MODULO 1 - .NET e C#

AULA 01 – TIPOS E VARIAVEIS

```
using System;

namespace Tipos_Variveis
    {
    class Program
        {
        public static void Main(string[] args)
            {
            int numeroInt = 10;
            int maiorNumero = int.MaxValue;//Maior numero int
            int menorNumero = int.MaxValue;//menor numero int

            long numeroLong = 12121212312313213;
            long maiorNumeroLong = long.MaxValue;
            long menorNumeroLong = long.MinValue;

            decimal numeroDecimal = 10.52m;//Usa M no fim. Expressa com decimal mais
preciso (Altura, preços, peso..)

            double numeroDouble = 12.3;
            double menorNumeroDouble = double.MinValue;
            double maiorNumeroDouble = double.MaxValue;

            bool verdadeiro = true;
            bool falso = false;

            var numero = 10;//Var infere o tipo de varivel automatico, vale para tudo

            string nome = "Willian Vasselo";
            char letra = 'L';//Representa uma letra, sexo

            //Usado para boletos por exemplo
            DateTime entradaNaEmpresa = new DateTime(2021, 2 , 3);
            TimeSpan quantoTempoEstanaEmpresa = DateTime.Now - entradaNaEmpresa;
            }
        }
    }
```

AULA 02 – CONVERSÃO ENTRE TIPOS

```
//4 Tipos de Conversões: IMPLICITAS / EXPLICITAS / CLASSE CONVERT / METODO PARSE

using System;

namespace Conversão_Tipos
{
    public class Conversão_Tipos
    {
        public static void Main(string[] args)
        {
            //IMPLICITA: Pois converter de Int para um maior ele aceita;
            int notaAluno = 10;
            double notaAlunoDouble = notaAluno;

            //EXPLICITA: É necessario a Sintaxe (int), pois ocorre perda de
informação;
            int numeroDoubleComoInt = (int) notaAlunoDouble;

            //CALSSE CONVERT: Convertendo de Strin para Int é preciso usar a Sintaxe:
Convert.ToInt32(variavel)
            string notaString = "10";
            int notaCovert = Convert.ToInt32(notaString);

            //METODO PARSE: Sintaxe int.Parse(variavel) > Funciona para numerais, mas
se for alfanumerico da excessão:
            int notaParse = int.Parse(notaString);

            //PARSE: Validando numeral ou alfanumerico
            if (int.TryParse(notaString, out int notaTryParse))
            {
                //Se não conseguir converter retorna o Iválido
            } else{
                System.Console.WriteLine("Número formato Inválido!");
            }
        }
    }
}
```

AULA 03 – OPERADORES

```
//ARITMETICOS / COMPARAÇÃO / IGUALDADE / LÓGICOS

using System;

namespace Operadores
{
    public class Operadores
    {
        public static void Main(string[] args)
        {
            //ARITMÉTICOS (Unários ++, --, + e -)

            int numeroOperador = 4;
            System.Console.WriteLine(numeroOperador++);// 4 > Na linha debaixo é 5
            System.Console.WriteLine(numeroOperador--);// 5 > Na linha debaixo é 4
            //Usando o operador antes da variavel - Incrementa ou Decrementa antes > e
após depois
            System.Console.WriteLine(++numeroOperador);// 5
            System.Console.WriteLine(--numeroOperador);// 4

            System.Console.WriteLine(numeroOperador);
            System.Console.WriteLine(-numeroOperador);//Sinal -
            System.Console.WriteLine(-(-numeroOperador));//Sinal +

            //OPERADORES (Binários + - * /)
            var soma = 4 + 5;
            var subtracao = 4 - 5;
            var multiplicacao = 4 * 5;
            var divisao = 20 / 3;
            var divisaoDouble = (double)20 / 3;

            var multiplos = (4 * 5) + 10;

            //OPERADORES DE COMPARAÇÃO ( > >= < <=) - Retorna True ou False

            System.Console.WriteLine(4 > 5);//FALSE
            System.Console.WriteLine(5 > 5);//FALSE
            System.Console.WriteLine(6 >= 5);//TRUE

            System.Console.WriteLine(5 < 4);//FALSE
            System.Console.WriteLine(5 <= 5);//TRUE
            System.Console.WriteLine(5 < 6);//TRUE

            //OPERADORES IGUALDADE ( == , != )
            System.Console.WriteLine(5 == 5);//TRUE
            System.Console.WriteLine(5 == 4);//FALSE

            System.Console.WriteLine(5 != 5);//FALSE
            System.Console.WriteLine(4 != 5);//TRUE

            //OPERADORES LÓGICOS ( AND(&&) , OR(||) )
```

```
            //OR || - Retorna True se um deles for True
            System.Console.WriteLine(true || false);//TRUE
            System.Console.WriteLine(false || true);//TRUE
            System.Console.WriteLine(false || false);//FALSE
            System.Console.WriteLine(true || true);//TRUE

            //AND && - Retorna True somente quando os 2 são True
            System.Console.WriteLine(true && false);//FALSE
            System.Console.WriteLine(false && true);//FALSE
            System.Console.WriteLine(false && false);//FALSE
            System.Console.WriteLine(true && true);//FALSE

            // EXEMPLO

            var minhaNota = 7;
            var ultimoAno = true;
                //true             true  = true (Print)
            if (minhaNota >= 7 && ultimoAno)
            {
                System.Console.WriteLine("Aprovado!");
            }
            // var = 4 - false     true  =  true (Print)
            if (minhaNota >= 7 || ultimoAno)
            {
                System.Console.WriteLine("Aprovado!");
            }
        }
    }
}
```

```
//ESTRUTURAS DE CONDIÇOES: Permitem que o sistema tome uma decisão (IF / ELSE - SWITCH / CASE);

using System;

namespace Estruturas_Condicoes
{
    public class Estruturas_Condicoes
    {
        public static void Main(string[] args)
        {
            //IF/ELSE - Booleano caso o if não seja atendido o bloco Else é executado;
            System.Console.WriteLine("Digite 1 ou 2");
            var numeroDigitado = Console.ReadLine();

            if (numeroDigitado == "1")
            {
                 System.Console.WriteLine("O numero 1 foi digitado!");
            }
            else if (numeroDigitado == "2")
            {
                 System.Console.WriteLine("O numero 2 foi digitado!");
            }
            else
            {
                 System.Console.WriteLine("Outro numero foi digitado!");

            //SWITCH/CASE Utilizado para valores mais definidos. EX: Atendente pede para digitar 1, 2...

            System.Console.WriteLine("Digite 1 ou 2");
            var numeroDigitado = Console.ReadLine();

            switch (numeroDigitado)
            {
                case "1":
                    System.Console.WriteLine("O numero 1 foi digitado!");
                    break;
                case "2":
                    System.Console.WriteLine("O numero 2 foi digitado!");
                    break;
                case "3":
                    System.Console.WriteLine("Outro numero foi digitado!");
                    break;
            }


        }
    }
}
```

