

## COLEÇÕES

Alexandre Mendonça Fava  
[alexandre.fava@udesc.br](mailto:alexandre.fava@udesc.br)

---

Universidade do Estado de Santa Catarina – UDESC  
Programa de Pós-graduação em Computação Aplicada – PPGCA

# Roteiro

- ☐ Definição
  - ☐ Coleção
  - ☐ Arranjo
- ☐ Métodos
- ☐ forEach
- ☐ Modificadores de Acesso
- ☐ Resumo
- ☐ Referências

# Definição

## O que é coleção?

DEITEL, Paul. Java: como programar.

Alguns exemplos de coleções são suas músicas preferidas

armazenadas no smartphone ou media player, sua lista de contatos, as cartas que você mantém em um jogo de cartas, os membros do seu time favorito e os cursos que você faz simultaneamente na escola.

# Definição

[https://pt.wikipedia.org/wiki/Coleção\\_\(tipo\\_de\\_dado\\_abstrato\)](https://pt.wikipedia.org/wiki/Coleção_(tipo_de_dado_abstrato))

## ☰ Coleção (tipo de dado abstrato) 🌐 7 línguas ▼

Artigo **Discussão**

Ferramentas ▼

Origem: Wikipédia, a enciclopédia livre.

Em **ciência da computação**, uma **coleção** ou **contêiner** é um agrupamento de algum número variável de itens de dados que têm algum significado compartilhado para o problema que está sendo resolvido e precisam ser operados juntos, de alguma forma controlada. Geralmente, os itens de dados serão do mesmo tipo.

# Definição

Origem: Wikipédia, a enciclopédia livre.

Em [ciência da computação](#), uma **coleção** ou **contêiner** é um agrupamento de algum número variável de itens de dados que têm algum significado compartilhado para o problema que está sendo resolvido e precisam ser operados juntos, de alguma forma controlada. Geralmente, os itens de dados serão do mesmo tipo.

## Diferença dos Arrays [\[ edit \]](#)

Coleções e arrays são semelhantes no sentido de que ambos contêm referências a objetos e podem ser gerenciados como um grupo. No entanto, diferentemente dos arrays, coleções não precisam receber um determinado tamanho quando instanciados. Coleções podem aumentar e diminuir de tamanho automaticamente quando objetos são adicionados ou removidos.

# Definição

**Coleções**  
(collection)

**Arranjos**  
(arrays)

São um conjunto de dados  $n$ -dimensional

**Dados:** objetos e/ou literais

$\forall n \mid n \in \mathbb{N}^*$

Para todo  $n$  tal que  $n$  pertença aos números naturais

*e.g.:* não existe arranjo com 2,5 de tamanho

# Definição

## Coleções

(*collection*)

- ✓ Classe
- ✓ Tamanho dinâmico
- ✓ Mais recursos

## Arranjos

(*arrays*)

- ✓ Estrutura
- ✓ Tamanho estático
- ✓ Mais eficientes

# Definição

## Coleções (*collection*)

- ✓ Classe
- ✓ Tamanho dinâmico
- ✓ Mais recursos



Coleções possuem **recursos** adicionais proporcionando uma série de operações já prontas e disponíveis ao programador

## Arranjos (*arrays*)

- ✓ Estrutura
- ✓ Tamanho estático
- ✓ Mais eficientes



Arranjos são mais **eficientes** em termos de execução e economia de memória que coleções (considerando operações básicas)



# Arranjos

## Código (Main.java)

```
1 public class Main{  
2     public static void main(String[] args){  
3         int arranjo[] = new int[4];  
4     }  
5 }
```

**É OBRIGATÓRIA A DECLARAÇÃO DO TAMANHO**

(DEPOIS NÃO HÁ COMO ALTERAR, POIS É CONFIGURADO COMO FINAL)

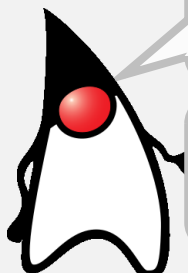
**TODOS OS ARRANJOS SÃO OBJETOS**

(OU SEJA, POSSUEM ATRIBUTOS E MÉTODOS)

Também  
consigo  
iniciar  
usando  
colchetes.

