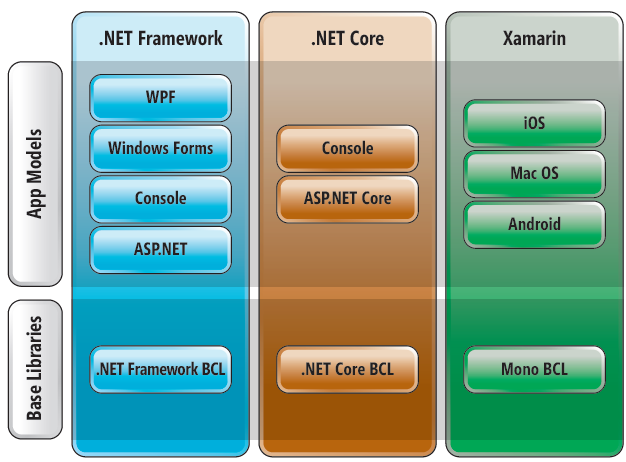
**POO com C# e .NET: O Código que Ganha Vida**

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');

**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

**Função**

* Modularização
* Delegação
* Reaproveitamento
* Parâmetros
* retorno

**Tratamento de erros – Erro ou Comportamento Inesperado**

* Erro ou comportamento – pilha de exceções – **stack trace(rastreamento)**
* Estrutura: **try**-**cath ou try**-**cath-finally** ; **upcasting é permitido**
  + **try** – execução normal do trecho de código; cenário
  + **cath** – código executado quando ocorre uma exceção
  + **finally** – executa o bloco de código independente de existir um erro
    - **Ex**.: Fechamento de arquivo ou conexão com o DB
  + **throw** – serve para lançar exceção / **"corta" o método**
* .NET, é um objeto herdado da class **System.Exception**
  + **SystemException - .NET**
  + **ApplicationException –** para criação personalizadas
* Sintaxe
  + **try** {comandos} **catch** (ExceptionType **e**) {comandos};
  + **try** {cmd} **catch** (ExceptionType1 **e**) {cmd}

**catch** (ExceptionType2 **e**) {cmd};

* + **try** {comandos} **catch** (ExceptionType **e**) {cmd} **finally** {cmd};

**Exemplo**:

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

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

}

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

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

}

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

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

}

**catch** (**Exception** e) {

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

}

**finally** {

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

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

**Manipulação de arquivos**

* using **System.IO**; // necessário esta biblioteca
  + File – **static members**, FileInfo – **instance members**, IOException
  + FileStream – **para** **I/O de Arquivos**; **StremReader** <- FileStream
  + StreamWriter <- FileStream
  + Directory – **static members**, DirectoryInfo – **instance members**, IOException
  + Class **Path**. Ex.: **Path**.GetDirectoryName(**path**); // **GetFileName**
* Escrita no Arquivo
  + **System**.**IO**.File.**WriteAllText**(**arq.**, **texto**) ou File.**WriteAllText** ...
  + string path = @"c:\projweb\", filename = "teste.txt", content = "Oi";
  + File.**WriteAllText**(path+filename, content);
* Excluir Arquivo
  + **System**.**IO**.File.**Delete**(Path.GetDirectoryName(typeof(**Program**)))
* Acrescentar no Final do Arquivo
  + File.**AppendAllTest**( path+filename, "conteúdo");
* Ler o Arquivo
  + **string** fileContent = File.**ReadAllTest**(path+filename, "conteúdo");
  + **string[]** lines = **File**.ReadAllLines(path+filename);
* Testar Existência do Arquivo
  + **File**.Exists(path+filename) ...
* **Ex1**.: **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)

* **Ex2**.: **string** path = **@(**caminho path);

**FileStream** fs = **new** FileStream(path, FileMode.Open);

**StreamReader** sr = **new** StreamReader(fs);

**while** (!sr.EndOfStream) {

**string** line = sr.ReadLine();

ConsoleWriteLine(line); } // **Nota**: FileStream da **NetFlix**

* **using ( StreamReader** sr = **File.**OpenText(path) **)** – garante que objetos **IDisposable** sejam fechados **automaticamente**, não precisa .close()
  + **Objetos IDisposable** não são gerenciados pelo **CLR**