EXEMPLO - IF/IF-ELSE/ELSE

```
using System;

namespace Estruturas_Condicoes
{
    public class Estruturas_Condicoes
    {
        public static void Main(string[] args)
        {
           var notaDigitada = Console.ReadLine();
           var nota = int.Parse(notaDigitada);

           if (nota >= 70)
           {
                System.Console.WriteLine("Parabens foi aprovado!");
           }
           else if (nota >= 40)
           {
                System.Console.WriteLine("Em recuperação!");
           }
           else
           {
                System.Console.WriteLine("Reprovado!");
           }
        }
    }
}
```

EXEMPLO - SWITCH CASE

```
using System;

namespace Estruturas_Condicoes
{
    public class Estruturas_Condicoes
    {
        public static void Main(string[] args)
        {
             System.Console.WriteLine("Seja Bem Vindo!!");

             System.Console.WriteLine("Digite 1 - Contratação de um plano!");
             System.Console.WriteLine("Digite 2 - Reclamação!");
             System.Console.WriteLine("Digite 3 - Segunda via do Boleto.");
             System.Console.WriteLine("Digite 4 - Sair!");

             var opcao = Console.ReadLine();
                    //Varivel
             switch (opcao)
             {
               case "1":
```

```
                    System.Console.WriteLine("Informações de um plano novo!");
                    break;
               case "2":
                    System.Console.WriteLine("Fale sobre sua reclamaçaõ!");
                    break;
               case "3":
                    System.Console.WriteLine("Qual o boleto deseja?");
                    break;
               case "4":
                    System.Console.WriteLine("Ate mais!");
                    break;
          }
     }
}
}
```

AULA 05 – ESTRUTURAS DE REPETIÇÃO

```
//ESTRUTURAS DE REPETIÇÃO: WHILE - FOR - FOREACH

using System;

namespace Estruturas_Repeticao
```

```
{
    public class Estruturas_Repeticao
    {
        static void Main(string[] args)
        {
            //FOR: Retorna um Booleano - Composta> Inicializador > Condição > 
Interador
            System.Console.WriteLine("Digite numeros separados por espaço");
            var numerosDigitados = Console.ReadLine();
            var arrayNumeros = numerosDigitados.Split(' ');//Split quebrar uma 
sequencia de texto

            //Inicializador       Condição         Soma +1
            for (int i = 0; i < arrayNumeros.Length; i++)
            {
                System.Console.WriteLine("Numero: " + arrayNumeros[i]);
            }

            //WHILE: Executa o bloco de codigo enquanto a condição for verdadeira
            System.Console.WriteLine("Digite numeros separados por espaço");
            var numerosDigitados = Console.ReadLine();
            var arrayNumeros = numerosDigitados.Split(' ');
            var contador = 0;//Precisa inicializar manual

            while (contador < arrayNumeros.Length)
            {
                System.Console.WriteLine("Numero: " + arrayNumeros[contador]);
                contador++;//Interador manual
            }

            //FOREACH: Percorre elementos sem precisar definir inicio/paradar - Mais 
simples
            System.Console.WriteLine("Digite numeros separados por espaço");
            var numerosDigitados = Console.ReadLine();
            var arrayNumeros = numerosDigitados.Split(' ');
                        //numero é um elemento da lista
            foreach (var numero in arrayNumeros)
            {
                System.Console.WriteLine("Numero: " + numero);
            }
        }
    }
}
```

EXEMPLO – FOR

```
using System;

namespace Estruturas_Repeticao
{
    public class Estruturas_Repeticao
```

```
    {
        static void Main(string[] args)
        {
            System.Console.WriteLine("Digite uma sequencia de numeros, separados por
espaço");//0 1 2 ...
            var numerosTexto = Console.ReadLine();
            var numeros = numerosTexto.Split(' ');

            System.Console.WriteLine("Numeros: ");
             //Inicia em 0 - Length tamanho do array
            for (int i = 0; i < numeros.Length; i++)
            {
                        //Imprimi da posição i = 0;
                System.Console.WriteLine(numeros[i]);
            }
        }
    }
}
SAIDA
1
2
3
```

EXEMPLO – WHILE

```
using System;

namespace Estruturas_Repeticao
{
    public class Estruturas_Repeticao
    {
        static void Main(string[] args)
        {
            System.Console.WriteLine("Digite uma sequencia de numeros, separados por
espaço");
            var numerosTexto = Console.ReadLine();
            var numeros = numerosTexto.Split(' ');

            System.Console.WriteLine("Numeros: ");

            var contador = 0;

            while (contador < numeros.Length)
            {
                System.Console.WriteLine(numeros[contador]);
                contador++;//Interador importante para não ser infinito
            }
        }
    }
}
SAIDA
```

```
1
2
3
```

EXEMPLO – FOREACH

```
using System;

namespace Estruturas_Repeticao
{
    public class Estruturas_Repeticao
    {
        static void Main(string[] args)
        {
            System.Console.WriteLine("Digite uma sequencia de numeros, separados por
espaço");
            var numerosTexto = Console.ReadLine();
            var numeros = numerosTexto.Split(' ');

            System.Console.WriteLine("Numeros: ");

            var contador = 0;
            //Define o tipo da coleção - corre a variavel
            foreach (var numero in numeros)
            {
                System.Console.WriteLine(numero);
            }
        }
    }
}
SAIDA
1
2
3
```

