

POO (LEI/LCC)

2024/2025

Ficha Prática #05

Map<K,V>

Conteúdo

1	Objetivos	3
2	API essencial de Map	3
3	Exercícios	3

1 Objectivos

- Aprender a trabalhar com Set.
- Aprender a trabalhar com Map.

2 API essencial de Map

A API do tipo `Map<K, V>`, comum a ambas as implementações, é apresentada de seguida.

Categoria de Métodos	API de <code>Map<K, V></code>
Inserção de elementos	<code>put(K k, V v);</code> <code>putAll(Map<? extends K, ? extends V> m);</code>
Remoção de elementos	<code>remove(Object k);</code>
Consulta e comparação de conteúdos	<code>V get(Object k);</code> <code>boolean containsKey(Object k);</code> <code>boolean isEmpty();</code> <code>boolean containsValue(Object v);</code> <code>int size();</code>
Criação de Iteradores	<code>Set<K> keySet();</code> <code>Collection<V> values();</code> <code>Set<Map.Entry<K,V>> entrySet();</code>
Outros	<code>boolean equals(Object o);</code> <code>Object clone();</code>

Para mais informações sobre as APIs consulte os apontamentos e a API do Java.

3 Exercícios

1. Desenvolva uma classe `Lugar` que represente a informação básica de um lugar de estacionamento, existente num dado parque. Sobre cada lugar pretende ter-se a seguinte informação:

```

1  /** Matricula do veiculo estacionado */
2  private String matricula;
3  /** Nome do proprietario */
4  private String nome;
5  /** Tempo atribuido ao lugar, em minutos */
6  private int minutos;
7  /** Indica se lugar é permanente, ou de aluguer
   8  */
   private boolean permanente;

```

Crie em seguida uma classe Parque contendo o nome do parque em questão e uma representação dos lugares do parque, associando a cada matrícula, a informação do lugar associado.

- (a) Desenhe o diagrama de classe com a arquitectura proposta. Para saber quais os métodos a considerar leia todo o enunciado da questão.
 - (b) Para além dos construtores e métodos usuais, a classe Parque deverá definir ainda os seguintes métodos de instância:
 - Método que devolve todas as matriculas dos lugares ocupados;
 - Método que regista uma nova ocupação de lugar;
 - Método que remove o lugar de dada matrícula;
 - Método que altera o tempo disponível de um lugar, para uma dada matrícula;
 - Método que devolve a quantidade total de minutos atribuídos. Implemente com **iterador interno** e **iterador externo**;
 - Método que verifica existe lugar atribuído a uma dada matrícula;
 - Método que cria uma lista com as matriculas com tempo atribuído $> x$, em que o lugar seja permanente. Implemente com **iterador interno** e **iterador externo**;
 - Método que devolve uma cópia dos lugares;
 - Método que devolve a informação de um lugar para uma dada matrícula;
2. Considere a classe Aluno definida nos slides das aulas teóricas. Crie agora a classe TurmaAlunos que associa a cada número de aluno a informação da instância relacionada (assume-se que o número de aluno é do tipo String). A classe TurmaAlunos tem, além dos alunos registados, informação sobre o nome da turma e o código da UC.
- (a) Desenhe o diagrama de classe com a arquitectura proposta. Para saber quais os métodos a considerar leia todo o enunciado da questão.
 - (b) Codifique os seguintes métodos:
 - i. métodos usuais da classe TurmaAlunos, nomeadamente construtores, *getters* e *setters*, equals, toString, clone e compareTo
 - ii. adicionar um novo aluno à turma, `public void insereAluno(Aluno a)`
 - iii. dado um código de aluno devolver a instância de Aluno associada, `public Aluno getAluno(String codAluno)`
 - iv. remover um aluno dado o seu código, `public void removeAluno(String codAluno)`
 - v. devolver a informação de todos os números de aluno existentes, `public Set<String> todosOsCodigos()`

