# Aula 7 — Modularização

Processamento da Informação

Universidade Federal do ABC

# Funções (= *métodos* em Java)

- Já usamos algumas funções sem nos darmos conta:
  - System.out.println
  - Math.random, Math.sqrt, ...
  - scanner.nextInt(), nextFloat(), ...
  - ...
- Há algumas rotinas de código que podem ser utilizadas repetidamente e em diversos contextos diferentes. Frequentemente é conveniente as **encapsular** em funções.
- Funções/Métodos servem para reunir o código responsável por desempenhar um papel e o deixar disponível através de um nome coerente (ex. seno, coseno, raiz quadrada, ....).
- Isso nos foi conveniente até agora para nos importarmos apenas com **o que** elas fazem e não **como** elas fazem (pense em como implementaria Math.random, por exemplo).

# Funções

- Funções, assim como em matemática, podem receber **parâmetros** (ocasionalmente também chamados de **argumentos**)
- Considere a função $f(a, b) = a^b$. Ela recebe dois parâmetros, **a** e **b** e **devolve** o valor de **a** elevado a **b**.
  - A função equivalente já existe em Java e é chamada Math.pow.

```
...
double a = scanner.nextDouble();
double b = scanner.nextDouble();
double pot = Math.pow(a, b);
...
```

# Funções

- Considere a fórmula de Heron para o cálculo da área do triângulo dados os seus três lados, a, b e c.

```
public static double area(double a, double b, double c) {
    double s = (a + b + c) / 2;
    double area = Math.sqrt(s * (s-a) * (s-b) * (s-c));
    return area;
}
```

- A primeira linha é chamada de **assinatura** da função. Ela define o tipo da função (double), e os parâmetros (seus tipos e seus nomes).
- **return** devolve o valor calculado pela função. O tipo do valor devolvido **deve** ser igual ao tipo da função

# Funções - Comentários

- Funções podem chamar (e frequentemente o fazem) outras funções.
- Se você está copiando e colando o mesmo código frequentemente e com apenas pequenas adaptações, considere encapsulá-lo em uma função.
- Ao contrário de funções matemáticas, métodos em Java podem não devolver valor algum. Neste caso esses métodos executam algum procedimento que é util para o utilizador. Ex.: println.

# O tipo **void**

- O tipo **void** é um tipo especial.
- Ele representa "nada", ou seja, uma variável deste tipo não faz sentido. Contudo é útil para indicar o tipo de uma função que não devolve valor algum. Por exemplo, a função abaixo:

```
public static void imprime (int numero) {
    System.out.printf("Número %d\n", numero);
}
```

- A função **main** é do tipo **void**

# Ex. 1 - Fatorial

- Escreva uma função que computa o fatorial de um número *n* passado por parâmetro. Sua função deve ter a seguinte assinatura: **public static long fat(long n)**. OBS: Caso n $\leq 0$ seu programa deve retornar 1.
- Use a função anterior e crie um programa que imprima os valores de n! para n $= 1, \ldots, 20$.

# Ex.2 - Parte 1 - Verificação de primalidade

Encapsule o código de verificação de primalidade abaixo em uma funçao com assinatura: **public static boolean ePrimo (int n)** e altere o programa para usar a função mantendo o seu comportamento.

```
import java.util.Scanner;
public class Main {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    int n = sc.nextInt();
    int limite = (int)Math.sqrt(n);
    boolean eh_primo = n > 1 && (n == 2 || n % 2 != 0);
    for (int i = 3; i <= limite && eh_primo; i += 2)
        eh_primo = (n % i) != 0;
    System.out.println(eh_primo ? "Primo" : "Nao primo");
  }
}
```

# Ex. 2 - Parte 2 - Números primos gêmeos

Números primos gêmeos são pares de números primos $(p_1, p_2)$ tais que $p_2 = p_1 + 2$.

- Usando a função desenvolvida na parte 1, escreva um programa que imprime os $n$ primeiros pares de números primos gêmeos.
  - Entrada: $n$, o número de pares a serem impressos
  - Saída: lista com os $n$ primeiros pares de números primos gêmeos

| Entrada | Saída |
|---|---|
| 2 | (3, 5), (5, 7) |
| 5 | (3, 5), (5, 7), (11, 13), (17, 19), (29, 31) |

# Lista (submissão via TIDIA)

- Esta lista de exercícios deve ser submetida via TIDIA em um único arquivo JAVA contendo as implementações de cada uma das funções pedidas

1. Implemente uma função que retorna se dois números do tipo *double* são iguais de acordo com uma tolerância

```
...
public static boolean igual(double x, double y, double tol){
    // implementação
}
...
```

| Entrada | Saída |
|---|---|
| 2.1 2.2 0.01 | false |
| 0.3 0.33 0.1 | true |

# Lista (submissão via TIDIA)

2. Implemente uma função que retorna o número de vogais em uma string (que só tem letras sem acento e espaço)

- Dica: use `String.toLowerCase()` para garantir que todas as letras estejam minúsculas

| Entrada | Saída |
|---|---|
| TestE | 2 |
| tsktsk | 0 |

# Lista (submissão via TIDIA)

3. Implemente uma função que retorna o número de palavras em uma string (que só tem letras sem acento e espaço)

| Entrada | Saída |
|---|---|
| Wow | 1 |
| Deu certo | 2 |

# Lista (submissão via TIDIA)

4. Implemente uma função que retorna o número de consoantes em uma string (que só tem letras sem acento e espaço) usando as funções 2 e 3

| Entrada | Saída |
|---|---|
| Wow | 2 |
| Deu certo | 4 |

# Lista (submissão via TIDIA)

5. Implemente uma função que retorna se um ponto está "dentro" de um círculo (dado centro e raio)

```
...
public static boolean estaDentro(double x, double y,
                                 double cx, double cy,
                                 double r){
    // implementação
}
...
```

| Entrada | Saída |
|---|---|
| 1 1 2 2 5 | true |
| 1 1 0 0 0.2 | false |

# Lista (submissão via TIDIA)

6. Implemente uma função que retorna a mediana de 3 elementos

| Entrada | Saída |
|---|---|
| 1 2 3 | 2 |
| 10 1 9 | 9 |

# Lista (submissão via TIDIA)

7. Implemente uma função que retorna o número em decimal correspondente a um número binário armazenado em uma string

| Entrada | Saída |
|---|---|
| 111 | 7 |
| 1010 | 10 |

# Lista (submissão via TIDIA)

8. Implemente uma função que converte uma placa de carro em um número inteiro não negativo a partir de 0
   - Uma placa tem formato LLLNNNN em que cada L corresponde a uma letra e cada N corresponde a um dígito (0,1,...,9)

| Entrada | Saída |
|---|---|
| AAA0000 | 0 |
| BAA0000 | 6760000 |