AULA 06 – TRABALHANDO STRINGS

- LENGTH – Tamanho da cadeia de caracteres;
- EMPTY – Cadeia de caracteres vazio;
- TO UPPER/LOWER – Retorna em letra minúscula ou maiúscula;
- SPLIT – Quebra a string, baseado no separador usado. EX: csv, espaço em branco...
- TRIM, TRIMEND, TRIMSTART – Remover espaços em branco, trim remove das 2 pontas;
- ISNULLORWHUTESPACE – Retorna um bool verifica se o string é um valor nulo ou é composto por espaços em branco;
- REPLACE – Receber 2 caracteres e substitui;

```
//STRINGS Princiapis Metodos: LENGTH / EMPTY / TO UPPER/LOWER / SPLIT / TRIM, TRIMEND, TRIMSTART / ISNULLORWHUTESPACE / REPLACE

using System;

namespace Strings
{
    public class Strings
    {
        public static void Main(string[] args)
        {
            var paragrafo = "   C# é uma linguagem moderna e versatil." + "Com C# consigo desenvolver para Web, Desktop, Jogos, " + "Mobile, entre outros.  ";

            //LENGTH - Tamanho do string
            var tamanho = paragrafo.Length;

            //EMPTY - Retorna string vazio
            var vazio = string.Empty;

            //TOLOWER / TOUPPER - Minusculo / maiusculo
            var paragrafoMinusculo = paragrafo.ToLower();
            var paragrafoMaiusculo = paragrafo.ToUpper();

            //SPLIT Obtem frases atraves da divisao escolhida
            var frases = paragrafo.Split();

            //TRIM / TRIMEND / TRISTART - Remove so espaços
            var paragrafoTrim = paragrafo.Trim();
            var paragrafoTrimend = paragrafo.TrimEnd();
            var paragrafoTristart = paragrafo.TrimStart();

            //ISNULLORWHITESPACE
            var isnullorwhitespace = string.IsNullOrWhiteSpace(paragrafo);

            //REPLACE mudar a string
                                        //Informar o que vai mudar
            var paragrafoCsharp = paragrafo.Replace("C#", "C-sharp");

            //READKEY - Para aplicação finalizar a execução
            Console.ReadKey();

        }
    }
}
```

EXEMPLO

```
using System;

class Program {
    static void Main(string[] args) {
```

```
        string nullString = null;
        string emptyString = "";
        string spaceString = "    ";
        string tabString = "\t";
        string newlineString = "\n";
        string nonEmptyString = "texto";
        Console.WriteLine(string.IsNullOrEmpty(nullString));//True
        Console.WriteLine(string.IsNullOrEmpty(emptyString));//True
        Console.WriteLine(string.IsNullOrEmpty(spaceString));//False
        Console.WriteLine(string.IsNullOrEmpty(tabString));//False
        Console.WriteLine(string.IsNullOrEmpty(newlineString));//False
        Console.WriteLine(string.IsNullOrEmpty(nonEmptyString));//False
        Console.WriteLine();
        Console.WriteLine(string.IsNullOrWhiteSpace(nullString));//True
        Console.WriteLine(string.IsNullOrWhiteSpace(emptyString));//True
        Console.WriteLine(string.IsNullOrWhiteSpace(spaceString));//True
        Console.WriteLine(string.IsNullOrWhiteSpace(tabString));//True
        Console.WriteLine(string.IsNullOrWhiteSpace(newlineString));//True
        Console.WriteLine(string.IsNullOrWhiteSpace(nonEmptyString));//False
    }
}
```

AULA 07 - TRABALHANDO STRINGS – BUSCAS

- INDEXOF (String) ou (Char) – Retorna o índice(caractere de inicio) da primeira ocorrência da busca;
- LASTINDEXOF (String) ou (Char) – Retorna o índice da ultima ocorrência da busca;
- STARTSWITH (String) ou (Char) – Retorna bool(String/Char começa com:);
- SUBSTRING (Int32) ou (Int32, Int32) – Dois usos: Pode passar um int e retorna a primeira posição, Com 2 parametros marca o inicio e a segunda mostra o comprimento do mesmo;
- CONTAINS (String) ou (Char) – Retorna bool e verifica em todas posições se contém;
O cifrão ($) na frente de uma cadeia de caracteres permite que você coloque expressões como nomes de variáveis em chaves na cadeia de caracteres. O valor da expressão é inserido na cadeia de caracteres no lugar da expressão. Essa sintaxe é conhecida como cadeias de caracteres interpoladas.

```
//STRINGS BUSCAS - INDEXOF / LASTINDEXOF / STARTSWITH / SUBSTRING / CONTAINS

using System;

namespace Strings
{
    public class Strings
    {
        public static void Main(string[] args)
        {
            var outroParagrafo = "C# é uma linguagem moderna e versatil." + "Com C#
consigo desenvolver para Web, Desktop, Jogos, " + "Mobile, entre outros.";

            //INDEXOF
            var indexOf = outroParagrafo.IndexOf("C#");

            //LASTINDEXOF
            var lastIndexOf = outroParagrafo.StartsWith("C#");

            //SUBSTRING
            var indexOfMobile = outroParagrafo.IndexOf("Mobile");
                        //Variavel palavra Mobile - Quantidade de caracteres
            var substringMobile = outroParagrafo.Substring(indexOfMobile, 6);

            //CONTAINS
            //Localiza a string e ignora se é maiusculo ...
            var containsJogos = outroParagrafo.Contains("jogos",
StringComparison.OrdinalIgnoreCase);
                                    //Exatamente igual a string digitada
            var containsJogosExatos = outroParagrafo.Contains("Jogos");

            var containsRuim = outroParagrafo.Contains("Ruim");
            Console.ReadKey();
        }
    }
}
```

AULA 08 – ARRAYS E LISTAS

- ARRAYS – Ou Matriz, é uma estrutura de dados que permite o armazenamento de dados do mesmo tipo, caso queria armazenar outros tipos um Object precisa ser definido. Permite a interação de seus valores através do: FOR, FOREACH e WHILE. Pouco Usado em C#
- LISTA – Fortemente tipada de objetos, mas oferece grande quantidade de Metodos auxiliares: Inserção, Remoção, Buscas, ... Muito estendida com o Metodo LINQ (namespace System.Linq). Muito usadas no dia a dia do desenvolvedor .NET;

METODOS E PROPRIEDADES – LISTAS:
- COUNT – Contador de elementos dentro da lista;
- ADD / ADD RANGE – Add adiciona um elemento na lista / Addrange adiciona uma coleção de elementos;
- CONTAINS – Permite verificar se a lista contém um certo elemento;
- SORT / REVERSE – Metodos de orndenação da lista ou reverte;
- FOREACH – Permite que execute para cada item da lista um bloco de códigos;
- REMOVE / REMOVEALL – Remove um item ou todos;
- CLEAR – Limpar os elementos dentro de uma lista;

