



Quem se prepara, não para.



Linguagens de Programação

3º período

Prof. Dr. João Paulo Aramuni

Sumário

- Java
 - Recursão

Sumário

- Java
 - **Recursão**

- Recursão
 - Em Java, um método pode chamar a si mesmo. Esse processo se chama recursão e dizemos que um método é recursivo quando chama a si próprio.
 - Em geral, recursão é o processo em que algo é definido a partir de si mesmo e é um pouco parecido com uma definição circular. O componente-chave do método recursivo é a instrução que executa uma chamada a ele próprio.
 - A recursão é um mecanismo de controle poderoso.

- Recursão
 - Vantagens da programação recursiva:
 - **Legibilidade:** Em certos casos, o código recursivo pode ser mais fácil de entender e ler, especialmente quando a lógica do problema é naturalmente recursiva.
 - **Abstração:** A recursão pode ajudar a simplificar a solução de problemas complexos, permitindo que você se concentre na definição do problema e divida-o em problemas menores.
 - **Redução de código:** Em muitos casos, a solução recursiva pode levar a menos linhas de código em comparação com soluções iterativas.

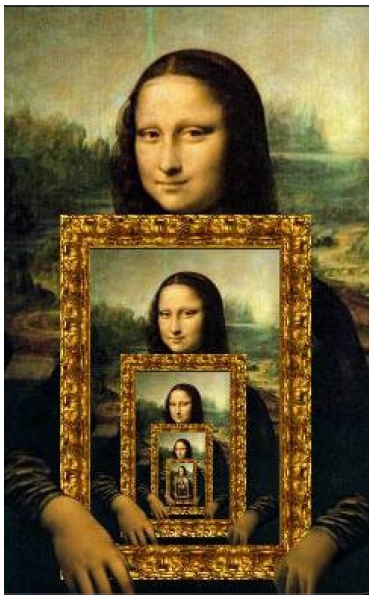
- Recursão
 - Vantagens da programação recursiva:
 - **Solução elegante:** Em algumas situações, a recursão pode levar a uma solução mais elegante, que é mais fácil de implementar e manter.
 - **Algoritmos dividir e conquistar:** A recursão é frequentemente usada para implementar algoritmos de "dividir e conquistar", que são eficientes para resolver problemas como ordenação, busca binária e outras tarefas complexas.

- Recursão
 - Desvantagens da programação recursiva:
 - **Consumo de memória:** Em algumas situações, a recursão pode levar a um alto consumo de memória, especialmente quando há muitas chamadas recursivas aninhadas. Isso pode resultar em estouro da pilha de chamadas (stack overflow).
 - **Desempenho:** A abordagem recursiva pode ser menos eficiente em termos de desempenho em comparação com uma solução iterativa, devido ao alto número de chamadas de função.

- Recursão
 - Desvantagens da programação recursiva:
 - **Dificuldade de depuração:** O processo de depuração pode ser mais complicado com funções recursivas, pois as chamadas podem ser aninhadas e mais difíceis de rastrear.
 - **Limite de recursão:** Alguns ambientes de programação têm limites para a profundidade máxima de recursão, o que pode limitar a capacidade de resolver problemas muito grandes.
 - **Complexidade da implementação:** Em certos casos, a recursão pode exigir uma lógica mais complexa para garantir que o algoritmo termine corretamente.

