



**TECNOLOGIA E INOVAÇÃO  
EM PROL DA INDÚSTRIA**



## Curso técnico em Informática

# Desenvolvimento de Sistemas II

## 180h

Prof<sup>a</sup>: Francisleide Almeida



# Vetores

- Os vetores são arrays unidimensionais, eles guardam elementos de um mesmo tipo

```
class Program
{
    static void Main(string[] args)
    {
        int[] vet = new int[10];
        for (int i = 0; i < 10; i++)
            vet[i] = i;
        for (int i = 0; i < vet.Length; i++)
            Console.WriteLine(vet[i]);
    }
}
```

# Ordenar Vetor

- Simples...

```
class Program
{
    static void Main(string[] args)
    {
        int[] vet = new int[4];
        for (int i = 4, j = 0; i > 0; i--, j++)
            vet[j] = i;
        Array.Sort(vet); // ordena o array
        for (int i = 0; i < vet.Length; i++)
            Console.WriteLine(vet[i]);
    }
}
```

# Matrizes

- Declarando matrizes

```
class Program
{
    static void Main(string[] args)
    {
        int[,] mat = new int[2, 2];
        mat[0, 0] = 1;
        mat[0, 1] = 2;
        mat[1, 0] = 3;
        mat[1, 1] = 4;
        for (int i = 0; i < 2; i++)
            for (int j = 0; j < 2; j++)
                Console.WriteLine(mat[i, j]);
    }
}
```

# GetValue

- Pode-se pegar os elementos de um array utilizando o método

```
class Program
{
    static void Main(string[] args)
    {
        int[] vet = { 1, 2, 3, 4 };
        for (int i = 0; i < vet.Length; i++)
            Console.WriteLine(vet.GetValue(i));
    }
}
```

# Enum

- Enum são listas de valores constantes.

```
public enum mes { janeiro, fevereiro, dezembro };

class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine((int)mes.janeiro);
        Console.WriteLine((int)mes.fevereiro);
        Console.WriteLine((int)mes.dezembro);
    }
}
```

- Foi criada a enum “mes” que possui os valores: janeiro, fevereiro e dezembro.



# Execução



```
C:\Windows\system32\cmd.exe  
01  
1  
2  
Pressione qualquer tecla para continuar. . .
```

O valor de janeiro é 0, o valor de fevereiro é 1  
e o valor de dezembro é 2.

# Orientação a Objetos

# Criando classes

```
class Pessoa
{
    private string nome;
    private int idade;

    public Pessoa(string nome, int idade)
    {
        this.nome = nome;
        this.idade = idade;
    }
    public string getNome() { return nome; }
    public int getIdade() { return idade; }
}
```

# Criando classes

```
class Program
{
    static void Main(string[] args)
    {
        Pessoa pessoa = new Pessoa("Maria", 30);
        Console.WriteLine(pessoa.getNome());
        Console.WriteLine(pessoa.getIdade());
    }
}
```

C:\Windows\system32\cmd.exe

Maria

30

pressione qualquer tecla para continuar. . .

# Modificadores de Acesso

- Os modificadores de acesso são palavras-chave usadas para especificar a acessibilidade declarada de um membro ou de um tipo. Esta seção apresenta os quatro modificadores de acesso:
  - Public;
  - Protected;
  - Internal;
  - Private.

- Public:
  - O acesso não é restrito – pode ser acessado por qualquer outra classe;
- Protected:
  - O acesso é limitado à classe que os contém ou aos tipos derivados da classe que os contém.
- Internal:
  - o acesso é limitado ao projeto(assembly) atual.



# Modificadores de Acesso

- Protected Internal:
  - O acesso é limitado ao conjunto atual ou tipos derivados da classe que contém.
- Private:
  - O acesso é limitado ao tipo recipiente.
- Private:
  - O acesso é limitado à classe que contém ou tipos derivados da classe recipiente dentro do projeto (assembly) atual.

# Get e Set

- Declaração de propriedades:

```
namespace Project2
{
    class Pessoa
    {
        private string nome;

        public string Nome
        {
            get
            {
                return nome;
            }
            set
            {
                nome = value;
            }
        }
    }
}
```

# Get e Set

- **SIMPLIFICANDO A DECLARAÇÃO DE PROPRIEDADES COM AUTO-IMPLEMENTED PROPERTIES**
  - Para facilitar a declaração das properties, a partir do C# 3.0, temos as propriedades que são implementadas automaticamente pelo compilador, as **auto-implemented properties**

```
namespace Project2
{
    class Pessoa
    {
        public string Nome {get; set;}
    }
}
```

# Métodos

- O C# permite métodos com nomes iguais, mas parâmetros diferentes, veja:

```
class Program
{
    static void mostrarDados(string nome)
    {
        Console.WriteLine(nome);
    }

    static void mostrarDados(string nome, int idade)
    {
        Console.WriteLine(nome);
        Console.WriteLine(idade);
    }

    static void Main(string[] args)
    {
        mostrarDados("Maria");
        mostrarDados("Maria", 30);
    }
}
```

# Parâmetros opcionais

- C# permite parâmetros opcionais.  
Parâmetros opcionais são parâmetros que tem um valor pré-definido caso não seja passado um valor para ele.

```
class Program
{
    static void setNome(string nome = "Sem nome")
    {
        Console.WriteLine(nome);
    }

    public static void Main()
    {
        setNome();
    }
}
```

# Herança

- Herança é um princípio de orientação a objetos, que permite que classes compartilhem atributos e métodos. O C# não permite herança múltipla, ou seja, cada classe só pode ter um pai.



# Herança

- Classe pai, chamada Carro:

```
class Carro
{
    private int placa;

    public void setPlaca(int placa)
    {
        this.placa = placa;
    }
    public int getPlaca()
    {
        return placa;
    }
}
```

# Herança

- Classe filha, Camaro

```
class Camaro: Carro
{
    public Camaro(int placa)
    {
        setPlaca(placa);
    }
}
```

# Herança

- Execução:

```
class Program
{
    public static void Main()
    {
        Carro carro = new Camaro(1234);
        Console.WriteLine(carro.getPlaca());
    }
}
```

# Sobrescrita de métodos

- Comportamento diferentes para mesmo método:

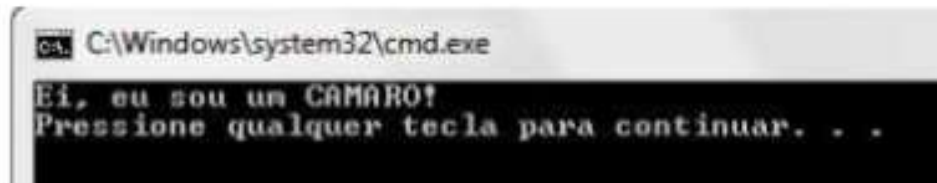
```
class Carro
{
    public virtual void showMsg()
    {
        Console.WriteLine("Ei, eu sou um carro!");
    }
}

class Camaro: Carro
{
    public override void showMsg()
    {
        Console.WriteLine("Ei, eu sou um CAMARO!");
    }
}
```

# Sobrescrita de métodos

- Execução

```
class Program
{
    public static void Main()
    {
        Carro carro = new Camaro();
        carro.showMsg();
    }
}
```



C:\Windows\system32\cmd.exe  
Ei, eu sou um CAMARO!  
Pressione qualquer tecla para continuar. . .