Agenda:

10/08 (4ª): Introdução e Fundamentos (C# e .NET)

12/08 (6ª): Programação orientada a objetos (classes, métodos e propriedades de acesso) - parte 1

15/08 (2ª): Programação orientada a objetos (classes, métodos e propriedades de acesso) – parte 2

17/08 (4ª): Interfaces, Herança e Polimorfismo

19/08 (6ª): Collections and Generics

Compilador C# online: https://www.onlinegdb.com/online_csharp_compiler

Códigos utilizados durante as aulas

Fundamentos
- Primeiro programa
- Variáveis e Constantes
- Inferência
- Interpolação
- Notação Ponto
- Lendo dados
- Formatando Números
- Conversões
- Operadores Aritméticos
- Operadores Relacionais
- Operadores Lógicos
- Operadores de atribuição
- Operadores Unários
- Operador Ternário

Estrururas de Controle
- If/else/else if
- Switch

Uso do try/catch/finally
Excessões Personalizadas

Fundamentos

1. *Primeiro programa*

```
using System;

namespace CursoCSharp.Fundamentos {
    class PrimeiroPrograma {
        public static void Executar() {
            Console.Write("Primeiro ");
            Console.WriteLine("Programa");
            Console.WriteLine("Terminou!");
        }
    }
}
```

2. *Variáveis e Constantes*

```
  // area da circunferencia
double raio = 4.5;
const double PI = 3.14;

raio = 5.5;
// PI = 3.1415;

double area = PI * raio * raio;
Console.WriteLine(area);
Console.WriteLine("Área é " + area);

// Tipos internos

bool estaChovendo = true;
Console.WriteLine("Está chovendo " + estaChovendo);

byte idade = 45;
```

```
Console.WriteLine("Idade " + idade);

sbyte saldoDeGols = sbyte.MinValue;
Console.WriteLine("Saldo de Gols " + saldoDeGols);

short salario = short.MaxValue;
Console.WriteLine("Salário " + salario);

int menorValorInt = int.MinValue; // Mais usado dos inteiros!
Console.WriteLine("Menor int " + menorValorInt);

uint populacaoBrasileira = 207_600_000;
Console.WriteLine("População Brasileira " + populacaoBrasileira);

long menorValorLong = long.MinValue;
Console.WriteLine("Menor long " + menorValorLong);
ulong populacaoMundial = 7_600_000_000;
Console.WriteLine("População Mundial " + populacaoMundial);

float precoComputador = 1299.99F;
Console.WriteLine("Preço Computador " + precoComputador);

double valorDeMercadoDaApple = 1_000_000_000_000.00; // Mais usado dos reais!
Console.WriteLine("Valor Apple " + valorDeMercadoDaApple);

decimal distanciaEntreEstrelas = decimal.MaxValue;
Console.WriteLine("Distância entre Estrelas " + distanciaEntreEstrelas);

char letra = 'b';
Console.WriteLine("Letra " + letra);

string texto = "Seja bem vindo ao Curso de C#!";
Console.WriteLine(texto);
```

3. *Inferência*

```
        var nome = "Leonardo";
        // nome = 123;
        Console.WriteLine(nome);

        // int idade;
        var idade = 32;
```

```
Console.WriteLine(idade);

int a;
a = 3;

int b = 2;

Console.WriteLine(a + b);
```

4. Interpolação

```
string nome = "Notebook Gamer";
string marca = "Dell";
double preco = 5800.00;

Console.WriteLine("O " + nome + " da marca "
    + marca + " custa " + preco + ".");
Console.WriteLine("O {0} da marca {1} custa {2}.",
    nome, marca, preco);
Console.WriteLine($"A marca {marca} é legal!");
Console.WriteLine($"1 + 1 = {1 + 1}!");
```

5. Notação Ponto

```
var saudacao = "olá".ToUpper().Insert(3, " World!")
    .Replace("World!", "Mundo!");
Console.WriteLine(saudacao);

Console.WriteLine("Teste".Length);

string valorImportante = null;
Console.WriteLine(valorImportante?.Length);
```

6. Lendo dados

```
Console.Write("Qual é o seu nome? ");
string nome = Console.ReadLine();

Console.Write("Qual é a sua idade? ");
```

```
int idade = int.Parse(Console.ReadLine());

Console.Write("Qual é o seu salário? ");
double salario = double.Parse(Console.ReadLine(),
    CultureInfo.InvariantCulture);

Console.WriteLine($"{nome} {idade} R${salario}");
```

7. *Formatando Números*

```
double valor = 15.175;
Console.WriteLine(valor.ToString("F1"));
Console.WriteLine(valor.ToString("C"));
Console.WriteLine(valor.ToString("P"));
Console.WriteLine(valor.ToString("#.##"));

CultureInfo cultura = new CultureInfo("en-US");
Console.WriteLine(valor.ToString("C0", cultura));

int inteiro = 256;
Console.WriteLine(inteiro.ToString("D10"));
```

8. *Conversões*

```
int inteiro = 10;
double quebrado = inteiro;
Console.WriteLine(quebrado);

double nota = 9.7;
int notaTruncada = (int) nota;
Console.WriteLine("Nota truncada: {0}", notaTruncada);

Console.Write("Digite sua idade: ");
string idadeString = Console.ReadLine();
```

```
int idadeInteiro = int.Parse(idadeString);
Console.WriteLine("Idade inserida: {0}", idadeInteiro);

idadeInteiro = Convert.ToInt32(idadeString);
Console.WriteLine("Resultado: {0}", idadeInteiro);

Console.Write("Digite o primeiro número: ");
string palavra = Console.ReadLine();
int numero1;
int.TryParse(palavra, out numero1);
Console.WriteLine("Resultado 1: {0}", numero1);

Console.Write("Digite o segundo número: ");
int.TryParse(Console.ReadLine(), out int numero2);
Console.WriteLine("Resultado 2: {0}", numero2);
```

Próximos tópicos serão vistos na aula do dia 24/06

9. *Operadores Aritméticos*

```
// Preço Desconto
var preco = 1000.0;
var imposto = 355;
var desconto = 0.1;

double total = preco + imposto;
var totalComDesconto = total - (total * desconto);
Console.WriteLine("O preço final é {0}", totalComDesconto);

// IMC
double peso = 91.2;
double altura = 1.82;
double imc = peso / Math.Pow(altura, 2);
Console.WriteLine($"IMC é {imc}.");

// Número Par/Impar
int par = 24;
int impar = 55;
Console.WriteLine("{0}/2 tem resto {1}", par, par % 2);
Console.WriteLine("{0}/2 tem resto {1}", impar, impar % 2);
```

