

# POO (MiEI/LCC)

2015/2016

Ficha Prática #05

ArrayList<E>  
(v. 1.1)

## Conteúdo

<b>1</b>	<b>Objectivos</b>	<b>3</b>
<b>2</b>	<b>API essencial</b>	<b>3</b>
2.1	ArrayList . . . . .	3
2.2	Iteradores externos . . . . .	3
2.3	Iteradores internos . . . . .	4
2.4	Expressões lambda . . . . .	4
<b>3</b>	<b>Exercícios</b>	<b>4</b>

## 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 Java8 para obter mais informação sobre os métodos.

<b>Categoria de Métodos</b>	<b>API de ArrayList&lt;E&gt;</b>
Construtores	<code>new ArrayList&lt;&gt;(); new ArrayList&lt;&gt;(int dim); new ArrayList&lt;&gt;(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 Java8 e os apontamentos teóricos para obter mais informação sobre os métodos e sua utilização.

<b>Categoria de Métodos</b>	<b>API de Iterator</b>
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 Java8 e os apontamentos teóricos para obter mais informação sobre os métodos e sua utilização.

Categoria de Métodos	API de <code>Iterator</code>
Consulta	<code>allMatch(Predicate); anyMatch(Predicate); noneMatch(Predicate);</code>
<i>Folds</i>	<code>count(); collect(Collector); reduce(E, BinaryOperator); toArray()</code>
Alteração	<code>map(Function); 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. Compreensão do funcionamento de `ArrayList<E>`  
Crie no BlueJ, uma pasta de projecto de nome `TesteArrayList` e importe para tal pasta a classe `Ponto2D` anteriormente desenvolvida. Em seguida, usando `Tools/Use Library Class...`, crie uma instância vazia de um `ArrayList<String>` e outra de um `ArrayList<Ponto2D>`. De seguida, com ambas as instâncias de `ArrayList` criadas, invoque os métodos da API de `ArrayList<E>`, Cf. `add()`, `get()`, `addAll()`, `remove()`, `removeAll()`, `size()`, `indexOf()`, `contains()`, `retainAll()`, etc., e conclua sobre a semântica de cada método usando `INSPECT` após a sua invocação.
2. Desenvolva uma classe `Playlist` que represente uma lista de músicas a serem reproduzidas, representadas como um `ArrayList<Faixa>`. Desenvolva, para além dos construtores e métodos usuais (`get`, `set`), métodos que implementem as seguintes funcionalidades:
  - Determinar o número total de faixas na playlist:  

```
public int numFaixas()
```

- Adicionar uma nova no final da playlist:  
`public void addFaixa(Faixa f)`
  - Remover uma faixa da playlist:  
`public void removeFaixa(Faixa m)`
  - Dado um List de Faixa, juntar tais faixas à playlist receptora. Implementar com iterador externo:  
`public void adicionar(List<Faixa> faixas)`  
 e iterador interno:  
`public void adicionarF(List<Faixa> faixas)`
  - Determinar quantas faixas têm uma classificação superior à Faixa dada como parâmetro. Implementar com iterador externo:  
`public int classificacaoSuperior(Faixa f)`  
 e iterador interno:  
`public int classificacaoSuperiorF(Faixa f)`
  - Determinar se alguma faixa tem duração superior ao valor passado como parâmetro. Implementar com iterador externo:  
`public boolean duracaoSuperior(double d)`  
 e iterador interno:  
`public boolean duracaoSuperiorF(double d)`
  - Devolver uma cópia listagem de músicas, em que o valor da sua classificação seja alterado para o valor passado como parâmetro. Implementar com iterador externo:  
`public List<Faixa> getCopiaFaixas(n)`  
 e iterador interno:  
`public List<Faixa> getCopiaFaixasF(n)`
  - Determinar a duração total da playlist. Implementar com iterador externo:  
`public double duracaoTotal()`  
 e iterador interno:  
`public double duracaoTotalF()`
  - Remover as faixas de determinado autor. Implementar com iterador externo:  
`public void removeFaixas(String autor)`  
 e iterador interno:  
`public void removeFaixasF(String autor)`
  - Não esquecer os métodos `equals()`, `toString()` e `clone()`.
3. Uma ficha de informação de um país, `FichaPais`, contém 3 atributos: nome do país, continente e população (real, em milhões). Crie uma classe `ListaPaises` que permita criar listas de

