C#

Brenno Barros Soares, Wallace Kruger Jr e Wanderson Ralph

Introdução

História, Visão Geral e Mini Tutorial de C#

História do C#

No final da década de 90, a Microsoft tinha diversas tecnologias e LPs.

Toda vez que um programador precisava migrar para uma nova linguagem, era necessário aprender a nova linguagem, bibliotecas, conceitos etc.

Para solucionar esse problema a Microsoft recorreu à linguagem Java.

A Microsoft assinou um acordo de licenciamento com a Sun para utilizar o Java em ambiente Windows.

História do C#

Porém Java não se comunicava bem com as bibliotecas de código de máquina que já existiam.

Por isso a Microsoft decidiu criar sua própria implementação de Java, chamado J++.

Mas J++ era uma versão de Java que só podia ser executada em ambiente Microsoft.

A Microsoft foi processada e foi obrigada a repensar a estratégia.

História do C#

Salvação da lavoura: Plataforma .NET

Ambiente de desenvolvimento da Microsoft projetado para trabalhar com diferentes LP’s, compartilhando um mesmo conjunto de bibliotecas.

Isso facilitava o programador migrar de uma linguagem para a outra.

Além a plataforma, a empresa precisava de uma nova LP, começaram a trabalhar no COOL (C-like Object Oriented Language). Baseada em Java, C, C++, Smalltalk, Delphi e VB.

E assim nasceu o C#

O projeto COOL foi lançado como linguagem C#

Muita gente acredita que o nome veio da superposição de 4 símbolos +

C++++

Mas na verdade o símbolo se refere ao sinal musical sustenido.

Visão geral

C# é uma linguagem de programação surgida em 2001 interpretada e multi-paradigma (imperativa, funcional, declarativa, **orientada a objetos** e genérica)

Foi desenvolvida pela Microsoft como parte da plataforma .NET

Alia o poder de C++ com a simplicidade de Visual Basic

Visão geral

O código fonte é compilado para Common Intermediate Language (CIL), este é interpretado pela máquina virtual Common Language Runtime (CLR)

É projetada para funcionar na Common Language Infrastructure da plataforma .NET framework

Objetivos de C#

Ser simples, moderna, de propósito geral e orientada a objetos

Deve fornecer suporte para princípios de engenharia de software (ser: fortemente tipada, verificar limites de array, verificar variáveis não inicializadas, coleta automática de lixo etc)

Ser portável

Ser eficiente em desempenho e memória, mas sem competir diretamente com C ou assembly

O que eu preciso para programar em C#?

Para executarmos uma aplicação C#, precisamos da máquina virtual da linguagem, além das bibliotecas do .NET framework

Ao instalarmos o Visual Studio, todo esse ambiente de execução de programas é automaticamente instalado

No caso de querermos executar programas apenas (computador do cliente), precisamos apenas do ambiente de execução. Para isso, podemos utilizar o .Net Framework Redistributable, fornecido pela própria Microsoft

Mas e o Mac? E o Linux?

Existem vários projetos que permitem tanto programar quanto rodar aplicações C#

Os principais são o Mono Project e o .NET Core

Pronto!

Com o ambiente configurado, é só criar um novo projeto C#

O Visual Studio, Mono e .NET Core, tem a interface base bem parecida, e são bem intuitivos

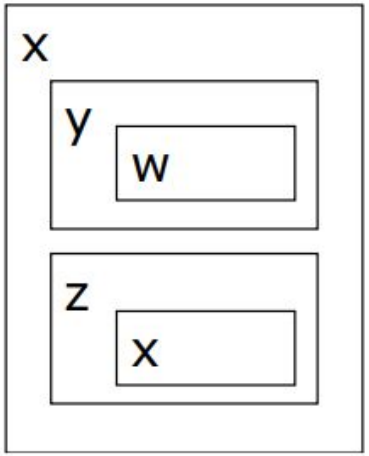
Hello World

```
// Um "Hello World!" em C#
using System;
namespace HelloWorld
{
    class Hello
    {
        static void Main()
        {
            Console.WriteLine("Hello World!");
        }
    }
}
```

Amarrações

Escopo estático, blocos aninhados (Algol-Like)

```
namespace N
{                   // escopo de namespace, identificadores de grupos
  class C
  {                 // escopo de classe, define/declara variáveis membro e funções
    void f (bool b)
    {               // escopo do bloco mais externo (função), inclui instruções executáveis
      if (b)
      {             // escopo do bloco mais interno por declarações executadas condicionalmente
                ...
      }
    }
  }
}
```



Modificadores de acesso	Descrição
public	Membro é acessado de qualquer lugar
private	Membro pode ser acessado somente dentro da classe que o define
internal	Membro pode ser acessado por qualquer membro que esteja no assemblv
protected	Pode ser acessado dentro da classe que o define e pelas classes que a herdam
protected internal	Pode ser acessado por todos membros do assembly ou por membros que o herdem.

```
public class AtualizadorDeContas
{
    public void Atualiza(Conta conta)
    {
        ...
    }
}
```

C# é case sensitive!

```
int AA = 20;

double Aa = 30.5;

int aA = 15;

double aa = 40.1;
```

Convenção de nomes (Pascal Casing)

- Parâmetros e Objetos:

```
string nomeCompleto;

double valorDesconto;
```

- Métodos / Classes:

```
public void CalculaDesconto()

class ContaCorrente
```

Para saber mais: http://msdn.microsoft.com/en-us/library/ms229040(v=vs.110).aspx

Palavras Reservadas:

abstract	as	base	bool
break	byte	case	catch
char	checked	class	const
continue	decimal	default	delegate
do	double	else	enum
event	explicit	extern	false
finally	fixed	float	for
foreach	goto	if	implicit
in	in (modificador genérico)	int	interface
internal	is	lock	long
namespace	new	null	object
operator	out	out (modificador genérico)	override
params	private	protected	public
readonly	ref	return	sbyte
sealed	short	sizeof	stackalloc
static	string	struct	switch
this	throw	true	try
typeof	uint	ulong	unchecked
unsafe	ushort	using	using static
virtual	void	volatile	while

Palavras Chaves:

add	alias	ascending
async	await	descending
dynamic	from	get
global	group	into
join	let	nameof
orderby	parcial (tipo)	partial (método)
remove	select	set
value	var	when (condição de filtro)
where (restrição de tipo genérico)	where (cláusula de consulta)	yield

Valores e Tipos de Dados

Valores e Tipos de Dados

- C# possui tipagem estática e forte.Todas as variáveis e constantes têm um tipo fixado em suas definições (em tempo de compilação), assim como cada expressão que é avaliada como um valor;
- Em C# existem duas categorias de tipos, os tipos valores e tipos referência, o tipo valor armazenam dados, tipos de referência armazenam uma referência, que apontam para os dados, que vivem em algum outro lugar na memória do computador;
- O sistema de tipos de C# é unificado, ou seja, seja qual for o seu tipo todas as variáveis podem ser tratadas como objetos, pois todos os tipos derivam direta ou indiretamente do tipo base objeto. Isto é, uma classe que representa a base para todos os objetos do sistema, a classe Object.

