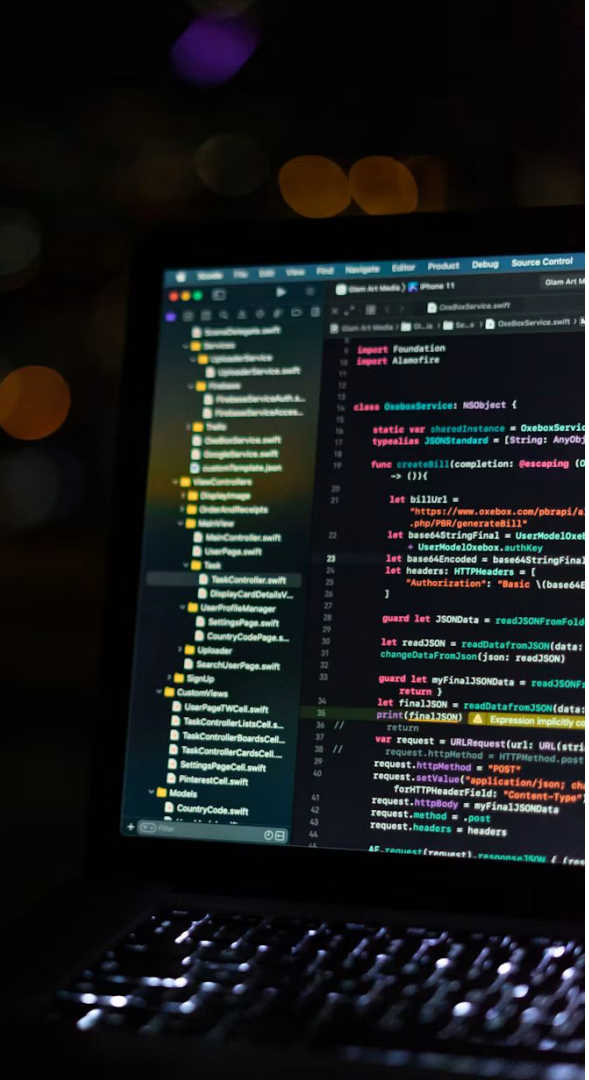


5954018 - Programação Orientada a Objetos

Aula 7 - Arrays

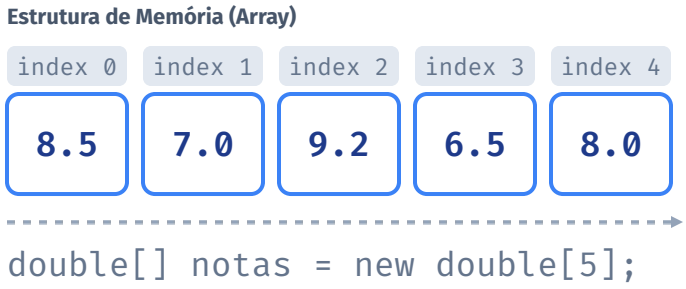
Prof. Dr. Denis Martins

DCM | FFCLRP | USP

Arrays em Java

Arrays unidimensionais e multidimensionais

> *"Como armazenar e processar vários valores do mesmo tipo sem criar dezenas de variáveis?"*



Objetivos de Aprendizagem | Autoavaliação de progresso

Utilize esta lista como um guia para verificar se você consolidou os conceitos essenciais desta aula.

1 Conceito de Array
Explicar o que é um array e sua estrutura de armazenamento.

2 Sintaxe Inicial em Java
Declarar, criar e inicializar arrays na linguagem Java.

3 Acesso por Índices
Acessar e modificar elementos utilizando seus respectivos índices.

4 Iteração e Laços
Percorrer arrays utilizando as estruturas for e for-each.

5 Propriedade length
Utilizar o atributo length para obter o tamanho do array.

6 Classe Utilitária Arrays
Utilizar operações úteis do pacote java.util.Arrays.

7 Arrays de Objetos
Criar e gerenciar arrays contendo referências a objetos.

8 Arrays Multidimensionais
Criar e percorrer estruturas de arrays multidimensionais.

Créditos: **slides baseados no material de aula original do Prof. Clever Farias.**

Problema: muitas variáveis

```java
double nota1 = 8.5;
double nota2 = 7.0;
double nota3 = 9.2;
double nota4 = 6.5;
double nota5 = 8.0;
```

Perguntas:

- E se precisarmos armazenar as notas de 50 alunos?
- Como calcularíamos a média dessas 50 variáveis?

Criar uma variável para cada valor não é uma solução escalável.

Visualização na Memória

Código Repetitivo

Gerenciar dezenas de variáveis isoladas dificulta a manutenção, reutilização e cria alto risco de erros.

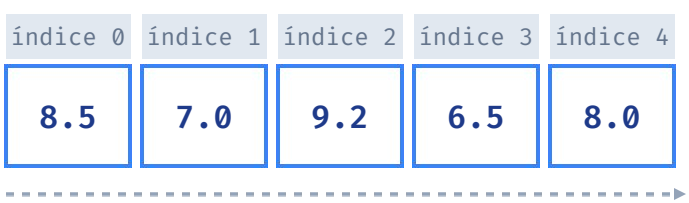
Solução: Array

Um **array** é uma estrutura que permite armazenar vários elementos do **mesmo tipo** utilizando uma única variável.

```
double[] notas = {8.5, 7.0, 9.2, 6.5, 8.0};
```

Em Java, o primeiro índice é 0.