Eu prefiro declarar assim: `int[] arranjo = new int[4];`

Eu prefiro declarar: `int arranjo[4];` Ou: `int *arranjo;`



# Arranjos

## Código (Main.java)

```
1 public class Main{  
2     public static void main(String[] args){
```

DEITEL, Paul. Java: como programar.

## 7.3 Declarando e criando arrays

Os objetos array ocupam espaço na memória.

Como outros objetos, arrays são criados com a palavra-chave new:

```
int[] c = new int[12];
```

Criar o array também pode ser realizado em dois passos, como a seguir:

```
int[] c; // declara a variável de array  
c = new int[12]; // cria o array; atribui à variável de array
```

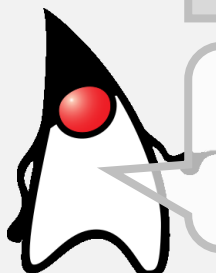
# Arranjos

## Código (Main.java)

```
1 public class Main{  
2     public static void main(String[] args){  
3         int arranjo[] = new int[4];  
4     }  
5 }
```

Arranjos são inicializados na linguagem JAVA com o valor **padrão do tipo** no momento da declaração

<b>Lógico:</b> false	<b>Numérico:</b> zero	<b>Objetos:</b> null
----------------------	-----------------------	----------------------



Diferente da **linguagem C** os meus arranjos **não** são alocados com **lixo** na memória.

**Lixo** é você!



# Arranjos

## Código (Main.java)

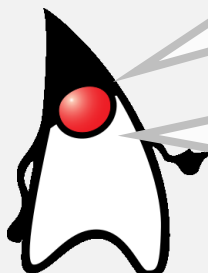
```
1 public class Main{  
2     public static void main(String[] args){  
3         int arranjo[] = {10, 20, 30, 40};  
4     }  
5 }
```

Arranjos pode ser inicializados com um conjunto de valores  
separados por vírgula e entre colchetes

O new está “nos bastidores”.  
Agora **PARA** de incomodar.

Para de me copiar! Cadê seu new?

Mas se você quer tanto o new, dá para fazer assim:  
`int[] arranjo = new int[]{10, 20, 30, 40};`



# Arranjos

Código (Main.java)

DEITEL, Paul. Java: como programar.

Quando o compilador encontrar uma declaração de array que inclua uma **lista de inicializador**, ele *conta* o número de inicializadores na lista para determinar o tamanho do array, depois configura a operação new apropriada **“nos bastidores”**.

O new está “nos bastidores”.  
Agora **PARA** de incomodar.

Para de me copiar! Cadê seu new?

Mas se você quer tanto o new, dá para fazer assim:

```
int[] arranjo = new int[]{10, 20, 30, 40};
```



# Arranjos

## Código (Main.java)

```
1 public class Main{  
2     public static void main(String[] args){  
3         int arranjo[] = new int[4];  
4         arranjo[0] = 10;  
5         arranjo[1] = 20;  
6         arranjo[2] = 30;  
7         arranjo[3] = 40;  
8     }  
9 }
```

### ÍNDICE OU SUBSCRITO:

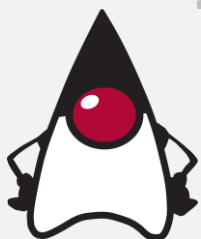
Corresponde a posição do literal  
(ou objeto) dentro do arranjo

Índices no JAVA devem ser (ou resultar em)  
números inteiros não negativos

Também é possível inicializar posição por posição...

Eu aceito índices negativos, lero lero.

Ninguém te perguntou **Python**, cadê a **linguagem C**?



# Arranjos

## Código (Main.java)

```
1 public class Main{
2     public static void main(String[] args){
3         int arranjo[] = {10, 20, 30, 40};
4         System.out.println(arranjo[1]);
5         System.out.println(arranjo[2]);
6         System.out.println(arranjo[0]);
7     }
8 }
9
10
11
12
```

## Saída

# Arranjos

## Código (Main.java)

```
1 public class Main{
2     public static void main(String[] args){
3         int arranjo[] = {10, 20, 30, 40};
4         for(int indice=0; indice<4; indice++){
5             System.out.println(indice);
6         }
7     }
8 }
9
10
11
12
```

## Saída



# Arranjos

## Código (Main.java)

```
1 public class Main{
2     public static void main(String[] args){
3         int arranjo[] = {10, 20, 30, 40};
4         for(int indice=0; indice<4; indice++){
5             System.out.println(arranjo[indice]);
6         }
7     }
8 }
```

## Saída

# Arranjos

## Código (Main.java)

```
1 public class Main{
2     public static void main(String[] args){
3         int arranjo[] = {10, 20, 30, 40};
4         for(int i=0; i<4; i++){
5             System.out.println(arranjo[i]);
6         }
7     }
8 }
```

Muito bom saber que ainda continuam usando em estruturas de repetição as minhas variáveis padrão: i, j e k.

