Características das Linguagens de Programação I

Conteúdo: Strings, arrays e funções

Prof. Dsc. Giomar Sequeiros
giomar@eng.uerj.br

Strings

Strings

- As strings em Java são objetos imutáveis da classe String que representam uma sequência de **caracteres**.

- As Strings embora sejam tratadas de forma simples, como se fossem tipos primitivos, oferecem diversas funcionalidades por meio de métodos que permitem a manipulação eficiente de texto.
- Declaração:

```
String nome = "Uma String";  // Maneira direta
String outraString = new String("Outra String");  // Usando o construtor
```

Strings: Funções (1)

- **length():** Retorna o tamanho da string (número de caracteres).

```
String texto = "Java";
int tamanho = texto.length();  // Retorna 4
```

- **charAt(int index):** Retorna o caractere na posição especificada (índice começa em 0).

```
char caractere = texto.charAt(1);  // Retorna 'a'
```

- **substring(int beginIndex):** Retorna uma parte da string, começando no índice especificado até o final.

```
String parte = texto.substring(2);  // Retorna "va"
```

- **substring(int beginIndex, int endIndex):** Retorna uma parte da string, entre o índice inicial (inclusive) e o índice final (exclusivo).

```
String parte = texto.substring(1, 3);  // Retorna "av"
```

Strings: Funções (2)

- **equals(String anotherString):** Compara duas strings, verificando se são exatamente iguais (diferencia maiúsculas e minúsculas).

```
String a = "Java";
String b = "java";
boolean igual = a.equals(b);  // Retorna false
```

- **equalsIgnoreCase(String anotherString):** Compara duas strings, ignorando diferenças entre maiúsculas e minúscula**s.**

```
boolean igual = a.equalsIgnoreCase(b);  // Retorna true
```

- **toLowerCase() e toUpperCase():** Converte toda a string para minúsculas ou maiúsculas.

```
String minuscula = texto.toLowerCase();  // Retorna "java"
String maiuscula = texto.toUpperCase();  // Retorna "JAVA"
```

Strings: Funções (3)

- **trim():** Remove espaços em branco no início e no fim da string.

```
String espacos = "   Olá, Java!   ";
String semEspacos = espacos.trim();  // Retorna "Olá, Java!"
```

- **replace(char oldChar, char newChar):** Substitui todas as ocorrências de um caractere por outro.

```
String novaString = texto.replace('a', 'o');  // Retorna "Jovo"
```

- **split(String regex):** Divide a string com base no padrão especificado (regex) e retorna um array de Strings.

```
String frase = "Java é fácil";
String[] palavras = frase.split(" ");  // Retorna ["Java", "é", "fácil"]
```

Strings: Funções (4)

- **indexOf(String str):** Retorna a posição da primeira ocorrência da sequência de caracteres especificada.

```
int indice = texto.indexOf("a");  // Retorna 1
```

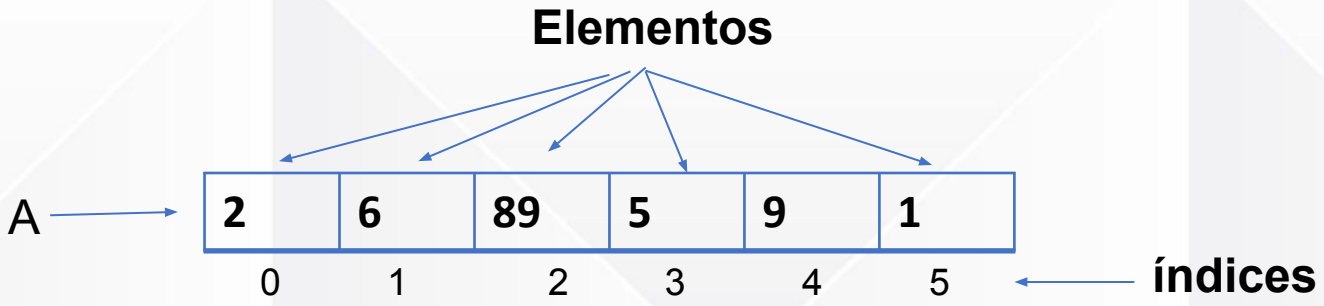
- **compareTo(String anotherString):** Compara duas strings lexicograficamente (ordem alfabética). Retorna um valor negativo, zero ou positivo.

```
String x = "abc";
String y = "abd";
int comparacao = x.compareTo(y); // Retorna negativo, pois "abc" é menor que "abd"
```

Arrays

Arrays

- Arrays são estruturas que armazenam uma coleção de elementos do mesmo tipo.
- Arrays têm tamanho fixo, ou seja, após sua criação, o número de elementos não pode ser alterado.
- O índice de um array começa em 0, e os elementos são acessados através desse índice.



