

RECURSIVIDADE

PUC MINAS

ALGORITMOS E ESTRUTURAS DE DADOS II

RECURSIVIDADE

- Para definir novos conceitos;
 - costumamos usar elementos conhecidos.

RECURSIVIDADE – EXEMPLO

- Pensemos no seguinte problema:
 - “Defina o conjunto dos números naturais.”

RECURSIVIDADE – EXEMPLO

- Pensemos no seguinte problema:
 - “Defina o conjunto dos números naturais.”
 - $0 \in \mathbb{N}$;
 - Se $n \in \mathbb{N}$;
 - então $(n + 1) \in \mathbb{N}$.

DEFINIÇÃO RECURSIVA



DEFINIÇÃO RECURSIVA

- Um **objeto definido em termos dele próprio**.
- Geralmente, utilizada para definição de conjuntos infinitos.
- Em programação:
 - um **método recursivo** caracteriza-se por **chamar a si mesmo**;
 - útil para desenvolver algoritmos mais concisos.

DEFINIÇÃO RECURSIVA

1. Caso base ou âncora:

- elementos básicos de um conjunto;
- ponto de parada do método recursivo.

2. Regras de formação:

- regras para construção de novos elementos;
 - a partir dos elementos básicos.

DEFINIÇÃO RECURSIVA – EXEMPLO

- Definição recursiva do **fatorial** de um número:

DEFINIÇÃO RECURSIVA – EXEMPLO

- Definição recursiva do **fatorial** de um número:
 - $0! = 1$

DEFINIÇÃO RECURSIVA – EXEMPLO

- Definição recursiva do **fatorial** de um número:
 - $0! = 1$
 - $1! = 1 * 0! = 1 * 1 = 1$

DEFINIÇÃO RECURSIVA – EXEMPLO

- Definição recursiva do **fatorial** de um número:
 - $0! = 1$
 - $1! = 1 * 0! = 1 * 1 = 1$
 - $2! = 2 * 1! = 2 * 1 = 2$

DEFINIÇÃO RECURSIVA – EXEMPLO

- Definição recursiva do **fatorial** de um número:
 - $0! = 1$
 - $1! = 1 * 0! = 1 * 1 = 1$
 - $2! = 2 * 1! = 2 * 1 = 2$
 - $3! = 3 * 2! = 3 * 2 = 6$

DEFINIÇÃO RECURSIVA – EXEMPLO

- Definição recursiva do **fatorial** de um número:
 - $0! = 1$
 - $1! = 1 * 0! = 1 * 1 = 1$
 - $2! = 2 * 1! = 2 * 1 = 2$
 - $3! = 3 * 2! = 3 * 2 = 6$
 - ...
 - $n! = n * (n - 1)!$

DEFINIÇÃO RECURSIVA – EXEMPLO

- Definição recursiva do **fatorial** de um número:

- $0! = 1$  Caso base

- $1! = 1 * 0! = 1 * 1 = 1$

- $2! = 2 * 1! = 2 * 1 = 2$

- $3! = 3 * 2! = 3 * 2 = 6$

...

- $n! = n * (n - 1)!$  Regra de formação

DEFINIÇÃO RECURSIVA – EXEMPLO

- A **solução do caso base** é dada por **definição**;
 - **não necessita da recursão** para ser alcançada.
- A **solução dos demais casos** pode ser alcançada **reescrevendo-se o problema**;
 - em função de um **problema menor e mais simples**.

DEFINIÇÃO RECURSIVA – EXEMPLO

- Definições recursivas têm um **efeito indesejado**;
 - necessário calcular todos os números da sequência para descobrir o fatorial de n .
- Encontrar uma **fórmula para solucionar o problema sem referenciar outros elementos da sequência**;
 - problema difícil e nem sempre pode ser resolvido.


