**BỘ GIÁO DỤC VÀ ĐÀO TẠO**

**TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT TP. HỒ CHÍ MINH**

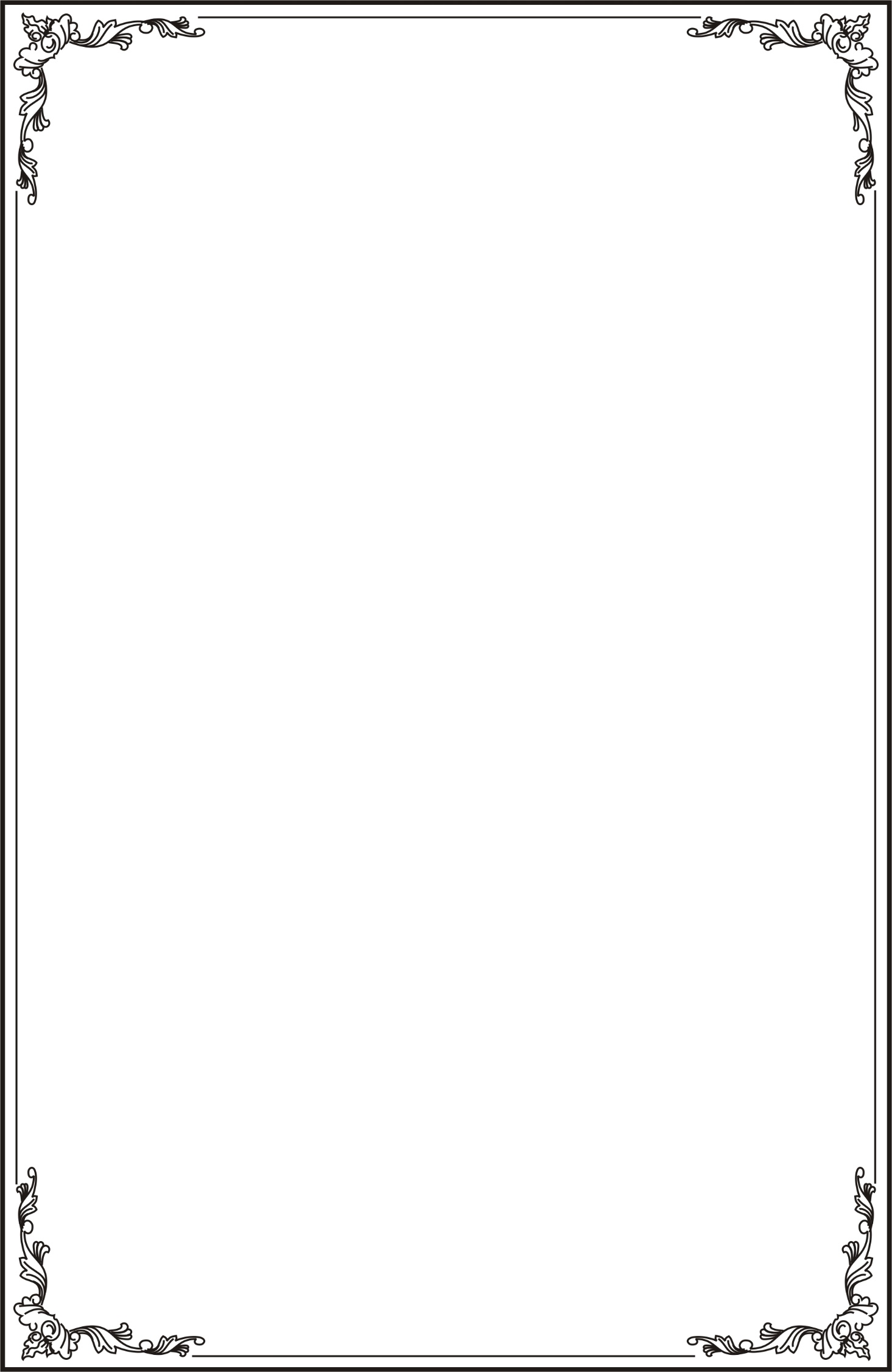
# **KHOA CÔNG NGHỆ THÔNG TIN**

-----🙞🙜🕮🙞🙜-----

Ảnh có chứa ký hiệu, ngoài trời

Mô tả được tạo tự động

# **BÁO CÁO TUẦN 10**

**MÔN HỌC: Lập trình hướng đối tượng(OOP)**

**BÁO CÁO: Tìm hiểu và trình bày công dụng các phương thức chính**

**GIÁO VIÊN HƯỚNG DẪN: *TS. NGUYỄN TRẦN THI VĂN***

**SINH VIÊN THỰC HIỆN:**

**TRẦN THỊ LỆ XUÂN 18133066**

**TRƯƠNG HÙNG ANH 18133001**

**NGUYỄN MINH SANG 18133049**

**NGUYỄN ANH TRIỀU 18133058**

**TP. HỒ CHÍ MINH \_ 2019**

**BỘ GIÁO DỤC VÀ ĐÀO TẠO**

**TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT TP. HỒ CHÍ MINH**

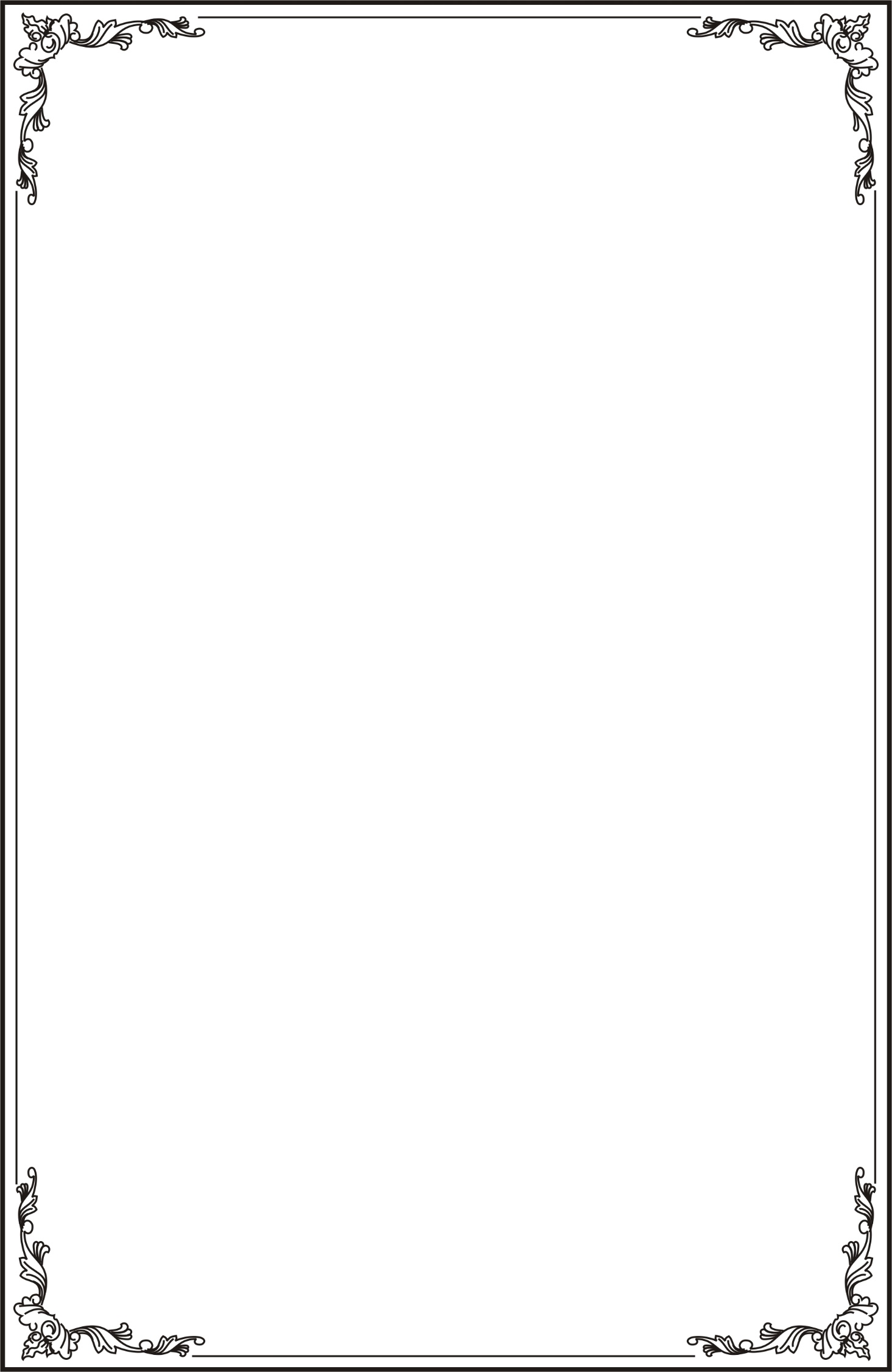
