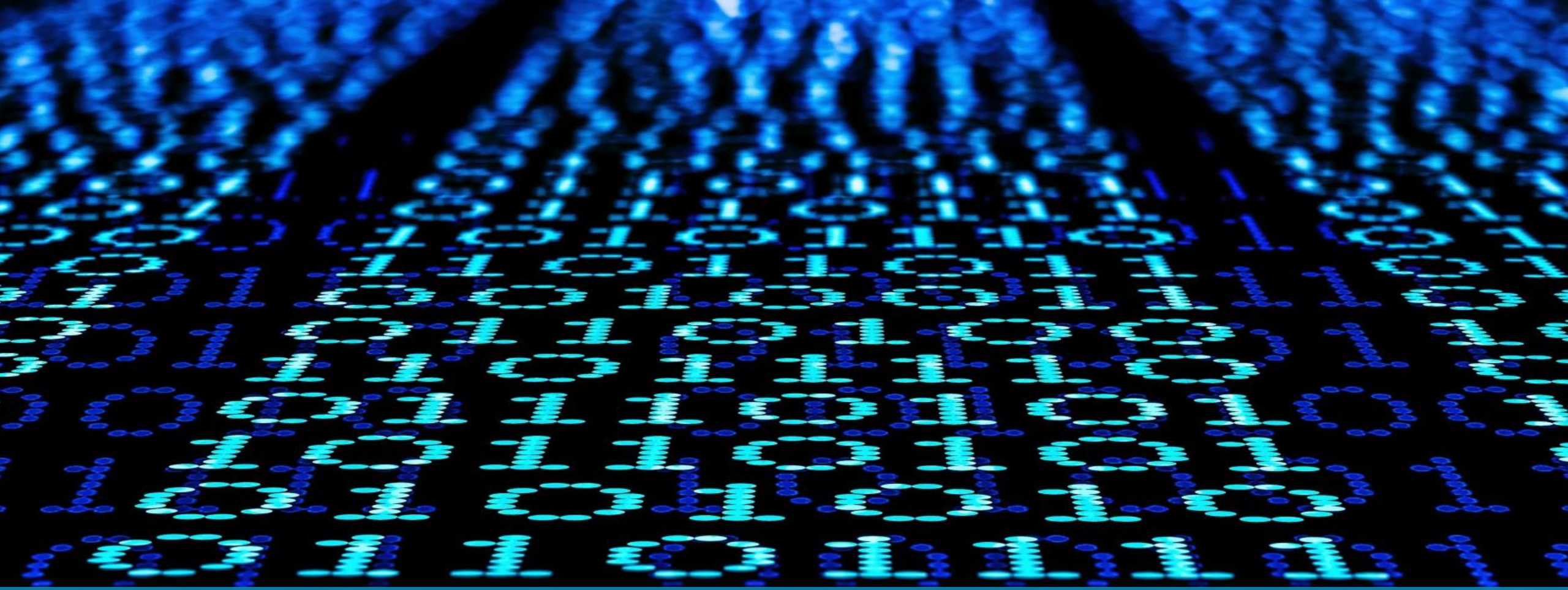
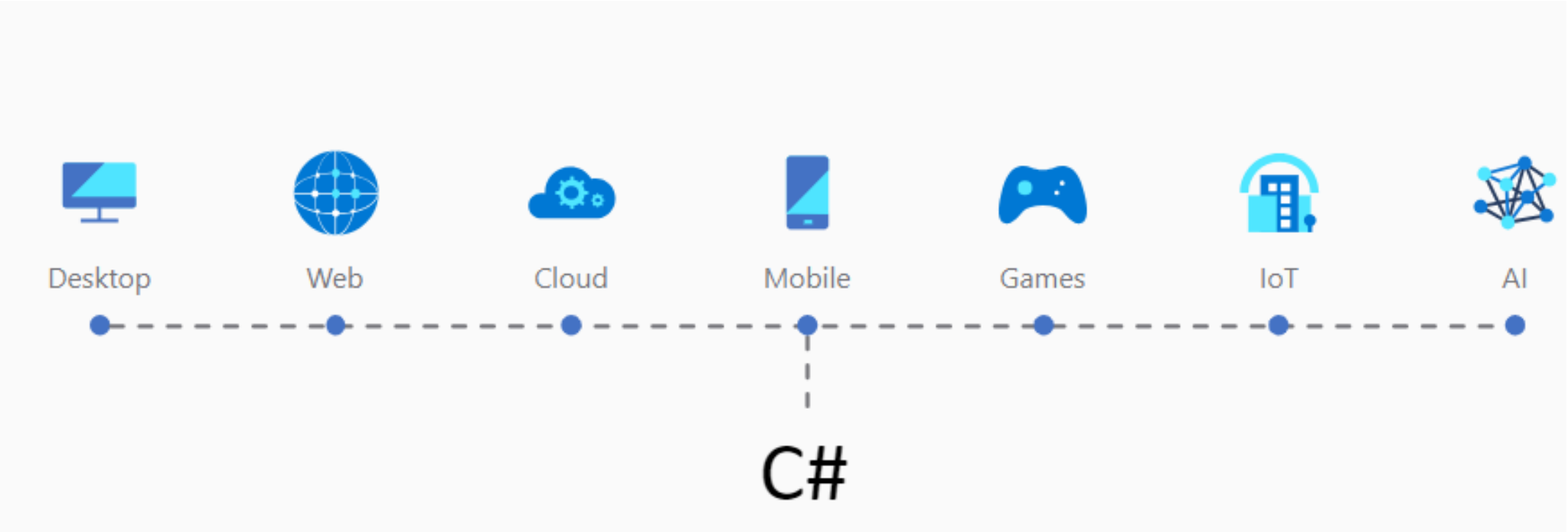