PROGRAMANDO RECURSIVAMENTE

- Cálculo do fatorial de um número:

```
int fatorial(int n) {  
    if (n==0)  
        return 1;  
    else  
        return (n * fatorial(n-1));  
}
```

PROGRAMANDO RECURSIVAMENTE

- Cálculo do fatorial de um número:

```
int fatorial(int n) {  
    if (n==0)  
        return 1;  Caso base: 0! = 1  
    else  
        return (n * fatorial(n-1));  
}
```

Regra de formação: $n! = n * (n-1)!$

CHAMADA RECURSIVA

- Um método pode ser chamado pelo programa principal (método **main**) e por outros métodos;
 - inclusive ele mesmo.
- Sempre que um **método é iniciado**;
 - um **novo conjunto de variáveis locais e de parâmetros é alocado**.
- Quando ocorre um **retorno**;
 - o **conjunto de variáveis locais e parâmetros do método que estava sendo executado é desativado**;
 - dando lugar ao conjunto de valores do método que fez a chamada.

CHAMADA RECURSIVA

- Quando um método é chamado;
 - uma pilha é utilizada para armazenar o estado do método chamador.
- Estado de um método:
 - caracterizado pelo conteúdo de todas as suas variáveis locais, seus parâmetros e endereço de retorno.
- Endereço de retorno:
 - serve para que o sistema “lembre-se” onde deve retomar a execução do método chamador.

CHAMADA RECURSIVA

- Todo algoritmo deve terminar após a execução de uma quantidade finita de passos.
- Um **método recursivo** deve **terminar** sua **execução** em algum instante;
 - deve **atingir**, em algum momento, o **caso base**;
 - sua **condição de parada**.
- Caso contrário, o método poderá entrar em *looping* e ser executado indefinidamente;
 - estouro da pilha do sistema.

CHAMADA RECURSIVA – EXEMPLO

- No exemplo do cálculo do fatorial ... :
 - ... a **aplicação repetitiva da etapa indutiva leva ao caso base;**
 - condição de parada;
 - última etapa na sequência de chamadas recursivas.

CHAMADA RECURSIVA – EXEMPLO

```
public static void main (String[]  
args) {  
    int x = 4;  
    int fat = fatorial (x);  
    System.out.println("Fatorial  
de " + x + ": " + fat);  
}
```

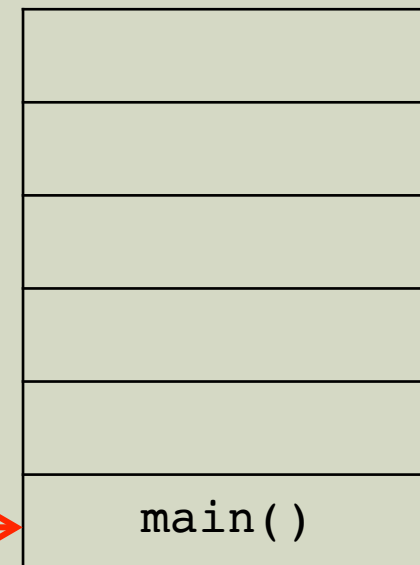
CHAMADA RECURSIVA – EXEMPLO

```
public static void main (String[]  
args) {  
    int x = 4;  
    int fat = fatorial (x); ←  
    System.out.println("Fatorial  
de " + x + ": " + fat);  
}
```


CHAMADA RECURSIVA – EXEMPLO

```
public static void main (String[] args) {  
    int x = 4;  
    int fat = fatorial (x); ←  
    System.out.println("Fatorial  
de " + x + ": " + fat);  
}
```

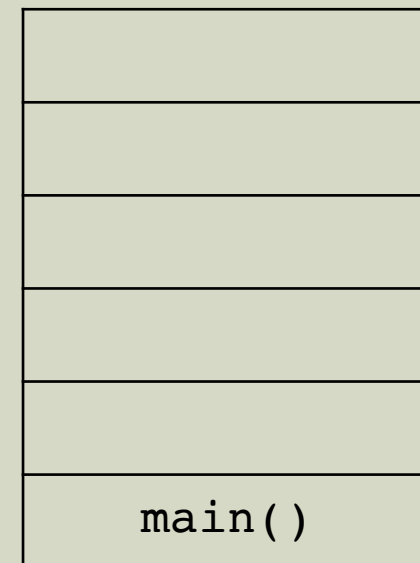
Sempre que um método chama outro, seu estado é armazenado na pilha de execução do programa.



