

---

# Programação Web

## Aulas Teóricas – Capítulo 1 – 1.2

### 1º Semestre - 2023/2024

---

*Departamento de Engenharia Informática e de Sistemas*  
*Instituto Superior de Engenharia de Coimbra/Instituto Politécnico de Coimbra*



---

**Estes Slides foram reformulados para o presente Ano Lectivo.  
No entanto para a elaboração dos mesmos foram incluídos partes de  
conteúdos utilizados nesta Unidade Curricular em Anos Lectivos  
anteriores os quais tiveram contribuições de diversos docentes,  
nomeadamente dos docentes Jorge Barbosa, Cristiana Areias,  
Francisco Leite!  
A matéria e alguns exemplos seguem as orientações existentes nos  
livros indicados na Bibliografia!**

---

---

# Programação Web

## C# – Conceitos Avançados - 2ª Parte

---

*Departamento de Engenharia Informática e de Sistemas  
Instituto Superior de Engenharia de Coimbra/Instituto Politécnico de Coimbra*



# Evolução do C#

- **C# é uma linguagem de programação em Constante Evolução**
  - Actualmente está na Versão 11
- Ao contrário de muitas outras linguagens de programação, o C# a cada nova Release apresenta bastantes inovações
  - Em muitos aspectos, a actual versão “nada tem a ver” com a versão original
  - Do mesmo modo paradigmas das linguagens “mães” do C# foram alteradas nas novas versões

# Evolução do C#

- Ela é hoje a principal linguagem de desenvolvimento da Microsoft para *desktop*, *web* e *mobile*
- Muitas das novas funcionalidades que tem incorporado rapidamente são copiadas noutras linguagens
  - Também tem incorporado muitas inovações que têm aparecido noutras linguagens

# Evolução do C#

- Esta constante evolução do C# faz com que existam novas formas de se resolver velhos problemas
- Desse modo com o C# tem de se ter consciência de que só porque algo até hoje foi feito de determinada maneira e “funcionava”, não significa que seja a melhor forma de implementar pois pode haver um modo melhor
  - Obriga a uma constante actualização de conhecimentos
  - Consegue-se com “menos fazer mais”

# Evolução do C#

- A contínua evolução dos frameworks .NET e .NET Core no sentido da sua unificação, que já é em parte uma realidade a partir do .NET 5 e a recente nova release .NET 6, LTS, (a 7 é a atual mas é STS) “obrigam” a ter-se em consideração na programação, em particular para a Web, de novos paradigmas
  - O conhecimento do C# na versão 11 (em desenvolvimento a V12 é prevista para ser lançada até ao final do corrente ano) permite acompanhar as necessárias mudanças que a unificação dos frameworks .NET e .NET Core impõe

# *top level statements* do C#

- As novas instruções do C# também são designadas por Instruções de Nível Superior - *top level statements*
  - Nota: Estas novas instruções não aparecerem necessariamente só na versão 9 mas podem ter aparecido até esta versão
- Não há necessidade de se ter um ficheiro com a designação *Program.cs*
  - Assim, o *main* pode estar num ficheiro com outro nome



# *top level statments* do C#

- Não há a necessidade de inserir o esqueleto de código – *scaffolding* - como é obrigatório em Java, C++ e em versões anteriores do C#
  - De certo modo isto permite que o C# 9 se comporte como Python, Ruby, etc
- Se por exemplo no Visual Studio 2022 (ou superior) criarmos um projecto em modo *Console*, o VS automaticamente criará um ficheiro *Program.cs* onde estará o *main* com o código mostrado de seguida

# *top level statements* do C#

```
using System;  
namespace TesteConsoleCore  
{  
    class Program  
    {  
        static void Main(string[] args)  
        {  
            Console.WriteLine("Hello World!");  
        }  
    }  
}
```

# *top level statements* do C#

- O código apresentado é o “tradicional que o VS cria automaticamente como *scaffolding* dos ficheiros C# de um projecto e estará correcto e completamente funcional e permite exhibir a tradicional mensagem do 1º programa desenvolvido numa nova linguagem que se esteja a apreender:

Hello World!

- No entanto, usando os novos recursos inerentes às *top level statements* bastaria o código que se apresenta a seguir

# *top level statements do C#*

using System;

Console.WriteLine("Hello World!");

Ou até mesmo usando somente:

System.Console.WriteLine("Hello World!");

- Experimente esta possibilidade criando no VS um projecto em modo Console selecionado o *framework .NET 5* e introduzindo uma das variantes acima e de seguida corra essa aplicação (Ctrl+F5)
- Depois de executar, experimente nas Propriedades do projecto do VS mudar o framework para o *.Net Core 3.1* (o anterior ao 5) e verificará que já não compilará, dando erro

# *top level statments* do C#

- As *top level statments*, ***tls***, dispensam então a habitual utilização de código “excessivo” e permitem por isso escrever scripts simples e concisos que podem ser utilizados em qualquer classe do *framework .NET*, retornar valores e executar código assíncrono
- Actualmente ainda existe a restrição de só se poder ter um único ficheiro de código no projeto que utilize as ***tls***

# Strings no C#

- Uma *string* ou cadeia de caracteres é uma coleção sequencial de caracteres usada para representar texto
- Nas plataformas *.NET* e *.NET Core*, o texto de uma *string* é representado como uma sequência de unidades de código UTF-16
  - Desse modo, é possível nestas plataformas armazenar em memória até 2GB (aproximadamente mil milhões de caracteres)

# Strings no C#

- **Interpolação de strings**
- No C# 6.0 foi introduzido um recurso chamado interpolação de strings.
  - Nas versões anteriores do C# a concatenação de strings era mais trabalhosa
  - Exemplo com o método *Format* da classe *String* :

```
var aluno = new Aluno();  
var mensagem = string.Format("{0} é o curso do aluno {1}", aluno.Curso,aluno.Nome);
```

# Strings no C#

- **Interpolação de strings**

- Nas versões actuais do C#, podemos utilizar a seguinte alternativa, que é mais simples

```
var aluno = new Aluno();
```

```
var mensagem = $"{aluno.Curso} é o curso do aluno {aluno.Nome}";
```

- Para usarmos a interpolação de *strings*, é necessário preceder a *string* com o caractere \$



# Strings no C#

## ▪ Caractere @

- Já referimos a utilização do @ para a construção de *strings* mais facilmente sem recorrer tanto às sequências de *escape*, usando antes *verbatim strings*

- Assim, em vez de se utilizar

```
string myFileName = "C:\\myfolder\\myfile.txt";
```

pode-se utilizar

```
string myFileName = @"C:\myfolder\myfile.txt";
```

# Strings no C#

```
using System;
namespace TesteComStrings
{
    class Program
    {
        static void Main(string[] args)
        {
            var mensagem = "Caro aluno,\n\n\tA matéria tratada nesta secção respeita à
construção de strings de um modo mais simplificado. \nAs facilidades oferecidas permitirão
a escrita de um código mais legível.\n\nUtilize estas novas características!\nBom estudo!";

            Console.WriteLine(mensagem);
        }
    }
}
```

# Strings no C#

- Pelo contrário, no exemplo seguinte a *string* é construída utilizando-se o @ para facilitar a formatação da mesma
- Observe que a formatação, espaços, mudanças de linha, etc, está incorporada na própria *string*

# Strings no C#

```
using System;
namespace TesteComStrings
{
    class Program
    {
        static void Main(string[] args)
        {
            var mensagem = @"Caro aluno,
```

A matéria tratada nesta secção respeita à construção de strings de um modo mais simplificado.

As facilidades oferecidas permitirão a escrita de um código mais legível.

