

POO (LEI/LCC)

2022/2023

Ficha Prática #06

Hierarquia, herança, polimorfismo, interfaces, exceções e serialização de informação.

Conteúdo

1	Objectivos	3
2	Projeto Fitness	3
2.1	Fase 1	4
2.2	Fase 2	6
2.3	Fase 3	6
2.4	Fase 4	7

1 Objectivos

Durante três aulas, o projecto Fitness servirá para abordar vários conceitos leccionados em POO. Do mesmo modo, servirá como auto-avaliação para que os alunos possam fazer um ponto da situação sobre a disciplina.

O objectivo deste projecto é o desenvolvimento, de uma forma incremental, de uma aplicação que combina todos os conceitos leccionados ao longo do semestre. Na primeira fase serão abordados os conceitos de hierarquias de classes, assim como mapeamentos (c.f. API `Map<K,V>` já praticada anteriormente na Ficha 5). A segunda fase abordará os conceitos de Interface, nomeadamente as interfaces `Comparable<T>` e `Comparator<T>`. A terceira fase abordará a criação de interfaces criadas pelo programador e na quarta e última fase será abordado o tema de *streams* de dados assim como o tratamento de excepções.

2 Projeto Fitness

Considere que se pretende desenvolver um sistema para o armazenamento e gestão das actividades físicas realizadas por utilizadores da aplicação. A génese da aplicação a implementar é similar à das conhecidas aplicações de fitness como o Strava, Runkeeper e outras. A aplicação a criar é na sua essência um ambiente de recolha de informação e que possibilitará que os utilizadores registem a sua informação das actividades desportivas que realizam. Cada utilizador realiza um conjunto de actividades desportivas (corrida, caminhada, bicicleta, natação, ginástica, etc.) que regista na aplicação. Cada modalidade tem as suas características pelo que deverá ser guardada a informação necessária para descrever a sua actividade.

Utilizadores

Para cada utilizador guarda-se a seguinte informação pessoal:

- email, que é a chave do utilizador;
- password;
- nome;
- género;
- altura;
- peso;
- data de nascimento;
- data de registo no sistema;

- desporto favorito - que é o desporto em que pratica mais actividades;

Além desta informação, que deve poder ser editada, o utilizador regista também a informação das actividades que realizou;

2.1 Fase 1

A aplicação deve conhecer uma lista das actividades desportivas mais conhecidas e deverá também possibilitar que se adicionem novos tipos de actividades. Todas as actividades, independentemente da sua natureza, possuem informação sobre:

- um código único que a permite identificar
- a descrição da actividade (e.g. "corrida matinal", "pilates semanal", etc.)
- a data da sua realização
- o tempo (em minutos) que demorou

A aplicação neste momento tem três tipos de actividades já identificadas: **Corrida**, **Canoagem** e **Abdominais**. A Corrida acrescenta informação sobre a distância percorrida, a altimetria ganha e o percurso efectuado (por simplificação assumo que o percurso é uma string de caracteres. Existirá depois um método que dada uma string destas cria um png).

A actividade de Canoagem, tem informação adicional respeitante à embarcação utilizada (uma String - e.g. "K2", "K4"), ao valor em km/h do vento, à direcção deste, à distância percorrida e ao número de voltas à pista.

A actividade de Abdominais tem informação sobre o tipo de abdominal e número de repetições.

Para todas estas actividades deve ser possível calcular o valor calórico dispendido, sendo que a fórmula de cálculo é dependente da actividade e do utilizador. Neste momento sabe-se que para as actividades existentes as fórmulas são:

$$CaloriasCorrida = distancia * peso_{utilizador} * tempo * idade / 50$$

$$CaloriasCanoagem = distancia * vento_{contra} * tempo * idade / 4$$

$$CaloriasAbdominais = tempo * repeticoes * 3/5$$

1. Desenhe o diagrama de classes que modela a arquitectura proposta.
2. Crie as classes Utilizador, Corrida, Canoagem e Abdominal e a classe Fitness, e implemente nesta os seguintes métodos:
 - (a) Verificar a existência de um utilizador, dado o seu código.
`public boolean existeUtilizador(String email)`
 - (b) Devolver a quantidade de utilizadores existentes na aplicação de fitness
`public int quantos()`

- (c) Devolver o número total de actividades desportivas, de um dado tipo, efectuadas por um utilizador.
`public int quantos(String actividade, String email)`
- (d) Devolver a informação de um utilizador, dado o seu código.
`public Utilizador getUtilizador(String email)`
- (e) Adicionar uma actividade ao registo de um utilizador.
`public void adiciona(String email, Actividade act)`
- (f) Devolver uma lista contendo a cópia de todas as actividades existentes na aplicação.
`public List<Actividade> getAllActividades()`
- (g) Adicionar a informação de um conjunto de actividades de um utilizador e que foram feitas numa outra aplicação e que passam agora a fazer parte do *Fitness*.
`public void adiciona(String email, Set<Actividade> activs)`
- (h) Indicar o número total de minutos que foram dispendidos por um utilizador em actividades de fitness
`public int tempoTotalUtilizador(String email)`
- (i) Devolver a actividade com maior dispêndio de calorias
`public Actividade actividadeMaisExigente()`
- (j) Obter o utilizador com maior dispêndio de calorias
`public Utilizador utilizadorMaisActivo()`
- (k) Actualizar, para todos os utilizadores, a informação relativa ao seu desporto favorito, fazendo com que seja verdade que este é o desporto em que registam mais actividades por parte desse utilizador.
`public void actualizaDesportoFav()`

