Recursividade

Pesquisa, Ordenação e Técnicas de Armazenamento

Prof. Msc. Bruno de A. Iizuka Moritani
bruno.moritani@anhembi.br

Agenda

- Processos Iterativos
- Recursividade
- Exercícios

Processos Iterativos

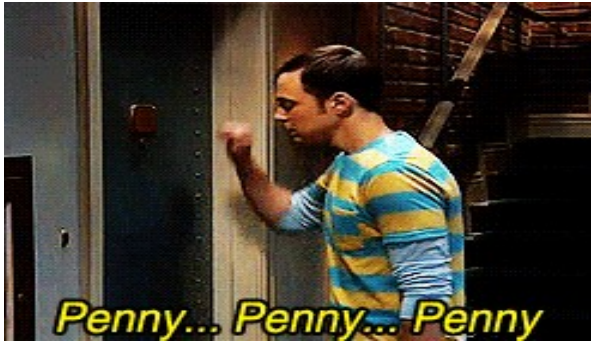
- Iterar
  - *i·te·rar*
    1. Fazer ou dizer novamente: *Iterou as provocações e todos se revoltaram.*
- Referem-se a repetições
  - Podem ser obtidos por meio de estruturas de laço
    - `While`
    - `do...while`
    - `for`

Processos Iterativos

```
public static void contagemRegressiva(int n) {
   while (n > 0) {
      System.out.println(n);
      n--;
   }
   System.out.println(”Feliz Ano Novo!");
}
public static void main(String [] args) {
   contagemRegressiva(10);
}
```

Processos Iterativos

```
public static void baterNaPorta(int n) {
   for (int i = 0 ; i < n ; i++){
      System.out.println(“Knock!”);
      System.out.println(“Knock!”);
      System.out.println(“Knock!”);
      System.out.println(“Penny!”);
   }
}
public static void main(String [] args) {
   baterNaPorta(3);
}
```



