Sumário

[**Introdução ao .NET** 3](#_Toc125020589)

[**Diferenças entre .NET Framework (Legado) e .NET** 3](#_Toc125020590)

[**Compilador do .NET e seu funcionamento** 3](#_Toc125020591)

[**Configuração de Ambientes e IDES** 4](#_Toc125020592)

[**Instalando o VS Code e o .NET SDK** 5](#_Toc125020593)

[**Criando o Projeto e instalando extensão** 6](#_Toc125020594)

[**Sintase e Indentação** 8](#_Toc125020595)

[Estrutura de um Projeto 8](#_Toc125020596)

[**Conceito de Classe** 8](#_Toc125020597)

[**Tipos de Dados** 10](#_Toc125020598)

[Variáveis 10](#_Toc125020599)

[Convenções 11](#_Toc125020600)

[Tipo de Operadores 12](#_Toc125020601)

[Operadores de Atribuição 12](#_Toc125020602)

[Combinando Operadores 12](#_Toc125020603)

[Convertendo tipos de variáveis 12](#_Toc125020604)

[Diferença entre Convert e Parse 13](#_Toc125020605)

[Conversão para String 13](#_Toc125020606)

[Cast Implícito (Conversão interna) 13](#_Toc125020607)

[Ordem dos Operadores 13](#_Toc125020608)

[Operadores Condicionais (Operadores Lógicos) 14](#_Toc125020609)

[Operadores Lógicos 15](#_Toc125020610)

[Operador OR (Pipe, || ) – significa ou 15](#_Toc125020611)

[Operador AND – (&&) Significa E 16](#_Toc125020612)

[Operador NOT (!) de negação 17](#_Toc125020613)

[Operadores Aritméticos e a Classe Math 17](#_Toc125020614)

[Usando Potência 19](#_Toc125020615)

[Funções Trigonométricas 20](#_Toc125020616)

[Incremento e Decremento 21](#_Toc125020617)

[Calculando a Raiz Quadrada 21](#_Toc125020618)

[Estrutura de Repetição 22](#_Toc125020619)

[Laço FOR 22](#_Toc125020620)

[Laço While 22](#_Toc125020621)

[DO WHILE 23](#_Toc125020622)

[Estrutura de um Programa em método principal 25](#_Toc125020623)

[Array e Listas 25](#_Toc125020624)

[Foreach 26](#_Toc125020625)

[Aumentando o tamanho do array 26](#_Toc125020626)

[Copiando dados de uma array existe para um novo array 26](#_Toc125020627)

[Listas 27](#_Toc125020628)

[Comentários de Códigos e Boas Práticas 27](#_Toc125020629)

[Propriedades, Métodos e Construtores 29](#_Toc125020630)

[Propriedades 29](#_Toc125020631)

[Ações GET e SET 30](#_Toc125020632)

[Validando o GET e SET 30](#_Toc125020633)

[Usando o Body Expressions 31](#_Toc125020634)

[Métodos 31](#_Toc125020635)

[Construtores 31](#_Toc125020636)

[Manipulando Valores com C# 31](#_Toc125020637)

[Concatenação de Strings 31](#_Toc125020638)

[Formatando Valores Monetários 32](#_Toc125020639)

# **Introdução ao .NET**

**O .NET é uma plataforma de desenvolvimento unificado** que permite a construção de sistema e aplicações.

C# é uma linguagem de programação usada no .NET.

## **Diferenças entre .NET Framework (Legado) e .NET**

.NET Framework > Só roda em Windows – A última versão é a 4.8

.NET (Core) > Roda em multiplataforma (MacOs, Linux e Windows) – O .NET core só tem essa nomenclatura até a 3.1 a-parti da versão 5 é só .NET

## **Compilador do .NET e seu funcionamento**

**Linguagem de alto nível:** A linguagem que entendemos e escrevemos nosso código fonte. (Programadores)

**Linguagem de baixo nível:** A linguagem que a máquina entende. Possui pouco abstração, sendo difícil de entender. (Compilador que faz essa tradução)

O que é um compilador?

É um programa que realiza a conversão de linguagem de alto nível(Programador) para baixo nível(Maquina).

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

**Compilador do .NET**

**Diagrama

Descrição gerada automaticamente**

**Compilador:** É um programa que realiza a conversão de linguagem de alto nível para baixo nível. Exemplo: C#, Java

**Transpilador:** É a conversão de uma linguagem ou implementação para outra. A sua saída permanece em linguagem de alto nível. Exemplo: typescript para Javascript.

