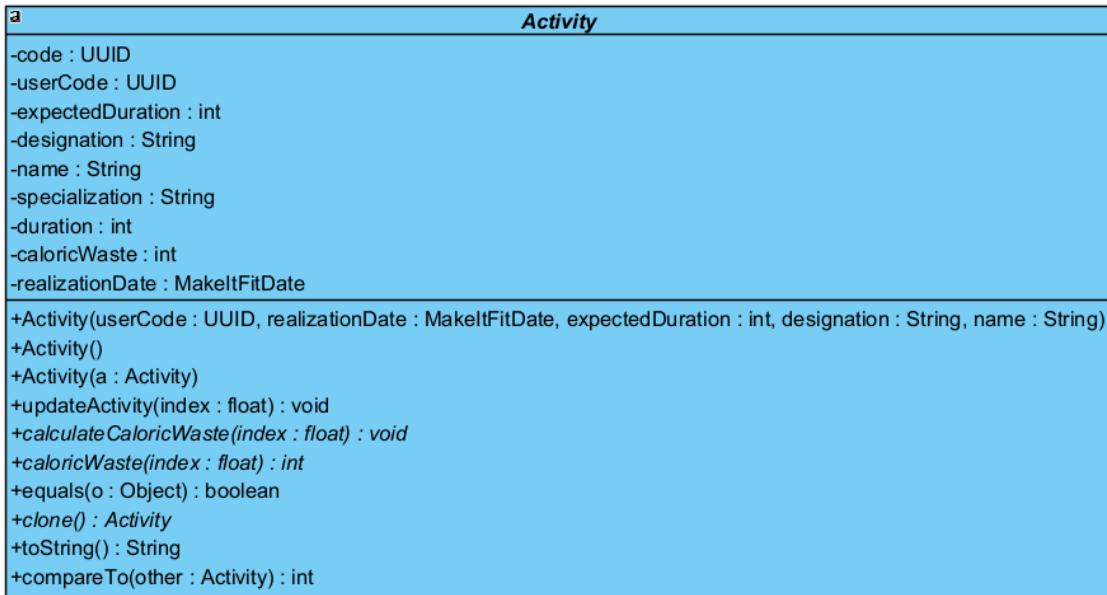
Anexo 7: Exemplo de representação conceitual do padrão Observer na aplicação.



**Anexo 8: Representação da interface "ActivityInterface" no diagrama de classes.**



**Anexo 9: Representação da classe abstrata "Activity" no diagrama de classes.**



**Anexo 10: Representação da classe abstrata "Repetitions" no diagrama de classes.**

a	<b>Repetitions</b>
-repetitions : int	
-series : int	
+Repetitions(userCode : UUID, realizationDate : MakeltFitDate, expectedDuration : int, designation : String, name : String, repetitions : int, series : int)	
+Repetitions()	
+Repetitions(r : Repetitions)	
+calculateCaloricWaste(index : float) : void	
+caloricWaste(index : float) : int	
+equals(o : Object) : boolean	
+clone() : Repetitions	
+toString() : String	

## Anexo 11: Representação da classe abstrata "Distance" no diagrama de classes.

a	<b>Distance</b>
-distance : double	
+Distance(userCode : UUID, realizationDate : MakeltFitDate, expectedDuration : int, designation : String, name : String, distance : double)	
+Distance()	
+Distance(d : Distance)	
+calculateCaloricWaste(index : float) : void	
+caloricWaste(index : float) : int	
+equals(o : Object) : boolean	
+clone() : Distance	
+toString() : String	

## Anexo 12: Representação da classe abstrata "RepetitionsWithWeights" no diagrama de classes.

a	<b>RepetitionsWithWeights</b>
-weight : double	
+RepetitionsWithWeights(userCode : UUID, realizationDate : MakeltFitDate, expectedDuration : int, designation : String, name : String, repetitions : int, series : int, weight : double)	
+RepetitionsWithWeights()	
+RepetitionsWithWeights(r : RepetitionsWithWeights)	
+calculateCaloricWaste(index : float) : void	
+caloricWaste(index : float) : int	
+toString() : String	
+equals(o : Object) : boolean	
+clone() : RepetitionsWithWeights	