10. Operadores Relacionais

```
// double nota = 6.0;
Console.Write("Digite a nota: ");
double.TryParse(Console.ReadLine(), out double nota);
double notaDeCorte = 7.0;

Console.WriteLine("Nota inválida? {0}", nota > 10.0);
Console.WriteLine("Nota inválida? {0}", nota < 0.0);
Console.WriteLine("Perfeito? {0}", nota == 10.0);
Console.WriteLine("Tem como melhorar? {0}", nota != 10.0);
Console.WriteLine("Passou por média? {0}", nota >= notaDeCorte);
Console.WriteLine("Recuperação? {0}", nota < notaDeCorte);
Console.WriteLine("Reprovado? {0}", nota <= 3.0);
```

11. Operadores Lógicos

```
var executouTrabalho1 = false;
var executouTrabalho2 = false;

bool comprouTv50 = executouTrabalho1 && executouTrabalho2;
Console.WriteLine("Comprou a Tv 50? {0}", comprouTv50);

var comprouSorvete = executouTrabalho1 || executouTrabalho2;
Console.WriteLine("Comprou o sorvete? {0}", comprouSorvete);

var comprouTv32 = executouTrabalho1 ^ executouTrabalho2;
Console.WriteLine("Comprou a Tv 32? {0}", comprouTv32);

Console.WriteLine("Mais saudável? {0}", !comprouSorvete);
```

12. Operadores de atribuição

```
var num1 = 3;
num1 = 7;
num1 += 10; // num1 = num1 + 10;
num1 -= 3; // num1 = num1 - 3;
num1 *= 5; // num1 = num1 * 5;
num1 /= 2; // num1 = num1 / 2;

Console.WriteLine(num1);

int a = 1;
int b = a;

a++; // a = a + 1;
b--; // b = b - 1;

Console.WriteLine($"{a} {b}");

// Não se preocupe com o código
dynamic c = new System.Dynamic.ExpandoObject();
c.nome = "João";

dynamic d = c;
d.nome = "Maria";

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

13. Operadores Unários

```
var valorNegativo = -5;
var numero1 = 2;
var numero2 = 3;
var booleano = true;

Console.WriteLine(-valorNegativo);
Console.WriteLine(!booleano);

numero1++;
```

```
Console.WriteLine(numero1);

--numero1;
Console.WriteLine(numero1);

Console.WriteLine(numero1++ == --numero2);
Console.WriteLine($"{numero1} {numero2}");
```

14. Operador Ternário

```
var nota = 9.0;
bool bomComportamento = true;
string resultado = nota >= 7.0 && bomComportamento
    ? "Aprovado" : "Reprovado";
Console.WriteLine(resultado);
```

Estrururas de Controle

15. If/else/else if

```
Console.Write("Digite a nota do aluno: ");

string entrada = Console.ReadLine();
Double.TryParse(entrada, out double nota);

if(nota >= 9.0) {
    Console.WriteLine("Quadro de honra!");
} else if(nota >= 7.0) {
    Console.WriteLine("Aprovado!");
} else if(nota >= 5.0) {
    Console.WriteLine("Recuperação");
} else {
    Console.WriteLine("Te vejo na proxima...");
}
```

```
                Console.WriteLine("Fim!!");
```

16. Switch

```
                Console.Write("Avalie meu atendimento com uma nota de 1 a 5:
");
                int.TryParse(Console.ReadLine(), out int nota);

                switch (nota) {
                    case 0:
                        Console.WriteLine("Péssimo");
                        break;
                    case 1:
                    case 2:
                        Console.WriteLine("Ruim");
                        break;
                    case 3:
                        Console.WriteLine("Regular");
                        break;
                    case 4:
                        Console.WriteLine("Bom");
                        break;
                    case 5:
                        Console.WriteLine("Ótimo");
                        Console.WriteLine("Parabéns!");
                        break;
                    default:
                        Console.WriteLine("Nota inválida");
                        break;
                }

                Console.WriteLine("Obrigado por responder!");
```

17. While

```
                int palpite = 0;
                Random random = new Random();

                int numeroSecreto = random.Next(1, 16);
```

```
bool numeroEncontrado = false;
int tentativasRestantes = 5;
int tentativas = 0;

while (tentativasRestantes > 0 && !numeroEncontrado) {
   Console.Write("Insira o seu palpite: ");
   string entrada = Console.ReadLine();
   int.TryParse(entrada, out palpite);

   tentativas++;
   tentativasRestantes--;

   if (numeroSecreto == palpite) {
       numeroEncontrado = true;
       var corAnterior = Console.BackgroundColor;
       Console.BackgroundColor = ConsoleColor.Green;
       Console.WriteLine("Numero encontrado em {0} tentativas",
           tentativas);
       Console.BackgroundColor = corAnterior;
   } else if (palpite > numeroSecreto) {
       Console.WriteLine("Menor... Tente novamente!");
       Console.WriteLine("Tentativas restantes: {0}",
           tentativasRestantes);
   } else {
       Console.WriteLine("Maior... Tente novamente!");
       Console.WriteLine("Tentativas restantes: {0}",
           tentativasRestantes);
   }
}
```

18. Do While

```
string entrada;

do {
    Console.WriteLine("Qual o seu nome?");
    entrada = Console.ReadLine();

    Console.WriteLine("Seja bem-vindo {0}", entrada);
    Console.WriteLine("Deseja continuar?(S/N)");
    entrada = Console.ReadLine();
} while (entrada.ToLower() == "s");
```

19. For

```
double somatorio = 0;
string entrada;

Console.Write("Informe o tamano da turma: ");
entrada = Console.ReadLine();
int.TryParse(entrada, out int tamanhoTurma);

for (int i = 1; i <= tamanhoTurma; i++) {
    Console.Write("Informe a nota do aluno {0}: ", i);
    entrada = Console.ReadLine();
    double.TryParse(entrada, out double notaAtual);

    somatorio += notaAtual;
}

double media = tamanhoTurma > 0 ? somatorio / tamanhoTurma : 0;
Console.WriteLine("Media da turma: {0}", media);
```

20. For each