Java

- Recursão
 - Exemplos em C



```
c / Aramuni.c
Code Blame 204 lines (175 loc) · 4.71 KB

8  int palindrome(char str[100], int tam) {
9      return (tam <= 1) || (*str == str[tam - 1] && palindrome(str + 1, tam - 2));
10 }
11
12 int strlen(char *orig, int cont) {
13     return (*orig++) ? cont++, strlen(orig, cont) : cont;
14 }
15
16 void strcpy(char *str1, char *str2) {
17     return (*str1) ? *str2 = *str1, str1++, str2++, strcpy(str1, str2) : 0;
18 }
19
20 void strcat(char *str1, char *str2, int p) {
21     return (*str2) ? str1[p] = *str2, p++, str2++, strcat(str1, str2, p) : 0;
22 }
23
24 void strInverte(char *str1, char *str2, int p, int i, int j) {
25     return (p != 0) ? (str1++, p--, strInverte(str1, str2, p, i, j)): (i++, (j++ != i) ? (i--, *str2 = *str1, str1--, str2++, strInverte(str1, str2, p, i, j)): 0);
26 }
27
28 void strLimpa(char *str1) {
29     return (*str1) ? *str1 = ' ', str1++, strLimpa(str1) : 0;
30 }
31
32 int contLetra(char palavra[100], int cont, char letra, int i) {
33     return (palavra[i] == letra) ? contLetra(palavra, ++cont, letra, ++i): (palavra[i] ? contLetra(palavra, cont, letra, ++i) : cont);
34 }
35
36 int fatorial(int c) {
37     return (c == 0 || c == 1) ? 1 : (c * fatorial(c - 1));
38 }
39
40 int fibonacci(int n) {
41     return (n == 0) ? 0: ((n == 1) ? 1 : (fibonacci(n - 1) + fibonacci(n - 2)));
42 }
43
44 int mdc(int a, int b) {
45     return (a == 0) ? b : ((b == 0) ? a : mdc(b, a % b));
46 }
47
48 int powExpo(int base, int expo) {
49     return (expo == 0) ? 1 : (expo == 1 ? base : (base * powExpo(base, expo - 1)));
50 }
```

Java

- Recursão
 - Exemplo em Java

FATORIAL

$0! = 1$

$1! = 1$

$2! = 2 \times 1 = 2$

$3! = 3 \times 2 \times 1 = 6$

$4! = 4 \times 3 \times 2 \times 1 = 24$

$5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$

$6! = 6 \times 5 \times 4 \times 3 \times 2 \times 1 = 720$

$7! = 7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1 = 5040$

$8! = 8 \times 7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1 = 40320$

$9! = 9 \times 8 \times 7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1 = 362880$

$10! = 10 \times 9 \times 8 \times 7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1 = 3628800$

```
ExemploRecursao.java X
1 package meuapp;
2
3 import java.util.Scanner;
4
5 public class ExemploRecursao {
6
7     // Um exemplo simples de recursão.
8     // Esta é uma função recursiva.
9     private static int factR(int n) {
10         int result;
11         if (n == 1) {
12             return 1;
13         }
14         // Executa a chamada recursiva a factR( ).
15         result = factR(n - 1) * n;
16         return result;
17     }
18
19     public static void main(String[] args) {
20         Scanner ler = new Scanner(System.in);
21         System.out.println("Digite um numero para calcular o fatorial: ");
22         int num = ler.nextInt();
23         ler.close();
24         System.out.println(factR(num));
25     }
26 }
27
```

Console X Problems @ Javadoc Declaration Search Breakpoints

<terminated> ExemploRecursao [Java Application] /Library/Java/JavaVirtualMachines/jdk-17.jdk/Contents/Home

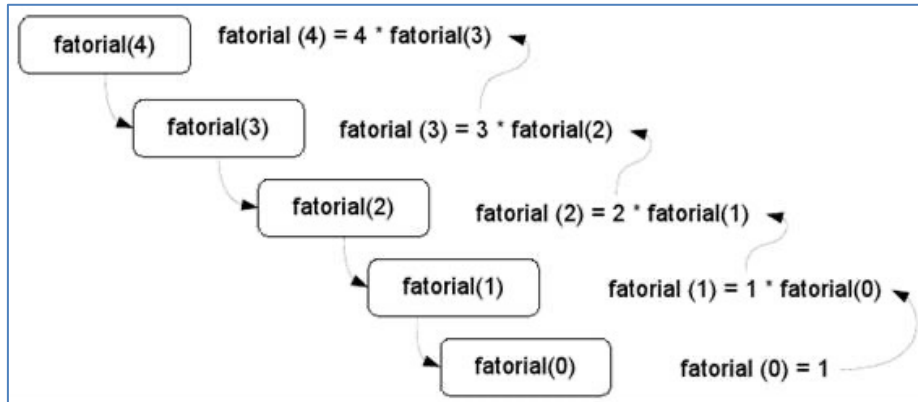
Digite um numero para calcular o fatorial:

5

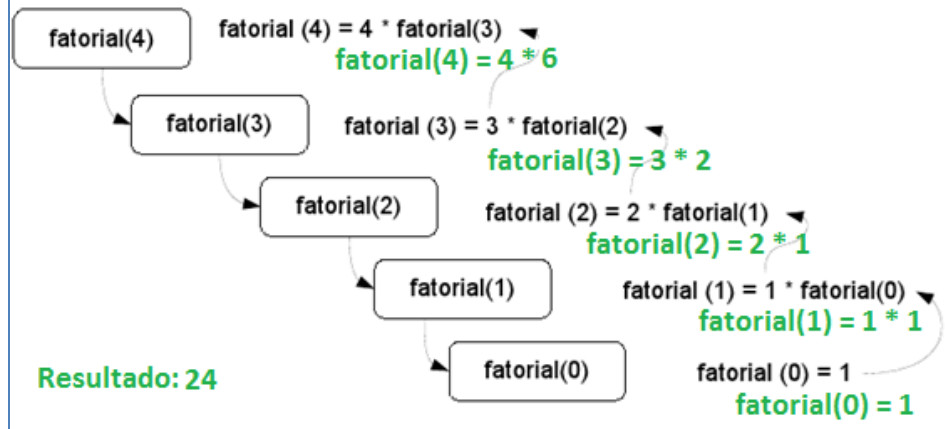
120

- Recursão
 - A operação do método recursivo `factR()` é mais complexa em relação à um método comum.
 - Quando `factR()` é chamado com um argumento igual a 1, ele retorna 1; caso contrário, retorna o produto de `factR(n - 1) * n`.
 - Para avaliar essa expressão, `factR()` é chamado com `n-1`. Esse processo se repete até `n` ser igual a 1 (caso base) e as chamadas ao método começarem a retornar.

Recursão



***Lembrem-se que a leitura é de baixo para cima!**



- Recursão
 - Quando um método chama a si próprio, é alocado espaço de armazenamento na pilha para novas variáveis e parâmetros locais e o código do método é executado com essas novas variáveis desde o início.
 - Uma chamada recursiva **não faz uma nova cópia do método**. Só os argumentos são novos. À medida que cada chamada recursiva retorna, as variáveis e parâmetros locais antigos são removidos da pilha e a execução é retomada no ponto de chamada dentro do método.
 - Poderíamos dizer que os métodos recursivos “**empilham-se**” para frente e para trás.

- Recursão
 - Outros exemplos em Java

```
public static void main(String[] args) {  
    Scanner ler = new Scanner(System.in);  
    System.out.println("Digite um numero para calcular o fatorial: ");  
    int num = ler.nextInt();  
    System.out.println(factR(num));  
  
    System.out.println("Digite dois numeros para o MDC: ");  
    int num1 = ler.nextInt();  
    int num2 = ler.nextInt();  
    System.out.println(mdc(num1, num2));  
  
    System.out.println("Digite um numero para contar os pares: ");  
    int num3 = ler.nextInt();  
    System.out.println(contaPares(num3));  
  
    System.out.println("Digite um numero para calcular a potência: ");  
    int num4 = ler.nextInt();  
    System.out.println("Digite a potência: ");  
    int num5 = ler.nextInt();  
    System.out.println(potencia(num4, num5));  
  
    System.out.println("Digite um numero para calcular a Fibonacci: ");  
    int num6 = ler.nextInt();  
    System.out.println(fibo(num6));  
  
    ler.close();  
}
```