Tipos primitivos

Tipo	Tamanho	Valores Possíveis
bool	1 byte	true e false
byte	1 byte	0 a 255
sbyte	1 byte	-128 a 127
short	2 bytes	-32768 a 32767
ushort	2 bytes	0 a 65535
int	4 bytes	-2147483648 a 2147483647
uint	4 bytes	0 to 4294967295
long	8 bytes	–9223372036854775808L to 9223372036854775807L
ulong	8 bytes	0 a 18446744073709551615
float	4 bytes	Números até 10 elevado a 38. Exemplo: 10.0f, 12.5f
double	8 bytes	Números até 10 elevado a 308. Exemplo: 10.0, 12.33
decimal	16 bytes	números com até 28 casas decimais. Exemplo 10.991m, 33.333m
char	2 bytes	Caracteres delimitados por aspas simples. Exemplo: 'a', 'ç', 'o'

Os tipos listados nesta tabela são conhecidos como tipos primitivos ou value types da linguagem C#

Tabela de Valores Padrão

Tipo de valor	Valor padrão
bool	`false`
byte	0
char	\0'
decimal	0,0M
double	0,0D
enum	O valor produzido pela expressão (E)0, em que E é o identificador de enumeração.
float	0,0F
int	0
long	0L
sbyte	0
short	0
struct	O valor produzido pela configuração de todos os campos tipo-valor para seus valores padrão e todos os campos tipo-referência para null.
uint	0
ulong	0
ushort	0

Tipos de Valor

- Tipo Inteiro:

```
int valor = -500;

uint valorPositivo = 400;

short valorPequeno = 2;

long valorGrande = 99999999999;

byte b = 1;
```

Tipos de Valor

- Tipo caractere:

```
char letra = ‘a’;

char simbolo = ‘,’ ;
```

- Tipo booleano:

```
bool flamengoGanha = true ou false;
```

Tipos de Valor

- Tipo decimal:

```
decimal pi = 3.14159m;
```

Obs: Números com até 28 casas decimais.

- Tipo ponto flutuante:

```
float a = 10.0f;

double d = 10.0;
```

Tipos de Valor

- Estrutura:

```
struct livro
{
    public string nome;
    public string autor;
    public double valor;
}
```

Tipos de Valor

- Enumerado:

```
enum dias {dom, seg, ter, qua, qui, sex, sab};

enum mesLet { mar, abr, mai, jun, ago, set, out, nov };
```

Tipos de Referência

- String:

```
string titulo = "Arquitetura e Design de Software!";

string titulo = "Arquitetura" + " e " + " Design de Software";

titulo += “!”;
```

Obs: Strings são imutáveis!

Tipos de Referência

- Arrays:

```
int[] numeros = new int[5];

numeros[0] = 1;
numeros[1] = 600;
numeros[2] = 257;
numeros[3] = 12;
numeros[4] = 42;

int[] numeros = new int[] { 1, 600, 257, 12, 42};
```

Tipos de Referência

- Arrays Multidimensionais (Matrizes):

```
int[,] x = new int[2, 3];

x[1, 2] = 10;

int [, ,] dimensoes = new int[ 4, 5, 6];

string[,] nomes = { { “Breno”, “Wallace”}, { “Wanderson”, “Vitor”} };

Console.WriteLine(nomes[0,1]); → Wallace
```

Tipos de Referência

- Classes:

```
class Conta{
    public int numero;
    public string titular;
    public double saldo;
    public void Saca(){...}
...}
```

Na main()

```
...Conta c = new Conta();
c.saldo += 100;
Console.WriteLine(c.saldo);
```

→ **100**

Tipos de Referência

- Conjuntos:
  - Listas:

```
List<Conta> lista = new List<Conta>();

Conta c1 = new Conta();

lista.Add(c1);

lista.RemoveAt(0);
```

Tipos de Referência

- Classes Abstratas:

```
public abstract class Conta
{
  public virtual void Saca(double valor){
    //não faz nada
  }
//  ...
}
```

```
public class ContaCorrente : Conta
{
  public override void Saca(double
valor)
  {
    this.Saldo -= (valor + 0.10);
  }
//  ...
}
```

```
public class ContaPoupanca :
Conta
{
  public override void Saca(double
valor)
  {
    this.Saldo -= valor;
  }
//  ...
}
```

Tipos de Referência

- Delegates:

```
public delegate void TestDelegate(string message);

public delegate int TestDelegate(int m, long num);
```

- Ponteiros:

Uso do Unsafe;

Variáveis e Constantes

Caracterização de variáveis em C#

Variáveis podem ser caracterizadas por:

-Nome

-Endereço

-Tipo

-Valor

-Tempo de vida

-Escopo de visibilidade

Endereço

Normalmente, não se pode acessar o endereço de uma variável em C#, a não ser que você utilize o modificador unsafe e utilize ponteiros:

```
// compilar com: /unsafe

class UnsafeTest
{
    // Método unsafe:
    unsafe static void
SquarePtrParam(int* p)
    {
        *p *= *p;
    }
```

Endereço

```
unsafe static void Main()
{
    int i = 5;
    // Método unsafe: Operador address-of (&):
    SquarePtrParam(&i);
     var x = &i;
    Console.WriteLine(i);
    Console.WriteLine("{0}",(int)x)
    }
}
// Outputs: 25 e o endereço da variável
```

Tipo

Variáveis podem ser de especificação explícita ou semântica (var)

```
// i é compilada como int
var i = 5;

// s é compilada como string
var s = "Hello";

// a é compilada como int[]
var a = new[] { 0, 1, 2 };
```

Tipo

Mas é importante entender que a palavra-chave “var” não significa “variante” e não indica que a variável é vagamente tipada ou de associação tardia.

