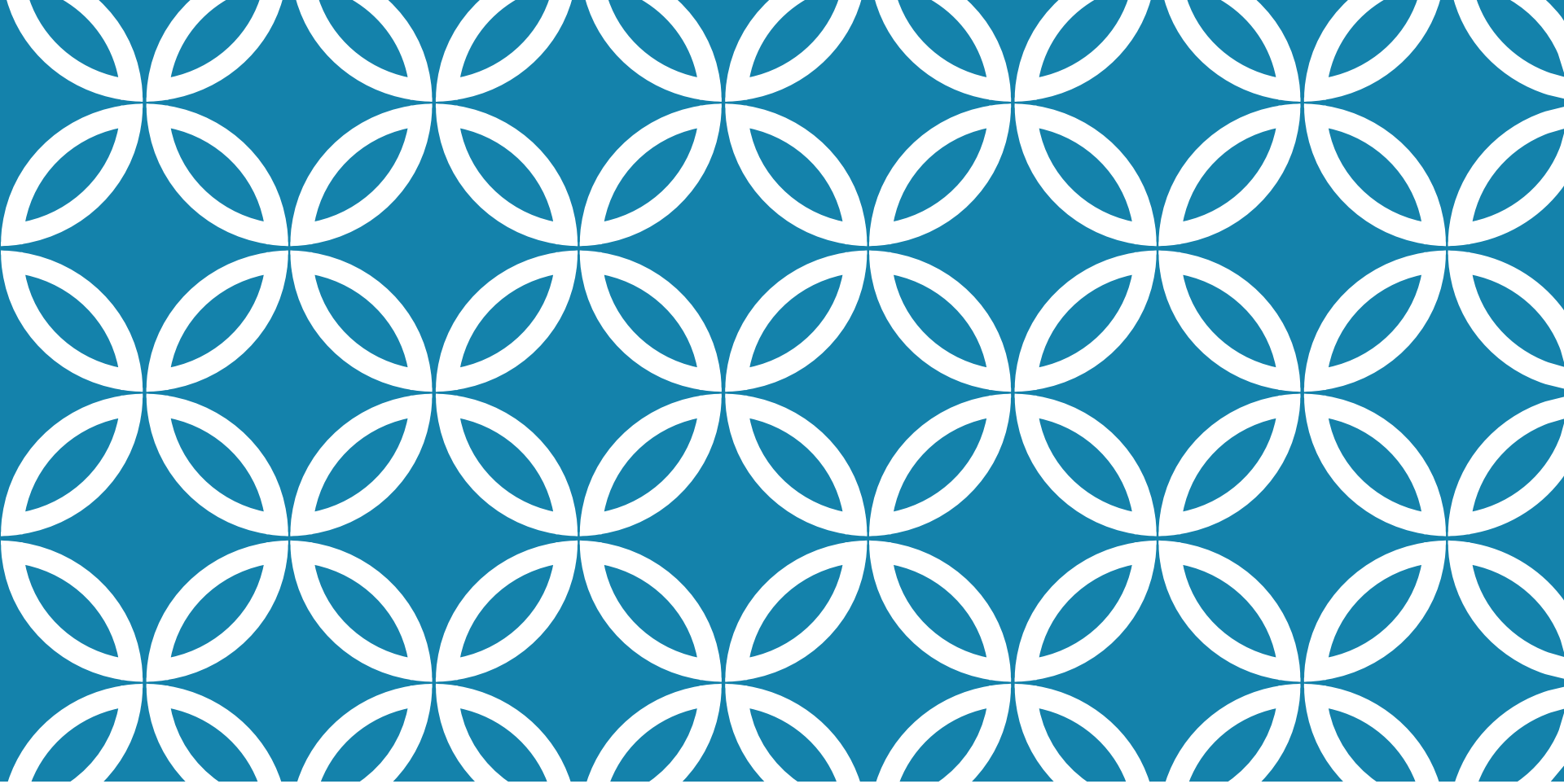


Arrays

Rafael Vieira Coelho

Introdução

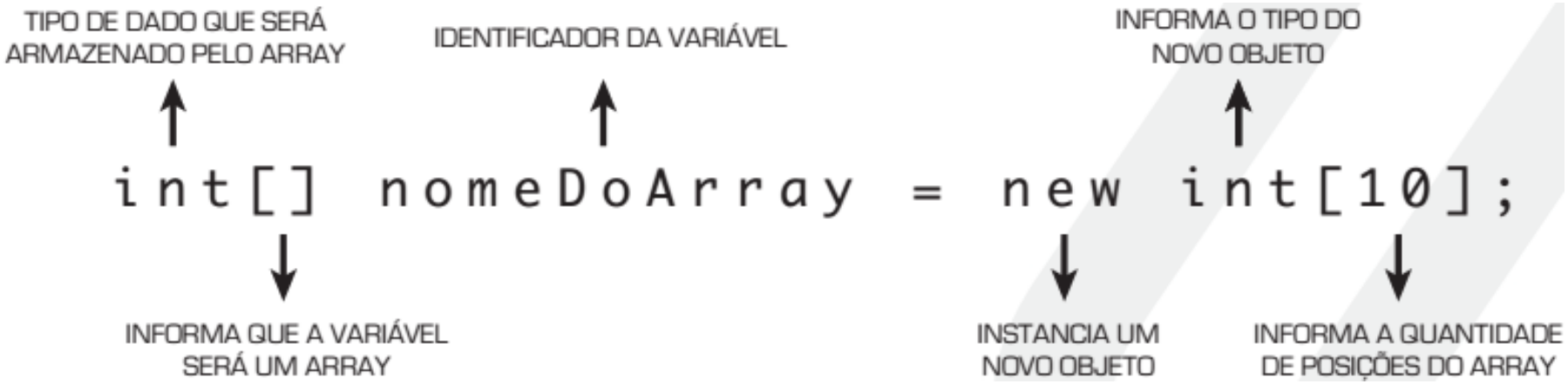
Gerar uma variável para uma lista, como uma lista de números de contas de uma agência não seria prático, implicando em alterações constantes de código-fonte

```
1  int numero1;
2  int numero2;
3  int numero3;
4  ...
```