MATRIZ

```
using System;

namespace Arrays_Listas
{
    public class MyClass
    {
        public static void Main(string[] args)
        {
            //ARRAYS     Outra maniera de inicializar
            //int[] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
            var numbers = new int[] { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
            var numbersCopy = new int[10];
            //Percorrer a matriz
            for (var i = 0; i < numbers.Length; i++)
            {   //Copiando uma variavel para outra
                numbersCopy[i] = numbers[i];
            }
            //Convertendo numeros Strings em Int
             var numbersString = "0 1 2 3 4 5 6 7 8 9";
             var numberArray = numbersString.Split(' ');
             var numberCovertedFromString = Array.ConvertAll(numberArray,
Convert.ToInt32);
        }
    }
}
```

LISTAS

```
using System;
using System.Collections.Generic;

namespace Arrays_Listas
{
    public class MyClass
    {
```

```
        public static void Main(string[] args)
        {

            //Inicializar List<> - CTRL + . = using System.Collections.Generic; Lista
na utiliza a capacidade
            var list = new List<int> { 0, 1, 2, 3, 4};

            //METODOS:
            //ADD - ADDRANGE
            list.Add(5);//Add um valor unitario
            list.AddRange(new List<int> {6, 7});//Add uma Lista
            list.AddRange(new int[] { 8, 9});//Add um Array

            //CONTAINS - Contar quantos elementos tem na lista
            var count = list.Count;
            var contains14 = list.Contains(14);//False
            var contains2 = list.Contains(2);//True

            System.Console.WriteLine("Lista Reversa");
            //REVERSE
            list.Reverse();
            //Passa uam instrução para cada elementno da lista
            list.ForEach(1 => Console.WriteLine(1));

            System.Console.WriteLine("Lista Ordenada");
            list.Sort();
            list.ForEach(1 => Console.WriteLine(1));

            list.Remove(4);
            list.RemoveAll(1 => 1 > 5);//Remove todo maior que 5

            list.Clear();
        }
    }
}
```

AULA 09 - LINQ – Language Integrated Query
É uma sintaxe para consultas em .NET, amplamente usado no dia a dia, extendendo a capacidade como List.

MÉTODOS
- ANY – Voce passa uma condição e verifica se contém algum elemento;
- SINGLE / SIGLEORDEFAULT – Focado em busca, passa uma expressão e localiza algum que atende ou retorna Null, mais de um elemento lança uma excessão;
- FIRST / FIRSTORDEFAULT – Primeiro elemento que atende a expressão, mas de um elemento lança uma excessão; Single é melhor
- ORDERBY / ORDERBYDESCENDING – Passa um campo para que ordene crescente ou decrescente;
- WHERE – Recebe uma condição e retorna todos elementos que atendem a condição da coleção; Usado para filtragem de dados;
- SELECT / SELECTMANY – Projeções de dados, pode transformar os elementos em outros tipos. Quando não quer retornar todos dados por sigilo. Junta dados em uma coleção;
- SUM / MIN / MAX / COUNT – Soma baseada em uma propriedade; Seleciona o item com menor ou maior valor; Count retorna a quantidade de elementos da coleção;

Classe criada para exemplos

```
using System;
namespace Aula_09___LINQ.Model
{
    public class Student
    {
        public Student(int id, string fullName, string document, int grade)
        {
            Id = id;
            FullName = fullName;
            Document = document;
            Grade = grade;
            PhoneNumbers = new List<string>
{"12345678910","12344654910","124565878910","12348321910", "123487938910"};
        }
        public int Id { get; set; }
        public string FullName { get; set; }
        public string Document { get; set; }
        public int Grade { set; get; }
        public List<string> PhoneNumbers { get; set; }

    }
}
```

```
using System;
using Aula_09___LINQ.Model;

namespace LINQ
{
    public class MyClass
    {
        public static void Main(string[] args)
        {
            var students = new List<Student>
            {
                new Student(1, "Will", "12345678910", 100),
                new Student(1, "Marcos", "12344654910", 15),
                new Student(1, "Fabio", "124565878910", 105),
                new Student(1, "Will", "12348321910", 88),
                new Student(1, "Roger", "123487938910", 50),
            };

            //ANY - Existe agum student nesta lista
            var any = students.Any();
            //Tem algum aluno com nota 100 condição
            var any100 = students.Any(s => s.Grade == 100);

            //SINGLE se for 1 - Retorna caso diferente lança excessão
            var single = students.Single(s => s.Id == 1);
            //Existe algum aluno com nota igual a zero
```

```
            var singleOrDefault = students.SingleOrDefault(s => s.Document ==
"12345678910");

            //FIRST - Retorna a primeira expressão correta que achar
            var first = students.First(s => s.FullName == "Will");
            //Retorna Null se não atender
            var FirstOrDefault = students.FirstOrDefault(s => s.Grade == 0);

            //ORDERED - Ordem crescente e decrescente
            var orderedByGrade = students.OrderBy(s => s.Grade);
            var orderedByGradeDescending = students.OrderByDescending(s => s.Grade);
            //WWHERE muito versatil, busca alunos com nota igual ou maior que 70
            var approvedStudents = students.Where(s => s.Grade >= 70);

            //SELECT selecionando apenas as notas deles
            var grades = students.Select(s => s.Grade);
            //Seleciona e junta todos em uma coleção
            var phoneNUmbers = students.SelectMany(s => s.PhoneNumbers);
            //SUM somando todas as notas dos alunos
            var sum = students.Sum(s => s.Grade);
            var min = students.Min(s => s.Grade);
            var max = students.Max(s => s.Grade);
            var count = students.Count;;


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

AULA 10 – TRABALHANDO DATAS – DATETIME

Estrutura utilizada para data em C#; Permite a formatação em vários formatos, de acordo com a cultura e outros fatores; Sem dados inicio: 01/01/0001 00:00:00;

PRINCIPAIS METODOS:
- AddDays / AddHours / AddSeconds / AddMinutes / AddMonths – Aceitam valores negativos, pega uma data para busca, adicionar datas, horas .. (EX: AddDays = -3 tres dias atrás)
- AddShortDateString/ToShortDateString/ToLOngDateString/ToLOngTimeString – Formatação de datas, representam um formato de data/horas;
- ToString – Permite que receba uma formatação e consegue representar uma data mais simples;
PRINCIPAIS PROPRIEDADES:
-Date – Apenas a parcela da data;
- Day / Month / Minute / Hour / Second / Year – Parcelas das datas
- Now / Today – Now inclui data e hora / Today refere a data de hoje as 00:00
- DayOfWeek / DayOfYear – Permite que pega as datas baseadas no dia da semana do ano ...