## Anexo 13: Representação da classe abstrata "DistanceWithAltimetry" no diagrama de classes.

a	<b>DistanceWithAltimetry</b>
-elevationGain : double	
-elevationLoss : double	
+DistanceWithAltimetry(userCode : UUID, realizationDate : MakeltFitDate, expectedDuration : int, designation : String, name : String, distance : double, elevationGain : double, elevationLoss : double)	
+DistanceWithAltimetry()	
+DistanceWithAltimetry(a : DistanceWithAltimetry)	
+calculateCaloricWaste(index : float) : void	
+caloricWaste(index : float) : int	
+equals(o : Object) : boolean	
+clone() : DistanceWithAltimetry	
+toString() : String	

## Anexo 14: Representação da classe concreta "PushUp" no diagrama de classes.

```
a PushUp
+PushUp(userCode : UUID, realizationDate : MakeltFitDate, expectedDuration : int, designation : String, name : String, repetitions : int, series : int)
+PushUp()
+PushUp(p : PushUp)
+calculateCaloricWaste(index : float) : void
+caloricWaste(index : float) : int
+toString() : String
>equals(o : Object) : boolean
+clone() : PushUp
```

## Anexo 15: Representação da classe concreta "Running" no diagrama de classes.

```
a Running
-averageSpeed : double
+Running(userCode : UUID, realizationDate : MakeltFitDate, expectedDuration : int, designation : String, name : String, distance : double, speed : double)
+Running()
+Running(r : Running)
+getSpeed() : double
+setSpeed(speed : double) : void
+calculateCaloricWaste(index : float) : void
+caloricWaste(index : float) : int
+toString() : String
>equals(o : Object) : boolean
+clone() : Running
```

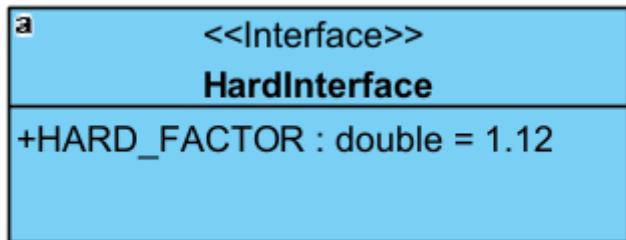
## Anexo 16: Representação da classe concreta "WeightSquat" no diagrama de classes.

```
a WeightSquat
+WeightSquat(userCode : UUID, realizationDate : MakeltFitDate, expectedDuration : int, designation : String, name : String, repetitions : int, series : int, weight : double)
+WeightSquat()
+WeightSquat(w : WeightSquat)
+calculateCaloricWaste(index : float) : void
+caloricWaste(index : float) : int
+toString() : String
>equals(o : Object) : boolean
+clone() : WeightSquat
```

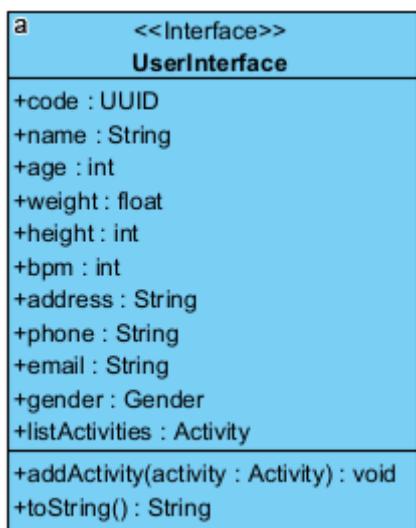
## Anexo 17: Representação da classe concreta "Trail" no diagrama de classes.

```
a Trail
+TRAIL_TYPE_EASY : int = 0
+TRAIL_TYPE_MEDIUM : int = 1
+TRAIL_TYPE_HARD : int = 2
-trailType : int
+Trail(userCode : UUID, realizationDate : MakeltFitDate, expectedDuration : int, designation : String, name : String, distance : double, elevationGain : double, elevationLoss : double, trailType : int)
+Trail()
+Trail(trail : Trail)
+calculateCaloricWaste(index : float) : void
+caloricWaste(index : float) : int
+toString() : String
>equals(obj : Object) : boolean
+clone() : Trail
```