Java

- Recursão
 - Outros exemplos em Java

```
private static int factR(int n) {
    int result;
    if (n == 1) {
        return 1;
    }
    // Executa a chamada recursiva a factR( ).
    result = factR(n - 1) * n;
    return result;
}

private static int mdc(int a, int b) {
    if (b == 0) {
        return a;
    }
    return mdc(b, (a % b));
}

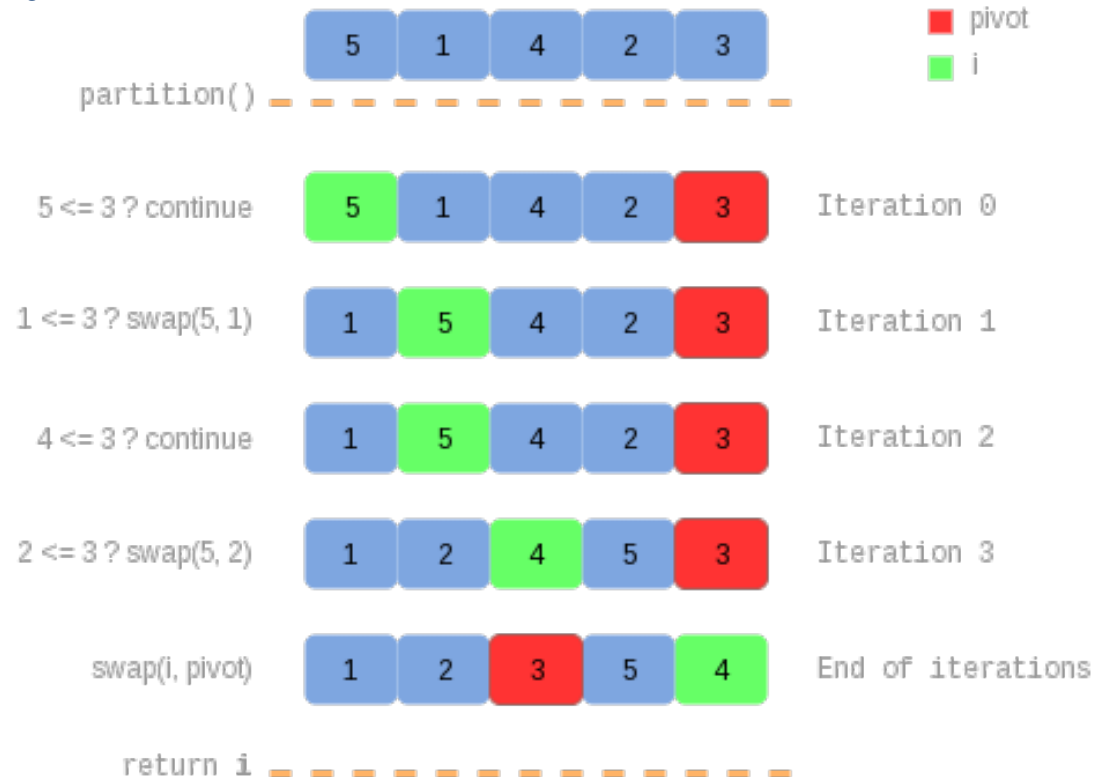
private static int contaPares(int n) {
    if (n == 0 || n == 1) {
        return 1;
    }
    if (n % 2 == 0) {
        return contaPares(n - 1) + 1;
    } else {
        return contaPares(n - 1);
    }
}

public static double potencia(int n, int pot) {
    if (pot == 1) {
        return n;
    }
    return n * potencia(n, pot - 1);
}

public static int fibo(int n) {
    if (n < 2) {
        return n;
    }
    return fibo(n - 1) + fibo(n - 2);
}
```


- Recursão
 - Exemplo QuickSort
 - O algoritmo **Quicksort** utiliza o paradigma de programação *Dividir para Conquistar*. Esse paradigma é uma abordagem **recursiva** em que a entrada do algoritmo é ramificada múltiplas vezes a fim de quebrar o problema maior em problema menores da mesma natureza.
 - No caso do Quicksort o método ***partition()*** é responsável por implementar o paradigma. Vamos entender mais sobre o seu funcionamento e papel dentro do algoritmo de ordenação Quicksort.