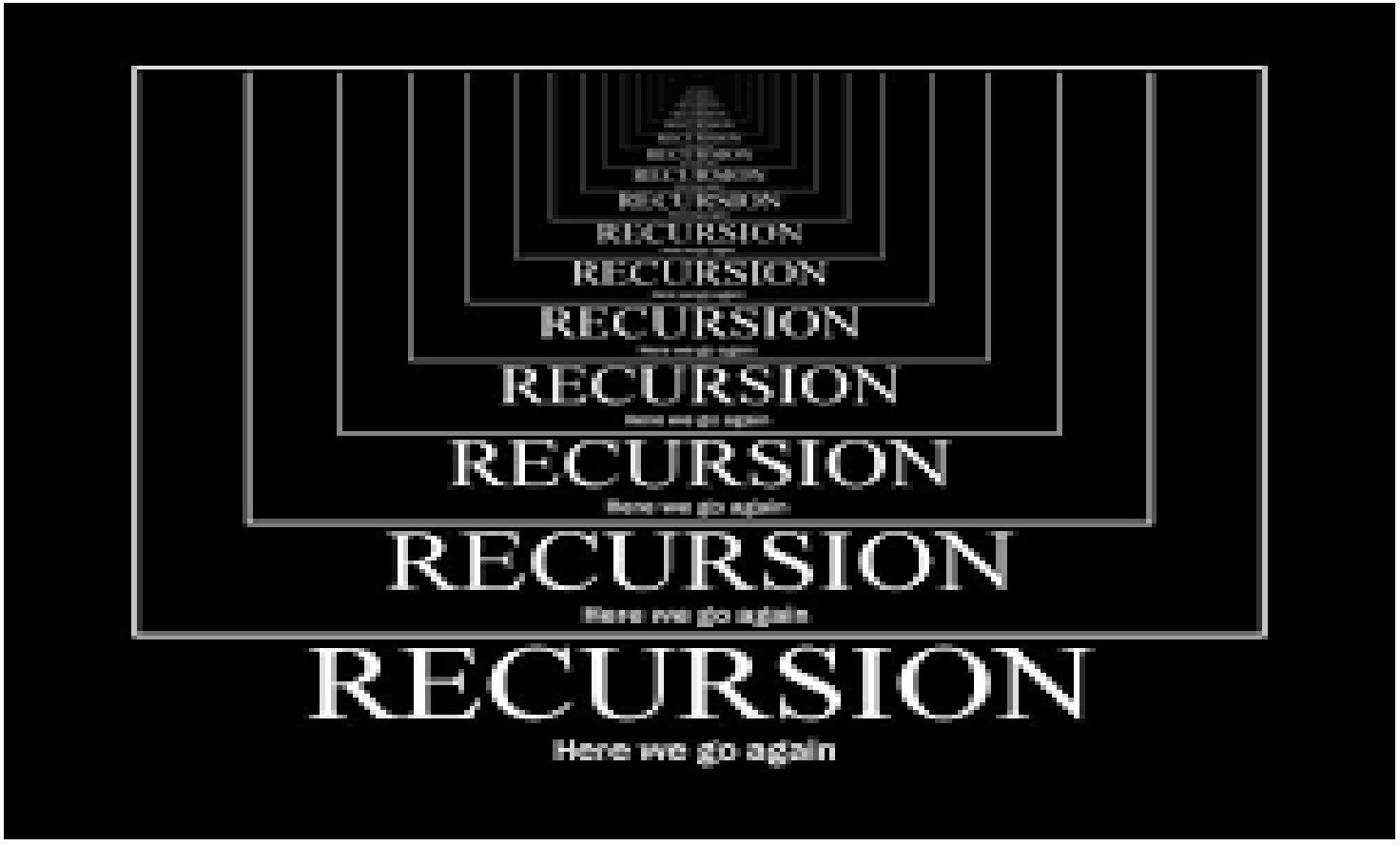
Exercício 01

- Crie um programa que peça um número inteiro ao usuário e retorne a soma de todos os números de 1 até o número que o usuário digitou.
  - 1 + 2 + 3 + ... + n
  - Obs.: Crie um método para realizar a soma

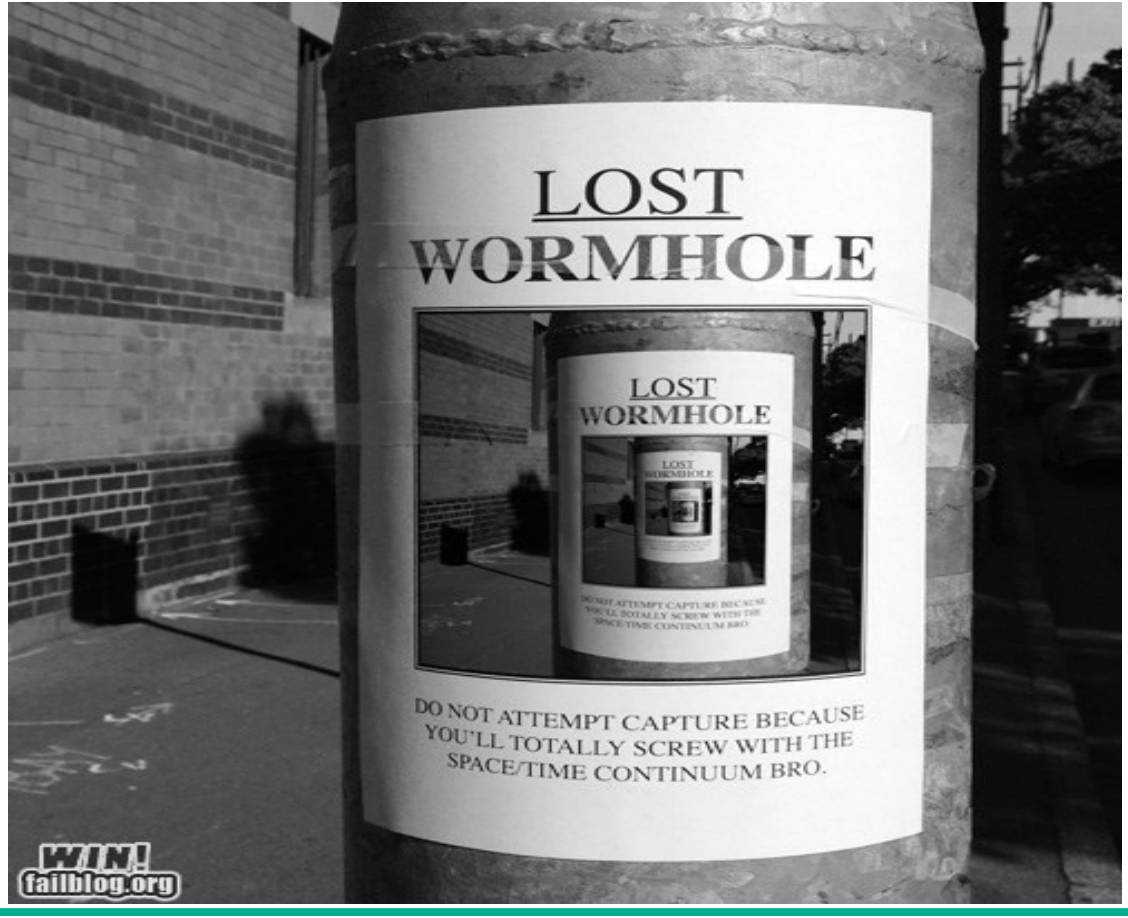
Resposta Exercício 01

```
public class POTAAula02Exercicio01 {

    public static int soma(int n) {
        int soma = 0;
        for (int i = 1; i <= n; i++) {
            soma = soma + i;
        }
        return soma;
    }

    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        int n;
        System.out.println("Digite um inteiro positivo");
        n = scan.nextInt();
        System.out.println("Soma: " + soma(n));
    }

}
```

