

Curso

Algoritmos & Programação em JAVA

Atualizado até o Java 21 & Eclipse 2023-09



Prof. Msc. Antonio B. C. Sampaio Jr
ENGENHEIRO DE SOFTWARE & PROFESSOR

@abctreinamentos
@amazoncodebr

www.abctreinamentos.com.br
www.amazoncode.com.br



CONTEÚDO PROGRAMÁTICO



- UNIDADE 1 – INTRODUÇÃO
- UNIDADE 2 – CONTRUÇÃO DE ALGORITMOS
- UNIDADE 3 – ESTRUTURAS DE SELEÇÃO
- UNIDADE 4 – ESTRUTURAS DE REPETIÇÃO
- UNIDADE 5 – TÓPICOS FINAIS
 - Depuração dos Programas no Eclipse
 - Estruturas de Dados Homogêneas – Vetores e Matrizes

CONTEÚDO PROGRAMÁTICO



- UNIDADE 5 – TÓPICOS FINAIS (Continuação)
 - Depuração dos Programas no Eclipse
 - Estruturas de Dados Heterogêneas – Registros e Arquivos
 - Records [NOVO]
 - Novos Métodos para Leitura e Escrita em Arquivos [NOVO]
 - Modularização

CONTEÚDO PROGRAMÁTICO



- **UNIDADE 5 – TÓPICOS FINAIS (Continuação)**
 - Orientação a Objetos
 - Algoritmos com Qualidade
 - Conclusão

UNIDADE 5

TÓPICOS FINAIS

Depuração dos Programas no Eclipse

Detecção de Erros

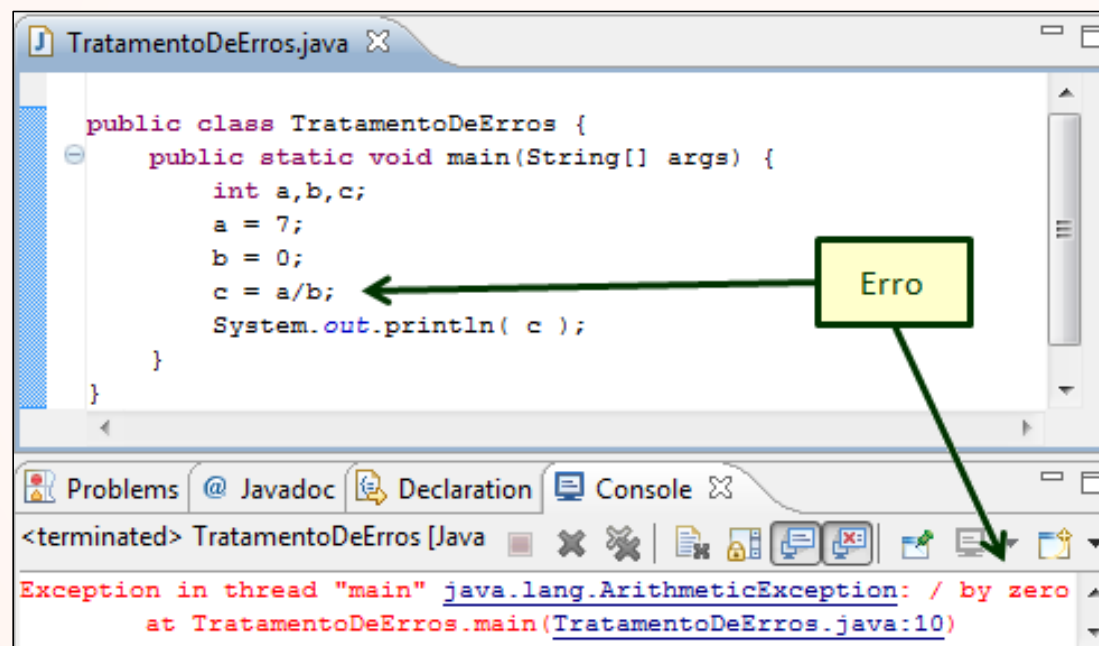
- Em programação, depuração de programas é o processo de detectar e remover erros no código. Existem dois tipos de erros em um programa: **ERROS DE SINTAXE e ERROS DE LÓGICA.**
- Os **ERROS DE SINTAXE** são aqueles gerados pela digitação incorreta de um comando, nome de variável, a não digitação de uma chave de abertura { ou fechamento de laço }.
- O próprio editor de código (Eclipse) destaca em vermelho os **ERROS DE SINTAXE.**

Detecção de Erros

```
15     if (sexo.equalsIgnoreCase("M"))
16     {
17         peso_ideal_homem = (altura*72.7) - 58;
18         System.out.println("O peso ideal do homem é de "+peso_ideal_homem);
19
20     else
21     {
22         peso_ideal_mulher = (altura*62.1) - 44.7;
23         System.out.println("O peso ideal da mulher é de "+peso_ideal_mulher);
24     }
```


Detecção de Erros

- Os **ERROS DE LÓGICA** são aqueles gerados quando o programa é executado e o resultado gerado é bem diferente do esperado.

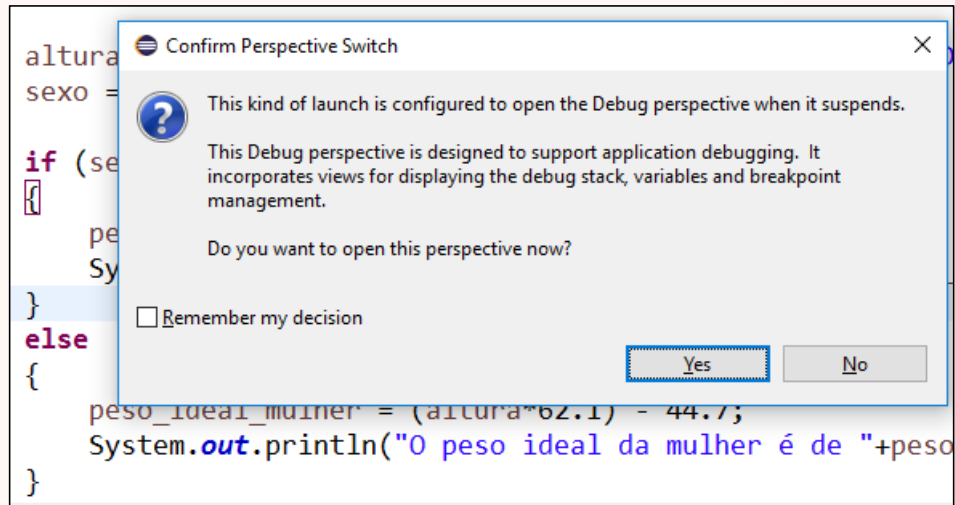
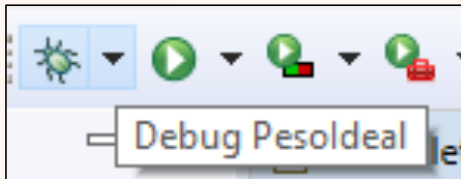