**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**

* **Pilares**: Encapsulamento; Herança; Polimorfismo
* 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**
    - **Views**
* 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;
  + protected – permitir acesso aos **atributos/met.** somente a **subclasse**
  + 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;** }
* Associação: Composição e Agregação
* **LINQ p/ acessar BD / expressões lambda / delegates**

**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.

**Impressão de Objetos**

* ToString() – **override** ToString
* Classe StringBuilder sb = **new** StringBuilder() – equivalente a TStringList

**Herança/Extensão – símbolo (:) - Associação entre Classes e não Objeto**

* Reuso / Polimorfismo
* **:** => equivalente ao **extends** do Java
* **class base** é a **superclasse**, é o **super**() no Java. **Subclasse** class derivada
* Relação é um
* Generalização / Especialização
* **Exemplo**: **Conta**/ContaJuridica; **Pessoa**/PessoaFisica/PessoaJuridica
  + class **ContaJuridica** : **Conta** {
    - public **double** LoanLimit { get; set; }
    - public **ContaJuridica**() {}
    - public **ContaJuridica**(int **number**, string **holder**, double **balance**, double **loanLimit**) : base(**number, holder, balance**) { LoanLimit = loanLimit; }

**Upcasting / Downcasting**

* **Upcasting** – casting da **subclasse** para a **superclasse - polimorfismo**
* **Downcasting** – casting da **superclasse** para a **subclasse – palavra is/as**
  + Métodos que recebem parâmetros genéricos, ex.: **Equals**
* **Exemplo**: **Conta**/ContaJuridica/ ContaPoupanca
  + **Conta** conta = new **Conta**(...);
  + **ContaJuridica** contaJur = new **ContaJuridica**(...);
  + **Conta** conta2 = **contaJur**; // upcasting
  + **Conta** conta3 = new **ContaJuridica**(...); // upcasting
  + **Conta** conta4 = new **ContaPoupanca**(...); // upcasting
  + **ContaJuridica** contaJu1 = (**ContaJuridica**) conta2; // downcasting
    - if ( conta4 **is** **ContaJuridica**) // downcasting; **poupança**
      * **ContaJuridica** contaJu2 = (**ContaJuridica**)conta3; ou
      * **ContaJuridica** contaJu2 = conta3 **as** **ContaJuridica**;

**Polimorfismo – implementação do método de uma super na subclasse**

* Sobrecarga/Sobrescrita **(overload);** métodos com mesmo nome e **assinatura diferente**
* Sobreposição – **virtual,** libera o método para sobreposição. **override/base**
  + **Class Conta // Superclasse**
    - public virtual void Saque(double quantia) {Saldo -= quantia + 5.0;}
  + **Class ContaPoupanca // Subclasse / override**
    - public override void Saque(double quantia) {Saldo -= quantia;}
  + **Class ContaPoupanca // Subclasse / override / base**
    - public override void Saque(double quantia) { **base.Saque(quantia);**
      * Saldo -= 3.0;}
* **sealed** class – evita que uma classe seja **herdada** – **proteção**
  + **Exemplo**: **sealed** **class** ContaPoupanca{...}
* **sealed** método – evita que um **método sobreposto** possa ser sobreposto novamente – **proteção, tem que ser em um método override**
  + **Exemplo**: public **selead** **override** void Saque(double quantia){...}