```
var palavra = "Opa!";

foreach (var letra in palavra) {
    Console.WriteLine(letra);
}

var alunos = new string[] { "Ana", "Bia", "Carlos" };

foreach (string aluno in alunos) {
    Console.WriteLine(aluno);
```

```
}
```

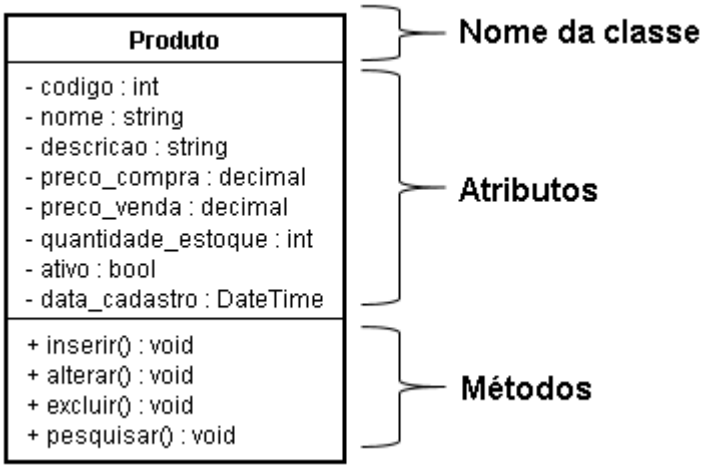
21. Break

```
Random random = new Random();
int numero = random.Next(1, 51);

Console.WriteLine("O número que queremos é {0}.", numero);

for (int i = 1; i <= 50; i++) {
    Console.Write("{0} é o numero que queremos? ", i);
    if (i == numero) {
        Console.WriteLine("Sim!");
        break;
    } else {
        Console.WriteLine("Não!");
    }
}

Console.WriteLine("Fim!");
```

22. Continue

```
int intervalo = 50;
        Console.WriteLine("Numeros pares entre 1 e {0}!",
intervalo);

for (int i = 1; i <= intervalo; i++) {
    if (i % 2 == 1) {
        continue;
    }

    Console.Write(i + " ");
}
```

Programação Orientada a Objetos - Parte 1 e Parte 2 - Classes e Métodos



Objetos (Instâncias) da classe Cliente

23. Classes vs Objeto, atributos e métodos

Classe

```
class Pessoa
{
    public string Nome; //atributos
```

```
        public int Idade;

        public string Apresentar() {  //métodos
            return string.Format(
                $"Olá! Me chamo {Nome} e tenho {Idade} anos.");
        }

        public void ApresentarNoConsole() {
            Console.WriteLine(Apresentar());
        }

        public void Zerar() {
            Nome = "";
            Idade = 0;
        }
    }
```

Instância da classe

```
            Pessoa sicrano = new Pessoa();
            sicrano.Nome = "Renato";
            sicrano.Idade = 21;

            // Console.WriteLine($"{sicrano.Nome} tem {sicrano.Idade} anos.");

            sicrano.ApresentarNoConsole();
            sicrano.Zerar();
            sicrano.ApresentarNoConsole();

            var fulano = new Pessoa();
            fulano.Nome = "Beto";
            fulano.Idade = 21;

            var apresentacaoDoFulano = fulano.Apresentar();
            Console.WriteLine(apresentacaoDoFulano.Length);
            Console.WriteLine(apresentacaoDoFulano);
```

24. Construtores

```
class Carro {
    public string Modelo;
    public string Fabricante;
    public int Ano;

    public Carro(string modelo, string fabricante, int ano) {
        Modelo = modelo;
        Fabricante = fabricante;
        Ano = ano;
    }

    public Carro() {

    }
}
```

```
            var carro1 = new Carro();
              carro1.Fabricante = "BMW";
              carro1.Modelo = "325i";
              carro1.Ano = 2017;
              Console.WriteLine(
                  $"{carro1.Fabricante} {carro1.Modelo} {carro1.Ano}");

              var carro2 = new Carro("Ka", "Ford", 2018);
              Console.WriteLine(
                  $"{carro2.Fabricante} {carro2.Modelo} {carro2.Ano}");

              var carro3 = new Carro() {
                  Fabricante = "Fiat",
                  Modelo = "Uno",
                  Ano = 2019
              };
              Console.WriteLine(
                  $"{carro3.Fabricante} {carro3.Modelo} {carro3.Ano}");
```

25. Métodos Com Retorno #01

```
        class CalculadoraComum {
```

```
        public int Somar(int a, int b) {
            return a + b;
        }

        public int Subtrair(int a, int b) {
            return a - b;
        }

        public int Multiplicar(int a, int b) {
            return a * b;
        }
    }
```

```
            var calculadoraComum = new CalculadoraComum();
            var resultado = calculadoraComum.Somar(5, 5);

            Console.WriteLine(resultado);
            Console.WriteLine(calculadoraComum.Subtrair(2, 7));
            Console.WriteLine(calculadoraComum.Multiplicar(4, 4));
```

26. Métodos Com Retorno #02

```
    class CalculadoraCadeia {
        int memoria;

        public CalculadoraCadeia Somar(int a) {
            memoria += a;
            return this;
        }

        public CalculadoraCadeia Multiplicar(int a) {
            memoria *= a;
            return this;
        }

        public CalculadoraCadeia Limpar() {
```

```
            memoria = 0;
            return this;
        }

        public CalculadoraCadeia Imprimir() {
            Console.WriteLine(memoria);
            return this;
        }

        public int Resultado() {
            return memoria;
        }
    }
```

```
            var calculadoraCadeia = new CalculadoraCadeia();
            calculadoraCadeia.Somar(3).Multiplicar(3).Imprimir()
                .Limpar().Imprimir();

            resultado = calculadoraCadeia.Somar(3).Multiplicar(2).Resultado();
            Console.WriteLine(resultado);
```

27. Métodos Estáticos

```
    public class CalculadoraEstatica {

        // Método de Classe ou Método estático!!!
        public static int Multiplicar(int a, int b) {
            return a * b;
        }

        // Método de instância!!!
        public int Somar(int a, int b) {
            return a + b;
        }
    }
```

```
            var resultado = CalculadoraEstatica.Multiplicar(2, 2);
            Console.WriteLine("Resultado: {0}", resultado);

            CalculadoraEstatica calc = new CalculadoraEstatica();
            Console.WriteLine(calc.Somar(2, 2));
```

28. Atributos Estáticos

```
    public class Produto {
        public string Nome;
        public double Preco;
        public static double Desconto = 0.1;

        public Produto(string nome, double preco, double desconto) {
            Nome = nome;
            Preco = preco;
            Desconto = desconto;
        }

        public Produto() {

        }

        public double CalcularDesconto() {
            return Preco - Preco * Desconto;
        }
    }
```

```
            var produto1 = new Produto("Caneta", 3.2, 0.1);
```

```
            var produto2 = new Produto() {
                Nome = "Borracha",
                Preco = 5.3
            };

            Produto.Desconto = 0.5;

            Console.WriteLine("Preço com desconto: {0}",
                produto1.CalcularDesconto());
            Console.WriteLine("Preço com desconto: {0}",
                produto2.CalcularDesconto());

            Produto.Desconto = 0.02;

            Console.WriteLine("Preço com desconto: {0}",
                produto1.CalcularDesconto());
            Console.WriteLine("Preço com desconto: {0}",
                produto2.CalcularDesconto());
```

29. Desafio Acessar Atributo

```
class Programa
{

    int a = 42; //Não pode haver modificação desta linha no desafio
    static void Main(string[] args)
    {
        // Desafio: acesse a variável a dentro do método Main.