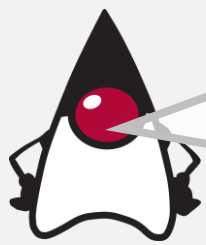
Fortran

## Saída

# Arranjos

DEITEL, Paul. Java: como programar.

Cada objeto array conhece seu comprimento e armazena-o em uma `variável de instância length`. Embora a variável de instância `length` de um array seja `public`, ela não pode ser alterada porque é uma `variável final`.



Os arranjos são criados em **tempo de execução** e não em **tempo de compilação**, ou seja, você pode declarar o tamanho depois.

# Arranjos

## Código (Main.java)

```
1 public class Main{  
2     public static void main(String[] args){  
3         int arranjo[] = {10, 20, 30, 40};  
4         System.out.println(arranjo.length);  
5     }  
6 }
```

Atributo público e estático da classe de arranjos da linguagem JAVA responsável por retornar o comprimento de um determinado arranjo (ou seja, sua quantidade de elementos)

## Saída

# Arranjos

## Código (Main.java)

```
1 public class Main{
2     public static void main(String[] args){
3         int arranjo[] = {10, 20, 30, 40};
4         for(int i=0; i<arranjo.length; i++){
5             System.out.println(arranjo[i]);
6         }
7     }
8 }
9
10
11
12
```

## Saída

REW ◀◀

00:03:00

# Estruturas de Repetição

APRIMORADA

Aulas atrás

A instrução **for** aprimorada itera sobre cada um dos item de um arranjo sem extrapolar seu limite

DEITEL, Paul. Java: como programar.

## 7.7 A instrução **for** aprimorada

A instrução **for** aprimorada itera pelos elementos de um array *sem* usar um contador, evitando assim a possibilidade de “pisar fora” do array.

A sintaxe de uma instrução **for** aprimorada é:

```
for (parâmetro : nomeDoArray)  
    instrução
```

# Arranjos

## for

Estrutura de repetição que executa um determinado bloco de código com base nos **limites** e nas **iterações** estabelecidas

```
for (i=0; i<4; i++)
```

Vai ver se eu estou na esquina. Estou muito bem só com meu for tradicional.

## forEach

Estrutura de repetição que percorre todos os **elementos** de um determinado **arranjo** ou **coleção**

NOTA: Não confundir essa estrutura de repetição com método `forEach()` da classe `Stream`.

```
for (int itens: arranjo)
```

Isso você não tem né? Fala que eu estou te copiando agora!





# Arranjos

## for

Caso haja a necessidade de modificar elementos do arranjo, então a instrução for tradicional controlada por contador deve ser utilizada

Hmmm, parece que o meu for ainda é melhor que o seu!

## forEach

A instrução for aprimorada só pode ser utilizada para obter elementos de um arranjo, ela não pode ser usada para modificar elementos

Fica quieto, o for nem é seu, foi o **FORTRAN** quem inventou!

Alguém me chamou?

Fortran

# Arranjos

## Código (Main.java)

```
1 public class Main{
2     public static void main(String[] args){
3         int arranjo[] = {10, 20, 30, 40};
4         for(int i : arranjo){
5             System.out.println(i);
6         }
7     }
8 }
```

Devem ser do mesmo tipo

DEITEL, Paul. Java: como programar.



### Dica de prevenção de erro 7.2

*A instrução for aprimorada simplifica o código iterando por um conjunto tornando o código mais legível e eliminando várias possibilidades de erro, como especificação inadequada do valor inicial da variável de controle, o teste de continuação de loop e a expressão de incremento.*

## Saída

# Arranjos

A classe `Arrays` não é considerada um recurso básico, por isso não está no pacote `java.lang`, para ter acesso aos seus recursos é preciso importar o pacote `java.util.Arrays`

DEITEL, Paul. Java: como programar.