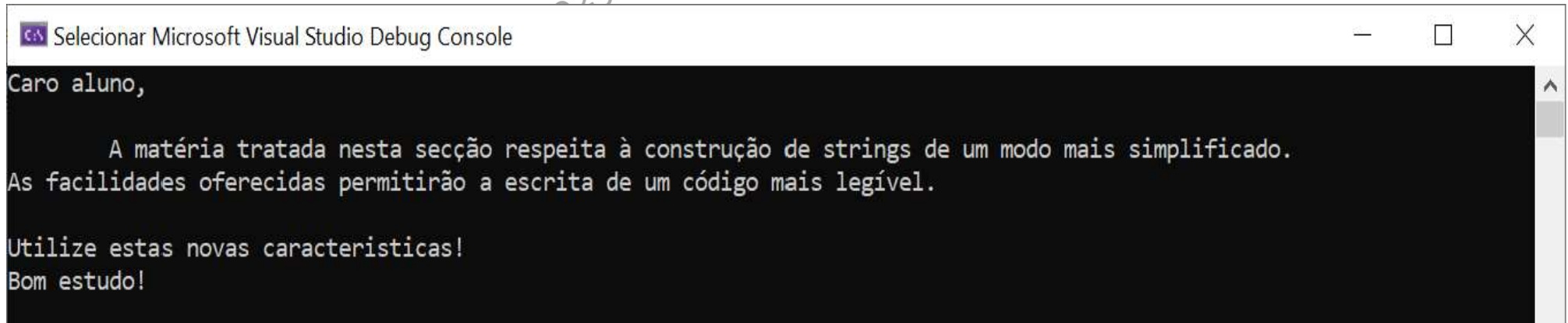
Utilize estas novas características!

Bom estudo!";

```
        Console.WriteLine(mensagem);
    }
}
```

# Strings no C#

- A saída, quer de uma versão quer da outra, será exactamente a mesma e é a mostrada na figura
- Eventualmente formatar a *string* usando o caracter @ como foi feito na 2ª versão tornará essa formatação mais fácil de ser feita e de manutenção futura também mais fácil



A screenshot of a Microsoft Visual Studio Debug Console window. The title bar reads "Selecionar Microsoft Visual Studio Debug Console". The console output is as follows:

```
Caro aluno,  
  
    A matéria tratada nesta secção respeita à construção de strings de um modo mais simplificado.  
As facilidades oferecidas permitirão a escrita de um código mais legível.  
  
Utilize estas novas características!  
Bom estudo!
```

# Strings no C#

- A utilização do @ poderá ser útil quando se criam *queries SQL*

```
var alunocursosql = @"SELECT*
```

```
FROM Aluno
```

```
WHERE curso = 'Eng.a Informática';
```

- O @ permite que essas *queries* sejam mais facilmente legíveis

# Strings no C#

- A utilização do @ também permite simplificar a passagem para um método de uma *string* de conexão de uma base de dados

```
optionsBuilder.UseSqlServer(@"Server=(localdb)\mssqllocaldb;Database=Loja;Trusted_Connection=True;");
```

# Strings no C#

- Conversão de *Strings*
- A conversão de *strings* em *arrays* de *bytes* ou em *array* de *strings* é algo habitualmente feito
  - O novo *framework* dispõe de uma panóplia de métodos específicos para isso e que facilitam bastante a programação
- Esta facilidade, é bastante útil quando seja necessário consumir métodos das classes dos *frameworks* *.NET* e *.NET Core* ou de bibliotecas de terceiros pois alguns métodos de classes destes *frameworks* obrigam a que uma *string* seja passada como um *array* de *bytes*



# Strings no C#

- Conversão de *strings* em *arrays* de *bytes*
- Para essa conversão utiliza-se o método estático *GetBytes*, presente na classe *Unicode* e na classe *UTF8* do namespace *System.Text.Encoding*

- Exemplo de conversão de *string* para *byte[]*

```
string mensagem = "Programação Web com C# e ASP.Net Core";
```

```
byte[] conteudo =
```

```
System.Text.Encoding.Unicode.GetBytes(mensagem) ;
```

# Strings no C#

- Conversão de *arrays* de *bytes* em *strings*
- Para essa conversão utiliza-se o método estático *GetString*, presente na classe *Unicode* e na classe *UTF8* do namespace *System.Text.Encoding*

```
string strfinal = Encoding.Unicode.GetString(strconvertida, 0, strconvertida.Length);
```

- Para ter uma melhor percepção de como estas conversões são feitas, nomeadamente de *array* de *bytes* para *string*, analise e execute o código exemplo que se segue

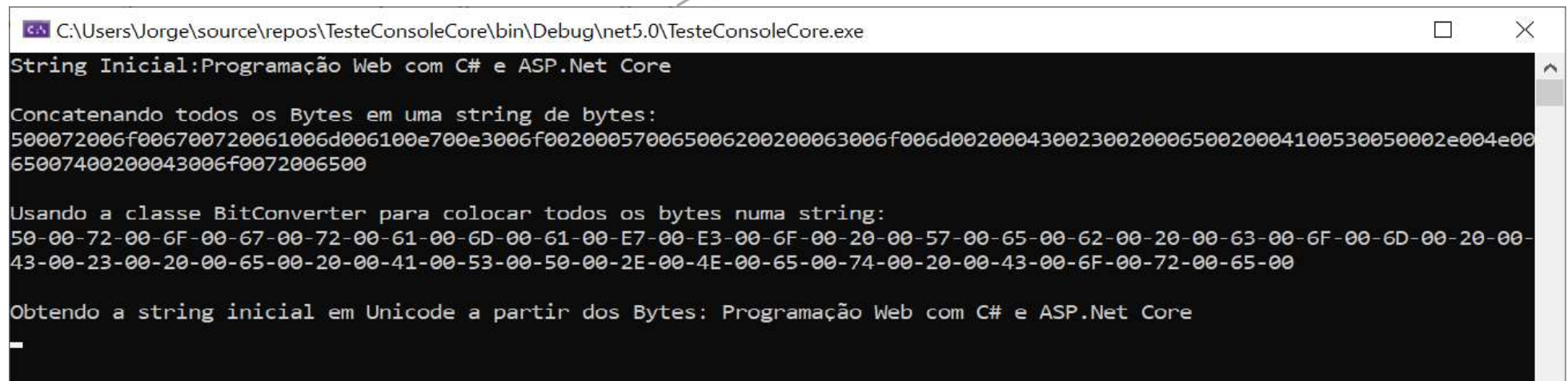
# Strings no C#

```
using System;
using System.Text;
namespace TesteComStrings
{
    class Program
    {
        static void Main(string[] args)
        {
            string strinicial = "Programação Web com C# e ASP.Net Core";
            Console.WriteLine("String Inicial:");
            Console.WriteLine(strinicial);
            byte[] strconvertida = System.Text.Encoding.Unicode.GetBytes(strinicial);
            Console.WriteLine("\nConcatenando todos os Bytes em uma string de bytes: ");
            StringBuilder builder = new StringBuilder();
            for (int i = 0; i < strconvertida.Length; i++)
            {
                builder.Append(strconvertida[i].ToString("x2"));
            }
            Console.WriteLine(builder.ToString());
            Console.WriteLine("\nUsando a classe BitConverter para colocar todos os bytes numa string:");
            string bitString = BitConverter.ToString(strconvertida);
            Console.WriteLine(bitString);
            Console.WriteLine("\nObtendo a string inicial em Unicode a partir dos Bytes: ");
            string strfinal = Encoding.Unicode.GetString(strconvertida, 0, strconvertida.Length);
            Console.WriteLine(strfinal);

            Console.ReadKey();
        }
    }
}
```

# Strings no C#

- Execute o código e interprete-o
- A saída deste código de conversão é a seguinte:



The screenshot shows a Windows command prompt window with the title bar "C:\Users\Jorge\source\repos\TesteConsoleCore\bin\Debug\net5.0\TesteConsoleCore.exe". The output of the program is as follows:

```
String Inicial:Programação Web com C# e ASP.Net Core

Concatenando todos os Bytes em uma string de bytes:
500072006f006700720061006d006100e700e3006f002000570065006200200063006f006d002000430023002000650020004100530050002e004e00
65007400200043006f0072006500

Usando a classe BitConverter para colocar todos os bytes numa string:
50-00-72-00-6F-00-67-00-72-00-61-00-6D-00-61-00-E7-00-E3-00-6F-00-20-00-57-00-65-00-62-00-20-00-63-00-6F-00-6D-00-20-00-
43-00-23-00-20-00-65-00-20-00-41-00-53-00-50-00-2E-00-4E-00-65-00-74-00-20-00-43-00-6F-00-72-00-65-00

Obtendo a string inicial em Unicode a partir dos Bytes: Programação Web com C# e ASP.Net Core
```

# Strings no C#

- Partição de uma *string* num *array* de *strings*
- Para se efectuarem partições de *strings* em *arrays* de *strings* utiliza-se o método *Split* da classe *String*
  - Utiliza-se uma lista de delimitadores parametrizada: os *split delimiters*, que podem ser um único caractere, um *array* de caracteres ou até mesmo um *array* de *strings*.

# Strings no C#

- Utilização da forma mais simples de uso do método Split:

```
using System;
namespace TeseComStrings
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Unidades Curriculares:");
            Console.WriteLine();
            //Separação entre cada UC é feita apenas por vírgulas
            string unidadesCurriculares = "Programação Web, Ética e Deontologia, Projecto ou Estágio, Introdução às Bases de
Dados";
            string[] lista = unidadesCurriculares.Split(", ");
            foreach (string uc in lista)
                Console.WriteLine(uc);

            Console.ReadKey();
        }
    }
}
```

# Strings no C#

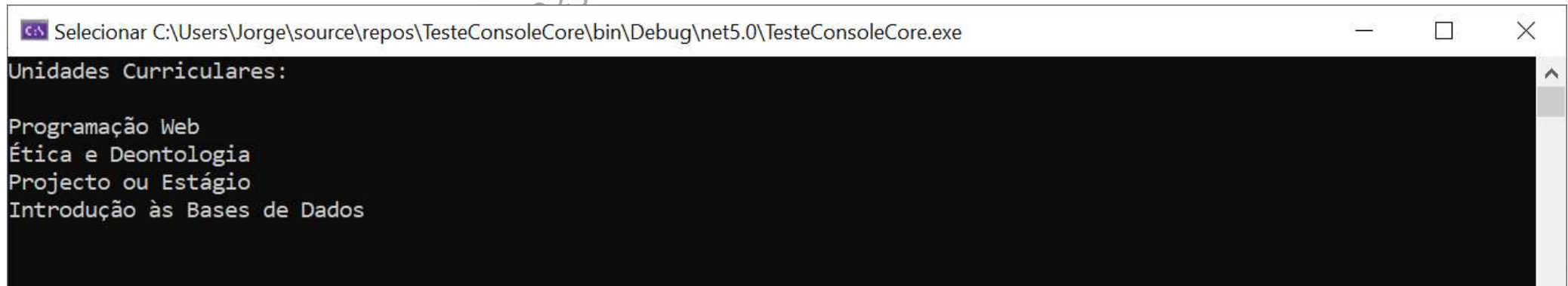
- Utilização da forma mais simples de uso do método Split:

```
using System;
namespace TeseComStrings
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Unidades Curriculares:");
            Console.WriteLine();
            //Separação entre cada UC é feita apenas por vírgulas
            string unidadesCurriculares = "Programação Web, Ética e Deontologia, Projecto ou Estágio, Introdução às Bases de
Dados";
            string[] lista = unidadesCurriculares.Split(", ");
            foreach (string uc in lista)
                Console.WriteLine(uc);

            Console.ReadKey();
        }
    }
}
```

# Strings no C#

- Execute o código e interprete-o
  - A saída deste código utilizando o método *Split* é a seguinte:



```
Selecionar C:\Users\Jorge\source\repos\TesteConsoleCore\bin\Debug\net5.0\TesteConsoleCore.exe
Unidades Curriculares:
Programação Web
Ética e Deontologia
Projecto ou Estágio
Introdução às Bases de Dados
```



# Strings no C#

- No exemplo mostrado no código seguinte aborda-se uma situação mais complexa dado que se pretende partir, com pouco código, uma *string* complexa numa lista de palavras
  - Para o efeito, utiliza-se o seguinte *array* de delimitadores:  
***string[] lista = pensamento.Split(new Char[] { ' ', ',', '.', ':', '-', '\n', '\t' });***
  - Este array vai ser como que o filtro que vai permitir “partir” as palavras separadas por cada um dos elementos constantes deste array de delimitadores

# Strings no C#

```
using System;
namespace TesteComStrings
{
    class Program
    {
        static void Main(string[] args)
        {
            string pensamento = @"Nosso cérebro é o melhor brinquedo já criado:
Nele se encontram todos os segredos, inclusive o da felicidade.
A vida é maravilhosa se você não tem medo dela.";
            Console.WriteLine("Pensamento de Charles Chaplin:");
            Console.WriteLine();
            Console.WriteLine(pensamento);
            Console.WriteLine();
            Console.WriteLine("Palavras:");
            Console.WriteLine();
            string[] lista = pensamento.Split(new Char[] { ' ', ',', '.', ':', '-', '\n', '\t' });

            foreach (string palavra in lista)
            {
                if (palavra.Trim() != "")
                    Console.WriteLine(palavra);
            }
            Console.ReadKey();
        }
    }
}
```

# Strings no C#

Selecionar C:\Users\Jorge\source\repos\TesteConsoleCore\bin\Debug\net5.0\TesteConsoleCore.exe

Pensamento de Charles Chaplin:

Nosso cérebro é o melhor brinquedo já criado:  
Nele se encontram todos os segredos, inclusive o da felicidade.  
A vida é maravilhosa se você não tem medo dela.

Palavras:

Nosso  
cérebro  
é  
o  
melhor  
brinquedo  
já  
criado  
Nele  
se  
encontram  
todos  
os  
segredos  
inclusive  
o  
da  
felicidade  
A  
vida  
é  
maravilhosa  
se  
você  
não  
tem  
medo  
dela

# Strings no C#

- **Verificar se uma *string* é nula ou vazia**
- A abordagem normal para se fazer esta verificação seria habitualmente a seguinte:

```
string nome = string.Empty;  
if (nome == null || nome == string.Empty)  
{  
    Console.WriteLine("variável nome está vazia ou contém null");  
}  
else  
{  
    Console.WriteLine("nome: {nome}");  
}
```

# Strings no C#

- Podemos realizar a mesma validação com menos código  
Caso utilizemos o método `IsNullOrEmpty` da classe `String`, podemos realizar essa validação de um modo muito simples:

```
string nome = string.Empty;  
if (string.IsNullOrEmpty(nome))  
{  
    Console.WriteLine("variável nome está vazia ou contém null");  
}  
Else  
{  
    Console.WriteLine("nome: {nome}");  
}
```

# Strings no C#

- **Outras Formas de Formatar *Strings***
- No C# existem outras formas de formatar *strings* para além da interpolação de *strings*
- Para a formatação num valor específico, existem o método estático *Format* da classe *String* e o método *ToString* da classe base *Object*

# Strings no C#

- Utilizar ***String.Format*** para separar data e hora

```
using System;
namespace TesteComStrings
{
    class Program
    {
        static void Main(string[] args)
        {
            // Formatação de data e hora
            DateTime agora = DateTime.Now;
            string dtStr = String.Format("Hoje é dia {0:d} e são {0:t}", agora);
            Console.WriteLine(dtStr);
        }
    }
}
```

# Strings no C#

- Utilizar *String.Format* para separar data e hora – V 2
- Caso esteja a usar o C# 10, em vez do código mostrado atrás, em *Program.cs* basta colocar só o seguinte:

```
DateTime agora = DateTime.Now;  
Console.WriteLine($"Hoje é dia {agora:d} e são {agora:t}");
```



# Strings no C#

- Execute o código e interprete-o
  - A saída deste código utilizando o ***String.Format*** é a seguinte:



The image shows a screenshot of the Microsoft Visual Studio Debug Console. The title bar reads "Selecionar Microsoft Visual Studio Debug Console". The console output displays the text "Hoje é dia 20/07/2021 e são 16:50".