**Anexo 18: Representação da interface "HardInterface" no diagrama de classes.**



**Anexo 19: Representação da interface "UserInterface" no diagrama de classes.**



**Anexo 20: Representação da classe abstrata "User" no diagrama de classes.**

a	User
-code : UUID	
-name : String	
-age : int	
-weight : float	
-height : int	
-bpm : int	
-level : int	
-address : String	
-phone : String	
-email : String	
-index : float	
-gender : Gender	
+activities : Activity	
+User(name : String, age : int, gender : Gender, weight : float, height : int, bpm : int, level : int, address : String, phone : String, email : String)	
+User()	
+User(u : User)	
+getListActivities() : List<Activity>	
+addActivity(activity : Activity) : void	
+addActivities(activities : List<Activity>) : void	
+removeActivity(activityCode : UUID) : void	
+updateActivities() : void	
+calculateIndex(weight : float, height : int, bpm : int) : float	
+clone() : User	
+toString() : String	
+equals(o : Object) : boolean	
+compareTo(u : User) : int	

## Anexo 21: Representação da classe concreta "Amateur" no diagrama de classes.

a	Amateur
+Amateur(name : String, age : int, gender : Gender, weight : float, height : int, bpm : int, level : int, address : String, phone : String, email : String)	
+Amateur(amateur : Amateur)	
+clone() : Amateur	
+toString() : String	

## Anexo 22: Representação da classe concreta "Occasional" no diagrama de classes.

a	Occasional
-frequency : int	
+Occasional(name : String, age : int, gender : Gender, weight : float, height : int, bpm : int, level : int, address : String, phone : String, email : String, frequency : int)	
+Occasional(o : Occasional)	
+clone() : Occasional	
+toString() : String	

## Anexo 23: Representação da classe concreta "Professional" no diagrama de classes.

a	Professional
-specialization : String	
-frequency : int	
+Professional(name : String, age : int, gender : Gender, weight : float, height : int, bpm : int, level : int, address : String, phone : String, email : String, frequency : int)	
+Professional(p : Professional)	
+updateSpecialization() : void	
+clone() : Professional	
+toString() : String	

## Anexo 24: Representação da classe concreta "UserManager" no diagrama de classes.

a	UserManager
<pre> -usersByCode : Map&lt;UUID, User&gt; -usersByEmail : Map&lt;String, User&gt;  +UserManager() +createUser(name : String, age : int, gender : Gender, weight : float, height : int, bpm : int, level : int, address : String, phone : String, email : String, frequency : int, type : String) : User +insertUser(user : User) : void +removeUserByCode(code : UUID) : void +removeUserByEmail(email : String) : void +existsUserWithEmail(email : String) : boolean +getUserByCode(code : UUID) : User +getUserByEmail(email : String) : User +updateUser(user : User) : void +getAllUsers() : List&lt;User&gt; +getActivitiesFromUser(email : String) : List&lt;Activity&gt; +addActivityToUser(email : String, activity : Activity) : void +removeActivityFromUser(email : String, activityCode : UUID) : void +addActivitiesToUser(userCode : UUID, activities : List&lt;Activity&gt;) : void +updateSystem() : void </pre>	