ED'S BUILT-IN C#

Alunos:
Katyane dos santos
Vênisson Cardoso dos santos

INTRODUÇÃO ED'S BUILD-IN EM C#

- C# é uma linguagem de programação orientada a objetos da Microsoft
- Sintaxe semelhante a C, C++, Java e Object Pascal
- Projetado para desenvolvedores trabalharem de forma natural com componentes de software
- Ênfase no desenvolvimento de aplicativos robustos e duráveis
- Controle de versões para garantir estabilidade ao longo do tempo.
- Possui aplicações nas diferentes plataformas



PLATAFORMA COMPATÍVEIS COM C#

FUNCIONAMENTO DO C#

C# é executado no ambiente ".NET" (Network Enabled Technologies).

O CLR (Common Language Runtime) é o sistema de execução virtual que suporta o C#.

O CLR é uma base para ambientes de execução e desenvolvimento integrados.

O código C# é compilado em IL (linguagem intermediária) compatível com a CLI (Common Language Infrastructure).

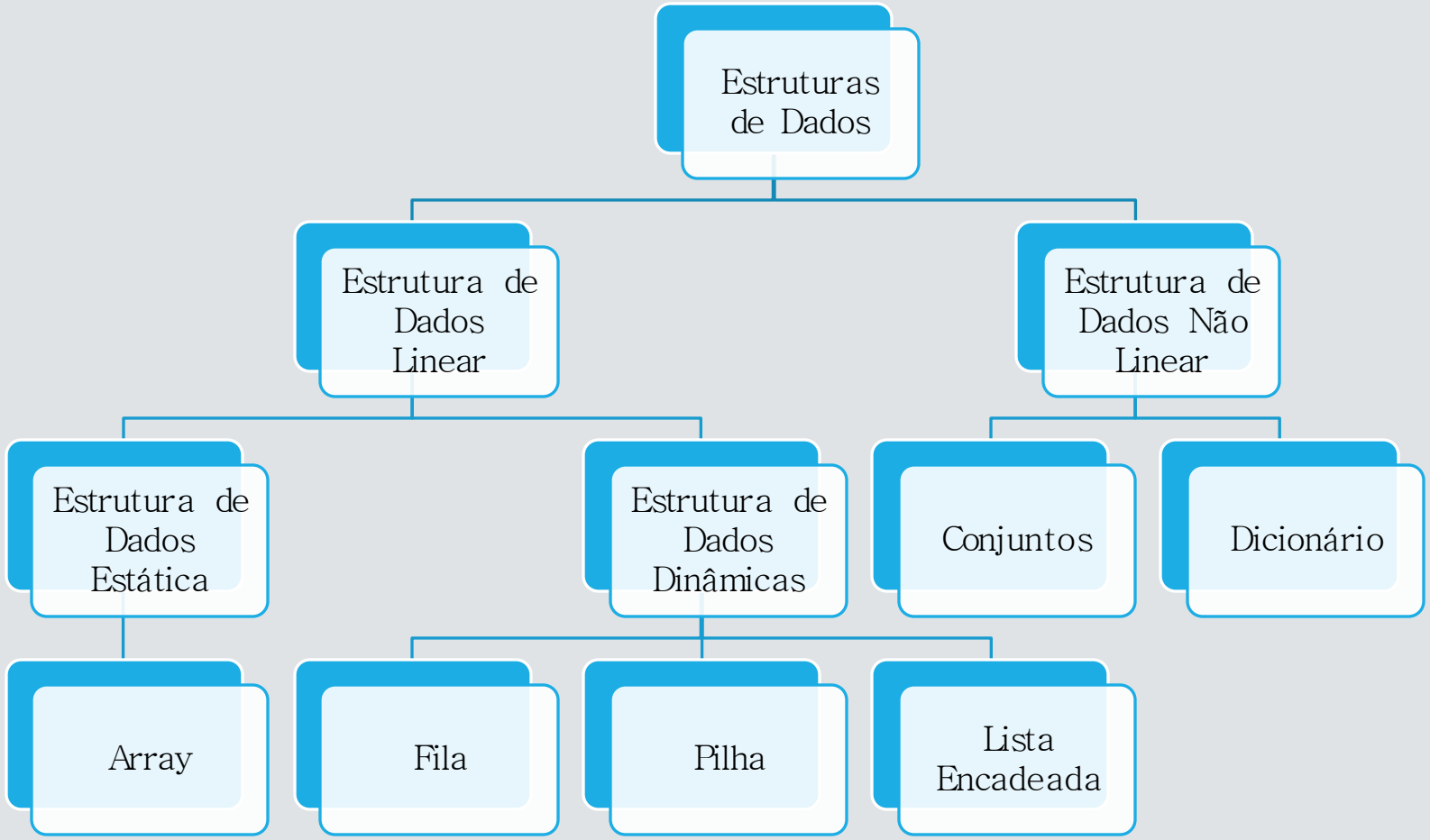
IL e recursos são armazenados em assemblies (geralmente com extensão .dll).

