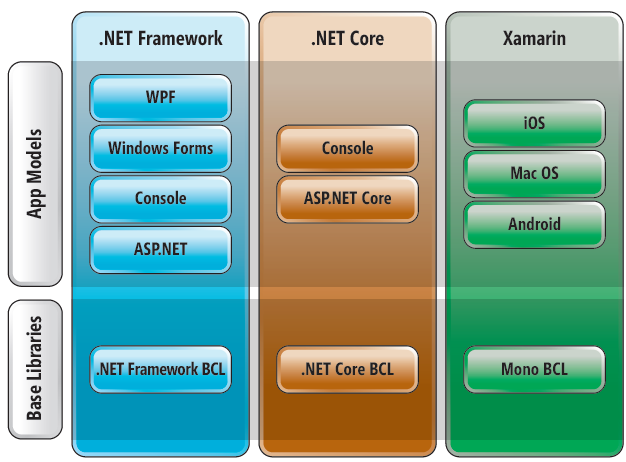
**POO com C# e .NET: O Código que Ganha Vida**

O **.NET Framework (C#**, VB.NET e **F#)** foi introduzido pela Microsoft em 2002, como uma plataforma de desenvolvimento para construir **aplicações Windows**.

Ele trouxe consigo o **Common Language Runtime** (**CLR**), que gerencia a execução do código, e o **Framework Class Library** (**FCL**), uma vasta coleção de classes reutilizáveis para desenvolvimento.

Em 2016, a Microsoft lançou o **.NET Core**, uma versão modular, open-source e **multiplataforma** do .NET.

Em 2020, a Microsoft lançou o **.NET unificado** ou simplesmente **.NET (.NET Framework 4.8 e .Net Core 8.0)**. Atualmente está na versão **.NET 9**, **.NET 10 LTS**, p/ **NOV/25**. .NetCore->**API**; Zamarin->**Mobile**; Unity->**Games**.



<https://learn.microsoft.com/pt-br/archive/msdn-magazine/2017/september/net-standard-demystifying-net-core-and-net-standard>

**Tópicos**

* IDE Ambiente: **Visual Studio Community**
* Linguagem POO C#
  + - Lógica de programação
      * **Teoria**
      * **Prática**
* POO
* Tecnologias .NET
  + **ASP.NET Core** para App Web MVC / API
  + **Entity Framework** para mapeamento **ORM**
* Projeto App Web
  + **Entidades**:
    - **Pedido**: **id**: Long, **idCliente**: Long, **moment**: Date, **status**: OrderStatus
    - **ItemPedido**: **quantity**: Integer, **price**: Double
    - **Cliente**: **id**: Long, **name**: String, **email**: String, **birthDate**: Date
    - **Produto**: **id**: Long, **name**: String; **price**: Double
  + **Relacionamentos:**
  + **Pedido/Cliente**: Um para Muitos, **-client**
  + **Pedido/Produto**: Muitos para Muitos, **-item** e **-product**
  + **ItemPedido**: Entidade Associativa Composição

**.Net (**pré compilação **CIL**; compilação **JIT)**

* **CIL** – Common Intermediate Language – **platform agnostic** - **bytecode**
* **BCL** – Base Class Library
* **CLR** – Common Language Runtime – **VM**, possui garbage collection
* **JIT** – Compilação just-in-time

**Estrutura de Uma Aplicação Web**

* namespaces: agrupamento lógico das classes
* assembly (DLL/EXE) **projetos**: agrupamento físico das classes – **build**
* aplicação ou **solution**: agrupamento de **assemblies**

**Download do Visual Studio Comunnity - IDE**

* <https://visualstudio.microsoft.com/pt-br/vs/community/>

**Extensão dos Arquivos**

* .sln = **solution**
* .csproj = **arquivo de configuração da solution - dependências**
* .cs= **class**
* subpastas: bin e obj – **arquivos compilados**

