FATEC – VOTORANTIM

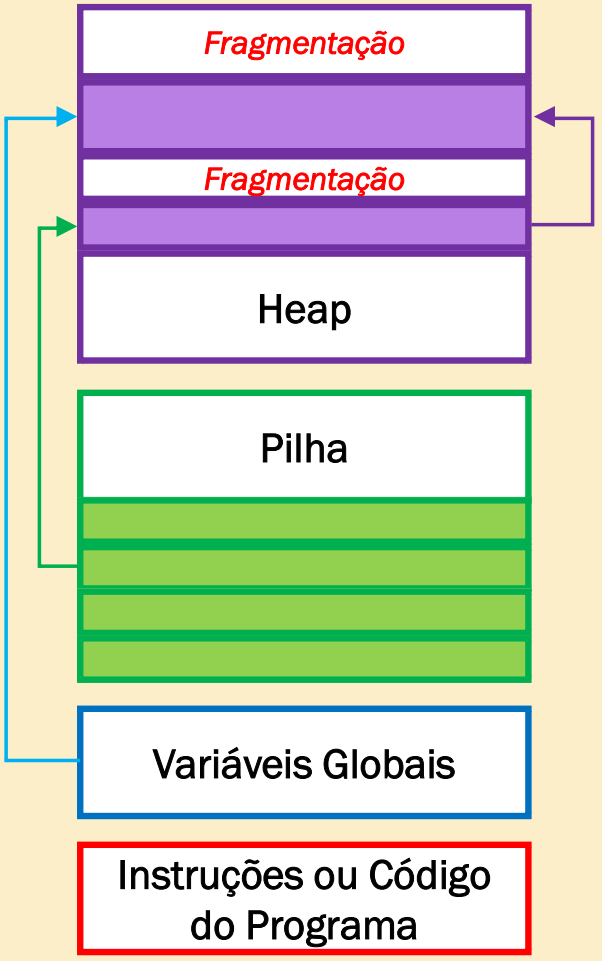
Prof° Rodrigo Diver

Aula 2

- **Alocação de Memória**
- **Recursividade**

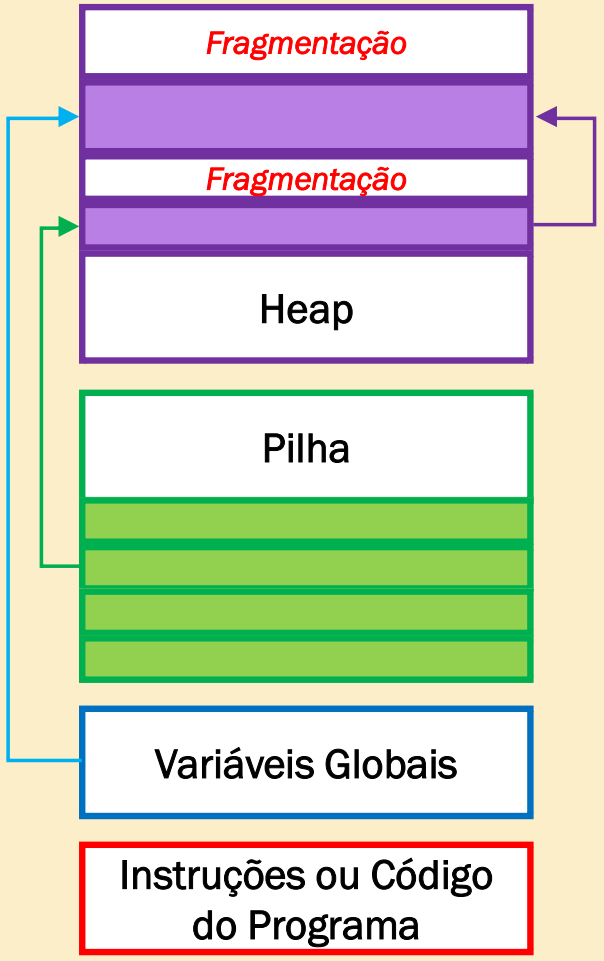
Alocação de Memória

- A maioria das linguagens de programação utilizam um esquema de segmentação de memória em quatro regiões logicamente distintas.
  - **Instruções:** utilizado para armazenar o código do programa sendo executado.
  - **Variáveis Globais:** utilizado para armazenar variáveis declaradas no escopo inicial do programa, como por exemplo constantes.
  - **Pilha:** utilizado para armazenar as variáveis locais e ponteiros de endereçamento para variáveis dinâmicas
  - **Heap:** utilizado para armazenar variáveis alocadas dinamicamente.