## 7.15 Classe `Arrays`

A classe `Arrays` ajuda a evitar reinventar a roda fornecendo métodos `static` para manipulações de array comuns. Esses métodos incluem `sort` para *classificar* um array (isto é, organizar os elementos em ordem), `binarySearch` para *procurar* um array *classificado* `equals` para *comparar* arrays e `fill` para *inserir valores em um array*. Esses métodos são sobrecarregados para arrays de tipo primitivo e arrays de objetos.

# Arranjos

`Arrays.sort(Object[] a)`

## Propósito:

Ordena um arranjo de objetos em **ordem crescente**, de acordo com a **ordem natural** de seus elementos

### Código (Main.java)

```
1 import java.util.Arrays;
2
3 public class Main{
4     public static void main(String[] args){
5         int arranjo[] = {40, 30, 20, 10};
6         Arrays.sort(arranjo);
7         for(int elemento: arranjo)
8             System.out.println(elemento);
9     }
```

### Saída

# Arranjos

`Arrays.sort(Object[] a)`

## Propósito:

Ordena um arranjo de objetos em **ordem crescente**, de acordo com a **ordem natural** de seus elementos

### Código (Main.java)

```
1 import java.util.Arrays;
2
3 public class Main{
4     public static void main(String[] args){
5         char arranjo[] = {'D', 'C', 'B', 'A'};
6         Arrays.sort(arranjo);
7         for(char elemento: arranjo)
8             System.out.println(elemento);
9     }
```

### Saída

# Arranjos

`Arrays.sort(Object[] a)`

## Propósito:

Ordena um arranjo de objetos em **ordem crescente**, de acordo com a **ordem natural** de seus elementos

### Código (Main.java)

```
1 import java.util.Arrays;
2
3 public class Main{
4     public static void main(String[] args){
5         String arranjo[] = {"2", "1", "10"};
6         Arrays.sort(arranjo);
7         for(String elemento: arranjo)
8             System.out.println(elemento);
9     }
```

### Saída

# Arranjos

`Arrays.fill(int[] a, int val)`

## Propósito:

Preenche o arranjo do tipo `int` (inteiro) com o valor do tipo `int` (inteiro)

### Código (Main.java)

```
1 import java.util.Arrays;
2
3 public class Main{
4     public static void main(String[] args){
5         int arranjo[] = {10, 20, 30, 40};
6         Arrays.fill(arranjo, 10);
7         for(String elemento: arranjo)
8             System.out.println(elemento);
9     }
```

### Saída

# Arranjos

`Arrays.binarySearch(int[] a, int key)`

## Propósito:

Pesquisa no arranjo do tipo `int` (inteiro) o valor (chave) do tipo `int` (inteiro)

### Código (Main.java)

```
1 import java.util.Arrays;
2
3 public class Main{
4     public static void main(String[] args){
5         int arranjo[] = {10, 20, 30, 40};
6         int local;
7         local = Arrays.binarySearch(arranjo, 10);
8         System.out.println(local);
9     }
```

### Saída



# Arranjos

`Arrays.binarySearch(int[] a, int key)`

## Propósito:

Pesquisa no arranjo do tipo `int` (inteiro) o valor (chave) do tipo `int` (inteiro)

### Código (Main.java)

```
1 import java.util.Arrays;
2
3 public class Main{
4     public static void main(String[] args){
5         int arranjo[] = {10, 20, 30, 40};
6         int local;
7         local = Arrays.binarySearch(arranjo, 20);
8         System.out.println(local);
9     }
```

### Saída

# Arranjos

```
Arrays.binarySearch(int[] a, int key)
```

## Propósito:

Pesquisa no arranjo do tipo `int` (inteiro) o valor (chave) do tipo `int` (inteiro)

## Código (Main.java)

```
1 import java.util.Arrays;
2
3 public class Main{
4     public static void main(String[] args){
5         int arranjo[] = {10, 20, 30, 40};
6         int local;
7         local = Arrays.binarySearch(arranjo, 25);
8         System.out.println(local);
9     }
```

Não é isso, mas pode pensar assim que funciona....

## Saída

Posição -1 que o número precisa ser inserido para manter o arranjo ordenado?