**Comentários**

* // - linha
* /\* \*/ - bloco

**Estrutura Básica App Console: versão clássica**

**using** System**;**

**namespace** HelloWorld;

public **class** Program {

**public static void Main**(string[] args){ // **entrypoint**

**Console**.**WriteLine**("Hello World.");

}

}

**Estrutura Básica App Console: nova versão**

* A partir do C# versão10
* .NET versão 6

**Console**.**WriteLine**("Hello World.");

**Dicas IDE**

* CTRL + F5 - executar o programa; cw + TAB+TAB – Console.WriteLine
* CTRL + K + D – indentar o programa
* Tools -> Options -> International settings – alterar o idioma
* Tools -> Options -> Text Editor -> C# -> Code Style -> Formatting -> New Lines – definir o padrão de indentação
* Display -> solution manager – exibir o gerenciador de soluções

**Tipos de Dados Primitivos: C# vs .Net Framework (biblioteca System)**

**Ex**.: long n5 = **56L**; char = '\u0041'; ou char = 'A'; float = **5.6f**;

|  |  |
| --- | --- |
| **C#** | **.Net Framework** |
| bool | Boolean |
| **byte** | **Byte (não sinal)** |
| sbyte | SByte |
| char | Char |
| decimal | Decimal |
| double | Double |
| float | Single |
| Int | Int32 |
| uint | UInt32 |
| long | Int64 |
| ulong | UInt64 |
| object | Object**(qualquer tipo)** |
| short | Int16 |
| ushort | UInt16 |
| string | String**(imutável)**  **Tipo referência** |

**Tipos em Memória**

* Valor – **memória Stack;** tipo **struct** é por valor
* Referência – **memória Heap;** tipo **class** é por referência, ponteiro
* Instanciação **– alocação dinâmica de memória**
* Tipos Primitivos **– alocação estática de memória**

**Nota: Garbage collector**, monitora os objetos alocados dinamicamente pelo programa na memória HEAP, desalocando aqueles que não possuem referência.Criada no **HEAP**, quando instanciadas por **NEW**. Existe desalocação de memória por escopo, variáveis locais **STACK**, dentro de blocos.

**Propriedade Estáticas**

* int.MinValue;
* int.MaxValue;

**Convenções de Projetos – Definição de Nomes**

* Camel Case ex.: – **lastName** – parâmetros; variáveis
* Pascal Case ex.: – **LastName** – namespaces; classe; properties; métodos
* Padrão ex.: **\_lastName** – atributos internos da classe

**Declaração de variáveis / atribuição**

* atribuição: =
* atribuição composta/aumentada; +=; -=; \*=; /=; %=
* int idade = 55;
* string nome;
  + nome = "Juju";
* bool resposta = true;
* char sexo = 'f';
* decimal salario = 5.65;
* double x=**null**; //Erro struct; **Nullable**<double> x=**null**; double**?** x = **null**;

**Métodos Null para tipos Struct**

* double**?** x = **null**; double**?** y = **55**;
  + x.GetValueOrDefault();
  + x.HasValue; // true ou false
  + x.Value; // erro, senão tiver valor
* Operador de **coalescência** nula **??**: double y = x **??** **0.0**; // **y não nullable**

**Valores Padrão de Tipo**

* int : 0
* bool : false
* char : caractere código 0
* object : null

**const / var / dynamic / obj / escopo**

* **const** int idade = 55;
* **var** nome = "Juju"; // **inferência de tipo de dados**
  + nome = "Maria";
* **dynamic** numero = 5; // **qualquer tipo de dados**
  + numero = "55";
* **obj**
* **escopo/inicialização**
  + double x; if (**condição**) x++; Console.**WriteLine**(x); // erro

**Conversão de variáveis**

* implícita entre tipos
  + **float** x = 3.8f; **double** y = x; // retornou float
  + **double** res; **int** a = 5; b = 2; res = a / b; // retornou int
  + **int** res = 5 / 2;
* explicita entre tipos compatíveis
  + **double** a = 3.8; **float** b = (**float**)a; / /**a tem 8 bytes**; **b tem 4 bytes**
  + **double** a = 3.8; **int** b = (**int**)a;
  + **double** res; **int** a = 5; b = 2; res = (**double**)a / b; **double** res = 5.0/2;
  + **double** base = Math.**Pow**(5.0**, 3.0**);