**Linguagem compilada**: São linguagens onde o código fonte é traduzido para o código de máquina. Exemplo: C#, Java

**Linguagem interpretada**: São linguagens que fazem a leitura e interpretação diretamente do código fonte. Exemplo: Javascript, PHP

## **Configuração de Ambientes e IDES**

Uma IDE (Integrated Development Environment), ou ambiente de desenvolvimento integrado, é um software que facilita e integra diversas facilidades para a escrita e depuração do código.

O **VS (Visual Studio)** Principal IDE para o .NET, com suporte para o C#, C++, Python, Node.js, Unity e mobile.

**Vantagens**

* Versão gratuita (Community)
* Debug rico em detalhes
* Disponível para Windows e Mac

**Desvantagens**

* Não disponível para Linux
* Performance. Exige muitos recursos da máquina

O **VS Code** é um editor de texto usado para facilitar o desenvolvimento de diversas linguagens.

**Vantagens**

* Totalmente gratuito
* Possibilidade de instalar extensões
* Disponível para Windows, Mac e Linux
* Muito leve em performance

**Desvantagens**

* É necessária uma configuração inicial
* Não é muito intuitivo para algumas funcionalidades

**RIDER** uma IDE para .NET

**Vantagens**

* Rico em funcionalidades
* Forte integração com o .NET
* Facilidade em trabalhar com o Unity
* Recomendações de refatoração de código
* Atalhos e comandos que aumentam a produtividade

**Desvantagens**

* Pago
* Performance. Exige muitos recursos da máquina

## **Instalando o VS Code e o .NET SDK**

Para baixar o pacote .NET SDK e o Runtime

SDK - Para o desenvolvimento e já tem o Runtime

Runtime - Para execução da aplicação (Necessário ter nas maquinas que irá rodar a aplicação)

https://dotnet.microsoft.com/en-us/download

No prompt de comando (CMD) digitar o comando abaixo para verificar as info do dotnet e versão

Comando: dotnet –info

Texto

Descrição gerada automaticamente

Para instalar o VS Code

<https://code.visualstudio.com/>

## **Criando o Projeto e instalando extensão**

Criando o projeto

dotnet new console

-- Criamos um projeto do tipo console onde o resultado é para nossa linha de comando no terminal

Instalando a extensão do C# - Essa extensão serve para o autocomplete e diz o que o método está fazendo. Traz tudo do C# métodos etc. e além de mostrar que tem erro caso a sintaxe não é a esperada. As Extensões trazem acesso as funcionalidades da linguagem.

Tela de celular com aplicativo aberto

Descrição gerada automaticamente

Comando para execução

dotnet run (Esse comando ele ira compilar e executar o programa)

Tela de celular com aplicativo aberto

Descrição gerada automaticamente

Extensões auxiliares para facilitar e desenvolvimento

C# extensions - Essa extensão facilita a criação de novas classes

Tela de celular com aplicativo aberto

Descrição gerada automaticamente

Vscode icons – Uma extensão visual dos ícones dos arquivos

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

## **Sintase e Indentação**

### Estrutura de um Projeto

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

.csproj - Representa uma estrutura de um projeto – os Metadados

.cs – Representa nossa classe

## **Conceito de Classe**

É uma abstração do mundo real e representar no programa de software

Uma imagem contendo Interface gráfica do usuário

Descrição gerada automaticamente

Toda classe começa com a letra maiúscula no C#

Tela de vídeo game

Descrição gerada automaticamentepublic class Pessoa{ }

Caminho mas nem sem sempre é o caminho físico

namespace Projeto\_CSharp.Moldes

{

public class Pessoa

{

public string? Nome { get; set; }

Atributos

public int Idade { get; set; }

public void Apresentar()

{

Console.WriteLine($"Olá, Meu nome é {Nome}, e tenho {Idade} anos");

}

}

}

## **Tipos de Dados**

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

## Variáveis

string apresentacao = " String";

int quantidade =1;

//Manipulação de variaveis

quantidade = 10;

double altura = 1.80;

decimal preco = 1.80M;

bool condicao = true;

Console.WriteLine("Valor da Variavel apresentacao do tipo" + apresentacao);

Console.WriteLine("Valor da Variavel quantidade do tipo int: " + quantidade);

Console.WriteLine("Valor da Variavel altura do tipo double: " + altura.ToString("0.00"));

Console.WriteLine("Valor da Variavel preco do tipo decimal: " + preco);

Console.WriteLine("Valor da Variavel condicao do tipo bool: " + condicao);

//Variavel do tipo DateTime

DateTime dataAtual = DateTime.Now;