    }
}
```

30. Parâmetros Variáveis

```
namespace ClassesEMetodos
{
    class Params
    {
        public static void Recepcionar(params string[] pessoas) {
            foreach (var pessoa in pessoas) {
                Console.WriteLine("Olá {0}", pessoa);
            }
        }

    }

    class Program
    {
        static void Main(string[] args)
        {
             Params.Recepcionar("Pedro", "Manu", "Roger","Ana", "Bia");
        }
    }
}
```

31. Parâmetros Nomeados

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

namespace ClassesEMetodos
{
    class ParametrosNomeados
    {
        public static void Formatar(int dia, int mes, int ano) {
            Console.WriteLine("{0:D2}/{1:D2}/{2}", dia, mes, ano);
        }

    }

    class Program
    {
        static void Main(string[] args)
        {
             ParametrosNomeados.Formatar(mes: 1, dia: 6, ano: 1996);
        }
    }
}
```

32. Getters & Setters

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

namespace ClassesEMetodos
{
    public class Moto {
        private string Marca;
        private string Modelo;
        private uint Cilindrada;

        public Moto(string marca, string modelo, uint cilindrada) {
            //Marca = marca;
            //Modelo = modelo;
            //Cilindrada = cilindrada;

            SetMarca(marca);
            SetModelo(modelo);
            SetCilindrada(cilindrada);
        }

        public Moto() {

        }

        public string GetMarca() {
            return Marca;
        }

        public void SetMarca(string marca) {
            Marca = marca;
        }

        public string GetModelo() {
            return Modelo;
        }

        public void SetModelo(string modelo) {
            Modelo = modelo;
        }
```

```
            public uint GetCilindrada() {
                return Cilindrada;
            }

            public void SetCilindrada(uint cilindrada) {
                // 1 Opção
                //if(cilindrada > 0) {
                //    Cilindrada = cilindrada;
                //}

                // 2 Opção
                //Cilindrada = Math.Abs(cilindrada);

                Cilindrada = cilindrada;
            }
        }

        class GetSet
        {
            static void Main(string[] args) {
                var moto1 = new Moto("Kawasaki", "Ninja ZX-6R", 636);

                Console.WriteLine(moto1.GetMarca());
                Console.WriteLine(moto1.GetModelo());
                Console.WriteLine(moto1.GetCilindrada());

                var moto2 = new Moto();
                moto2.SetMarca("Honda");
                moto2.SetModelo("CG Titan");
                moto2.SetCilindrada(150);
                            Console.WriteLine(moto2.GetMarca()  +  "  "  +
moto2.GetModelo()
                    + " " + moto2.GetCilindrada());
            }
        }
    }
```

33. Propriedades

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

namespace CursoCSharp.ClassesEMetodos
{
    public class CarroOpcional {
        double desconto = 0.1;

        string nome;
        public string Nome {
            get {
                return "Opcional: " + nome;
            }
            set {
                nome = value;
            }
        }

        // Propriedade autoimplementada
        public double Preco { get; set; }
        // Somente leitura
        public double PrecoComDesconto {
            get => Preco - (desconto * Preco); // Lambda
            //get {
            //    return Preco - (desconto * Preco);
            //}
```

```
        }

        public CarroOpcional() {

        }

        public CarroOpcional(string nome, double preco) {
            Nome = nome;
            Preco = preco;
        }
    }

    class Props
    {
         static void Main(string[] args) {
            var op1 = new CarroOpcional("Ar Condicionado", 3499.9);
            Console.WriteLine(op1.PrecoComDesconto);

            // op1.PrecoComDesconto = 3000;

            var op2 = new CarroOpcional();
            op2.Nome = "Direção Elétrica";
            op2.Preco = 2349.9;

            Console.WriteLine(op1.Nome);
            Console.WriteLine(op1.Preco);

            Console.WriteLine(op2.Nome);
            Console.WriteLine(op2.Preco);
            Console.WriteLine(op2.PrecoComDesconto);
        }
    }
}
```

34. Atributos Readonly

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

namespace ClassesEMetodos
{
    public class Cliente {
        public string Nome;
        public readonly DateTime Nascimento; //setado pelo construtor

        public Cliente(string nome, DateTime nascimento) {
            Nome = nome;
            Nascimento = nascimento;
            Nascimento = new DateTime(2020, 10, 10);
        }

        public string GetDataDeNascimento() {
            return String.Format("{0}/{1}/{2}", Nascimento.Day,
                Nascimento.Month, Nascimento.Year);
        }
    }

    class Readonly
    {
        static void Main(string[] args) {
            var novoCliente = new Cliente("Ana Silva",
                new DateTime(1987, 5, 22));
```

```
            Console.WriteLine(novoCliente.Nome);
            Console.WriteLine(novoCliente.GetDataDeNascimento());

            // novoCliente.Nascimento = new DateTime(2020, 10, 10);
        }
    }
}
```

35. Enumerações (Enum)

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

namespace ClassesEMetodos
{
    public enum Genero { Acao, Aventura, Terror, Animacao, Comedia };

    public class Filme {
        public string Titulo;
        public Genero GeneroDoFilme;
    }

    class ExemploEnum
    {
        static void Main(string[] args)  {
            int id = (int)Genero.Acao;
            Console.WriteLine(id);

            var filmeParaFamilia = new Filme();
            filmeParaFamilia.Titulo = "De volta para o futuro";
            filmeParaFamilia.GeneroDoFilme = Genero.Aventura;

            Console.WriteLine("{0} é {1}!", filmeParaFamilia.Titulo,
                filmeParaFamilia.GeneroDoFilme);
        }
    }
}
```