CHAMADA RECURSIVA – EXEMPLO

```
int fatorial(int n) { ←  
    if (n==0)          n = 4  
        return 1;  
    else  
        return (n *  
                fatorial(n-1));  
}
```

Pilha de execução do programa

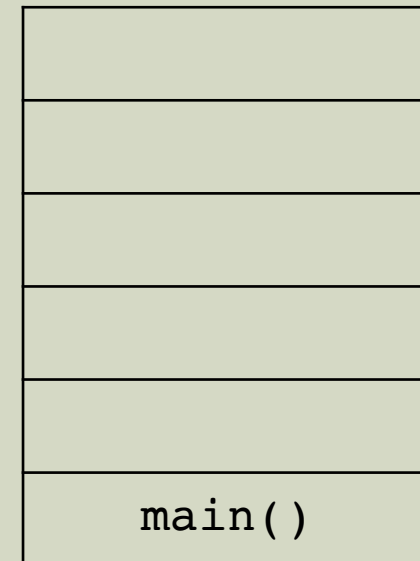


CHAMADA RECURSIVA – EXEMPLO

```
int fatorial(int n) {  
    if (n==0)                n = 4  
        return 1;  
    else  
        return (n *  
                fatorial(n-1) ); ←  
}
```

} Após a verificação da condição **if**, como **n** é diferente de zero, invoca-se o método novamente, mas agora decrementando o valor de **n**. (Observe que não ocorre o retorno ainda).

Pilha de execução do programa

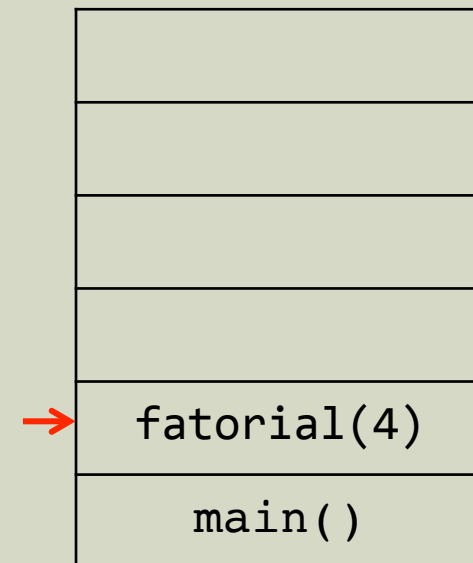


CHAMADA RECURSIVA – EXEMPLO

```
int fatorial(int n) {  
    if (n==0)                n = 4  
        return 1;  
    else  
        return (n *  
                fatorial(n-1) ); ←  
}
```

Sempre que um método chama outro, seu estado é armazenado na pilha de execução do programa.

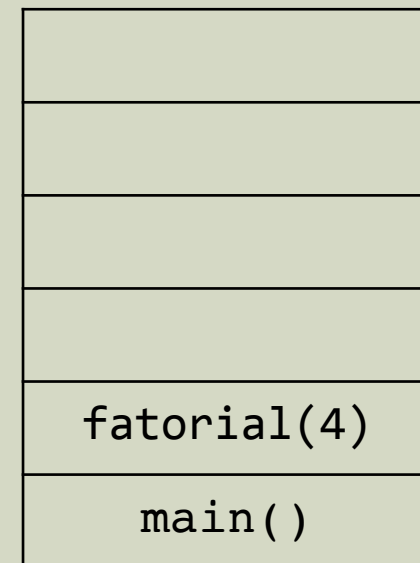
Pilha de execução do programa



CHAMADA RECURSIVA – EXEMPLO

```
int fatorial(int n) { ←  
    if (n==0)          n = 3  
        return 1;  
    else  
        return (n *  
                fatorial(n-1));  
}
```

Pilha de execução do programa



CHAMADA RECURSIVA – EXEMPLO

```
int fatorial(int n) {  
    if (n==0)                n = 3  
        return 1;  
    else  
        return (n *  
                fatorial(n-1) ); ←
```

} Como **n** ainda é diferente de zero, invoca-se o método novamente, mas agora decrementando o valor de **n**. (Observe que não ocorre o retorno ainda).