Arrays: Declaração

- Declaração e inicialização separadas:

```
int[] numeros;  // Declaração
numeros = new int[5];  // Inicialização (um array de 5 inteiros)
```

- Declaração e inicialização em uma linha:

```
int[] numeros = new int[5];  // Inicialização (um array de 5 inteiros)
```

Arrays: Inicialização

- Atribuição de valores

```
numeros[0] = 10;
numeros[1] = 20;
numeros[2] = 30;
numeros[3] = 40;
numeros[4] = 50;
```

- Inicialização direta

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

Arrays: Acesso a elementos

- Os elementos de um array são acessados pelo seu índice (começando em 0). Exemplo:

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

```
System.out.println(numeros[0]);  // Imprime o primeiro elemento, que é 10
System.out.println(numeros[1]);  // Imprime o segundo elemento, que é 20
```

Arrays: Acesso a elementos

- Percorrendo os elementos usando o comando for

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

- Percorrendo os elementos usando o comando for-each

```
for (int numero : numeros) {
    System.out.println(numero);
}
```

Arrays: Exemplo

- Criar de um array lendo os elementos a partir da entrada do teclado:

```
final int TAMANHO = 5;  // Define o tamanho do array
int[] numeros = new int[TAMANHO];
Scanner scanner = new Scanner(System.in);

System.out.println("Digite " + TAMANHO + " números:");
```

```
// Entrada dos números
for (int i = 0; i < TAMANHO; i++) {
    System.out.print("Número " + (i + 1) + ": ");
    numeros[i] = scanner.nextInt();  // Lê cada número do array
}
// Exibe os números armazenados no array
System.out.println("Elementos do array:");
for (int i = 0; i < TAMANHO; i++) {
    System.out.println(numeros[i]);
}
```

```
scanner.close();
```

Arrays: Exemplo

Saída:

```
Digite 5 números:
Número 1: 10
Número 2: 20
Número 3: 30
Número 4: 40
Número 5: 50
Elementos do array:
10
20
30
40
50
```

Arrays multidimensionais: matrizes

- Um array 2D é uma matriz, onde podemos armazenar dados em linhas e colunas. Declaração:

```
int[][] matriz = new int[3][3];  // Declara uma matriz 3x3
```

- Inicialização atribuindo valores:

```
matriz[0][0] = 1;
matriz[0][1] = 2;
matriz[0][2] = 3;
matriz[1][0] = 4;
matriz[1][1] = 5;
matriz[1][2] = 6;
matriz[2][0] = 7;
matriz[2][1] = 8;
matriz[2][2] = 9;
```

Arrays multidimensionais: matrizes

- Inicialização direta:

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

Percorrendo uma matriz

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

Funções

Funções

- São blocos de código que realizam uma **tarefa específica**. Elas podem ser chamadas em qualquer lugar do programa para executar sua tarefa e podem receber argumentos e retornar um valor. Sintaxe básica:

```
retorno nomeDaFuncao(tipoParametro1 nomeParametro1, tipoParametro2 nomeParametro2, ...)
{
    // Corpo da função
    return valorRetornado;  // Opcional, dependendo do tipo de retorno
}
```

- Onde:
  - **retorno:** Define o tipo de dado que a função vai retornar. Se a função não retorna nada, usamos void.
  - **nomeDaFuncao:** Nome da função, seguindo as convenções de nomenclatura (camelCase).
  - **tipoParametro:** Tipo do(s) parâmetro(s) que a função aceita, seguido do nome do parâmetro.
  - **return:** Retorna o valor do tipo especificado (se não for void).

Funções: exemplo 1

- Exemplo básico de função sem retorno (void)

```
public class Exemplo {
    public static void saudacao() {
        System.out.println("Olá, bem-vindo!");
    }

    public static void main(String[] args) {
        saudacao();
    }
}
```

Métodos static pertencem à classe, não às instâncias (objetos).