36. Struct

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

namespace ClassesEMetodos
{
    interface Ponto { // falaremos sobre interface em breve
        void MoverNaDiagonal(int delta);
    }

    struct Coordenada: Ponto {
        public int X;
        public int Y;

        public Coordenada(int x, int y) {
            X = x;
            Y = y;
        }

        public void MoverNaDiagonal(int delta) {
            X += delta;
            Y += delta;
        }
    }

    class ExemploStruct
    {
        static void Main(string[] args) {
            Coordenada coordenadaInicial;
            coordenadaInicial.X = 2;
```

```
            coordenadaInicial.Y = 2;

            Console.WriteLine("Coordenada Inicial:");
            Console.WriteLine("X = {0}", coordenadaInicial.X);
            Console.WriteLine("Y = {0}", coordenadaInicial.Y);

            var coordenadaFinal = new Coordenada(x: 9, y: 1);
            coordenadaFinal.MoverNaDiagonal(10);

            Console.WriteLine("Coordenada Final:");
            Console.WriteLine("X = {0}", coordenadaFinal.X);
            Console.WriteLine("Y = {0}", coordenadaFinal.Y);
        }
    }
}
```

37. Class vs Struct

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

namespace ClassesEMetodos
{
    public struct SPonto {
        public int X;
        public int Y;
    }

    public class CPonto {
        public int X;
        public int Y;
    }

    class StructVsClasse
    {
        static void Main(string[] args) {
            SPonto ponto1 = new SPonto { X = 1, Y = 3 };
            SPonto copiaPonto1 = ponto1; // Atribuição por VALOR!!!
            ponto1.X = 3;

            Console.WriteLine("Ponto 1 X:{0}", ponto1.X);
```

```
            Console.WriteLine("Copia Ponto 1 X:{0}", copiaPonto1.X);

            CPonto ponto2 = new CPonto { X = 2, Y = 4 };
            CPonto copiaPonto2 = ponto2; // Atribuição por REFERÊNCIA!!!
            ponto2.X = 4;

            Console.WriteLine("Ponto 2 X:{0}", ponto2.X);
            Console.WriteLine("Copia Ponto 2 X:{0}", copiaPonto2.X);
        }
    }
}
```

38. Atribuição por Valor vs Referência

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

namespace ClassesEMetodos
{
    public class Dependente {
        public string Nome;
        public int Idade;
    }

    class ValorVsReferencia
    {
        static void Main(string[] args) {
            int numero = 3;
            int copiaNumero = numero;
            Console.WriteLine($"{numero} {copiaNumero}");

            numero++;
            Console.WriteLine($"{numero} {copiaNumero}");

            Dependente dep = new Dependente {
                Nome = "Beto",
                Idade = 20
            };

            Dependente copiaDep = dep;
```

```
                Console.WriteLine($"{dep.Nome} {copiaDep.Nome}");
                Console.WriteLine($"{dep.Idade} {copiaDep.Idade}");

                copiaDep.Nome = "Renato";
                dep.Idade = 21;

                Console.WriteLine($"{dep.Nome} {copiaDep.Nome}");
                Console.WriteLine($"{dep.Idade} {copiaDep.Idade}");
            }
        }
    }
```

39. Parâmetros por Referência (Ref/Out)

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

    namespace ClassesEMetodos
    {
        class ParametrosPorReferencia
        {
            public static void AlterarRef(ref int numero) {
                numero = numero + 1000;
            }

             public static void AlterarOut(out int numero1, out int numero2)
{
                numero1 = 0;
                numero2 = 0;
                numero1 = numero1 + 15;
                numero2 = numero2 + 30;
            }

            static void Main(string[] args) {
                int a = 3;
                AlterarRef(ref a);
                Console.WriteLine(a);
```

```
                // int b;
                AlterarOut(out int b, out int c);
                Console.WriteLine($"{b} {c}");
            }
        }
    }
```

40. Parâmetro com Valor Padrão

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

namespace CursoCSharp.ClassesEMetodos
{
    class ParametroPadrao
    {
        public static int Somar(int a = 1, int b = 1) {
            return a + b;
        }

        static void Main(string[] args) {
            Console.WriteLine(Somar(10, 23));
            Console.WriteLine(Somar(50));
            Console.WriteLine(Somar());
            Console.WriteLine(Somar(b: 7));
        }
    }
}
```

41. Herança

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

namespace HerancaNS
{
    public class Carro {
        protected readonly int VelocidadeMaxima;
        int VelocidadeAtual;

        public Carro(int velocidadeMaxima) {
            VelocidadeMaxima = velocidadeMaxima;
        }

        protected int AlterarVelocidade(int delta) {
            int novaVelocidade = VelocidadeAtual + delta;

            if (novaVelocidade < 0) {
                VelocidadeAtual = 0;
            } else if (novaVelocidade > VelocidadeMaxima) {
                VelocidadeAtual = VelocidadeMaxima;
            } else {
                VelocidadeAtual = novaVelocidade;
            }

            return VelocidadeAtual;
        }

        public virtual int Acelerar() {
```

```
            return AlterarVelocidade(5);
        }

        public int Frear() {
            return AlterarVelocidade(-5);
        }
    }




    public class Uno : Carro {
        public Uno() : base(200) {

        }
    }

    public class Ferrari : Carro {
        public Ferrari() : base(350) {

        }

        public override int Acelerar() {
            return AlterarVelocidade(15);
        }

        // Oculta o método da classe Pai
        public new int Frear() {
            return AlterarVelocidade(-15);
        }
    }

    class Heranca
    {
        public static void Main() {
            Console.WriteLine("Uno...");
            Uno carro1 = new Uno();
            Console.WriteLine(carro1.Acelerar());
            Console.WriteLine(carro1.Acelerar());
            Console.WriteLine(carro1.Frear());
            Console.WriteLine(carro1.Frear());
            Console.WriteLine(carro1.Frear());
```

```
            Console.WriteLine("Ferrari...");
            Ferrari carro2 = new Ferrari();
            Console.WriteLine(carro2.Acelerar());
            Console.WriteLine(carro2.Acelerar());
            Console.WriteLine(carro2.Frear());
            Console.WriteLine(carro2.Frear());
            Console.WriteLine(carro2.Frear());

            Console.WriteLine("Ferrari com tipo Carro...");
            Carro carro3 = new Ferrari();
            Console.WriteLine(carro3.Acelerar());
            Console.WriteLine(carro3.Acelerar());
            Console.WriteLine(carro3.Frear());
            Console.WriteLine(carro3.Frear());
            Console.WriteLine(carro3.Frear());

            Console.WriteLine("Uno com tipo Carro...");
            carro3 = new Uno(); // Polimorfismo
            Console.WriteLine(carro3.Acelerar());
            Console.WriteLine(carro3.Acelerar());
            Console.WriteLine(carro3.Frear());
            Console.WriteLine(carro3.Frear());
            Console.WriteLine(carro3.Frear());
        }
    }
}
```

