

POO (MiEI/LCC)

2021/2022

Ficha Prática #04

List<E>

Conteúdo

1	Objectivos	3
2	API essencial	3
2.1	ArrayList	3
2.2	Iteradores externos	3
2.3	Iteradores internos	4
2.4	Expressões lambda	4
3	Exercícios	4

1 Objectivos

- Aprender a trabalhar com ArrayList.
- Aprender a trabalhar com iteradores externos.
- Aprender a trabalhar com iteradores internos.

2 API essencial

2.1 ArrayList

Esta tabela lista alguns dos métodos mais relevantes de ArrayList. Consulte a API oficial do Java para obter mais informação sobre os métodos.

Categoria de Métodos	API de ArrayList<E>
Construtores	<code>new ArrayList<>(); new ArrayList<>(int dim); new ArrayList<>(Collection)</code>
Inserção de elementos	<code>add(E o); add(int index, E o)</code>
Remoção de elementos	<code>remove(Object o); remove(int index); removeAll(Collection); retainAll(Collection); removeIf(Predicate)</code>
Consulta e comparação de conteúdos	<code>E get(int index); int indexOf(Object o); int lastIndexOf(Object o); boolean contains(Object o); boolean isEmpty(); boolean containsAll(Collection); int size()</code>
Iteradores externos	<code>Iterator iterator()</code>
Iteradores internos	<code>Stream stream(); forEach(Consumer)</code>
Modificação	<code>set(int index, E elem); clear(); replaceAll(UnaryOperator)</code>
Subgrupo	<code>List sublist(int de, int ate)</code>
Conversão	<code>Object[] toArray()</code>
Outros	<code>boolean equals(Object o); boolean isEmpty()</code>

2.2 Iteradores externos

Esta tabela lista alguns dos métodos mais relevantes de Iterator. Consulte a API oficial do Java e os apontamentos teóricos para obter mais informação sobre os métodos e sua utilização.

Categoria de Métodos	API de Iterator
Consulta	<code>hasNext(); next()</code>
Alteração da colecção	<code>remove()</code>

2.3 Iteradores internos

Esta tabela lista alguns dos métodos mais relevantes de `Stream`. Consulte a API oficial do Java e os apontamentos teóricos para obter mais informação sobre os métodos e sua utilização.

Categoria de Métodos	API de Iterator
Consulta	<code>allMatch(Predicate)</code> ; <code>anyMatch(Predicate)</code> ; <code>noneMatch(Predicate)</code> ;
<i>Folds</i>	<code>count()</code> ; <code>collect(Collector)</code> ; <code>reduce(E, BinaryOperator)</code> ; <code>toArray()</code>
Alteração	<code>map(Function)</code> ; <code>Filter(Predicate)</code>

2.4 Expressões lambda

Notação para expressões lambda:

(parametros) -> {corpo da expressão}

3 Exercícios

Resolva os exercícios que requeiram a utilização de iteradores, quer com iteradores internos, quer com iteradores externos, por forma a comparar as duas abordagens.

1. Uma *Stack* (ou pilha) é uma estrutura linear do tipo LIFO ("last in first out"), ou seja, o último elemento a ser inserido é o primeiro a ser removido. Uma *stack* possui assim apenas um extremo para inserção e para remoção. Implemente uma *Stack* de *Strings*, com as usuais operações sobre *stacks*:

- (a) `String top()`: que determina o elemento no topo da *stack*;
- (b) `void push(String s)`: insere no topo;
- (c) `void pop()`: remove o elemento do topo da *stack*, se esta não estiver vazia;
- (d) `boolean empty()`: determina se a *stack* está vazia;
- (e) `int length()`: determina o comprimento da *stack*;

2. Considere o exercício da *Encomenda* que realizou na Ficha 3. Crie agora uma nova implementação dessa classe, *EncEficiente*, que exiba o mesmo comportamento e que tenha como estrutura de dados interna um `ArrayList<LinhaEncomenda>`.

Considere também que queremos que nesta nova implementação tenha em atenção que o número de encomenda deva ser atribuído de forma sequencial (e não necessita de ser enviado como parâmetro do construtor de *Encomenda*). Recorda-se o enunciado do exercício:

Para uma **Encomenda** guardam-se os seguintes atributos:

- *nome do cliente*
- *número fiscal do cliente*
- *morada do cliente*
- *número da encomenda*
- *data da encomenda*
- *as linhas da encomenda*

- (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, além dos métodos usuais, os métodos:
 - i. métodos usuais de acesso e alteração das variáveis de instância
 - ii. método que determina o valor total da encomenda, `public double calculaValorTotal()`
 - iii. método que determina o valor total dos descontos obtidos nos diversos produtos encomendados, `public double calculaValorDesconto()`
 - iv. método que determina o número total de produtos a receber, `public int numeroTotalProdutos()`
 - v. método que determina se um produto vai ser encomendado, `public boolean existeProdutoEncomenda(String refProduto)`
 - vi. método que adiciona uma nova linha de encomenda, `public void adicionaLinha(LinhaEncomenda linha)`
 - vii. método que remove uma linha de encomenda dado a referência do produto, `public void removeProduto(String codProd)`
3. Considere a classe *Lampada* da Ficha 3. Crie agora a classe *CasaInteligente* que agrega uma lista de lâmpadas e faz a sua gestão.
- (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 nesta classe, além dos métodos usuais, os seguintes métodos:
 - i. `public void addLampada(Lampada l)`, que adiciona mais uma lâmpada à casa
 - ii. `public void ligaLampadaNormal(int index)`, que liga no modo de consumo máximo a lâmpada que está na posição indicada
 - iii. `public void ligaLampadaEco(int index)`, que liga no modo de consumo económico a lâmpada que está na posição indicada