# Strings no C#

## ■ Utilizar *String.Format* para valores monetários

```
using System;
namespace TesteComStrings
{
    class Program
    {
        static void Main(string[] args)
        {
            //Formatação de valores monetários
            decimal gasolina = 1.634m;
            string precoGasolina = String.Format("Gasolina: {0:C3}", gasolina);
            Console.WriteLine(precoGasolina);
        }
    }
}
```

# Strings no C#

- Utilizar ***String.Format*** baseado numa “máscara”

```
using System;
namespace TesteComStrings
{
    class Program
    {
        static void Main(string[] args)
        {
            //Formatação usando uma máscara
            string mascara = "0000-000";
            int codigopostal = 3030190;
            Console.WriteLine("O Código Postal da Morada do ISEC é: " +
codigopostal.ToString(mascara));
        }
    }
}
```

# Strings no C#

- Execute o código e interprete-o
  - A saída deste código utilizando o ***String.Format*** com máscara é a seguinte:



The screenshot shows a window titled "Selecionar Microsoft Visual Studio Debug Console". The output text inside the console is "O Código Postal da Morada do ISEC é: 3030-190".

# Strings no C#

- **Validação de Dados Utilizando Expressões Regulares**
- As expressões regulares são usadas como ferramentas para executar com pouco código tarefas complexas, como localizar e extrair múltiplas ocorrências de sequências de caracteres específicas dentro de um texto e validar se os dados fornecidos respeitam um formato pré-definido
  - A classe *Regex* do *namespace* *System.Text.RegularExpressions* é usada para este tipo de validações com base em expressões regulares e contém métodos para executar todo tipo de operações envolvendo expressões regulares
  - O método *IsMatch* é usado para verificar se a *string* passada como parâmetro combina com a expressão regular definida

# Strings no C#

- **Validação de Dados Utilizando Expressões Regulares**
- Uma expressão regular (ER) nem sempre é algo simples de ser construída pois pode ser complexo condensar numa única sequência de metacaracteres várias regras que compõem o padrão
- Para facilitar a construção deste tipo de expressões regulares em casos mais complexos e trabalhosos deve-se seguir uma abordagem de ir construindo a expressão aos “poucos”
  - Significa dividir a ER nas partes que a compõe e documentar cada uma delas no momento da sua criação, após efetuar uma bateria de testes

# Strings no C#

## ▪ Validação de Dados Utilizando Expressões Regulares

```
using System;
using System.Text.RegularExpressions;
namespace TesteComStrings {
    class Program {
        static void Main(string[] args) {
            string formato = @"^\d{4}\-\d{3}$";
            string strCodigoPostal = string.Empty;
            Regex regex = new Regex(formato);
            while (true) {
                Console.Write("Código Postal: ");
                strCodigoPostal = Console.ReadLine().Trim();
                if (strCodigoPostal.ToUpper() == "FIM") break;
                string resultado = regex.IsMatch(strCodigoPostal) ? "válido" : "inválido";
                Console.WriteLine($"{strCodigoPostal} é um formato de Código Postal {resultado} !\n");
            }
        }
    }
}
```

# Strings no C#

- Execute o código e interprete-o
  - A saída deste código de validação recorrendo-se a expressões regulares é a seguinte:



A screenshot of a Windows console window titled "C:\Users\Jorge\source\repos\TesteConsoleCore\bin\Debug\net5.0\TesteConsoleCore.exe". The console displays the following text:

```
Código Postal: 3030-199
3030-199 é um formato de Código Postal válido !

Código Postal: 300-199
300-199 é um formato de Código Postal inválido !

Código Postal: 2320-122
2320-122 é um formato de Código Postal válido !

Código Postal: 3030-200
3030-200 é um formato de Código Postal válido !

Código Postal:
```



# Strings no C#

- **Validação de Dados Utilizando Expressões Regulares**
- Na tabela mostram-se algumas das partes habituais em expressões regulares, nomeadamente as que foram usadas no exemplo anterior

Parte	Significado
^	Início de linha
\d	Algarismos
{n}	n é o número de ocorrências do algarismo
\.	Ponto obrigatório
\-	Hífen obrigatório
\$	Fim da Linha

# Strings no C#

- **Validação de Dados Utilizando Expressões Regulares**
- Normalmente números como o NIF, o CC, Número de Cartão de Crédito existem dígitos verificadores no final do número que também precisam ser validados para além da validação com a máscara.
- Nestes casos, é preciso combinar expressões regulares com código de validação dos dígitos verificadores.
  - Exemplifica-se a validação do NIF

# Strings no C#

## ■ Algoritmo de Validação do NIF (Ver 2013)

- O algoritmo para validação do NIF é bastante simples: é um número composto por 9 dígitos, sendo o último um dígito de controlo, é este dígito que iremos calcular para verificar se o NIF está correcto.
- Antes de procedermos ao cálculo do dígito de controlo, temos de apurar algumas condições, uma bastante importante tem a ver com o 1º dígito, este terá que ser válido, o primeiro dígito do NIF tem a ver com o tipo de entidade que o possui, que pode ser uma das seguintes:
- NIF e Número de Contribuinte – Número de Identificação Fiscal
  - 1 – pessoa singular
  - 2 – pessoa singular

# Strings no C#

- NIPC – Número de Identificação de Pessoa Colectiva
  - 5 (pessoa colectiva),
  - 6 (pessoa colectiva pública),
  - 8 (empresário em nome individual),
  - 9 (pessoa colectiva irregular ou número provisório).
- Passado com sucesso esta condição, calcula-se o dígito de controle
- Para calcular o dígito de controlo iremos multiplicar individualmente e por ordem os restantes 8 algarismos, fazendo-o da seguinte forma:
  - O 1.º dígito multiplicamos por 9, o 2.º dígito por 8, 3.º dígito por 7, 4.º dígito por 6, 5.º dígito por 5, 6.º dígito por 4, 7.º dígito por 3 e o 8.º dígito por 2.
- Iremos somar todos os resultados da operação anterior.
- Com o resultado da soma anterior iremos dividir por 11 e aproveitar o resto dessa divisão. Se o resto for 0 ou 1, o dígito de controle será 0, caso seja outro algarismo, o dígito de controle será o resultado de  $11 - \text{resto}$ .