42. Construtor This

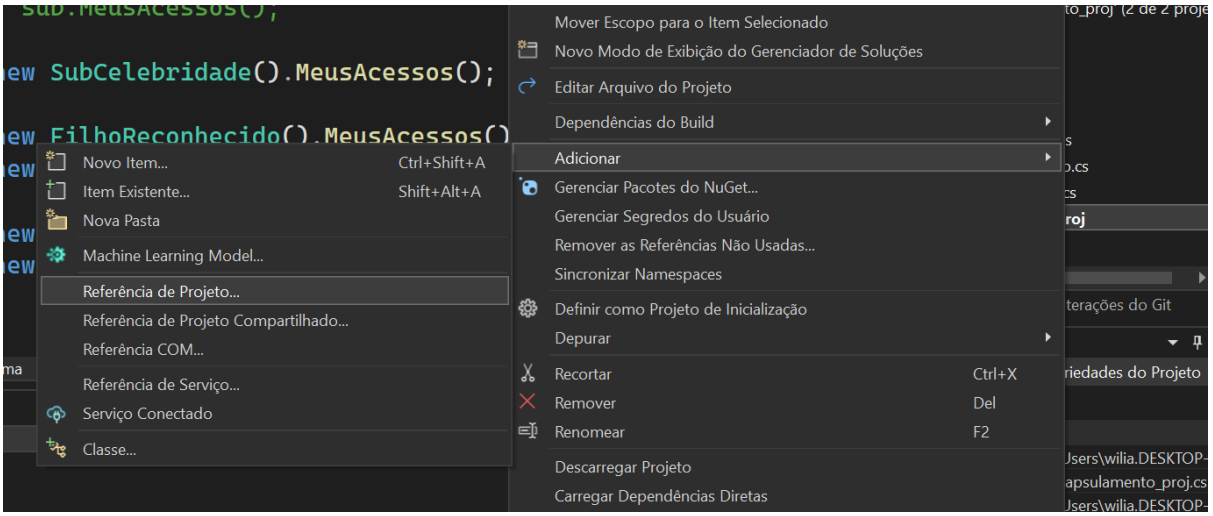
```
namespace ConstrutorThis
{
    public class Animal {
        public string Nome { get; set; }

        public Animal(string nome) {
            Nome = nome;
        }

    }


    public class Cachorro : Animal {
        public double Altura { get; set; }

        public Cachorro(string nome):base(nome) {
            Console.WriteLine($"Cachorro {nome} inicializado");
        }

        public Cachorro(string nome, double altura) : this(nome)
        {
            Altura = altura;
        }
```

```
        public override string ToString() {
            return $"{Nome} tem {Altura} cm de altura!";

        }
    }


    public class Golden : Cachorro
    {
        public double Altura { get; set; }

        public Golden(string nome) : base(nome)
        {
            Console.WriteLine($"Golden {nome} inicializado");
        }

        public Golden(string nome, double altura) : this(nome)
        {
            Altura = altura;
        }

        public override string ToString()
        {
            return $"{Nome} tem {Altura} cm de altura!";

        }
    }


    internal class Program
    {
        static void Main(string[] args)
        {
            var spike = new Cachorro("Spike");
            var max = new Cachorro("Max", 40.0);

            var nina = new Golden("Nina", 50.0);


            Console.WriteLine(spike);
            //Console.WriteLine(max.ToString());
```

```
        }
    }
}
```

43. Encapsulamento

Crie um projeto do tipo “Biblioteca de Classes” na mesma solução do projeto “Aplicativo de Console”. Clique com o botão direito no aplicativo do projeto(console) e referencie a biblioteca de classes criada.



Classes no projeto do tipo “Biblioteca de Classes”:

```
namespace Encapsulamento
{
    public class SubCelebridade
    {
        // Todos
        public string InfoPublica = "Tenho um instagram!";
```

```
        // herança
        protected string CorDoOlho = "Verde";

        // mesmo projeto (assembly)
        internal ulong NumeroCelular = 5511999999999;

        // herança ou mesmo projeto
        protected internal string JeitoDeFalar = "Uso muitas gírias";

        // mesma class ou herança no mesmo projeto (c# >= 7.2)
        private protected string SegredoFamilia = "Bla bla";

        // private é o padrão
        bool UsaMuitoPhotoshop = true;

        public void MeusAcessos()
        {
            Console.WriteLine("SubCelebridade...");

            Console.WriteLine(InfoPublica);
            Console.WriteLine(CorDoOlho);
            Console.WriteLine(NumeroCelular);
            Console.WriteLine(JeitoDeFalar);
            Console.WriteLine(SegredoFamilia);
            Console.WriteLine(UsaMuitoPhotoshop);
        }
    }


}
```

```
namespace Encapsulamento
{
    public class FilhoReconhecido : SubCelebridade
    {
        public new void MeusAcessos()
        {
            Console.WriteLine("FilhoReconhecido...");

            Console.WriteLine(InfoPublica);  //public
            Console.WriteLine(CorDoOlho);    //protected (herança)
```

```
            Console.WriteLine(NumeroCelular); //internal (mesmo projeto)
            Console.WriteLine(JeitoDeFalar);  //protected internal (herança ou
mesmo projeto)
                Console.WriteLine(SegredoFamilia); //private protected (mesma
classe ou herança no mesmo projeto (c# >= 7.2))
            // Console.WriteLine(UsaMuitoPhotoshop); //private (padrão)
        }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Encapsulamento
{
    public class AmigoProximo
    {
        public readonly SubCelebridade amiga = new SubCelebridade();

        public void MeusAcessos()
        {
            Console.WriteLine("AmigoProximo...");

            //Console.WriteLine(InfoPublica);  //public
            //Console.WriteLine(CorDoOlho);    //protected (herança)
            //Console.WriteLine(NumeroCelular); //internal (mesmo projeto)
            //Console.WriteLine(JeitoDeFalar);  //protected internal (herança ou mesmo projeto)
             //Console.WriteLine(SegredoFamilia); //private protected (mesma classe ou herança no
mesmo projeto (c# >= 7.2))
            //Console.WriteLine(UsaMuitoPhotoshop); //private (padrão)

            Console.WriteLine(amiga.InfoPublica);
            // Console.WriteLine(amiga.CorDoOlho);
            Console.WriteLine(amiga.NumeroCelular);
            Console.WriteLine(amiga.JeitoDeFalar);
            // Console.WriteLine(amiga.SegredoFamilia);
            // Console.WriteLine(amiga.UsaMuitoPhotoshop);
        }
    }
}
```