Detecção de Erros

- O **ECLIPSE** oferece dois importantes recursos de depuração: os **breakpoints** (pontos de parada) e a janela **watch** (observador).



```
11  
12     altura = Double.parseDouble(JOptionPane.showInputDialog("Digite a altura"));  
13     sexo = JOptionPane.showInputDialog("Digite o sexo");
```



Detecção de Erros

11



Debug console showing the execution of the application. The application is suspended at a breakpoint at line 13 of Pesoldeal.java. The console shows the following information:

- Debug console: Step Into (F5)
- Thread [main] (Suspended (breakpoint at line 13 in Pesoldeal))
- Pesoldeal.main(String[]) line: 13
- C:\Program Files\Java\jre-10.0.1\bin\javaw.exe (11 de jul de 2018 16:09:20)

Variables window showing the current state of the program:

Name	Value
no method return value	
args	String[0] (id=58)
altura	1.0

Exercícios

- 1) Analisar o programa **SequenciaS.java**.
- 2) Analisar o programa **Fibonacci.java**.

Estruturas de Dados Homogêneas

Vetores e Matrizes

Vetores e Matrizes

- Vetores e Matrizes são estruturas de dados muito simples que podem ajudar muito quando se têm muitas variáveis do mesmo tipo em um algoritmo. Imagine a seguinte situação: é necessário criar um algoritmo que lê o nome e as 4 notas de 50 alunos, calcular a média de cada aluno e informar quais foram aprovados e quais foram reprovados.

Vetores e Matrizes

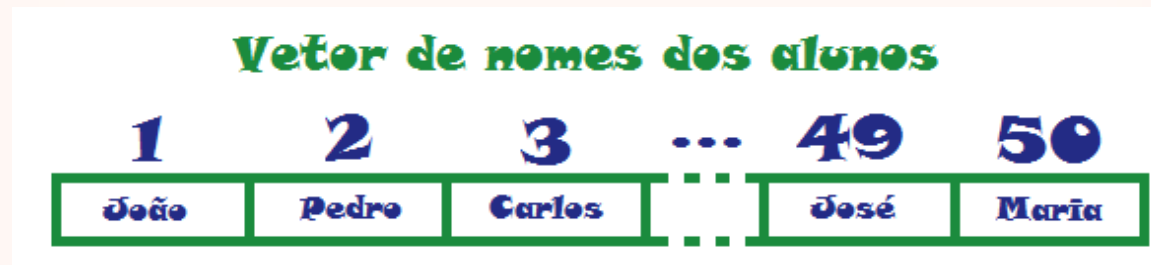
- Sem o uso Vetores e Matrizes, serão necessários criar:
 - 50 variáveis para armazenar os nomes dos alunos;
 - $(4 * 50)$ 200 variáveis para armazenar as 4 notas de cada aluno;
 - 50 variáveis para armazenar as médias de cada aluno;
 - **300 variáveis** no total serão criadas, sem contar a quantidade de linhas de código necessárias para ler todos os dados do usuário, calcular as médias e apresentar os seus resultados.
- A **BOA NOTÍCIA** é que o uso de Vetores e Matrizes (também conhecidos como ARRAYS) simplifica em muito esse trabalho!

Vetores

- Os vetores (ou arrays unidimensionais) são estruturas que permitem armazenar uma lista de dados (do mesmo tipo) na memória principal do computador.
- São muito úteis para inserir, consultar ou remover itens de uma lista.
- Um índice numérico identifica cada elementos da lista.

Vetores

- A estrutura vetor tem uma alocação contígua de memória indexada a partir de 1. A sua capacidade é fixa e deve ser informada no momento de sua declaração.



<https://dicasdeprogramacao.com.br/o-que-sao-vetores-e-matrizes-arrays/>

Declaração de Vetores

Declaração de Tipos

```
vet = VETOR[1..50] de character;
```

Declaração de Variáveis

```
vet: lista_alunos;
```

- Para referenciarmos um item do vetor, devemos indicar o seu nome, seguido por um número entre colchetes (índice) e o seu conteúdo.

```
lista_alunos[1] ← "João";  
lista_alunos[2] ← "Pedro";  
lista_alunos[3] ← "Carlos";
```

Vetores em JAVA

- No Java os vetores são tratados como uma área de memória fixa e sequencial dividida em partes iguais que são indexadas a partir de 0.
- ```
String lista_alunos[]; // Declaração
lista_alunos = new String[50]; // Criação
lista_alunos[0] = "João"; // Atribuição
lista_alunos[1] = "Pedro"; // Atribuição
lista_alunos[2] = "Carlos"; // Atribuição
```
- A capacidade de um vetor em Java é fixa (neste exemplo, 10) e deve ser informada no momento de criação do vetor.
  - É importante ressaltar que não é possível redimensionar um vetor em Java.

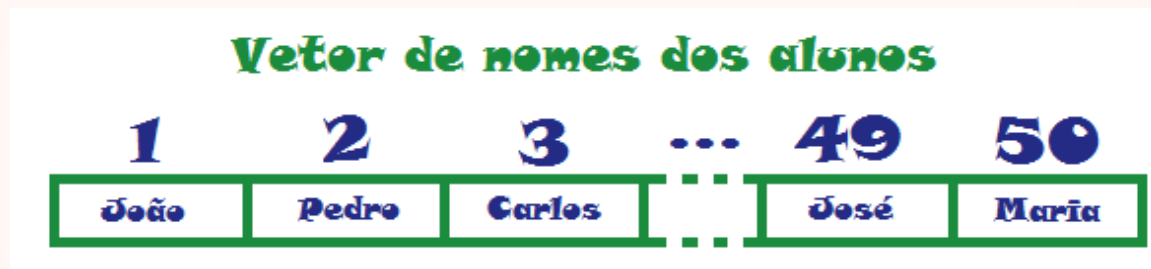
# Vetores em JAVA

- O tamanho de um vetor pode ser obtido pela propriedade **length**.
- Os elementos de um vetor são obtidos pelo código abaixo:

```
for(int i=0; i<lista_alunos.length; i++)
 System.out.println(lista_alunos[i]);
```

## Exercícios

- 1) Elaborar um programa que faça o cadastro do nome de 05 alunos e, ao final desta etapa, gere uma listagem desses nomes.



- 2) Escreva um programa que leia um vetor A de 10 elementos, construa e imprima outro vetor B da seguinte forma:

- Os elementos de ordem par são correspondentes a  $(2 \cdot A)$ ;
- Os elementos de ordem ímpar são correspondentes a  $(A/2)$ .