Visualização na Memória



Criando um Array

Declaração e Instanciação em duas etapas:

```
int[] numeros;
numeros = new int[5];
```

Sintaxe Simplificada (Em uma única linha):

```
int[] numeros = new int[5];
```

Ao instanciar com `new int[5]`, os elementos são inicializados automaticamente com `0`.

Visualização na Memória

Valores Padrão ao Instanciar

Tipo de Dado vs. Valor Inicial de Inicialização	
`int`	`0`
`double`	`0.0`
`boolean`	`false`
`objetos (String, etc.)`	`null`

```java
int[] valores = new int[3];
// Elementos alocados e preenchidos com 0 automaticamente
```

Visualização na Memória

Importante:
Em Java, a memória alocada por **new** é sempre zerada automaticamente com o valor padrão do tipo.

Inicialização Direta

Inicialização Direta (Sintaxe Literal)

```
int[] numeros = {10, 20, 30, 40, 50};
```

Comparação: Alocação Passo a Passo

```
int[] numeros = new int[5];
numeros[0] = 10; numeros[1] = 20;  ...
```

Importante:
A inicialização direta é conveniente quando os valores já são conhecidos.

Visualização na Memória

Acessando Elementos por Índice

Declaração e Leitura

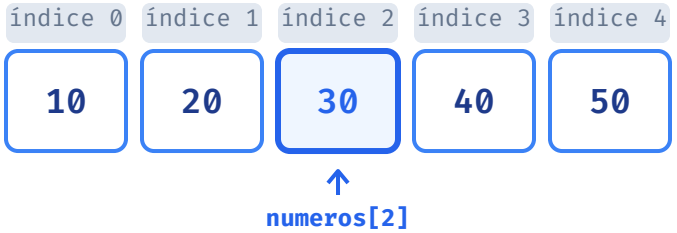
```
int[] numeros = {10, 20, 30, 40, 50};
System.out.println(numeros[0]); // Imprime 10
```

Acesso Específico

```
System.out.println(numeros[2]); // Imprime 30
// O índice indica a posição exata do elemento na memória.
```

Importante:
Os índices de um array sempre começam em **0** e vão até **tamanho - 1**.

Visualização na Memória



Pergunta rápida

Qual valor será exibido por:
`System.out.println(numeros[3]);`**?**

A) 30
B) 40
C) 50

Modificando Elementos

Declaração Inicial

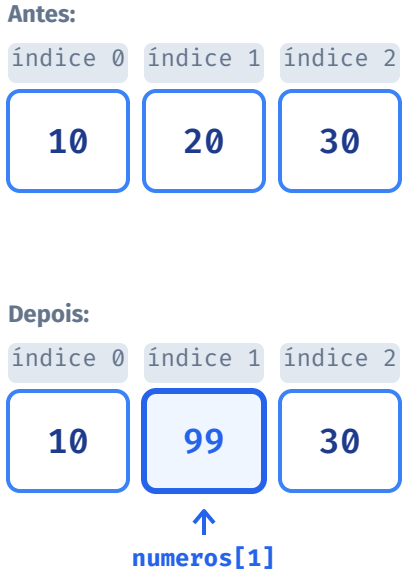
```
int[] numeros = {10, 20, 30};
```

Modificação de Valor

```
numeros[1] = 99;
// Atribui o novo valor 99 ao elemento no índice 1.
```

Importante:
O índice permite tanto **consultar** quanto **modificar** uma posição do array.

Visualização na Memória



Atividade Rápida: Preveja o Resultado

Código Java

```java
int[] valores = {5, 10, 15, 20};
valores[1] = 25;
valores[3] = valores[0];
System.out.println(valores[1]);
System.out.println(valores[3]);
```

Dica:
Acompanhe a alteração dos valores no array passo a passo para determinar o que será impresso pelas instruções **`println`**.

Visualização na Memória

[0]	[1]	[2]	[3]
5	25	15	5

Pergunta rápida

O que será impresso no console?

A) 10 e 20
B) 25 e 5
C) 25 e 20
D) 5 e 25

Quantos elementos existem?

Propriedade length

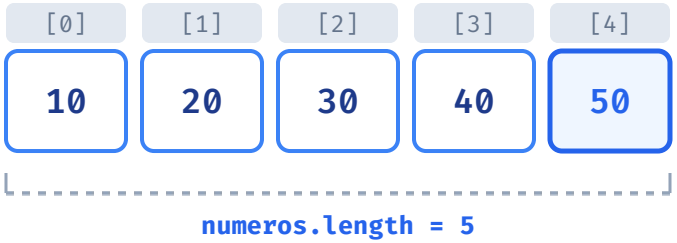
```
int[] numeros = {10, 20, 30, 40, 50};
System.out.println(numeros.length); // Imprime 5
```

Acessando o Último Elemento

```
int ultimoIndice = numeros.length - 1;
System.out.println(numeros[ultimoIndice]); // Imprime 50
// Como os índices começam em 0, o último índice é sempre length - 1.
```

Importante:
A propriedade **`length`** retorna o número total de elementos no array (tamanho fixo), não o último índice válido.

Visualização na Memória



Resumo de Índices

Total de elementos: **5**
Índices válidos: **0, 1, 2, 3, 4**
Último índice: **length - 1 (4)**

Percorrendo um Array com `for`

Estrutura do Laço for

```
int[] numeros = {10, 20, 30, 40, 50};
   for (int i = 0; i < numeros.length; i++) {
   System.out.println(numeros[i]);
}
```

`int i = 0` — **Inicialização:** começa na primeira posição (índice 0).

`i < numeros.length` — **Condição:** permanece enquanto estiver dentro do array.