```
using System;

namespace Date_Time
{
```

```
    public class MyClass
    {
        public static void Main(string[] args)
        {
            var now = DateTime.Now;//Horario e data atual
            var today = DateTime.Today;//Mostra apenas a data

            //Metodos que usam Somas
            var ThreeDaysAgo = today.AddDays(-3);//3 dias atras > now o today inclui
horas também
            var sixMonthsLater = today.AddYears(6);//daqui a 6 meses > Assinatura
semestral
            var twoYearsLater = today.AddYears(2);

            var shortDate = now.ToShortDateString();//26/03/2022
            var longDate = now.ToLongDateString();//sabado 26 de março de 2022.

            var shortTime = now.ToShortTimeString();1//18:35
            var longTime = now.ToLongTimeString();//18:35:55
            //Somente datas, dias, meses...
            var date = now.Date;//26/03/2022
            var day = now.Day;//26
            var month = now.Month;//03
            var year = now.Year;//2022
            var hour = now.Hour;//18
            var minute = now.Minute;//37
            var second = now.Second;//25
            //Retorna o dia da semana
            var dayOfWeek = now.DayOfWeek;

            //Se o dia da semana for sado ou domingo - è fim de semana
            if (dayOfWeek == DayOfWeek.Saturday || dayOfWeek == DayOfWeek.Sunday)
            {
                System.Console.WriteLine("É fim de semana!");
            }

            var dayOfYear = now.DayOfYear;//Qual o dia do ano

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

FORMATAÇÃO DE DATAS
Diferentes maneiras de formatar

```
using System;

namespace Date_Time
{
```

```
    public class MyClass
    {
        public static void Main(string[] args)
        {
            var now = DateTime.Now;
            //Listagem dos formatos que exitem para datas
            System.Console.WriteLine("Formato: ");
            var formats = new string[]
{"d","D","f","F","g","G","m","o","r","s","t","T","u","U","Y"};

            foreach (var format in formats)
            {
                //2 formas de imprimir o resultado
                System.Console.WriteLine("Data no Formato {0}: {1}", format,
now.ToString(format));
                //Usando o $ se chama interpolação
                System.Console.WriteLine($"Data no formato {format} :
{now.ToString(format)}");
            }
            //Maneiras de formatar manualmente
            System.Console.WriteLine($"Data no format d: {now:d}");
            System.Console.WriteLine($"Data no formato MM-dd-yyyy: {now:MM-dd-yyyy}");

            var dateFormat = now.ToString("MM-dd-yyyy");
            var dateFormatBr = now.ToString("dd/MM/yyyy HH:mm:ss");

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

SAÍDA
Formato:
Data no Formato d: 26/03/2022
Data no Formato D: sábado, 26 de março de 2022
Data no Formato f: sábado, 26 de março de 2022 19:04
Data no Formato F: sábado, 26 de março de 2022 19:04:23
Data no Formato g: 26/03/2022 19:04
Data no Formato G: 26/03/2022 19:04:23
Data no Formato m: 26 de março
Data no Formato o: 2022-03-26T19:04:23.9139988-03:00
Data no Formato r: Sat, 26 Mar 2022 19:04:23 GMT
Data no Formato s: 2022-03-26T19:04:23
Data no Formato t: 19:04
Data no Formato T: 19:04:23
Data no Formato u: 2022-03-26 19:04:23Z
Data no Formato U: sábado, 26 de março de 2022 22:04:23
Data no Formato Y: março de 2022
Data no format d: 26/03/2022
Data no formato MM-dd-yyyy: 03-26-2022

AULA 11 – ARQUIVOS E DIRETORIOS

- CLASSE DIRECTORY – Classe estática

- CLASSE DIRECTORYINFO – Classe que podemos instanciar e passar o caminho;

Podemos criar, movimentar, interação. Também possuem as propriedades abaixo:
- Name / Parent / Root / Exists – Parent é um diretório acima é o diretório pai, retorna o caminho completo dele; Root retorna o caminho raiz do C; Exists diz se o diretório existe;
- Create, CreateDirectory – Create usa para criar pasta da Classe DirectoryInfo; CreateDirectory usa para criar pasta da Classe Create;
- Move / MoveTo – Move e MoveTo usamos para DirectoryInfo e Directory;
- GetFiles – Usada para interação dos arquivos dentro da pasta;
- GetDirectories – Mostra as pastas dentro do diretório;
- Delete – Apaga diretórios;

```
using System;

namespace Arquivos_Diretorios
{
    public class MyClass
    {
        static void Main(string[] args)
        {
            var folderName = "pasta";
            var subFolderName = "subpasta";
            var subFolderNameStatic = "subpasta_usingstatic";//Para diferenciar da
Directory e DirectoryInfo

            //Criando objeto instanciando a subpasta
            var directoryInfoSubFolder = new DirectoryInfo(subFolderName);
            //Verificando se existe a pasta
            if (!Directory.Exists(subFolderName) || directoryInfoSubFolder.Exists)
            {
                //Criando a pasta 2 maneiras, create se tiver instanciado > Estão na
pasta BIN quando criadas
                Directory.CreateDirectory(subFolderNameStatic);
                Directory.CreateDirectory(folderName);
                directoryInfoSubFolder.Create();
                //Movendo de uma para outra pasta > Colocando a Subpasta para dentro
da pasta
                directoryInfoSubFolder.MoveTo($"{folderName}//{subFolderName}");
                //Outra maneira para mostrar o caminho das pastas usando o @ e \\
                Directory.Move(subFolderNameStatic,
@$"{folderName}\\{subFolderNameStatic}");
            }

            var name = directoryInfoSubFolder.Name;
            var parent = directoryInfoSubFolder.Parent;
            var root = directoryInfoSubFolder.Root;
            var exist = directoryInfoSubFolder.Exists;

            //Como percorrer dentro das pastas
            foreach (var directory in Directory.GetDirectories(folderName))
            {
                System.Console.WriteLine(directory);
            }
```

```
            //Deletando a subFolderName
            Directory.Delete($@"{folderName}\\{subFolderName}");

            Console.ReadKey();