## Anexo 25: Representação da classe concreta "TrainingPlan" no diagrama de classes.

a	TrainingPlan
<pre> &lt;&lt;Property&gt;&gt; -userCode : UUID &lt;&lt;Property&gt;&gt; -code : UUID &lt;&lt;Property&gt;&gt; -activities : MyTuple&lt;Integer, Activity&gt; &lt;&lt;Property&gt;&gt; -startDate : MakeltFitDate  +TrainingPlan(userCode : UUID, startDate : MakeltFitDate) +TrainingPlan() +TrainingPlan(tp : TrainingPlan) +addActivity(repetitions : int, activity : Activity) : void +removeActivity(code : UUID) : void +updateActivities(currentDate : MakeltFitDate, index : float) : void +extractActivities(currentDate : MakeltFitDate) : List&lt;Activity&gt; +toString() : String +equals(o : Object) : boolean +compareTo(other : TrainingPlan) : int </pre>	

## Anexo 26: Representação da classe concreta "TrainingPlanManager" no diagrama de classes.

a	TrainingPlanManager
<pre> -trainingPlans : Map&lt;UUID, TrainingPlan&gt; +TrainingPlanManager() +createTrainingPlan(userCode : UUID, startDate : MakeltFitDate) : TrainingPlan +constructTrainingPlanByObjectives(trainingPlan : TrainingPlan, index : float, hardActivities : boolean, maxActivitiesPerDay : int, maxDifferentActivities : int, weeklyRecurrence : int, minimumCaloricWaste : int) : TrainingPlan +insertTrainingPlan(trainingPlan : TrainingPlan) : void +removeTrainingPlan(code : UUID) : void +getTrainingPlan(code : UUID) : TrainingPlan +updateTrainingPlan(trainingPlan : TrainingPlan) : void +getAllTrainingPlans() : List&lt;TrainingPlan&gt; +addActivity(code : UUID, repetitions : int, activity : Activity) : void +removeActivity(code : UUID, activity : UUID) : void +getTrainingPlansFromUser(userCode : UUID) : List&lt;TrainingPlan&gt; +updateActivities(currentDate : MakeltFitDate, index : float) : void +extractActivities(currentDate : MakeltFitDate, userCode : UUID) : List&lt;Activity&gt; </pre>	

**Anexo 27: Representação da classe concreta "MakeltFitDate" no diagrama de classes.**

a	MakeltFitDate
-date : LocalDate	
+MakeltFitDate()	
+of(year : int, month : int, dayOfMonth : int) : MakeltFitDate	
-MakeltFitDate(date : LocalDate)	
+fromString(date : String) : MakeltFitDate	
+getDayOfWeek() : int	
+getDayOfMonth() : int	
+getMonth() : int	
+getYear() : int	
+isBeforeOrSame(date : MakeltFitDate) : boolean	
+isBefore(date : MakeltFitDate) : boolean	
+isAfterOrSame(date : MakeltFitDate) : boolean	
+isAfter(date : MakeltFitDate) : boolean	
+plusDays(days : int) : MakeltFitDate	
+equals(o : Object) : boolean	
+distance(date : MakeltFitDate) : int	
+toString() : String	
+compareTo(object : MakeltFitDate) : int	
+clone(object : MakeltFitDate) : int	

**Anexo 28: Representação da classe concreta "Emailvalidator" no diagrama de classes.**

a	EmailValidator
-EMAIL_PATTERN : String = "[a-zA-Z0-9.+_-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}\$_"	
+isValidEmail(emailString : String) : boolean	

**Anexo 29: Representação da classe concreta "TimeManager" no diagrama de classes.**

a	TimeManager
-currentDate : MakeltFitDate	
+TimeManager()	
+TimeManager(currentDate : MakeltFitDate)	
+advanceTime(days : int) : MakeltFitDate	

**Anexo 30: Representação da classe concreta "Menu" no diagrama de classes.**

<b>a</b>	<b>Menu</b>
-keepRunning : boolean	
-items : MenuItem	
+Menu(items : List<MenuItem>)	
+display() : void	
+getUserChoice() : int	
+executeSelectedOption() : void	
+run() : void	

