

# **Biblioteca de estrutura de dados**

# Bibliografia

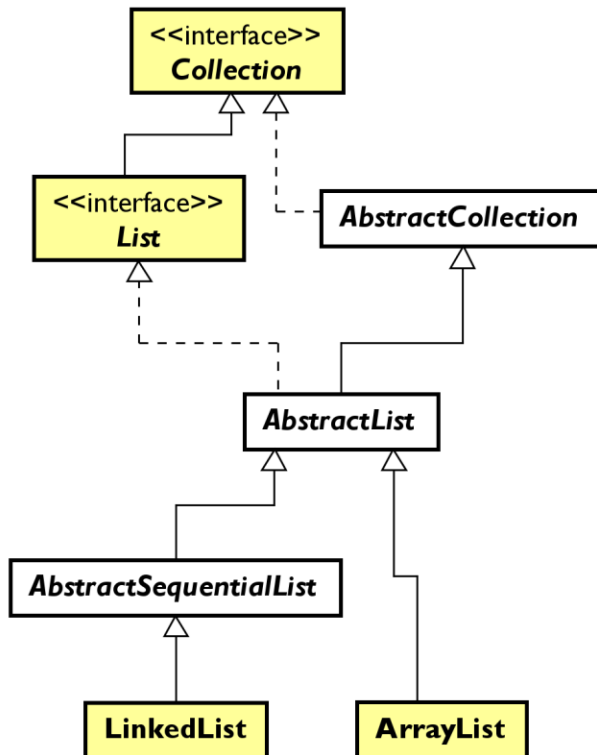
- Documentação Oficial da Java Collections Framework.  
<https://docs.oracle.com/javase/8/docs/technotes/guides/collections/overview.html>
- HORSTMANN, Cay S. **Big Java**. Porto Alegre : Bookman, 2004. xi, 1125 p, il., 1 CD-ROM. Acompanha CD-ROM.
- DEITEL, Paul J; DEITEL, Harvey M. **Java: como programar**. 8. ed. São Paulo: Pearson, 2010. xxix, 1144 p, il.
- LIANG, Y. Daniel. **Introduction to Java Programming and Data Structures**. 11ª ed. Person, 2019. 1232p.

# Introdução

- Uma “Estruturas de dados” é uma forma de armazenar e organizar dados na memória do computador.
- “Escolher a melhor estrutura de dados e algoritmos para uma tarefa particular é uma das chaves para o desenvolvimento de software de alta performance” (LIANG, 2019)
- Java possui uma biblioteca para manipular estruturas de dados, denominado de **Java Collection Framework (JCF)**
  - Embora seja chamado de “framework”, a JCF é uma biblioteca
- As classes de manipulação de estrutura de dados estão agrupadas em dois grandes grupos: **Coleções** e **Mapas**

# Coleções

# Principais coleções



ArrayList implementa uma lista estática  
LinkedList implementa uma lista dinâmica

## Diferenças

Operação	ArrayList	LinkedList
get(index)	Extremamente rápido	Lento
add(E)	Rápido na maior parte das vezes	Extremamente rápido
add(index, E)	Lento	Lento
remove(index, E)	Lento	Lento

# Interface Collection

E

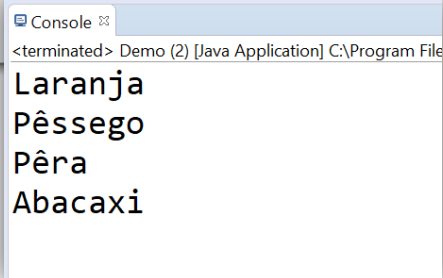
<<interface>> <i>Collection</i>
<b>+ size() : int</b> <b>+ isEmpty() : boolean</b> <b>+ contains(o : Object) : boolean</b> <b>+ iterator() : Iterator&lt;E&gt;</b> <b>+ toArray() : Object[]</b> <b>+ toArray(a : T[]) : T[]</b> <b>+ add(e : E) : boolean</b> <b>+ remove(o : Object) : boolean</b> <b>+ containsAll(c : Collection&lt;?&gt;) : boolean</b> <b>+ addAll(c : Collection&lt;E&gt;) : boolean</b> <b>+ removeAll(c : Collection&lt;?&gt;) : boolean</b> <b>+ retainAll(c : Collection&lt;?&gt;) : boolean</b> <b>+ clear() : void</b> <b>+ equals(o : Object) : boolean</b> <b>+ hashCode() : int</b>

Método	Descrição
size()	Retorna a quantidade de elementos armazenados
isEmpty()	Retorna true se a estrutura estiver vazia
contains(Object)	Retorna true se o objeto está armazenado na estrutura
iterator()	Retorna um objeto que permite percorrer a estrutura
toArray()	Devolve um vetor com os dados da estrutura
add(E)	Adiciona o objeto na estrutura
remove(Object)	Remove o objeto da estrutura
containsAll(Collection)	Retorna true se todos os elementos pertencerem à coleção
addAll(Collection)	Adiciona todos os elementos na coleção
removeAll	Remove todos os elementos que pertencem à coleção
retainAll(Collection)	Mantém apenas os elementos que estão na coleção informada como parâmetro
clear()	Remove os elementos da coleção