Apenas significa que o compilador determina e atribui o tipo mais apropriado.

Variáveis Globais

Declarando uma variável global dentro de uma classe:

```
public class VarGlobais
{
    public static int x = 100;
    public static int y;
}
```

Obs: Para acessar usa-se: VarGlobais.x ou VarGlobais.y

Constantes

Campos cujos valores são definidos em tempo de compilação e nunca podem ser alterados.

```
static class Constants
{
    public const double Pi = 3.14159;
    public const int SpeedOfLight = 300000;
// km por segundo

}
```

Constantes

```
class Program
{
    static void Main()
    {
        double radius = 5.3;
        double area = Constants.Pi * (radius * radius);
        int secsFromSun = 149476000 / Constants.SpeedOfLight;
    }
}
```

Coletor de Lixo

C# oferece coletor de lixo

Explicando resumidamente como o coletor de lixo funciona:

Ele é dividido em 3 gerações:

Geração 0: Geração mais “jovem”, contém objetos de vida útil curta. Ex: Variável temporária.

A coleta de lixo ocorre com mais frequência nessa geração.

Coletor de Lixo

Geração 1: Contém objetos de vida útil curta, e serve como buffer entre objetos de vida útil curta e longa.

Geração 2: Contém objetos de vida útil longa. Ex: constantes e variáveis globais

Coletor de Lixo

Coletas de lixo ocorrem em gerações específicas conforme as condições permitem. Coletar na geração 2, coleta objetos nessa geração e em suas gerações mais jovens. Uma coleta na geração 2 também é conhecida como coleta de lixo completa.

Objetos não recuperados em uma coleta são os sobreviventes e são promovidos à próxima geração. (Exceto se já estiverem na geração 2)

Serialização

Existem **várias** maneiras de serializar em C#

As duas mais utilizadas são a serialização Binária e a serialização XML

Basicamente, na serialização binária, a performance é melhor, porém o código é mais complexo, menos legível e menos portável.

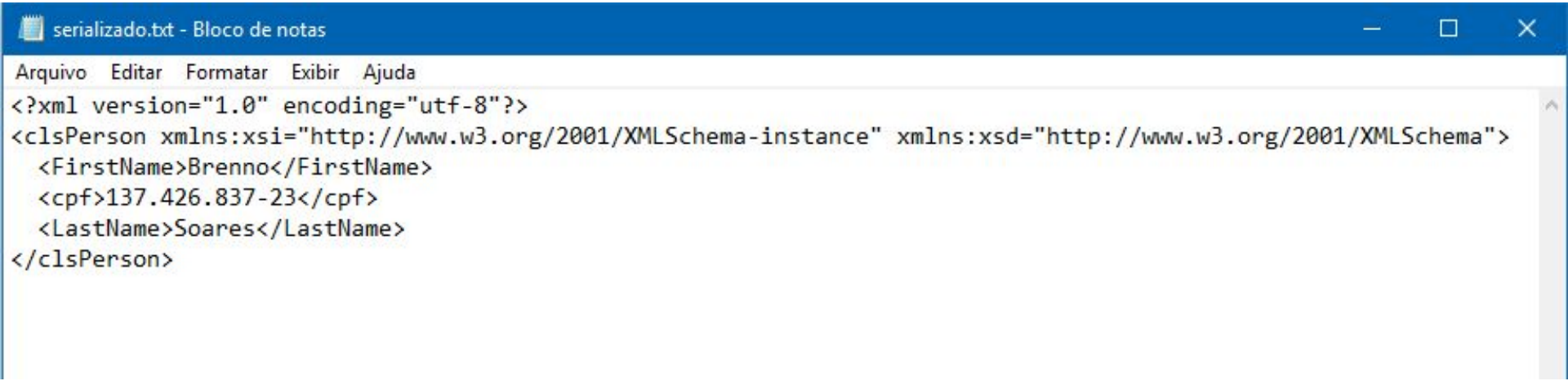
A serialização XML perde em desempenho, mas produz um código mais legível, e mais flexibilidade e portabilidade.

Exemplo

```
namespace Serialização
{
    public class clsPerson
    {
        public string FirstName;
        public string cpf;
        public string LastName;
    }
```

```
        static void Main(string[] args)
        {
            clsPerson p = new clsPerson();
            StreamWriter arquivo = new
StreamWriter(@"C:\Users\Brenno\serializado.xml");
            p.FirstName = "Brenno";
            p.LastName = "Soares";
            p.cpf = "137.426.837-23";
            System.Xml.Serialization.XmlSerializer x = new
System.Xml.Serialization.XmlSerializer(p.GetType());
            x.Serialize(arquivo, p);
        }
```