**Anexo 31: Representação da classe concreta "MenuItem" no diagrama de classes.**

<b>a</b>	<b>MenuItem</b>
-name : String	
-preCondition : PreCondition	
-action : Handler	
+MenuItem(name : String, action : Handler)	
+MenuItem(name : String, preCondition : PreCondition, action : Handler)	
+execute() : void	
+isValid() : boolean	

**Anexo 32: Representação da interface "PreCondition" no diagrama de classes.**

<b>a</b>	<b>&lt;&lt;Interface&gt;&gt;</b>
	<b>PreCondition</b>
+validate() : boolean	

**Anexo 33: Representação da interface "Handler" no diagrama de classes.**



**Anexo 34: Representação da classe concreta "QueriesManager" no diagrama de classes.**

a	QueriesManager
-mostDoneActivity : MostDoneActivity	
-howManyAltimetryDone : HowManyAltimetryDone	
-howManyKMsDone : HowManyKMsDone	
-mostDemandingTrainingPlan : MostDemandingTrainingPlan	
-whoBurnsMoreCalories : WhoBurnsMoreCalories	
-whoDidTheMostActivities : WhoDidTheMostActivities	
-userManager : UserManager	
-trainingPlanManager : TrainingPlanManager	
+QueriesManager(userManager : UserManager, trainingPlanManager : TrainingPlanManager)	
+executeQueryHowManyAltimetryDone(userManager : UserManager, email : String, date1 : MakelFitDate, date2 : MakelFitDate) : double	
+executeQueryHowManyKMsDone(userManager : UserManager, email : String) : double	
+executeQueryHowManyKMsDone(userManager : UserManager, email : String, date1 : MakelFitDate, date2 : MakelFitDate) : double	
+executeQueryHowManyKMsDone(userManager : UserManager, email : String) : double	
+executeQueryMostDoneActivity(userManager : UserManager) : String	
+executeQueryMostDemandingTrainingPlan(trainingPlanManager : TrainingPlanManager, userManager : UserManager) : TrainingPlan	
+executeQuerywhoBurnsMoreCalories(userManager : UserManager, date1 : MakelFitDate, date2 : MakelFitDate) : User	
+executeQuerywhoBurnsMoreCalories(userManager : UserManager) : User	
+executeQueryWhoDidTheMostActivities(userManager : UserManager, date1 : MakelFitDate, date2 : MakelFitDate) : User	
+executeQueryWhoDidTheMostActivities(userManager : UserManager) : User	

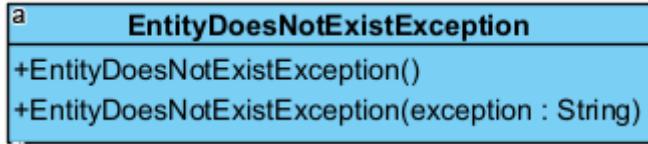
**Anexo 35: Representação da classe concreta "ExistingEntityConflictException" no diagrama de classes.**

a	ExistingEntityConflictException
+ExistingEntityConflictException()	
+ExistingEntityConflictException(exception : String)	

