



Java Fundamentos

RECODE
pro



Módulo 03 – Java Fundamentos Aula 04



Array (Vetor)

Os Arrays (vetores) são caracterizados por se tratar de uma única variável de um determinado tamanho que armazena várias informações do mesmo tipo. Essas informações são gravadas na memória sequencialmente e são referenciadas através de índices. Chamamos de vetor as variáveis unidimensionais.

A declaração de vetores em Linguagem Java deve obedecer a seguinte sintaxe:

tipo_de_dado[] nome_vetor = new tipo_de_dado [tamanho];

`int[] vetor_exemplo = new int[10];`

`tipo[] nome_vetor = { lista_de_valores };`

`int[] vetor_exemplo = { 0,1,2,3,4,5,6,7,8,9 };`

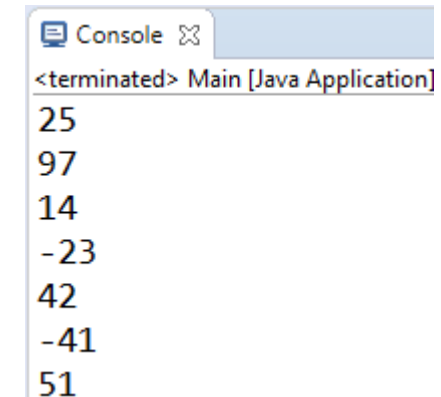
Módulo 03 – Java Fundamentos Aula 04



Este exemplo utiliza vetor com valores já atribuídos e lista os dados armazenados.

```
public class Main {  
  
    public static void main(String[] args) {  
  
        int[] vetor_exemplo = { 25, 97, 14, -23, 42, -41, 51 };  
        for (int x = 0; x < vetor_exemplo.length; x++) {  
            System.out.println(vetor_exemplo[x]);  
        }  
    }  
}
```

Saída no console

A screenshot of a Java IDE console window. The window has a title bar with a blue icon and the text "Console". Below the title bar, the text "<terminated> Main [Java Application]" is displayed. The console output shows the following values: 25, 97, 14, -23, 42, -41, and 51, each on a new line.

```
Console  
<terminated> Main [Java Application]  
25  
97  
14  
-23  
42  
-41  
51
```

Módulo 03 – Java Fundamentos Aula 04



Neste exemplo, serão armazenados 7 números reais em um vetor. O usuário deverá informar estes números e ao final todos os valores devem ser exibidos.

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner entrada = new Scanner(System.in);
        double[] valores = new double[7];
        for (int x = 0; x < 7; x++) {
            System.out.print("Informe " + (x + 1) + "º valor: ");
            valores[x] = entrada.nextDouble();
        }
        System.out.println("== Valores armazenados no vetor ==");
        for (int x = 0; x < 7; x++) {
            System.out.println("Valor na posição " + x + " = " + valores[x]);
        }
        entrada.close();
    }
}
```

Módulo 03 – Java Fundamentos Aula 04



EXERCÍCIO: Armazenar o nome e o salário de 5 funcionários. Após o cadastro destas informações, deverá ser digitado o índice de reajuste salarial para todos os funcionários. O programa deverá calcular o novo salário e exibir todos os dados na tela.

Módulo 03 – Java Fundamentos Aula 04



RESOLUÇÃO

```
import java.util.Scanner;

public class Main {

    public static void main(String[] args) {

        Scanner entrada = new Scanner(System.in);

        int TAM = 5;

        String[] nome = new String[TAM];

        double[] salario = new double[TAM];

        double reajuste;

        for (int x = 0; x < TAM; x++) {

            System.out.println("Nome do " + (x + 1) + "º Funcionário: ");

            nome[x] = entrada.next();

            System.out.println("Salário: R$");

            salario[x] = entrada.nextDouble();

        }

        System.out.println(" == Informe o Valor do Reajuste (%): == ");

        reajuste = entrada.nextDouble();

        System.out.println(" == Informações Atualizadas == ");

        for (int x = 0; x < TAM; x++)

            System.out.println("NOME: " + nome[x] + " - SALARIO: " +

                (salario[x] += (reajuste * salario[x] / 100)));

        entrada.close();

    }

}
```

Módulo 03 – Java Fundamentos Aula 04



Java Collections Listas

As listas são estruturas de dados de armazenamento sequencial assim como os arrays. Mas, diferentemente dos arrays, as listas não possuem capacidade fixa o que facilita bastante o trabalho. List é a interface Java que define os métodos que uma lista deve implementar. As principais implementações da interface List são: ArrayList, LinkedList e Vector. Cada implementação possui suas características sendo apropriadas para contextos diferentes

Módulo 03 – Java Fundamentos Aula 04



Vamos implementar o ArrayList

```
ArrayList<String> dados = new ArrayList<>();

dados.add("Flavio");
dados.add("Antonia");
dados.add("Joao");
dados.add("Raquel");
dados.add("Rochele");

System.out.println("qtd de itens na lista: " + dados.size());
```

Métodos ArrayList

Método: size()

O método size() informa a quantidade de elementos armazenado na lista.

Módulo 03 – Java Fundamentos Aula 04



Métodos ArrayList

Método: clear()

O método clear() remove todos os elementos da lista.

Método: contains(Object)

Para verificar se um elemento está contido em uma lista podemos utilizar o método contains(Object)

Método: remove(Object)

Podemos retirar elementos de uma lista através do método remove(Object). Este método remove a primeira ocorrência do elemento passado como parâmetro.

Método: remove(int)

Outra forma de retirar elementos de uma lista é através do método remove(int)

Método: indexOf(Object)

Para descobrir o índice da primeira ocorrência de um determinado elemento podemos utilizar o método indexOf(Object).

Módulo 03 – Java Fundamentos Aula 04



Exemplos Métodos ArrayList