Programa Principal (Console):

```
using Encapsulamento;
namespace Encapsulamento_proj
{
    public class FilhoNaoReconhecido : SubCelebridade
    {
        public new void MeusAcessos()
        {
            Console.WriteLine("FilhoNaoReconhecido...");

            //Console.WriteLine(InfoPublica);  //public
            //Console.WriteLine(CorDoOlho);    //protected (herança)
            //Console.WriteLine(NumeroCelular); //internal (mesmo projeto)
             //Console.WriteLine(JeitoDeFalar);  //protected internal (herança
ou mesmo projeto)
              //Console.WriteLine(SegredoFamilia); //private protected (mesma
classe ou herança no mesmo projeto (c# >= 7.2))
            //Console.WriteLine(UsaMuitoPhotoshop); //private (padrão)

            Console.WriteLine(InfoPublica);
            Console.WriteLine(CorDoOlho);
            // Console.WriteLine(NumeroCelular);
            Console.WriteLine(JeitoDeFalar);
            // Console.WriteLine(SegredoFamilia);
            // Console.WriteLine(UsaMuitoPhotoshop);
        }
    }
```

```
    public class AmigoDistante
    {
        public readonly SubCelebridade amiga = new SubCelebridade();

        public void MeusAcessos()
        {
            Console.WriteLine("AmigoDistante...");

            //Console.WriteLine(InfoPublica);  //public
            //Console.WriteLine(CorDoOlho);    //protected (herança)
            //Console.WriteLine(NumeroCelular); //internal (mesmo projeto)
             //Console.WriteLine(JeitoDeFalar);  //protected internal (herança
ou mesmo projeto)
               //Console.WriteLine(SegredoFamilia); //private protected (mesma
classe ou herança no mesmo projeto (c# >= 7.2))
            //Console.WriteLine(UsaMuitoPhotoshop); //private (padrão)

            Console.WriteLine(amiga.InfoPublica);
            //Console.WriteLine(amiga.CorDoOlho);
            //Console.WriteLine(amiga.NumeroCelular);
            //Console.WriteLine(amiga.JeitoDeFalar);
            //Console.WriteLine(amiga.SegredoFamilia);
            //Console.WriteLine(amiga.UsaMuitoPhotoshop);
        }
    }
    internal class Program
    {
        static void Main(string[] args)
        {
           // SubCelebridade sub = new SubCelebridade();
           // sub.MeusAcessos();

            new SubCelebridade().MeusAcessos();

            new FilhoReconhecido().MeusAcessos();
            new AmigoProximo().MeusAcessos();

            new FilhoNaoReconhecido().MeusAcessos();
            new AmigoDistante().MeusAcessos();
        }
    }
}
```

44. Polimorfismo

```
namespace Polimorfismo
{
    public class Comida {
        public double Peso;

        public Comida(double peso) {
            Peso = peso;
        }

        public Comida()
        {
        }
    }

    public class Feijao : Comida {
        public Feijao(double peso) : base(peso) { }

    }
    public class Arroz : Comida {

    }

    public class Carne : Comida {

    }

    public class Pessoa {
        public double Peso;
```

```
        /*
        public void Comer(Feijao feijao) {
            Peso += feijao.Peso;
        }
        public void Comer(Arroz arroz)
        {
            Peso += arroz.Peso;
        }

        public void Comer(Carne carne)
        {
            Peso += carne.Peso;
        }*/

        public void Comer( Comida comida) {
            Peso += comida.Peso;
        }

    }



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

            Feijao ingrediente1 = new Feijao(0.3);
            Arroz ingrediente2 = new Arroz();
            ingrediente2.Peso = 0.25;

            Carne ingrediente3 = new Carne();
            ingrediente3.Peso = 0.5;

            Pessoa cliente = new Pessoa();
            cliente.Peso = 80.2;

            cliente.Comer(ingrediente1);
            cliente.Comer(ingrediente2);
            cliente.Comer(ingrediente3);
```

```
        Console.WriteLine($"Agora o peso do cliente é {cliente.Peso}Kg!");



        }
    }
}
```

45. Classe abstrata

```
namespace ClasseAbstrata
{
    public abstract class Celular
    {
        public abstract string Assistente();

        public string Tocar()
        {
            return "Trim trim trim...";
        }
    }

    public class Samsung : Celular
    {
        public override string Assistente()
        {
            return "Olá! Meu nome é Bixby!";
        }
    }

    public class IPhone : Celular
    {
        public override string Assistente()
        {
            return "Olá! Meu nome é Siri!";
        }
    }
```

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

          var celulares = new List<Celular> {new IPhone(), new Samsung() };

            foreach (var celular in celulares) {
                Console.WriteLine(celular.Assistente());

            }
        }
    }
}
```

46. Interface

```
namespace Interface
{
    interface OperacaoBinaria {
        int Operacao(int a, int b);
    }

    class Soma : OperacaoBinaria {
        public int Operacao(int a, int b) {
            return a + b;
        }
    }

    class Subtracao : OperacaoBinaria {
        public int Operacao(int a, int b) {
            return a - b;
        }
    }

    class Multiplicacao : OperacaoBinaria
    {
        public int Operacao(int a, int b)
        {
            return a * b;
```

```
        }
    }

    class Calculadora {
        List<OperacaoBinaria> operacoes = new List<OperacaoBinaria> {
            new Soma(),
            new Subtracao(),
            new Multiplicacao() };

        public string ExecutarOperações(int a, int b) {

            string resultado = "";
            foreach (var op in operacoes)
            {
                resultado +=
                    $"Usando {op.GetType().Name} = {op.Operacao(a, b)}\n";
            }
            return resultado;
        }

    }

    internal class Program
    {
        static void Main(string[] args)
        {
            var calc = new Calculadora();
            var resultado = calc.ExecutarOperações(30, 5);
            Console.WriteLine(resultado);
        }
    }
}
```

47. Classe e Metodo Sealed

```
namespace SealedClass
{

    sealed class SemFilho {
        public double ValorDaFortuna() {
            return 1_500_600.25;
        }
    }