Recursividade



Recursividade



Recursividade

- Recursividade
  - 1. Que se pode repetir até ao infinito.
- Uma função é chamada de recursiva, quando ela invoca a si mesma, direta ou indiretamente, uma ou mais vezes para resolver subproblemas correlatos.

Recursividade

- É outra forma de se obter repetições

Recursividade

- Abordagem de dividir para conquistar:
  - Desmembre o problema em vários subproblemas semelhantes ao original
  - Resolva recursivamente os subproblemas
  - Combine essas soluções com o objetivo de criar uma solução para o problema original

Recursividade

- É necessário estabelecer pelo menos dois elementos:
  - Uma condição de parada. Geralmente, essa condição estabelece um evento que encerra a auto-chamada consecutiva;
  - Uma mudança de estado a cada chamada, ou seja, o estabelecimento de alguma diferença entre o estado inicial e o próximo estado do método.

Recursividade Exemplo

- Crie um programa que peça um número inteiro ao usuário e retorne a soma de todos os números de 1 até o número que o usuário digitou.
  - 1 + 2 + 3 + ... + n = ?

Recursividade - Exemplo

- Vamos criar uma função soma(n):
  - Se n = 5, essa função deve retornar:
    - soma(5) = 5 + 4 + 3 + 2 + 1
  - Se n = 4, essa função deve retornar:
    - Soma(4) = 4 + 3 + 2 + 1
  - Se n = 3, essa função deve retornar:
    - Soma(3) = 3 + 2 + 1
  - Se n = 2, essa função deve retornar:
    - Soma(2) = 2 + 1
  - Se n = 1, essa função deve retornar:
    - soma(1) = 1

Recursividade - Exemplo

- Para a recursividade, é necessário identificar padrões
  - Soma(5) = 5 + 4 + 3 + 2 +1
    - ou
  - Soma(5) = 5 + Soma(4)
  - Soma(4) = 4 + 3 + 2 +1
    - ou
  - Soma(4) = 4 + Soma(3)

Recursividade - Exemplo

- Fórmula geral:
  - soma(n) = n + soma(n-1)
- Ou seja:
  - soma(n) = n + soma(n-1)
  - = n + (n-1) + soma(n-2)
  - = n + (n-1) + (n-2) + soma(n-3)...

Recursividade - Exemplo

- E quando essa soma para?
  - Para quando o último elemento dessa soma for 1.
  - soma(n) = n +(n-1) + (n-2) + (n-3) + ... + 1