Pilha de execução do programa

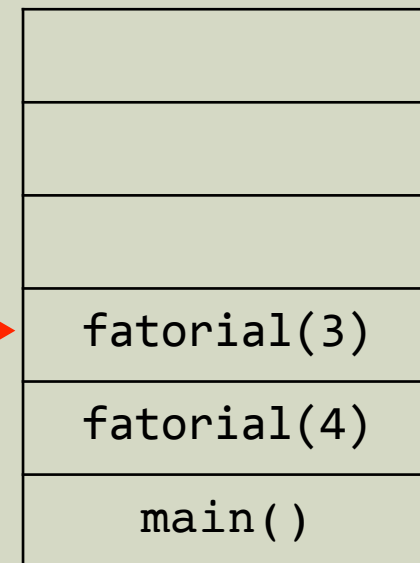
fatorial(4)
main()

CHAMADA RECURSIVA – EXEMPLO

```
int fatorial(int n) {  
    if (n==0)                n = 3  
        return 1;  
    else  
        return (n *  
                fatorial(n-1) ); ←  
}
```

Sempre que um método chama outro, seu estado é armazenado na pilha de execução do programa.

Pilha de execução do programa



CHAMADA RECURSIVA – EXEMPLO

```
int fatorial(int n) { ←  
    if (n==0)          n = 2  
        return 1;  
    else  
        return (n *  
                fatorial(n-1));  
}
```

Pilha de execução do programa

fatorial(3)
fatorial(4)
main()

CHAMADA RECURSIVA – EXEMPLO

```
int fatorial(int n) {  
    if (n==0)                n = 2  
        return 1;  
    else  
        return (n *  
                fatorial(n-1) ); ←  
}
```

} Como **n** ainda é diferente de zero, invoca-se o método novamente, mas agora decrementando o valor de **n**. (Observe que não ocorre o retorno ainda).

Pilha de execução do programa

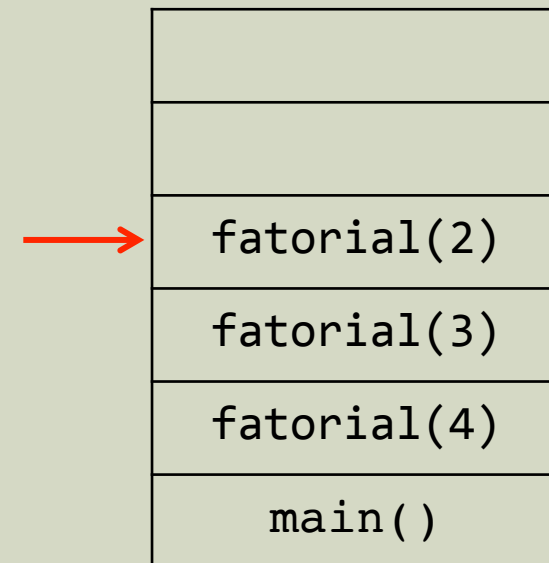
fatorial(3)
fatorial(4)
main()

CHAMADA RECURSIVA – EXEMPLO

```
int fatorial(int n) {  
    if (n==0)                n = 2  
        return 1;  
    else  
        return (n *  
                fatorial(n-1) ); ←  
}
```

Sempre que um método chama outro, seu estado é armazenado na pilha de execução do programa.