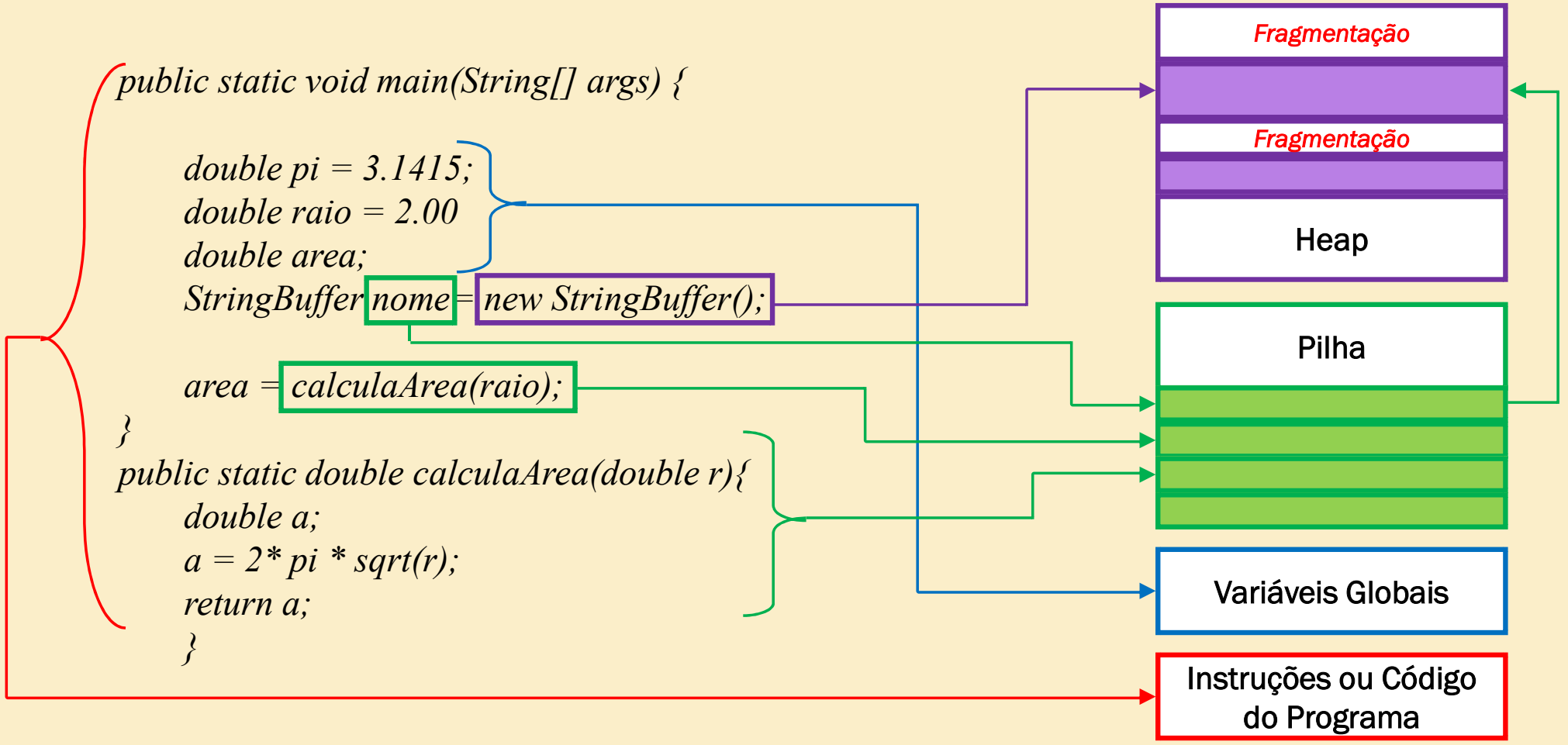


Alocação de Memória

- **Escopo de duração das variáveis:**
  - **Variáveis Globais:** espaço de memória continua alocada enquanto o programa estiver executando.
  - **Variáveis locais:** espaço de memória continua alocada enquanto a função que a declarou estiver executando.
- **Espaço para alocação:**
  - **Variáveis Estáticas:** o tamanho do espaço em memória para a alocação da variável é conhecida.
  - **Variáveis Dinâmicas:** o tamanho do espaço em memória para a alocação é desconhecido, compilador cria um espaço no **Heap** e registra o endereço em um ponteiro na **Pilha.**