# Arranjos

`Arrays.equals(int[] a, int[] a2)`

## Propósito:

Retorna true se os dois arranjos do tipo int (inteiro) forem iguais entre si

### Código (Main.java)

```
1 import java.util.Arrays;
2
3 public class Main{
4     public static void main(String[] args){
5         int arranjo[] = {10, 20, 30, 40};
6         int desarranjo[] = {10, 20, 30, 40};
7         boolean valor;
8         valor=Arrays.equals(arranjo, desarranjo);
9         System.out.println(valor);
```

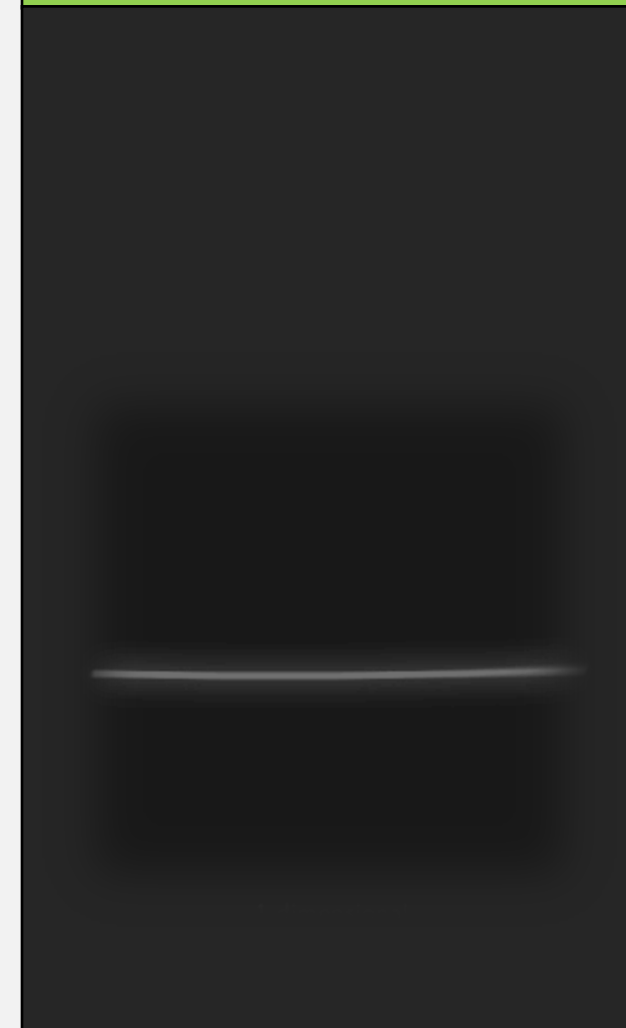
### Saída

# Arranjos de Arranjos

## Código (Main.java)

```
1 public class Main{
2     public static void main(String[] args){
3         int arranjo[][] = new int[2][3];
4         arranjo[0][0] = 10;
5         arranjo[0][1] = 20;
6         arranjo[0][2] = 30;
7         arranjo[1][0] = 40;
8         arranjo[1][1] = 50;
9         arranjo[1][2] = 60;
10        for(int linha=0; linha<2; linha++){
11            for(int coluna=0; coluna<3; coluna++){
12                System.out.printf(arranjo[linha][coluna] + " ");
13            }
14            System.out.println();
15        }
```

## Saída



# Arranjos de Arranjos

DEITEL, Paul. Java: como programar.

	Coluna 0	Coluna 1	Coluna 2	Coluna 3
Linha 0	a[ 0 ][ 0 ]	a[ 0 ][ 1 ]	a[ 0 ][ 2 ]	a[ 0 ][ 3 ]
Linha 1	a[ 1 ][ 0 ]	a[ 1 ][ 1 ]	a[ 1 ][ 2 ]	a[ 1 ][ 3 ]
Linha 2	a[ 2 ][ 0 ]	a[ 2 ][ 1 ]	a[ 2 ][ 2 ]	a[ 2 ][ 3 ]

Índice de coluna

Índice de linha

Nome do array

# Coleções

DEITEL, Paul. Java: como programar.

## 7.16 Introdução a coleções e classe ArrayList

A Java API fornece várias estruturas de dados predefinidas, chamadas *coleções*, usadas para armazenar grupos de objetos relacionados na memória. Essas classes fornecem métodos eficientes que organizam, armazenam e recuperam seus dados *sem a necessidade de conhecer como os dados são armazenados*.

