

[< VOLTAR](#)

Arrays e Coleções de Dados

Como na maior parte das linguagens de programação, Java implementa coleções de dados, que podem ajudar muito na hora de programarmos alguma estrutura de dados, como vetores, tabelas, listas etc. Veremos aqui as principais formas de implementarmos essas coleções de dados em Java, com aplicação das boas práticas de programação.

NESTE TÓPICO

- > Tipos de coleções em Java
- > Arrays
- > Vetores multidimensionais
- > Array tipado com uma classe criada por você
- > Implementações prontas Java
- > ArrayList



Tipos de coleções em Java

Em Java podemos implementar manualmente coleções de dados usando arrays, com quantas dimensões forem necessárias, ou podemos usar coleções prontas, pré-implementadas pela linguagem e já otimizadas.

Imagine um jogo em que o personagem tem um posse um conjunto de armas diferentes, com poderes e características diferentes umas das outras. Cada arma ocupa uma posição em sua “mochila de armas”. Para implementarmos isso, podemos usar a coleção de dados, onde cada arma está armazenada dentro de uma mesma estrutura que armazena armas.

Arrays

Um array, em Java, é **estrutura de dados** que armazena uma série de objetos em sequência, **todos do mesmo tipo**, em posições distintas da memória. Array em Java é o mesmo que “vetor” em estruturas de dados. Um array é a implementação manual dessa estrutura de dados.

Em Java, para declarar um array utiliza-se dois colchetes: []. Veja um exemplo, abaixo de declaração de um array de inteiros com 10 posições, ou seja, uma estrutura de dados que armazena até 10 valores inteiros:

```
1. //...
2.     int x[] = new int[10];
3. //...
```

É muito importante ressaltar que em cada posição exemplo acima pode-se armazenar um número inteiro e a numeração das posições inicia-se em 0 (zero!).

Isso quer dizer que se um array possui tamanho 10, as posições de endereçamento possível são de 0 a 9.

DICA - COMO SABER AS POSIÇÕES ENDEREÇÁVEIS DE UM ARRAY?

Um array qualquer de tamanho N possui, sempre, posições endereçáveis de 0 até N - 1, pois o 0 conta. Isso é muito importante!

Veja abaixo, um exemplo de array de três posições do tipo Sting e a implementação do código que atribui valores a cada posição do array:

```
1. //...
2.     //Declaração do array de strings de tamanho 3
3.     String nomes[] = new String[3];
4.
5.     //seta, manualmente, cada posição do array, de 0 a 2 (três posições)
6.     nomes[0] = "Josefino";
7.     nomes[1] = "Josefina";
8.     nomes[2] = "Marcelinho";
9.
10.    //E para acessar cada posição, podemos usar um laço, de 0 a 2 (0, 1, 2 - três posições):
11.    for (int i = 0; i <= 2; i++) {
12.        System.out.print("Na posição " + i + " do array, temos: ");
13.        System.out.println(nomes[i]);
14.    }
15. //...
```



O resultado da execução deste código pode ser visto abaixo:

```
run:
Na posição 0 do array, temos: Josefino
Na posição 1 do array, temos: Josefina
Na posição 2 do array, temos: Marcelinho
CONSTRUÍDO COM SUCESSO (tempo total: 1 segundo)
```

Resultado da execução do código acima

Graficamente, podemos ainda dizer que o exemplo acima cria uma estrutura de dados como vista na tabela abaixo:

Posição #	0	1	2
Valor	Josefino	Josefina	Marcelinho

Demonstração gráfica do exemplo de array de Strings

É **muito** importante tomar muito cuidado na hora de percorrer ou acessar os valores de um array, para não acessar uma posição não existente, ou seja, se um array tem tamanho 6, não podemos acessar a posição 7, pois ela não existe. Isso daria um erro no programa que, se não tratado (veremos isso ainda - tratamento de erros) pode finalizar o programa inesperadamente e fazer com que o usuário perca todo seu trabalho.