Alocação de Memória



Alocação de Memória

Processo de limpeza:

- Em linguagens de programação com C/C++ o processo de limpeza da memória alocada é responsabilidade do programador.
- Na linguagem C existe 4 funções especificas para isso:
  - **Malloc**: permite a alocação de uma nova área de memória para uma variável, retornando um ponteiro para o endereço de memória.
  - **Calloc**: similar ao Malloc, mas a variável é inicializada com zero.
  - **Realoc**: permite realocar o tamanho da área ocupada pela variável.
  - **Free**: permite que uma área alocada seja liberada.

Alocação de Memória

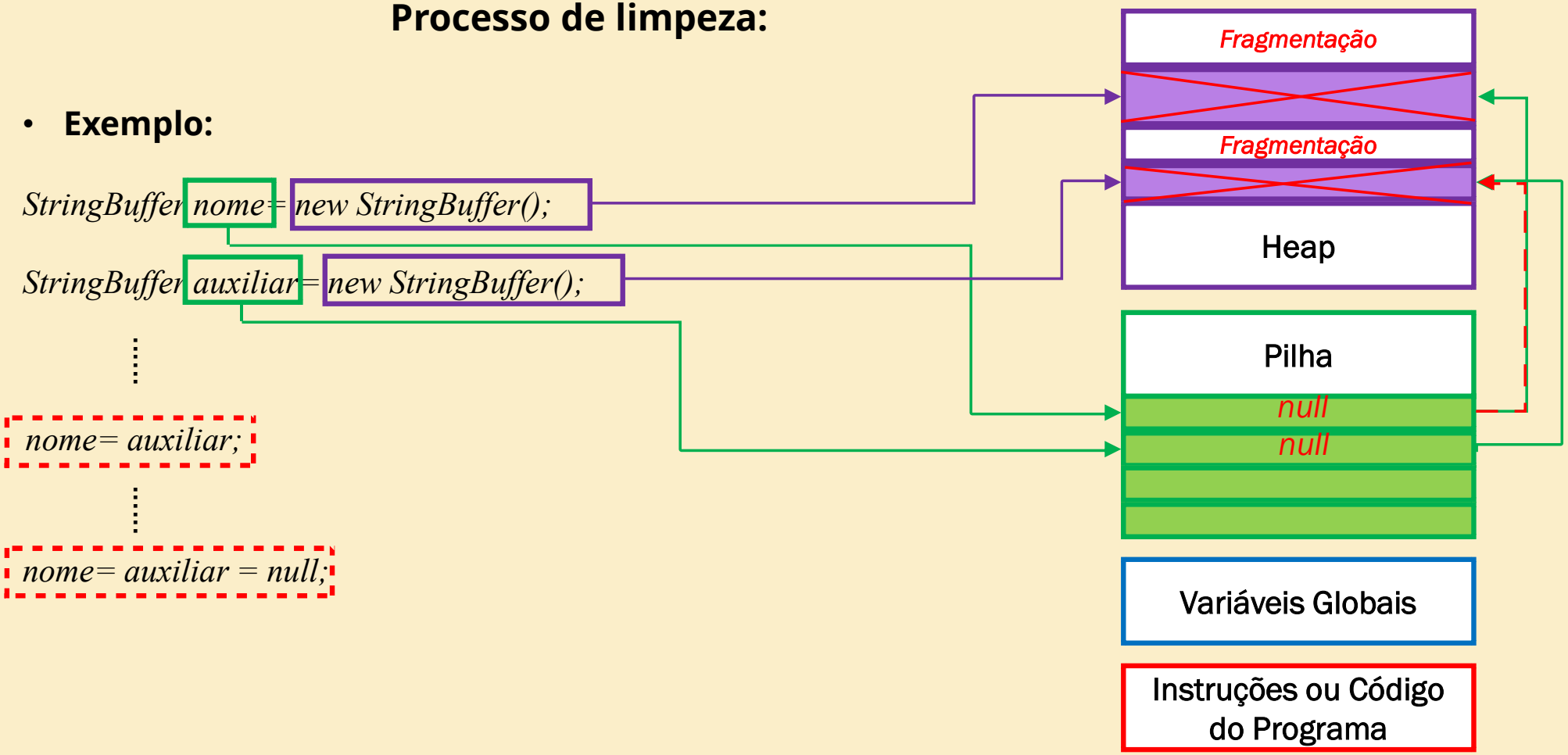
Processo de limpeza:

- Em Java e Python o programador não precisa gerenciar explicitamente a memória do sistema.
- Pois existem mecanismos automáticos de gerenciamento e limpeza da memória, conhecido como coletor de lixo (**Garbage Collector**).
- A técnica de Coleta de Lixo consiste na recuperação segura do espaço de memória ocupado por um objeto que não é mais referenciado dentro de uma aplicação.
- Em linguagens orientadas a objeto é importante que objetos que não são mais referenciados, sejam desalocados para liberar a memória ocupada.

Alocação de Memória

Processo de limpeza:

- **Exemplo:**



Alocação de Memória

Garbage Collector:

- **Possui a função:**
  - Alocar a memória.
  - Assegurar que quaisquer objetos referenciados permaneçam na memória.
  - Recuperar a memória alocada pelos objetos que não são mais alcançáveis pelas referências do código em execução.

- Em geral, quando não há interferência do programador, a máquina virtual executa o Coletor de Lixo quando percebe que a memória está ficando sem espaço.

Alocação de Memória

Garbage Collector:

- De qualquer maneira, uma aplicação pode encerrar sua execução sem que o **Garbage Collector** seja executado uma única vez.
- Quando um objeto deixa de ser referenciado, o espaço de memória ocupado por ele **não é imediatamente desalocado**.
- O **Garbage Collector** pode ser configurado para priorizar diferentes comportamentos:
  - Diminuir a quantidade de intervenções utilizando assim mais memória.
  - Realizar limpezas mais constantes otimizando o uso de memória.

Alocação de Memória

Garbage Collector:

- Leia mais sobre Garbage Collector em Java: https://www.devmedia.com.br/introducao-ao-java-garbage-collection/30326

Alocação de Memória

Exercício

- Pesquise sobre **Garbage Collection,** execute o código abaixo e responda:
  - Com base na quantidade de memória utilizada e na quantidade de variáveis declaras.
  - *Qual o tamanho em bytes de cada variável do tipo **double** ?*

Alocação de Memória

```
public static void main(String[] args) {
        Runtime rt = Runtime.getRuntime();
        System.out.println("Memoria total da JVM: " + rt.totalMemory());
        double m1,m2;
        m1= rt.freeMemory();
        System.out.println("Memoria antes da criação dos objetos: " + rt.freeMemory());

        double vetor[]= new double[100000];
        for (int i = 0; i < 100000; i++) {
            vetor[i]=i;
        }
        m2= rt.freeMemory();
        System.out.println("Memoria depois da criação dos objetos: " + rt.freeMemory());
        System.out.println("Memoria utilizada: " + (m1- m2) + " bytes");
        rt.gc();
        System.out.println("Memoria depois executar o gc: " + rt.freeMemory());
}
```

Recursão

- Recursão pode ser encontrada em matemática, computação e no cotidiano.
- Remetendo a ideia de repetição.
- Repetição de um objeto dentro dele mesmo.
- Equivalente a um *looping* ou laço continuo.
- Imagine um espelho de frente com outro espelho.
- Ou um sonho onde você sonha que está sonhando.

Recursão

- Algoritmos recursivos podem simplificar o entendimento de códigos complexos.
- Porém nem sempre um algoritmo recursivo terá o melhor desempenho.
- De forma geral problemas matemáticos naturalmente recursivos podem ser facilmente implementados utilizando funções recursivas.
- O grande potencial da recursão é a possibilidade de definir elementos com base em versões mais simples desses mesmo elementos.
- Em termos computacionais, trata-se do paradigma de dividir um problema maior em partes menores, que são resolvidos pela mesma função recursiva.