# **KHOA CÔNG NGHỆ THÔNG TIN**

-----🙞🙜🕮🙞🙜-----

Ảnh có chứa ký hiệu, ngoài trời

Mô tả được tạo tự động

# **BÁO CÁO TUẦN 10**

**MÔN HỌC: Lập trình hướng đối tượng(OOP)**

**BÁO CÁO: Tìm hiểu và trình bày công dụng các phương thức chính**

**GIÁO VIÊN HƯỚNG DẪN: *TS. NGUYỄN TRẦN THI VĂN***

**SINH VIÊN THỰC HIỆN:**

**TRẦN THỊ LỆ XUÂN 18133066**

**TRƯƠNG HÙNG ANH 18133001**

**NGUYỄN MINH SANG 18133049**

**NGUYỄN ANH TRIỀU 18133058**

**TP. HỒ CHÍ MINH \_ 2019**

**ĐIỂM SỐ**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **TIÊU CHÍ** | **NỘI DUNG** | **BỐ CỤC** | **TRÌNH BÀY** | **TỔNG** |
| **ĐIỂM** |  |  |  |  |

**NHẬN XÉT CỦA GVHD**

*Ký tên*

***TS. Nguyễn Trần Thi Văn***

**LỜI CẢM ƠN**

Đặc biệt nhóm chúng em cũng xin cảm ơn một cách chân thành nhất đối với Thầy TS. Nguyễn Trần Thi Văn, người đã trực tiếp hướng dẫn nhóm hoàn thành khóa luận báo cáo về tìm hiểu và trình bày công dụng các phương thức chính. Trong quá trình thực hiện Thầy đã giúp đỡ và hướng dẫn chúng em những lúc gặp khó khan, định hướng đi đúng đắn nhất, tạo ra động lực để nghiên cứu những kiến thức mới.

Bài báo cáo về tìm hiểu và trình bày công dụng các phương thức chính có thể có nhiều khuyết điểm và hạn chế. Kính mong thầy góp ý chân thành để bài báo cáo chúng em hoàn thiện hơn. Cho chúng em những định hướng trong tương lai, có thêm kinh nghiệm những bài báo cáo sau này Cuối cùng chúng em chúc Thầy sức khỏe thật dồi dào nhiệt huyết trong giảng dạy để góp phần đào tạo cho những lớp sinh viên đi sau trở thành những nhân tài tương lai của đất nước sau này.