Para praticar, implemente um programa que leia um array de 10 inteiros e imprima para o usuário o maior valor inserido dentro do array. Dica: Use uma variável auxiliar para armazenar o primeiro valor e, se encontrar algum valor maior enquanto percorre o array, o valor dessa variável é substituído pelo valor encontrado. A cada laço será preciso comparar o valor na posição em que está com o valor na variável auxiliar.

Vetores multidimensionais

Para auxiliar em alguns casos, podemos ter quantas dimensões forem necessárias em nosso array, ou seja, ter mais de uma possível posição de endereçamento.

Segundo Teruel, o mais comum é a implementação de arrays de duas posições (bidimensionais), resultando em algo parecido com uma tabela de duas colunas.

Para exemplificar, vamos implementar um array bidimensional que armazena na **primeira posição** um nome e na **segunda posição** o sobrenome. Neste caso, a implementação ficará assim:



```

1. //...
2.     String nomesCompleto[3][3] = new String[3][3];
3.
4.     //Pessoa 1:
5.     nomesCompleto[0][0] = "Josefino"; //Primeira linha, primeira coluna
6.     nomesCompleto[0][1] = " Da Silva Sauro"; //Primeira linha, segunda coluna
7.
8.     //Pessoa 2:
9.     nomesCompleto[1][0] = "Josefina"; //Segunda linha, primeira coluna
10.    nomesCompleto[1][1] = " Da Silvo Saura"; //Segunda linha, segunda coluna
11.
12.    //Pessoa 3:
13.    nomesCompleto[2][0] = "Marcelinho"; //Terceira linha, primeira coluna
14.    nomesCompleto[2][1] = " Supimpa Supimposo"; //Terceira linha, segunda colun
a
15.
16.    //Impressão dos nomes com os sobrenomes:
17.    for(int i = 0; i <= 2; i++){
18.        System.out.print("Nome armazenado na linha " + i + ": ");
19.        //Pega a posição i (linha), a coluna do nome (0) e concatena com a colun
a do sobrenome (1)
20.        System.out.println(nomesCompleto[i][0] + nomesCompleto[i][1]);
21.    }
22. //...

```

E o resultado dessa implementação, será:

```

run:
Nome armazenado na linha 0: Josefino Da Silva Sauro
Nome armazenado na linha 1: Josefina Da Silvo Saura
Nome armazenado na linha 2: Marcelinho Supimpa Supimposo
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)

```

Resultado da execução do exemplo de um array bidimensional



Note que, para este exemplo, utilizou-se um array bidimensional cujo número de linhas e colunas era o mesmo. Podemos, em Java, criar um array com quantas linhas e colunas forem necessárias, de tamanhos diferentes.

Array tipado com uma classe criada por você

Atém de criar arrays de tipos primitivos, como exemplificado até agora, podemos criar um array cujo tipo é implementado por você. Imagine, por exemplo ter uma lista de pessoas em uma só estrutura de dados, ou seja, uma lista do tipo “Pessoa” e cada pessoa dessa lista ter suas próprias características. Essa é uma das grandes vantagens do mundo orientada a objetos, pois isso é possível em Java.

Para este exemplo, precisamos antes, implementar a classe que usaremos para tipar nosso array, neste caso, a classe “Pessoa”. Nossa pessoa terá os seguintes atributos:

- Nome (String)
- Idade (int)
- E-mail (String)

A codificação dessa classe, poderá ser assim:

```
1. public class Pessoa {
2.     String nome;
3.     int idade;
4.     String email;
5. }
```