Console.WriteLine("Valor da Variavel do Tipo Data " + dataAtual);

//Adicionando dias

DateTime dataAtual1 = DateTime.Now.AddDays(5);

Console.WriteLine("Adcionando dias na data " + dataAtual1);

//imprimindo só a data

DateTime dataAtual2 = DateTime.Now.AddDays(5);

Console.WriteLine("Imprimindo só a data " + dataAtual2.ToString("dd/MM/yyyy"));

## Convenções

As convenções que utilizamos no C# são:

cameCase (Primeira palavra com a primeira letra minúscula seguindo da segunda palavra com a primeira letra Maiúscula)

Exemplo: Public class pessoaFisica { }

PascalCase (Primeira palavra com a primeira letra maiúscula seguindo da segunda palavra com a primeira letra Maiúscula)

Exemplo: Public class PessoaFisica { }

AS classes sempre iniciar a primeira letra da primeira palavra com a letra maiúscula menos para métodos e atributos também.

As variáveis começar a primeira letra da primeira palavra com leta minúscula

## Tipo de Operadores

### Operadores de Atribuição

O Operador de atribuição (=) atribui o valor do operador do lado direito a uma variável, uma (=) ou um elemento indexador fornecido pelo operando do lado esquerdo. O resultado de uma expressão de atribuição é o valor atribuído a um operando do lado esquerdo. O tipo de operandos do lado direito deve ser do mesmo tipo ou implicitamente conversível para o operando do lado esquerdo.

Int a = 10;

Int b = 20;

Int c = a + b;

Console.WriteLine(c);

Saida = 30

## Combinando Operadores

Int a = 10;

Int b = 20;

Int c = a + b;

0 = 10 + 20;

c = c + 5; ou podemos fazer da seguinte maneira c +=5;

30 = 30 + 5;

Console.WriteLine(c);

Saida = 35

## Convertendo tipos de variáveis

Nesse caso aqui tenho um tipo de variável int que recebo uma string e estou convertendo para int 32 bits

Int a = Convert.ToInt32(“5”);

Console.WriteLine(a);

Ou podemos fazer dessa maneira

Int a = int.Parse(“5”);

Console.WriteLine(a);

## Diferença entre Convert e Parse

Quando usa o Convert se passar o valor nulo(null) o programa retorna zero

Quando utiliza o Parse não aceita valores nulos o programa da erro

## Conversão para String

Int inteiro = 5;

String a = inteiro.ToString();

Console.WriteLine(a)

## Cast Implícito (Conversão interna)

No cast implícito faz uma conversão interno, ou seja, quando um tipo cabe dentro do outro e não precisamos passar o Convert e nem o Parse

Exemplo

Int a = 5 ;

Double b = a;

Console.WriteLine(b)

Ou seja o int cabe dento de um valor double

## Ordem dos Operadores

Para o C# a prioridade da operação é o que esta dentro dos parentesse

Exemplo

Double a = 4 / 2 + 2 ;

Console.WriteLine(a);

Saída = 4

Double a = 4 / (2 + 2) ; - Quando colocado em ( ) a ordem de execução é o que esta no()

Console.WriteLine(a);

Saída = 1

Convertendo de Maneira segura

Usasse o TryParse – ele pega um valor e tenta fazer a conversão caso de erro ele não para a execução ele continua

Exemplo

String a = “15c”;

Int b = 0;

Int.TryParse(a, out b);

Console.WriteLine(b);

Console.WriteLine(“Conversao realizada com sucesso!”);

Saida

0

Conversao realizada com sucesso

Ou seja ele trouxe o 0 no resultado pois ele tentou converter o 15c e como o 15c não é um numero inteiro ele não conseguiu converter e mesmo assim não parou o programa

## Operadores Condicionais (Operadores Lógicos)

O operador condicional possibilita mudar o fluxo de execução de seu código, indicado um caminho que ele deve percorrer.

Exemplo

Int quantidadeEmEstoque = 10;

Int quantidadeCompra = 4;

If (quantidadeEmEstoque >= quantidadeCompra){

Console.WriteLine(“Venda Realizada.”);

} else {

Console.WriteLine(“N, Temos a quantidade desejada em estoque .”);

}

Saida – Venda Realizada.

