1 - Conceito de Recursividade

A recursividade é uma técnica que pode ser usada para resolver problemas computacionais que possuem uma característica recursiva, ou seja, problemas com a propriedade onde

"cada instância do problema contém uma instância menor do mesmo problema".

Considere como exemplo o cálculo do fatorial:

n! = n * (n-1)!

ou seja, a instância (n-1)! é usada para resolver a instância n!.

Na computação uma função recursiva é aquela que faz uma chamada para si mesma. Considere como exemplo o método fatorial(5) na Figura 1, para resolver o problema o método fatorial será chamado recursivamente para os valores fatorial(4), fatorial(3), fatorial(2) e fatorial(1). O processamento empilha as chamadas para posteriormente resolver a instrução nro (fatorial(n-1), pois o valor da chamada fatorial(4) só estará disponível após várias chamadas do método fatorial.

A recursividade pode ser direta ou indireta:

- Direta: assim como no exemplo da Figura 1, ou seja, uma função *A* chama ela própria;
- Indireta: uma função *A* chamada um função *B* que, por sua vez, chama *A*.

```
package aula;

public class Principal {
   public static void main(String[] args) {
      System.out.println( fatorial(5) );
   }

   public static long fatorial(int nro){
      if( nro <=  1 ){
         return 1;
      }
      else{
         return nro * fatorial(nro-1);
      }
   }
}
```

Figura 1 – Código da classe Principal.

2 - Exercícios

1 – Criar um método que calcula a sequência de Fibonacci de forma recursiva, ou seja, não pode usar estrutura de repetição.

2 – Criar um método que calcula a potência de forma recursiva, ou seja, não pode usar estrutura de repetição.

3 – Criar um método que calcula o somatório dos elementos de um array de forma recursiva, ou seja, não pode usar estrutura de repetição.

4 – Criar um método que obtém o maior valor de um array de forma recursiva, ou seja, não pode usar estrutura de repetição.