# Programação Orientada à Objetos

## Aula 04
## Funções e Arrays

Henrique Poyatos
henrique.poyatos@bandtec.com.br

# Funções / Métodos
## O que são?

- Métodos são procedimentos de classe, onde estão implementados os processos disponíveis a objetos (instâncias) da classe.
- Os métodos permitem que objetos de uma classe realizem tratamento de dados (normalmente atributos da própria classe), cálculos, comunicação com outros objetos e todo procedimento necessário às ações das instâncias da classe.
- Em C#, *métodos* são funções de classes. Os programas vistos até aqui no curso tinham a função M*ain()* como método das respectivas classes públicas que deram nome aos nossos programas.

# Funções
## O Retorno

```
public int metodo()
{
        [COMANDOS]
        …
        return [VARIÁVEL DO TIPO INTEIRO]
}

public string metodo()
{
        [COMANDOS]
        …
        return [VARIÁVEL DO TIPO STRING]
}

public void metodo()
{
        [COMANDOS]
        …
        // Aqui não existe cláusula return, pois não há nada
        a ser retornado.
}
```

# Métodos
## Parâmetros - Entrada

```
public void metodo(int a)
//entrando um valor inteiro,
{
        será conhecido aqui dentro pela variável a;
        [COMANDOS]
}

public void metodo(int a, int b)
//entrando dois valores inteiros,
{
        serão conhecidos aqui dentro pelas
       [COMANDOS]
        variáveis a e b, respectivamente.
}

public void metodo() //Aqui não existem parâmetros
{
        [COMANDOS]
}
```

# Métodos
## Classe Math

- Os métodos da classe `Math` permitem realizar cálculos comuns necessários em expressões matemáticas.
- Exemplos de chamadas de métodos da classe `Math`:
  - ***Função raiz quadrada:*** `double y = Math.sqrt(10.0);`
  - ***Função mínimo.*** `double z = Math.min(x,10);`
- Os métodos da classe `Math` são **métodos estáticos**, ou seja, não necessitam de objetos da classe para sua chamada. Por essa razão você deve precer as chamadas dos métodos com o nome da classe seguido de ponto (como já fizemos nos programas anteriores):
  - `JOptionPane.showMessageDialog(...)`
  - `System.exit( 0 );`

# Métodos
## Classe Math

| Operador | Descrição | Exemplo |
|---|---|---|
| + | Adição | 5 + 6 = 11 |
| - | Subtração | 5 – 2 = 3 |
| * | Multiplicação | 6 * 3 = 18 |
| / | Divisão | 10 / 2 = 5 |
| % | Resto da Divisão | 10 % 3 = 1 |
| Math.Pow(x, y) | X elevado à potência y | Math.Pow(2, 4) = 16 |
| Math.Sqrt(x) | Raiz quadrada de x | Math.Sqrt(16) = 4 |

# Métodos
## Classe Math

| Operador | Descrição | Exemplo |
|---|---|---|
| **Math.Pow(x, y)** | X elevado à potência y | Math.Pow(2, 4) = 16 |
| **Math.Sqrt(x)** | Raiz quadrada de x | Math.Sqrt(16) = 4 |
| **Math.Abs(x)** | Valor absoluto | Math.Abs(-10) = 10 |
| **res = Math(x, y, out di** | Resto e divisão | resto = Math(15, 6, out divid)<br>resto = 3 e divid = 2 |
| **Math.Round(x)** | Arredondamento | Math.Round(5.111) = 5<br>Math.Round(5.999) = 6 |
| **Math.Round(x, dec)** | Arredondamento com casas decimais | Math.Round(5.344, 2) = 5.34<br>Math.Round(5.346, 2) = 5.35 |
| **Math.Truncate(x)** | Truncamento | Math.Truncate(5.111) = 5<br>Math.Truncate(5.999) = 5 |

# Arrays
## Vetores

Um array é um conjunto de itens de dados de um mesmo tipo, acessado através de um índice numérico.

**Sintaxe:**

```
tipo[] identificador = new tipo[tamanho];
```

**Exemplo:**

```
int[] numerosInteiros = new int[4];
```

Para atribuir valores às posições do array, utilizaremos a seguinte sintaxe.

**Sintaxe:**

```
identificador[posição] = valor;
```

**Exemplo:**

```
int[0] = 95;    //primeira posição do array
int[1] = 36;     //segunda posição do array
int[2] = 48;    //terceira posição do array
```

# Arrays
## Vetores

A inicialização pode ser feita junto à sua declaração. Desta forma não é necessário especificar o tamanho do array, uma vez que os próprios itens inicializados já determinam o seu tamanho.

**Sintaxe:**

```
tipo[] identificador = new tipo[] { valor1, valor2, valor3, ... };
```

**Exemplo:**

```
string[] nomes = new string[] { "Pedro", "João", "Zé", "Maria" };

//Imprimindo o terceiro nome
Console.WriteLine("Nome da terceira posição: " + nomes[2]);
Console.ReadKey();
```

# Arrays
## Laço Específico - foreach()

Estrutura de controle do fluxo de execução que permite repetir um número definido de vezes, a partir de um array **limites fixos**.

**Sintaxe:**

```
foreach (tipo identificador in array)
{
    instrução 1;
    instrução 2;
    ...
    instrução N;
}
```

**Exemplo:**

```
string[] nomes = new string[] { "Pedro", "João", "Zé", "Maria" };

foreach (string item in nomes)
{
    Console.WriteLine("Nome: " + item);
}
Console.ReadKey();
```

# Arrays
## Tamanho – .Length & .Count()

Existem duas formas de obter o tamanho de um array.

**Exemplo:**

```
string[] nomes = new string[] { “Pedro”, “João”, “Zé”, “Maria” };

Console.WriteLine(“Tamanho: “ + nomes.Length);
```

**Exemplo:**

```
string[] nomes = new string[] { “Pedro”, “João”, “Zé”, “Maria” };

Console.WriteLine(“Tamanho: “ + nomes.Count());
```

# Arrays
## Array.Reverse()

Operação para inverter os itens em um array

**Exemplo:**

```
string[] nomes = new string[] { "Pedro", "João", "Zé", "Maria" };

Array.Reverse(nomes);

foreach (string item in nomes)
{
    Console.WriteLine("Nome: " + item);
}
Console.ReadKey();
```

# Arrays
## Array.Sort()

Operação para ordenar os itens em um array

**Exemplo:**

```
string[] nomes = new string[] { "Pedro", "João", "Zé", "Maria" };

Array.Sort(nomes);

foreach (string item in nomes)
{
    Console.WriteLine("Nome: " + item);
}
Console.ReadKey();
```

# Arrays
## Array.IndexOf()

Operação para ordenar os itens em um array

**Exemplo:**

```
string[] nomes = new string[] { "Pedro", "João", "Zé", "Maria" };

int posicao = Array.IndexOf(nomes, "Zé");

if (posicao != -1)
{
    Console.WriteLine("O Zé foi encontrado na posição: " + posicao);
}
else
{
    Console.WriteLine("O Zé não foi encontrado!");
}
Console.ReadKey();
```