Ou podemos fazer dessa forma

int quantidadeEmEstoque = 10;

int quantidadeCompra = 0;

bool possivelVenda = quantidadeCompra > 0 && quantidadeEmEstoque >= quantidadeCompra;

Console.WriteLine($"Quantidade em Estoque: {quantidadeEmEstoque}");

Console.WriteLine($"Quantidade de Compra: {quantidadeCompra}");

Console.WriteLine($"É Possivel a compra: {possivelVenda}");

if (quantidadeCompra == 0){//==Comparação

Console.WriteLine("Venda Invalida!.");

} else if (possivelVenda){

Console.WriteLine("Venda Realizada.");

} else {

Console.WriteLine("Desculpe. Não temos a Quantidade desejada em estoque");

}

## Operadores Lógicos

### Operador OR (Pipe, || ) – significa ou

O operador || ele precisa que uma ou outra condição seja verdadeira

Exemplo de Código

bool ehMaiorDeIdade = false;

bool possuiAutorizarcaoResponsavel = false;

if (ehMaiorDeIdade || possuiAutorizarcaoResponsavel){

Console.WriteLine("Entrada Liberada!");

}

else{

Console.WriteLine("Entrada Não Liberada!");

}

Outra forma de Fazer

Console.WriteLine("Digite sua idade");

int idade1 = Int32.Parse(Console.ReadLine());

bool possuiAutorizarcaoResponsavel = false;

if (idade1 >= 18){

    Console.WriteLine("Entrada Liberada!");

}else if (possuiAutorizarcaoResponsavel){

    Console.WriteLine("Entrada Liberada!");

}

else{

    Console.WriteLine("Entrada Não Liberada!");

}

### Operador AND – (&&) Significa E

No Operador AND ele precisa que todas as condições sejam verdadeiras

bool presencaMinina = true;

Console.WriteLine("Digite sua Nota1");

double nota1 = Convert.ToDouble(Console.ReadLine());

Console.WriteLine("Digite sua Nota2");

double nota2 = Convert.ToDouble(Console.ReadLine());

Console.WriteLine("Digite sua Nota3");

double nota3 = Convert.ToDouble(Console.ReadLine());

double media = (nota1 + nota2 + nota3) / 3;

if (presencaMinina && media >= 7.5 ){

    Console.WriteLine("Sua Media é " + media + " Aprovado ");

}else if (presencaMinina && media >= 5 && media < 7.5 ){

    Console.WriteLine("Sua Media é " + media + " Recuperação");

} else {

    Console.WriteLine("Sua Media é " + media + " Reprovado");

}

### Operador NOT (!) de negação

Ele inverte os valores booleano ele inverte uma condição

Exemplo se um resultado for true ele atribui false e se for false ele atribui true

bool choveu = true;

bool estaTarde = true;

//estou negando quando o ! esta na frente da variavel

if (!choveu && !estaTarde){

    Console.WriteLine("Vou Pedalar");

} else {

    Console.WriteLine("Vou Pedalar outro dia");

}

### Operadores Aritméticos e a Classe Math

Tabela

Descrição gerada automaticamente

Criando uma classe para os cálculos

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace Projeto\_CSharp.Moldes

{

    public class Calculadora //Classe

    {

        public void Somar(int x, int y)//Metodo

        {

                Console.WriteLine($"{x} + {y} = {x + y}");

        }

         public void Subtrair(int x, int y)//Metodo

        {

                Console.WriteLine($"{x} - {y} = {x - y}");

        }

         public void Multiplicar(int x, int y)//Metodo

        {

                Console.WriteLine($"{x} \* {y} = {x \* y}");

        }

         public void Dividir(int x, int y)//Metodo

        {

                Console.WriteLine($"{x} / {y} = {x / y}");

        }

    }

}

Chamando a classe criada

//Chamando a classe Calculadora instanciando

Calculadora calc = new Calculadora();

Console.WriteLine("Digite um numero inteiro");

int s = Int32.Parse(Console.ReadLine());

Console.WriteLine("Digite um numero inteiro");

int s1 = Int32.Parse(Console.ReadLine());

//chamando as Operações

calc.Somar(s, s1);

calc.Subtrair(s,s1);

calc.Multiplicar(s,s1);

calc.Dividir(s,s1);

### Usando Potência

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace Projeto\_CSharp.Moldes

{

    public class Calculadora //Classe

    {

        public void Potencia(int x, int y)

        {

            double potencia = Math.Pow(x, y);

            Console.WriteLine($"{x} ^ {y} = {potencia}" );

        }

    }

}

//Chamando a classe Calculadora instanciando

Calculadora calc = new Calculadora();

Console.WriteLine("Digite um numero inteiro");

int s = Int32.Parse(Console.ReadLine());

Console.WriteLine("Digite um numero inteiro");

int s1 = Int32.Parse(Console.ReadLine());