“Dividir para conquistar”.

Recursão

- **Para implementar uma função ou método recursivo é necessário estabelecer:**
  - **Uma condição de parada:** estabelecendo uma solução trivial que encerre a chamada recursiva.
  - **Uma mudança de estado:** estabelecendo uma diferença entre o estado inicial e o estado final após a nova chamada recursiva.
  - **Por exemplo:** o decremento de um parâmetro da função recursiva a cada interação e o estabelecimento de um condicional de parada, como ***n=1***, atende os dois requisitos de uma função recursiva.

Recursão

- *Um exemplo clássico de uma função matemática recursiva é o cálculo do **fatorial** de um número **n**.*

4! = 4 x 3 x 2 x 1 = 24 ➔ n! =

- *Se n =0 ➔ 1*
- *Se n>=1 ➔ n.(n-1).(n-2).(n-3) .....*

```
public int fatorial(int n) {
    // Se n for igual a 0 (zero) então retorna 1.
    if (n == 0) {
        return 1;
    }
    else{
    /* Para qualquer outro número, calcula o seu valor multiplicado pelo fatorial de seu antecessor. */
        return  n * fatorial(n - 1);
    }
}
```

Leia: http://www.universidadejava.com.br/java/java-fatorial/

Recursão

Atenção

- A maioria dos algoritmos recursivos consomem mais recursos computacionais que seus equivalentes interativos. Por isso é importante tomar cuidado com o uso de algoritmos recursivos.

- **Você deve utilizar a recursão quando**:
  - O problema é naturalmente recursivo.
  - A recursividade não gera aumento considerável do custo computacional.
  - Existe um condicional de parada bem definido que permite prever a quantidade de interações geradas pela função recursiva.

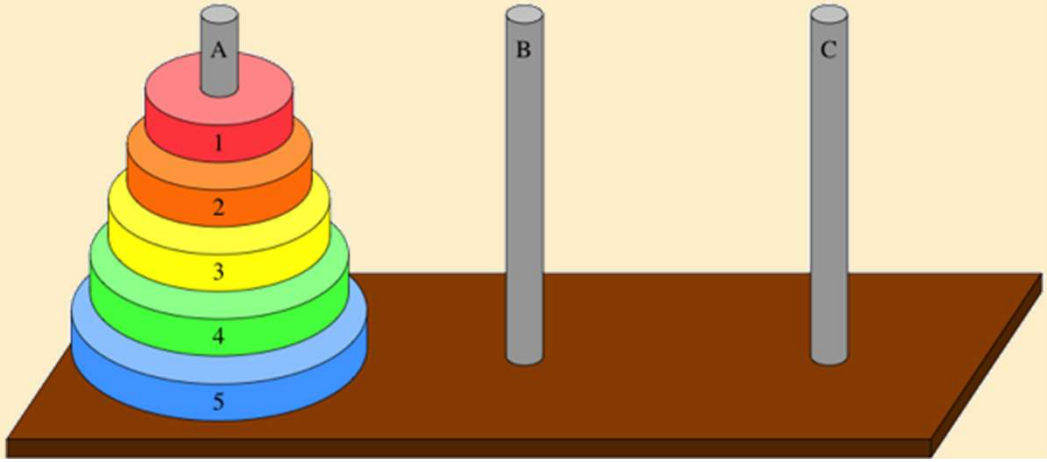
- **Você NÃO deve utilizar a recursão quando:**
  - A função recursiva é ineficiente, se comparada a funções interativas similares.
  - Não existe um condicional de parada.
  - Não é possível prever o número de interações até o estabelecimento do condicional de parada.

Recursão

Exemplo

- **Torre de Hanoi**
  - Consiste de três hastes **A (origem), B (destino) e C (auxiliar).**
  - O objetivo do jogo é transferir todos os discos da origem para o destino.

