1. Definição e tipos

Dizemos que um método é recursivo quando este faz uma chamada a ele mesmo, seja direta ou indiretamente.

Tipos de métodos recursivos:

- Método recursivo direto

```
void metodoA(){
    if(CONDICAO){
        //código
    else{
        metodoA(); //chamada recursiva direta
    }
}
```

- Método recursivo indireto

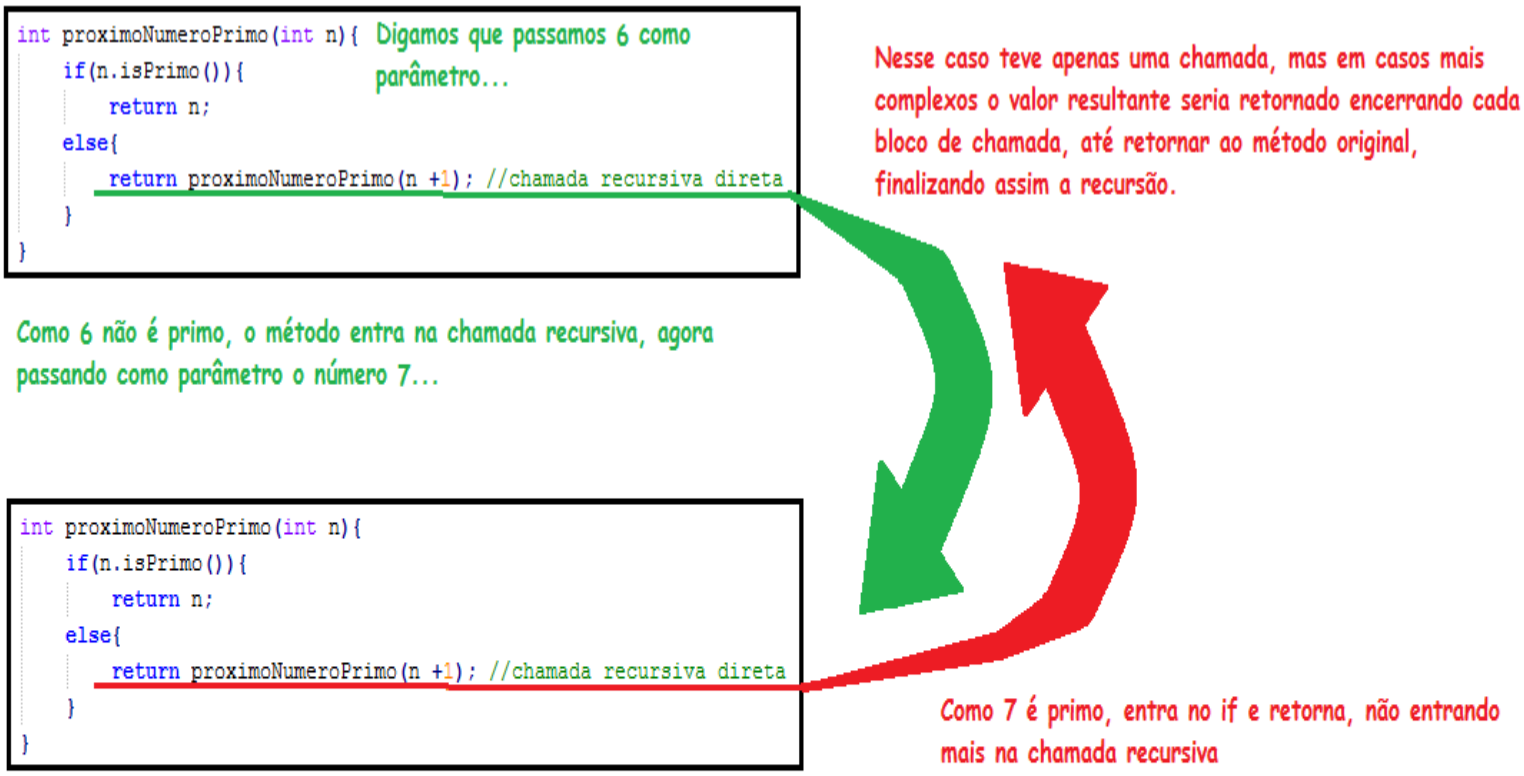
```
void metodoB(){
    if(CONDICAO){
        //código
    else{
        metodoC();
    }
}

void metodoC(){ //metodo chamado pelo metodoB
    if(CONDICAO){
        //código
    else{
        metodoB(); //chamada recursiva indireta
    }
}
```

ATENÇÃO!