// Chamando a função

Funções: exemplo 2

- Exemplo de função com parâmetros

```
public class Exemplo {
    public static void exibirMensagem(String mensagem) {
        System.out.println(mensagem);
    }

    public static void main(String[] args) {
        exibirMensagem("Olá, Mundo!");
    }
}
```

```
// Chamando a função com um argumento
```

Funções: exemplo 3

- Exemplo de função com parâmetros

```
public class Exemplo {
    public static int somar(int a, int b) {
        return a + b;
    }

    public static void main(String[] args) {
        int resultado = somar(10, 20);
        System.out.println("A soma é: " + resultado);
    }
}
```

Funções: exemplo 4

- Função que retorna o fatorial de um número

```
public class Exemplo {

    public static int fatorial(int n) {
        int resultado = 1;
        for (int i = 1; i <= n; i++) {
            resultado *= i;
        }
        return resultado;
    }

    public static void main(String[] args) {
        int numero = 5; // usar leitura por teclado
        int resultado = fatorial(numero);
        System.out.println("O fatorial de " + numero + " é: " + resultado);
    }
}
```

Exercício 1

- Escreva uma função que receba um número inteiro positivo e retorne a decomposição desse número em fatores primos.

Por exemplo, para o número 24, a função deve retornar os fatores primos 2, 2, 2, 3.

Dica: Utilize a técnica de divisões sucessivas, exemplos:

72	2
36	2
18	2
9	3
3	3
1	$2^3 . 3^2$

196	2
98	2
49	7
7	7
1	$2^2 . 7^2$

324	2
162	2
81	3
27	3
9	3
3	3
1	$2^2 . 3^4$

7623	3
2541	3
847	7
121	11
11	11
1	$3^2 . 7 . 11^2$

Exercício 1: solução

Primeiramente criamos uma função para verificar se um número é primo ou não:

```
public static boolean ehPrimo(int numero) {
    if (numero <= 1) {
        return false;
    }
    for (int i = 2; i <= Math.sqrt(numero); i++) {
        if (numero % i == 0) {
            return false;
        }
    }
    return true;
}
```

Exercício 1: solução

Criamos a função para decompor um número em seus fatores primos:

```
public static void decomporEmFatores(int numero) {
    System.out.print("Fatores primos de " + numero + ": ");

    for (int fator = 2; fator <= numero; fator++) {
        if (ehPrimo(fator)) {
            while (numero % fator == 0) {
                System.out.print(fator + " ");
                numero /= fator;
            }
        }
    }
    System.out.println();
}
```

```
public static void main(String[] args) {
    int numero = 60; // Substitua por leitura por teclado
    decomporEmFatores(numero);
}
```

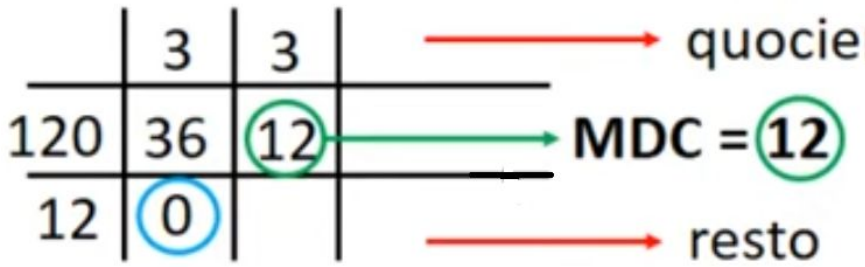
Exercício 2

- Escreva uma **função** que receba **dois** números inteiros e calcule o máximo divisor comum (**MDC**) entre eles. Retorne o resultado.

- Exemplo usando o **algoritmo de Euclides** (divisões sucessivas)
  - MDC de **120** e **36**

```
 120 | 36
-108   3
  12
```

```
 36 | 12
(0)  3
```



Exercício 2

- Outro exemplo usando o algoritmo de Euclides
  - MDC de **528** e **240**

```
 528 | 240
-480   2
 48
```

	2	5	
528	240	48	MDC = 48
48	0		

quociente

resto

```
240 | 48
(0)   5
```

Exercício 2: Solução

```
public static int calcularMDC(int num1, int num2) {
    int resto;

    // Garantir que num1 seja maior que num2
    if (num1 < num2) {
        int temp = num1;
        num1 = num2;
        num2 = temp;
    }
    // Algoritmo de Euclides para calcular o MDC
    while (num2 != 0) {
        resto = num1 % num2;
        num1 = num2;
        num2 = resto;
    }
    return num1;
}
```

Exercícios

- Crie uma função que receba um array e retorne true se os elementos estão na ordem crescente e retorne false caso contrário.

- Crie uma função que receba um array e retorne um array com os elementos na ordem crescente

Funções recursivas

Recursividade:

- Em matemática, pode ser definida como: o ato de definir um objeto (geralmente uma função), em termos do próprio objeto.
- Em computação, ocorre quando: um dos passos de um determinado algoritmo envolve a repetição desse mesmo algoritmo
- Um procedimento que se utiliza da recursão é dito recursivo.
- Também é dito recursivo qualquer objeto que seja resultado de um procedimento recursivo.