        }
    }
}
```

TRABALHANDO COM ARQUIVOS
- Classe File – Classe estatica
- Classe FileInfo – Classe intaciar e passamos o caminho para o arquivo;

PRINCIAPIS METODOS E PROPRIEDADES
- DirectoryName / Name / Length – Nome da pasta, nome do arquivo, tamanho do arquivo;
- Exists – Checar se existe;
- MoveTo – Movendo de um local para outro;
- Delete – Deleta arquivo;
- CreateText – Cria um arquivo de texto com o nome que passarmos;

```
  //Criando a variavel e passando o caminho
            var file = @"pasta\texto.txt";
            //Verificando se o arquivo existe
            if (!File.Exists(file))
            {
                File.CreateText(file);
            }
            //Incializando o FileInfo passando o caminho
            var fileInfo = new FileInfo(file);

            System.Console.WriteLine($"Nome: {fileInfo.Name}, Tamanho:
{fileInfo.Length}, Data de Atualização: {fileInfo.LastWriteTime}");
```

AULA 12 – DEBUGGING E EXCEÇÕES

Debugging ou depuração, significa que a aplicação esta sendo executada com um "debugger" anexado a ela; Atraves do debugger conseguimos ver o que o código esta fazendo enquanto executa; Podemos alterar valores e ver as variáveis, seguindo passo a passo a execução da aplicação; Para verificar se o código esta rodando corretamente ou achar o erro, fazendo alterações para correção;
COMO DEBUGAR
- Defina um break point, é o ponto onde o VSCode vai suspender a execução;
- Definir o break point logo antes do comportamento errado do código;

DEBUG vs RELEASE
- São dois modos para realizar a build da aplicação;
- Debug não é otimizado, permite inserir pontos de interrupção e outros, para executar o código passo a passo e inspecionar valores;
- Release focado em otimização, é focado para ambientes de produção; Gera uma publicação um pacote baseado na aplicação;

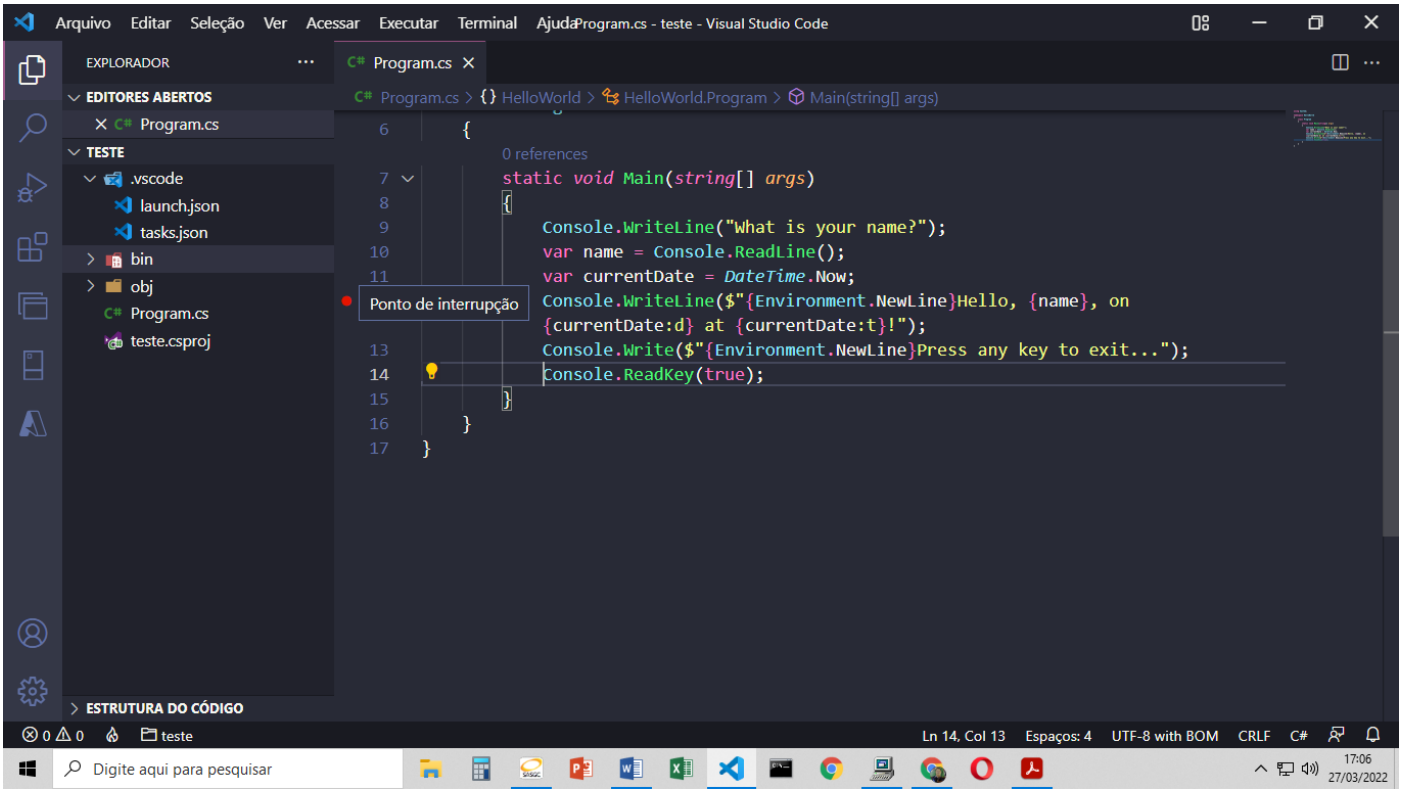
DEBUGGANDO VSCode:
Definir um ponto de interrupção

Um *ponto de interrupção* interrompe temporariamente a execução do aplicativo antes que a linha com o ponto de interrupção seja executada.

1. **Abra o arquivo *Program.cs*.**