Um assembly contém informações sobre tipos, versões e cultura no manifesto.

Durante a execução, o assembly é carregado no CLR e traduzido em instruções nativas da máquina.

O CLR também fornece serviços como coleta de lixo, tratamento de exceções e gerenciamento

CLASSIFICAÇÃO DAS ESTRUTURA DE DADOS



Vantagens Significativas

Estruturas de Dados Integradas

Facilidade de Uso

Segurança de Tipos

Gerenciamento Automático de Memória

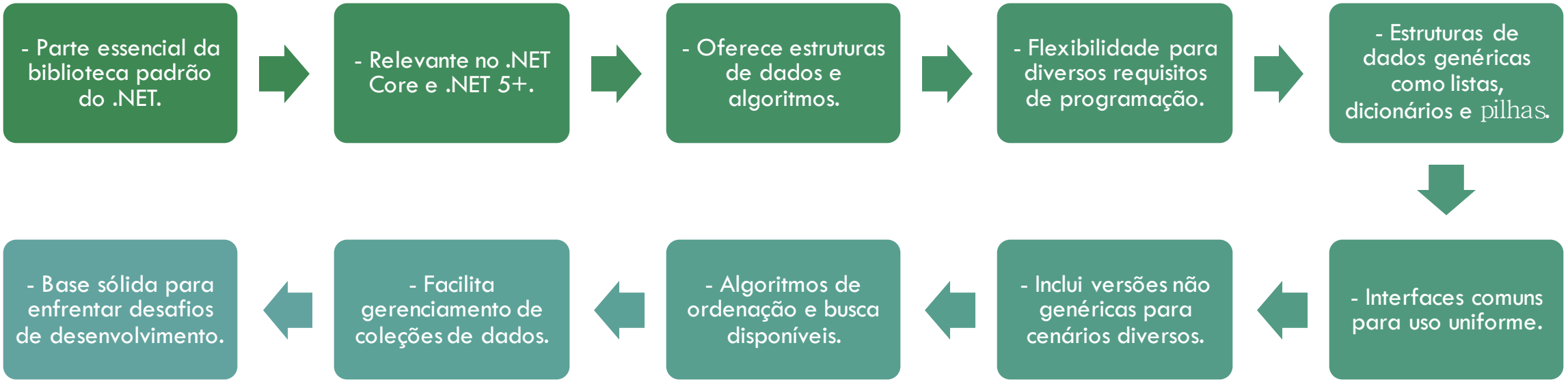
Integração com a Plataforma .NET

Comunidade Ativa e Suporte da Microsoft

Desenvolvimento Rápido e Segurança

Integração Natural com Tecnologias Microsoft

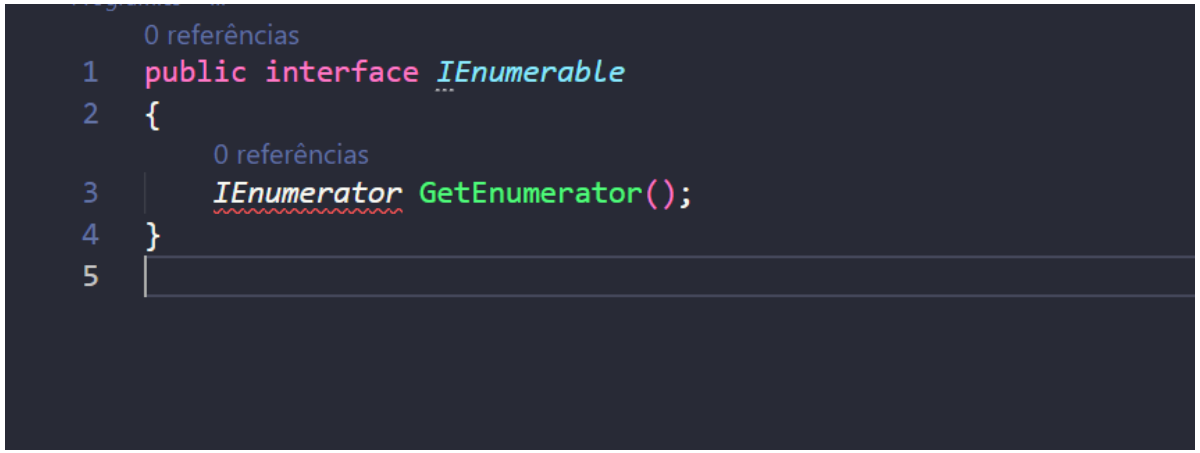
MOTIVAÇÃO PARA USAR ESTRUTURAS DE DADOS EM C#



SYSTEM.COLLECTIONS EM C#

INTERFACE IENUMERABLE EM C#

- Fundamental no namespace System.Collections.
- Utilizada para representar coleções iteráveis em C#.
- Oferece iteração sequencial em várias estruturas de dados.