`i++` — **Incremento:** avança para a próxima posição a cada passo.

Visualização na Memória

[0]	[1]	[2]	[3]	[4]
10	20	30	40	50

Execução Passo a Passo (Iterações)

```
i = 0 → numeros[0] = 10
i = 1 → numeros[1] = 20
i = 2 → numeros[2] = 30
i = 3 → numeros[3] = 40
i = 4 → numeros[4] = 50
i = 5 → 5 < 5 (FALSO, para o loop)
```

Saída no Console:

```
10 20 30 40 50
```

Erro clássico: índice inválido

Código Java

```java
int[] numeros = new int[5];
System.out.println(numeros[5]); // Erro!
```

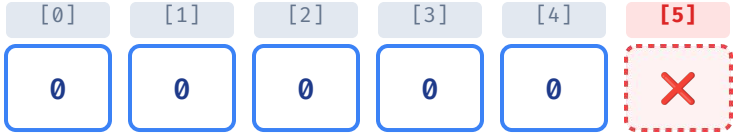
Exceção Lançada em Tempo de Execução

`ArrayIndexOutOfBoundsException`

Causa do Erro: O índice **5** está fora dos limites permitidos para este array.

- O tamanho declarado é **5**, o que significa que há exatamente 5 posições.
- Como os índices iniciam em **0**, o último índice válido é **4** (n - 1).

Visualização na Memória



Análise Visual do Acesso:

- Índices válidos: **0, 1, 2, 3, 4** (5 posições)
- Tentativa: **numeros[5]** ❌
- Resultado: Tentativa de ler além do fim do array na memória.

Como evitar este erro?

1. **Verifique os limites:** Sempre certifique-se de que o índice é estritamente menor que `length`.
2. **Use laços de forma segura:**
   `for (int i = 0; i < numeros.length; i++)`
3. **Atenção ao operador:** Use **<** e nunca **⩽** com `length`.

Percorrendo com `for-each`

Sintaxe do Laço for-each

```
int[] numeros = {10, 20, 30, 40, 50};
for (int numero : numeros) {
   System.out.println(numero);
}
```

Leitura Natural do Código:

"Para cada **numero** existente em **numeros**..."

Comparação: for Tradicional vs. for-each

`for tradicional`

- Permite acessar o índice
- Permite alterar posição
- Útil quando índice importa

`i + ": " + notas[i]`

`for-each`

- Trabalha diretamente com valores
- Sintaxe mais simples e limpa
- Útil para percorrer elementos

`for (double nota : notas)`

Visualização da Iteração

Fluxo de Execução Simplificado

1ª iteração: numero = 10
2ª iteração: numero = 20
3ª iteração: numero = 30
4ª iteração: numero = 40
5ª iteração: numero = 50

Fim do array → Encerra

Saída no Console:

```
10 20 30 40 50
```

Exercício 1: Busca no Array

Dado do Problema

```
int[] numeros = { 42, 15, 8, 23, 16, 4, 30, 11 };
```

Objetivo e Requisitos Técnico

Solicite um número via Scanner e informe se foi encontrado e em qual índice.

- **Percorrer o array com estrutura de repetição (for/while).**
- **Utilizar índices; não usar índice fixo para a busca.**
- **Imprimir mensagem de "Número não encontrado" se ausente.**

Desafio (Múltiplas Ocorrências)

Modifique o array para: `{ 42, 15, 8, 23, 15, 4, 30, 15 }`
Apresente todos os índices onde o número ocorre e a quantidade total de ocorrências.

Exemplo de Saída Esperada

Console - Busca Simples

```
Digite um número: 23
Número encontrado! Índices: 3

Digite um número: 100
Número não encontrado.
```

Console - Desafio

```
Digite um número: 15
Número encontrado nos índices: 1 4 7
Total de ocorrências: 3
```

Dica de Implementação:

- **Use uma variável `boolean encontrada = false;` para controlar se o número foi localizado na busca simples.**
- **No desafio, utilize um `int contador = 0;` para acumular quantas vezes o número aparece durante o laço `for`.**

A classe Arrays

Importação e Conceito

```
import java.util.Arrays;
```

Arrays é uma classe utilitária do Java que oferece diversos métodos **static** para manipular e formatar arrays de maneira eficiente.

Principais Métodos Estáticos

- **`Arrays.toString()`** → Retorna uma representação em String dos elementos
- **`Arrays.sort()`** → Ordena os elementos do array em ordem crescente
- **`Arrays.equals()`** → Compara se dois arrays são totalmente iguais

Por que chamamos com o nome da classe?

Porque **`toString()`**, **`sort()`** e **`equals()`** são métodos **`static`**, não dependendo de uma instância para serem executados.

Exemplo Prático e Chamada

Chamada do Método

```
int[] numeros = {10, 20, 30};
String res = Arrays.toString(numeros);
System.out.println(res);
```

Saída formatada no Console:

```
"[10, 20, 30]"
```

Vantagem do toString():

Sem o **`Arrays.toString()`**, imprimir um array diretamente exibirá apenas a referência de memória (ex: `[I@15db9742`).

Com ele, os valores são formatados de forma legível entre colchetes!