Quando desejamos armazenar uma grande quantidade de valores de um determinado tipo, podemos usar **arrays** (coleção de dados homogênea)

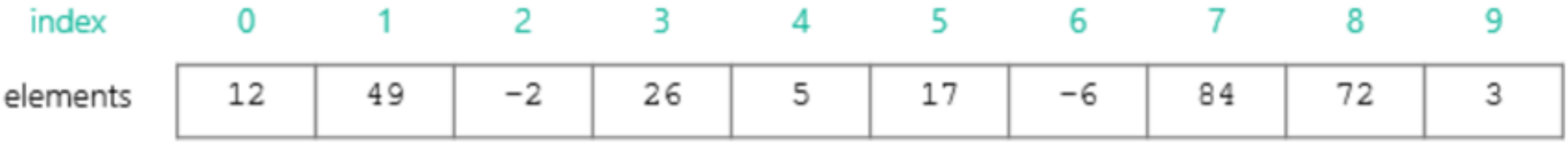
Criando um Array

Em Java, os arrays são criados através do comando **`new`**.



Estrutura de um Array

O array é composto por uma série de elementos de mesmo tipo que podem ser acessados com base em seu índice (posição no array).



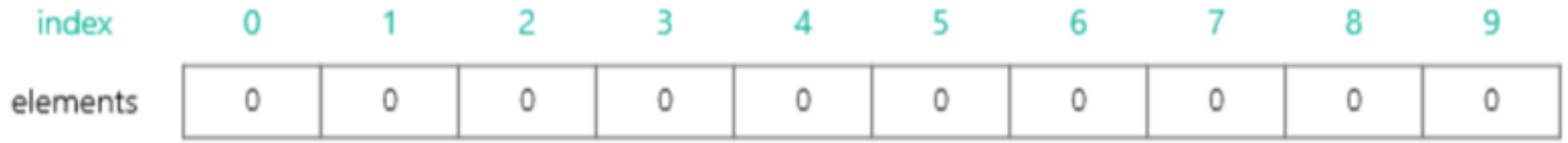
Declarando um array de inteiros:

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

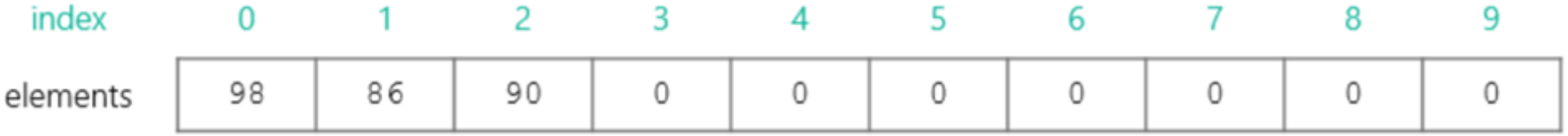
Obs: quando não inicializado, o valor do array é null, podendo gerar a exceção **NullPointerException** ao acessá-lo.

Modificando o Array

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




```
studentGrades[0] = 98;
studentGrades[1] = 86;
studentGrades[2] = 90;
```



Obs: Acessar posições fora do intervalo de índices de um array gera o erro **ArrayIndexOutOfBoundsException**

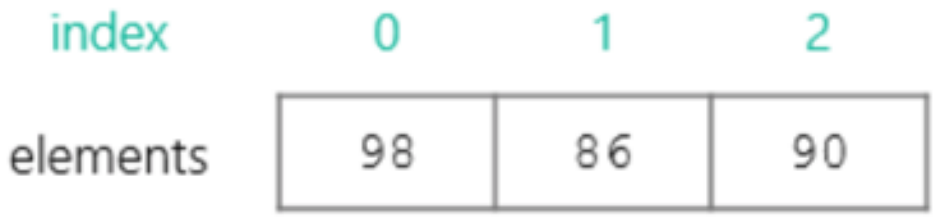
Inicialização de Arrays

Quando sabemos previamente qual o conteúdo dos elementos do array, podemos inicializá-lo:

```
dataType[] name = {dataValue1, dataValue2, dataValue3};
```

No exemplo abaixo, colocamos os valores 98, 86 e 90 no array studentGrades:

```
int[] studentGrades = {98, 86, 90};
```



Campo Length

Todo array tem uma propriedade que informa o número de elementos do mesmo (número inteiro).

```
int[] a = new int[5];
double[] b = new double[10];
char[] c = new char[20];
int aLength = a.length; // returns 5
int bLength = b.length; // returns 10
int cLength = c.length; // returns 20
```

Obs: Lembre-se que a posição inicial de um array é 0 e a posição máxima é length -1.