2. **Defina um *ponto de interrupção* na linha que exibe o nome, a data e a hora clicando na margem esquerda da janela de código. A margem esquerda está à esquerda dos números de linha. Outras maneiras de definir um ponto de interrupção são pressionando F9 ou escolhendo executaralternância de ponto de interrupção no menu enquanto a linha de código está selecionada.**

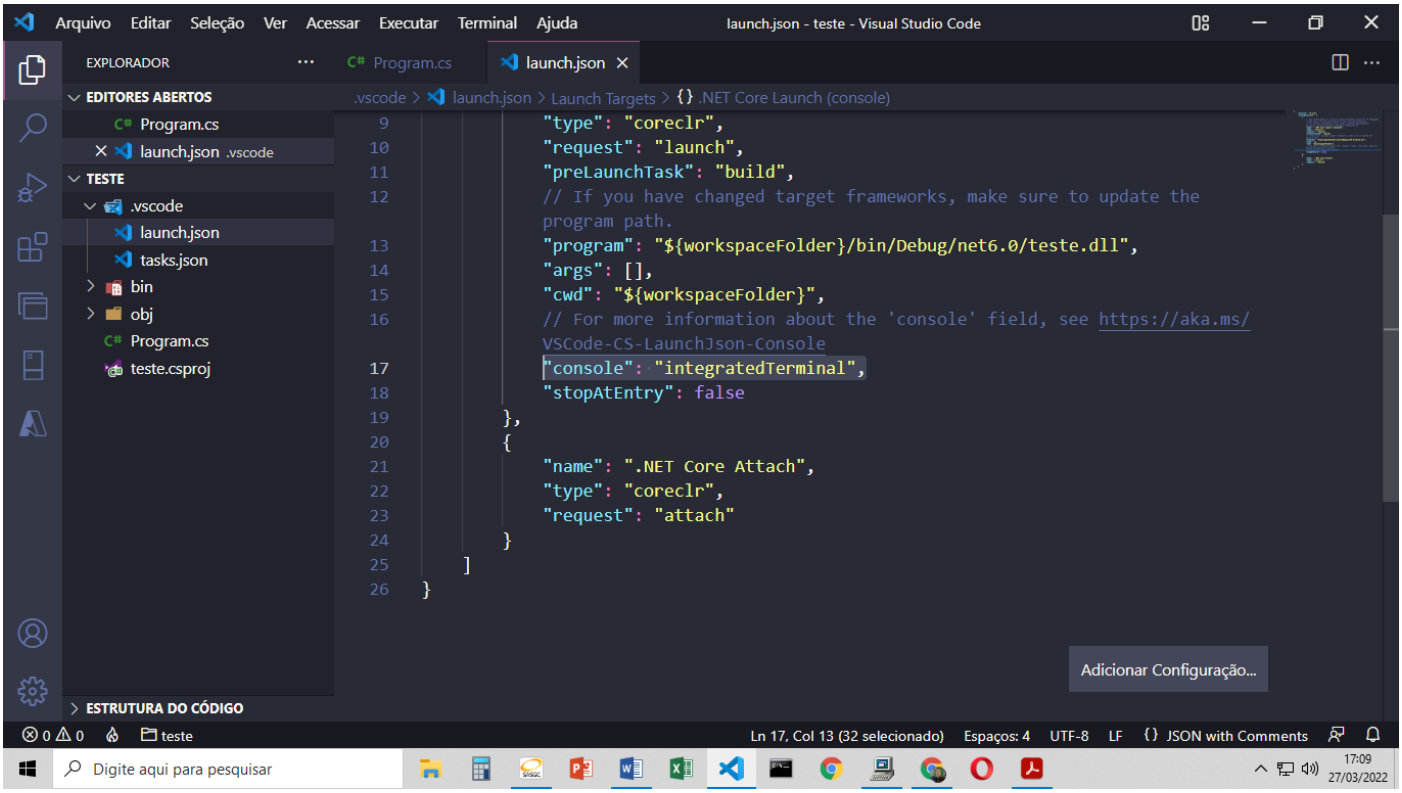
Visual Studio Code indica a linha na qual o ponto de interrupção é definido exibindo um ponto vermelho na margem esquerda.



CONFIGURAR ENTRADA DO TERMINAL

O ponto de interrupção está localizado após uma chamada de Console.ReadLine método. O console de depuração não aceita a entrada de terminal para um programa em execução. Para lidar com a entrada de terminal durante a depuração, você pode usar o terminal integrado (uma das janelas do Visual Studio Code) ou um terminal externo. Neste tutorial, você usa o terminal integrado.

1. **Abra *.vscode/launch.json*.**
2. **Altere a console configuração de internalConsole para integratedTerminal**
3. **"console": "integratedTerminal",**
4. **Salve suas alterações.**
5. **F5 – Para iniciar / F10 – Andar passo a passo;**
6. **Será iniciado no Terminal;**



EXCEÇÕES

Quando programamos fazemos o melhor para diminuir os erros e nos proteger deles. Cada vez que um erro ocorre o framework .NET lança um Exception, que é um objeto com dados do erro.

EXPETIONS COMUNS:

- FormatException – Tenta converter algum valor, mas o valor não esta no formato correto;
- ArgumentNullException – Algum método que ele não espera um valor nulo e passamos nulo pra ele;
- OverFlowException – Quanto tentamos passar um valor muito grande para um int;
- IndexOutOfRangeException – Quando tentamos acessar um índice de uma matriz mas ele não existe para aquela matriz;
- IOException – Refere a operações IO, gerenciamento de arquivos;
- NullReferenceException – Ocorre quando tentamos executar um objeto, mas ele é uma instancia nula;

COMO TRATAR EXCEÇÕES

- Podemos tratar através de um bloco try-catch-finally;
- Atraves desse bloco podemos tratar exceções de maneira genérica ou definir a forma de tratamento delas;
- Finally é opcional, é usada sempre que deseje executar um certo código;