Saída da Serialização



Desserialização

```
static void Main(string[] args)
        {
            clsPerson p = new clsPerson();

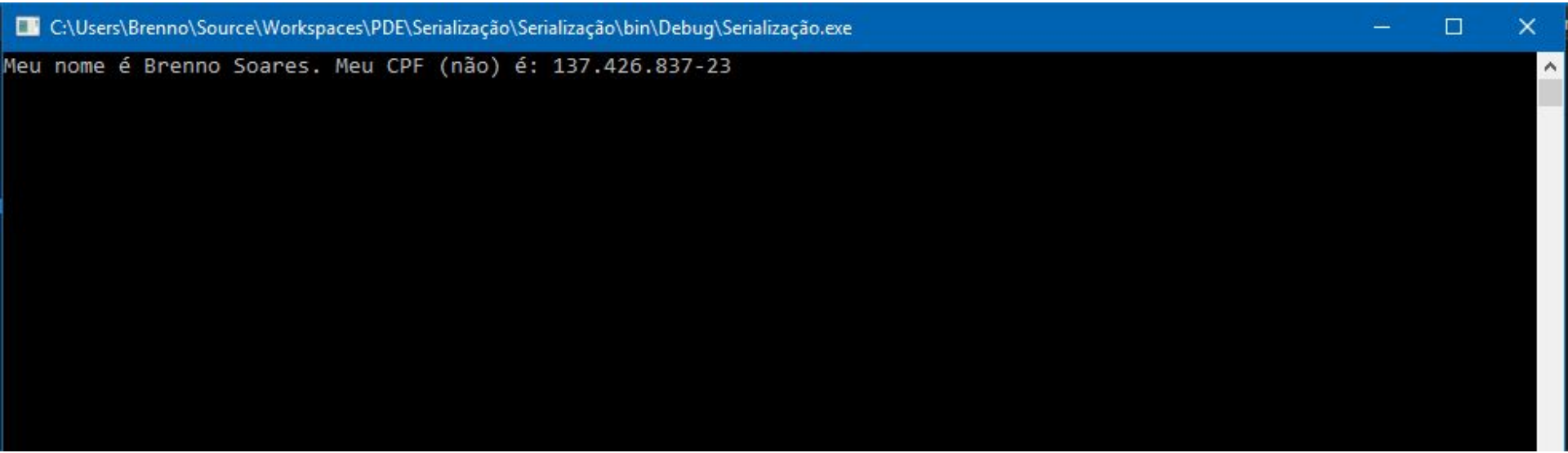
            XmlSerializer serializer = new XmlSerializer(typeof(clsPerson));

            StreamReader reader = new
StreamReader(@"C:\Users\Brenno\serializado.xml");
```

Desserialização

```
p = (clsPerson)serializer.Deserialize(reader);
        Console.WriteLine("Meu nome é " + p.FirstName + " " + p.LastName +
". Meu CPF (não) é: " + p.cpf);
        Console.ReadKey();
    }
```

Resultado


C:\Users\Brenno\Source\Workspaces\PDE\Serialização\Serialização\bin\Debug\Serialização.exe
Meu nome é Brenno Soares. Meu CPF (não) é: 137.426.837-23

Expressões

Agregações

- Constrói um valor a partir de seus componentes
  - ```
    int[] inteiros = { 1, 2, 3 };
    ```
  - ```
    IList<string> lstFrutos = new List<string>() { "Goiaba", "Jaca"};
    ```
  - ```
    Pessoa p = new Pessoa() { Nome =  "Wanderson" };
    ```
  - ```
    float[] floats = { i/2, i + 2, i * 3 };
    ```

Aritméticas

```
float f;
int num = -9;
f = num / 6;
f = num / 6.0f;
f = 3 * num;
f = 1 + 2;
f = 1 - 2;
f = +3;
num = 10 % 3;
```

Binárias

```
int j;
j = ~0;      // j = -1
j = 10 & 2; // j = 2
j = 10 | 2; // j = 10
j = 10 ^ 2; // j = 8
j = 10 << 2;// j = 40
j = 10 >> 2;// j = 2
```

Condicionais

```
string str = j % 2 == 0 ? "Par" : "Impar";
```

Booleanas

```
bool c = a && b;
bool c = a || b;
bool c = a ^ b;
bool c = !a;
```

Relacionais

```
int maior = a > b ? a : b;

int menor = a < b ? a : b;

bool c = a >= b;

bool c = a <= b;
```

Condicionais nulos

Testa a presença de nulos antes de executar uma operação de acesso de membro (?.) ou de índice (?[])

```
Candidato candidato;
Eleicao eleicao;
int? nvotos = null; //Permite que um int seja nulo
nvotos = eleicao?.Candidatos?[0].Partido?.TotalVotos;
```

Condicionais nulos

- Retorna null se alguma das verificações acusar nulo.
- Curto-Circuito
- Em vez de fazer um if antes de chamar um método:

```
candidato?.Imprimir();
```

- Não irá imprimir se candidato for nulo

Coalescência nula

Retorna o operando esquerdo se o operando não for nulo; caso contrário, ele retornará o operando direito

```
eleicao.Candidatos[0] = candidato ?? new Candidato();
```

Funções

```
float Dividir(float dividendo, float divisor){
    return dividendo / divisor;
}

float f = Dividir(1, 2);
```

Delegate

- É como ponteiro para funções em C
- Mas é fortemente tipado

```
delegate float Operacao(float a, float b);

Operacao op = Dividir;
f =  op(1, 2);  //f = 0,5

op = Somar;
f = op(1, 2);   //f = 3,0
```

Função Anônima ou Lambda

- Para passar um bloco de código que pode ser utilizado por um método, o C# introduziu as funções anônimas ou lambdas.
  - `(Candidato c) => { return c.Eleito == true; }`
- E podemos passar essa função para um método Filtra
  - `Candidatos.Filtra((Candidato c) => { return c.Eleito == true; });`
- No caso de C#
  - `Candidatos.Where(c => c.Eleito);`

Lambda

```
List<string> lst = new List<string>() { "Ok", "teste", "ok" };

lst.ForEach(  p => Console.WriteLine(p)  );

int nOk = lst.Count(p => p == “ok”); //nOk  =  2
```

Filtros - LINQ

- Consulta integrada à linguagem
- Para implementar um filtro, lambda pode ficar muito complexo
- Então c# possui uma sintaxe baseada em SQL
  - `from c in eleicao.Candidatos`
- Adicionando uma condição
  - ```
    from c in eleicao.Candidatos
        where c.Eleito
    ```
    ```

Filtros - LINQ

```
from c in eleicao.Candidatos
join p in eleicao.Partidos
on c.Partido.Nome equals p.Nome
    where c.Eleito && c.NumeroVotos > 1000
    orderby c.Nome descending
    select new { c.Numero, p.Nome };
```

Referenciamento

- [ ] Acessar um elemento de vetor. Ex: vet[0]
- * Desreferenciamento. Ex: *x
- . Acesso a membro. Ex: x.y
- -> Combina a desreferência de ponteiro e o acesso de membro. Ex: x->y
- & Endereço do operando. Ex: &x

Categóricas

- Realizam operações sobre o tipo de dados:
  - Conversão de Tipo
    - **(T)x:**
      - `double a = doble(intSize); //4.0`
    - **as:**
      - `string a = b as string;`
      - **retorna null se não for possivel converter;**

Categóricas

- **Typeof**: Objeto System.Type que representa o operando.
  - `Type t = typeof(ExampleClass);`
- **defalt(T):** Retorna o valor padrão do tipo T
  - `var s = default(int);//0`
- **Sizeof** : Tamanho do tipo
  - `int intSize = sizeof(int); //4`

Categóricas