# Strings no C#

## ■ Algoritmo de Validação do NIF (Ver 2019)

- É constituído por nove dígitos, sendo os oito primeiros sequenciais e o último um dígito de controlo.
- O NIF pode pertencer a uma de várias gamas de números, definidas pelos dígitos iniciais, com as seguintes interpretações:
  - 1 a 3: Pessoa singular, a gama 3 começou a ser atribuída em junho de 2019;[3]
  - 45: Pessoa singular. Os algarismos iniciais "45" correspondem aos cidadãos não residentes que apenas obtenham em território português rendimentos sujeitos a retenção na fonte a título definitivo;
  - 5: Pessoa colectiva obrigada a registo no Registo Nacional de Pessoas Colectivas;

# Strings no C#

## ▪ Algoritmo de Validação do NIF (Ver 2019)

- 5: Pessoa colectiva obrigada a registo no Registo Nacional de Pessoas Colectivas;[5]
- 6: Organismo da Administração Pública Central, Regional ou Local;
- 70, 74 e 75: Herança Indivisa, em que o autor da sucessão não era empresário individual, ou Herança Indivisa em que o cônjuge sobrevivente tem rendimentos comerciais;
- 71: Não residentes colectivos sujeitos a retenção na fonte a título definitivo;
- 72: Fundos de investimento;
- 77: Atribuição Oficiosa de NIF de sujeito passivo (entidades que não requerem NIF junto do RNPC);
- 78: Atribuição oficiosa a não residentes abrangidos pelo processo VAT REFUND;
- 79: Regime excepcional - Expo 98;

# Strings no C#

## ■ Algoritmo de Validação do NIF (Ver 2019)

- 8: "empresário em nome individual" (actualmente obsoleto, já não é utilizado nem é válido);
- 90 e 91: Condomínios, Sociedade Irregulares, Heranças Indivisas cujo autor da sucessão era empresário individual;
- 98: Não residentes sem estabelecimento estável;
- 99: Sociedades civis sem personalidade jurídica.
- O nono e último dígito é o dígito de controlo. É calculado utilizando o algoritmo módulo 11.
- Obter dígito de controlo
- editar
- O NIF tem 9 dígitos, sendo o último o dígito de controlo. Para ser calculado o dígito de controlo:

# Strings no C#

## ■ Algoritmo de Validação do NIF (Ver 2019)

- Multiplique o 8.º dígito por 2, o 7.º dígito por 3, o 6.º dígito por 4, o 5.º dígito por 5, o 4.º dígito por 6, o 3.º dígito por 7, o 2.º dígito por 8 e o 1.º dígito por 9;
  - Some os resultados;
  - Calcule o resto da divisão do número por 11;
  - Se o resto for 0 (zero) ou 1 (um) o dígito de controlo será 0 (zero);
  - Se for outro qualquer algarismo X, o dígito de controlo será o resultado da subtracção  $11 - X$ .
- Ver:
- [https://pt.wikipedia.org/wiki/N%C3%BAmero\\_de\\_identifica%C3%A7%C3%A3o\\_fiscal](https://pt.wikipedia.org/wiki/N%C3%BAmero_de_identifica%C3%A7%C3%A3o_fiscal)



# Strings no C# - Validar NIF V2019

## ▪ Validação do NIF com C# - V 2019

```
public static bool Nif(string nifNumber)
{
    int tamanhoNumero = 9; // Tamanho do número NIF

    string filteredNumber = Regex.Match(nifNumber, @"[0-9]+").Value; // extrair Número

    if (filteredNumber.Length != tamanhoNumero || int.Parse(filteredNumber[0].ToString()) == 0) { return false; } // Verificar Tamanho, e zero no inicio

    int calculoChecksum = 0;
    // Calcular check sum
    for (int i = 0; i < tamanhoNumero - 1; i++)
    {
        calculoChecksum += (int.Parse(filteredNumber[i].ToString()))*(tamanhoNumero - i);
    }

    int digitoVerificacao = 11-(calculoChecksum % 11);

    if (digitoVerificacao > 9) { digitoVerificacao = 0; }
    // retornar validação
    return digitoVerificacao == int.Parse(filteredNumber[tamanhoNumero - 1].ToString());
}
```

# Strings no C# - Validar NIF V2013

## ▪ Validação do NIF com Expressões Regulares + Checkdigit V 2013

```
using System;
using System.Text.RegularExpressions;
namespace TesteComStrings
{
    class Program    {
        public static bool IsValidContrib(string Contrib)
        {
            string[] s = new string[9];
            string C = null;
            int i = 0;
            long checkDigit = 0;

            s[0] = Convert.ToString(Contrib[0]);
            s[1] = Convert.ToString(Contrib[1]);
            s[2] = Convert.ToString(Contrib[2]);
            s[3] = Convert.ToString(Contrib[3]);
            s[4] = Convert.ToString(Contrib[4]);
            s[5] = Convert.ToString(Contrib[5]);
```

# Strings no C# - Validar NIF V2013

```
s[6] = Convert.ToString(Contrib[6]);
s[7] = Convert.ToString(Contrib[7]);
s[8] = Convert.ToString(Contrib[8]);
C = s[0];
if (s[0] == "1" || s[0] == "2" || s[0] == "5" || s[0] == "6" || s[0] == "9")
{
    checkDigit = Convert.ToInt32(C) * 9;

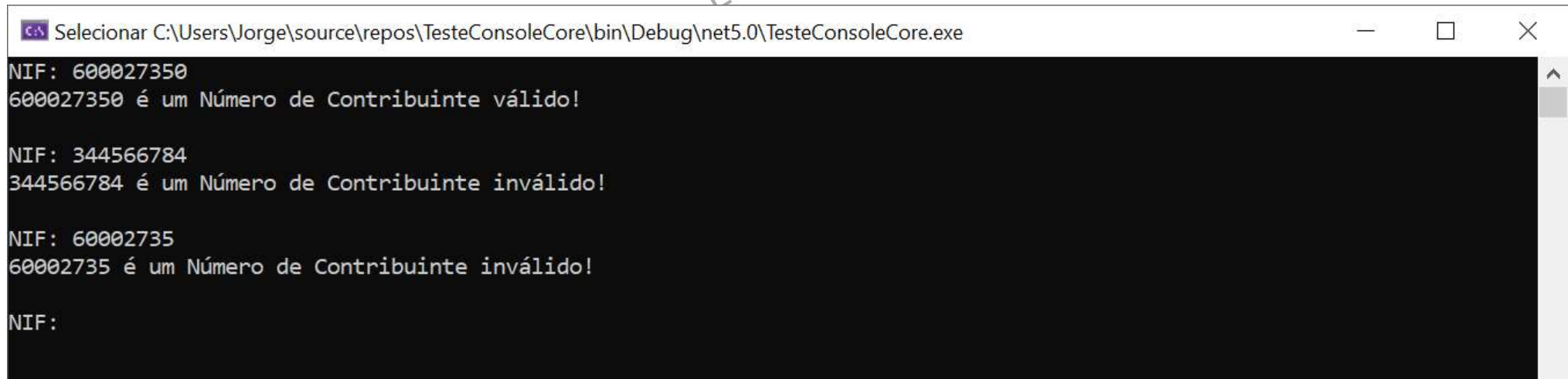
    for (i = 2; i <= 8; i++)
    {
        checkDigit = checkDigit + (Convert.ToInt32(s[i - 1]) * (10 - i));
    }
    checkDigit = 11 - (checkDigit % 11);
    if ((checkDigit >= 10))
        checkDigit = 0;
    if ((checkDigit == Convert.ToInt32(s[8])))
        return true;
}
return false;
}
```

# Strings no C# - Validar NIF V2013

```
static void Main(string[] args)
{
    string padrao = @"^\d{9}$";
    string nif = String.Empty;
    Regex regex = new Regex(padrao);
    while (true)
    {
        Console.Write("NIF: ");
        nif = Console.ReadLine().Trim();
        if (nif.ToUpper() == "FIM") break;
        string resultado = (regex.IsMatch(nif) && IsValidContrib(nif)) ? "válido" : "inválido";
        Console.WriteLine($"{nif} é um Número de Contribuinte {resultado}!\n");
    }
}
```

# Strings no C#

- Execute o código e interprete-o
  - A saída deste código de validação do NIF recorrendo-se a expressões regulares e ao Checkdigit é a seguinte:



A screenshot of a Windows command prompt window. The title bar shows the file path: "C:\Users\Jorge\source\repos\TesteConsoleCore\bin\Debug\net5.0\TesteConsoleCore.exe". The window has standard Windows window controls (minimize, maximize, close). The command prompt shows the following text:

```
NIF: 600027350
600027350 é um Número de Contribuinte válido!

NIF: 344566784
344566784 é um Número de Contribuinte inválido!

NIF: 60002735
60002735 é um Número de Contribuinte inválido!

NIF:
```

# Strings no C#

## ▪ Validação do Cartão de Cidadão:

- <https://www.autenticacao.gov.pt/documents/20126/0/Valida%C3%A7%C3%A3o+de+N%C3%BAmero+de+Documento+do+Cart%C3%A3o+de+Cidad%C3%A3o+%281%29.pdf/7d5745ba-2bcc-e861-3954-bafe9f7591a0?version=1.0&t=1658411665319&download=true>

## ▪ Validação do NIF (Ver 2019):

- [https://pt.wikipedia.org/wiki/N%C3%BAmero\\_de\\_identifica%C3%A7%C3%A3o\\_fiscal](https://pt.wikipedia.org/wiki/N%C3%BAmero_de_identifica%C3%A7%C3%A3o_fiscal)

## ▪ Validação de Cartões de Crédito:

- <https://cleilsontechinfo.netlify.app/jekyll/update/2019/12/08/um-guia-completo-para-validar-e-formatar-cartoes-de-credito.html>

# Strings no C#

## ■ Obter uma *String* aleatoriamente de uma lista de *strings*

```
using System;
namespace TesteComStrings
{
    class Program
    {
        static void Main(string[] args)
        {
            string[] alunos = { "João Marques", "Patrícia Ferreira", "Ana Maria Vicente", "Carlos Renato", "Vera Santos",
"Paulo Luz", "Jorge Miguel", "Mafalda Bernardo", "Diogo Costa", "Julia Pires" };
            Random rand = new Random();
            for (int i = 0; i < alunos.Length; i++)
            {
                int indice = rand.Next(alunos.Length);
                Console.WriteLine($"Aluno: {alunos[indice]}");
                Console.ReadKey();
            }
        }
    }
}
```

# Strings no C#

- **UTILIZANDO COMENTÁRIOS ESPECIAIS NO CÓDIGO**
- O *Visual Studio* reconhece um tipo especial de comentário que utiliza *tokens*, como TODO , HACK ou outros *tokens* criados pelo próprio programador.
- Esse tipo de comentário é usado como atalho para sinalizar que um trecho de código não está finalizado, possibilitando ao programador retornar rapidamente a esse ponto do código-fonte e dar seguimento ao trabalho.
- Para sinalizar que algo precisa ser continuado depois, basta adicionar uma linha de comentário contendo o *token* TODO

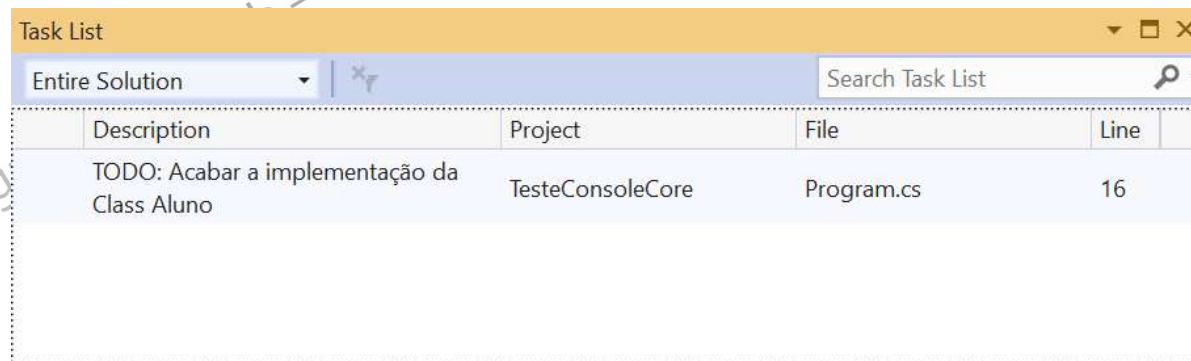


# Strings no C#

## ▪ EXEMPLO UTILIZANDO O TODO

//TODO: Acabar a implementação da Class Aluno

- Pode-se voltar a esta zona do código utilizando a opção Lista de Tarefas (Task List) do Visual Studio do menu View e seleccionar de seguida a opção Task List.
- Veja que o comentário aparece na lista:



The screenshot shows the 'Task List' window in Visual Studio. It has a title bar 'Task List' with standard window controls. Below the title bar is a search bar labeled 'Search Task List' and a dropdown menu set to 'Entire Solution'. The main area is a table with the following data:

Description	Project	File	Line
TODO: Acabar a implementação da Class Aluno	TesteConsoleCore	Program.cs	16

# Operadores no C#

- Para além dos Operadores mais comuns, o C# possui alguns operadores que pelas suas particularidades iremos referir mais em pormenor
  - Esses operadores são:
    - Operador condicional nulo ?
    - Operador de coalescência nula ??
    - Operador ternário
    - Operador *is* e *is not*

# Operadores no C#

- **Operador condicional nulo - *null-conditional operator* ?**
  - A necessidade de escrever código para testar explicitamente a condição nula antes de poder usar um objeto ou suas propriedades para evitar que uma exceção fosse disparada é maçadora e isto também faz com que o código possa ter várias estruturas condicionais
  - Com a introdução do operador condicional nulo, *null-conditional operator*, o ? (ponto de interrogação) utilizado como nos tipos anuláveis (*nullabe types*), facilitou bastante a escrita de código
    - **Basta colocar o sinal ? depois da instância e antes de chamar a propriedade**

# Operadores no C#

Assim, em vez de

```
if (aluno != null && aluno.Endereco != null)
{
    Console.WriteLine($"{aluno.Nome}\n{alunoCurso} - aluno.Idade}\n{aluno.Endereco.Rua} -
{aluno.Endereco.CodigoPostal} - {aluno.Endereco.Pais}\nMail: {aluno.Mail}");
}
```

Pode-se ter

```
Console.WriteLine($"{aluno?.Nome}\n{aluno?.Curso} - {aluno?.Idade}\n{aluno?.Endereco?.Rua} -
{aluno?.Endereco?.CodigoPostal} - {aluno?.Endereco?.Pais}\nMail: {aluno?.Mail}");
```

Sem haver o risco de ocorrer alguma exceção!

# Operadores no C#

## ▪ Operador de Coalescência Nula ??

- O operador de coalescência nula ?? Permite-nos verificar se há um valor nulo numa variável, campo ou propriedade e caso exista atribuir um valor *default*
- Este operador retornará o operando esquerdo se o operando não for nulo; caso contrário retornará o operando direito
  - Isto é feito numa única linha de código.

# Operadores no C#

## ▪ Operador de Coalescência Nula ??

- Em vez de

```
static void Main(string[] args)
{
    Aluno aluno = null;
    string nome;
    aluno!=null && aluno.Nome!=null)
    {
        nome = aluno.Nome;
    }
    Else
    {
        nome = "O nome do aluno não foi indicado!";
    }
    Console.WriteLine(nome);
}
```

# Operadores no C#

## ▪ Operador de Coalescência Nula ??

Pode-se fazer

```
Aluno aluno = null;
```

```
var nome = aluno?.Nome ?? "O nome do aluno não foi indicado!";
```

```
Console.WriteLine(nome);
```

- Basta colocar o sinal ?? depois da instância e antes de chamar a propriedade
- Isto é feito numa única linha de código.