Recursividade - Exemplo

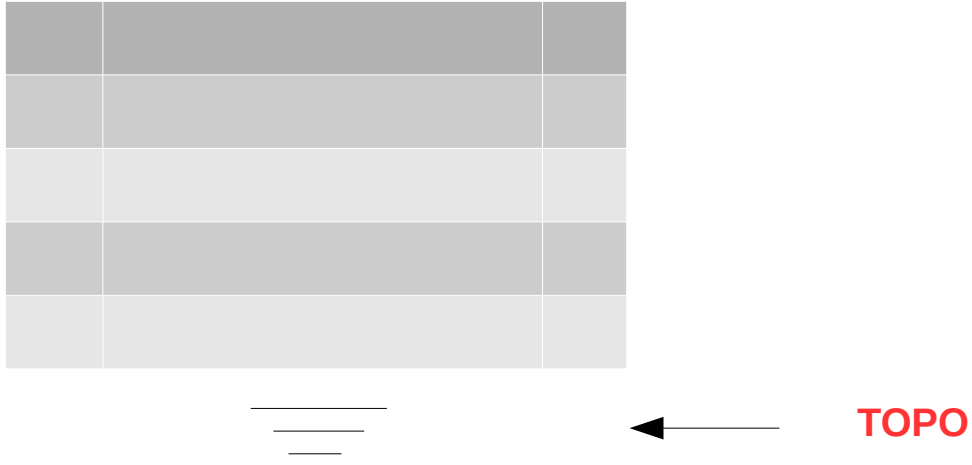
- A função recebe um número, e a primeira coisa que ela deve fazer é verificar se esse valor é 1
  - Se for, deve retornar 1, afinal:
    - soma(1) = 1
  - Se não for 1, deve retornar:
    - n + soma(n-1)

Recursividade - Exemplo

```
public class POTAAula02Exemplo01 {

    public static int soma(int n) {
        if (n == 1) {
            return 1;
        } else {
            return (n + soma(n - 1));
        }
    }

    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        int n;
        System.out.println("Digite um inteiro positivo");
        n = scan.nextInt();
        System.out.println("Soma: " + soma(n));
    }

}
```

Recursividade

- E como o programa armazena esses valores?
  - É criada uma pilha (implícita ao programador) para armazenar esses valores durante a execução.

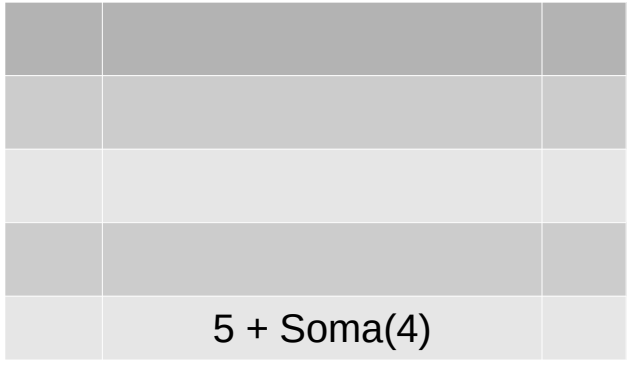


Recursividade

- Exemplo: n = 5

```
public static int soma(int n) {
    if (n == 1) {
        return 1;
    } else {
        return (n + soma(n - 1));
    }
}
```

Insere na Pilha
5 + soma (4)

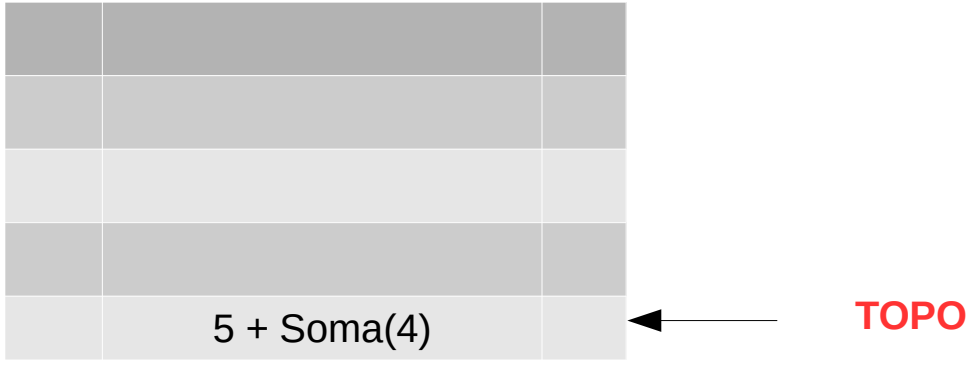


Recursividade

- Exemplo: n = 5

```
public static int soma(int n) {
    if (n == 1) {
        return 1;
    } else {
        return (n + soma(n - 1));
    }
}
```