//chamando as Operações

calc.Potencia(s,s1);

### Funções Trigonométricas

Seno, Coseno e Tangente

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace Projeto\_CSharp.Moldes

{

    public class Calculadora //Classe

    {

        public void Seno(double angulo)

        {                    //angulo \* pi divido por pi

            double radiano = angulo \* Math.PI / 180; //Convertendo em radiano

            double seno = Math.Sin(radiano);    //quantos dígitos depois a virgula no método Math.Round

            Console.WriteLine($"Seno de {angulo}° = {Math.Round(seno,4)}");

        }

        public void Coseno(double angulo)

        {

            double radiano = angulo \* Math.PI / 180;

            double coseno = Math.Cos(radiano);

            Console.WriteLine($"Coseno de {angulo}° = {Math.Round(coseno,4)}");

        }

        public void Tangente(double angulo)

        {

            double radiano = angulo \* Math.PI / 180;

            double tangente = Math.Sin(radiano);

            Console.WriteLine($"Tangente de {angulo}° = {Math.Round(tangente,4)}");

        }

    }

}

Calculadora grau = new Calculadora();

Console.WriteLine("Digite um angulo");

int graus = Int32.Parse(Console.ReadLine());

grau.Seno(graus);

grau.Coseno(graus);

grau.Tangente(graus);

### Incremento e Decremento

//Icrementando e Demecremento

Console.WriteLine("Digite um numero para Incrementar e Decrementar");

int num = Int32.Parse(Console.ReadLine());

Console.WriteLine(num++ + " Incrementando + 1 = " + num );

Console.WriteLine(num-- + " Decrementando - 1 = " + num );

## Calculando a Raiz Quadrada

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace Projeto\_CSharp.Moldes

{

    public class Calculadora //Classe

    {

        public void RaizQuadrada(double x)

        {

            double raiz = Math.Sqrt(x);

            Console.WriteLine($"Raiz Quadrada de {x} = {raiz}");

        }

    }

}

//Raiz Quadrada

Calculadora RzQuadarda = new Calculadora();

Console.WriteLine("Digite um Numero para raiz quadrada");

int rQuadrada = Int32.Parse(Console.ReadLine());

RzQuadarda.RaizQuadrada(rQuadrada);

## Estrutura de Repetição

Diagrama

Descrição gerada automaticamente

### Laço FOR

//Laço For

Console.WriteLine("Digite um Numero para tabuada do ");

int tabuada = Int32.Parse(Console.ReadLine());

for (int contador = 0; contador <= 10; contador++){

    Console.WriteLine($"{tabuada} x {contador} = {tabuada \* contador}");

}

### Laço While

Console.WriteLine("Digite um Numero para tabuada com laço WHILE ");

int tabuada1 = Int32.Parse(Console.ReadLine());

int contador1 = 0;

while (contador1 <= 10){

    Console.WriteLine($"{tabuada1} x {contador1} = {tabuada1 \* contador1}");

    contador1++;

}

Parando o meu laço em um determinado numero interrompendo o fluxo de execução

Exemplo

Console.WriteLine("Digite um Numero para tabuada com laço WHILE ");

int tabuada1 = Int32.Parse(Console.ReadLine());

int contador1 = 0;

while (contador1 <= 10){

    Console.WriteLine($"{tabuada1} x {contador1} = {tabuada1 \* contador1}");

    contador1++;

if (contador == 6) // Aqui ele ira executar 5 vezes

{

Break;

}

}

### DO WHILE

int soma1 = 0, numero = 0;

do {

    Console.WriteLine("Digite um numero para somar e (0 para parar)");

    numero = Convert.ToInt32(Console.ReadLine());

    soma1 += numero;

} while(numero != 0);

Console.WriteLine($"Total da soma dos numeros digitados é: {soma1}");

Construindo um MENU

//Construindo um MENU

string? opcao;

bool  exibirMenu = true;

//while(true)

while(exibirMenu)

{

    Console.Clear();

    Console.WriteLine("Digite a sua opção:");

    Console.WriteLine("1 - Cadastrar Cliente");

    Console.WriteLine("2 - Buscar Cliente");

    Console.WriteLine("3 - Apagar Cliente");

    Console.WriteLine("4 - Encerrar");

   opcao = Console.ReadLine();

   switch (opcao)

   {

        case "1":

            Console.WriteLine("Cadastro de Cliente");

            break;

        case "2":

            Console.WriteLine("Buscar Cliente");

            break;

        case "3":

            Console.WriteLine("Apagar Cliente");

            break;

        case "4":

            Console.WriteLine("Encerrar");

            //Environment.Exit(0); //Aquie ele ira sair do meu programa

            exibirMenu = false;

            break;

        default:

            Console.WriteLine("Opção invaçida");

            break;

   }

}

Console.WriteLine("O Programa se Encerrou");

## Estrutura de um Programa em método principal

Arquivos de Projeto principais

.csproj: constem informações referente a um projeto (build, debug, versão)

.sln: Contem informações que carregam um agrupamento de projetos

Diagrama

Descrição gerada automaticamente

## Array e Listas

É uma estrutura de dados que armazena valores do mesmo tipo, com um tamanho fixo.