Visualizando com `Arrays.toString()`

Problema: Imprimir Array Direitamente

```
int[] numeros = {10, 20, 30};
System.out.println(numeros); // Imprime a referência de memória!
Saída: [I@15db9742
```

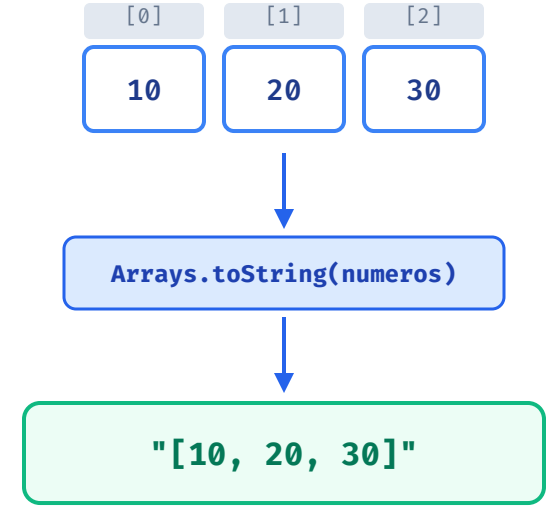
Solução: Utilizando Arrays.toString()

```
int[] numeros = {10, 20, 30};
System.out.println(Arrays.toString(numeros));
Saída Formatada: "[10, 20, 30]"
```

O que acontece internamente?

A classe `Arrays` percorre cada elemento do vetor e constrói uma `String` legível contendo todos os valores separados por vírgula e delimitados por colchetes `[ ... ]`.

Diagrama de Conversão



Retorno formatado:
Gera uma representação textual pronta para exibição no console, evitando imprimir ponteiros de memória.

Ordenando com `Arrays.sort()`

Exemplo de Código Java

```
int[] valores = {8, 5, 10, 7, 6};
Arrays.sort(valores);
System.out.println(Arrays.toString(valores));
Saída: "[5, 6, 7, 8, 10]"
```

Conceito

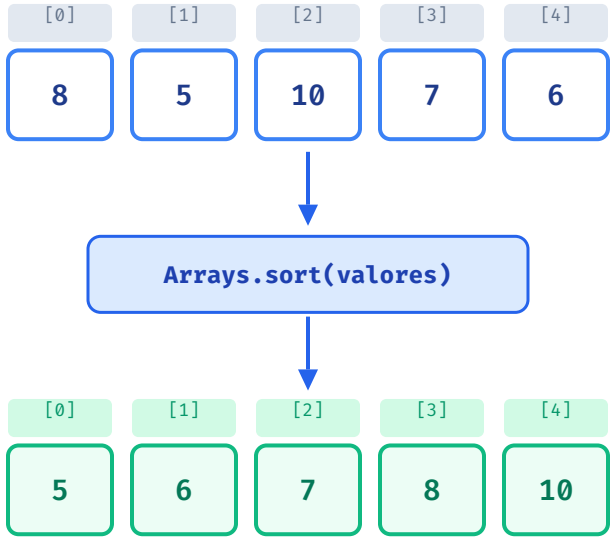
O método `Arrays.sort()` modifica o **próprio array** passado como parâmetro (ordenação *in-place*).

Os elementos são reordenados diretamente na memória, alterando a estrutura original do vetor para a ordem crescente.

Observações Importantes

- **Retorno void:** O método não retorna um novo array, ele altera o vetor existente.
- **Tipos Primitivos:** Utiliza uma variante do algoritmo Dual-Pivot Quicksort.

Diagrama de Ordenação



Alteração Direta:
Os elementos foram reorganizados na mesma referência de memória do array original.

Comparando com `Arrays.equals()`

Exemplo de Código Java

```
int[] a = {1, 2, 3};
int[] b = {1, 2, 3};
System.out.println(a == b); // false
System.out.println(Arrays.equals(a, b)); // true
```

Comparação de Referência (==)

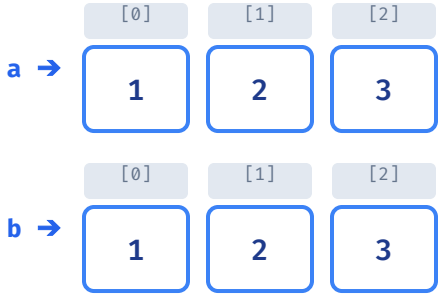
O operador == verifica se as variáveis apontam para o **mesmo endereço de memória**.

Mesmo com conteúdos idênticos, instâncias diferentes resultam em **`false`**.

Comparação de Conteúdo (Arrays.equals)

- **Comparação Elemento a Elemento:** O método percorre ambos os vetores verificando se cada posição possui o mesmo valor.
- **Resultado:** Retorna **`true`** se os arrays têm o mesmo tamanho e todos os elementos correspondentes forem iguais.

Explicação Visual na Memória



a == b (São o mesmo array na memória?)

`false`

Arrays.equals(a, b) (Os conteúdos são iguais?)

`true`

Exercício 2: Análise de Notas

Enunciado / Dado do Problema

```
double[] notas = {
  7.5, 8.0, 5.5, 9.0, 6.0, 8.5, 4.5, 7.0, 10.0, 6.5
};
```

Objetivos do Programa

Crie um programa que processe o array e apresente:

- 1. Todas as notas e a **média da turma**;
- 2. A **maior** e a **menor** nota;
- 3. Quantidade de **aprovados** (≥ 6.0) e **reprovados** (< 6.0);
- 4. Reordene com `Arrays.sort()` e exiba com `Arrays.toString()`.