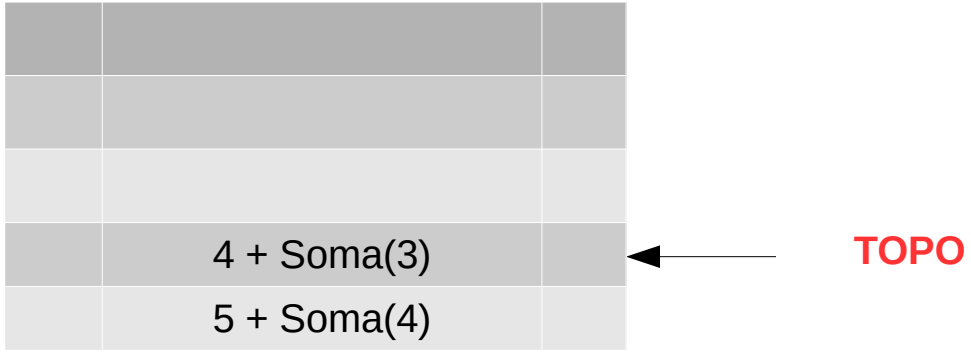
Insere na Pilha
4 + soma (3)



Recursividade

- Exemplo: n = 5

```
public static int soma(int n) {
    if (n == 1) {
        return 1;
    } else {
        return (n + soma(n - 1));
    }
}
```

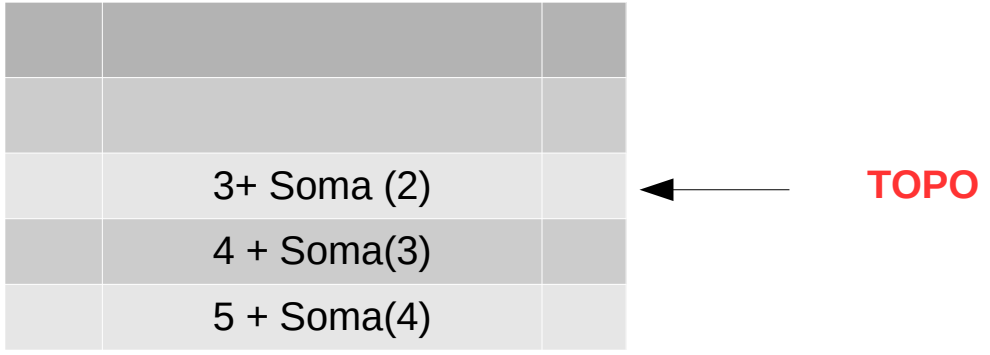


Insere na Pilha
3 + soma (2)

Recursividade

- Exemplo: n = 5

```
public static int soma(int n) {
    if (n == 1) {
        return 1;
    } else {
        return (n + soma(n - 1));
    }
}
```



Insere na Pilha
2 + soma (1)

Recursividade

- Exemplo: n = 5

```
public static int soma(int n) {
    if (n == 1) {
        return 1;
    } else {
        return (n + soma(n - 1));
    }
}
```

Insere na Pilha
1

	2+Soma(1)	
	3+ Soma (2)	
	4 + Soma(3)	
	5 + Soma(4)	

TOPO

Recursividade

- Exemplo: n = 5

```
public static int soma(int n) {
    if (n == 1) {
        return 1;
    } else {
        return (n + soma(n - 1));
    }
}
```

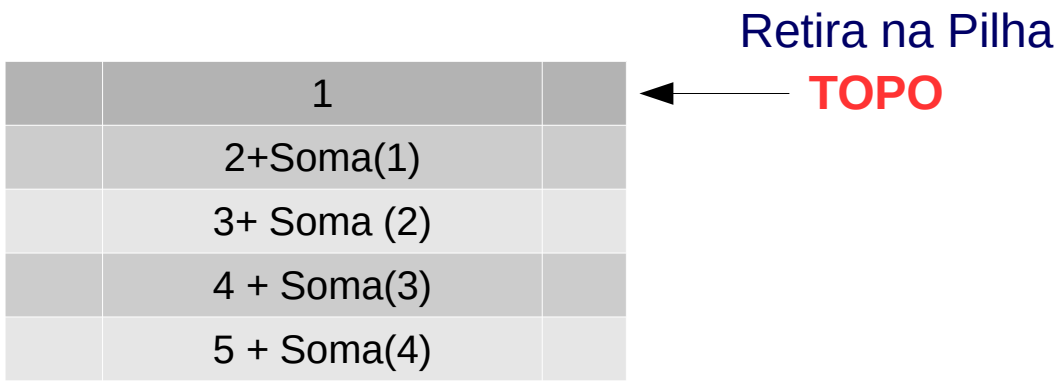
1
2+Soma(1)
3+ Soma (2)
4 + Soma(3)
5 + Soma(4)