E agora vamos criar uma classe que possa conter até 4 pessoas com as características mostradas acima e ainda imprima na console as informações das pessoas:

```
1. public class Exemplo {
2.
3.     public static void main(String args[]) {
4.
5.         //Cria o array que pode armazenar 4 pessoas:
6.         Pessoa listaPessoas[] = new Pessoa[4];
7.
8.         //Pessoa 1
9.         listaPessoas[0] = new Pessoa(); //É preciso inicializar
10.        listaPessoas[0].nome = "Josefino Silvio Santos";
11.        listaPessoas[0].idade = 41;
12.        listaPessoas[0].email = "josefino@uninove.edu.br";
13.
14.        //Pessoa 2
15.        listaPessoas[1] = new Pessoa();
16.
17.        listaPessoas[1].nome = "Josefina Rodrigues";
18.        listaPessoas[1].idade = 20;
19.        listaPessoas[1].email = "josefina@uninove.edu.br";
20.
21.        //Pessoa 3
22.        listaPessoas[2] = new Pessoa();
23.
24.        listaPessoas[2].nome = "Jaspion";
25.        listaPessoas[2].idade = 90;
26.        listaPessoas[2].email = "jaspion909@uninove.edu.br";
27.
28.        //Pessoa 4
29.        listaPessoas[3] = new Pessoa();
30.        listaPessoas[3].nome = "Power ranger azul";
31.        listaPessoas[3].idade = 35;
32.        listaPessoas[3].email = "lospowerranges@uninove.edu.br";
33.
34.        //Vamos mostrar na tela os dados das pessoas ,usando o while:
35.        int i = 0;
36.        while (i <= 3) {
37.            System.out.println("Dados da pessoa " + (i + 1) + ":"); //i + 1 para dei
38.            xar mais amigável a exibição
39.            System.out.println("\t" + listaPessoas[i].nome);
40.            System.out.println("\t" + listaPessoas[i].idade);
41.            System.out.println("\t" + listaPessoas[i].email);
42.            i++; //Itera o i
43.        }
44.    }
```



O resultado da execução deste código, pode ser visto abaixo:

```
run:
Dados da pessoa 1:
    Josefino Silvio Santos
    41
    josefino@uninove.edu.br
Dados da pessoa 2:
    Josefina Rodrigues
    20
    josefina@uninove.edu.br
Dados da pessoa 3:
    Jaspion
    90
    jaspion909@uninove.edu.br
Dados da pessoa 4:
    Power ranger azul
    35
    lospowerranges@uninove.edu.br
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)
|
```

Resultado da execução do exemplo de um array tipado

DICA: SE VOCÊ NÃO SABE O TAMANHO DO ARRAY, PERGUNTE A ELE

Se você não sabe o tamanho do array que está utilizando, você pode utilizar uma função que lhe retorna este tamanho total. Assim: `vetor.length`.

Mas cuidado! O `length` retorna o tamanho total de posições endereçáveis, ou seja, se você estiver iterando este array, não esqueça de ir de 0 até `vetor.length - 1`.



Implementações prontas no Java

Como mencionado acima, um array é uma implementação manual da estrutura de vetores. Contudo, em Java, podemos utilizar implementações prontas de estruturas de dados de coleções. É o caso das listas.

Embora você gaste mais memória utilizando estruturas prontas e não tenha tanto controle das estruturas prontas quanto tem com aquelas implementadas manualmente, existem vantagens de se trabalhar com estruturas de dados prontas em Java, como:

- Capacidade de alocação dinâmica de dados (a lista, por exemplo, pode ter o tamanho que você quiser e este tamanho pode mudar em tempo de execução;
- Métodos prontos de ordenação, retirada e acréscimo de valores em posições da lista de forma rápida
- Métodos prontos de consulta às listas

Será apresentada aqui, a principal implementação de coleção em Java: A ArrayList.