Int[] array = new int[4];

Int[] array = new int []{42, 75, 74, 61};

String[] nomes = {“jan”, “fev”};

Indice: é a posição de um determinado valor de um array, sempre começando com zero

Uma imagem contendo Diagrama

Descrição gerada automaticamente

//array e listas

int[] arrayInteiros = new int[3];

arrayInteiros[0] = 72;

arrayInteiros[1] = 64;

arrayInteiros[2] = 50;

for (int contador2 = 0; contador2 < arrayInteiros.Length; contador2++)

{

    Console.WriteLine($"Array Posição N° {contador2} - {arrayInteiros[contador2]}");

}

//Acessando um elemento do array

int elemento = arrayInteiros[2];

Console.WriteLine($"array de posição 2° equivale a dado de {elemento}");

Console.WriteLine(arrayInteiros[1]);

## Foreach

//Percorrendo o array com foreach

Console.WriteLine("Percorrendo o array com foreach");

int contadorForeach = 0;

foreach(int valor  in arrayInteiros)

{

    Console.WriteLine($"Posição {contadorForeach}° - {valor}");

    contadorForeach++;

}

### Aumentando o tamanho do array

//aumentando o tamanho do array

Array.Resize(ref arrayInteiros, arrayInteiros.Length + 1 );

### Copiando dados de uma array existe para um novo array

//copiando os dados de um array para um novo array

int [] arrayInteirosDobrado = new int[arrayInteiros.Length \* 2];

Array.Copy(arrayInteiros, arrayInteirosDobrado, arrayInteiros.Length);

### Listas

//Listas - A diferença entre o array é que não precisa passar o tamanho dessa lista

//Muito melhor de se trabalhar pois tem mais metados

List<string> listaString = new List<string>();

listaString.Add("SP");

listaString.Add("RJ");

listaString.Add("MG");

Console.WriteLine("Percorrendo uma lista com o for");

for (int contador3 = 0; contador3 < listaString.Count; contador3++ )

{

    Console.WriteLine($"Posição na lista N° {contador3} - {listaString[contador3]}");

}

Console.WriteLine("Percorrendo uma lista com o foreach");

int contadorForeachLista = 0;

foreach(string item in listaString)

{

    Console.WriteLine(($"Posição na lista N° {contadorForeachLista} - {item}"));

    contadorForeachLista++;

}

## Comentários de Códigos e Boas Práticas

Comentários

Os comentários servem para documentar o seu código, explicando um determinado método ou execução.

Auxiliam outros programadores a entender o que está acontecendo.

Tipos de Comentários

Comentário com //: é representado em uma única linha

Comentário com /\* --------\*/ : Permite escrever com varias linhas

Comentário com <summary> : Permite documentar classes métodos, parâmetros, etc.

Para comentar a Classe e Metodo usase o summary

Precisa habilitar uma opção

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Na classe só colocar ///

Texto

Descrição gerada automaticamente

Passando o mouse em cima da classe

Texto

Descrição gerada automaticamente

Texto

Descrição gerada automaticamente

Interface gráfica do usuário, Texto, Aplicativo, chat ou mensagem de texto

Descrição gerada automaticamente

# Propriedades, Métodos e Construtores

## Propriedades

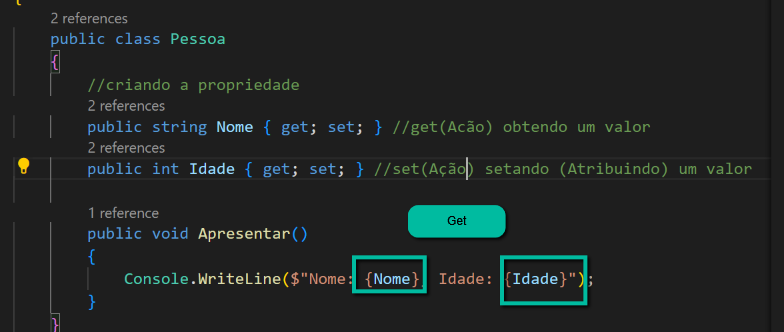
Uma Propriedade é um membro que oferece um mecanismo flexível para ler, gravar ou calcular o valor de um campo particular.