    // class SouFilho : SemFilho {  // não é possivel

    // }

    class Avo {
        public virtual bool HonrarNomeFamilia() {
            return true;
        }
    }

    class Pai : Avo {
        public override sealed bool HonrarNomeFamilia(){
            return true;
        }

    }

    class FilhoRebelde : Pai {
       // public override bool HonrarNomeFamilia() {
       //      return true;
```

```
        //  }
    }
    internal class Program
    {
        static void Main(string[] args)
        {
            SemFilho semFilho = new SemFilho();
            Console.WriteLine(semFilho.ValorDaFortuna());

            FilhoRebelde filhoRebelde = new FilhoRebelde();
            Console.WriteLine(filhoRebelde.HonrarNomeFamilia());

        }
    }
}
```

Delegates e Lambdas

48. Exemplo Lambda

```
namespace ExemploLambda
{
    internal class Program
    {
        static void Main(string[] args)
        {
            Action algoNoConsole = () => {
                Console.WriteLine("Lambda com C#");
            };

            algoNoConsole();

            Func<int> jogarDado = () =>
            {
                Random rand = new Random();
                return rand.Next(1, 7);
            };

            Console.WriteLine(jogarDado());
```

```
            Func<int, string> conversorHex = (numero) => {
                return numero.ToString("x");
            };

            Console.WriteLine(conversorHex(1234));
                Func<int, int, int, string> formatarData = (dia, mes, ano) =>
String.Format("{0:D2}/{1:D2}/{2:D2}", dia, mes, ano);
            Console.WriteLine(formatarData(1,2,1989));

        }
    }
}
```

49. Delegate com Lambda

```
namespace LambdasDelegate
{
    delegate double Operacao(double x, double y);

    internal class Program
    {
        static void Main(string[] args)
        {
            Operacao soma = (x, y) => x + y;
            Operacao sub = (x, y) => x - y;
            Operacao mult = (x, y) => x * y;

            Console.WriteLine(soma(2, 5));
            Console.WriteLine(mult(2, 5));
            Console.WriteLine(sub(2,5));
        }
    }
}
```

50. Usando Delegates

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

namespace MetodosEFuncoes
{
    class UsandoDelegates
    {
        delegate double Soma(double a, double b);
        delegate void ImprimirSoma(double a, double b);

        static double MinhaSoma(double x, double y) {
            return x + y;
        }

        static void MeuImprimirSoma(double a, double b) {
            Console.WriteLine(a + b);
        }

         static void Main(string[] args) {
                Soma op1 = MinhaSoma; //deve haver  compatibilidade entre as
assinaturas
            Console.WriteLine(op1(2, 3.9));

            ImprimirSoma op2 = MeuImprimirSoma;
            op2(5.4, 8);

            Func<double, double, double> op3 = MinhaSoma;
            Console.WriteLine(op3(2.5, 3));

            Action<double, double> op4 = MeuImprimirSoma;
```

```
            op4(7.7, 23.4);
        }
    }
}
```

51. Delegate com Funções Anônimas

```
namespace DelegateFuncAnon
{
    internal class Program
    {
        delegate string StringOperacao(string s);
        static void Main(string[] args)
        {
            StringOperacao inverter = delegate (string s)
            {
                char[] charArray = s.ToCharArray();
                Array.Reverse(charArray);
                return new string(charArray);

            };

            Console.WriteLine(inverter("C# é muito legal!"));
        }
    }
}
```

52. Passando Delegate como Parâmetro

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

namespace MetodosEFuncoes
{
    class DelegatesComoParametros
    {
        public delegate int Operacao(int x, int y);

        public static int Soma(int x, int y) {
            return x + y;
        }

        public static string Calculadora(Operacao op, int x, int y) {
            var resultado = op(x, y);
            return "Resultado: " + resultado;
        }

        static void Main(string[] args) {
            Operacao subtracao = (int x, int y) => x - y;
            Console.WriteLine(Calculadora(subtracao, 3, 2));

            Console.WriteLine(Calculadora(Soma, 3, 2));
        }
    }
}
```

53. Métodos de Extensão

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

namespace MetodosEFuncoes
{
    public static class ExtensoesInteiro { //o primeiro parametro recebe this
        public static int Soma(this int num, int outroNumero) {
            return num + outroNumero;
        }

        public static int Subtracao(this int num, int outroNumero) {
            return num - outroNumero;
        }
    }

    class MetodosDeExtensao
    {
        static void Main(string[] args)  {
            int numero = 5;

            Console.WriteLine(numero.Soma(3));
            Console.WriteLine(numero.Subtracao(10));

            Console.WriteLine(2.Soma(3));
            Console.WriteLine(2.Subtracao(4));
        }
    }
}
```

Excessões

54. Uso do try/catch/finally

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

namespace Excecoes
{
    public class Conta {
        double Saldo;

        public Conta(double saldo) {
            Saldo = saldo;
        }

        public void Sacar(double valor) {
            if (valor > Saldo) {
                throw new ArgumentException("Saldo insuficiente.");
            }

            Saldo -= valor;
        }
    }

    class PrimeiraExcecao
    {
         static void Main(string[] args) {
            var conta = new Conta(1_223.45);

            try { //sucesso
                // int.Parse("abc");

                conta.Sacar(1600);
                Console.WriteLine("Retirada com sucesso!");
            } catch (Exception ex) { //erro
                Console.WriteLine(ex.GetType().Name);
                Console.WriteLine(ex.Message);
            } finally { ///sempre executado
                Console.WriteLine("Obrigado!");
            }
```

```
        }
    }
}
```

55. Excessões Personalizadas

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

namespace Excecoes
{
    public class NegativoException : Exception {
        public NegativoException() { }

        public NegativoException(string message) : base(message) { }

        public NegativoException(string message, Exception inner)
            : base(message, inner) { }
    }

    public class ImparException : Exception {
        public ImparException(string message) : base(message) { }
    }

    class ExcecoesPersonalizadas
    {
        public static int PositivoPar() {
            Random random = new Random();
            int valor = random.Next(-30, 30);

            if (valor < 0) {
                throw new NegativoException("Número negativo... :(");
            }

            if(valor % 2 == 1) {
                throw new ImparException("Valor impar... :(");
            }

            return valor;
```

```
        }

        static void Main(string[] args) {
            try {
                Console.WriteLine(PositivoPar());
            } catch(NegativoException ex) {
                Console.WriteLine(ex.Message);
            } catch(ImparException ex) {
                Console.WriteLine(ex.Message);
            }
        }
    }
}
```