Percorrendo um Array

Para percorrermos um array, utilizaremos a instrução de repetição **`for`**

Podemos utilizar a instrução **`while`** também

```
1  int[] numeros = new int[100];
2  for(int i = 0; i < 100; i++) {
3    numeros[i] = i;
4  }
```

Percorrendo um Array

Para percorrer um array, é necessário saber a quantidade de posições do mesmo (como faremos isso?)

Podemos recuperar a quantidade de posições de um array acessando o seu atributo **`length`**

```
1  void imprimeArray(int[] numeros) {
2    for(int i = 0; i < numeros.length; i++) {
3      System.out.println(numeros[i]);
4    }
5  }
```

foreach

Para acessar todos os elementos de um array, é possível aplicar o comando **`for`** com uma sintaxe um pouco diferente, conhecido como **`foreach`**

```
1  void imprimeArray(int[] numeros) {
2    for(int numero : numeros) {
3      System.out.println(numero);
4    }
5  }
```

Arrays e Métodos

Quando vamos retornar ou passar por parâmetro um array, precisamos explicitamente colocar [] em sua declaração.

Exemplos:

```
public static int[] myMethod(int[] a) {}

public static double[] myMethod(double[] a) {}
```

Passando Array como Parâmetro

```
public static void printArray(int[] array) {
  for (int i = 0; i < array.length; i++) {
    System.out.print(array[i] + " ");
  }
}
```

Retornando um vetor como parâmetro

```
public static int[] inverso(int[] list) {
  int[] result = new int[list.length];

  for (int i = 0, j = result.length - 1; i < list.length; i++, j--) {
            result[j] = list[i];
  }
  return result;
}
```

Referência de Memória

Quando atribuímos um array **a** para outro array **b**, estamos passando a referência de memória onde o array a está sendo armazenado.

Se fizermos alguma alteração no array **b**, estaremos alterando o array **a**.

```
int a[] = new int[5]; // [0, 0, 0, 0, 0]
a[0] = 10;            // [10, 0, 0, 0, 0]
int b[] = a;          // [10, 0, 0, 0, 0]
b[0] = 5;             // [5, 0, 0, 0, 0]
```

```
a = [5, 0, 0, 0, 0]
b = [5, 0, 0, 0, 0]
```

Referência de Memória

O mesmo ocorre quando passamos um array por parâmetro e o modificamos dentro de um método.

```
public static void main(String[] args) {
    int[] a = new int[5];
    System.out.println("before method: " + Arrays.toString(a));
    myMethod(a);
    System.out.println("after method: " + Arrays.toString(a));
}

public static void myMethod(int[] b) {
    b[0] = 5;
}
```

```
before method: [0, 0, 0, 0, 0]
after method: [5, 0, 0, 0, 0]
```

Operações com Arrays

Nas bibliotecas da plataforma Java, existem métodos que realizam algumas tarefas úteis relacionadas a arrays, como por exemplo

1. Ordenação
2. Duplicação
3. Preenchimento

Operações com Arrays

1) Ordenando um Array

Considere um array de **`String`** criado para armazenar nomes de pessoas

Podemos ordenar esses nomes através do método **`Arrays.sort()`**

```
1  String[] nomes = new String[]{"rafael cosentino", "jonas hirata", "marcelo martins"};
2  Arrays.sort(nomes);
3
4  for(String nome : nomes) {
5    System.out.println(nome);
6  }
```

Operações com Arrays

2) Duplicando um Array

Para copiar o conteúdo de um array para outro com maior capacidade, podemos utilizar o método **`Arrays.copyOf()`**

```
1  String[] nomes = new String[] {"rafael", "jonas", "marcelo"};
2  String[] nomesDuplicados = Arrays.copyOf(nomes, 10);
```

Operações com Arrays

3) Preenchendo um Array

Podemos preencher todas as posições de um array com um valor específico utilizando o método **`Arrays.fill()`**

```
1  int[] numeros = new int[10];
2  java.util.Arrays.fill(numeros, 5);
```

Exercícios (USE MÉTODOS!)

Crie programas principais que façam as seguintes ações:

1) Calcular a soma dos valores inteiros contidos em um vetor qualquer.

2) Localizar, em um vetor qualquer de valores inteiros, um certo elemento deste, dado o seu valor.

3) Verificar se um vetor de valores do tipo char é um palíndromo. Um palíndromo é uma frase ou palavra que se pode ler, indiferentemente, da esquerda para a direita ou vice-versa e o resultado é o mesmo. Ex: “Radar".

4) Determinar a quantidade de vogais e de consoantes em um vetor de valores do tipo char.