Texto

Descrição gerada automaticamente

### Ações GET e SET

Get – Obtendo um valor



Set – Atribuindo um valor

Tela preta com letras brancas

Descrição gerada automaticamente

### Validando o GET e SET

Interface gráfica do usuário, Texto, Aplicativo, chat ou mensagem de texto

Descrição gerada automaticamente

### Usando o Body Expressions

Serve quando temos uma linha de codigo

Texto

Descrição gerada automaticamente

## Métodos

Um Método é um bloco de código que contêm uma série de instruções.

Diagrama

Descrição gerada automaticamente

## Construtores

Os construtores permitem que o programador defina valores padrão, limite a instanciação e grave códigos flexíveis e fáceis de ler.

# Manipulando Valores com C#

## Concatenação de Strings

Podemos formatar valores em diversas representações. Essa formatação de valores é conhecida por concatenação ou interpolação.

 public void ListarAlunos()

        {

            Console.WriteLine($"Alunos do curso de: {Nome}");

            for (int count = 0; count < Alunos.Count; count++)

            {

             // Usando a Concatenação

              int exibirNumSequencia = count + 1;

              string texto = "N° " + exibirNumSequencia + " - " + Alunos[count].NomeCompleto;

              Console.WriteLine(texto);

            }

}

 public void ListarAlunos()

        {

            Console.WriteLine($"Alunos do curso de: {Nome}");

            for (int count = 0; count < Alunos.Count; count++)

            {

             // Usando interpolação de String

             string texto = $"N° {count + 1} - {Alunos[count].NomeCompleto}";

              Console.WriteLine(texto);

            }

        }

## Formatando Valores Monetários

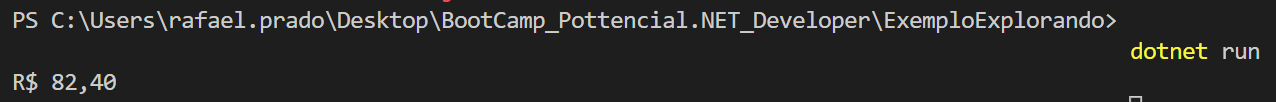
//Formantando Valores Monetarios

decimal valorMonetario = 82.40M;

Console.WriteLine($"{valorMonetario:C}");

Após a variável (valorMonetario) colocamos (:C) – E conforme a localização do nosso sistema do PC esta como brasil por isso aqui ele irá trazer na moeda real

Saida:



Alterando a localização do Código do sistema para exibir em outro valor de moeda

Interface gráfica do usuário, Texto

Descrição gerada automaticamente

Saida



Alterando a Localização da Cultura

Texto

Descrição gerada automaticamente

Formatação Personalizada

Texto

Descrição gerada automaticamente

Saida





Saida



Existem várias formas de formatar e representar um número

Representando Porcentagem

Interface gráfica do usuário, Texto

Descrição gerada automaticamente

Saida

Interface gráfica do usuário

Descrição gerada automaticamente

Outra forma de representar um número

Interface gráfica do usuário

Descrição gerada automaticamente

Saída



Formatando Tipo DateTime

Interface gráfica do usuário, Texto, Aplicativo, Site

Descrição gerada automaticamente

Saida

Placa azul com letras brancas em fundo preto

Descrição gerada automaticamente

Outro exemplo

Texto

Descrição gerada automaticamente

Saída

Texto

Descrição gerada automaticamente

Existem também várias formas de formatar data

## Exceções e Coleções

### Exceptions

- Os Recursos de manipulação de exceção da linguagem C# ajudam você a lidar com quaisquer situações excepcionais ou inesperadas que ocorram quando um programa for executado.

Uma exceção e quando o programa é programado para seguir um fluxo e ele não consegue ir adiante não sabe o que fazer para seguir o fluxo e sendo assim encerra o programa.