# Operadores no C#

## ▪ Operador Ternário

- O operador condicional ternário ?: retorna um de dois valores dependendo do valor de uma expressão booleana, cuja sintaxe é:

*condição ? primeira\_expressão : segunda\_expressão;*



# Operadores no C#

- A condição é avaliada como *true* ou *false*: se for verdadeira a *primeira\_expressão* será avaliada e se tornará o resultado; se for falsa a *segunda\_expressão* será avaliada e se tornará o resultado

```
using System;
namespace ProdutividadeEmCSharp
{
    class Program
    {
        static void Main(string[] args)
        {
            string estatuto;
            Console.Write("indique a sua idade:");
            int idade = Convert.ToInt32(Console.ReadLine());
            estatuto = (idade > 18) ? "Maior de idade" : "Menor de idade";
            Console.WriteLine(estatuto);
        }
    }
}
```

# Operadores no C#

- **Operador Lógico *is not***
- Este operador *is not* do C# 9.0 permite tornar mais legíveis testes que antes eram feitos usando o operador *is*
  - *Assim, em vez de termos:*

```
if (!(valor is null))  
{  
    //Código a executar  
}  
if (!(valor is string))  
{  
    //Código a executar  
}
```

# Operadores no C#

- **Operador Lógico *is not***

- *Podemos ter:*

```
if (valor is not null)
{
    //Código a executar
}
if (valor is not string)
{
    //Código a executar
}
```

# Operadores no C#

- **Sobrecarga de Operadores - *operator override***
  - O C# permite que alguns dos operadores existentes sejam sobrecarregados, alterando seu significado quando aplicados a tipos pré-definidos pelo programador
    - Basta criar métodos estáticos e preceder o operador a sobrecarregar pela palavra-chave ***operator***
  - Para se perceber o modo de proceder analise o exemplo

# Operadores no C#

```
using System;
namespace TesteComStrings {
    class Distancia {
        public int Metros { get; set; }
        public Distancia(int metros) {
            Metros = metros;
        }
        public static Distancia operator +(Distancia a, Distancia b) {
            Distancia resultado = new Distancia(0);
            resultado.Metros = a.Metros + b.Metros;
            return resultado;
        }
        public static Distancia operator -(Distancia a, Distancia b) {
            Distancia resultado = new Distancia(0);
            resultado.Metros = a.Metros - b.Metros;
            return resultado;
        }
    }
}
```

# Operadores no C#

```
public static Distancia operator ++(Distancia a) {  
    a.Metros++;  
    return a;  
}  
public static Distancia operator --(Distancia a) {  
    a.Metros--;  
    return a;  
}  
public static bool operator ==(Distancia a, Distancia b)  
    => a.Metros == b.Metros;  
public static bool operator !=(Distancia a, Distancia b)  
    => a.Metros != b.Metros;  
public static bool operator <(Distancia a, Distancia b)  
    => a.Metros < b.Metros;  
public static bool operator >(Distancia a, Distancia b)  
    => a.Metros > b.Metros;
```

# Operadores no C#

```
public static bool operator <=(Distancia a, Distancia b)
    => a.Metros <= b.Metros;
public static bool operator >=(Distancia a, Distancia b)
    => a.Metros >= b.Metros;
}
class Program {
    static void Main(string[] args) {
        Distancia distancia1 = new Distancia(10);
        Distancia distancia2 = new Distancia(15);
        distancia1++;
        distancia2--;
        Distancia distancia3 = distancia1 + distancia2;
        Console.WriteLine($"Distancia 1: {distancia1.Metros}");
        Console.WriteLine($"Distancia 2: {distancia2.Metros}");
        Console.WriteLine($"Distancia 3: {distancia3.Metros}");
    }
}
```

# Operadores no C#

```
if (distancia1 == distancia2)
    Console.WriteLine("A distancia 1 é igual à distancia 2!");
else
    Console.WriteLine("A distancia 1 é diferente da distancia 2!");
    if (distancia1 >= distancia2)
        Console.WriteLine("A distancia 1 é maior ou igual à distancia 2!");
    else
        Console.WriteLine("A distancia 1 é menor do que a distancia 2!");
    }
}
```

Programação Web – 2023/2024 – LEI D+CE+PL – ISEC/IPC @JB



# Code Snippets para o C# no VS

## ■ Code Snippets para o C# no Visual Studio

- Os *Code Snippets* é um conjunto de códigos pré-definidos com a finalidade de ajudar a inserir trechos de códigos já prontos pressionando algumas teclas ou inserir com o rato
  - O "*Code Snippet Inserter*" tem características similares ao *IntelliSense*
- Há várias maneiras de utilizar *Code Snippets* nos projectos do *Visual Studio*:
  - No editor pressione as teclas de atalho **Ctrl+K+X**
  - No *Menu*, selecione *Edit, IntelliSense e Insert Snippet*
  - *Através dos atalhos constantes da tabela – escreva o atalho e aparece a opção e então clique na tecla Tab*

Consultar: <http://www.linhadecodigo.com.br/artigo/1374/tempo-e-dinheiro-use-code-snippets-com-csharp.aspx#ixzz79siHrGZC>

# Code Snippets para o C# no VS

## ▪ **Atalhos Code Snippets para o C# no Visual Studio**

Atalho	Descrição
do	Criar um ciclo
else	Criar um bloco else
for	Criar um ciclo for
foreach	Criar um ciclo
forr	Cria um ciclo for que decrementa a variável do ciclo
if	Cria um bloco if
switch	Cria um bloco switch
while	Cria um ciclo while

*Ao digitar estas palavras chave aparece o menu de contexto e para inserir o código respectivo clique duas vezes em Tab*

# Estrutura Switch no C#

## ■ ESTRUTURAS SWITCH

- A estrutura condicional *switch* até a versão 6 do C# *tinha* algumas limitações e não era tão flexível como as estruturas equivalentes do Visual Basic e do Delphi
- Com a versão 7 o *switch* passou a suportar *pattern matching* e estas limitações deixaram de existir e o código gerado com a estrutura condicional *switch* ficou mais conciso e legível
  - Estas alterações ao *switch* permitem utilizações menos triviais
  - No exemplo seguinte mostra-se a utilização da cláusula ***when*** no ***switch***

# Estrutura Switch no C#

## •Exemplo 1

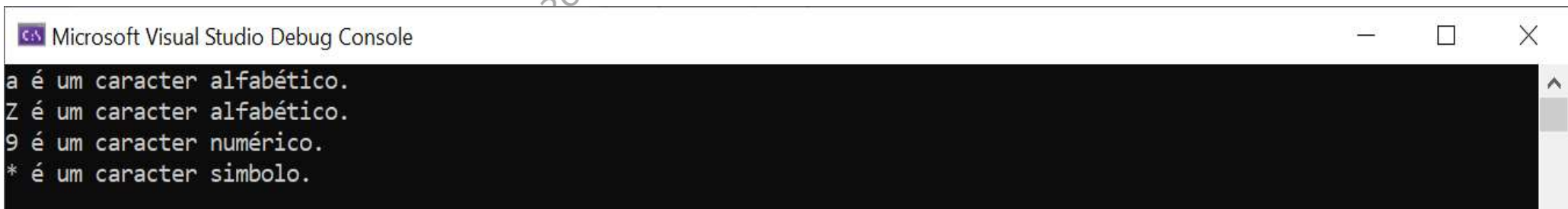
```
using System;
namespace TesteComStrings {
    class Program {
        private static void IdentificaCaracter(char c)
        {
            switch (c)
            {
                case char l when (c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z'):
                    Console.WriteLine($"{c} é um caracter alfabético.");
                    break;
                case char n when c >= '0' && c <= '9':
                    Console.WriteLine($"{c} é um caracter numérico.");
                    break;
```

# Estrutura Switch no C#

```
        case char n when c >= '!' && c <= '/':  
            Console.WriteLine($"{c} é um caracter simbolo.");  
            break;  
        default:  
            break;  
    }  
}  
static void Main(string[] args)  
{  
    IdentificaCaracter('a');  
    IdentificaCaracter('Z');  
    IdentificaCaracter('9');  
    IdentificaCaracter('*');  
}  
}
```