TOPO

Recursividade

- Exemplo: n = 5

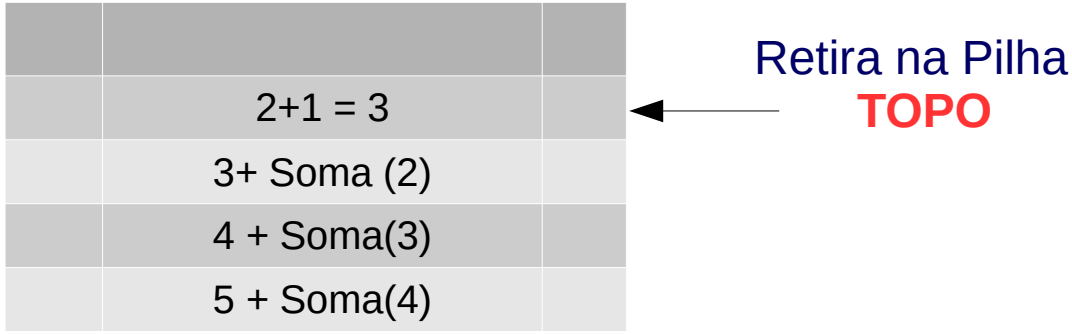
```
public static int soma(int n) {
    if (n == 1) {
        return 1;
    } else {
        return (n + soma(n - 1));
    }
}
```



Recursividade

- Exemplo: n = 5

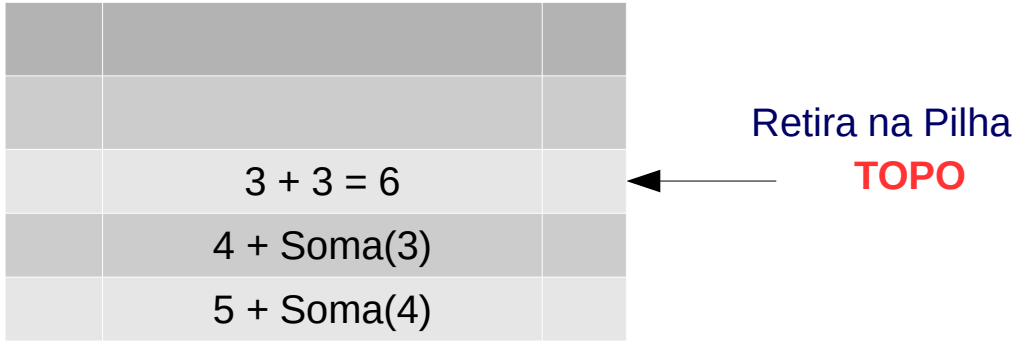
```
public static int soma(int n) {
    if (n == 1) {
        return 1;
    } else {
        return (n + soma(n - 1));
    }
}
```



Recursividade

- Exemplo: n = 5

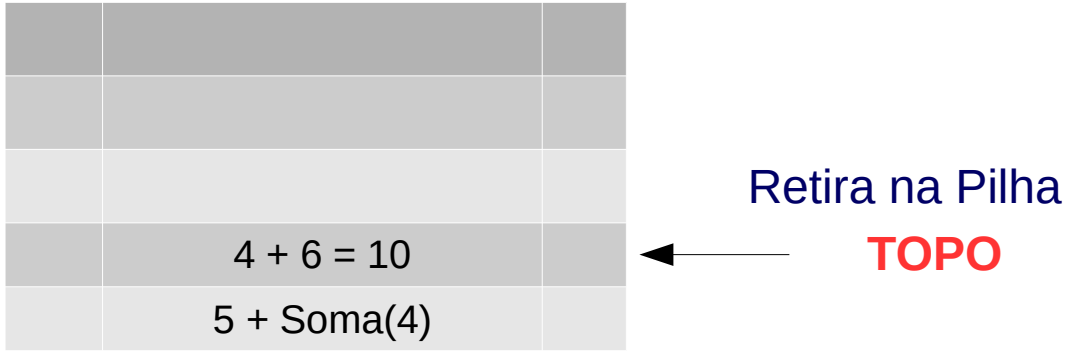
```
public static int soma(int n) {
    if (n == 1) {
        return 1;
    } else {
        return (n + soma(n - 1));
    }
}
```