**Operadores/expressões**

* ( ),
* aritméticos: \* / %, +-
* comparação – relacionais: >; >=; <; <=; ==; !=
* lógicos: **precedência**: !; &&; ||
* ternário – if ... else ...: condição? valor verdadeiro: valor falso;
* unários: +a; -a; a++; ++a => **b = ++a**; a--; --a; ! inversão lógica, !true
* is: verificar se é de determinado tipo
* as: conversão de tipo
* typeof: tipo do objeto
* sizeof: tamanho do objeto
* nameof: nome do objeto // variável ou método
* bit a bit – bitwise: manipulação com bits

**Entrada de Dados: input**

* **input** // entrada de dados
  + string nome = Console.**ReadLine**() // retorna string
  + int idade = Convert.**ToInt32**(**Console.ReadLine()**);
  + int idade = int.**Parse**(**Console.ReadLine()**);
  + double altura = Convert.**ToDouble**(**Console.ReadLine()**);
  + string s = Console. **ReadLine**(); // uva banana pera
    - string vet[] = s.Split(' ');
  + string s = Console. **ReadLine**().Split(' '); // uva banana pera
  + char ch = char.**Parse**(**Console.ReadLine()**); // retorna string

**Saída de Dados: output**

* **output** // saída de dados
  + Console.**Write**();
  + Console.**Write**($"{**variável**}"); // interpolação
  + Console.**Write**($"{5 \* 3, 6}"); // interpolação c/ espaço de 6 antes

Console.**Write**($"{"oi", 6}"); // interpolação c/ espaço de 6 antes

* + Console.**WriteLine**();
  + Console.**WriteLine**(altura.**ToString**("**F3**")); // três casas decimais
  + Console.**WriteLine**(altura.**ToString**("**F2**", **CultureInfo**.**InvariantCulture**)); // 2 casas decim. com separador .
    - using **System.Globalization**;

**placeholders, concatenação, interpolação**

ex.: int idade = 55; double = saldo = 566; string = "maria";

* placeholders: Console.**WriteLine**("{0} tem {1} anos e saldo {2:F2}.", nome, idade, saldo);
* concatenação: Console.**WriteLine**(nome + " tem " + idade + " anos e saldo " + saldo.toString("F2"));
* Console.**WriteLine**("{0} tem {1} anos e saldo {2}.", nome, idade, saldo);
* Interpolação: Console.**WriteLine**($"{nome} tem {idade} anos e saldo {saldo:F2}.");

**Estrutura de Controle: condicional/seleção/encadeamento**

* if ( exp ){ ... }
* if ( exp ){ ... } else { ... }
* switch/case/break/default – múltipla escolha
* Condicional ternária: int x = (**condição**) **?** valor se true **:** valor se false;

**Nota**: A chave {} é obrigatória se o bloco existir mais de um comando.

**Tipos de Dados Derivados: arrays / coleções / structs / enums**

* **Vetores** – arranjos unidimensionais – homogênea – tamanho fixo
  + **string**[] frutas = {"uva", "pera", "mamão", "banana", "caja"};
    - frutas.Length; string fruta = frutas[2];
  + **int**[] inteiros = **new** int[5]; // necessário informar o tamanho
  + **int**[] inteiros = **new** int[] {10, 20, 30, 40};
  + Produto[] vect = **new** Produto[3]; // **class**
    - vect[0] = **new** Produto { Name = "Juju", Preco = 55.5};
  + params – em parâmetros de função
    - **class** Calc { public static int Sum(params int[] numbers) { }}
    - **class** Program { int sum = Calc.Sum(5, 6);}
* **Matriz** – arranjos bidimensionais – homogênea – tamanho fixo
  + double[,] mat = new double[2,3];
  + mat.Length, mat.Rank, mat.GetLength(0), mat.GetLength(1)
* **List** - **namespace** **System.Collections.Generic**
  + **List**<int> numberList = **new** List<int>(); // Lista vazia
  + **List**<string> frutas = **new** List<string>() {"uva", "pera", "mamão", "banana", "caja"}
    - frutas.**Add**("umbu"); // final da lista
    - frutas.**Insert**(2, "abacaxi"); // na posição da lista
    - frutas.**Remove**("umbu");
    - frutas.**RemoveAt**(2); // pela posição
    - frutas.**RemoveRange**(2, 3); // posição e quantidade
    - frutas.**Count**;
    - frutas.**Clear**();
  + Predicados(lambda) – função anônima
    - string f = frutas.**Find**(x => x[0] == 'u');
    - string f = frutas.**FindLast**(x => x[0] == 'u');
* **generics; set; dicionary**