- vi. devolver a informação de quantos alunos existem na turma, `public int qtsAlunos()`
 - vii. devolver os alunos ordenados por ordem alfabética do seu nome, `public Collection<Aluno> alunosOrdemAlfabetica()`
 - viii. devolver os alunos ordenados por ordem decrescente do seu número, `public Set<Aluno> alunosOrdemDescrescenteNumero()` (assume-se que não existem números repetidos, daí ser viável devolver um Set sem correr o risco de a comparação eliminar resultados).
3. Considere a classe `VideoYouTube` que realizou na Ficha 3. Pretende-se agora desenvolver uma classe, `SistemaVideos`, que permita guardar vários videos associando a cada código de video a informação respectiva.
- (a) Desenhe o diagrama de classe com a arquitectura proposta. Para saber quais os métodos a considerar leia todo o enunciado da questão.
 - (b) Codifique os métodos:
 - i. métodos usuais da classe `SistemaVideos`, nomeadamente construtores, *getters* e *setters*, *equals*, *toString*, *clone* e *compareTo*
 - ii. adicionar um novo video ao sistema, `public void addVideo(Video v)`
 - iii. dado um código de video devolver a instância associada, `public Video getVideo(String codVideo)`
 - iv. remover um video dado um código, `public void removeVideo(String codVideo)`
 - v. dado um código de video adicionar mais um *like* ao mesmo, `public void addLike(String codVideo)`
 - vi. devolver o código do video com mais *likes*, `public String topLikes()`
 - vii. devolver o código do video com mais *likes* num intervalo de tempo, `public String topLikes(LocalDate dinicial, LocalDate dfinal)`
 - viii. devolve o video mais longo, `public Video videoMaisLongo()`
4. Considere a class `Encomenda` que desenvolveu na Ficha 4 (implementação com `ArrayList`). Considere agora que se pretende desenvolver uma classe `GestãoEncomendas`, que associa a cada encomenda (identificada pelo seu código) a informação respectiva.
- (a) Desenhe o diagrama de classe com a arquitectura proposta. Para saber quais os métodos a considerar leia todo o enunciado da questão.

(b) Desenvolva esta classe codificando os métodos habituais e ainda os métodos seguintes:

- i. método que determina os códigos de encomenda existentes, `public Set<Integer> todosCodigosEnc()`
- ii. método que adiciona mais uma encomenda ao sistema, `public void addEncomenda(Encomenda enc)`
- iii. método que dado um código de encomenda devolve a informação respectiva, `public Encomenda getEncomenda(Integer codEnc)`
- iv. método que remove uma encomenda dado o seu código, `public void removeEncomenda(Integer codEnc)`
- v. método que determina a encomenda (identificada pelo código) com mais produtos encomendados, `public Integer encomendaComMaisProdutos()`
- vi. método que determina todas as encomendas em que um determinado produto, identificado pelo código, está presente, `public Set<Integer> encomendasComProduto(String codProd)`
- vii. método que determina todas as encomendas com data posterior a uma data fornecida como parâmetro, `public Set<Integer> encomendasAposData(LocalDate d)`
- viii. método que determina a encomenda de maior valor, `public Encomenda encomendaMaiorValor()`
- ix. método que determina uma ordenação das encomendas por quantidade crescente de produtos por encomenda, `public Set<Encomenda> encomendasOrdenadasPorQtProdutos()`
- x. método que devolve uma ordenação, por ordem decrescente de valor de encomenda, de todas as encomendas do sistema, `public Set<Encomenda> encomendasValorDecrescente()`
- xi. método que calcula um map em que associa cada código de produto à lista das encomendas em que foi referida, `public Map<String,List<Integer>> encomendasDeProduto()`

5. Pretende-se criar uma classe para representar grafos dirigidos. Para tal, foi decidido utilizar uma lista de adjacência que associa, a cada vértice, os vértices que podem ser visitados a partir dele. Foi já definida a seguinte estrutura base:

```
1 import java.util.Set;
2 import java.util.Map;
3 import java.util.HashMap;
4
5 public class Grafo {
6     // variáveis de instância
```

```

7 |     private Map<Integer, Set<Integer>> adj;
   |                               // "lista" de adjacência
8 | }

```

- (a) Desenhe o diagrama de classe com a arquitetura proposta. Para saber quais os métodos a considerar leia todo o enunciado da questão.
- (b) Complete a classe definindo:
- Os construtores `Grafo()` e `Grafo(Map<Integer, Set<Integer>> adj)`.
 - `void addArco(Integer vOrig, Integer vDest)` – método que adiciona um arco ao grafo. Note que todos os vértices do grafo devem ter uma entrada na lista de adjacência; que eventualmente poderá ser vazia.
 - `boolean isSink(Integer v)` – método que determina se um vértice é um *sink* (não existem arcos a sair dele – ou seja, está-lhe associado um conjunto vazio de vértices na “lista” de adjacência).
 - `boolean isSource(Integer v)` – método que determina se um vértice é um *source* (não existem arcos a entrar nele).
 - `int size()` – método que calcula o tamanho do grafo (o tamanho de um grafo com n vértices e m arcos é $n + m$).
 - `boolean haCaminho(Integer vOrig, Integer vDest)` – método que determina se existe um caminho entre os dois vértices passados como parâmetro. Tenha em consideração que poderão existir ciclos no grafo.
 - `List<String> getCaminho(Integer vOrig, Integer vDest)` – método que calcula o caminho entre dois vértices. O método deverá devolver `null` caso não exista caminho.
 - `Set<Map.Entry<Integer, Integer>> fanOut (Integer v)` – método que calcula o conjunto de todos os arcos que saem de um vértice.
 - `Set<Map.Entry<Integer, Integer>> fanIn(Integer v)` – método que calcula o conjunto de todos os arcos que entram num vértice.
 - `boolean subGrafo(Grafo g)` – método que determina se o grafo é subgrafo de g (todos os seus vértices e arcos são vértices/arcos de g).
6. Considere a classe `FBPost` que codificou na Ficha 3. Pretende-se agora que reescreva o sistema de gestão de posts utilizando como variável de instância um `Map`. A classe `FBFeedMap` tem como variável de instância um `Map<String, List<FBPost>>` em que associa um nome de utilizador aos posts por ele criados.