**Classes e Métodos Abstratos**

* Superclass abstrata – não pode ser instanciada – garantir a herança total
* Subclasses não abstratas – podem ser instanciadas
* Por padrão **notação UML**: o nome da class deve ficar em **ITÁLICO**
* Pode criar **List**<Conta> lista = **new** ...; **foreach** (Conta acc in lista) {...}
* **Exemplo**: **abstract** class **Conta**{...}
* Métodos abstratos: métodos que não possuem implementação
  + Quando a superclasse é muito genérica
  + Se a classe possuir pelo menos um método abstrato, então esta classe deve ser abstrata.
  + **Exemplo Classes**:
    - Figura: atributo(cor) e método Area() // superclasse
      * Retangulo: atributos (Largura e Altura) // subclass
      * Circulo: atributo(Raio) // subclass
    - public **abstract** double Area(); // superclasse
  + Não é necessário utilizar o **virtual** no método, pq eh abstract

**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**.

**Interface – Operações de uma classe/struct que deve implementar**

* Contrato / Somente Assinaturas, depois, serão criadas classes concretas que implementam a interface
* Motivo: Baixo Acoplamento e flexíveis
* **Exemplo**: **interface** **IShape** {double **Area()**; double **Perimeter()**;}
* **inversão de controle:** consiste em retirar da classe a responsabilidade de instanciar suas dependências.
* **Injeção de dependência por meio do construtor:** Forma de realizar a inversão de controle: um componente externo instancia a dependência, que é então injetada no objeto **pai**. Pode ser implementada de várias formas:
  + Construtor
    - RentalService rentalService = **new RentalService**(hour, day, **new BrazilTaxService()**)
  + Objeto de Instanciação(builder / factory);
  + Container / framework

Diagrama

O conteúdo gerado por IA pode estar incorreto.

**Aspectos em Comum Entre Herança e Interfaces**

* Relação é um
* Generalização / Especialização
* Polimorfismo
* Diferença Fundamental:
  + **Herança**: Reuso
  + **Interface**: Contrato a ser cumprido

Diagrama

O conteúdo gerado por IA pode estar incorreto.

Diagrama

O conteúdo gerado por IA pode estar incorreto.

Diagrama

O conteúdo gerado por IA pode estar incorreto.

**Interface IComparable – Comparação de Objetos**

* Comparação de Objetos, por exemplo, utilizando o **List<Obj>.Sort();**
* <https://learn.microsoft.com/en-us/dotnet/api/system.icomparable?view=net-9.0&redirectedfrom=MSDN>

**Generics; Set; Dicionary**

* Generics: permitem que classes, interfaces e métodos possam ser parametrizados por tipo.
  + Reuso
  + Type Safety – Erro de compilação, elementos de tipos diferentes
  + Performance
  + Uso Comum: Coleções
  + Exemplos:
    - **class** ClasseService**<T>**{...}
    - **public** **T** Max**<T>**(List**<T>** list) {…} // **método**
    - **public** **T** Max**<T>**(List**<T>** list) **where** **T** : IComparable{…}
  + **GetHashCode**: operações da classe object para comparar objetos, mais rápido e resposta não 100%. E retorna um número inteiro. **Garante quando os objetos são diferentes**, por conta da colisão, lógica do algoritmo, **se derem iguais**, faz um **novo teste** utilizando o **método Equals()**.
    - **string** a = "julia"; **string** b = "josy"; Console.**WriteLine**(a.**GetHashCode**()); // Número inteiro