Auto avaliação 1 Como forma de auto avaliação, propõe-se a implementação da classe *FitnessList*, com os mesmos requisitos que *Fitness*, mas utilizando como estrutura de dados um *List<Utilizador>*.

Auto avaliação 2 Propõe-se também a implementação da classe *FitnessSet*, com os mesmos requisitos que *Fitness*, mas utilizando como estrutura de dados um *TreeSet<Utilizador>*.

2.2 Fase 2

1. Pretende-se agora poder consultar a informação de forma ordenada por diferentes critérios. Assim, altere a classe `Fitness` para suportar as seguintes funcionalidades.

- (a) Determinar o utilizador que efectuou mais actividades em todo o sistema. Em caso de empate será devolvido aquele que tiver maior antiguidade no sistema.

```
public Utilizador utilizadorComMaisRegistos()
```

- (b) Obter um `Set<Utilizador>`, ordenado de acordo com a ordem natural dos utilizadores (assuma que a ordem natural dos utilizadores é em primeiro lugar a ordem crescente do seu consumo de calorias e que o segundo factor de comparação é a ordem alfabética do nome)

```
public Set<Utilizador> ordenarUtilizadores()
```

- (c) Faça o mesmo que na alínea anterior, mas agora para a assinatura:

```
public List<Utilizador> ordenarUtilizadores()
```

- (d) Obter um `List<Utilizador>`, ordenado de acordo com o comparador fornecido:

```
public List<Utilizador> ordenarUtilizador(Comparator<Utilizador>
```

c). Experimente criar ordenações em função do número de calorias dispendidas e em função do número de actividades efectuadas (experimente comparadores por ordem crescente e decrescente).

- (e) Guardar (de forma a que sejam identificáveis por nome) comparadores na classe `Fitness`, permitindo assim ter disponíveis diferentes critérios de ordenação.

- (f) Obter um iterador de `Utilizador`, ordenado de acordo com o critério indicado:

```
public Iterator<Utilizador> ordenarUtilizador(String  
criterio)
```

- (g) Obter um `Map` em que se associa a cada actividade os seus top 3 utilizadores com mais dispêndio de calorias nesta actividade, ordenados de forma decrescente por calorias. A chave do `Map` será o nome da classe da actividade.

```
public Map<String, List<Utilizador>> podiumPorActiv()
```

2.3 Fase 3

Considere que a aplicação `Fitness` tem uma vertente de gamification e que por cada metro percorrido atribui pontos que depois são usados para fazer um ranking dos utilizadores. Esses pontos só são atribuídos nas actividades que possuam distância (neste caso apenas a `Corrida` e a `Canoagem`) e cada uma das actividades terá uma fórmula diferente para calcular o valor de pontos resultante. Por

exemplo na corrida o número de pontos é multiplicado pela distância acumulada e na canoagem é 1.5 vezes o valor do vento.

1. Desenvolva a interface `FazMetros`, que deverá ser implementada por estes tipos de actividades. Esta interface deverá garantir as funcionalidades a seguir descritas:
 - definir o total de pontos a atribuir por cada metro
 - obter o valor de pontos que se está a atribuir por cada metro
 - obter o total de pontos que uma determinada actividade originou.
2. Actualize as classes `Corrida` e `Canoagem` de modo a que implementem a interface.
3. Acrescente, na classe `Fitness`, um método para obter a lista das actividades efectuadas e que atribuem pontos.

```
public List<FazMetros> daoPontos()
```

2.4 Fase 4

Deverá ser possível exportar toda a informação relativa a utilizadores e actividades, representada em memória pela classe `Fitness`. Implemente assim as seguintes funcionalidades:

1. Exportar a informação de todas as actividades de um utilizador para um ficheiro CSV
2. Gravar e carregar a instância de `Fitness` num (e de um) ficheiro de objectos.

Por forma a melhorar a implementação efectuada até agora de `Fitness`, codifique agora excepções para suportar o tratamento de erros a seguir descritos:

3. Utilizador inexistente numa consulta
4. Adicionar um utilizador repetido
5. Adicionar uma actividade a um utilizador inexistente

Finalmente, para completar a implementação do sistema

6. Desenvolva a classe `FitnessApp` que deverá dar acesso às funcionalidades da classe `Fitness` através de um menu em modo texto.