Pilha de execução do programa



CHAMADA RECURSIVA – EXEMPLO

```
int fatorial(int n) { ←  
    if (n==0)          n = 1  
        return 1;  
    else  
        return (n *  
                fatorial(n-1));  
}
```

Pilha de execução do programa

fatorial(2)
fatorial(3)
fatorial(4)
main()

CHAMADA RECURSIVA – EXEMPLO

```
int fatorial(int n) {  
    if (n==0)                n = 1  
        return 1;  
    else  
        return (n *  
                fatorial(n-1) ); ←
```

} Como **n** ainda é diferente de zero, invoca-se o método novamente, mas agora decrementando o valor de **n**. (Observe que não ocorre o retorno ainda).

Pilha de execução do programa

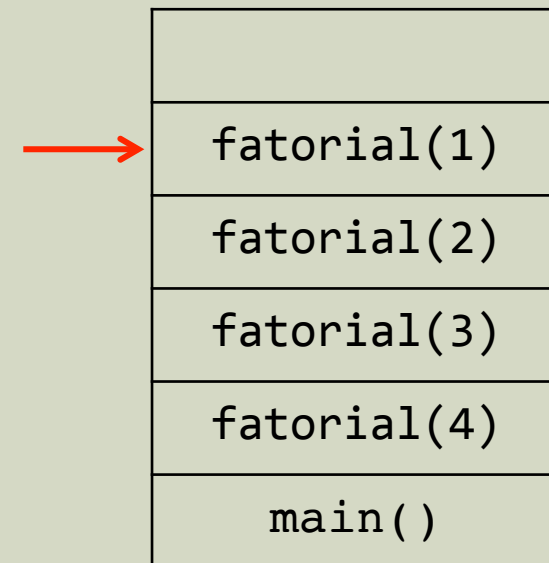
fatorial(2)
fatorial(3)
fatorial(4)
main()

CHAMADA RECURSIVA – EXEMPLO

```
int fatorial(int n) {  
    if (n==0)                n = 1  
        return 1;  
    else  
        return (n *  
                fatorial(n-1) ); ←  
}
```

Sempre que um método chama outro, seu estado é armazenado na pilha de execução do programa.

Pilha de execução do programa



CHAMADA RECURSIVA – EXEMPLO

```
int fatorial(int n) { ←  
    if (n==0)          n = 0  
        return 1;  
    else  
        return (n *  
                fatorial(n-1));  
}
```

Pilha de execução do programa

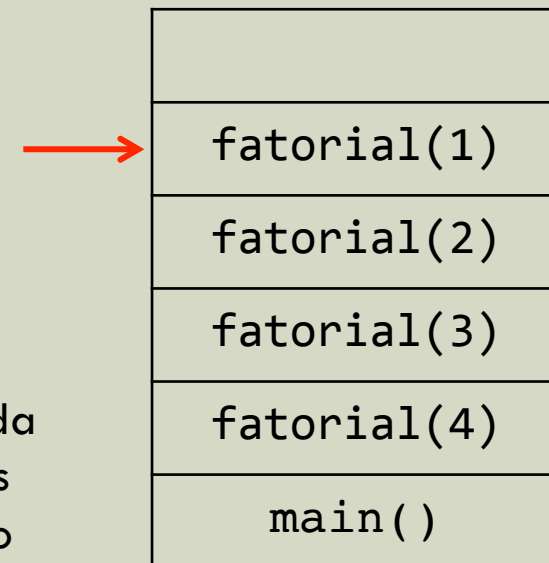
fatorial(1)
fatorial(2)
fatorial(3)
fatorial(4)
main()

CHAMADA RECURSIVA – EXEMPLO

```
int fatorial(int n) {  
    if (n==0)                n = 0  
        return 1; ←  
    else  
        return (n *  
                fatorial(n-1));  
}
```

} Como **n** é igual a zero, apenas retorna 1 para a chamada imediatamente anterior do próprio método. As chamadas recursivas acontecem até que o caso base seja alcançado (condição de parada).