- Assinatura da Interface IEnumerable:



USO DA INTERFACE IENUMERABLE

Método GetEnumerator() retorna um objeto IEnumerator.
IEnumerator permite acesso durante a iteração.
Loops foreach simplificam iteração sem índices.
Exemplo de código demonstrando uso da IEnumerable.

Program.cs

```
using System;
using System.Collections;

0 referências
class Program
{
    0 referências
    static void Main()
    {
        int[] numeros = { 1, 2, 3, 4, 5 };
        IEnumerable enumerable = numeros;
        IEnumerator enumerator = enumerable.GetEnumerator();

        Console.WriteLine("Iterando pelos números:");

        while (enumerator.MoveNext())
        {
            int numero = (int)enumerator.Current;
            Console.WriteLine(numero);
        }
    }
}
```

INTERFACE ICOLLECTION EM C#

Utilizada para representar coleções de elementos acessíveis e manipuláveis.

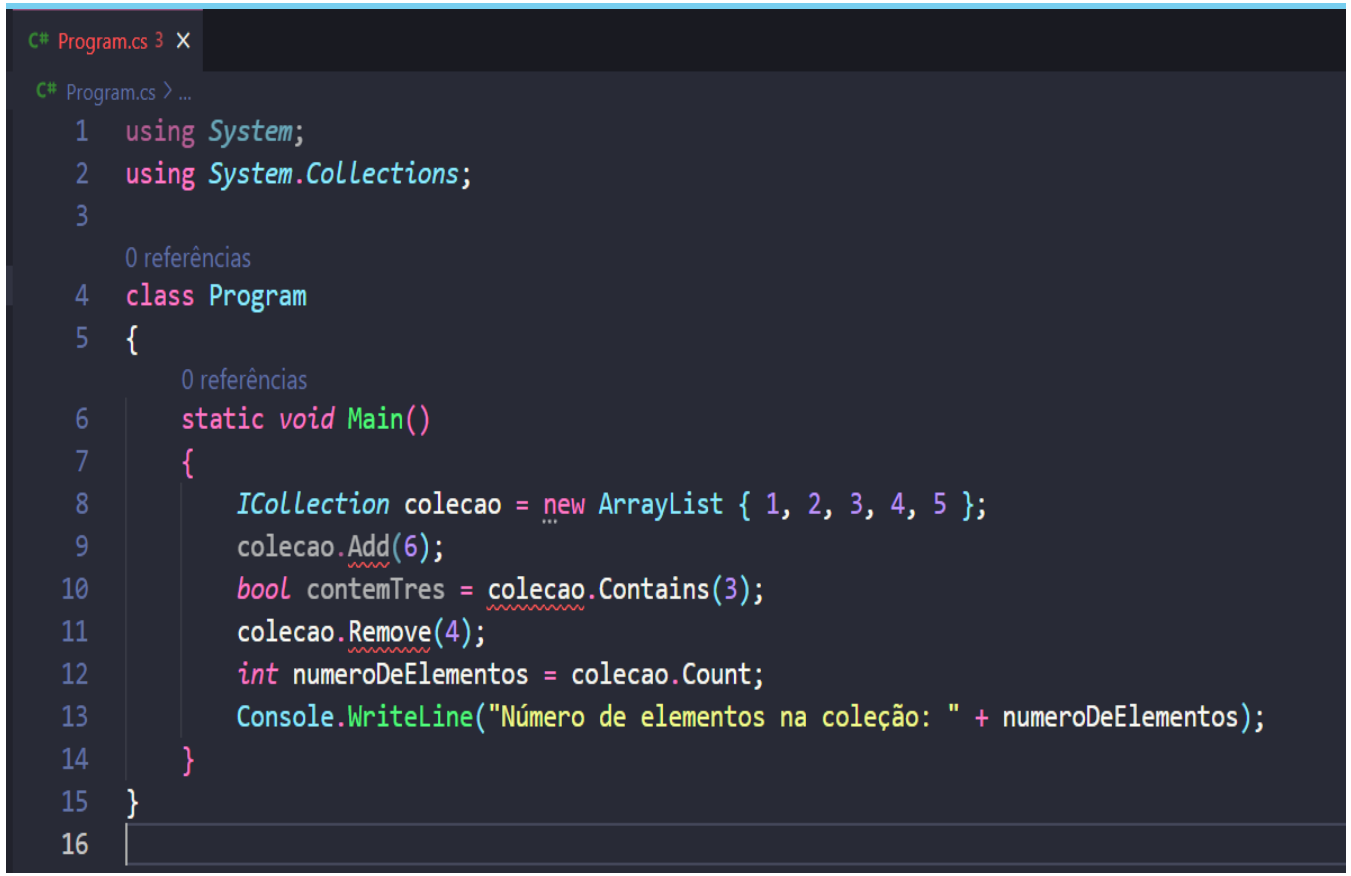
Métodos e propriedades para interação flexível.