# Matrizes

- As matrizes (ou arrays multidimensionais) são estruturas que permitem armazenar dados em uma tabela na memória principal do computador. São muito úteis para inserir, consultar ou remover itens de uma tabela.

| 1    | 2     |
|------|-------|
| João | Pedro |

nomes dos alunos

- Dois índices numéricos (linha e coluna) identificam cada um dos elementos da tabela.

**Matriz das notas dos alunos**

|    | 1   | 2   | 3   | 4   |
|----|-----|-----|-----|-----|
| 1  | 9,5 | 10  | 8   | 7,5 |
| 2  | 10  | 9   | 9   | 5,5 |
| 3  | 9   | 8,5 | 9,5 | 7   |
| ⋮  |     |     |     |     |
| 49 | 7   | 10  | 10  | 9   |
| 50 | 7   | 8,5 | 5,5 | 4   |

# Declaração de Matrizes

## Declaração de Tipos

```
mat = VETOR[1..50,1..4] de real;
```

## Declaração de Variáveis

```
mat: notas_dos_alunos;
```

- Para referenciar um item da matriz, devemos indicar o seu nome, seguido por dois índices (linha e coluna) entre colchetes e o seu conteúdo.

```
notas_dos_alunos[1][1] ← 9,5;
```

```
notas_dos_alunos[1][2] ← 10;
```

```
notas_dos_alunos[1][3] ← 8;
```

```
notas_dos_alunos[1][4] ← 7,5;
```

# Matrizes em JAVA

- No Java as matrizes são tratadas como uma área de memória fixa e sequencial dividida em partes iguais que são indexadas a partir de 0.

```
double notas_dos_alunos[][]; // Declaração
notas_dos_alunos[][] = new double[50][4]; // Criação
notas_dos_alunos[0][0] = 9.5; // Atribuição
notas_dos_alunos[0][1] = 10; // Atribuição
notas_dos_alunos[0][2] = 8; // Atribuição
notas_dos_alunos[0][3] = 7.5; // Atribuição
```

- A capacidade de uma matriz em Java é fixa (neste exemplo, 50x4) e deve ser informada no momento de criação do vetor.
- É importante ressaltar que não é possível redimensionar uma matriz em Java.



# Matrizes em JAVA

- Os elementos de uma matriz são obtidos pelo código abaixo:

```
for(int i=0; i<50; i++) //Linhas
{
 for(int j=0; j<4; j++) //Colunas
 {
 System.out.println(notas_dos_alunos[i][j]);
 }
}
```

## Exercícios

- 1) Refatorar o programa que faz o cadastro de 05 alunos para também ler as suas 04 notas, calcular a sua média e informar, em outro vetor lógico, se o aluno foi aprovado ou reprovado. O número de alunos aprovados e o número de alunos reprovados deverão ser informados.
- 2) Construa uma matriz Identidade  $I=3$  (apenas os valores diagonais são = 1). Todos os outros são 0.

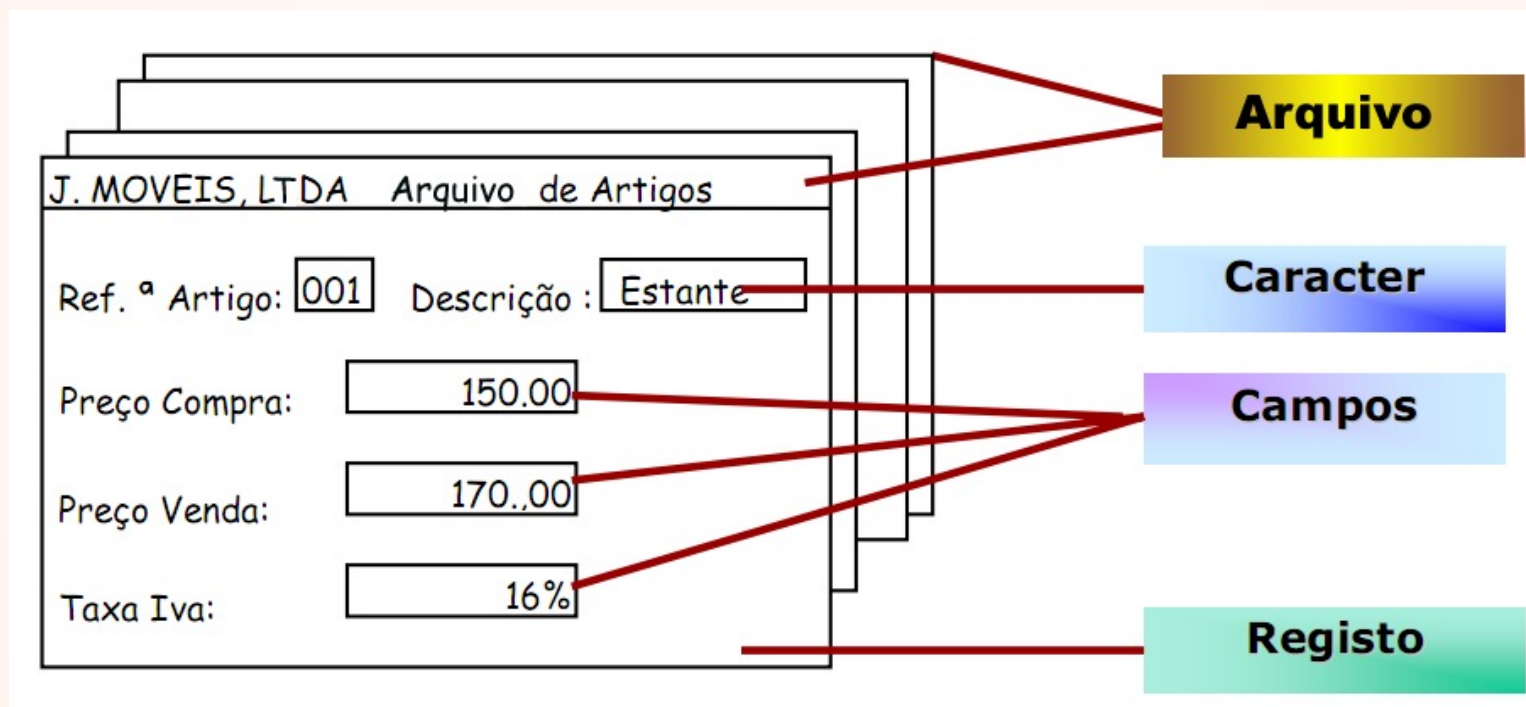
$$I_3 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

matriz identidade de ordem 3

# **Estruturas de Dados Heterogêneas**

## **Registros e Arquivos**

# Registros e Arquivos



# Registros

- Registro é uma estrutura composta por um conjunto de variáveis de tipos diferentes (campos) que estão relacionados e são referenciados por um mesmo indicador que é o nome do registro.

| Nome   | Fone      | Cidade  |
|--------|-----------|---------|
| Elízio | 1111-2222 | Mossoró |

- Um campo é a menor unidade destinada ao armazenamento de valores existentes em um registro.

| Nome   | Fone      | Cidade  |
|--------|-----------|---------|
| Elízio | 1111-2222 | Mossoró |

- Cada campo contém um tipo de dado.