Console.**WriteLine**(b.**GetHashCode**()); // Número inteiro

* + **Equals**: operações da classe object para comparar objetos, mais lento e resposta 100%.
    - **string** a = "julia"; **string** b = "josy"; Console.**WriteLine**(a.**Equals**(b)); // False
* Set – **using System.Collections.Generics**;
  + Não admite repetições; elementos não possuem posição; Inserção e remoção são rápidas; operações: **interseção, união e diferença**.
  + HashSet<T>
    - Armazenamento em **tabela Hash**; extremamente rápido: inserção, remoção e busca **O(1)**; A ordem dos elementos não são garantidas.
  + SortedSet<T>
    - Armazenamento em árvore; rápido: inserção e remoção e busca O(log(n)); os elementos são armazenados ordenadamente conforme implementação **IComparecer**<T>.
  + Metodos**:.Add;.Clear**;**.Contains**;**.Count**;**UnionWith**(other);**IntersectWith**(other);**ExceptWith**(other);**Remove**(T);**RemoveWhere**(predicate) – **condição**.
  + **HashSet**<string> set = **new** **HashSet**<string>(); set.**Add**("TV");
  + **SortedSet**<int> a = **new** **SortedSet**<int>() {0, 2, 4, 6, 8};
  + **IEnumerable**<T> // para comparação de objetos numéricos.
* Dicionary<**TKey,TValue**> coleção de pares chave/valor.
  + Baseada em Hash
  + não admite repetições do objeto chave; os elementos são indexados pelo objeto chave – não possuem posição; acesso: inserção, remoção e busca O(1) são rápidas.
  + uso comum: cookies; local storage; qualquer modelo chave/valor.
* SortedDicionary<**TKey,TValue**> coleção de pares chave/valor.
  + Baseada em árvore
  + Remoção, inserção e busca: O(log(n)); Implementação IComparer<T>
  + **Dictionary**[key]; .**Add**(key, value); .**Clear**(); .**Remove**(key). .**ConstainsKey**(key) etc
  + **Dictionary**<string, string> cookies = **new** **Dictionary**<string, string>(); cookies["users"] = "Julia";

**Extension Methods**

* São métodos que estendem a funcionalidade de um tipo, sem precisar alterar o código fonte deste tipo, nem herdar desse tipo;
* Necessário criar uma classe estática;
* Criar um método estático
* O primeiro parâmetro do método deverá ter o prefixo this, seguido da declaração de um parâmetro do tipo que se deseja estender. Esta será uma referência para o próprio objeto.
* Ex.:
  + **static** **class** DateTimeExtensions {
    - public **static** string **ElapsedTime**(**this** DateTime thisObj) {
      * TimeSpan duration = DateTime.Now.**Subtract**(thisObj);
      * if ( duration.**TotalHours** > 24.0 )
        + **return** duration.**TotalHours**.ToString("F1", CultureInfo.InvariantCulture) + " hours";
      * else
        + **return** duration.**TotalDays**.ToString("F1", CultureInfo.InvariantCulture) + " days";
      * }
    - }

**Paradigmas / Programação Funcional / Lambda**

* **Paradigmas**
  + Imperativa(**como**) – **Comandos**: C; COBOL; Pascal; Fortran
  + Procedural – Comandos através de procedures: COBOL
  + OO – Orientada a objetos: Object Pascal; C++; Java <8; C#<3
  + Orientada a Eventos: Delphi;
  + POA – Orientada a Aspectos – Módulo que encapsula uma preocupação transversal, tais como: log de execução; tratamento de exceções; segurança/autenticação; transações; monitoramento de performance. Complemento do POO.
  + Funcional/Declarativa(**o quê**) – **Expressões**; Funções; Lambda: MathLab; R; Haskell; Clousure; Clean; Herlang
  + Lógica – Expressões; Funções; True/False: Prolog
  + Multiparadigmas: Javascript; Python; Java >8+; C#>3+; Go; Ruby
* Cálculo Lambda: Formalismo matemático base da programação funcional
* Expressão Lambda: **Função anônima** de primeira classe;
* **Delegates**: é um tipo referência
  + Referência (**com type safety -** **segurança do tipo**), para um ou mais métodos
    - **delegate** double **BinaryNumericOperation**(**double n1, double n2**);
    - BinaryNumericOperation op = CalculationService.Metodo;
      * Na class servisse, deve existir métodos com assinatura específica, que **retornam double** **com dois parâmetros do tipo double**;
    - Ou **BinaryNumericOperation** op = **new** **BinaryNumericOperation**(**CalculationService**.**Sum**);
    - double result = op.(a, b); **ou** double result = op.Invoke(a, b);
    - **Multicast delegates**
      * Para vários métodos void
      * Operador +-, na ordem que foram adicionados
      * Ex.:
        + delegate void **BinaryNumericOperation** op(double a, double b);
        + double a = 5;
        + double b = 6;
        + **BinaryNumericOperation** op **CalculationService**.**ShowSum**;
        + op += **CalculationService**.**ShowMax**;
        + op(a, b); // vai executar os dois métodos
  + Uso comum: comunicação entre objetos; programação funcional
    - Action
    - Func
    - Predicate
* **Predicate**: representa um objeto do tipo e retorna um valor booleano
  + É uma referência para um método;
  + public delegate bool **Predicate**<in T>(T obj);
  + list.RemoveAll(p => p.Price >= 50.00);
* **Action**: representa um método void que recebe ZERO ou mais argumentos
  + public delegate void Action();
  + public delegate void Action<in T>(T obj);
  + public delegate void Action<in T1, in T2>(T1 arg1, T2 arg2);
  + public delegate void Action<in T1, in T2, in T3>(T1 arg1, T2 arg2, T3 arg3);
* **Func**: representa um método que recebe ZERO ou mais argumentos e retorna um valor. using System.Collections.Generics; using System.Linq;
  + public **delegate** TResult Func<**out** TResult >();
  + public **delegate** TResult Func<**in** T, **out** TResult>(T obj);
  + public **delegate** TResult Func<**in** T1, **in** T2, **out** TResult>(T1 arg1, T2 arg2);
  + Ex.:
    - **Func**<**Product**, **string**> func = **NameUpper**; ou
    - **// Func**<**Product**, **string**> func = p => p.Name.**ToUpper**();
    - **List**<string> result = list.**Select**(func).**ToList**();
    - static string **NameUpper**(**Product** p) {