Funções recursivas: Exemplo fatorial

Recursividade:

- **Caso base**: Parte não recursiva, também chamada de âncora, ocorre quando a resposta para o problema é trivial.
- **Chamada recursiva**: Parte da definição que especifica como cada elemento (solução) é gerado a partir do precedente.

Ex.: A função **fatorial n!** pode ser definida como, dado um número inteiro positivo n:

$$n! = \begin{cases} 1; & \text{se } n = 0 \text{ (caso base);} \\ n*(n - 1)!; & \text{se } n > 0 \text{ (chamada recursiva):} \end{cases}$$

Funções recursivas: Exemplo fatorial

- **Implementação:**

$$n! = \begin{cases} 1; & \text{se } n = 0 \text{ (caso base);} \\ n*(n - 1)!; & \text{se } n > 0 \text{ (chamada recursiva):} \end{cases}$$

```java
public static int fatorial(int n) {
    if (n == 0) {
        return 1;  // Caso base: fatorial de 0 é 1
    } else {
        return n * fatorial(n - 1);  // Passo recursivo
    }
}
```

```java
public static void main(String[] args) {
    int numero = 5;
    int resultado = fatorial(numero);
    System.out.println("O fatorial de " + numero + " é: " + resultado);
}
```

Funções recursivas: Exemplo fatorial

```java
public static int fatorial(int n) {
    if (n == 0) {
        return 1;  // Caso base: fatorial de 0 é 1
    } else {
        return n * fatorial(n - 1);  // Passo recursivo
    }
}
```

- **Árvore de execução** do exemplo **fatorial(4)**:

fatorial(4)
└── 4 * fatorial(3)
    └── 3 * fatorial(2)
        └── 2 * fatorial(1)
            └── 1 * fatorial(0)
                └── 1

Funções recursivas: Exemplo soma dígitos

Definição:

$$somaDigitos(n) = \begin{cases} 0; & \text{se } n = 0 \text{ (caso base);} \\ (n \,\%\, 10) + somaDigitos(n/10); & \text{se } n > 0 \text{ (chamada recursiva):} \end{cases}$$

Funções recursivas: Exemplo soma dígitos

Implementação:

somaDigitos(n) =
- 0; se n = 0 (caso base);
- (n % 10) + somaDigitos(n/10); se n > 0 (chamada recursiva):

```
public static int somaDigitos(int n) {
    if (n == 0) {
        return 0;
    } else {
        return (n % 10) + somaDigitos(n / 10);
    }
}
```

Funções recursivas: Exemplo soma dígitos

```
public static int somaDigitos(int n) {
    if (n == 0) {
        return 0;
    } else {
        return (n % 10) + somaDigitos(n / 10);
    }
}
```

- **Árvore de execução** do exemplo **somaDigitos(245):**

somaDigitos(245)
└── (5) + somaDigitos(24)
    └── 5 + (4) + somaDigitos(2)
        └── 5 + 4 + (2) + somaDigitos(0)
            └── 5 + 4 + 2 + 0

Exercícios

- Crie uma função recursiva que receba um número inteiro positivo n e retorne a soma dos n primeiros números inteiros.
  - Crie a definição recursiva
  - Mostre a execução para n=5

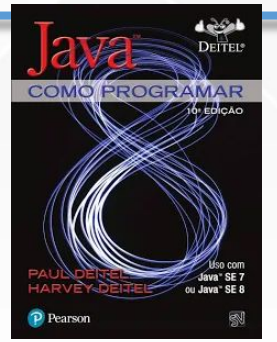
- Escrever um programa recursivo para calcular o produto de dois números naturais “a” e “b”.
  - Crie a definição recursiva
  - Mostre a execução para a=5 e b=4

- Escrever um programa recursivo para calcular a potência de um número: a^n.
  - Crie a definição recursiva
  - Mostre a execução para a=3, n=5

Referências

BIBLIOGRAFIA BÁSICA:

- DEITEL, Harvery M.. Java : como programar. 10ª ed. São Paulo: Pearson - Prentice Hall, 2017.



- BORATTI, Isaías Camilo. Programação Orientada a Objetos em Java : Conceitos Fundamentais de Programação Orientada a Objetos. 1ª ed. Florianópolis: VisualBooks, 2007.

- SIERRA, Kathy; BATES, Bert. Use a Cabeça! Java. 2ª ed. Rio de Janeiro: Alta Books, 2007.