Você já usou arrays para armazenar sequências de objetos. Arrays não mudam automaticamente o tamanho em tempo de execução para acomodar elementos adicionais. A *classe de coleção ArrayList<T>* (pacote `java.util`) fornece uma solução conveniente para esse problema — ela *pode alterar dinamicamente seu tamanho para acomodar mais elementos*.

# Coleções

adicionais. A **classe de coleção `ArrayList<T>`** (pacote `java.util`) fornece uma solução conveniente para esse problema — ela **pode alterar *dinamicamente* seu tamanho para acomodar mais elementos.**

Ah, essa é a **notação losango** (Diamond Operator).

Serve para criar uma **classe genérica**.

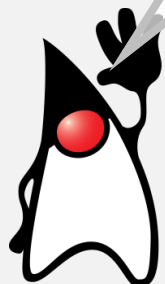
Dentro dos colchetes angulares basta inserir o tipo que deseja, por exemplo:

`ArrayList<Integer>`

O que `<T>` significa?

Sempre foi assim?

**Somente tipos não primitivos podem ser usados para declarar variáveis e criar coleções**



REW ◀◀

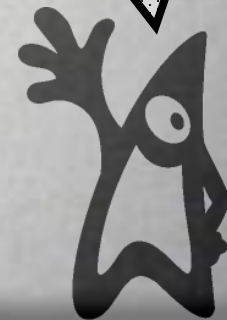
00:03:00



## Código (Main.java)

```
1 import java.util.ArrayList;
2
3 public class Main{
4     public static void main(String[] args){
5         int arranjo[] = new int[4];
6         ArrayList colecao = new ArrayList();
7     }
8 }
9
10 /* Antigamente não era possível informar o tipo de dado
11    * da coleção
12    *
13    * Tudo era um 'Object'
14    */
15
16
```

for(Object itens: colecao)

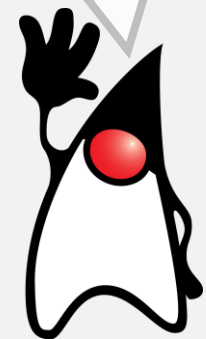


# Coleções

## Código (Main.java)

```
1 import java.util.ArrayList;
2
3 public class Main{
4     public static void main(String[] args){
5         int arranjo[] = new int[4];
6         ArrayList<Integer> arranjo = new ArrayList<Integer>();
7     }
8 }
9
10 /* Logo no início com a introdução das classes genéticas
11  * era necessário declarar o tipo dos dois lados
12  *
13  * Em tempo de compilação tudo funciona como antigamente
14  * Em tempo de execução agora o tipo é instanciado
15  * Nesse caso: 'Integer' (podendo usar seus métodos)
16  */
```

for(Integer itens: colecao)



# Coleções

## Código (Main.java)

```
1 import java.util.ArrayList;
2
3 public class Main{
4     public static void main(String[] args){
5         int arranjo[] = new int[4];
6         ArrayList<Integer> arranjo = new ArrayList<>(4);
7     }
8 }
```

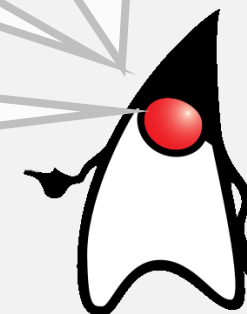
Caso precise aumentar eu faço isso adicionando um bit a mais de tamanho na fórmula:

$$\text{size} = \text{size} + (\text{size} \gg 1)$$

Assim o tamanho sempre dobra, evitando atualizar o tamanho de um em um...

Você pode inserir um tamanho inicial para a coleção

Se você **não** me informar eu vou colocar 10.



# Coleções

## DEITEL, Paul. Java: como programar.


Método	Descrição
add	Adiciona um elemento ao <i>final</i> do ArrayList.
clear	Remove todos os elementos do ArrayList.
contains	Retorna true se o ArrayList contém o elemento especificado; caso contrário, retorna false.
get	Retorna o elemento no índice especificado.
indexOf	Retorna o índice da primeira ocorrência do elemento especificado no ArrayList.
remove	Sobrecarregado. Remove a primeira ocorrência do valor especificado ou o elemento no índice especificado.
Size	Retorna o número de elementos armazenados em ArrayList.
trimToSize	Corta a capacidade do ArrayList para o número atual de elementos.