- (a) Desenhe o diagrama de classe com a arquitectura proposta. Para saber quais os métodos a considerar leia todo o enunciado da questão.
- (b) Desenvolva esta classe codificando os métodos habituais e ainda os métodos seguintes:
 - i. método que permite adicionar um post de um utilizador, `public void addPost(String user, FBPost post)`
 - ii. método que remove os posts de um utilizador entre duas datas, `public void removePosts(String user, LocalDateTime di, LocalDateTime df)`
 - iii. método que determina quantos posts foram publicados durante um período de tempo, `public int postsNumPeriodo(LocalDateTime di, LocalDateTime df)`
 - iv. método que determina o utilizador mais activo num determinado período de tempo, `public String utilizadorMaisActivo(LocalDateTime di, LocalDateTime df)`
 - v. método que determina a timeline do sistema ordenando cronologicamente todos os posts, `public List<FBPost> timelineGlobal()`
 - vi. método que valida que não existe nenhum utilizador que tenham feito mais que um post num determinado instante, `public boolean validaPostsSimultaneos()`

7. Considere a classe `Lampada` desenvolvida na Ficha 3 e

- altere a classe `Lampada` para que passe a conter um identificador único por cada lâmpada que seja criada.
- crie agora um novo tipo de lâmpada, a `LampadaLed`, que acrescenta à especificação anteriormente feita a informação relativa à potência de mesma (um valor inteiro que regista os watt de cada lâmpada), a intensidade luminosa, os lumem (um valor inteiro), e acrescenta também um novo algoritmo de cálculo do consumo em que no modo ON o consumo é reduzido por um factor multiplicativo de $1/watts$ e no modo ECO em $1.25/watts$.

Crie agora uma classe `CasaInteligente` em que se guarda a informação das lâmpadas (normais ou com led) que existem na casa e que possui também uma relação em que se identifica para cada divisão da casa (ex: cozinha, sala, etc.) as lâmpadas que existem nessas mesmas divisões.

- (a) Desenhe o diagrama de classe com a arquitectura proposta. Para saber quais os métodos a considerar leia todo o enunciado da questão.

- (b) Desenvolva esta classe codificando os métodos habituais e ainda os métodos seguintes:
- i. método que permite especificar o layout da casa, fornecendo o nome das divisões, `public void setUpLayout(Collection<String> nomesDivisooes)`
 - ii. método que adiciona uma lâmpada à casa numa determinada divisão, `public void addLampada(Lampada l, String divisao)`
 - iii. método que determina se a casa tem uma determinada divisão, `public boolean existeDivisao(String divisao)`
 - iv. método que devolve as lâmpadas existentes numa divisão, `public Collection lampadasEmDivisao(String divisao)`
 - v. método que coloca as lâmpadas de uma divisão num estado, `public void setEstado(String divisao, Modo estado)` (recorde a implementação de `Lampada` anteriormente fornecida)
 - vi. método que determina a divisão que tem mais lâmpadas, `public String maisIluminada()`
 - vii. método que determina o consumo total da casa, `public double consumoCasa()`
 - viii. método que remove uma lâmpada da casa dado o seu identificador, `public Lampada removeLamp(String ident)`, sabendo que se assume que o identificador da lâmpada é uma `String` (no seu caso poderá ser de tipo diferente) e que se devolve a lâmpada a remover (ou `null` caso ela não exista).
 - ix. método que determina a `LampadaLed` que menos consumiu, `public LampadaLed maisEconomica()`
 - x. métodos para gravar em modo texto e modo objecto a informação da `CasaInteligente`.