Explicações básicas sobre Recursividade

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

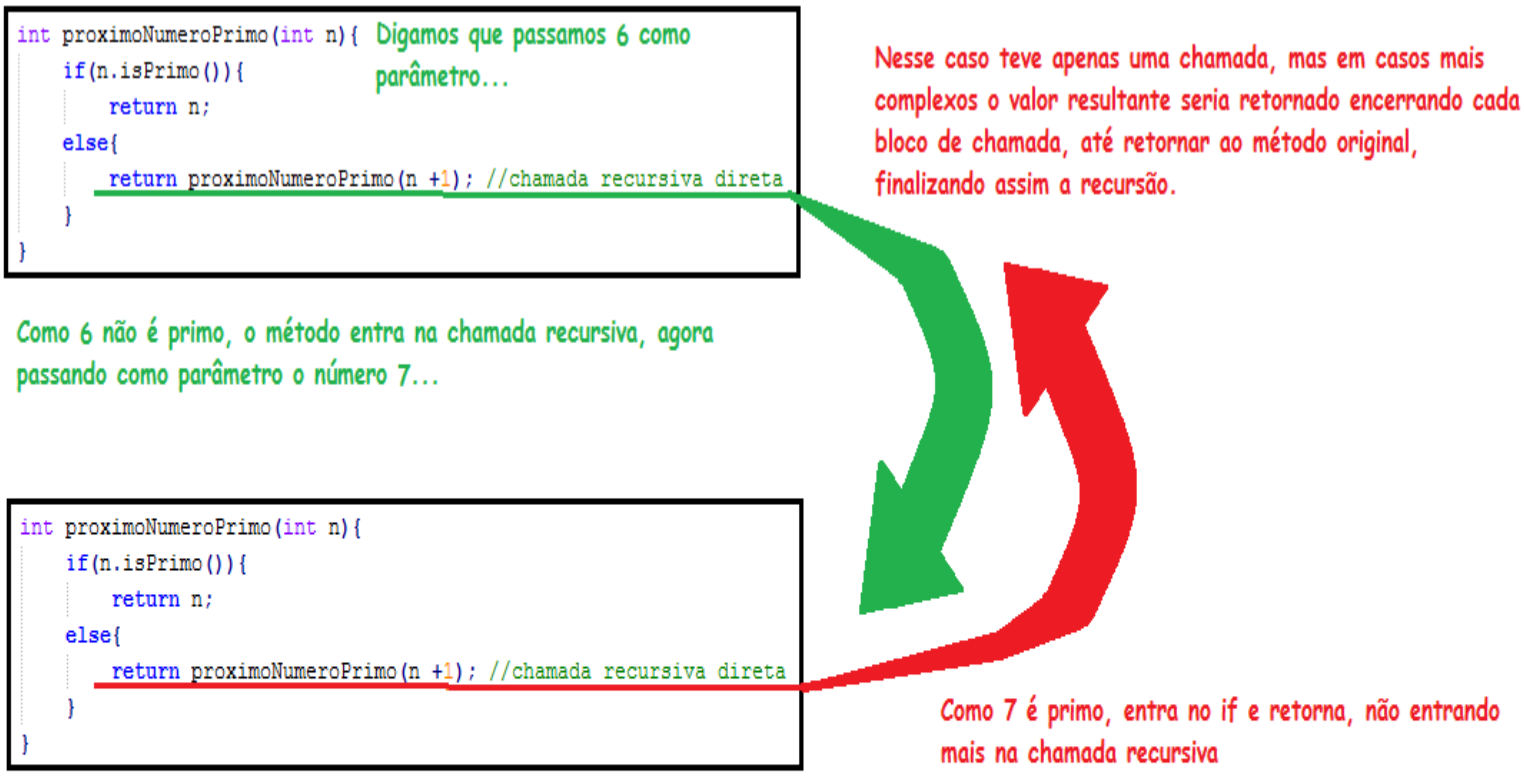
```
void metodoB(){
    if(CONDICAO){
        //código
    else{
        metodoC();
    }
}

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 resolvidos

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:

```
/**
 * @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:

```
public static int somatorioRecursivo(int n){
    int soma = 0;
    if (n == 0){
        soma = 0;
    }else{
        soma = n + somatorioRecursivo(n-1);
    }

    return soma;
}
```

OBS: Note que este mesmo código sem recursão seria o seguinte:

```java
public static int somatorioIterativo(int n){
    int soma = 0;

    for (int i = 0; i <= n; i++){
        soma = soma + i;
    }

    return soma;
}
```

3) Observe a classe recursao.GeradorSequencia.java. Execute a classe e tente compreender o porquê da diferença entre os métodos recursivos.

Exercícios a resolver

1) O fatorial de um número N é descrito como o produtório de 1 até N. Entretanto, o fatorial também pode ser definido de forma recursiva, onde fatorial de 0 é, por definição 1 e o fatorial de qualquer outro número positivo é o próprio número multiplicado pelo fatorial do número anterior. A tabela abaixo mostra a saída de uma execução (parcial) do fatorial.

```
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
```

Observe a classe MetodosRecursivos.java e implemente o código que calcula o fatorial e imprime conforme acima (dentro do método **calcularFatorial**)

2) A série/sequência de Fibonacci tem como primeiros termos os números 1 e 1 e, a seguir, cada termo subsequente é obtido pela soma dos dois termos predecessores:
1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, …

Observe a classe MetodosRecursivos.java e implemente o código que calcula o n-esimo termo da sequencia de Fibonacci e o imprime acima (dentro do método **calcularFibonacci**)

3) Imagine um método que é capaz de varrer um array de objetos e contar a quantidade de objetos não nulos. Essa contagem pode ser feita utilizando-se recursão.
Ex: dado o array [1,2,null,null,5], o método deve retornar 3.

Observe a classe MetodosRecursivos.java e implemente o código que conta os elementos não-nulos de um array (parâmetro) (dentro do método **countNotNull**)

4) O Cálculo da potência de 2 pode ser feita de forma recursiva. Para isso, basta apenas fixar a base (em 2) e receber o expoente como parâmetro, retornando 2^x.
Exemplo: ao passar *4* como parâmetro o método deve retornar *16*.

Observe a classe MetodosRecursivos.java e implemente o código que conta retorna a n-ésima potencia de 2 de um expoente passado como parâmetro (dentro do método **potenciaDe2**)

5) Uma progressão aritmética (PA) é uma sequencia que possui a seguinte lei de formação:

$$a_n = a_{n-1} + r$$

onde, a_1 é o primeiro termo (termo inicial) e *r* é a razão da progressão aritmética. Apesar de existirem fórmulas que calculam o n-ésimo termo de uma PA, a geração de qualquer termo pode ser feita de forma genérica usando-se recursão. Implemente o método progressãoAritmetica() na classe recursao.MetodosRecursivos.java com este fim.

6) Uma progressão geométrica (PG) é uma sequência que possui a seguinte lei de formação:

$$a_n = r.a_{n-1}$$

onde, a_1 é o primeiro termo (termo inicial) e *r* é a razão da progressão geométrica. Apesar de existirem fórmulas que calculam o n-ésimo termo de uma PG, a geração de qualquer termo pode ser feita de forma genérica usando-se recursão. Implemente o método progressaoGeometrica() na classe recursao.MetodosRecursivos.java com este fim.