Pilha de execução do programa



CHAMADA RECURSIVA – EXEMPLO

```
int fatorial(int n) {  
    if (n==0)                n = 1  
        return 1;  
    else  
        return (n *  
                fatorial(n-1) ); ←  
}
```

} A chamada atual retornará o valor da chamada anterior (igual a 1) vezes n (igual a 1, nesta chamada).

Pilha de execução do programa

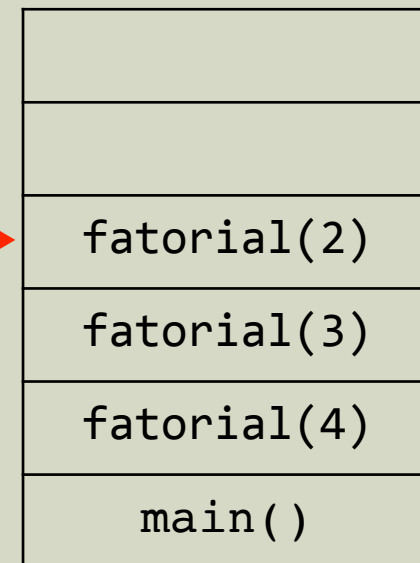
fatorial(2)
fatorial(3)
fatorial(4)
main()

CHAMADA RECURSIVA – EXEMPLO

```
int fatorial(int n) {  
    if (n==0)                n = 1  
        return 1;  
    else  
        return (n *  
                fatorial(n-1) ); ←  
}
```

Retorna 1 para a chamada imediatamente anterior do próprio método.

Pilha de execução do programa



CHAMADA RECURSIVA – EXEMPLO

```
int fatorial(int n) {  
    if (n==0)                n = 2  
        return 1;  
    else  
        return (n *  
                fatorial(n-1) ); ←  
}
```

A chamada atual retornará o valor da chamada anterior (igual a 1) vezes n (igual a 2, nesta chamada).

Pilha de execução do programa

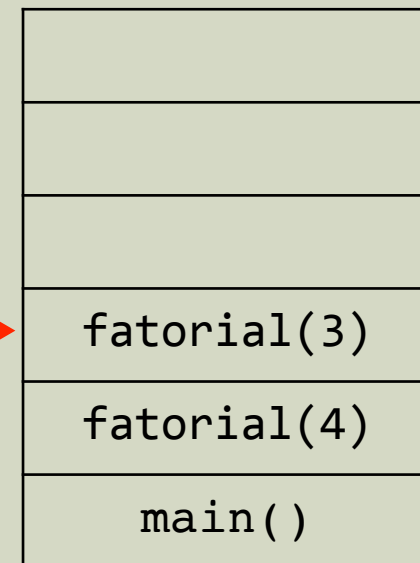
fatorial(3)
fatorial(4)
main()

CHAMADA RECURSIVA – EXEMPLO

```
int fatorial(int n) {  
    if (n==0)                n = 2  
        return 1;  
    else  
        return (n *  
                fatorial(n-1) ); ←  
}
```

Retorna 2 para a chamada imediatamente anterior do próprio método.

Pilha de execução do programa



CHAMADA RECURSIVA – EXEMPLO

```
int fatorial(int n) {  
    if (n==0)                n = 3  
        return 1;  
    else  
        return (n *  
                fatorial(n-1) ); ←
```

} A chamada atual retornará o valor da chamada anterior (igual a 2) vezes n (igual a 3, nesta chamada).

Pilha de execução do programa

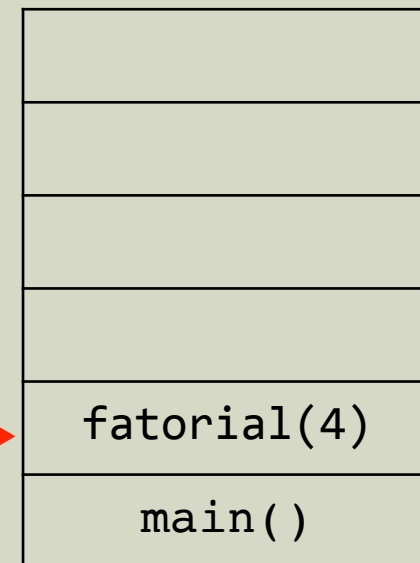
fatorial(4)
main()

CHAMADA RECURSIVA – EXEMPLO

```
int fatorial(int n) {  
    if (n==0)                n = 3  
        return 1;  
    else  
        return (n *  
                fatorial(n-1) ); ←  
}
```

Retorna 6 para a chamada imediatamente anterior do próprio método.