# Declaração de **Registros**

## Declaração de Tipos

```
reg = REGISTRO
 nome: caracter;
 fone: inteiro;
 cidade: caracter;
```

## Declaração de Variáveis

```
reg: contato;
```

- Para fazermos a leitura de um campo de um registro, basta indicarmos o nome do registro seguido pelo nome do campo.

```
leia(contato.nome) ;
leia(contato.fone) ;
leia(contato.cidade) ;
```

# Arquivos

- Um Arquivo é um conjunto de registros do mesmo tipo (possuem o mesmo formato padrão (layout)). Os arquivos no formato texto geralmente são encontrados com nomes com a extensão padrão TXT (abreviação de **TEXT** – texto em inglês). Arquivos texto são considerados universais, pois podem ser facilmente lidos por quaisquer programas.

Arquivo de dados

| Nome   | Fone      | Cidade       |
|--------|-----------|--------------|
| Elízio | 1111-2222 | Mossoró      |
| Breno  | 8888-3333 | Apodi        |
| Hélio  | 8887-5746 | Assu         |
| Diego  | 9082-3856 | Natal        |
| Alice  | 3862-8473 | João Pessoa  |
| Felipe | 7563-5009 | Areia Branca |
| Carlos | 3211-4957 | Recife       |

# Declaração de Arquivos

## Declaração de Tipos

```
reg = REGISTRO
```

```
 nome: character;
```

```
 fone: character;
```

```
 cidade: character;
```

```
 arq = ARQUIVO DE reg;
```

## Declaração de Variáveis

```
 arq: contatos;
```

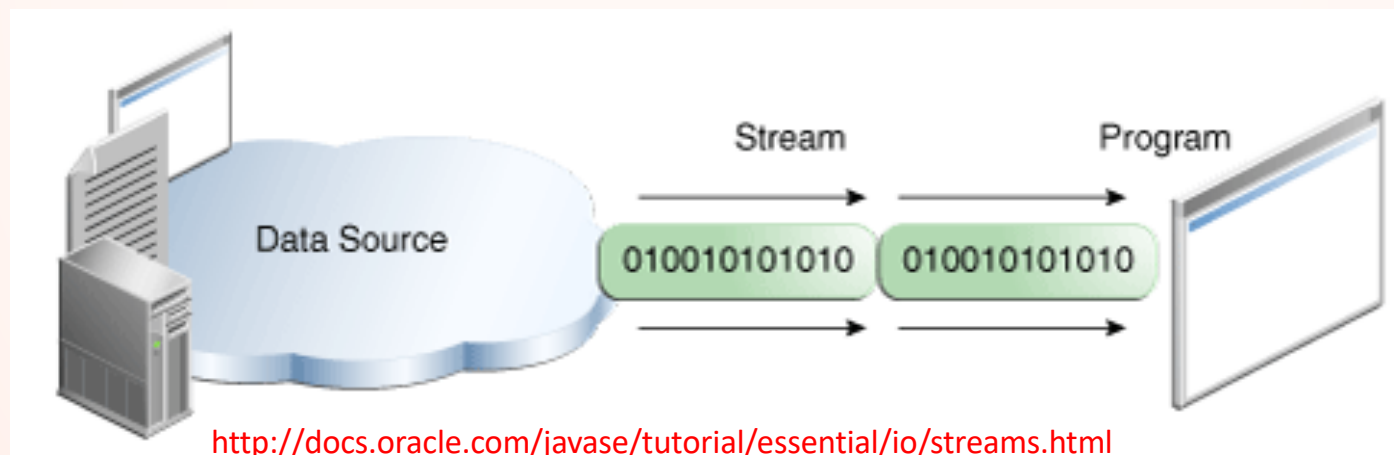
- Para fazermos a leitura de um campo de um registro de um arquivo, basta indicarmos o nome do arquivo, o nome do registro seguido pelo nome do campo.

```
leia(contatos.reg.nome);
```



# Arquivos em JAVA

- Toda operação de entrada e saída em uma aplicação Java faz uso de um objeto que identifica um fluxo (*stream*) de informações.
- Um *stream* é uma sequência de dados transmitidos de uma fonte de entrada para um destino de saída e vice-versa.



# Arquivos em JAVA

- Programas Java implementam o processamento de arquivos utilizando as classes do pacote **java.io**.
- A hierarquia de classes oferecida por este pacote oferece mais de 50 classes distintas para o processamento de entrada e saída em arquivos baseados em bytes e caracteres e arquivos de acesso aleatório.

# Arquivos em JAVA

- As principais classes/interfaces são:
  - `FileInputStream`: para entrada baseada em bytes de um arquivo.
  - `FileOutputStream`: para saída baseada em bytes para um arquivo.
  - `RandomAccessFile`: para entrada e saída baseada em bytes de e para um arquivo.
  - **`FileReader`: para entrada baseada em caracteres de um arquivo.**
  - **`FileWriter`: para saída baseada em caracteres para um arquivo.**

# Arquivos em JAVA

```
String[] nomes = {"Elízio", "Breno", "Hélío"};
String[] fones = {"1111-2222", "8888-3333", "8887-5746"};
String[] cidades = {"Mossoró", "Apodi", "Assu"};
FileWriter arq = new FileWriter("contatos.txt");
PrintWriter gravaarq = new PrintWriter(arq);
gravaarq.println("=====");
gravaarq.println("Nomes || Telefones || Cidades ");
for (int i = 0; i < nomes.length; i++) {
 gravaarq.println("=====");
 gravaarq.print(nomes[i] + " || ");
 gravaarq.print(fones[i] + " || ");
 gravaarq.println(cidades[i]);
}
gravaarq.println("=====");
arq.close();
gravaarq.close();}
```

## Exercícios

- 1) Elaborar um programa que crie o arquivo **contatos.txt** com os 03 registros abaixo.

| Nome   | Fone      | Cidade  |
|--------|-----------|---------|
| Elízio | 1111-2222 | Mossoró |
| Breno  | 8888-3333 | Apodi   |
| Hélio  | 8887-5746 | Assu    |

- 2) Refatorar o programa matriz Identidade l=3 para criar um arquivo **matriz.txt** com os seus valores.