Exemplo de Saída Esperada

Console Output

```
Notas: [7.5, 8.0, 5.5, 9.0, ...]
Média: 7.25
Maior nota: 10.0
Menor nota: 4.5
Aprovados: 8
Reprovados: 2
Notas ordenadas:
[4.5, 5.5, 6.0, 6.5, 7.0, ...]
```

Exercício 3: Simulação de Dados

Objetivos do Programa

Crie um programa em Java para simular a rolagem:

- 1. Utilize a classe **Random** para gerar os lançamentos de 2 dados (valores de 1 a 6);
- 2. Calcule a **soma dos dois valores** (variação de 2 a 12);
- 3. Simule o lançamento por exatamente **36.000 vezes**;
- 4. Use um **array unidimensional** para armazenar a frequência de cada soma;
- 5. Exiba os resultados em formato **tabular** (Soma, Total, Porcentagem).

Exemplo Random

```
Random rand = new Random();
rand.nextInt(6);
```

Arrays de Objetos: Pokemon[]

Declaração e Instanciação

```
// Declarando um array capaz de guardar 3 Pokémons
Pokemon[] pokemon = new Pokemon[3];
```

Estado Inicial da Memória

- **Armazenamento de Referências:** O array não cria os objetos **Pokemon** automaticamente. Em vez disso, ele reserva posições capazes de guardar *ponteiros/referências* para esses objetos.
- **Valor Padrão (null):** Assim que o array é instanciado, cada posição é inicializada com o valor padrão **null**, indicando que nenhuma posição aponta para um objeto real ainda.
- **Próximo Passo:** Para utilizar o vetor, é necessário instanciar individualmente cada objeto **Pokemon** e atribuí-lo às posições (ex: `pokemon[0] = new Pokemon();`).

Visualização na Memória (Heap)

Observação Importante:

Tentar acessar membros de um elemento antes de sua instanciação (ex: **pokemon[0].nome**) lançará uma exceção de ponteiro nulo (`NullPointerException`).

Arrays de Objetos: Inserção

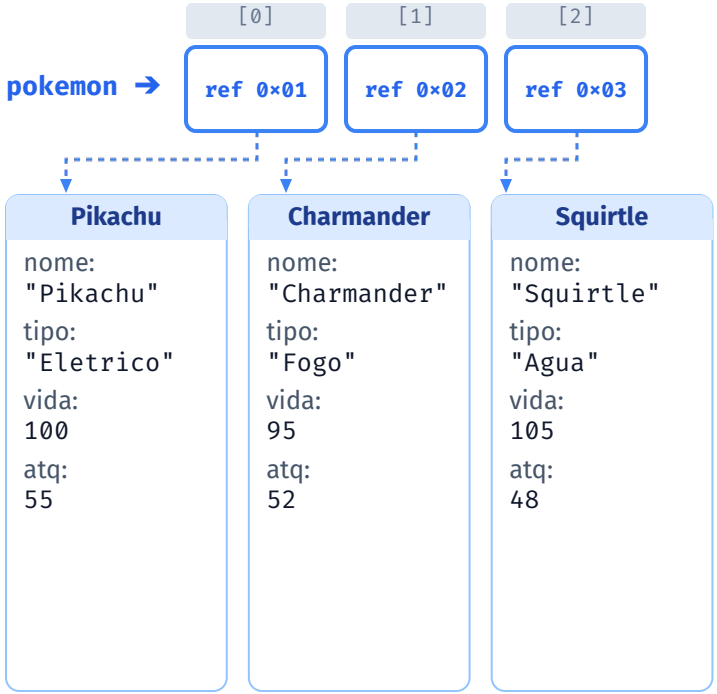
Instanciação e Atribuição de Objetos

```
pokemon[0] = new Pokemon("Pikachu", "Eletrico", 100, 55);
pokemon[1] = new Pokemon("Charmander", "Fogo", 95, 52);
pokemon[2] = new Pokemon("Squirtle", "Agua", 105, 48);
```

Instanciando Objetos no Array

- **Alocação no Heap:** Cada chamada ao operador **new** aloca dinamicamente um novo objeto **Pokemon** na memória Heap.
- **Atualização de Referências:** O valor **null** inicial de cada posição do vetor é substituído pelo ponteiro/endereço do objeto criado.
- **Acesso Seguro:** Após a atribuição, é possível acessar atributos e métodos (ex: **pokemons[0].nome**) sem gerar **NullPointerException**.

Visualização na Memória (Heap)



Arrays de Objetos: Percorrendo

Laço For-Each (Iteração de Objetos)

```
// Percorrendo cada referência do vetor
for (Pokemon pkm : pokemon) {
   System.out.println(pkm.getNome());
}
```

Importante:

O array armazena três classes ou três objetos?

- **Resposta:** Armazena **três referências para objetos** (instâncias da classe Pokemon) alocados no Heap.
- A classe é apenas o *molde/blueprint*. O vetor guarda os endereços para os dados reais em memória.

Execução Passo a Passo

- A cada iteração, a variável local **pkm** recebe temporariamente uma cópia do ponteiro/referência guardado no vetor.
- O método getNome() é chamado diretamente no objeto apontado no Heap.

Saída no Console & Estado

Saída do Programa (Console)

```
Pikachu
Charmander
Squirtle
```

Mapeamento na Memória:

1. pokemon[0] ➔ ref 0x01 ➔ **Pikachu**
2. pokemon[1] ➔ ref 0x02 ➔ **Charmander**
3. pokemon[2] ➔ ref 0x03 ➔ **Squirtle**

Lembre-se: O vetor contém *endereços* (ponteiros), não as instâncias fisicamente dentro dele. A classe é o tipo, as instâncias no Heap são os objetos.

Arrays de Objetos: Cuidado com null

Código Java (Erro de Execução)

```
Pokemon[] pokemon = new Pokemon[3];
System.out.println(pokemon[0].getNome()); // Erro!
```

O que acontece aqui?