ArrayList

O ArrayList é uma coleção do Java implementada pela classe `java.util.ArrayList`, ou seja, sempre que for utilizá-la, é preciso adicionar no início de seu código a instrução de importação dessa classe, assim:

```
1. import java.util.ArrayList;
```

Usar um ArrayList não é difícil. É muito importante saber que, para usá-lo, é preciso indicar o tipo de objeto que está sendo armazenado nessa estrutura de dados. A sintaxe de sua declaração é assim:

```
1. ArrayList<ObjetoUsado> nomeDaColecao= new ArrayList<ObjetoUsado>();
```

Por exemplo, se você está criando uma coleção de nomes de clientes, ou seja, de valores do tipo **String**, sua ArrayList ficará assim:

```
1. ArrayList<String> clientes = new ArrayList<String>();
```

Quando você cria um objeto que é do tipo ArrayList, você possui a disposição os métodos que a classe implementa. Os principais métodos são:

- **add(VALOR)**: Serve para acrescentar um item na estrutura. você deve passar como parâmetro o valor que está sendo inserido,, que deve ser do mesmo tipo da lista.
Exemplo: `clientes.add("Transformers");`
- **add (ÍNDICE, VALOR)**: Você também pode dizer em qual posição (índice) quer acrescentar o valor, numericamente. Lembre-se que, assim como os vetores, a contagem nos ArrayLists começa no índice 0.
Exemplo: `clientes.add(1, "Power Rangers");`
- **size()**: Retorna a você o tamanho da lista. Lembre-se que, como o primeiro índice é o 0, ele é considerado no tamanho.
Exemplo: `int x = clientes.size();`
- **get(POSIÇÃO)**: Retorna o objeto que está na posição (índice) informada.
Exemplo: `String cliente1 = clientes.get(1);`



- **remove(VALOR):** Remove o valor informado. O objeto passado deve ser do mesmo tipo da ArrayList.
Exemplo: `clientes.remove("Transformers");`
- **clear():** Limpa, completamente, a lista.
Exemplo: `clientes.clear();`

Para ficar um pouco mais claro, veja, abaixo, a implementação de uma ArrayList tipada com a classe "Pessoa" (a mesma demonstrada anteriormente, no exemplo dos vetores, ou seja, use a mesma classe "Pessoa" acima):