Recursividade

- Exemplo: n = 5

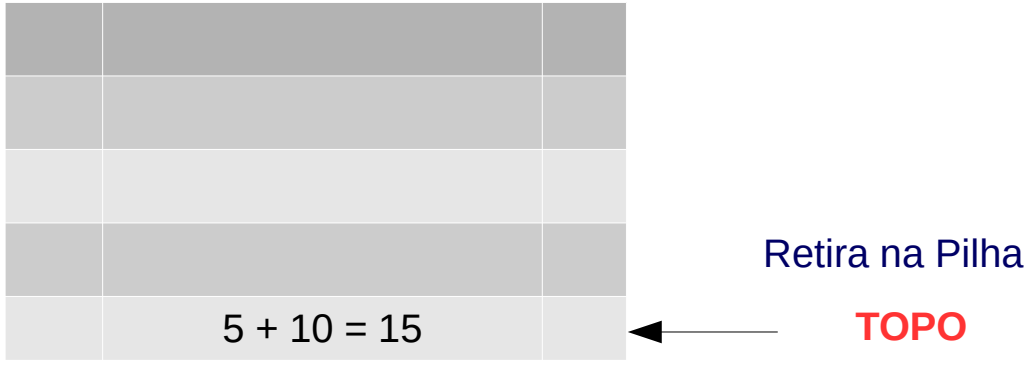
```
public static int soma(int n) {
    if (n == 1) {
        return 1;
    } else {
        return (n + soma(n - 1));
    }
}
```



Recursividade

- Exemplo: n = 5

```
public static int soma(int n) {
    if (n == 1) {
        return 1;
    } else {
        return (n + soma(n - 1));
    }
}
```



Recursividade

- Recursividade gera um grande consumo de memória, e por tal motivo, a pilha criada possui um limite de elementos que pode ser armazenado.
- Em Java, caso esse limite seja alcançado, uma exceção é lançada:
  - `java.lang.StackOverflowError`
- Isso é ocasionado normalmente por causa de uma recursividade infinita.

Recursividade

- Dada uma função recursiva, sempre é possível escrever uma função equivalente, sem recursão.
- Para se eliminar a recursão, na maioria das vezes é necessário recorrer ao uso de uma pilha.

Eliminação da Recursão
Ideia Geral

- Cada chamada recursiva é substituida por comandos para
  - empilhar o valor das variáveis locais e parâmetros
  - alterar os valores das variáveis locais e parâmetros
  - voltar ao início da função (normalmente, por loop)
- O encerramento de uma chamada (retorno) é substituído por comandos para
  - desempilhar os valores dos parâmetros e variáveis locais.
  - continuar a execução a partir do ponto onde seria o retorno da chamada recursiva correspondente (esse ‘ponto’ pode ser representado por um valor adicional na pilha).

Recursividade

- Deve-se utilizar recursividade quando:
  - O problema é naturalmente recursivo e a versão recursiva do algoritmo não gera ineficiência evidente;
  - O algoritmo se torna compacto, sem perda de clareza;
  - É possível prever que o número de chamadas não vai provocar interrupção no processo.

Recursividade

- Não deve-se utilizar recursão quando:
  - A solução recursiva causa ineficiência;
  - O uso de recursão acarreta número maior de cálculos que a versão iterativa;
  - Quando parâmetros consideravelmente grandes têm que ser passados por valor;
  - Não é possível prever o número de chamadas que podem causar sobrecarga da pilha.

Dúvidas

Exercícios

Exercício

2) E como ficaria o programa de contagem regressiva apresentado no primeiro slide caso ele fosse implementado recursivamente?

Resposta Exercício 02

```
public class POTAAula02Exercicio02 {

    public static void contagemRegressiva(int n) {
        if (n == 0) {
            System.out.println("Feliz Ano novo!");
        } else {
            System.out.println(n);
            contagemRegressiva(n - 1);
        }
    }

    public static void main(String[] args) {
        contagemRegressiva(10);
    }

}
```

Exercício

3) Faça um programa recursivo que verifique se uma palavra é palindromo.

– Exemplo: ovo, ana, abba, 010010

Obrigado

bruno.moritani@anhembi.br