- **Exceção Lançada:** O código compila normalmente, mas ao executar lança `NullPointerException`.
- **Causa:** A posição `pokemon[0]` armazena `null`. Tentar chamar `.getNome()` em uma referência nula resulta em erro.

Importante:

- **Instanciação do Array ≠ Instanciação dos Objetos:** Criar o array **não cria automaticamente os objetos** que serão armazenados nele.
- **Solução:** Antes de acessar métodos ou atributos, instancie o objeto na posição desejada (ex: `pokemon[0] = new Pokemon( ... );`).

Estado da Memória (Heap)

NullPointerException

Ao tentar chamar um método em uma referência nula, a JVM interrompe a execução do programa e exibe o erro:

```
Exception in thread "main"
java.lang.NullPointerException: Cannot
invoke "Pokemon.getNome()" because
"pokemon[0]" is null
```

Exercício 4: Time Pokémon

Enunciado & Entrada de Dados

```java
// Criar array para exatamente 6 Pokémon
Pokemon[] time = new Pokemon[6];
// Solicitar via Scanner: nome, tipo, hp, atk
Scanner sc = new Scanner(System.in);
for (int i = 0; i < time.length; i++) { ... }
```

Objetivos do Programa

- **1.** Cadastrar 6 objetos no array `time[]` via `Scanner`.
- **2.** Exibir o time completo com dados formatados.
- **3.** Descobrir o **Maior ATK**, **Maior HP** e a **Média de ATK**.
- **4. Desafio:** Filtrar e exibir Pokémon por tipo informado pelo usuário.

Requisitos Técnicos

- Utilizar `Pokemon[]` instanciando exatamente 6 objetos.
- Usar laços `for` ou `for-each` para percorrer o array.
- Usar `getters` (não acessar atributos `private` diretamente).
- **Proibido:** Criar variáveis isoladas (`pokemon1`, `pokemon2`...).

Exemplo de Saída Esperada

Console / Saída

```
═══ SEU TIME ═══
1 - Pikachu (Eletrico) HP: 100 | ATK: 55
2 - Charmander (Fogo) HP: 95 | ATK: 52
...
═══ ESTATÍSTICAS ═══
Maior ATK: Pikachu (55)
Maior HP: Blastoise (120)
Média de ATK do time: 58.3
═══ DESAFIO (Filtro por Tipo) ═══
Digite um tipo: Fogo
Pokémon do tipo Fogo: Charmander, Vulpix
```

Estrutura da Classe Pokemon

```java
public class Pokemon {
  private String nome, tipo;
  private int hp, atk;
  // Construtor e Getters
  public String getNome() { return
nome; }
  public int getAtk() { return atk; }
}
```

Matrizes: Array de Arrays

Declaração e Instanciação de Matriz

```
// Matriz de 2 linhas e 3 colunas (2×3)
int[][] matriz = new int[2][3];
```

Definição e Conceito

- **Arrays Multidimensionais:** Representam informações organizadas em mais de uma dimensão (ex: linhas e colunas).
- **Array de Arrays:** Em Java, não existe uma estrutura nativa de matriz; ela é implementada como um **array cujos elementos são outros arrays**.
- **Dimensões:** `matriz[2][3]` cria um array principal de tamanho 2 (linhas), onde cada posição aponta para um array de tamanho 3 (colunas).
- **Valores Padrão:** Como o tipo é `int`, todas as posições são inicializadas com o valor **0**.

Representação e Estado Inicial

Exemplo de Acesso/Atribuição

```
matriz[0][0] = 1;
matriz[0][1] = 2;
matriz[0][2] = 3;
matriz[1][0] = 4;
matriz[1][1] = 5;
matriz[1][2] = 6;
```

Matrizes: Inicialização & Acesso

Inicialização Direta de Matriz

```java
// Sintaxe simplificada para declarar e popular a matriz
int[][] matriz = { {1, 2, 3}, {4, 5, 6} };
```

Acessando Posições Específicas

```java
// Acesso via indexação: matriz[linha][coluna]
System.out.println(matriz[0][2]); // Imprime: 3
System.out.println(matriz[1][1]); // Imprime: 5
```

Pergunta Rápida

Qual valor está armazenado em `matriz[1][2]`?

Visualize em Java Tutor: https://pythontutor.com/java.html

Visualização da Estrutura (2x3)

Mapeamento de Índices

Regra de Acesso

- **Índices:** Primeiro valor especifica a *linha*, o segundo a *coluna*. Ex: `matriz[linha][coluna]`.
- `matriz[0][2]` ➔ Linha 0, Coluna 2 ➔ **3**
- `matriz[1][1]` ➔ Linha 1, Coluna 1 ➔ **5**
- `matriz[1][2]` ➔ Linha 1, Coluna 2 ➔ **6**

Matrizes: Percorrendo & length

Laço Aninhado (Percorrendo a Matriz)

```
for ( int i = 0; i < matriz.length; i++) {
   for ( int j = 0; j < matriz[i].length; j++) {
      System.out.print(matriz[i][j] + " ");
   }
   System.out.println();
}
```

Funcionamento do Loop Aninhado

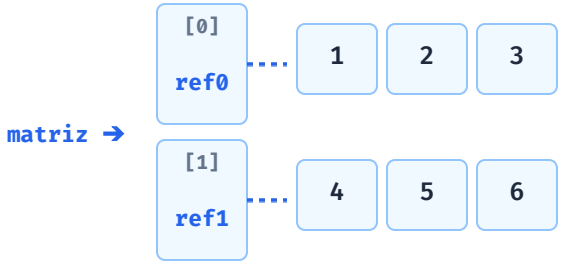
- **for externo (i):** Controla a **linha** atual selecionada (varia de `0` até `matriz.length - 1`).
- **for interno (j):** Itera pelas **colunas** da linha `i` (varia de `0` até `matriz[i].length - 1`).