- Exemplo QuickSort



- Exemplo QuickSort

QuickSort	
Elementos	Caso médio ($n \log n$)
100	0 ms
1.000	0 ms
10.000	39 ms
100.000	43 ms
200.000	50 ms

<https://www.youtube.com/watch?v=WP7KDIjG6IM>

<https://www.youtube.com/watch?v=tIYMCYooo3c>

- Exemplo QuickSort

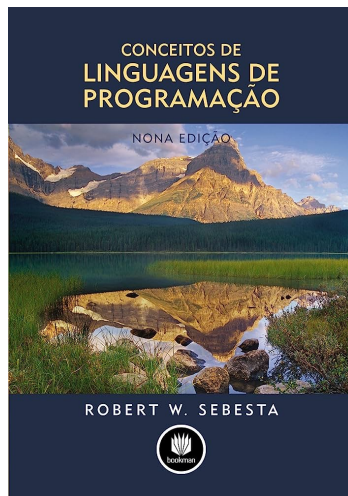
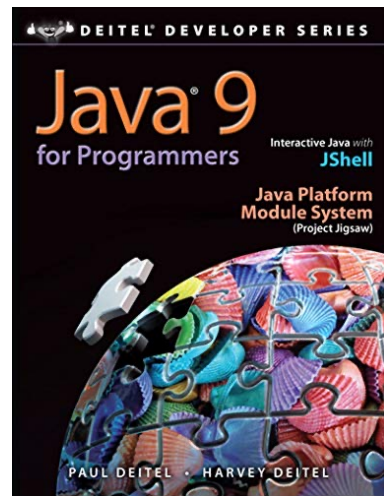
```
3 public class QuickSort {
4
5     public static void main(String[] args) {
6         int quantidade = 100;
7         int[] vetor = new int[quantidade];
8
9         for (int i = 0; i < vetor.length; i++) {
10             vetor[i] = (int) (Math.random() * quantidade);
11             System.out.println(vetor[i]);
12         }
13
14         long tempoInicial = System.currentTimeMillis();
15
16         quickSort(vetor, 0, vetor.length - 1);
17
18         long tempoFinal = System.currentTimeMillis();
19
20         System.out.println("Executado em = " + (tempoFinal - tempoInicial) + " ms");
21         System.out.println("\nVetor Ordenado:");
22         for (int i = 0; i < vetor.length; i++) {
23             System.out.println(vetor[i]);
24         }
25     }
26 }
27
```

- Exemplo QuickSort

Recurso

```
27
28 private static void quickSort(int[] vetor, int inicio, int fim) {
29     if (inicio < fim) {
30         int posicaoPivo = separar(vetor, inicio, fim);
31         quickSort(vetor, inicio, posicaoPivo - 1);
32         quickSort(vetor, posicaoPivo + 1, fim);
33     }
34 }
35
36 private static int separar(int[] vetor, int inicio, int fim) {
37     int pivo = vetor[inicio];
38     int i = inicio + 1, f = fim;
39     while (i <= f) {
40         if (vetor[i] <= pivo)
41             i++;
42         else if (pivo < vetor[f])
43             f--;
44         else {
45             int troca = vetor[i];
46             vetor[i] = vetor[f];
47             vetor[f] = troca;
48             i++;
49             f--;
50         }
51     }
52     vetor[inicio] = vetor[f];
53     vetor[f] = pivo;
54     return f;
55 }
56 }
```

Referências





joao.aramuni@newtonpaiva.br