POO (MiEI/LCC)

2020/2021

Ficha Prática #05

Map<K,V>

Conteúdo

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 `Map<K,V>`**
Inserção de elementos	put(K k, V v); putAll(Map<?extends K, ?extends V> m);
Remoção de elementos	remove(Object k);

Consulta e comparação de conteúdos	V get(Object k);
	boolean containsKey(Object k);
	boolean isEmpty();
	boolean containsValue(Object v);
	int size();

Criação de Iteradores	Set<K> keySet(); Collection<V> values(); Set<Map.Entry<K,V>> entrySet();
Outros	boolean equals(Object o); Object clone()

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 matricula, 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 matricula;
- Método que altera o tempo disponível de um lugar, para uma dada matricula;
- 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 matricula;

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<String> 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(String codEnc)`

iv. método que remove uma encomenda dado o seu código, `public void removeEncomenda(String codEnc)`

v. método que determina a encomenda (identificada pelo código) com mais produtos encomendados, `public String encomendaComMaisProdutos()`

vi. método que determina todas as encomendas em que um determinado produto, identificado pelo código, está presente, `public Set<String> encomendasComProduto(String codProd)`

vii. método que determina todas as encomendas com data posterior a uma data fornecida como parâmetro, `public Set<String> encomendasAposData(LocalDate d)`

viii. método que devolve uma ordenação, por ordem decrescente de valor de encomenda, de todas as encomendas do sistema, `public Set<Encomenda> encomendasValorDecrescente()`

ix. método que calcula um map em que associa cada código de produto à lista das encomendas em que foi referida, `public Map<String,List<String>> 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 arquitectura proposta. Para saber quais os métodos a considerar leia todo o enunciado da questão.

(b) Complete a classe definindo:

i. Os construtores `Grafo()` e `Grafo(Map<Integer, Set<Integer>> adj)`.

ii. `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.

iii. `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).

iv. `boolean isSource(Integer v)` – método que determina se um vértice é um *source* (não existem arcos a entrar nele).

v. `int size()` – método que calcula o tamanho do grafo (o tamanho de um grafo com n vértices e m arcos é $n + m$).

vi. `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.

vii. `List<String> getCaminho(Integer vOrig, Integer vDest)` – método que calcula o caminho entre dois vértices. O método deverá devolver `null` caso não exista caminho.

viii. `Set<Map.Entry<Integer, Integer>> fanOut (Integer v)` – método que calcula o conjunto de todos os arcos que saem de um vértice.

ix. `Set<Map.Entry<Integer, Integer>> fanIn(Integer v)` – método que calcula o conjunto de todos os arcos que entram num vértice.

x. `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()`