return p.Name.**ToUpper**(); }

**Linq - Language Integrated Query – System.Linq**

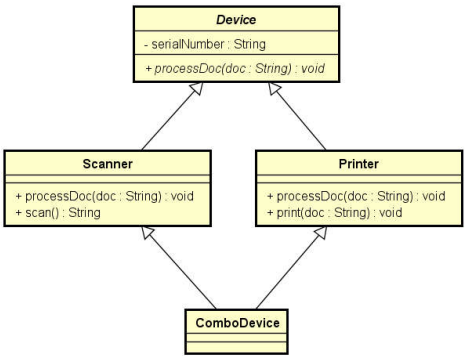
* Conjunto de tecnologias diretamente da linguagem C#:
  + Coleções; Objetos
* Expressões Lambdas
* Expressões de sintaxe similar à SQL
* Três Passos: **DataSource** => **Query** => **Execution**
  + Criar data source:
    - coleção, array, recurso de E/S, etc
  + Defina query
  + Executar a query – **foreach** ou alguma operação terminal, execução tardia(**lazy**).
* Exemplo:
  + int[] numbers = **new** int[] {2, 4, 7, 9, 12};

**var** result = numbers.**Where**(x => x % 2 == 0).**Select**(x => x \* 10); // **IEnumerable<int>** no lugar do var

**foreach**(int x in result) { Console.WriteLine(x); }

* **Algumas Operações**:
  + Filtering; Where, OfType
  + Sorting: OrderBy
  + Set: Distinct, Except
  + Quantification: All, Any
  + Projection: Select, SelectMany
  + Partition: Skip, Take
  + Join: Join, GroupJoin
  + Grouping: GroupBy
  + Equality: SequenceEquals
  + Element: First, Last, ElemenAt
  + Conversions: AsEnumerable
  + Concatenation: Concat
  + Aggregation: Aggregate, Avarege, Count, Max, Min, Sum

**Problema do Diamante: Herança e Interfaces**



**Nota**: Solução Interface

**Projeto API Web**

**Projeto 1**

**Diagrama

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 – ASP .NET Core MVC e Entity Framework - ORM**

**SalesWebMVC Home – Departments – Sellers – Sales – About – Contact**

**Projeto API Web – Framework para criação de aplicação Web**

* Controllers
* Views
* Models

**Diagrama

O conteúdo gerado por IA pode estar incorreto.**

**Tabela

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