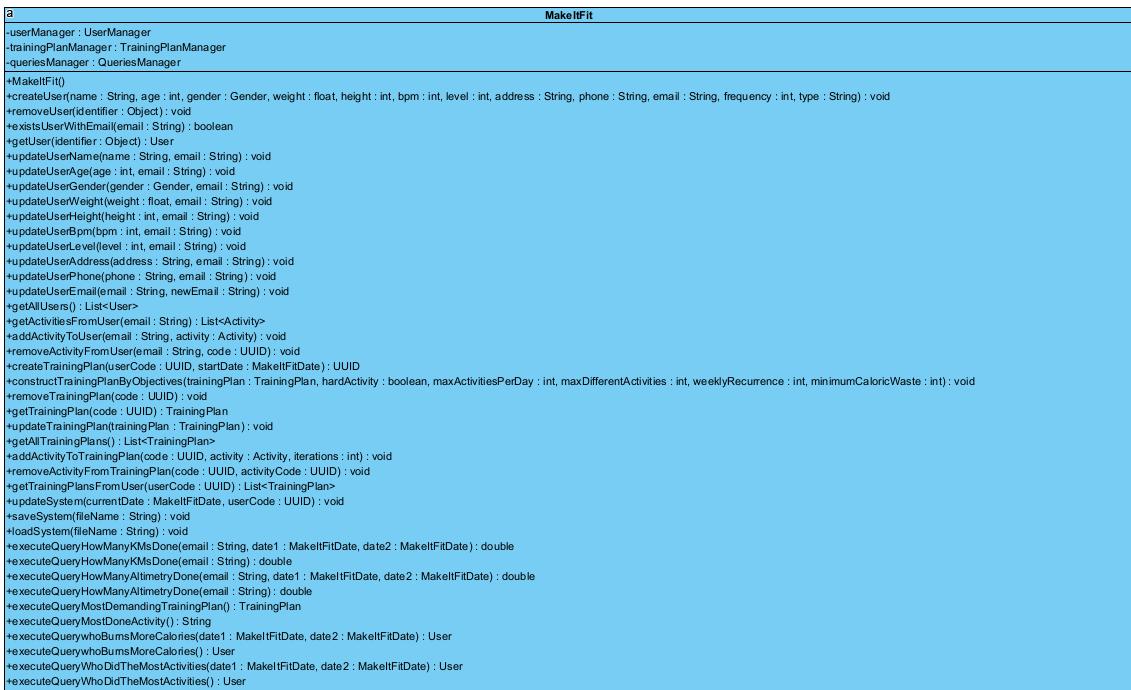
**Anexo 36: Representação da classe concreta "InvalidTypeException" no diagrama de classes.**

a	InvalidTypeException
+InvalidTypeException()	
+InvalidTypeException(exception : String)	