1. **SYSTEM.ARRAY**
   1. ***Khái niệm***

* Mảng là một trong những cấu trúc dữ liệu cơ bản trong các ngôn ngữ lập trình, cung cấp một vùng lưu trữ cho các giá trị dữ liệu tuần tự
* System.Array là lớp cơ sở cung cấp các phương thức để tạo, thao tác, tìm kiếm và sắp xếp các mảng, do đó đóng vai trò là lớp cơ sở cho tất cả các mảng trong thời gian chạy ngôn ngữ chung
* System.Array thực hiện một số giao diện, như IClonizable, IList, ICollection và IEnumerable. Nó cũng có nhiều thuộc tính và phương thức để giúp bạn xác định thông tin về mảng của bạn.
  1. ***Ví dụ về một số các phương thức***
* *AsReadOnly <T> (T []):* Trả về một trình bao bọc chỉ đọc cho mảng đã chỉ định.
* *BinarySearch (Array, Int32, Int32, Object):* Tìm kiếm một loạt các phần tử trong mảng được sắp xếp một chiều cho một giá trị, sử dụng giao diện IComparable được triển khai bởi từng phần tử của mảng và theo giá trị đã chỉ định
* *BinarySearch (Array, Int32, Int32, Object, IComparer):* Tìm kiếm một loạt các phần tử trong mảng được sắp xếp một chiều cho một giá trị, sử dụng giao diện IComparer được chỉ định
* *BinarySearch(Array, Object):* Tìm kiếm toàn bộ mảng được sắp xếp một chiều cho một phần tử cụ thể, sử dụng giao diện IComparable được triển khai bởi từng phần tử của mảng và theo đối tượng đã chỉ định
* *BinarySearch (Array, Object, IComparer):* Tìm kiếm toàn bộ mảng được sắp xếp một chiều cho một giá trị bằng giao diện IComparer đã chỉ định
* *Copy (Array, Array,Int32):* Sao chép một loạt các phần tử từ một Mảng bắt đầu từ phần tử đầu tiên và dán chúng vào một Mảng khác bắt đầu từ phần tử đầu tiên. Độ dài được chỉ định là số nguyên 32 bit
* *Copy (Array, Array,Int64):* Sao chép một loạt các phần tử từ một Mảng bắt đầu từ phần tử đầu tiên và dán chúng vào một Mảng khác bắt đầu từ phần tử đầu tiên. Độ dài được chỉ định là số nguyên 64 bit
* *Copy (Array,Int32, Array,Int32, Int32):* Sao chép một loạt các phần tử từ một Mảng bắt đầu từ chỉ mục nguồn được chỉ định và dán chúng vào một Mảng khác bắt đầu từ chỉ mục đích đã chỉ định. Độ dài và chỉ mục được chỉ định là số nguyên 32 bit
* *Copy (Array,Int64, Array,Int64, Int64):* Sao chép một loạt các phần tử từ một Mảng bắt đầu từ chỉ mục nguồn được chỉ định và dán chúng vào một Mảng khác bắt đầu từ chỉ mục đích đã chỉ định. Độ dài và chỉ mục được chỉ định là số nguyên 64 bit
* *CopyTo (Array,Int32):* Sao chép tất cả các phần tử của mảng một chiều hiện tại sang mảng một chiều được chỉ định bắt đầu từ chỉ mục mảng đích đã chỉ định. Chỉ số được chỉ định là số nguyên 32 bit
* *CopyTo(Array, Int64):* Sao chép tất cả các phần tử của mảng một chiều hiện tại sang mảng một chiều được chỉ định bắt đầu từ chỉ mục mảng đích đã chỉ định. Chỉ số được chỉ định là số nguyên 64 bit
* Empty<T>() : Trả về một mảng trống
* Sort(Array): Sắp xếp các phần tử trong toàn bộ Mảng một chiều bằng cách sử dụng triển khai IComparable của từng phần tử của Mảng.
* Sort(Array, Array): Sắp xếp một cặp đối tượng Mảng một chiều (một đối tượng chứa các khóa và đối tượng kia chứa các mục tương ứng) dựa trên các khóa trong Mảng đầu tiên bằng cách sử dụng triển khai IComparable của mỗi khóa.
* Sort(Array, Array, IComparer): Sắp xếp một cặp đối tượng Mảng một chiều (một đối tượng chứa các khóa và đối tượng kia chứa các mục tương ứng) dựa trên các khóa trong Mảng đầu tiên bằng cách sử dụng IComparer đã chỉ định.
* GetLongLength(Int32): Lấy số nguyên 64 bit đại diện cho số phần tử trong thứ nguyên được chỉ định của Mảng.
* GetValue(Int32): Nhận giá trị tại vị trí đã chỉ định trong Mảng một chiều. Chỉ số được chỉ định là số nguyên 32 bit.
  1. ***Ví dụ chương trình***

using System;

using System.Diagnostics.CodeAnalysis;

namespace BaoCaoTuan10

{

class TreEm1 : IComparable<TreEm1>

{

public string ten;

public int tuoi;

public string Ten { get; set; }

public int Tuoi { get; set; }

public int CompareTo([AllowNull] TreEm1 other)

{

int result = tuoi - other.tuoi;

if (0== result)

{

result = this.ten.CompareTo(other.ten);

}

return result;

}

public override string ToString()

{

return string.Format("{0},{1}", tuoi, ten);

}

}

class Program

{

static void Main(string[] args)

{

TreEm1[] treEms = new TreEm1[]

{

new TreEm1{tuoi =1, ten = "Na"},

new TreEm1{tuoi =4, ten = "Hoang"},

new TreEm1{tuoi =2, ten = "Linh"},

};

Array.Sort(treEms);

Console.ReadKey();

}

}

}

1. **ARRAYLIST**
   1. ***Khái niệm***

* Là một Collections giúp lưu trữ và quản lý một danh sách các đối tượng theo kiểu mảng (truy cập các phần tử bên trong thông qua chỉ số index).
* Rất giống mảng các object nhưng có thể thêm hoặc xoá các phần tử một cách linh hoạt và có thể tự điều chỉnh kích cỡ một cách tự động
  1. ***Ví dụ về một số các phương thức***