```
bool a = expr is type;
```

- Verdadeiro se:
  - expr for uma instância do mesmo tipo que type.
  - expr for uma instância de um tipo derivado de type.
  - *expr* é uma instância de um tipo que implementa a interface de *type*.

Categóricas

```
public class Class1 : IFormatProvider {}

var cl1 = new Class1();
Console.WriteLine(cl1 is IFormatProvider);//True
Console.WriteLine(cl1 is Object);         //True
Console.WriteLine(cl1 is Class1);         //True
Console.WriteLine(cl1 is Class2);         //False
```

Categóricas

- new
- nameof

Curto Circuito

```
for (int i = 0; i < 100; i++)
   a = i++;
//a = 0

if (b < 2 * c || a > c)
{
   a++;
}
```

Comandos

Atribuição

- Simples

```
int a = 0, b = 1, c = 10;
c = a + 3*b;
```

- Múltipla

```
c = a = 15;
```

Atribuição

- Composta

```
a += 3;
a -= 4;
a *= 5;
a /= 6;
a %= 7;
```

```
a &= b;
a |= b;
a ^= b;
a <<= b;
a >>= b;
```

Atribuição

- Unária

```
a++;
++a;
a--;
--a;
```

- Como expressão

```
while ((l = sr.ReadLine()) != null)
```

Condicionais

- if, else if, else

```
if (a == b)
{
   Console.WriteLine("Igual");
}
else if (a > b)
   Console.WriteLine("Maior");
else
   Console.WriteLine("Menor");
```

Condicionais

- switch, case, default

```
switch (a) {
      case 1:
        Console.WriteLine("1");
           break;
      case 2:
      case 3:
        Console.WriteLine("2 ou 3");
           break;
      default:
        Console.WriteLine("Nenhum anterior");
           break;
}
```

Interativos

- do while

```
int x = 0;
do
{
   Console.WriteLine(x);
   x++;
} while (x < 5);
```

Interativos

- While

```
int n = 1;
while (n < 6)
{
   Console.WriteLine(n);
   n++;
}
```

Interativos

- for

```
for (int i = 1; i <= 5; i++)
{
   Console.WriteLine(i);
}
```

Interativos

- foreach

```
foreach (Candidato candidato in eleicao.Candidatos)
{
   Console.WriteLine(candidato.Nome);
}
```

Desvios incondicionais - Escape

- break
  - Termina um loop ou switch

```
for (int i = 0; i < 10; i++){
    for (int j = 0; j < 10; j++){
        if (j == i)
            break;
     }
     //Vem pra cá
}
```

Desvios incondicionais - Escape

- continue
  - Vai para próxima iteração

```
for (int i = 1; i <= 10; i++){
      if (i < 9)
         continue;
      Console.WriteLine(i);
}
```

```
Output:
9
10
```

Desvios incondicionais - Irrestrito

- goto
  - Não é possível ir para dentro de uma função.

```
//Procurar
for (int i = 0; i < x; i++){
    for (int j = 0; j < y; j++){
        if (array[i, j].Equals(n))
            goto Encontrou;
    }
}
Encontrou:
    Console.WriteLine("\o/ {0}", n);
```

Desvios incondicionais

- yield
  - Quando você usa a palavra-chave yield em uma instrução, você indica que o método, o operador ou o acessador get em que ela é exibida é um iterador.

```
// Mostra as potências de 2
foreach (int i in Power(2, 8)){
     Console.Write("{0} ", i);
}
```

Desvios incondicionais - Escape

```
IEnumerable<int> Power(int number, int exponent){
        int result = 1;

        for (int i = 0; i < exponent; i++)
        {
            result = result * number;
            yield return result;
        }
}
// Output: 2 4 8 16 32 64 128 256
```

Desvios incondicionais - Escapes

- return
  - Sai de um método
- throw
  - Lança Exceção

Modularização

Namespaces

- C# utiliza o conceito de namespace para agrupar classes relacionadas.

```
namespace N2
{
    class A {}

    class B {}
}
```

Namespaces

- Segundo a convenção de nomes adotada pela Microsoft , os namespaces devem ter a forma:
  - **NomeDaEmpresa.NomeDoProjeto.ModuloDoSistema.**
  - No nosso caso
    - **UFES.LP.PDE.Importacao**

Namespaces

- Uso

```
System.Console.WriteLine("Olá");

using System;
Console.WriteLine("Olá");
```

Namespaces

- Aninhados

```
namespace N1
{
    namespace N2
    {
        class A {}

        class B {}
    }
}
```

```
namespace N1.N2
{
        class A {}

        class B {}
}
```

Namespaces

- Alias

```
namespace N3
{
    using R = N1.N2;

    class B: R.A {}
}
```

Modularização

Abstração de Dados

Tipo Simples

- Struct

```
public struct Livro{
    public decimal Preco;
    public string Titulo;
    public string Autor;
}
Livro livro = new Livro {
   Preco = 1, Titulo = "2", Autor = "3"
};
```

Tipo abstrato de dados - TADs

```
class Candidato
{
    public Candidato(string nome, string numero){
        this.Nome = nome;
        this.Numero = numero;
    }
    public string Nome { get; private set; }
    public string Numero { get; }
    public bool Eleito{ get; set; }
}
```

TADs

- Propriedades

```
public string Nome { get; }

public bool Eleito{ get; set; }

public string UltimoNome{
    get{
        return Nome.Split(' ').Last();
    }
}
```

TADs - Código cliente

```
static void Main(string[] args)
{
   Candidato c = new Candidato("Brenno", "1513289")
   {
      Eleito = true
   };
   Console.WriteLine(c.ToString());
}
```

Modularização

Abstração de Processos

Parâmetros