Este exemplo faz o seguinte: Ele cria uma lista contendo 4 pessoas. Imprime os dados de todas elas (uma a uma) usando um laço. Depois disso, procura uma pessoa chamada "Jaspion" para remoção da lista e imprime a lista de nomes que ficaram ainda na estrutura.





```

1. import java.util.ArrayList;
2.
3. public class ExemploArrayList {
4.
5.     public static void main(String args[]) {
6.
7.         //Criando a lista do tipo pessoa
8.         ArrayList<Pessoa> listaPessoas = new ArrayList<>();
9.
10.        //Cria um objeto de "Pessoa" para podemos adiciona-lo a lista
11.        Pessoa pessoaLocal = new Pessoa();
12.        pessoaLocal.nome = "Sonic";
13.        pessoaLocal.idade = 200;
14.        pessoaLocal.email = "sonic@uninove.edu.br";
15.
16.        //Adiciona a pessoa a lista:
17.        listaPessoas.add(pessoaLocal);
18.
19.        //Já que a pessoa criada já foi acrescentada à lista,
20.        //podemos reaproveitar a variável local para acrescentar outra pessoa:
21.        pessoaLocal = new Pessoa(); //"reseta" a variável para poder ser reutilizada
22.        pessoaLocal.nome = "Lara Croft";
23.        pessoaLocal.idade = 21;
24.        pessoaLocal.email = "laracroftJava@uninove.edu.br";
25.
26.        //Acrescenta:
27.        listaPessoas.add(pessoaLocal);
28.
29.        //Mais uma pessoa
30.        pessoaLocal = new Pessoa(); //"reseta" a variável para poder ser reutilizada
31.        pessoaLocal.nome = "Mario Bross";
32.        pessoaLocal.idade = 180;
33.        pessoaLocal.email = "donkeykongnaomepega@uninove.edu.br";
34.
35.        //acrescenta:
36.        listaPessoas.add(pessoaLocal);
37.
38.        //Só mais uma pessoa
39.        pessoaLocal = new Pessoa(); //"reseta" a variável para poder ser reutilizada
40.        pessoaLocal.nome = "Jaspion";
41.        pessoaLocal.idade = 89;
42.        pessoaLocal.email = "jaspionJava@uninove.edu.br";
43.        listaPessoas.add(pessoaLocal);
44.
45.        //Percorrendo a lista com um laço e acessando seus itens
46.        for (int i = 0; i <= listaPessoas.size() - 1; i++) {
47.            System.out.println("Imprimindo dados da posição: " + i);
48.            System.out.println("\t- Nome: " + listaPessoas.get(i).nome);
49.            System.out.println("\t- Idade: " + listaPessoas.get(i).idade);
50.            System.out.println("\t- E-Mail: " + listaPessoas.get(i).email);
51.            System.out.println("-----");
52.        }
53.
54.        //Vamos procurar a pessoa que chama-se "Jaspion" e remove-la da lista
55.        //Para isso, é preciso percorrer a lista, procurando o nome e,
56.        //quando for encontrado, guardar o índice que a pessoa está.
57.        int posicaoRemovida = -999; //Variável auxiliar para remoção; um número qual
quer
58.        for (int j = 0; j < listaPessoas.size(); j++) {
59.            if (listaPessoas.get(j).nome.equals("Jaspion")) { //Se encontrar...
60.                posicaoRemovida = j; // Atribui a variável o valor do ÍNDICE
61.                break; //se já encontrou, o loop pode parar
62.            }
63.        }
64.        //Verifica se a variável auxiliar foi alterada. Se sim, remove
65.        if (posicaoRemovida != -999) {
66.            listaPessoas.remove(posicaoRemovida); //Remove na posição encontrada
67.            System.out.println(" >> O Jaspion estava na posição " + posicaoRemovida);
68.        }
69.

```

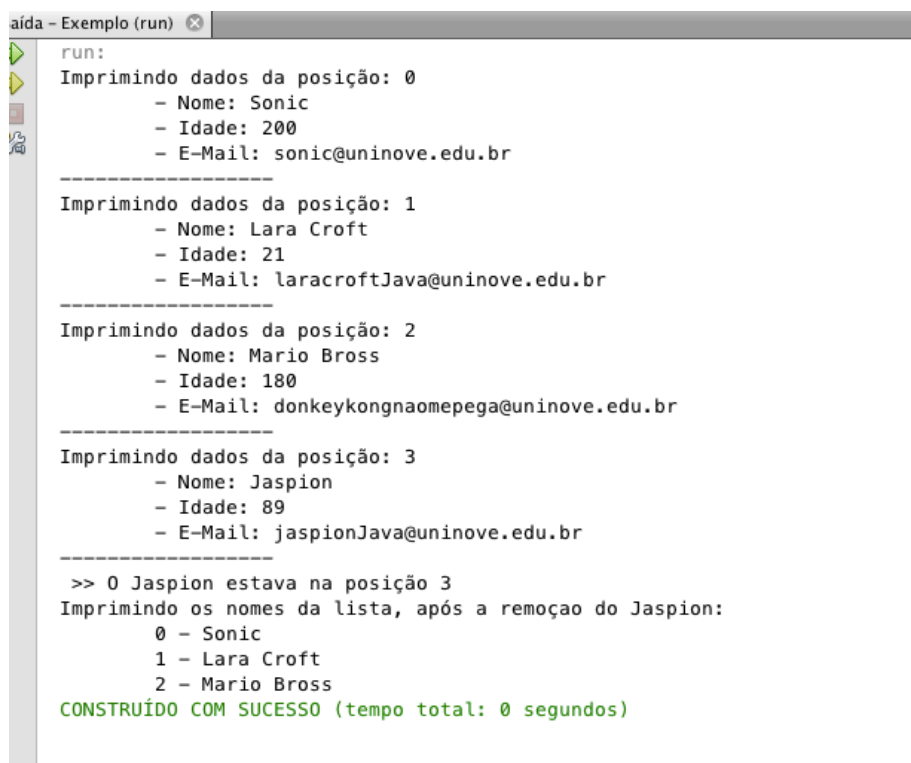


```

70.         //Para ter certeza da remoção, vamos percorrer e imprimir a lista novamente,
71.         //mas só os nomes agora:
72.         System.out.println("Imprimindo os nomes da lista, após a remoção do Jaspion:
");
73.         for (int k = 0; k < listaPessoas.size(); k++) {
74.             System.out.println("\t" + k + " - " + listaPessoas.get(k).nome);
75.         }
76.     }
77. }