# Exemplo

- O **ArrayList** implementa indiretamente **Collection**:

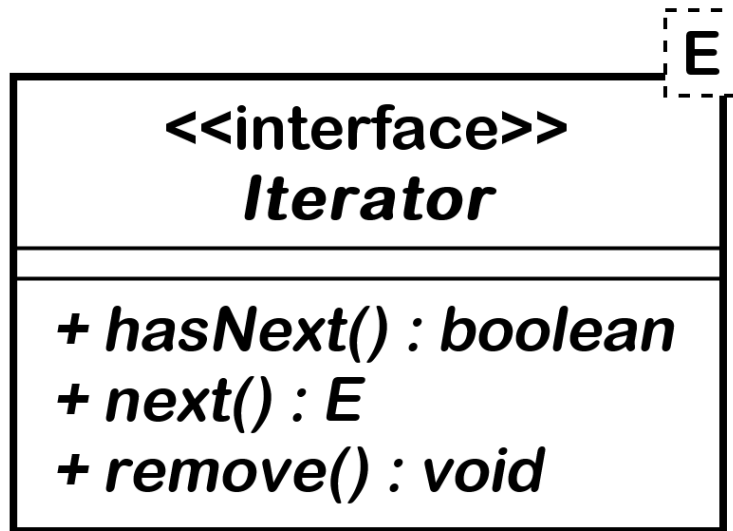
```
Collection<String> frutas = new ArrayList<>();  
frutas.add("Laranja");  
frutas.add("Morango");  
frutas.add("Pêssego");  
frutas.add("Pêra");  
frutas.add("Abacaxi");  
frutas.remove("Morango");  
  
for (String fruta: frutas) {  
    System.out.println(fruta);  
}  
  
frutas.clear();
```



```
<terminated> Demo (2) [Java Application] C:\Program File  
Laranja  
Pêssego  
Pêra  
Abacaxi
```

# O método iterator()

- A interface **Collection** prevê que todas as classes concretas que a implemente, retorne um *iterador*, através do método **iterator()**.



Um *iterador* é um objeto que possibilita percorrer uma estrutura de dados.

Método	Descrição
hasNext()	Retorna true se tem objetos na coleção ainda não lidos
next()	Retorna um objeto da coleção
remove()	Remove o último objeto lido pelo iterador (através de next()), da coleção



# Utilidade do iterador

Remover as frutas que começam com “P”:

```
Collection<String> frutas = new ArrayList<>();  
frutas.add("Laranja");  
frutas.add("Morango");  
frutas.add("Pêssego");  
frutas.add("Pêra");  
frutas.add("Abacaxi");  
  
for (String fruta : frutas) {  
    if (fruta.startsWith("P")) {  
        frutas.remove(fruta);  
    }  
}
```

Não é seguro, pois lança a exceção:

```
Exception in thread "main" java.util.ConcurrentModificationException  
at java.base/java.util.ArrayList$Itr.checkForComodification(ArrayList.java:1042)  
at java.base/java.util.ArrayList$Itr.next(ArrayList.java:996)  
at teste.Demo.main(Demo.java:18)
```

# Utilidade do iterador

- Usando o iterador:

```
Collection<String> frutas = new ArrayList<>();
frutas.add("Laranja");
frutas.add("Morango");
frutas.add("Pêssego");
frutas.add("Pêra");
frutas.add("Abacaxi");

Iterator<String> iterador = frutas.iterator();
while (iterador.hasNext()) {
    String fruta = iterador.next();
    if (fruta.startsWith("P"))
        iterador.remove();
}

for (String fruta : frutas) {
    System.out.println(fruta);
}
```

```
Console
<terminated> Demo (2) [Java Application]
Laranja
Morango
Abacaxi
```

# Interface List

<b>&lt;&lt;interface&gt;&gt; List</b>
<b>+ get(index : int) : E</b> <b>+ set(index : int, element : E) : E</b> <b>+ add(index : int, element : E) : void</b> <b>+ indexOf(o : Object) : int</b> <b>+ lastIndexOf(o : Object) : int</b> <b>+ listIterator() : ListIterator&lt;E&gt;</b> <b>+ subList(fromIndex : int, toIndex : int) : List&lt;E&gt;</b>

Método	Descrição
get(int)	Retorna o objeto que ocupa a posição informada
set()	Altera o objeto armazenado na posição indicada
add()	Acrescenta um objeto na posição indicada
indexOf()	Procura por um objeto
lastIndexOf()	Retorna a posição da última ocorrência do objeto
listIterator()	Devolve um iterador que suporta navegação bidirecional
subList()	Retorna uma sub-lista

<b>&lt;&lt;interface&gt;&gt; ListIterator</b>
<b>+ hasNext() : boolean</b> <b>+ next() : E</b> <b>+ hasPrevious() : boolean</b> <b>+ previous() : E</b> <b>+ nextIndex() : int</b> <b>+ previousIndex() : int</b> <b>+ remove() : void</b>

Interface que prevê a navegação bidirecional