Deve-se ter cuidado com métodos recursivos, pois uma condição de parada mal feita pode resultar em recursão infinita, lançando a exceção *StackOverflowError*, todo método recursivo tem que ter uma condição de parada bem definida e/ou um *return*.

2. Funcionamento

Para entender como funciona uma chamada recursiva, de forma bem abstrata podemos representá-la da seguinte forma:



Observações:

- Quando um método (método1) entra em uma chamada recursiva, vale a mesma regra de como se tivesse chamado outro método qualquer, a execução do método (método1) pára até que todas as chamadas seguintes se encerrem, só então continuará com a execução inicial.
- Deve-se sempre lembrar da lógica de como funciona um algoritmo recursivo, NUNCA esquecendo suas condições de parada (chamadas de *caso base)*, lembrando que, se necessário, um método pode ter mais de uma condição de parada, ou seja, mais de um caso base.
- Uma recursão tende sempre a resolver o problema o diminuindo e/ou dividindo-o em partes menores, até que se chegue no caso caso, parando a recursão. Tente sempre seguir essa lógica para evitar erros.

3. Exercícios

1) Escreva um método recursivo que gere (imprimindo na tela) a sequência dos números naturais passando como parâmetro o último número (final da sequência).

RESOLUÇÃO:

```java
/**
 * @author Pedro Victor
 */
public class GeraSequenciaNaturais {
    public static void main(String[] args) {
        geraSequencia(10);
    }

    private static void geraSequencia(int n) {
        if (n < 0) {
            System.out.println("Apenas para numeros naturais");
        } else
            geraSequenciaRecusive(n);
    }

    private static void geraSequenciaRecusive(int n) {
        if (n == 0) {
            System.out.print(n + " ");
        } else {
            geraSequenciaRecusive(n - 1);
            System.out.print(n + " ");
        }
    }
}
```

Saída: 0 1 2 3 4 5 6 7 8 9 10

2) Implemente um método recursivo que recebe um número inteiro positivo N e calcula o somatório dos números de 1 a N (inclusive).

RESOLUÇÃO:

```java
if (num == 0)
    return 0;
else
    return num + somatorio(--num);
```

OBS.: o código abaixo usa o operador ternário, funciona da seguinte forma:

(CONDIÇÃO) ? PRIMEIRO_RETORNO : SEGUNDO_RETORNO

Ou seja, equivale à:

```
public int somatorio(int num) {
    return (num == 0) ? 0 : num + somatorio(--num);
}
```

PORÉM, o uso do operador ternário não é aconselhável por questões de legibilidade e manutenção, já que é um pouco mais difícil de ler em comparação com a versão normal e estendida.

3) Execute as linhas de código a seguir, verifique a saída e explique o por quê da diferença.

```
public class GeraSequenciaTeste {
    public static void main(String[] args) {
        geraSequencia1(10);
        System.out.println();
        geraSequencia2(10);
    }

    private static void geraSequencia1(int n) {
        if (n == 0) {
            System.out.print(n + " ");
        } else {
            System.out.print(n + " ");
            geraSequencia1(n - 1);
        }
    }

    private static void geraSequencia2(int n) {
        if (n == 0) {
            System.out.print(n + " ");
        } else {
            geraSequencia2(n - 1);
            System.out.print(n + " ");
        }
    }
}
```

4) Escreva um método recursivo que, dado um inteiro, calcule seu valor fatorial.
Tabela com os valores até o número 12! para conferir.

```
0! = 1
1! = 1
2! = 2
3! = 6
4! = 24
5! = 120
6! = 720
7! = 5040
8! = 40320
9! = 362880
10! = 3628800
11! = 39916800
12! = 479001600
```

5) Escreva um método recursivo que gere a série de Fibonacci e explique sua execução.
OBS.: série/sequência de Fibonacci tem como primeiros termos os números 0 e 1 e, a seguir, cada termo subsequente é obtido pela soma dos dois termos predecessores:
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, …

6) Crie uma uma classe CounterElements. Essa classe deve conter um método CountNotNullElements que recebe como parâmetro um array de Generics(T[] array) e seu tipo de retorno deve ser um int. Dado esse array passado como parâmetro, implemente o corpo do método, **utilizando recursão**, de forma que ele conte a quantidade de elementos não-nulos (diferentes de null) nesse mesmo array.
Ex: dado o array [1,2,3,null,null], o método deve retornar 3.

7) Implemente um método que calcule a potência de um número dado seu expoente. A base é fixa, e será 2, ou seja, o programa deve calcular 2^x.
Exemplo: ao passar *4* como parâmetro o método deve retornar *16*.