# Records

# RECORDS

---

- **Definição**

- A partir do Java 14, já é possível criar classes imutáveis em Java de forma concisa.
- Os Registros (**Records**) simplificam a criação de classes que representam dados e incluem automaticamente métodos como *'equals()'*, *'hashCode()'*, *'toString()'*, e métodos de acesso.

```
public record Pessoa(String nome, int idade) {}

// Criando uma instância do registro
Pessoa pessoa = new Pessoa("Alice", 30);

// Acessando os campos do registro
String nome = pessoa.nome();
int idade = pessoa.idade();

// Usando o método gerado automaticamente toString()
System.out.println(pessoa); // Saída: Pessoa[nome=Alice, idade=30]
```

# RECORDS

---

- Classe Pessoa
  - Muito “verboso”.

```
import java.util.Objects;

public class Pessoa {
 private String nome;
 private int idade;

 // Construtor
 public Pessoa(String nome, int idade) {
 this.nome = nome;
 this.idade = idade;
 }

 // Métodos de acesso para o campo nome
 public String getNome() {
 return nome;
 }
}
```

```
public void setNome(String nome) {
 this.nome = nome;
}

// Métodos de acesso para o campo idade
public int getIdade() {
 return idade;
}

public void setIdade(int idade) {
 this.idade = idade;
}
```



# RECORDS

---

- Classe Pessoa
  - Muito “verboso”

```
// Método equals para comparar objetos Pessoa
@Override
public boolean equals(Object o) {
 if (this == o) return true;
 if (o == null || getClass() != o.getClass()) return false;
 Pessoa pessoa = (Pessoa) o;
 return idade == pessoa.idade &&
 Objects.equals(nome, pessoa.nome);
}

// Método hashCode para gerar um código hash para objetos Pessoa
@Override
public int hashCode() {
 return Objects.hash(nome, idade);
}
```

```
// Método toString para retornar uma representação de string da Pessoa
@Override
public String toString() {
 return "Pessoa{" +
 "nome='" + nome + '\'' +
 ", idade=" + idade +
 '}';
}
```

# CLASSES vs RECORDS

- Tabela Comparativa

| Aspecto                             | Classes                                                                           | Records                                                                                                 |
|-------------------------------------|-----------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------|
| Definição                           | Define um novo tipo de objeto com campos e métodos personalizados.                | Define uma nova forma de dados imutáveis, principalmente para armazenamento de dados.                   |
| Mutabilidade                        | Pode ser mutável ou imutável, dependendo da implementação.                        | Sempre imutável por padrão.                                                                             |
| Herança                             | Pode estender outras classes e implementar interfaces.                            | Não pode estender outras classes, mas pode implementar interfaces.                                      |
| Método Adicional                    | Pode ter qualquer número de métodos com comportamento personalizado.              | Pode ter métodos adicionais, mas geralmente usados para funcionalidades básicas relacionadas aos dados. |
| Acesso a Membros                    | Pode ter controle granular sobre acesso aos seus membros (encapsulamento).        | Sempre transparente - componentes (campos) são sempre acessíveis diretamente.                           |
| Controle de Comportamento Adicional | Pode ter comportamentos personalizados adicionais em seus métodos.                | Comportamentos adicionais são normalmente associados à manipulação de dados.                            |
| Equals, hashCode e ToString         | Precisa ser implementado manualmente para garantir a lógica de igualdade correta. | Gerado automaticamente a partir dos componentes (campos).                                               |

# **Novos Métodos para Leitura e Escrita em Arquivos**

# LEITURA E GRAVAÇÃO EM ARQUIVOS

---

- **Definição**

- O Java 11 facilitou enormemente a tarefa de ler/escrever String em arquivos com a criação dos métodos `readString()` e `writeString()`.

- **Leitura**

```
String texto = Files.readString(Path.of("arquivo"));
```

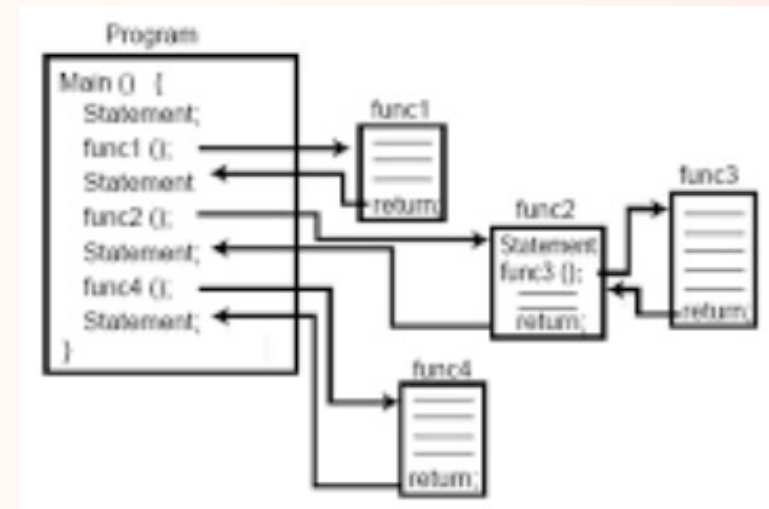
- **Escrita**

```
Files.writeString(Path.of("arquivo"), novoConteudo);
```

# Modularização

# Introdução

- Modularização significa separar o código em pequenos trechos reutilizáveis (procedimentos), de tal forma que seja fácil tanto reaproveitar estes trechos de código, bem como fazer a sua manutenção sem impactar diretamente os outros trechos de código.
- Com o uso do recurso de modularização, a programação se torna mais inteligente e menos suscetível a erros.



Fonte: Maurício Braga- <https://slideplayer.com.br/slide/65699/>

# Procedimentos

- Um procedimento é um bloco que contém um conjunto de instruções. Possui um nome e pode receber parâmetros e retornar valores.

## Declaração de Variáveis

```
inteiro: numero;
```

## Declaração de Procedimentos

```
Procedimento <nome> (<parâmetros?>)
```

```
 Início
```

```
 Declaração de Variáveis
```

```
 ...
```

```
 ...
```

```
 Fim
```

```
Início
```

```
 procedimento ();
```

```
Fim
```

# Procedimentos

## Declaração de Variáveis

```
inteiro: numero;
```

## Declaração de Procedimentos

```
Procedimento tabuada(n:inteiro)
```

```
Início
```

### Declaração de Variáveis

```
inteiro: i, total;
```

```
for(i=1; i<=10; i++)
```

```
{
```

```
 total = n * i;
```

```
 System.out.println
```

```
 ("Numero:" + numero + "x" + i + "=" + total);
```

```
}
```

```
Fim
```

```
...
```

```
Início
```

```
 leia(numero);
```

```
 tabuada(numero);
```

```
Fim
```



# Modularização em Java

- Há quatro formas de modularização em Java: (1) métodos, (2) classes, (3) pacotes e (4) modules (novidade do Java 9).