**Estrutura de Controle: repetição**

* while (condição)
* do ... while (condição)
* for (início; condição; incremento)
* foreach // para coleções / listas
  + List<int> numberList = new List<int>() {5, 3, 4, 6, 8, 7, 1, 0};

foreach( int number in numberList) Console.**WriteLine**(number);

**Nota**: **Estrutura sequencial**: **entrada** -> **processamento** -> **saída**

**DateTime – Representa um instante – struct; Multa/Juros**

* Número de **ticks**(100) nanosegundos, desde a meia noite do dia 1 de janeiro do ano 1 da era comum.
  + **DateTime data =** DateTime**.Now; // instante atual**
    - Console.WriteLine(data.Ticks); // representa o instante atual
  + **DateTime** data1 = new DateTime(2021, 11, 25);
  + **DateTime** data2 = new DateTime(2021, 11, 25, 19, 28, 10);
  + **DateTime** data3 = DateTime.**UtcNow**; // GMT -2 da hora universal
  + **DateTime** data4 = DateTime.**Today**; // Data Local com hora zerada
  + **DateTime** data5 = DateTime.**Parse**("2021-11-25");
  + **DateTime** data6 = DateTime.**Parse**("25/11/2021");
  + **DateTime** data7 = DateTime.**ParseExact**("2021-11-25", "yyyy-MM-dd", CultureInfo.InvariantCulture);
  + **DateTime** data8 = DateTime.**ParseExact**("25/11/2021", "dd-MM-yyyy", CultureInfo.InvariantCulture);
  + **DateTime** d = new DateTime(2021, 11, 25);
    - Console.WriteLine(d.Date);
    - Console.WriteLine(d.Day);
    - Console.WriteLine(d.DayOfWeek);
    - Console.WriteLine(d.DayOfYear);
    - Console.WriteLine(d.Month);
    - Console.WriteLine(d.Kind); // Local ou Universal
    - Console.WriteLine(d.ToLongDateString());
    - Console.WriteLine(d.AddHours(5));
    - Console.WriteLine(d.AddDays(7));

**Nota**: DateTimeKind – data local ou global; **ticks**, unidade do .NET

**DateTimeKind – padrão ISO 8601**

* Local – fuso horário do sistema GMT -3, -2 se horário de verão
* Utc – fuso horário GMT – Greenwich Mean Time – **horário de Londres**
* Unspecified
* **Exemplo**:
  + DateTime d1 = new Datetime(2025, 07, 21, DateTimeKind.Local);
  + DateTime d2 = new Datetime(2025, 07, 21, DateTimeKind.Utc);
  + DateTime d3 = new Datetime(2025, 07, 21); // Unspecified
  + Console.WritLine(d1.Kind);
  + Console.WritLine(d1.ToLocalTime());
  + Console.WritLine(d1.ToUniversalTime());
  + DateTime d4 = Datetime.Parse("2025-07-21T13:21:50Z"); // Utc
  + Console.WritLine(d4.ToUniversalTime().ToString(("yyyy-MM-ddTHH:mm:ssZ")); // padrão ISO 8601