Propriedade length em Matrizes

- `matriz.length` ➔ Retorna o **número total de linhas** (tamanho do array principal).
- `matriz[i].length` ➔ Retorna a **quantidade de elementos (colunas)** da linha `i`.

Visualização de Memória & Referências

Estrutura: Array de Arrays



Conceito

int[][] é um array cujos elementos são outros arrays. Cada linha pode inclusive ter tamanhos diferentes em Java (matrizes irregulares/jagged).

- `matriz.length` = 2 (linhas)
- `matriz[0].length` = 3 (colunas da linha 0)

Visualizando Matrizes com Arrays.deepToString()

Declaração da Matriz (Retomada)

```
// Matriz de 2 linhas e 3 colunas
int[][] matriz = { {1, 2, 3}, {4, 5, 6} };
```

Solução: Imprimindo a Matriz Completa

```
// Converte a matriz inteira em String formatada
System.out.println( Arrays.deepToString(matriz) );
```

Saída no Console

```
[[1, 2, 3], [4, 5, 6]]
```

Formatador Automático: O método recorre por todas as dimensões e gera uma representação legível e delimitada por colchetes.

Comparação de Métodos

Diferenças de Aplicação

`Arrays.toString(array);`
Adequado para arrays **unidimensionais**.

`Arrays.deepToString(matriz);`
Adequado para arrays **multidimensionais**.

Por que não usar Arrays.toString()?

- Se usado em uma matriz, `Arrays.toString(matriz)` apenas imprime os **endereços de memória / referências** dos sub-arrays internos (ex: `[I@15db9742`).
- Para inspecionar o **conteúdo real** de matrizes aninhadas, utilize sempre **`Arrays.deepToString()`**.

Exercício 4: Matriz de Vendas

Declaração da Matriz de Vendas

```
double[][] vendas = {
  {1200.0, 1500.0, 1800.0, 1400.0}, // Loja 1
  {1000.0, 1700.0, 1600.0, 1900.0}, // Loja 2
  {2000.0, 2100.0, 1950.0, 2200.0} // Loja 3
};
```

Mapeamento e Visualização dos Dados (3x4)

Empresa	Semana 1	Semana 2	Semana 3	Semana 4
Loja 1 [0]	1200.0	1500.0	1800.0	1400.0
Loja 2 [1]	1000.0	1700.0	1600.0	1900.0
Loja 3 [2]	2000.0	2100.0	1950.0	2200.0

- Linhas representam as Lojas (índices 0 a 2)
- Colunas representam as Semanas (índices 0 a 3)

Objetivos do Programa

- Exibir a matriz em formato tabulado.
- Calcular total vendido por loja e por semana.
- Calcular o total geral e identificar a loja com maior venda.

Requisitos de Implementação

Requisitos Técnicos

- Utilizar tipo **double[][]**.
- Processar via **laços aninhados** (**for**).
- Usar **vendas.length** (número de lojas).
- Usar **vendas[i].length** (semanas por loja).
- **Proibido calcular totais manualmente.**

Exemplo de Saída Esperada no Console

```
═══ VENDAS ═══
S1 S2 S3 S4
Loja 1 1200 1500 1800 1400
Loja 2 1000 1700 1600 1900
Loja 3 2000 2100 1950 2200

Total Loja 1: R$ 5900.0
Total Loja 2: R$ 6200.0
Total Loja 3: R$ 8250.0

Total Semana 1: R$ 4200.0
Total Semana 2: R$ 5300.0
Total Semana 3: R$ 5350.0
Total Semana 4: R$ 5500.0

Total Geral: R$ 20350.0
Maior Venda: Loja 3 (R$ 8250.0)
```

Encontre os erros no Código Array

Análise de Erros Comuns

Código A — Acesso Fora dos Limites

```
int[] x = new int[5];
System.out.println(x[5]);
```

Encontre os erros no Código Array

Código A — Acesso Fora dos Limites

```
int[] x = new int[5];
System.out.println(x[5]);
```

Erro: ArrayIndexOutOfBoundsException. Índices válidos vão de **0 a 4** (tamanho 5).

Análise de Erros Comuns

Regra de Índices (0 a N-1)

Em Java, arrays são indexados de **0** a **length - 1**. Tentar acessar a posição **length** causa estouro de limite.

Encontre os erros no Código Array

Código A — Acesso Fora dos Limites

```
int[] x = new int[5];
System.out.println(x[5]);
```

Erro: ArrayIndexOutOfBoundsException. Índices válidos vão de **0 a 4** (tamanho 5).

Código B — Array de Objetos Não Inicializado

```
Pokemon[] pokemon = new Pokemon[3];
pokemon[0].getNome();
```

Análise de Erros Comuns

Regra de Índices (0 a N-1)

Em Java, arrays são indexados de **0** a **length - 1**. Tentar acessar a posição **length** causa estouro de limite.

Encontre os erros no Código Array

Código A — Acesso Fora dos Limites

```
int[] x = new int[5];
System.out.println(x[5]);
```

Erro: **ArrayIndexOutOfBoundsException**. Índices válidos vão de **0 a 4** (tamanho 5).