Propriedade Count para contar elementos.

Propriedade IsSynchronized para sincronização multithreading.

Propriedade SyncRoot para acesso seguro.

Métodos para copiar, adicionar, limpar, verificar e remover elementos.



INTERFACE ILIST EM C#

```
C# Program.cs > ...
1   using System;
2   using System.Collections;
3
    0 referências
4   class Program
5   {
        0 referências
6       static void Main()
7       {
8           IList lista = new ArrayList { 1, 2, 3 };
9           Console.WriteLine("Primeiro elemento: " + lista[0]);
10          lista.Add(4);
11          lista.Insert(1, 5);
12          lista.Remove(3);
13          bool contemCinco = lista.Contains(5);
14          int numeroDeElementos = lista.Count;
15          Console.WriteLine("Número de elementos na lista: " + numeroDeElementos);
16      }
17  }
```

Representa uma coleção indexada de elementos.

Herda de ICollection e IEnumerable.

Indexação com acesso a elementos.

Métodos para adicionar, remover, verificar e pesquisar elementos.

Método para limpar a lista.

Facilita o acesso e manipulação de elementos em listas.

Oferece funcionalidades de adição, remoção e pesquisa.

CARATERÍSTICAS DA IMPLEMENTAÇÃO EM C# ARRAY

- Indexação eficiente
- Tamanho fixo
- Implementação genérica
- Incorpora algoritmos de ordenação e pesquisa
- Manipulação de memória
- Sobrecarga de métodos

C# Program.cs > ...

```
using System;
0 referências
public class SamplesArray
0 referências
{    public static void Main()
    {
        // Creates and initializes a new integer array and a new Object array.
        int[] myIntArray = new int[5] { 1, 2, 3, 4, 5 };
        Object[] myObjArray = new Object[5] { 26, 27, 28, 29, 30 };
        // Prints the initial values of both arrays.
        Console.WriteLine("Initially,");
        Console.Write("integer array:");
        PrintValues(myIntArray);
        Console.Write("Object array: ");
        PrintValues(myObjArray);
        // Copies the first two elements from the integer array to the Object array.
        System.Array.Copy(myIntArray, myObjArray, 2);
        // Prints the values of the modified arrays.
        Console.WriteLine("\nAfter copying the first two elements of the integer array to the Object array,");
        Console.Write("integer array:");
        PrintValues(myIntArray);
        Console.Write("Object array: ");
        PrintValues(myObjArray);
        // Copies the last two elements from the Object array to the integer array.
        System.Array.Copy(myObjArray, myObjArray.GetUpperBound(0) - 1, myIntArray, myIntArray.GetUpperBound(0) - 1, 2);
        // Prints the values of the modified arrays.
        Console.WriteLine("\nAfter copying the last two elements of the Object array to the integer array,");
        Console.Write("integer array:");
        PrintValues(myIntArray);
        Console.Write("Object array: ");
        PrintValues(myObjArray);
    }
```

Parte de código da implementação da classe Array

CARATERÍSTICAS DA IMPLEMENTAÇÃO EM C#
LISTA ENCADEADA

- Implementada por meio das classes LinkedList<T> e LInkedListNode<T>
- LinkedList<T> - Gerencia a lista encadeada, Mantém a cabeça e a cauda
- LinkedListNode – Representa um nó; Contém valor e referências.
- Possui Gerenciamento de Memória
- Tratamento de Erros