```
public float Dividir(float dividendo, float divisor){
    return dividendo / divisor;
}

float c = Dividir(2, 4);//Posicional

//Palavra Chave
c = Dividir(divisor: 2, dividendo: 4);

c = Dividir(dividendo: 2, divisor: 4);
```

Parâmetros

- Valores Defaults

```
public float Somar(float a, float b = 10){
    return a + b;
}
```

- Variáveis

```
public int Somatorio(params int[] valores){
    return valores.Sum();
}

Somatorio(1, 2, 3, 4, 5);
//Output: 15
```

Passagem

- Momento de passagem
  - Avaliação normal
- Tipos primitivos
  - Cópia
  - Unidirecional de entrada
- Se os parâmetros forem objetos
  - Passagem Bidirecional ou
  - Unidirecional de referência

Passando um argumento por referência

```
static void Method(ref int i)
{
    i = i + 44;
}

int val = 1;
Method(ref val);
Console.WriteLine(val);
// Output: 45
```

Passando um argumento por referência

- out
  - Faz o mesmo que ref, mas:
    - ref exige que a variável seja inicializada antes
- É possível sobrecarga do tipo:

```
public void SampleMethod(int i) { }
public void SampleMethod(ref int i) { }
```

- Mas não:

```
public void SampleMethod(out int i) { }
public void SampleMethod(ref int i) { }
```

Polimorfismo

Tipos de Polimorfismo:

- Ad-hoc:
  - Coerção
  - Sobrecarga
- Universal:
  - Paramétrico
  - Inclusão

Coerção

- Ampliação:

```
int valor = 1;
long valorGrande = valor;
```

OK!

Coerção

- Estreitamento:

```
int valor = 1;
short valorPequeno = valor;
```

ERRO DE COMPILAÇÃO!

Coerção

- Estreitamento:

```
int valor = 1;
short valorPequeno = (short) valor;
```

OK!

Sobrecarga

```
static int Somar(int x, int y)
{
  return x + y;
}
```

Sobrecarga

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

Sobrecarga

```
static long Somar(long x, long y)
{
  return x + y;
}
```

Sobrecarga

```
static void Main(string[] args)
{
        Console.WriteLine("***** Sobrecarga de Métodos *****\n");

        Console.WriteLine(Somar(10, 10));
        Console.WriteLine(Somar(3.5, 7.4));
        Console.WriteLine(Somar(8000000000000, 7000000000000));

        Console.ReadLine();
}
```

Paramétrico:

- Em C# suportado através do uso de Generics(namespace System.Collections.Generic):

//Criando uma lista de Objetos:

List<Object>lstObject = new List<Object>();

Inclusão:

- C# não suporta herança múltipla, possui interfaces e classes abstratas.

```
public class Conta
{
  public int Numero { get; set;}
  public double Saldo { get; private set; }
…
}
```

```
public class ContaPoupanca : Conta
{
...
}
```

Inclusão:

```
public class TotalizadorDeContas
{
  public double ValorTotal { get; private set; }

  public void Soma(Conta conta)
  {
    ValorTotal += conta.Saldo;
  }
}
```

Inclusão:

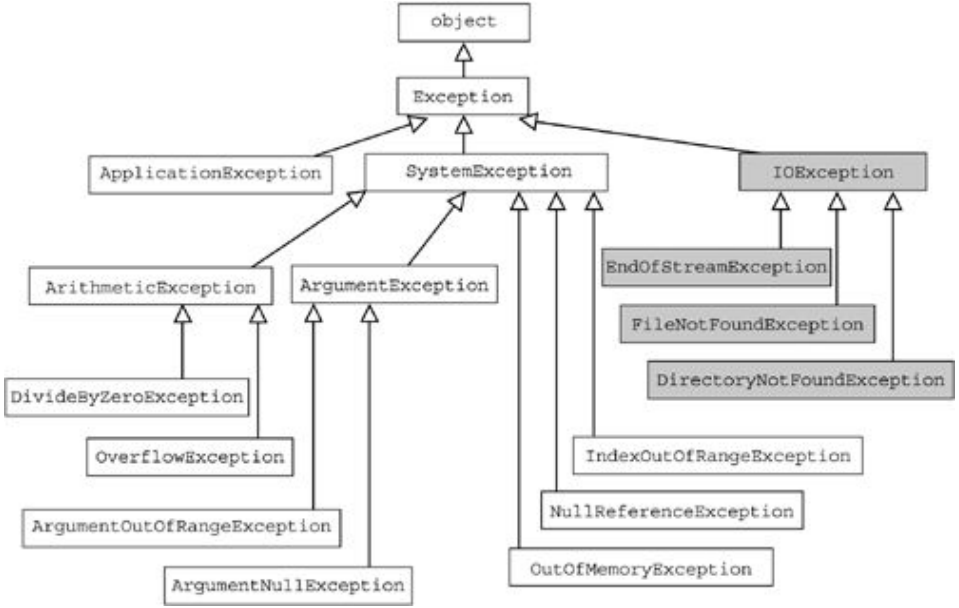
Na main():

```
Conta c1 = new Conta();
ContaPoupanca c2 = new ContaPoupanca();

TotalizadorDeContas t = new TotalizadorDeContas();
t.Soma(c1);
t.Soma(c2); // funciona!
```

Exceções

Hierarquia


object
Exception
ApplicationException
SystemException
IOException
ArithmeticException
ArgumentException
EndOfStreamException
FileNotFoundException
DirectoryNotFoundException
DivideByZeroException
OverflowException
IndexOutOfRangeException
ArgumentOutOfRangeException
NullReferenceException
ArgumentNullException
OutOfMemoryException

Exceções em C#

A linguagem não obriga o programador a tratar exceções

O que aumenta a redigibilidade, mas diminui a confiabilidade

Entrevista com Anders Hejlsberg, principal arquiteto de C# sobre os motivos pelo qual o time de desenvolvimento de C# optou por não implementar checked exceptions:
http://www.artima.com/intv/handcuffs.html

Exceções em C#

Para tratar exceções são utilizados os blocos:

- try: bloco que envolve instruções que podem lançar exceções
- catch: usado para definir um manipulador de exceções
- throw: gera exceções explicitamente
- finally: sempre é executado, geralmente utilizado para fechar fluxos que foram abertos no bloco try

Exceções em C#

Em C#, não existe a palavra-chave throws.

Se a exceção não for tratada, esta vai ser automaticamente “jogada para cima” até que seja capturada (ou não), interrompendo a execução

Exemplo 1