```
ArrayList<String> dados = new ArrayList<>();
dados.add("Flavio");
dados.add("Raquel");
dados.add("Rochele");
dados.add("Antonia");
dados.add("Joao");
// 1º Forma de iterar os elementos
for (String dado : dados) {
    System.out.println("Nome: " + dado);
}
// 2º Forma de iterar os elementos
for (int i = 0; i < dados.size(); i++) {
    System.out.println("Nome : " + dados.get(i));
}
// Removendo um item do ArrayList pelo nome
dados.remove("Raquel");
// Removendo um item do ArrayList pelo indice
dados.remove(1);
// 3º Forma de iterar os elementos
dados.forEach(dado -> {
    System.out.println("Nome: " + dado);
});
System.out.println("qtd de itens na lista: " + dados.size());
System.out.println(dados.contains("Flavio"));
dados.clear();
```

Módulo 03 – Java Fundamentos Aula 04



Exceções

- Uma exceção é qualquer condição de erro ou comportamento inesperado encontrado por um programa em execução
- Em Java, uma exceção é um objeto herdado da classe:
 - `java.lang.Exception` - o compilador obriga a tratar ou propagar
 - `java.lang.RuntimeException` - o compilador não obriga a tratar ou propagar
- Quando lançada, uma exceção é propagada na pilha de chamadas de métodos em execução, até que seja capturada (tratada) ou o programa seja encerrado

Módulo 03 – Java Fundamentos Aula 04



Estrutura try-catch

- **Bloco try**

Contém o código que representa a execução normal do trecho de código que pode acarretar em uma exceção.

- **Bloco catch**

Contém o código a ser executado caso uma exceção ocorra Deve ser especificado o tipo da exceção a ser tratada.

- **Bloco finally**

É um bloco que contém código a ser executado independentemente de ter ocorrido ou não uma exceção.



Módulo 03 – Java Fundamentos Aula 04



Sintaxe

```
try {  
    // código que inclui comandos/invocações de métodos  
    // que podem gerar uma situação de exceção.  
} catch (XException ex) {  
    // bloco de tratamento associado à condição de  
    // exceção XException ou a qualquer uma de suas subclasses  
} catch (YException ey) {  
    // bloco de tratamento para a situação de exceção  
    // YException ou a qualquer uma de suas subclasses  
} finally {  
    // bloco de código que sempre será executado após  
    // o bloco try, independentemente de sua conclusão  
    // ter ocorrido normalmente ou ter sido interrompida
```

Módulo 03 – Java Fundamentos Aula 04



Considere o seguinte exemplo:
Isso vai gerar um erro, porque **myNumbers [10]** não existe.

```
public class Main {  
  
    public static void main(String[] args) {  
  
        int[] myNumbers = { 1, 2, 3 };  
        System.out.println(myNumbers[10]); // error!  
  
    }  
}
```

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 10 out  
of bounds for length 3  
    at Main.main(Main.java:8)
```

Módulo 03 – Java Fundamentos Aula 04



Se ocorrer um erro, podemos usar `try...catch` para capturar o erro e executar algum código para tratá-lo:

```
public class Main {  
    public static void main(String[] args) {  
        try {  
            int[] myNumbers = { 1, 2, 3 };  
            System.out.println(myNumbers[10]);  
        } catch (Exception e) {  
            System.out.println("Algo deu errado.");  
        }  
    }  
}
```

Saída no console após o catch ter capturado o erro.
"Algo deu errado."

Módulo 03 – Java Fundamentos Aula 04



A instrução **finally** permite que você execute o código, depois **try...catch**, independentemente do resultado:

```
public class Main {  
    public static void main(String[] args) {  
        try {  
            int[] myNumbers = { 1, 2, 3 };  
            System.out.println(myNumbers[10]);  
        } catch (Exception e) {  
            System.out.println("Algo deu errado.");  
        } finally {  
            System.out.println("O 'try catch' terminou.");  
        }  
    }  
}
```


Módulo 03 – Java Fundamentos Aula 04



Podemos encadear vários blocos catch para capturar exceptions de classes diferentes.

```
public class Main {  
  
    public static void main(String[] args) {  
        Conta c = new Conta ();  
        try {  
            c.deposita (100) ;  
        } catch (IllegalArgumentException e) {  
            System.out.println(" Houve uma IllegalArgumentException ao depositar ");  
        } catch (SQLException e) {  
            System.out.println(" Houve uma SQLException ao depositar ");  
        }  
    }  
}
```

Módulo 03 – Java Fundamentos Aula 04



```
import java.io.File;
import java.io.IOException;
import java.util.Scanner;
public class Exe01 {
    public static void main(String[] args) {
        File file = new File("C:\\testearquivo\\saida.txt");
        Scanner sc = null;
        try {
            sc = new Scanner(file);
            while (sc.hasNextLine()) {
                System.out.println(sc.nextLine());
            }
        } catch (IOException e) {
            System.out.println("Erro ao abrir o arquivo: " + e.getMessage());
        } finally {
            if (sc != null) {
                sc.close();
            }
        }
    }
}
```

Lendo um arquivo texto e
tratando possíveis erros

Módulo 03 – Java Fundamentos Aula 04



PARA MAIS INFORMAÇÕES SOBRE EXCEÇÕES LEIA O ARTIGO DO LINK ABAIXO

<https://www.devmedia.com.br/tratando-excecoes-em-java/25514>

Módulo 03 – Java Fundamentos Aula 04



Criar uma agenda usando ArrayList com as operações de inserir, excluir, mostrar todos os contatos e pesquisar o contato por nome.

RECODE



www.recode.org.br

RECODE
pro

recodepro.org.br

Institucional



[/rederecode](#)



[/recoderede](#)