Arquivo Editar Seleção Ver Acessar Executar ···

ED

Program.cs 9+

Program.cs > Example > Main

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

0 referências
public class Example
{
    0 referências
    public static void Main()
    {
        // Create the link list.
        string[] words =
            { "the", "fox", "jumps", "over", "the", "dog" };
        LinkedList<string> sentence = new LinkedList<string>(words);
        Display(sentence, "The linked list values:");
        Console.WriteLine("sentence.Contains(\"jumps\") = {0}",
            sentence.Contains("jumps"));

        // Add the word 'today' to the beginning of the linked list.
        sentence.AddFirst("today");
        Display(sentence, "Test 1: Add 'today' to beginning of the list:");

        // Move the first node to be the last node.
        LinkedListNode<string> mark1 = sentence.First;
        sentence.RemoveFirst();
        sentence.AddLast(mark1);
        Display(sentence, "Test 2: Move first node to be last node:");

        // Change the last node to 'yesterday'.
        sentence.RemoveLast();
        sentence.AddLast("yesterday");
        Display(sentence, "Test 3: Change the last node to 'yesterday':");

        // Move the last node to be the first node.
        mark1 = sentence.Last;
        sentence.RemoveLast();
        sentence.AddFirst(mark1);
```

Parte do Código da implementação da classe Listas encadeadas

⊗ 0 ⚠ 16 0 Projetos: 1 Ln 9, Col 33 Espaços: 4 UTF-8 with BOM CRLF C#

Arquivo Editar Seleção Ver Acessar Executar ···

ED

C# Program.cs 9+

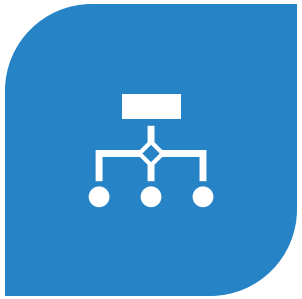
C# Program.cs > Example > Main

```
9         // Create the link list.
10        string[] words =
11            { "the", "fox", "jumps", "over", "the", "dog" };
12        LinkedList<string> sentence = new LinkedList<string>(words);
13        Display(sentence, "The linked list values:");
14        Console.WriteLine("sentence.Contains(\"jumps\") = {0}",
15            sentence.Contains("jumps"));
16
17        // Add the word 'today' to the beginning of the linked list.
18        sentence.AddFirst("today");
19        Display(sentence, "Test 1: Add 'today' to beginning of the list:");
20
21        // Move the first node to be the last node.
22        LinkedListNode<string> mark1 = sentence.First;
23        sentence.RemoveFirst();
24        sentence.AddLast(mark1);
25        Display(sentence, "Test 2: Move first node to be last node:");
26
27        // Change the last node to 'yesterday'.
28        sentence.RemoveLast();
29        sentence.AddLast("yesterday");
30        Display(sentence, "Test 3: Change the last node to 'yesterday':");
31
32        // Move the last node to be the first node.
33        mark1 = sentence.Last;
34        sentence.RemoveLast();
35        sentence.AddFirst(mark1);
36        Display(sentence, "Test 4: Move last node to be first node:");
37
38        // Indicate the last occurence of 'the'.
39        sentence.RemoveFirst();
40        LinkedListNode<string> current = sentence.FindLast("the");
41        IndicateNode(current, "Test 5: Indicate last occurence of 'the':");
42
43        // Add 'lazy' and 'old' after 'the' (the LinkedListNode named current).
44        sentence.AddAfter(current, "old");
45        sentence.AddAfter(current, "lazy");
```

Parte do Código da implementação da classe Listas encadeadas

⊗ 0 ⚠ 16 0 Projetos: 1 Ln 9, Col 33 Espaços: 4 UTF-8 with BOM CRLF C#

CARATERÍSTICAS DA IMPLEMENTAÇÃO EM C#
FILA



POSSUI COMO ESTRUTURA INTERNA UMA LISTA DUPLAMENTE ENCADEADAS;

OPERAÇÕES PRINCIPAIS: EMPILHAR (ENQUEUE), DESEMPILHAR (DEQUEUE), PRÓXIMO (PEEK);

SINCRONIZAÇÃO: MECANISMO PARA ACESSO MULTITHREAD SEGURO;

TRATAMENTO DE EXCEÇÕES

Program.cs

Program.cs > Example

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

0 referências
class Example
{
    0 referências
    public static void Main()
    {
        Queue<string> numbers = new Queue<string>();
        numbers.Enqueue("one");
        numbers.Enqueue("two");
        numbers.Enqueue("three");
        numbers.Enqueue("four");
        numbers.Enqueue("five");

        // A queue can be enumerated without disturbing its contents.
        foreach( string number in numbers )
        {
            Console.WriteLine(number);
        }

        Console.WriteLine("\nDequeuing '{0}'", numbers.Dequeue());
        Console.WriteLine("Peek at next item to dequeue: {0}",
            numbers.Peek());
        Console.WriteLine("Dequeuing '{0}'", numbers.Dequeue());

        // Create a copy of the queue, using the ToArray method and the
        // constructor that accepts an IEnumerable<T>.
        Queue<string> queueCopy = new Queue<string>(numbers.ToArray());

        Console.WriteLine("\nContents of the first copy:");
        foreach( string number in queueCopy )
        {
            Console.WriteLine(number);
        }