```
using System;

namespace Debbuging_Excecoes
{
    public class MyClass
    {
        static void Main(string[] args)
        {
            var seteString = "sete";
            string valorNull = null;
            var longValue = long.MaxValue.ToString();

            try
            {
                var formatException = int.Parse(seteString);
```

```
                var argumentNullException = int.Parse(valorNull);
                var overFlowException = int.Parse(longValue);
            }
            catch(FormatException ex)
            {
                System.Console.WriteLine($"Format Exception: {ex.Message}");
            }
            catch(argumentNullException ex)
            {
                System.Console.WriteLine($"ArgumentNull Exception: {ex.Message}");
            }
            catch(overFlowException ex)
            {
                System.Console.WriteLine($"OverFlow Exception: {ex.Message}");
            }
            finally
            {
                System.Console.WriteLine("Este código é executado sempre!");
            }
            Console.ReadKey();
        }
    }
}
```

AULA 13 – PROGRAMAÇÃO ORIENTADA A OBJETOS - POO
É um paradigma da programação;
- OBJETOS: São entidades ligadas ao mundo real com características, dados, informações, atributos propriedades. Comportamentos dos objetos, são chamados de métodos ou funções e estão dentro das classes;
C# é orientada a objetos com suporte a classes e objetos;
Os 4 pilares de POO são: Abstração, Herança, Encapsulamento e Polimorfismo;
Outros conceitos importantes em em Classes C# são: Construtores, Metodos e Campos;
- CLASSES: São criadas utilizando a palavra class; Classes agrupam dados e comportamentos de uma entidade única; Classes podem ser: Abstratc, Sealed, Partial, Static;
ABSTRACT: A Classe so pode ser uma classe base de outras classe, não pode ser instanciada diretamente; Outras classes pode herdar dela, mas ela não pode instanciar um objeto diretamente; EX: Numero matricula ... características diferentes das outras;
SEALED: A Classe não pode ser herdada por outras classes; Ao contrario de abstract, uma classe final;
PARTIAL: Permite que a divisão da classe seja dividida em dois ou mais arquivos;
STATIC: Indica que a classe so pode ter membros estáticos e não se pode criar uma instancia dela; Não podemos acessar diretamente ela;

PILARES DE POO

- ABSTRAÇÃO: Uma técnica que permite que escondemos do cliente detalhes da implementação, através de agrupamento de características e comportamentos relacionados; EX: Vai servir para outras coisas na aplicação, uma classe de matricula e checaria tudo;
Resulta na separação do código para um método ou classe separados do bloco anterior;
Ajuda na melhoria da qualidade do código, por separar as responsabilidades dentro da aplicação;

- HERANÇA: Técnica que permite reutilizar, estender e modificar outras classes; EX: Dentro da classe pessoa vai compartilhar informações para as classe professores, alunos, diretor...
A classe que é herdada é chamada de classe base ou pai, e a classe que herda é chamada de derivada ou filha;
É afetada por modificadores de acesso, que estejam aplicados na classe base. EX: Private(Classe filha não consegue acessar), Protected(Permite que as classes filhas acessem), Public;

- ENCAPSULAMENTO: Tecnica que permite controlar o acesso do cliente de um código, sobre dados e comportamentos internos de classe; É o cliente da classe consegue ver dentro do código. EX: Metodo que cria numero do pedido, não deve ficar publico;
Em C# é implantada através de modificadores de acesso, controlam o que o cliente vee;
Os principais modificadores de acesso são:
Public: Acesso não restrito; Qualquer classe e bloco pode chamar esse método;
Protected: Acesso limitado a classe que os contém ou aos seus tipos derivados; EX: Herança
Internal: O acesso é limitado ao Assembly atual do seu projeto;
Private: Acesso é limitado apenas a própria classe;

- POLIMORFISMO: Tecnica que permite que objetos de classes derivadas, se comportem de maneira diferente ao da classe base para o mesmo método; EX: Classe pessoa tem um método comunicar, posso reutiliza-la e adicionar um comportamento diferente, outra saudação ...
Se expressa pelas palavras Virtual e Override, sendo a primeira que define quais comportamentos poderão ser alterados na classe derivada; EX: Na classe derivada damos um outro comportamento, adicionamos na classe filha o Virtual e na classe pai Override para alterar a saudação;
Override é responsável pela implementação do método virtual na classe derivada;

CLASSE ESTUDANTE

```
namespace Aula_13___Programação_Orientada_Objetos.obj.Entities

{
    public class Estudante : Pessoa//Herdada prpiedades
    {
        public Estudante(string turma, string nome, string documento, DateTime
dataNascimento) : base(nome, documento, dataNascimento)//Base chama o construtor
da classe base
        {
            Turma = turma;
            Notas = new List<int> {5, 10, 4, 2 , 3, 5};//Ja inicializando com
valores
        }
        public string Turma { get; private set; }//Pode alterar comente dentro da
classe
        public List<int> Notas { get; private set; }

        public override void SeApresentar()//Add novas informações, sobscrever
        {
            base.SeApresentar();
            var media = Notas.Average();
            System.Console.WriteLine($"Sou um estudante da turma {Turma}, a media
das minahs notas é {media}");
        }
    }
}
```

CLASSE PESSOA

```
namespace Aula_13___Programação_Orientada_Objetos.obj
{
    public abstract class Pessoa//Abstract a classe vai ser herdada de outras
    {
        public Pessoa(string nome, string documento, DateTime
dataNascimento)//Construtor
        {
```

```
            Nome = nome;//Os parametos que passo no bloco de baixo inicializo aqui
            Documento = documento;
            DataNascimento = dataNascimento;
        }
        public string Nome { get; protected set; }//Nao quero que alte o valor
nome (protected)
        public string Documento { get; protected set; }
        public DateTime DataNascimento { get; protected set; }

        public virtual void SeApresentar()
        {
            //Criando um comportamento Polimorfismo
            System.Console.WriteLine($"Olá meu nome é {Nome}, nasci no dia
{DataNascimento} meu documento é {Documento}");
        }

    }
}
```

CLASSE PROFESSOR

```
using System;

namespace Aula_13___Programação_Orientada_Objetos.obj.Entities
{
    public class Professor : Pessoa
    {
        public Professor(decimal salario, string nome, string documento, DateTime
dataNascimento) : base(nome, documento, dataNascimento)//Contrutor
        {
            Salario = salario;
            Turmas = new List<string> {"A", "B"};
        }
        public List<string> Turmas { get; private set; }
        public decimal Salario { get; private set; }

        public override void SeApresentar()
        {
            var turmas = string.Join(',', Turmas);

            System.Console.WriteLine($"Meu salario é de R$ {Salario} dou aulas
para a turma {turmas}");
        }
    }
}
```

PROGRAM.CS

```
using System;
using Aula_13___Programação_Orientada_Objetos.obj;
using Aula_13___Programação_Orientada_Objetos.obj.Entities;

namespace POO
```

```
{
    public class MyClass
    {
        public static void Main(string[] args)
        {
            var professor = new Professor(5000m, "João", "12345", new
DateTime(1980, 1, 1));
            var estudante = new Estudante("A","Luis","54321", new DateTime(1992, 1
, 1));

            var pessoas = new List<Pessoa> { estudante, professor };

            foreach (var pessoa in pessoas)
            {
                pessoa.SeApresentar();
            }
            Console.Read();
        }
    }
}
```