### Tratando a exceção

(Try catch ) – O try catch é para tratar essas exceções caso o programa não ocorra do jeito esperado para que ele possa seguir sem encerrar o programa e no catch podemos usar quantos for necessário para tratar as exceção

// Tratando a Exceção

try

{

    //Para ler o arquivo

    string[] linhas = File.ReadAllLines("Arquivos/arquivo\_Leitura.txt");

                        //ler todas as linha   //caminho do arquivo

    foreach (string linha in linhas)

    {

        Console.WriteLine(linha);

    }

} catch(Exception ex)

{

    Console.WriteLine($"Ocorreu uma exceção generica.{ex.Message}");

}

### Exceção genérica e específica

// Tratando a Exceção

try

{

    //Para ler o arquivo

    string[] linhas = File.ReadAllLines("Arquivos/arquivo\_Leitura.txt");

                        //ler todas as linha   //caminho do arquivo

    foreach (string linha in linhas)

    {

        Console.WriteLine(linha);

    }

} catch(FileNotFoundException ex) //Exceção especifica

{

    Console.WriteLine("Erro na leitura do arquivo. "

                       + "Arquivo não encontrado " + ex.Message);

} catch(Exception ex) //Exceção genérica

{

    Console.WriteLine($"Ocorreu uma exceção genérica.{ex.Message}");

}

### Bloco Finally

Utilizado em conjunto com Try Catch

Este é sempre executado sempre mesmo quando há exceções e quando não tem exceções.

### Usando o Throw

Texto

Descrição gerada automaticamente com confiança média

namespace ExemploExplorando.Models

{

    public class ExemploExcecao

    {

        public void Metodo1()

        {

            try

            {

                Metodo2();

            }catch(Exception ex)

            {

                Console.WriteLine("Exceção Tratada. " + ex.Message);

            }

        }

        public void Metodo2()

        {

            Metodo3();

        }

        public void Metodo3()

        {

            Metodo4();

        }

        public void Metodo4()

        {

            throw new Exception("Ocorreu uma exceção");

        }

    }

}

## Coleções – Queue(Fila)

Aqui ele respeita o método FIFO

Interface gráfica do usuário

Descrição gerada automaticamente com confiança média

//Filas no C#

Queue<int> fila = new Queue<int>();

//Adicionando na fila

fila.Enqueue(2); //1°

fila.Enqueue(4); //2°

fila.Enqueue(6); //3°

foreach(int item in fila)

{

    Console.WriteLine(item);

}

//Removendo da fila

//Sempre remove o 1° da fila

Console.WriteLine($"Removendo o elemento: {fila.Dequeue()}");

foreach(int item in fila)

{

    Console.WriteLine(item);

}

Saida

Interface gráfica do usuário, Texto

Descrição gerada automaticamente

## Coleções – Stack (pilha)

Aqui respeita o método LIFO

Uma imagem contendo Interface gráfica do usuário

Descrição gerada automaticamente

//Pilha no C#

Stack<int> pilha = new Stack<int>();

//adicionando elementos na pinha

pilha.Push(4);

pilha.Push(6);

pilha.Push(8);

pilha.Push(10);

foreach(int item in pilha)

{

    Console.WriteLine(item);

}

//Removendo elemento da pilha

Console.WriteLine("Removendo elemento " + pilha.Pop());

foreach(int item in pilha)

{

    Console.WriteLine(item);

}

Saida

Texto

Descrição gerada automaticamente

## Coleção Dictionary

Um dictionary é uma coleção de chave-valor para armazenar valores únicos sem uma ordem especifica.

//declarando um dictionary

Dictionary<string, string> estados = new Dictionary<string, string>();

//Adicionando elementos

estados.Add("SP","São Paulo");

estados.Add("RJ","Rio de Janeiro");

estados.Add("MG", "Minas Gerais");

foreach(var item in estados)

{

    Console.WriteLine($"Chave: {item.Key}, Valor: {item.Value}");

}

Console.WriteLine("--------------");

Console.WriteLine($"Removendo o estodo do RJ " + estados.Remove("RJ"));

//Removendo

foreach(var item in estados)

{

    Console.WriteLine($"Chave: {item.Key}, Valor: {item.Value}");

}

Console.WriteLine("--------------");

//Alterando valor

Console.WriteLine($"Alterando o estodo de SP ");

estados["SP"] = "São Paulo - Capital";

foreach(var item in estados)

{

    Console.WriteLine($"Chave: {item.Key}, Valor: {item.Value}");

}

Saida

Texto

Descrição gerada automaticamente

Verificando se a chave já existe

//Verificando se existe a chave

string chave = "SP";

Console.WriteLine($"Verificando o elemento: {chave}");

if (estados.ContainsKey(chave))

{

    Console.WriteLine($"Valor ja exite: {chave}");

}

else

{

    Console.WriteLine($"Valor não existe, Pode se adicionado: {chave}");

}

Saída

Texto

Descrição gerada automaticamente

Acessando um valor do Dictionary

Console.WriteLine(estados["MG"]);