## → (1) MÉTODOS

- Os métodos são serviços implementados na forma de um conjunto de instruções em Java que realizam alguma tarefa específica e podem, como resultado, retornar um valor.

```
boolean método() {
 if (condição) {
 instrução;
 return true;
 }
 resto do método
 return false;
}
```

A palavra-chave **return** especifica o que será retornado após a chamada a um método. Se o método for **void**, não haverá o uso do **return**.

# Modularização em Java

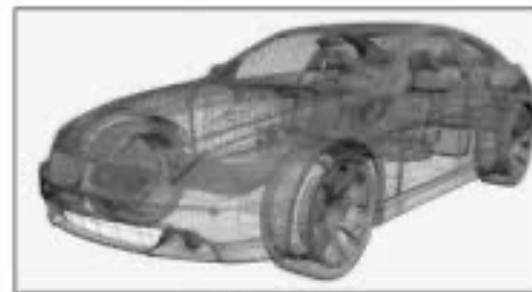
## → (1) MÉTODOS

```
public static void tabuada(int n)
{
 int i, total;
 for(i=1; i<=10; i++)
 {
 total = n * i;
 System.out.println("Numero: "+numero+"x"+i+"="+total) ;
 }
}
```

# Modularização em Java

## → (2) CLASSES

- Uma classe é um modelo ou protótipo que define as propriedades e métodos (comportamento) comuns a um conjunto de objetos;



Classe



Objeto

- Classes são “moldes” que definem as variáveis e os métodos comuns a todos os objetos de um determinado tipo.

# Modularização em Java

## → (2) CLASSES

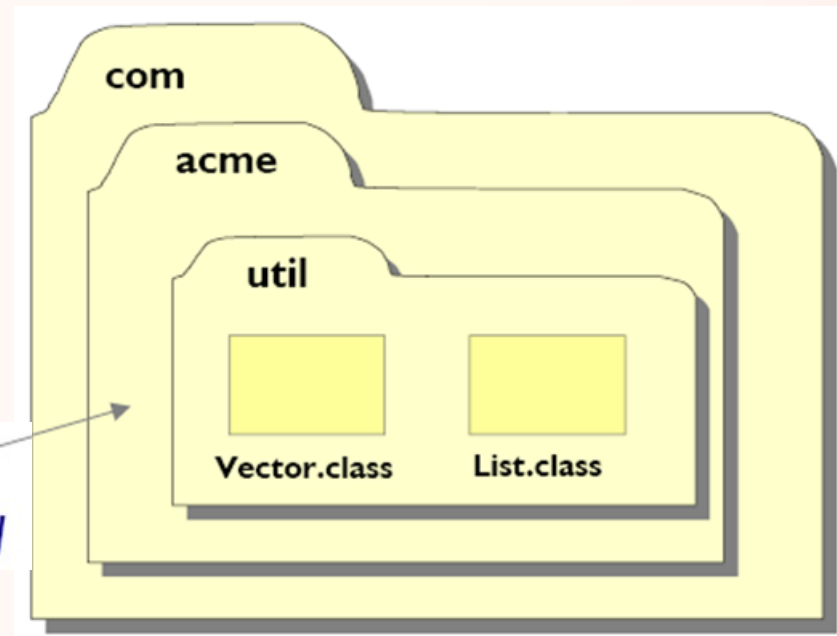
```
public class Matematica {
 public static void tabuada(int n)
 {
 int i, total;
 for(i=1; i<=10; i++){
 total = n * i;
 System.out.println("Numero:" + numero + "x" + i + "=" + total);
 }
 }
 public static void main(String args[]){
 //leia número;
 tabuada(numero);
 }
}
```

# Modularização em Java

## → (3) PACOTES

- São um conjunto de classes e interfaces relacionadas e outros pacotes que provêm acesso protegido e gerenciamento de espaço de nomes (namespaces).

*pacote*  
*com.acme.util*



# Modularização em Java

## → (3) PACOTES

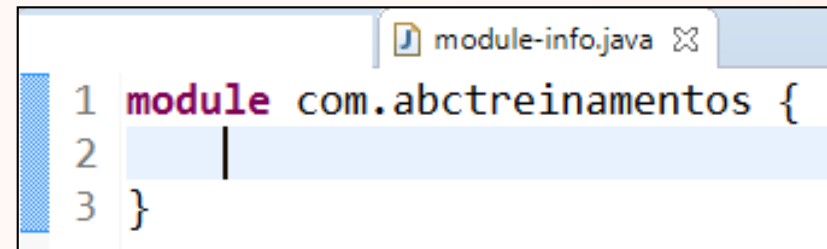
```
package unidade5;
public class Matematica {
 public static void tabuada(int n)
 {
 ...
 }
 public static void main(String args[]){
 //leia número;
 tabuada(numero);
 }
}
```

# Modularização em Java

## → (4) MODULES

- O Java Platform Module System especifica um formato (jar) de distribuição para coleções de código java e recursos associados e um arquivo **module-info.java**.

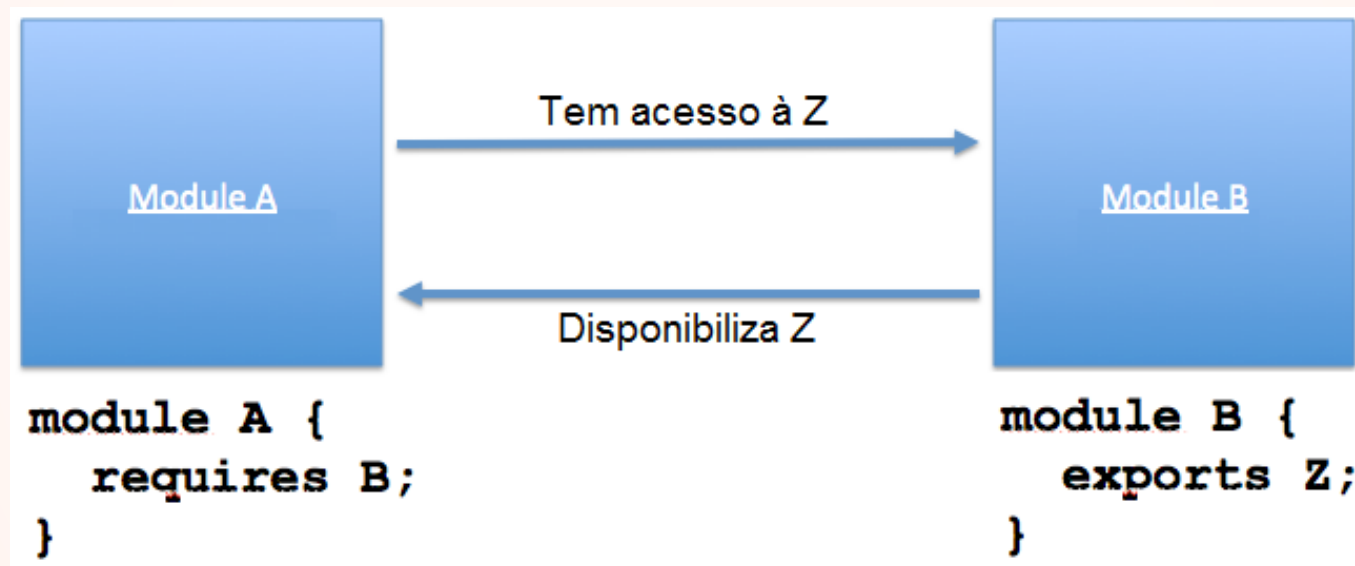
- O arquivo **module-info.java** declara:
  - o nome exclusivo do nosso módulo;
  - quais outros módulos o nosso módulo depende;
  - quais pacotes devem ser exportados para serem usados por outros módulos.



```
1 module com.abctreinamentos {
2 |
3 }
```