Pilha de execução do programa

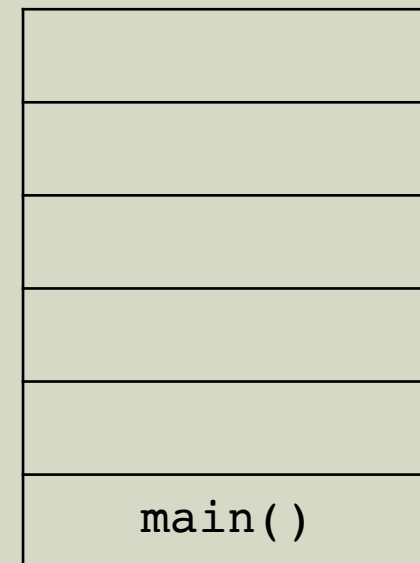


CHAMADA RECURSIVA – EXEMPLO

```
int fatorial(int n) {  
    if (n==0)                n = 4  
        return 1;  
    else  
        return (n *  
                fatorial(n-1) ); ←  
}
```

A chamada atual retornará o valor da chamada anterior (igual a 6) vezes n (igual a 4, nesta chamada).

Pilha de execução do programa

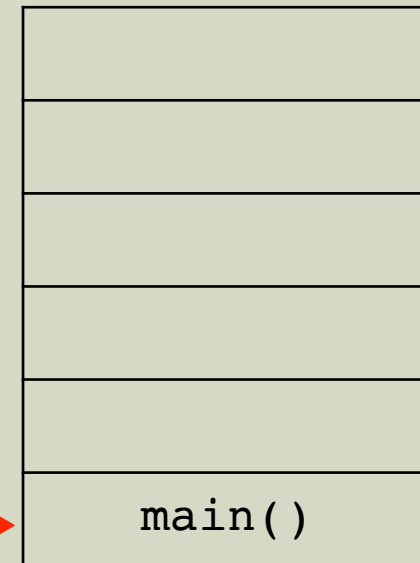


CHAMADA RECURSIVA – EXEMPLO

```
int fatorial(int n) {  
    if (n==0)                n = 4  
        return 1;  
    else  
        return (n *  
                fatorial(n-1)); ←  
}
```

Retorna 24 para o método chamador.

Pilha de execução do programa



CHAMADA RECURSIVA – EXEMPLO

```
public static void main (String[] args) {  
    int x = 4;  
    int fat = fatorial (x); ←  
    System.out.println("Fatorial  
de " + x + ": " + fat);  
}
```

Pilha de execução do programa



} Valor de retorno da primeira chamada do método fatorial() pelo método main(). A variável **fat** receberá o valor 24 (resultado de 4!).

O valor 24 será exibido na tela e o programa será finalizado.

RECURSIVIDADE DE CAUDA

- **Não há código a ser executado;**
 - **após a chamada recursiva.**
- **Não é necessário, portanto,**
armazenar o estado do método;
 - **antes da chamada recursiva.**

RECURSIVIDADE SEM CAUDA

- Há código a ser executado;
 - após a chamada recursiva.
- Maior dificuldade para compreensão da lógica de funcionamento do método recursivo.