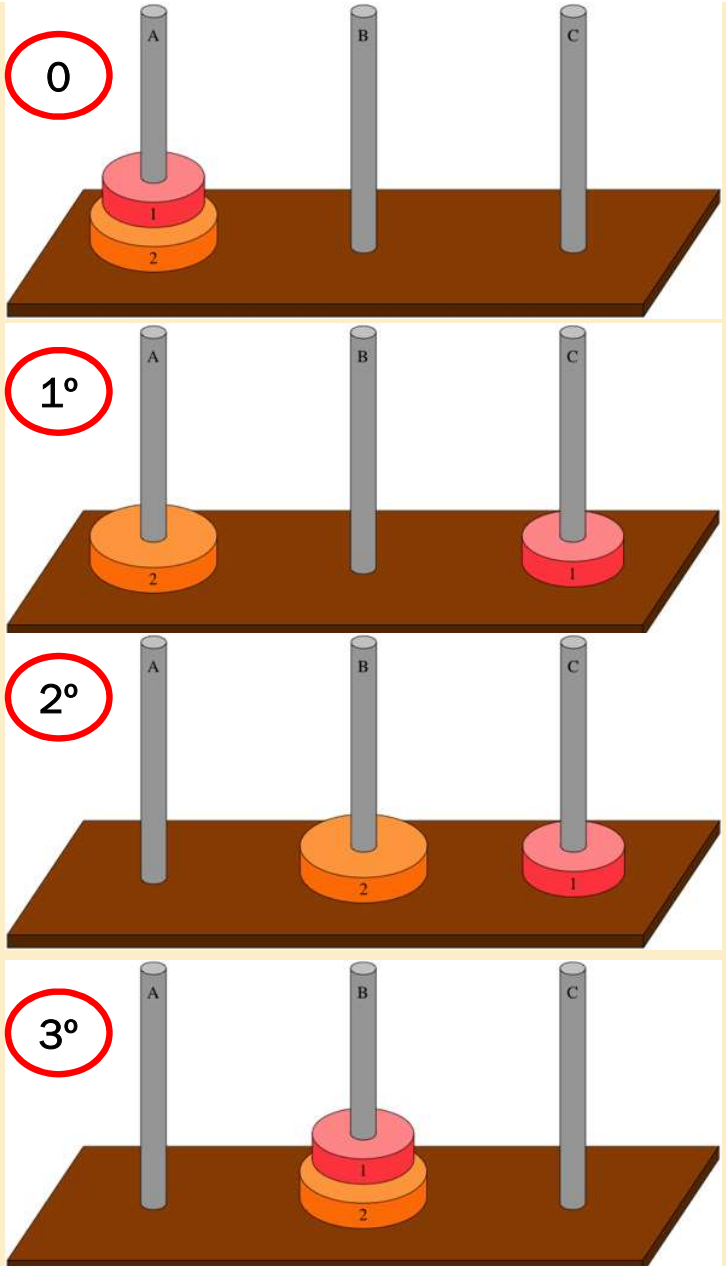
  - ***Respeitando as seguintes regras:***
    - Apenas um disco poderá ser movido por vez.
    - Um disco maior não pode ser movido em cima de um disco menor.



Recursão

Torre de Hanoi

- **Passo a passo da resolução com 2 discos**
  I. Mova o disco 1 para o pino auxiliar C.
  II. Mova o disco 2 para o pino destino B.
  III. Mova o disco 1 para o pino destino B.



Recursão

- Leia mais sobre:
  - Recursividade: https://pt.khanacademy.org/computing/computer-science/algorithms/recursive-algorithms/a/recursion

  - Torre de Hanoi: https://pt.khanacademy.org/computing/computer-science/algorithms/towers-of-hanoi/a/towers-of-hanoi

  Assista: https://www.youtube.com/watch?v=Q2BooYpqS6g&t=428s

  Se você gostar de matemática: https://www.youtube.com/watch?v=CLouA-TA6nc&t=474s

Bibliografia

- **Estrutura de Dados e Técnicas de Programação**
  - **Piva D.J.; Nakamiti, G. S.; Bianchi, F. et (2014)**
- **Histórico da Computação e Principais Componentes Computacionais**
  - https://materialpublic.imd.ufrn.br/curso/disciplina/4/14/1
- **Estruturas de Dados em Java (Apostila)**
  - Prof. Dr. Paulo Roberto Gomes Luzzardi (2010).
- **Estruturas de Dados Abertas(Livro)**
  - Pat Morin e Joao Araujo (2021)
- **Algoritmos e Estrutura de Dados em Java (Apostila)**
  - Caelum ensino e inovação. (2021)