**Figura 7.23** | Alguns métodos e propriedades da classe ArrayList<T>.

# Coleções

## indexOf(Object o)

**Descrição:** Retorna o índice da primeira ocorrência do objeto especificado na coleção

**Parâmetros:**  – Objeto a ser procurado

**Retorno:** O índice na coleção do objeto procurado

**Exceção:** **NullPointerException**  
Caso a operação resulte em null

**Código:**

```
ArrayList<Integer> colecao = new ArrayList<>();  
colecao.indexOf(2);  
colecao.indexOf(1);  
colecao.indexOf(0);
```

# Coleções

`remove(int index)`

**Descrição:** Remove o objeto na posição especificada na coleção

**Parâmetros:** `index` – Posição do objeto a ser removido

**Retorno:** Objeto removido

**Exceção:** `IndexOutOfBoundsException`  
Caso o índice esteja fora do intervalo

**Código:**

```
ArrayList<Integer> colecao = new ArrayList<>();  
colecao.remove(2);  
colecao.remove(1);  
colecao.remove(0);
```

# Coleções

## clear()

**Descrição:** Remove todos os objetos da coleção

**Parâmetros:** –

**Retorno:** `void`

**Exceção:** `UnsupportedOperationException`  
Caso a operação não seja suportada pela coleção

**Código:**

```
ArrayList<Integer> colecao = new ArrayList<>();  
colecao.clear();
```

# Coleções

## size()

**Descrição:** Retorna a quantidade de objetos no conjunto

**Parâmetros:** –

**Retorno:** Quantidade de objetos

**Exceção:** **IndexOutOfBoundsException**  
Caso o índice esteja fora do intervalo

**Código:**

```
ArrayList<Integer> colecao = new ArrayList<>();  
colecao.size();
```



# Coleções

```
forEach(Consumer<? super T> action)
```

**Descrição:** Executa a operação especificada para cada um dos objetos da coleção

**Parâmetros:** **action** – Operação para ser realizada

**Retorno:** **void**

**Exceção:** **NullPointerException**  
Caso a operação resulte em null

**Código:**

```
ArrayList<Integer> colecao = new ArrayList<>();  
colecao.forEach(variavel -> System.out.println(variavel);  
colecao.forEach(numero -> soma += numero);  
colecao.forEach(n -> {if(n%2==0){System.out.println(par);}});
```

# Coleções

`get(int index)`

**Descrição:** Retorna o objeto na posição especificada nesta coleção.

**Parâmetros:** `index` – Posição do objeto para retornar

**Retorno:** O objeto na posição especificada

**Exceção:** `IndexOutOfBoundsException`  
Caso o índice esteja fora do intervalo

**Código:**

```
ArrayList<Integer> colecao = new ArrayList<>();  
colecao.get(2);  
colecao.get(1);  
colecao.get(0);
```

# Coleções

`set(int index, E element)`

**Descrição:** Substitui o objeto na posição especificada nesta coleção pelo objeto especificado.

**Parâmetros:** `index` – Posição do objeto a ser removido  
`element` – Objeto a ser inserido

**Retorno:** O antigo objeto na posição especificada

**Exceção:** `IndexOutOfBoundsException`  
Caso o índice esteja fora do intervalo

**Código:**

```
ArrayList<Integer> colecao = new ArrayList<>();  
colecao.set(2, 10);  
colecao.set(1, 20);  
colecao.set(0, 30);
```

# Coleções

`equals(Object o)`

herdado

**Descrição:** Compara a coleção com o objeto especificado em termo de tipo, tamanho e disposição dos elementos

**Parâmetros:**

- o – Objeto a ser comparado

**Retorno:**

- `true` – caso a coleção seja idêntica ao objeto
- `false` – caso haja alguma diferença com o objeto

**Exceção:** –

**Código:**

```
ArrayList<Integer> colecao = new ArrayList<>();  
ArrayList<Integer> lista = new ArrayList<>();  
colecao.equals(lista);  
lista.equals(colecao);
```

# Coleções

## toString()

herdado

**Descrição:** Retorna a coleção formatada em texto entre colchetes e com seus elementos separados por vírgula

**Parâmetros:** –

**Retorno:** Cadeia de texto da coleção

**Exceção:** –

**Código:**

```
ArrayList<Integer> colecao = new ArrayList<>();  
colecao.toString();
```

# Coleções

`add(int index, E element)`

## Descrição:

Insere o objeto especificado na posição especificada nesta lista (demais objetos são deslocados à direita)