```
static void Main(string[] args)
        {
            StreamWriter sw = null;
               try
               {
                   sw = new StreamWriter(@"C:\Users\Breno\test.txt");
                   sw.WriteLine("Hello");
               }
```

```
catch (DirectoryNotFoundException ex)
{
    // Exceção mais específica primeiro
    Console.WriteLine(ex.Message);
    //Console.WriteLine(ex.ToString());
}
catch (IOException ex)
{
    // Mais geral depois
    Console.WriteLine(ex.Message);
    //Console.WriteLine(ex.ToString());
}
```

```
finally
{
    if (sw != null)
    sw.Close();
    Console.ReadKey();
}
}
```

Exemplo 2

```
class ExceptionTest
{
    static double SafeDivision(double x, double y)
    {
    if (y == 0)
        throw new System.DivideByZeroException();
    return x / y;
    }
```

```
    static void Main()
    {
    double a = 2, b = 0;
    double result;

    try
    {
        result = SafeDivision(a, b);
        Console.WriteLine("{0}
dividido por {1} = {2}", a, b, result);
    }
    catch
    {
        Console.WriteLine("1º
Mandamento da matemática: não
dividirás por 0");
    }
    }
}
```

Concorrência

Threads

- Um programa c# inicia numa thread principal automaticamente
- Suporte
  - using System.Threading;
    - Classe Thread
      - Start
      - Join
      - Sleep

Threads

```
public class Carro{
    private String nome;
    public Carro(String nome) { this.nome = nome; }
    public void Correr(){
        for (int i = 0; i < 10; i++){
          Thread.Sleep((int)(new Random().Next(0,1000)));
          Console.Write(nome);
            for (int j = 0; j < i; j++)
             Console.Write("--");
          Console.WriteLine(">");
       }
       Console.WriteLine(nome + " completou a prova.");
   }
}
```

Threads

```
Carro carroA = new Carro("Barrichelo");
Carro carroB = new Carro("Schumacher");

Thread t = new Thread(carroA.Correr);
t.Start();
carroB.Correr();//Thread Principal

t.Join();
```

Threads

Barrichelo>
Schumacher>
Schumacher-->
Barrichelo-->
Schumacher---->
Barrichelo---->
Schumacher------>
Barrichelo------>
Schumacher-------->
Barrichelo-------->
Barrichelo---------->
Schumacher---------->
Barrichelo------------>
Barrichelo completou a prova.
Schumacher------------>
Schumacher completou a prova.

Sincronização de thread

- Lock
  - Bloqueia um bloco de código até sua conclusão
- AutoResetEvent
  - Identificador de Espera
    - WaitOne
    - Set
- Semaphore

Sincronização de thread

```
public class BufferLimitado{
    private int capacidade;
    private int n;
    private int[] buffer;
    private int fim, ini;
    object bloqueador;
    AutoResetEvent autoEvent;

    public BufferLimitado(int capacidade){
        this.capacidade = capacidade;
        n = fim = ini = 0;
        autoEvent = new AutoResetEvent(false);
        bloqueador = new object();
        buffer = new int[capacidade];
    }
```

Sincronização de thread

```
public void inserir(int elemento)
{
    while (n == capacidade) autoEvent.WaitOne();
    buffer[fim] = elemento;
    fim = (fim + 1) % capacidade;
    lock (bloqueador)
    {
        n++;
    }
    autoEvent.Set();
}
```

Sincronização de thread

```
public int retirar(){
        while (n == 0) autoEvent.WaitOne();
        int elem = buffer[ini];
        ini = (ini + 1) % capacidade;
        lock (bloqueador)
        {
            n--;
        }
        autoEvent.Set();
        return elem;
    }
}
```

Sincronização de thread

```
public class Consumidor
{
  private BufferLimitado buffer;
  public Consumidor(BufferLimitado buffer)
  {
    this.buffer = buffer;
  }
  public void Consumir()
  {
    int elem;
    while (true)
    {
      elem = buffer.retirar();
      Console.WriteLine("consumido: " + elem);
      Thread.Sleep(new Random().Next(0, 1000));
    }
  }
}
```

```
public class Produtor{
  private BufferLimitado buffer;
  public Produtor(BufferLimitado buffer){
    this.buffer = buffer;
  }
  public void Produzir(){
    int elem;
    while (true)
    {
      elem = new Random().Next(0, 10000);
      buffer.inserir(elem);
      Console.WriteLine("produzido: " + elem);
      Thread.Sleep(new Random().Next(0, 1000));
    }
  }
}
```

Sincronização de thread

```
BufferLimitado buffer = new BufferLimitado(10);

Produtor produtor = new Produtor(buffer);
Consumidor consumidor = new Consumidor(buffer);

Thread t = new Thread(produtor.Produzir);