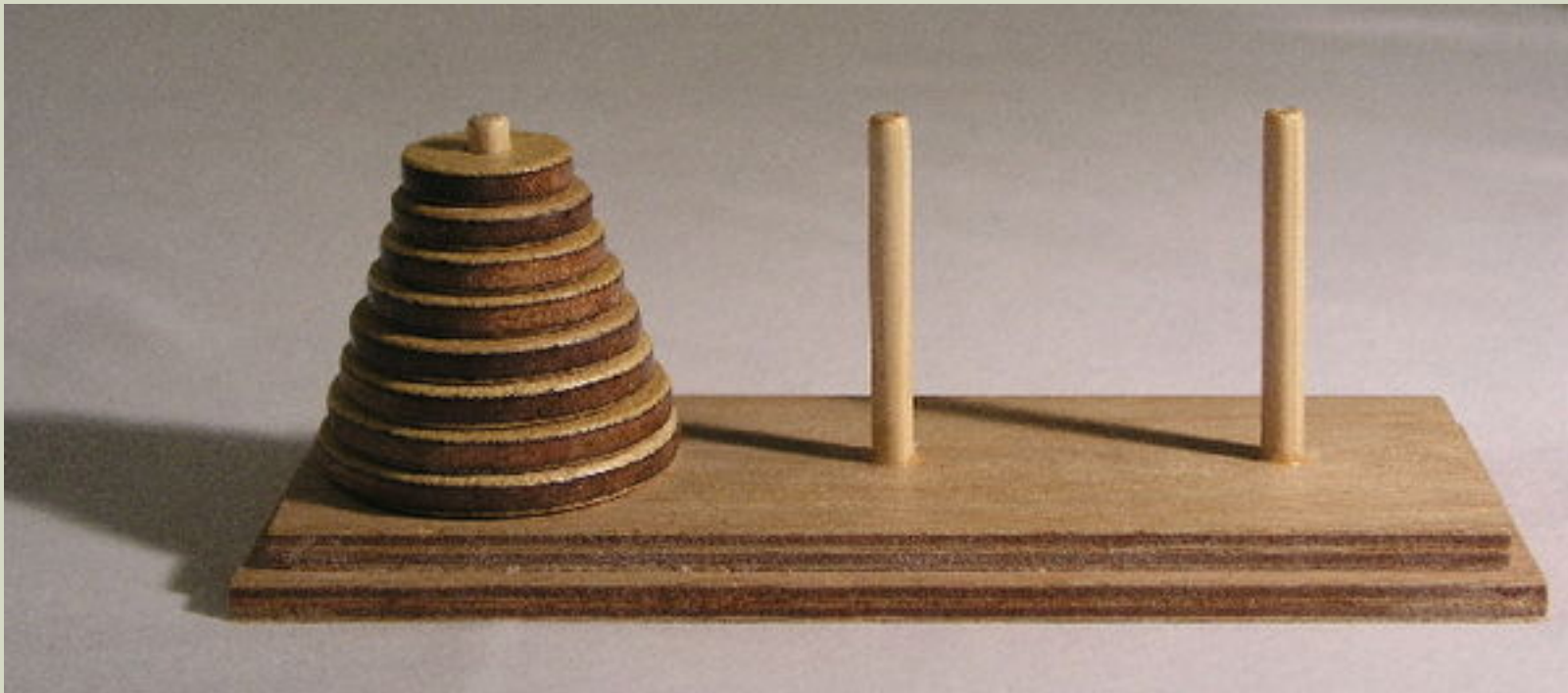
RECURSIVIDADE SEM CAUDA – EXEMPLO

```
public static void inverter (BufferedReader
teclado) throws IOException{
    char letra;
    letra = (char) teclado.read();
    if (letra != '\n'){
        inverter(teclado);
        System.out.print(letra);
    }
}
```

TORRES DE HANÓI

- Dados N discos;
 - cada um com diâmetro $i = 1, 2, 3, \dots, N$;
 - transferir todos os discos de um suporte para outro;
 - um de cada vez;
 - um disco só pode ficar sobre outro de diâmetro superior.

TORRES DE HANÓI



PUC Minas – Engenharia de Software – Algoritmos e Estruturas de Dados II – Prof.^a Eveline Alonso Veloso