- iv. `public int` `qtEmEco()`, que determina quantas lâmpadas é que estão ligadas em modo económico.
 - v. `public void` `removeLampada(int index)`, que remove a lâmpada da posição passada como parâmetro
 - vi. `public void` `ligaTodasEco()` e `public void` `ligaTodasMax()`, que liga todas as lâmpadas da casa respectivamente em modo Eco e em modo de consumo máximo
 - vii. `public double` `consumoTotal()`, que determina o consumo total da casa
 - viii. `public` `Lampada` `maisGastadora()`, que determina a lâmpada que mais consumiu até à data
 - ix. `public` `Set<Lampada>` `podiumEconomia()`, que devolve as três lâmpadas mais económicas da casa
 - x. `public void` `reset()`, que efectua o reset do contador parcial de consumo em todas as lâmpadas.
4. Considere que pretende desenvolver uma classe que implemente um comportamento similar ao da timeline do facebook. Genericamente a timeline pode ser vista como sendo uma lista de *posts* (classe `FBPost`), em que cada post é caracterizado pelos seguintes atributos:
- um identificador
 - o nome do utilizador que criou o post
 - o instante de criação do post
 - o conteúdo do post
 - o número de likes
 - uma lista de comentários associados ao post
- (a) Desenhe o diagrama de classe com a arquitectura proposta. Para saber quais os métodos a considerar leia todo o enunciado da questão.
- (b) A classe `FBFeed` implementa a timeline e deve implementar, além dos métodos usuais, os métodos que permitam:
- i. Determinar o número de posts de um user `public int` `nrPosts(String user)`
 - ii. Determinar a lista de posts de um user `public` `List<FBPost>` `postsOf(String user)`
 - iii. Determinar a lista de posts de um user num determinado intervalo de tempo `public` `List<FBPost>` `postsOf(String user, LocalDateTime inicio, LocalDateTime fim)`
 - iv. Obter um post dado o seu identificador `public` `FBPost` `getPost(int id)`

- v. Inserir um comentário num post `public void comment(FBPost post, String comentario)`
- vi. Inserir um comentário num post `public void comment(int postid, String comentario)`
- vii. Adicionar um like a um post `public void like(FBPost post)`
- viii. Adicionar um like a um post `public void like(int postid)`
- ix. Determinar a lista dos 5 posts (identificador) com mais comentários `public List<Integer> top5Comments()`. Desenvolva uma versão com iteradores externos e outra com iteradores internos.

5. Desenvolva uma classe `PedidodeSuporte` que permita registar um pedido de suporte informático. Este pedido deve registar quem efetuou o pedido, o instante em que o pedido foi submetido, o assunto a ser tratado e a descrição do pedido. Deve ser ainda possível registar quem tratou o pedido, o instante em que foi concluído e informação sobre o mesmo a ser transmitida ao cliente.

Desenvolva também a classe `SistemadeSuporte` que é a plataforma onde são guardados e geridos os pedidos de suporte. Os pedidos de suporte são atendidos pela ordem pela qual chegam ao sistema.

- (a) Desenhe o diagrama de classe com a arquitectura proposta. Para saber quais os métodos a considerar leia todo o enunciado da questão.
- (b) Além das operações normais das classes Java, codifique os seguintes métodos na classe `SistemadeSuporte`:
 - i. `public void inserePedido(PedidodeSuporte pedido)`, que insere um novo pedido de suporte no sistema
 - ii. `public PedidodeSuporte procuraPedido(String user, LocalDateTime data)`, que procura um pedido de suporte dada a identificação de quem o criou e a instante em que foi criado
 - iii. `public void resolvePedido(PedidodeSuporte pedido, String tecnico, String info)`, que resolve um pedido de suporte indicando o técnico e a informação relacionada com o pedido. Este método é responsável por preencher a informação de data/hora de resolução do pedido
 - iv. `public List<PedidodeSuporte> resolvidos()`, que devolve todos os pedidos já resolvidos. Crie uma versão com iteradores externos e outra com iteradores internos.
 - v. `public String colaboradorTop()`, que devolve o colaborador que resolveu mais pedidos de suporte. Crie uma versão com iteradores externos e outra com iteradores internos.

- vi. `public List<PedidodeSuporte>`
`resolvidos(LocalDateTime inicio, LocalDateTime fim)`, que devolve os pedidos resolvidos num determinado período de tempo. Crie uma versão com iteradores externos e outra com iteradores internos.
- vii. `public double` `tempoMedioResolucao()`, que calcula a média em minutos do tempo de resolução dos pedidos. Crie uma versão com iteradores externos e outra com iteradores internos.
- viii. `public double` `tempoMedioResolucao(LocalDateTime inicio, LocalDateTime fim)`, que calcula a média em minutos do tempo de resolução dos pedidos concluídos num determinado período
- ix. `public PedidodeSuporte` `pedidoMaisLongo()`, que devolve o pedido de suporte que demorou mais tempo a ser resolvido
- x. `public PedidodeSuporte` `pedidoMaisLongo(LocalDateTime inicio, LocalDateTime fim)`, que devolve, dos pedidos resolvidos num determinado intervalo, o que demorou mais tempo a ser resolvido
- xi. `public PedidodeSuporte` `pedidoMaisCurto()`, que devolve o pedido de suporte que demorou menos tempo a ser resolvido
- xii. `public PedidodeSuporte` `pedidoMaisCurto(LocalDateTime inicio, LocalDateTime fim)`, que devolve, dos pedidos resolvidos num determinado intervalo, o que demorou menos tempo a ser resolvido