```

Program.cs

Program.cs > Example

```
36          // Create an array twice the size of the queue and copy the
37          // elements of the queue, starting at the middle of the
38          // array.
39          string[] array2 = new string[numbers.Count * 2];
40          numbers.CopyTo(array2, numbers.Count);
41
42          // Create a second queue, using the constructor that accepts an
43          // IEnumerable(Of T).
44          Queue<string> queueCopy2 = new Queue<string>(array2);
45
46          Console.WriteLine("\nContents of the second copy, with duplicates and nulls:");
47          foreach( string number in queueCopy2 )
48          {
49              Console.WriteLine(number);
50          }
51
52          Console.WriteLine("\nqueueCopy.Contains(\"four\") = {0}",
53              queueCopy.Contains("four"));
54
55          Console.WriteLine("\nqueueCopy.Clear()");
56          queueCopy.Clear();
57          Console.WriteLine("\nqueueCopy.Count = {0}", queueCopy.Count);
58      }
59  }
```

CARATERÍSTICAS DA IMPLEMENTAÇÃO EM C# PILHA

- Implementado utilizando um array unidimensional
- Métodos principais – push; pop, peek;
- Possui Redimensionamento Dinâmico
- Tratamento de Exceções

Program.cs

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

0 referências
class Example
{
    0 referências
    public static void Main()
    {
        Stack<string> numbers = new Stack<string>();
        numbers.Push("one");
        numbers.Push("two");
        numbers.Push("three");
        numbers.Push("four");
        numbers.Push("five");

        // A stack can be enumerated without disturbing its contents.
        foreach( string number in numbers )
        {
            Console.WriteLine(number);
        }

        Console.WriteLine("\nPopping '{0}'", numbers.Pop());
        Console.WriteLine("Peek at next item to destack: {0}",
            numbers.Peek());
        Console.WriteLine("Popping '{0}'", numbers.Pop());

        // Create a copy of the stack, using the ToArray method and the
        // constructor that accepts an IEnumerable<T>.
        Stack<string> stack2 = new Stack<string>(numbers.ToArray());

        Console.WriteLine("\nContents of the first copy:");
        foreach( string number in stack2 )
        {
            Console.WriteLine(number);
        }
```

```
36          // Create an array twice the size of the stack and copy the
37          // elements of the stack, starting at the middle of the
38          // array.
39          string[] array2 = new string[numbers.Count * 2];
40          numbers.CopyTo(array2, numbers.Count);
41
42          // Create a second stack, using the constructor that accepts an
43          // IEnumerable(Of T).
44          Stack<string> stack3 = new Stack<string>(array2);
45
46          Console.WriteLine("\nContents of the second copy, with duplicates and nulls:");
47          foreach( string number in stack3 )
48          {
49              Console.WriteLine(number);
50          }
51
52          Console.WriteLine("\nstack2.Contains(\"four\") = {0}",
53              stack2.Contains("four"));
54
55          Console.WriteLine("\nstack2.Clear()");
56          stack2.Clear();
57          Console.WriteLine("\nstack2.Count = {0}", stack2.Count);
58      }
59  }
60
61
```

CARATERÍSTICAS DA IMPLEMENTAÇÃO EM C# DICIONÁRIO

Implementada com base em uma tabela de dispersão

Chave valor

Possuir hashing

Tratamento de colisões

Diversos métodos Adição (Add), remover (Remove), Limpar (Clear), Contem (ContainsKey)

C# Program.cs 1

C# Program.cs

```
// Create a new dictionary of strings, with string keys.
//
Dictionary<string, string> openWith =
    new Dictionary<string, string>();

// Add some elements to the dictionary. There are no
// duplicate keys, but some of the values are duplicates.
openWith.Add("txt", "notepad.exe");
openWith.Add("bmp", "paint.exe");
openWith.Add("dib", "paint.exe");
openWith.Add("rtf", "wordpad.exe");

// The Add method throws an exception if the new key is
// already in the dictionary.
try
{
    openWith.Add("txt", "winword.exe");
}
catch (ArgumentException)
{
    Console.WriteLine("An element with Key = \"txt\" already exists.");
}