FichaPais, por uma ordem qualquer, e implemente os seguintes métodos:

- Adicionar um novo país:  
`public void adicionar(String nome, String continente, double populacao)`
- Determinar o número total de países:  
`public int numPaíses()`
- Determinar o número de países de um continente dado. Implementar com iterador externo:  
`public int numPaíses(String continente)`  
e iterador interno:  
`public int numPaísesF(String continente)`
- Dado o nome de um país, devolver a sua ficha completa, caso exista. Implementar com iterador externo:  
`public FichaPais getFicha(String nome)`  
e iterador interno:  
`public FichaPais getFichaF(String nome)`
- Criar uma lista com os nomes dos países com uma população superior a um valor dado. Implementar com iterador externo:  
`public List<String> nomesPaíses(double valor)`  
e iterador interno:  
`public List<String> nomesPaísesF(double valor)`
- Determinar a lista com os nomes dos continentes dos países com população superior a dado valor. Implementar com iterador externo:  
`public List<String> nomesContinentes(double valor)`  
e iterador interno:  
`public List<String> nomesContinentesF(double valor)`
- Determinar o somatório das populações de dado continente. Implementar com iterador externo:  
`public double somatorio(String continente)`  
e iterador interno:  
`public double somatorioF(String continente)`
- Dada uma lista de FichaPais, para cada país que exista na lista de países<sup>1</sup> alterar a sua população com o valor na ficha; caso não exista inserir a ficha na lista. Implementar com iterador externo:  
`public void actualiza(ArrayList<FichaPais> fichas)`

---

<sup>1</sup>Comparando países com base no nome.

e iterador interno:

```
public void atualizaF(ArrayList<FichaPais> fichas)
```

- Dada uma lista de nomes de países, remover as suas fichas. Implementar com iterador externo:

```
public void remove(ArrayList<String> paises)
```

e iterador interno:

```
public void removeF(ArrayList<String> paises)
```

4. 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 nomes, com as usuais operações sobre stacks:

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

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 a classe `Mail` que represente cada um dos mails recebidos. Em seguida crie uma classe designada `MailList` que permita guardar todos os actuais e-mails existentes numa dada conta, e implemente as seguintes operações sobre os mesmos:

- Determinar o total de mails guardados.

```
public int totalEmails()
```

- Guardar um novo mail recebido.

```
public void adicionarEmail(Mail m)
```

- Determinar quantos mails têm por origem um dado endereço. Implementar com iterador externo:

```
public int from(String endereco)
```

e iterador interno:

```
public int fromF(String endereco)
```

- Criar uma lista contendo os índices dos mails que no assunto contêm uma palavra dada como parâmetro (qualquer que seja a posição desta). Implementar com iterador

externo:

```
public List<Integer> comAssunto(String s)
```

e iterador interno:

```
public List<Integer> comAssuntoF(String s)
```

- O mesmo que a questão anterior, mas criando uma lista contendo tais mails. Implementar com iterador externo:

```
public List<Email> comAssuntoL(String s)
```

e iterador interno:

```
public List<Email> comAssuntoLF(String s)
```

- Eliminar todos os e-mails recebidos antes de uma data que é dada como parâmetro. Implementar com iterador externo:

```
public void eliminarRecebidos(GregorianCalendar data)
```

e iterador interno:

```
public void eliminarRecebidosF(GregorianCalendar data)
```

- Criar uma lista dos mails do dia. Implementar com iterador externo:

```
public List<Mail> emailsDoDia()
```

e iterador interno:

```
public List<Mail> emailsDoDiaF()
```

- Dada uma lista de palavras, eliminar todos os mails que no seu assunto contenham uma qualquer destas (anti-spam). Implementar com iterador externo:

```
public void antiSpam(List<String> palavras)
```

e iterador interno:

```
public void antiSpamF(List<String> palavras)
```

- Eliminar todos os mails anteriores a uma data dada. Implementar com iterador externo:

```
public void eliminar(GregorianCalendar data)
```

e iterador interno:

```
public void eliminarF(GregorianCalendar data)
```