```

Depois de implementar essa classe (e a classe Pessoa), o resultado da execução deste exemplo pode ser visto em:



```

aída - Exemplo (run) x
run:
Imprimindo dados da posição: 0
- Nome: Sonic
- Idade: 200
- E-Mail: sonic@uninove.edu.br
-----
Imprimindo dados da posição: 1
- Nome: Lara Croft
- Idade: 21
- E-Mail: laracroftJava@uninove.edu.br
-----
Imprimindo dados da posição: 2
- Nome: Mario Bross
- Idade: 180
- E-Mail: donkeykongnaomepega@uninove.edu.br
-----
Imprimindo dados da posição: 3
- Nome: Jaspion
- Idade: 89
- E-Mail: jaspionJava@uninove.edu.br
-----
>> 0 Jaspion estava na posição 3
Imprimindo os nomes da lista, após a remoção do Jaspion:
0 - Sonic
1 - Lara Croft
2 - Mario Bross
CONSTRUÍDO COM SUCESSO (tempo total: 0 segundos)

```

Execução do exemplo de ArrayList

Note que, neste exemplo, após a remoção do item, o índice que era ocupado pelo objeto que foi removido não existe mais, ou seja, a própria lista cuidou da remoção e realocação dos objetos que permaneceram na lista, automaticamente. Se estivéssemos trabalhando com vetores, por exemplo, essa remoção seria bem mais complicada, pois seria necessário mover todos os demais itens manualmente, após a remoção.

Resumo da aula

Nesta aula, você aprendeu a:

- Criar e percorrer arrays, em Java
- Criar e percorrer arrays multidimensionais, em Java
- Criar e percorrer arrays de classes que você cria (tipado)

- Descobrir o tamanho de um array
- Criar, percorrer e remover um objeto de um arraylist

Pronto, chegamos ao final de mais uma aula e isso quer dizer que você está pronto(a) para criar uma aplicação Java um pouco mais avançada. Para praticar, tente implementar métodos de ordenação de listas usando o bubble sort e quicksort, por exemplo. Bons estudos e boa programação.

Quiz

Exercício Final

Arrays e Coleções de Dados

INICIAR ➤



Referências

Deitei P. e Deitel H., 2010, Java : Como programar, 8ª Edição, Pearson Pretice Hall

Teruel, E. C., 2015, Programação Orientada a Objetos com Java - sem mistérios - 1ª Ed., Editora Uninove

Schildt, H., 2015, Schildt, Java para iniciantes : crie, compile e execute programas Java rapidamente, Bookman



Avalie este tópico



ANTERIOR



Índice

Ajuda?
PRÓXIMO
([https://ava.un](https://ava.uninove.br/seu/AVA/topico/topico.php))

Gerando e Interceptando Exceções

Estruturas de Controle de Fluxo e Laço em
Java

Biblioteca
(<https://www.uninove.br/conheca-a-uninove/biblioteca/sobre-a-biblioteca/apresentacao/>)
Portal Uninove
(<http://www.uninove.br>)
Mapa do Site

idCurso=)