## Parâmetros:

**index** – Posição para inserir (opcional)  
**element** – Objeto a ser inserido

## Retorno:

**void**

## Exceção:

**IndexOutOfBoundsException**

Caso o índice esteja fora do intervalo

## Código:

```
ArrayList<Integer> colecao = new ArrayList<>();  
colecao.add(10);  
colecao.add(20);  
colecao.add(1, 15);
```

# Coleções

## DEITEL, Paul. Java: como programar.

Método	Descrição
sort	Classifica os elementos de uma List.
binarySearch	Localiza um objeto em uma List usando o algoritmo de pesquisa binária de alto desempenho introduzido na Seção 7.15 e discutido em detalhes na Seção 19.4.
reverse	Inverte os elementos de uma List.
shuffle	Ordena aleatoriamente os elementos de uma List.
fill	Configura todo elemento List para referir-se a um objeto especificado.
copy	Copia referências de uma List em outra.
min	Retorna o menor elemento em uma Collection.
max	Retorna o maior elemento em uma Collection.
addAll	Acrescenta todos os elementos em um array a uma Collection.
frequency	Calcula quantos elementos da coleção são iguais ao elemento especificado.
disjoint	Determina se duas coleções não têm nenhum elemento em comum.

**Figura 16.5** | Métodos Collections.

# Modificadores de Acesso

## **static:**

O modificador de acesso **final** em **objetos** ou **tipos primitivos** sinaliza que estes possuem apenas uma única referência na memória e quaisquer alterações vindas de qualquer classe, recaem sobre as demais

### Código (Main.java)

```
1 public class Main{
2     public static void main(String[] args){
3         Carro veiculo1      = new Carro();
4         Carro veiculo2      = new Carro();
5         veiculo1.tanque      = 50;
6         veiculo2.tanque      = 100;
7         System.out.println(veiculo1.tanque);
8         System.out.println(veiculo2.tanque);
```

### Código (Carro.java)

```
1 public class Carro{
2     static int tanque;
3     int velocidade;
4 }
5
6
7
8
```



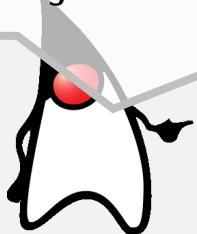
# Modificadores de Acesso

**final:**

O modificador de acesso `final` em **objetos** ou **tipos primitivos** sinaliza que estes não poderão ter seus valores reatribuídos após a inicialização, sendo instanciados como constantes\*

\*em métodos ele é responsável por mais algumas coisas

Resumindo... As declarações com a palavra-chave `final` implicam que objetos ou “variáveis” não poderão ser mais alterados em tempo de execução.



Hmm... Esse `final` está parecendo muito uma diretiva que eu tenho que me permite definir constantes sem consumir memória durante a execução, o nome dessa minha diretiva é: `#DEFINE`



# Resumo

- ❑ **Arranjo:** É um grupo valores todos do mesmo tipo. Arranjos são objetos no JAVA e portanto, são considerados **tipos por referência**.
- ❑ **Coleção:** É um objeto que armazena referências a outros objetos.
- ❑ **Inicialização:** Os valores são inicializados de forma padrão com base no tipo de dado.
- ❑ **Coleções Genéricas:** São coleções de um tipo **coringa**, codificadas de forma abrangente a se adaptar a qualquer outro tipo de dado.
- ❑ **Estrutura de Dados:** Corresponde as várias formas de organizar os dados computacionalmente. A linguagem de programação JAVA possui quatro estruturas fundamentais: **List**, **Queue**, **Set** e **Map** (**assunto para outra matéria**).

# Referências

DEITEL, Paul. **Java: como programar**. 4. ed. Porto Alegre: Bookman, 2003.

DEITEL, Paul. **Java: como programar**. 10. ed. São Paulo: Pearson Education do Brasil, 2017.

## COLEÇÕES

Alexandre Mendonça Fava  
[alexandre.fava@udesc.br](mailto:alexandre.fava@udesc.br)

---

Universidade do Estado de Santa Catarina – UDESC  
Programa de Pós-graduação em Computação Aplicada – PPGCA