# Estrutura Switch no C#

- Execute o código e interprete-o
  - A saída deste código onde se utiliza when no switch para classificar os caracteres é a seguinte:



```
Microsoft Visual Studio Debug Console

a é um caracter alfabético.
Z é um caracter alfabético.
9 é um caracter numérico.
* é um caracter simbolo.
```

# Estrutura Switch no C#

- Exemplo 2 – Classificar as Fases da Vida

```
using System;
namespace TesteComStrings
{
    class Program
    {
        private static void ClassificaFaseDaVida(int idade)
        {
            string fase = String.Empty;
            switch (idade)
            {
                case int i when i <= 11:
                    fase = "Infância";
                    break;
```

# Estrutura Switch no C#

- Exemplo 2 – Classificar as Fases da Vida

```
case int i when i >= 12 && i <= 20:
    fase = "Adolescência";
    break;
case int i when i >= 21 && i <= 74:
    fase = "Idade Adulta";
    break;
case int i when i >= 75:
    fase = "Velhice";
    break;
}
Console.WriteLine($"{idade} anos = {fase}");
}
```



# Estrutura Switch no C#

- Exemplo 2 – Classificar as Fases da Vida

```
static void Main(string[] args)
{
    ClassificaFaseDaVida(1);
    ClassificaFaseDaVida(18);
    ClassificaFaseDaVida(46);
    ClassificaFaseDaVida(84);
}
}
```

# Estrutura Switch no C#

- Execute o código e interprete-o
  - A saída deste código onde se utiliza *when* no *switch* para classificar as fases da vida é a seguinte:



```
Microsoft Visual Studio Debug Console
1 anos = Infância
18 anos = Adolescência
46 anos = Idade Adulta
84 anos = Velhice
```

# Estrutura Switch no C#

- **Exemplo 3** – Neste 3º exemplo vemos a utilização no *switch* de comparações recorrendo aos sinais de “menor”, “maior”, “menor ou igual”, “maior ou igual”, etc o que o aproxima bastante do que tradicionalmente se faz com o *if*

**Nota:** Tem de se compilar com pelo menos o C# 9 para este exemplo funcionar

```
using System;
namespace TesteComStrings {
    class Program    {
        static void Main(string[] args)    {
            var valor = 3;
            switch (valor)
            {
                case < 0:
                    Console.WriteLine($"Valor {valor} é menor que 0!");
                    break;
                case 0:
                    Console.WriteLine($"Valor {valor} é igual a 0!");
                    break;
```

# Estrutura Switch no C#

case > 0 and <= 10:

Console.WriteLine(\$"Valor {valor} é maior que 0 e menor ou igual a 10!");

break;

default:

Console.WriteLine(\$"Valor {valor} é maior que 10!");

break;

```
}  
}  
}  
}
```

# Estrutura Switch no C#

- Execute o código e interprete-o
  - A saída deste código onde se utiliza comparações no *switch* para enquadrar um número num intervalo de valores é a seguinte:



```
Microsoft Visual Studio Debug Console
Valor 3 é maior que 0 e menor ou igual a 10!
```

# Instrução Switch no C#

## ▪ INSTRUÇÕES SWITCH USANDO EXPRESSÕES SWITCH

- Instruções *Switch* não é o mesmo do que Expressões *Switch*
- Caracterizam-se por a *variável aparecer antes da palavra chave switch*
- Permite uma maior compactação
- Utiliza-se um operador *Lambda*

# Instrução Switch no C#

- Alterações na sintaxe desta construção:
  - O posicionamento da variável antes da palavra-chave *switch* ajuda a distinguir visualmente a expressão *switch* da instrução *switch*
  - Os corpos são expressões, não instruções
  - Os elementos **case** e **:** são substituídos por **=>** (mais conciso e intuitivo)
  - O caso **default** é substituído por um **\_**
    - No exemplo seguinte mostra-se a utilização de instruções **switch** em expressões **switch**

# Instrução Switch no C#

- **Exemplo 4 – Neste utilizam-se instruções *switch* em expressões *switch***  
***Nota: Tem de se compilar com pelo menos o C# 9 para este exemplo funcionar***

```
using System;
namespace TesteComStrings
{
    public enum Curso
    {
        Informatica,
        Civil,
        Mecanica,
        Electrotecnia
    }
    class Program
    {
        public static string Seleccionar(Curso curso) =>
```

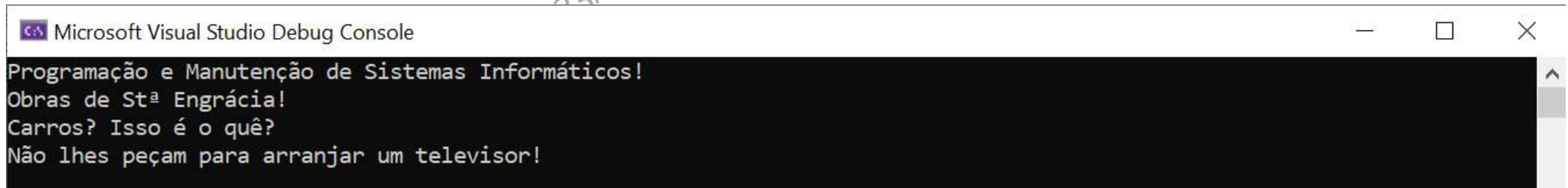


# Instrução Switch no C#

```
class Program {  
    public static string Seleccionar(Curso curso) =>  
        curso switch  
        {  
            Curso.Informatica => "Programação de Aplicações Web!",  
            Curso.Civil => "Obras de Stª Engrácia!",  
            Curso.Mecanica => "Carros? Isso é o quê?",  
            Curso.Electrotecnia => "Não lhes peçam para arranjar um televisor!",  
            _ => throw new ArgumentException(message: "Opção de Curso desconhecida!", paramName: nameof(curso))  
        };  
    static void Main(string[] args) {  
        Console.WriteLine(Seleccionar(Curso.Informatica));  
        Console.WriteLine(Seleccionar(Curso.Civil));  
        Console.WriteLine(Seleccionar(Curso.Mecanica));  
        Console.WriteLine(Seleccionar(Curso.Electrotecnia));  
    }  
}
```

# Instrução Switch no C#

- Execute o código e interprete-o
  - A saída deste código onde se utiliza instruções *switch* dentro de expressões *switch* é a seguinte:



```
Microsoft Visual Studio Debug Console

Programação e Manutenção de Sistemas Informáticos!
Obras de Stª Engrácia!
Carros? Isso é o quê?
Não lhes peçam para arranjar um televisor!
```

# Instrução Switch no C#

- O exemplo 3 reescrito com recurso a instruções *switch* e expressões *switch* ficaria como mostrado na listagem seguinte

**Nota:** Tem de se compilar com pelo menos o C# 9 para este exemplo funcionar

```
using System;
namespace TesteComStrings {
    class Program {
        static void Main(string[] args) {
            var valor = 3;
            var mensagem = valor switch {
                < 0 => $"Valor {valor} é menor que 0!",
                0 => $"Valor {valor} é igual a 0!",
                > 0 and <= 10 => $"Valor {valor} é maior que 0 e
                menor ou igual a 10!",
                _ => $"Valor {valor} é maior que 10!"
            };
            Console.WriteLine(mensagem);
        }
    }
}
```

# Tipos e Membros no C#

- Sintaxe Simplificada em Tipos Anuláveis

- A declaração formal de um tipo anulável é:

***Nullable<T> variável = null;***

- É recomendável usar a forma curta alternativa, mais legível e simples:

***T? variável = null;***