Código B — Array de Objetos Não Inicializado

```
Pokemon[] pokemon = new Pokemon[3];
pokemon[0].getNome();
```

Erro: **NullPointerException**. As posições contêm **null** até que os objetos sejam instanciados.

Análise de Erros Comuns

Regra de Índices (0 a N-1)

Em Java, arrays são indexados de **0** a **length - 1**. Tentar acessar a posição **length** causa estouro de limite.

Arrays de Tipos Referência

Criar o array de objetos **não instancia** os objetos. Cada posição precisa ser inicializada individualmente com **new**.

Encontre os erros no Código Array

Código A — Acesso Fora dos Limites

```
int[] x = new int[5];
System.out.println(x[5]);
```

Erro: **ArrayIndexOutOfBoundsException**. Índices válidos vão de **0 a 4** (tamanho 5).

Código B — Array de Objetos Não Inicializado

```
Pokemon[] pokemon = new Pokemon[3];
pokemon[0].getNome();
```

Erro: **NullPointerException**. As posições contêm **null** até que os objetos sejam instanciados.

Código C — Condição Incorreta no Laço

```
for (int i = 0; i ⩽ notas.length; i++) {
  System.out.println(notas[i]);
}
```

Análise de Erros Comuns

Regra de Índices (0 a N-1)

Em Java, arrays são indexados de **0** a **length - 1**. Tentar acessar a posição **length** causa estouro de limite.

Arrays de Tipos Referência

Criar o array de objetos **não instancia** os objetos. Cada posição precisa ser inicializada individualmente com **new**.

Encontre os erros no Código Array

Código A — Acesso Fora dos Limites

```
int[] x = new int[5];
System.out.println(x[5]);
```

Erro: **ArrayIndexOutOfBoundsException**. Índices válidos vão de **0 a 4** (tamanho 5).

Código B — Array de Objetos Não Inicializado

```
Pokemon[] pokemon = new Pokemon[3];
pokemon[0].getNome();
```

Erro: **NullPointerException**. As posições contêm **null** até que os objetos sejam instanciados.

Código C — Condição Incorreta no Laço

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

Erro: Usar **<=** acessa **notas[notas.length]**. O correto é usar **i < notas.length**.

Análise de Erros Comuns

Regra de Índices (0 a N-1)

Em Java, arrays são indexados de **0** a **length - 1**. Tentar acessar a posição **length** causa estouro de limite.

Arrays de Tipos Referência

Criar o array de objetos **não instancia** os objetos. Cada posição precisa ser inicializada individualmente com **new**.

Laços e Parada

Use sempre operador estritamente menor (**<**) em laços que percorrem arrays do início ao fim:

i < array.length.

Resumo: Arrays & Matrizes

Conceitos Fundamentais de Arrays

- Armazena vários elementos do **mesmo tipo** em posições contíguas.
- Índices começam sempre em `0` e vão até `length - 1`.
- O **tamanho é fixo** e definido no momento da criação.
- A propriedade `length` informa o número total de elementos.
- Estruturas `for` e `for-each` são usadas para percorrer elementos.
- Podem armazenar **tipos primitivos** ou **referências para objetos**.

Matrizes & Classe java.util.Arrays

- A sintaxe `[][]` permite representar estruturas multidimensionais.
- Em Java, matrizes são tratadas como **arrays de arrays**.
- A classe `java.util.Arrays` oferece utilitários auxiliares (ordenar, comparar, buscar).
- O método `Arrays.deepToString()` facilita a exibição de matrizes.

Pontos de Atenção & Boas Práticas

Indexação & Limites

Acessar posições fora do intervalo `0` a `length - 1` gera a exceção `ArrayIndexOutOfBoundsException`.

Arrays de Objetos

A criação do array inicializa as posições com `null`. É necessário instanciar cada objeto individualmente antes de usar.

Inspeção Visual de Matrizes

Use `Arrays.deepToString()` em matrizes multidimensionais. O uso de `toString()` simples exibe apenas referências de memória.

Exercício Prático: Jogo da Velha em Java

Requisitos do Sistema & Interface CLI

- **Modo de Exibição:** Exclusivamente no terminal (sem UI gráfica/web).
- **Estrutura de Dados:** Tabuleiro representado por **matriz 3x3** (`char[3][3]`).
- **Símbolos dos Jogadores:** Jogador 1 → `'X'` | Jogador 2 → `'O'`.
- **Condição de Parada:** Vitória de um jogador ou empate (velha).

Exemplo de Saída no Terminal

```
  0 1 2
0 X | - | O
 ---+---+---
1 - | X | -
 ---+---+---
2 O | - | X
Vez do Jogador 1 (X)
Informe a linha (0-2): 1
Informe a coluna (0-2): 2
Jogador 1 venceu!
```

Fluxo do Game Loop (Por Rodada)

1. **Exibir o Tabuleiro:** Imprimir a matriz 3x3 formatada.
2. **Identificar Vez:** Informar qual jogador joga ('X' ou 'O').
3. **Entrada de Dados:** Solicitar linha e coluna no terminal.
4. **Validação:** Verificar se a posição está dentro dos limites e livre.
5. **Registro:** Salvar o símbolo no elemento `matriz[l][c]`.
6. **Verificação de Vitória:** Checar 3 linhas, 3 colunas e 2 diagonais.
7. **Verificação de Empate:** Validar se todas as 9 posições foram preenchidas.
8. **Alternância:** Trocar o jogador ativo para a próxima rodada.

Dúvidas e Discussão