// The Item property is another name for the indexer, so you
// can omit its name when accessing elements.
Console.WriteLine("For key = \"rtf\", value = {0}.",
    openWith["rtf"]);

// The indexer can be used to change the value associated
// with a key.
openWith["rtf"] = "winword.exe";
Console.WriteLine("For key = \"rtf\", value = {0}.",
    openWith["rtf"]);

// If a key does not exist, setting the indexer for that key
// adds a new key/value pair.
openWith["doc"] = "winword.exe";
```

```
38
39  // The indexer throws an exception if the requested key is
40  // not in the dictionary.
41  try
42  {
43      Console.WriteLine("For key = \"tif\", value = {0}.",
44          openWith["tif"]);
45  }
46  catch (KeyNotFoundException)
47  {
48      Console.WriteLine("Key = \"tif\" is not found.");
49  }
50
51  // When a program often has to try keys that turn out not to
52  // be in the dictionary, TryGetValue can be a more efficient
53  // way to retrieve values.
54  string value = "";
55  if (openWith.TryGetValue("tif", out value))
56  {
57      Console.WriteLine("For key = \"tif\", value = {0}.", value);
58  }
59  else
60  {
61      Console.WriteLine("Key = \"tif\" is not found.");
62  }
63
64  // ContainsKey can be used to test keys before inserting
65  // them.
66  if (!openWith.ContainsKey("ht"))
67  {
68      openWith.Add("ht", "hypertrm.exe");
69      Console.WriteLine("Value added for key = \"ht\": {0}",
70          openWith["ht"]);
71  }
72
73  // When you use foreach to enumerate dictionary elements,
```

CARATERÍSTICAS DA IMPLEMENTAÇÃO EM C# HASHSET

- Hashset oferece uma implementação eficiente para armazenar elementos únicos;
- Utiliza uma tabela de hash interna;
- Possui um a algoritmo de hashing e o tratamento de colisões;
- Possui métodos de Adicão (Add), Remoção (Remove), União (UnionWith), Diferença (IntersectWith);

Program.cs

```cs
HashSet<int> evenNumbers = new HashSet<int>();
HashSet<int> oddNumbers = new HashSet<int>();

for (int i = 0; i < 5; i++)
{
    // Populate numbers with just even numbers.
    evenNumbers.Add(i * 2);

    // Populate oddNumbers with just odd numbers.
    oddNumbers.Add((i * 2) + 1);
}

Console.Write("evenNumbers contains {0} elements: ", evenNumbers.Count);
DisplaySet(evenNumbers);

Console.Write("oddNumbers contains {0} elements: ", oddNumbers.Count);
DisplaySet(oddNumbers);

// Create a new HashSet populated with even numbers.
HashSet<int> numbers = new HashSet<int>(evenNumbers);
Console.WriteLine("numbers UnionWith oddNumbers...");
numbers.UnionWith(oddNumbers);

Console.Write("numbers contains {0} elements: ", numbers.Count);
DisplaySet(numbers);

void DisplaySet(HashSet<int> collection)
{
    Console.Write("{");
    foreach (int i in collection)
    {
        Console.Write(" {0}", i);
    }
    Console.WriteLine(" }");
}
```

TODOS OS CÓDIGOS COMPLETOS DESSAS ESTRUTURAS DE DADOS A SEGUIR ENCONTRA-SE NO REPOSITÓRIO DA ".NET" NO GITHUB.

PERGUNTAS?

REFERÊNCIAS BIBLIOGRÁFICAS

MICROSOFT. System.Collections.Stack Class - .NET 7.0. Disponível em: https://learn.microsoft.com/en-us/dotnet/api/system.collections.stack?view=net-7.0. Acesso em: 1 out. 2023.

MICROSOFT. System.Collections.Queue Class - .NET 7.0. Disponível em: https://learn.microsoft.com/en-us/dotnet/api/system.collections.queue?view=net-7.0. Acesso em: 1 out. 2023.

MICROSOFT. System.Collections.ArrayList Class - .NET 7.0. Disponível em: https://learn.microsoft.com/en-us/dotnet/api/system.collections.arraylist?view=net-7.0. Acesso em: 1 out. 2023.

MICROSOFT. C# - Documentação do C# | Microsoft Docs. Disponível em: https://learn.microsoft.com/pt-br/dotnet/csharp/. Acesso em: 1 out. 2023.

MICROSOFT. System.Collections.Generic.IEnumerable<T> Interface - .NET 7.0. Disponível em: https://learn.microsoft.com/en-us/dotnet/api/system.collections.generic.ienumerable-1?view=net-7.0. Acesso em: 1 out. 2023.

AGRADECIMENTOS