## Anexo 37: Representação da classe concreta "EntityDoesNotExistException" no diagrama de classes.



## Anexo 38: Representação da classe concreta "MakeItFit" no diagrama de classes.



## Anexo 39: Representação da classe concreta "MakeItFitController" no diagrama de classes.

a	MakeItFitController
<pre>+email : String -name : String -trainingPlan : UUID -makeItFit : MakeItFit -timeManager : TimeManager +MakeItFitController() +login(email : String) : void +createUser(name : String, age : int, gender : Gender, weight : float, height : int, bpm : int, level : int, address : String, phone : String, frequency : int, type : String) : void +removeUser() : void +getUserDetails() : String +updateName(name : String) : void +updateAge(age : int) : void +updateGender(gender : Gender) : void +updateWeight(weight : float) : void +updateHeight(height : int) : void +updateBpm(bpm : int) : void +updateLevel(level : int) : void +updateAddress(address : String) : void +updatePhone(phone : String) : void +updateEmail(email : String) : void +getAllUsers() : String +getActivities() : String +addActivityToUser(date : MakeItFitDate, duration : int, designation : String, name : String, repetitions : int, series : int) : void +addActivityToUser(date : MakeItFitDate, duration : int, designation : String, name : String, distance : double, speed : double) : void +addActivityToUser(date : MakeItFitDate, duration : int, designation : String, name : String, distance : double, elevationGain : double, elevationLoss : double, trailType : int) : void +addActivityToUser(date : MakeItFitDate, duration : int, designation : String, name : String, repetitions : int, series : int, weight : double) : void +removeActivityFromUser(activity : UUID) : void +updateSystemDate(days : int) : void +createTrainingPlan(startDate : MakeItFitDate) : void +removeTrainingPlan() : void +constructTrainingPlanByObjectives(hardActivity : boolean, maxActivitiesPerDay : int, maxDifferentActivities : int, weeklyRecurrence : int, minimumCaloricWaste : int) : void +addActivityToTrainingPlan(date : MakeItFitDate, duration : int, designation : String, name : String, repetitions : int, series : int, iterations : int) : void +addActivityToTrainingPlan(date : MakeItFitDate, duration : int, designation : String, name : String, distance : double, speed : double, iterations : int) : void +addActivityToTrainingPlan(date : MakeItFitDate, duration : int, designation : String, name : String, distance : double, elevationGain : double, elevationLoss : double, trailType : int, iterations : int) : void +addActivityToTrainingPlan(date : MakeItFitDate, duration : int, designation : String, name : String, repetitions : int, series : int, weight : double, iterations : int) : void +removeActivityFromTrainingPlan(activity : UUID) : void +getTrainingPlansFromUser() : String +getTrainingPlans() : String +saveSystem(fileName : String) : void +loadSystem(fileName : String) : void +executeQueryHowManyKMsDone(date1 : MakeItFitDate, date2 : MakeItFitDate) : double +executeQueryHowManyKMsDone() : double +executeQueryHowManyAltimetryDone(date1 : MakeItFitDate, date2 : MakeItFitDate) : double +executeQueryHowManyAltimetryDone() : double +executeQueryMostDemandingTrainingPlan() : String +executeQueryMostDoneActivity() : String +executeQueryWhoBurnsMoreCalories() : String +executeQueryWhoDidTheMostActivities(date1 : MakeItFitDate, date2 : MakeItFitDate) : String +executeQueryWhoDidTheMostActivities() : String +executeQueryWhoDidTheMostActivitiesFromUser() : String</pre>	

## Anexo 40: Representação da classe concreta "AdminView" no diagrama de classes.

a	AdminView
<pre>+AdminView() +setEmail() : void +login() : void -createAppMenu() : Menu +removeUser() : void +printAllUsersDetails() : void +listAllTrainingPlans() : void +executeQueries() : void +createQueryMenu() : Menu +executeQueryHowManyKMsDone() : void +executeQueryHowManyAltimetryDone() : void +executeQueryMostDemandingTrainingPlan() : void +executeQueryMostDoneActivity() : void +executeQueryWhoBurnsMoreCalories() : void +executeQueryWhoDidTheMostActivities() : void +executeQueryGetAllActivitiesFromUser() : void</pre>	

## Anexo 41: Representação da classe abstrata "MakeItFitView" no diagrama de classes.

a	<b>MakeItFitView</b>
-userType : String	
-activityType : String	
+makeItFitController : MakeItFitController	
+MakeItFitView()	
+login() : void	
+createUser() : void	
+createUserChoiceMenu() : Menu	
+getUserDetails() : void	
+createUpdateUserMenu() : Menu	
-updateName() : void	
-updateAge() : void	
-updateGender() : void	
-updateWeight() : void	
-updateHeight() : void	
-updateBpm() : void	
-updateLevel() : void	
-updateAddress() : void	
-updatePhone() : void	
-updateEmail() : void	
+createActivityChooserMenu() : Menu	
+addActivityToUser() : void	
+removeActivityFromUser() : void	
+listActivities() : void	
+createTrainingPlan() : void	
+createTrainingPlanMenu() : Menu	
+constructTrainingPlan() : void	
+addActivityToTrainingPlan() : void	
+generateTrainingPlan() : void	
+removeActivityFromTrainingPlan() : void	
+removeTrainingPlan() : void	
+selectTrainingPlan() : void	
+listAllTrainingPlansFromUser() : void	
+advanceTimeManager() : void	
+saveSystem() : void	

## Anexo 42: Representação da classe concreta "UserView" no diagrama de classes.

a	<b>UserView</b>
+UserView()	
+login() : void	
-createAppMenu() : Menu	