t.Start();
consumidor.Consumir();//Thread Principal

t.Join();
```

Sincronização de thread

produzido: 5398
consumido: 5398
produzido: 6653
produzido: 6653
consumido: 7351
produzido: 7351
produzido: 1952
produzido: 2476
consumido: 2476
produzido: 8539
consumido: 8539
produzido: 1397
……..

Avaliação de Linguagens

Avaliação de Linguagens

Critérios Gerais	C	C++	Java	C#
Aplicabilidade	Sim	Sim	Parcial	Parcial
Confiabilidade	Não	Não	Sim	Parcial
Aprendizado	Não	Não	Não	Não
Eficiência	Sim	Sim	Parcial	Parcial
Portabilidade	Não	Não	Sim	Sim

Avaliação de Linguagens

Critérios Gerais	C	C++	Java	C#
Método de projeto	Estruturado	Estruturado e OO	OO	Imperativo, funcional, declarativo, OO e genérica
Evolutibilidade	Não	Parcial	Sim	Sim
Reusabilidade	Parcial	Sim	Sim	Sim
Integração	Sim	Sim	Parcial	Sim
Custo	Depende da Ferramenta	Depende da Ferramenta	Depende da Ferramenta	Depende da Ferramenta
Escopo	Sim	Sim	Sim	Sim

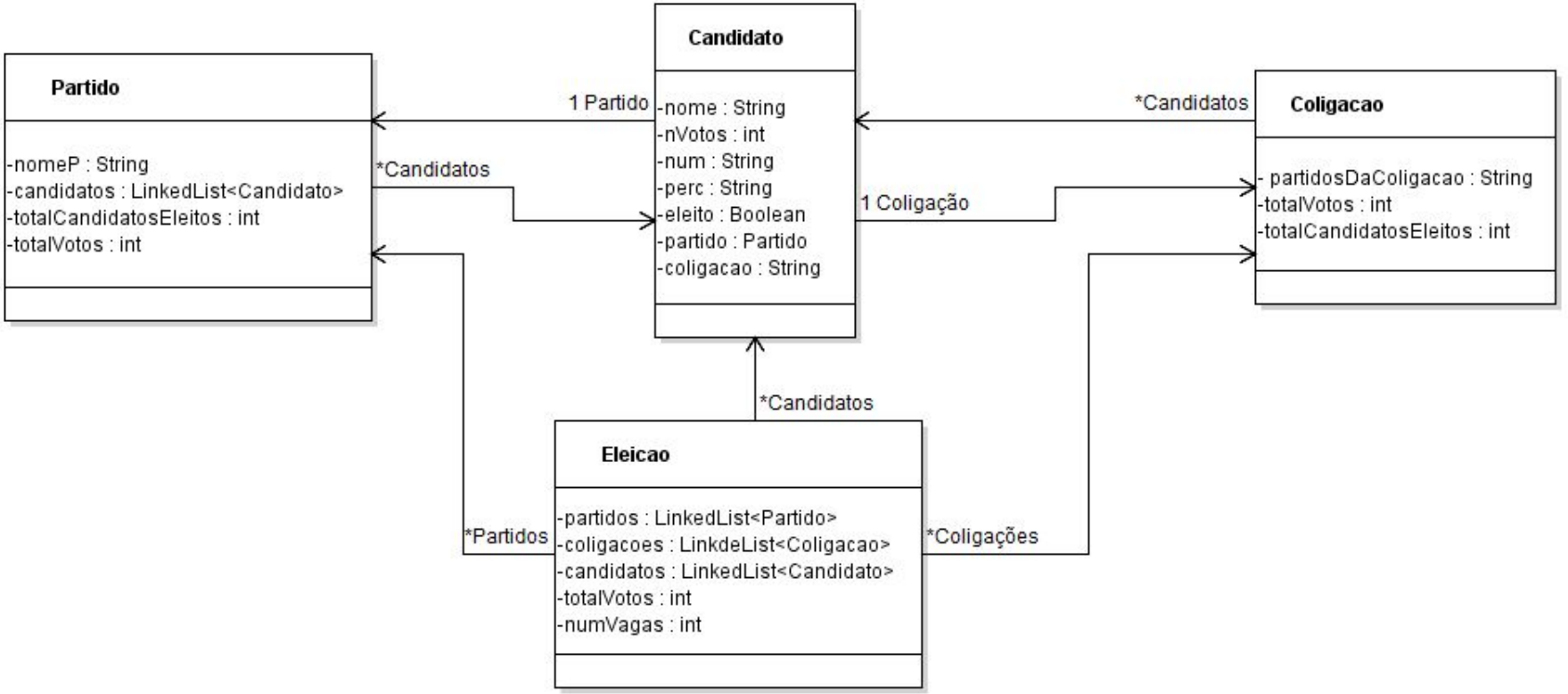
Avaliação de Linguagens

Critérios Gerais	C	C++	Java	C#
Expressões e comandos	Sim	Sim	Sim	Sim
Tipos primitivos e compostos	Sim	Sim	Sim	Sim
Gerenciamento de memória	Programador	Programador	Sistema	Sistema
Persistência dos dados	Biblioteca de Funções	Biblioteca de classes e funções	JDBC, biblioteca de classes, serialização	biblioteca de classes, serialização, Entity Framework, Nhibernate etc
Passagem de parâmetros	Lista variável e por valor	Lista variável, default, por valor e por referência	Lista variável, por valor e por cópia de referência	Lista variável, por valor e por cópia de referência ou por referência
Encapsulamento e proteção	Parcial	Sim	Sim	Sim

Avaliação de Linguagens

Critérios Gerais	C	C++	Java	C#
Sistema de tipos	Não	Parcial	Sim	Sim
Verificação de tipos	Estática	Estática/Dinâmica	Estática/Dinâmica	Estática/Dinâmica
Polimorfismo	Coerção e Sobrecarga	Todos	Todos	Todos
Exceções	Não	Parcial	Sim	Parcial
Concorrência	Não (biblioteca de funções)	Não (biblioteca de funções)	Sim	Sim

Trabalho


Candidato
-nome : String
-nVotos : int
-num : String
-perc : String
-eleito : Boolean
-partido : Partido
-coligacao : String
Partido
-nomeP : String
-candidatos : LinkedList<Candidato>
-totalCandidatosEleitos : int
-totalVotos : int
Coligacao
- partidosDaColigacao : String
-totalVotos : int
-totalCandidatosEleitos : int
Eleicao
-partidos : LinkedList<Partido>
-coligacoes : LinkdeList<Coligacao>
-candidatos : LinkedList<Candidato>
-totalVotos : int
-numVagas : int
1 Partido
*Candidatos
*Candidatos
1 Coligação
*Candidatos
*Partidos
*Coligações

Referências

https://www.caelum.com.br/apostila-csharp-orientacao-objetos/

https://docs.microsoft.com/pt-br/dotnet/csharp/

https://www.devmedia.com.br/tratamento-de-excecoes-em-c/26106

https://pt.stackoverflow.com/questions/34907/qual-a-diferen%C3%A7a-entre-uma-express%C3%A3o-lambda-um-closure-e-um-delegate

Referências

http://www.linhadecodigo.com.br/artigo/2770/escopo-e-nivel-de-acessos-no-csharp.aspx

https://docs.microsoft.com/en-us/dotnet/standard/design-guidelines/names-of-classes-structs-and-interfaces

https://docs.microsoft.com/pt-br/dotnet/csharp/language-reference/keywords/

https://docs.microsoft.com/pt-br/dotnet/csharp/programming-guide/concepts/threading/thread-synchronization