# Modularização em Java

→ (4) MODULES





## *Exercícios*

- 1) Elaborar o programa **unidade5.Matematica** com a função **tabuada(int n)**.
- 2) Refatorar o programa **unidade4.Fibonacci** com a função **fibonacci(int n)**.

# Orientação a Objetos

# O Mundo é Composto por Objetos!



# O que é um Objeto?

## DEFINIÇÕES

- Uma Abstração;
- Alguma coisa que faz sentido no domínio da aplicação.

## UTILIDADES

- Facilita a compreensão;
- Oferece base real para implementação no computador.

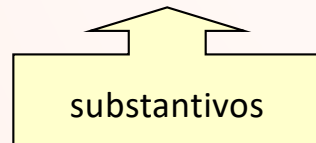


# O que é um Objeto?

- Um objeto é representado por um conjunto de **atributos** (também conhecidos como propriedades) e por um conjunto de **métodos** (que definem o comportamento de um dado objeto):

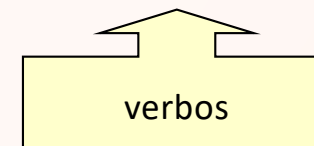
## Atributos

Motor  
Cor  
Potência  
Fabricante



## Métodos

Acelerar  
Retroceder  
Parar  
Abastecer



# Exemplo de um Objeto

## Atributos

Motor: V12  
Cor: Azul  
Potência: 600cv  
Fabricante: Ferrari



substantivos

## Métodos

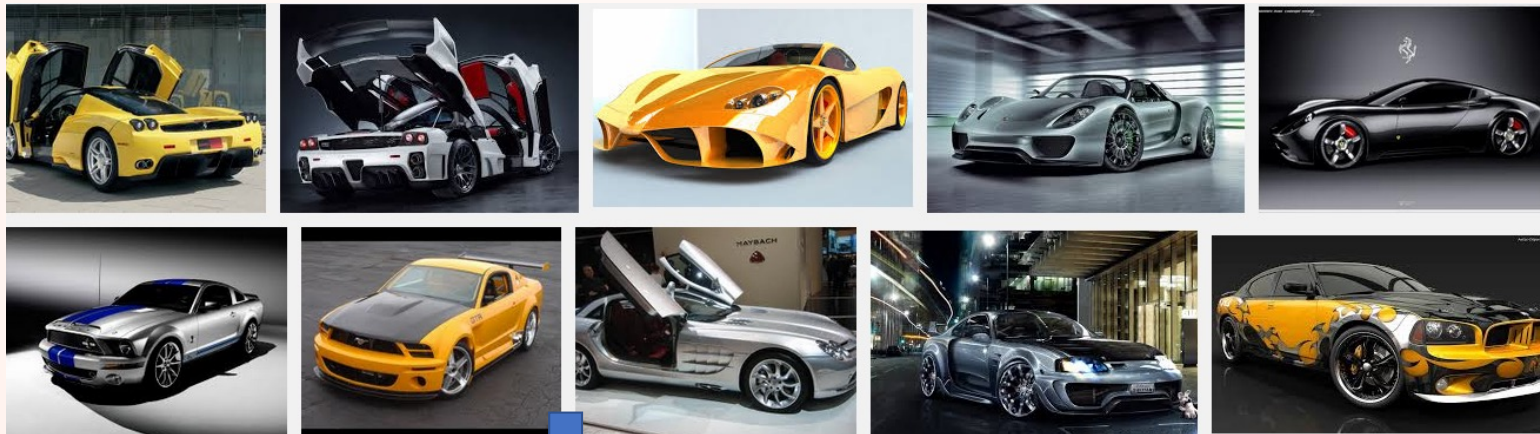
Acelerar  
Retroceder  
Parar  
Abastecer

verbos

# Classe de Objetos

## DEFINIÇÃO

- O grupo de objetos que possuem os mesmos atributos e métodos diz-se que pertencem à mesma classe.



**Classe Carro**

## *Exercícios*

- 1) Identificar as classes do programa **unidade4.IRPF**.
- 2) Para criar um sistema de informação que gerencie o aluguel de uma frota de carros, quais são as classes necessárias?



# **Algoritmos com Qualidade**

## Máximas de **Programação**

**“Qualquer tolo consegue escrever código para um computador entender. Bons programadores escrevem código que humanos consigam entender.”**

*Martin Fowler*

# Máximas de **Programação**

- Segundo os professores Guimarães e Lages, são 10 as regras básicas que devem ser seguidas para a elaboração de algoritmos com qualidade:
- **1) Algoritmos devem ser feitos para serem lidos por seres humanos;**
- **2) Escreva os comentários no momento em que estiver escrevendo o algoritmo;**
- **3) Os comentários deverão acrescentar alguma coisa;**
- **4) Use comentários no prólogo;**



## Máximas de Programação

- Segundo os professores Guimarães e Lages, são 10 as regras básicas que devem ser seguidas para a elaboração de algoritmos com qualidade:
- **5) Utilize espaços em branco para melhorar a legibilidade;**
- **6) Escolha nomes representativos para suas variáveis;**
- **7) Um comando por linha é suficiente;**
- **8) Utilize parênteses para aumentar a legibilidade e prevenir-se contra erros;**