**TimeSpan – Representa uma duração, intervalo de instantes - struct**

* + **TimeSpan** t1 = new TimeSpan(00, 01, 30); // 00h01min30s
    - Console.WriteLine(t1.Ticks); // 900000000
  + **TimeSpan** t1 = new TimeSpan(); // 00h00min00s
  + **TimeSpan** t2 = new TimeSpan(900000000**L**); // 0h1min30s
  + **TimeSpan** t3 = new TimeSpan(2, 11, 21);
  + **TimeSpan** t4 = new TimeSpan(1, 2, 11, 21); // 1.2:11: 21 // **1dia**
  + **TimeSpan** t5 = TimeSpan.FromDays(1.5); // 1.12:00:00

**Tratamento de erros**

* **try** {comandos} **catch** (ExceptionType ex) {comandos};
* **try** {cmd} **catch** (ExceptionType1 ex) {cmd} **catch** (ExceptionType2 ex) {cmd};
* **try** {comandos} **catch** (ExceptionType) {cmd} **finally** {cmd};

**Ex**.:

**try** {int resultado = 10 / **int.Parse**(Console.ReadLine());

**Console**.WriteLine("Resultado: " + resultado);

}

**catch** (**DivideByZeroException** ex) {

**Console**.WriteLine("Erro: Divisão por zero não é permitida.");

}

**catch** (**FormatException** ex) {

**Console**.WriteLine("Erro: Entrada inválida. Digite um número inteiro.");

}

**catch** (**Exception** ex) {

**Console**.WriteLine("Ocorreu um erro inesperado: " + ex.**Message**);

}

**finally** {

Console.WriteLine("Fim do tratamento de erros.");

} **Nota**: **01-prova-logica-de-programacao.pdf**

**Função**

* Modularização
* Delegação
* Reaproveitamento
* Parâmetros
* retorno

**Manipulação de arquivos**

* using System.IO; // necessário esta biblioteca
* System.IO.File.**WriteAllText**(**arquivo**, **texto**) ou File.WriteAllText ...
  + string path = **@**"c:\projweb\", filename = "teste.txt", content = "Oi";
  + File.**WriteAllText**(path+filename, content);
* System.IO.File.**Delete**(Path.GetDirectoryName( typeof(**Program**) ) )
* File.**AppendAllTest**( path+filename, "conteúdo");
* string fileContent = File.**ReadAllTest**( path+filename, "conteúdo");
* File.ReadAllLines(path+filename)
* File.Exists(path+filename) ...
  + **Ex**.: string path = @(caminho path);

string file = "nome do arquivo";

string filePath = path+ filePath;

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

if (File.Exists(filePath)) {

lista.addRange(File.ReadAllLines(filePath));}

while (true){ definir o menu string opc = Console.ReadLine(); switch(opc){case "1": ...; break;} ... }

//**Nota**: opc.**IsNullOrEmpty**(dado)

**Debug**

* F9 – marcar/desmarcar breakpoint
* F5 – iniciar/continuar debug
* F10 – executar um passo – pula a função
* F11 – executar um passo – entra na função
* SHIFT F11 – sair do modo de execução
* SHIFT F5 – para debug

**Debug/Janelas**

* Watch – expressões personalizadas
* Autos – expressões / variáveis que o Visual Studio achar interessante
* Locals – variáveis locais – escopo da função

