POO (MiEI/LCC)

2015/2016

Ficha Prática #06

Map<K,V>

Conteúdo

1 Objectivos

- Aprender a definir testes unitários.
- 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 8.

3 Exercícios

1. Considere as classes `Cache` e `Geocacher`, que representam o jogo *Geocaching*. Uma cache possui um conjunto de características bem definidas, e um geocacher possui um conjunto de caches (as caches que descobriu). Na implementação fornecida de ambas as classes, **é sabido que existem erros de implementação**, quer funcionais, quer no desrespeito pelo encapsulamento. **Desenvolva uma classe de testes unitários**, onde seja feito um teste a cada método de ambas as classes, para identificar os erros de implementação. **Proceda à correcção dos erros** encontrados.

2. 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 alocado */
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.

   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 um novo 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;
   - Implemente testes unitários para testar cada uma das alíneas anteriores;

3. Crie uma classe `Paises` que estabeleça uma correspondência entre o nome de um dado país e a informação sobre a sua

capital (`FichaDeCapital`), designadamente: nome da cidade, população, número de veículos, salário médio (real) e custo de vida mensal médio (real). Implemente os seguintes métodos na classe **Paises**:

- Determinar o número total de países;
- Devolver os nomes dos países com capitais com população acima de um valor dado. Implemente com **iterador interno** e **iterador externo**;
- Dado o nome de um país, devolver ficha completa da sua capital;
- Alterar a população da capital de um dado país;
- Inserir a informação de um novo país;
- Criar uma listagem com os nomes de todas as capitais registadas. Implemente com **iterador interno** e **iterador externo**;
- Determinar o somatório de todas as populações das capitais. Implemente com **iterador interno** e **iterador externo**;
- Dada um `Map` de nome de país para `FichaDeCapital`, para cada país que exista na lista de países alterar a sua ficha de capital e para cada país novo inserir a sua informação.
- Dada um conjunto de nomes de países, remover as suas fichas de capital;
- Implemente testes unitários para testar cada uma das alíneas anteriores;

4. Considerando a classe `ContaPrazo` anteriormente desenvolvida, crie agora uma classe `Banco` que associe a cada código de conta uma **ContaPrazo**. A classe **Banco** deverá implementar métodos que realizem as seguintes operações:

   - Inserir uma nova conta;
   - Determinar o conjunto de códigos das contas pertencentes a dado titular. Implemente com **iterador interno** e **iterador externo**;
   - O mesmo que o anterior mas para um conjunto de nomes de titulares;
   - Determinar os códigos das contas com capital superior a um valor dado;
   - Criar um `Map` das contas com taxa de juro superior a um valor dado. Implemente com **iterador interno** e **iterador externo**;

- Conjunto dos códigos das contas que vencem juros no dia de hoje. Implemente com **iterador interno** e **iterador externo**;
- Dada uma lista de códigos de contas incrementar as suas taxas de juro de um valor `X`. Implemente com **iterador interno** e **iterador externo**;
- Devolver os nomes de todos os titulares de contas. Implemente com **iterador interno** e **iterador externo**;
- Criar um `Map` que associe a cada nome de titular existente o valor total do capital investido nas suas várias contas (use métodos auxiliares);
- Implemente testes unitários para testar cada uma das alíneas anteriores;

5. Cada e-mail recebido numa dada conta de mail é guardado contendo o endereço de quem o enviou, a data de envio, a data de recepção, o assunto e o texto do mail (não se consideram anexos, etc.). Crie uma classe `MailMap` que associe a cada endereço de envio todos os mails recebidos (cf. classe `Mail`) e implemente as seguintes operações:

   - Determinar o total de endereços a partir dos quais se recebeu mail;
   - Guardar um novo mail recebido;
   - Determinar quantos mails têm por origem um dado endereço. Implemente com **iterador interno** e **iterador externo**;
   - Criar uma lista contendo todos os endereços que enviaram mails contendo no seu assunto uma lista de palavras dada como parâmetro. Implemente com **iterador interno** e **iterador externo**;
   - O mesmo que a questão anterior, mas criando um conjunto contendo os mails;
   - Eliminar todos os e-mails recebidos antes de uma data que é dada como parâmetro. Implemente com **iterador interno** e **iterador externo**;
   - Criar uma lista dos endereços que hoje enviaram mails. Implemente com **iterador interno** e **iterador externo**;
   - Dada uma lista de palavras, eliminar todos os mails de um dado endereço que no seu assunto contenham uma qualquer destas (anti-spam). Implemente com **iterador interno** e **iterador externo**;

- Eliminar todos os mails anteriores a uma data dada. Implemente com **iterador interno** e **iterador externo**;
- Criar uma listagem com todos os endereços de mail oriundos de Portugal. Implemente com **iterador interno** e **iterador externo**;
- Implemente testes unitários para testar cada uma das alíneas anteriores;