## Máximas de **Programação**

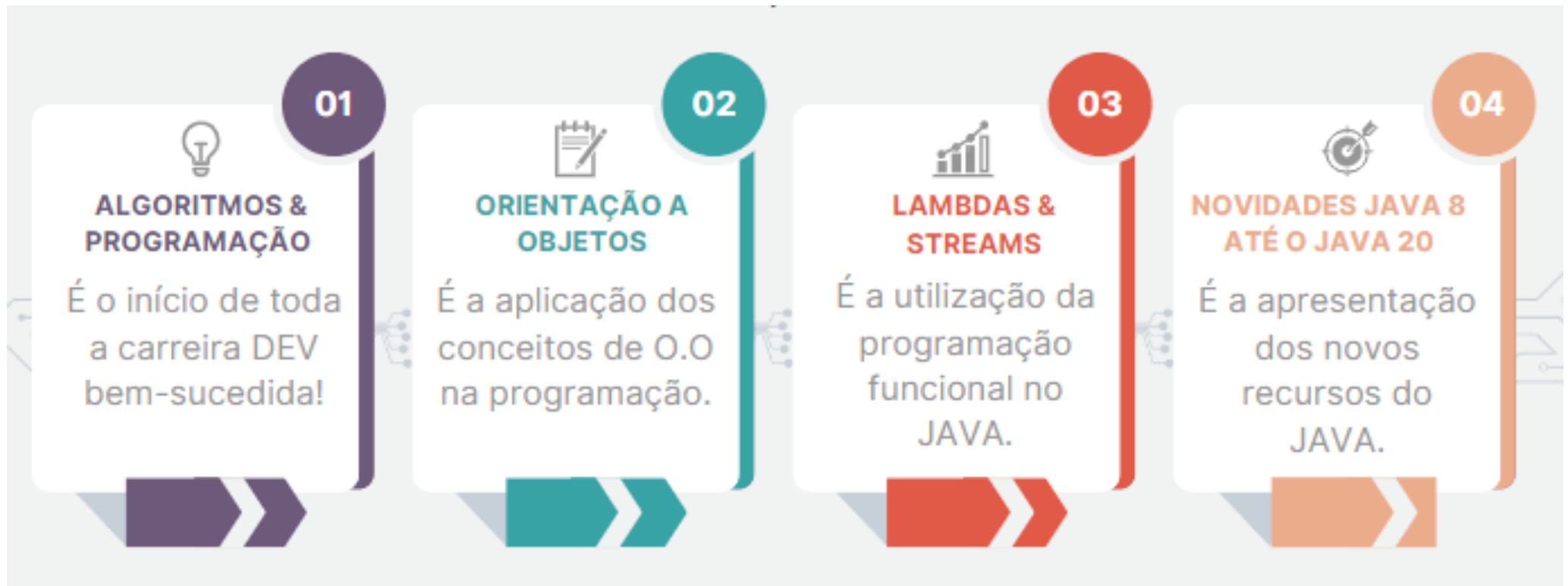
- Segundo os professores Guimarães e Lages, são 10 as regras básicas que devem ser seguidas para a elaboração de algoritmos com qualidade:
- **9) Utilize “identação” para mostrar a estrutura lógica do algoritmo;**
- **10) Lembre-se: toda vez que for feita uma modificação no algoritmo, os comentários associados devem ser alterados, e não apenas os comandos.**

## Exercício

- 1) Analise os mais de 50 programas Java criados ao longo deste curso e verifique se as 10 regras básicas para a construção de algoritmos com qualidade são seguidas.

# Conclusão

# A Nossa Trilha **DEV JAVA FULL STACK**

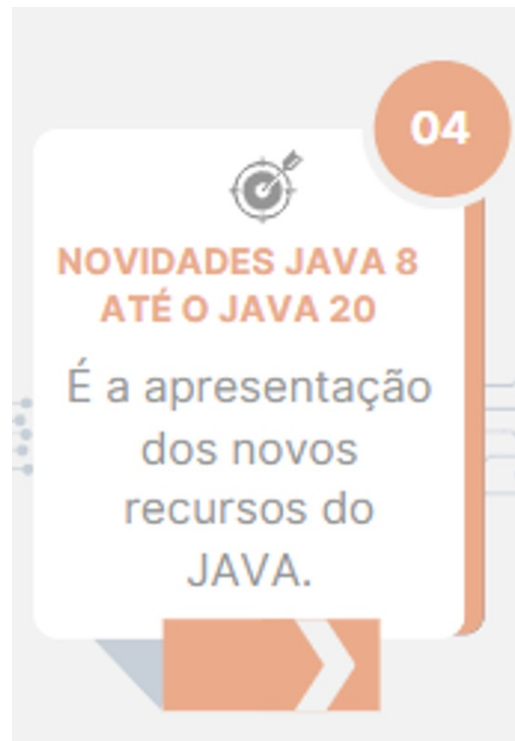




# A Nossa Trilha **DEV JAVA FULL STACK**



# A Nossa Trilha **DEV JAVA FULL STACK**



## Curso

### Domine as Inovações do Java com Spring Boot & MongoDB

As Mudanças mais Importantes  
do Java 8 até o Java 20



**Prof. Msc. Antonio B. C. Sampaio Jr**  
engenheiro de software & professor

@abctreinamentos  
@amazoncodebr

www.abctreinamentos.com.br  
www.amazoncode.com.br



# A Nossa Trilha DEV JAVA FULL STACK



Curso

## Aplicações JAVA com SPRING BOOT



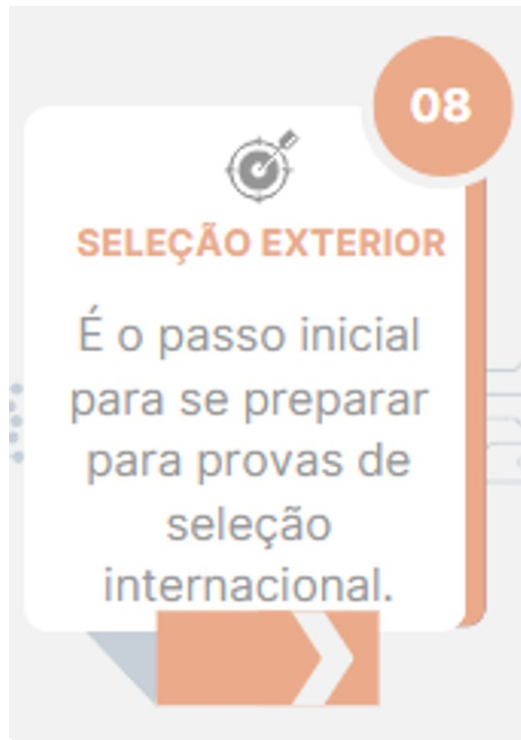
**Prof. Msc. Antonio B. C. Sampaio Jr**  
ENGENHEIRO DE SOFTWARE & PROFESSOR

@abctreinamentos  
@amazoncodebr

www.abctreinamentos.com.br  
www.amazoncode.com.br



# A Nossa Trilha **DEV JAVA FULL STACK**



**08**

**SELEÇÃO EXTERIOR**

É o passo inicial para se preparar para provas de seleção internacional.

## Curso

### Aprenda a Resolver Questões Java para Seleção no Exterior

Resolução de 18 Questões de Java



**Prof. Msc. Antonio B. C. Sampaio Jr**  
engenheiro de software & professor

@abctreinamentos  
@amazoncodebr

www.abctreinamentos.com.br  
www.amazoncode.com.br