**POO – Objetos – entidades do mundo real**

* Classes – **Tipo de Dado Derivado – unidade básica – definição do tipo**
  + namespaces de classes: **entities; services; repositories etc**
  + **Pode Representar, categorias**:
    - **Entidades**: Produto, Triangulo, Cliente
    - **DTO**: ProdutoDTO
    - **Serviços**: ProdutoService, EmailService, ClienteService
    - **Controladores**: ProdutoController, ClienteController
    - **Repositories**: ProdutoRepository
    - **Utilitários**: Calculadora, Compactador
    - **Config**: WebConfig, SwaggerConfig **etc**
* Membro de instância
  + Atributos, características do objeto, **prop**+**tab** – estrutura get/set
  + Métodos – comportamentos
  + Método ToString() – Toda classe é uma subclasse da classa Object
    - **GetType** – retorna o tipo do objeto
    - **Equals** – comparação de objetos
    - **GetHashCode** – retorna o código hash do objeto
    - **ToString** – converte o objeto em String
      * public **override** string **ToString**() { return ...}
  + Construtores - atalho no Visual Studio **ctor**
    - método especial
    - se possuir um **construtor padrão**, pode instanciar com **new Construtor {...}**, utilizando {} com os dados
  + **this** – ponteiro/referência para o próprio objeto
    - public **Produto**() { Quantidade = 55; }
    - public **Produto**(string nome, double preco) : **this**() {Nome = nome; Preco = preco; }
    - public **Produto**(string nome, double preco, int qtd) : **this**(nome, preco) {Quantidade = qtd; }
* Membros estáticos ou membros de classe
  + **static** – **atributos** e **métodos**
    - **readonly**, execução
    - **const**, compilação, equivalente ao final no Java
  + membros independentes de objetos, chamados pelo nome da classe
  + aplicações comuns
    - declaração de constantes
    - classes utilitárias: **Ex**.: Math.**sqrt**(double);
  + classe que só possui membros estáticos, será uma classe estática e não poderá ser instanciada
* Objetos: Instâncias da classe, são ponteiros
* Modificadores de Acesso
  + public, private, protected, internal, protected internal, private protected
  + private string \_nome; // utilizar o **\_,** **para atributos internos**
  + private int \_quantidade;
  + Encapsulamento – proteger os atributos e/ou métodos
    - O objeto deve estar em estado consistente: **private**
    - Métodos Get/Set para cada atributo, conforme regras de negócio
* Properties – definir acesso a atributos por métodos encapsulados
  + **prop** + **tab** + **tab**: monta a estrutura de definição do atributo
  + public string Nome { **get** { return **\_nome**; }

**set** { **\_nome** = value;} // value é o parâmetro

}

* + Auto Properties – quando não existir regra de negócio;
    - public string **Nome** { **get; set;** }
    - public double **Preco** { **get; private set;** }
* Polimorfismo: **overload** – sobrecarga e **override** – sobreposição
* Herança – extends
* Associação: Composição e Agregação
* **LINQ p/ acessar BD / expressões lambda / delegates**

**Enums – tipo valor**

* **OrderStatus** os = Enum.Parse<**OrderStatus**>("**Entregue**");
* **OrderStatus** os = (**OrderStatus**)Enum.Parse(**typeof**(OrderStatus), "**Entregue**");
* **enum** OrderStatus : **int** { Pendente = 0, Processando = 1, Enviado = 2, Entregue = 3} // **OrderStatus** : int = derivação de **tipo int.**
* **string orderTxt = OrderStatus.Pendente.**ToString()**;**
* **OrderStatus** os = Enum.Parse<**OrderStatus**>("Pendente");

**Enums – tipo valor**

É um tipo de associação que permite que um objeto contenha outro.

Relação **tem-um** ou **tem-vários**. **UML relação** (**todo/parte**) – **diamante preto**.

**Sugestão Para Definição de Classe**

* Definir atributos privados
* Properties autoimplementados
* Construtores
* Propriedades customizadas dos atributos privados
* Outros métodos da classe, incluindo o método ToString()

**Ex**.: **classes**: BankTerminal:**Menu**; Start; BankOperations: Sd; Depósito; Saque.

**Ex**.: Calcular a área de dois triângulos quaisquer, imprimir a maior área.

**Lados**: a, b, c; Métodos p/ reaproveitamento e delegação.

**Fórmula Heron**: area = raiz(p(p-a) (p-b) (p-c)), onde p = (a + b + c) / 2.

**Projeto API Web**

**Projeto 1Diagrama

O conteúdo gerado por IA pode estar incorreto.**

**Projeto 2**

**Diagrama

O conteúdo gerado por IA pode estar incorreto.**

**Projeto 3**

**Diagrama

O conteúdo gerado por IA pode estar incorreto.**

**Projeto 4**

**Diagrama

O conteúdo gerado por IA pode estar incorreto.**

**Tabela

O conteúdo gerado por IA pode estar incorreto.Próxima Seq**