Ảnh có chứa ảnh chụp màn hình

Mô tả được tạo tự động

* 1. ***Ví dụ về chương trinh***

using System;

using System.Diagnostics.CodeAnalysis;

using System.Collections;

namespace BaoCaoTuan10

{

public class Person

{

private string name;

private int age;

public string Name

{

get { return name; }

set { name = value; }

}

public int Age

{

get { return age; }

set { age = value; }

}

public Person(string Name, int Age)

{

this.Name = Name;

this.Age = Age;

}

public override string ToString()

{

return "Name: " + name + " | Age: " + age;

}

}

public class SortPersons : IComparer

{

public int Compare(object x, object y)

{

Person p1 = x as Person;

Person p2 = y as Person;

if (p1 == null || p2 == null)

{

throw new InvalidOperationException();

}

else

{

if (p1.Age > p2.Age)

{

return 1;

}

else if (p1.Age == p2.Age)

{

return 0;

}

else

{

return -1;

}

}

}

}

class Program

{

static void Main(string[] args)

{

ArrayList arrPersons = new ArrayList();

arrPersons.Add(new Person("Nguyen Van A", 18));

arrPersons.Add(new Person("Nguyen Van B", 25));

arrPersons.Add(new Person("Nguyen Van C", 20);

Console.WriteLine("Danh sach Person ban dau: ");

foreach (Person item in arrPersons)

{

Console.WriteLine(item.ToString());

}

arrPersons.Sort(new SortPersons());

Console.WriteLine();

Console.WriteLine("Danh sach Person da duoc sap xep theo tuoi tang dan: ");

foreach (Person item in arrPersons)

{

Console.WriteLine(item.ToString());

}

}

}

}

*3.Hashtable*

3.1. Định nghĩa

Cấu trúc dữ liệu Hash Table là một cấu trúc dữ liệu lưu giữ dữ liệu theo cách thức liên hợp. Trong Hash Table, dữ liệu được lưu giữ trong định dạng mảng, trong đó các giá trị dữ liệu có giá trị chỉ mục riêng. Việc truy cập dữ liệu trở nên nhanh hơn nếu chúng ta biết chỉ mục của dữ liệu cần tìm.

* 1. Công dụng

Giúp các hoạt động chèn và hoạt động tìm kiếm sẽ diễn ra rất nhanh, bất chấp kích cỡ của dữ liệu là bao nhiêu. Hash Table sử dụng mảng như là một kho lưu giữ trung gian và sử dụng kỹ thuật Hash để tạo chỉ mục tại nơi phần tử được chèn vào.

3.3 Một số thuộc tính

|  |  |
| --- | --- |
| Tên thuộc tính | Ý nghĩa |
| Count | Trả về một số nguyên là số phần tử hiện có trong Hasdtable |
| Keys | Trả về 1 danh sách chứa các Key trong hashtable |
| Values | Trả về 1 danh sách chứa các value trong hashtable |

3.4 Một số phương thức

|  |  |
| --- | --- |
| TÊN PHƯƠNG THỨC | Ý NGHĨA |
| ADD(object Key,object Value) | Thêm 1 cặp Key – Value vào hashtable. |
| CLEAR() | Xóa tất cả các phần tử trong hashtable. |
| CLONE() | Tạo 1 bản sao từ Hashtable hiện tại. |
| ContainsKey(object Key) | Kiểm tra đối tượng Key có tồn tại trong Hashtable hay không. |
| ContainsValue(object Value) | Kiểm tra đối tượng Value có tồn tại trong Hashtable hay không. |
| CopyTo(Array array,int Index) | Thục hiên sao cho tất cả phần tử trong Hashtable sang mảng một chiều array từ vị trí Index của array phải là mảng các object hoặc mảng các DictionaryEntry |
| Remove(object Key) | Xóa đối tượng có Key xuất hiện đầu tiên trong Hashtable . |

3.5. Một số lưu ý về HASHTABLE

Mỗi một phần tử trong hashtable(bao gồm 1 cặp Key-Value) được C# định nghĩa là 1 đối tượng có kiểu DictionnaryEntry. Trong DictionaryEntry có 2 thuộc tính chính:

* Key: trả về giá trị Key của phần tử hiện tại.
* Value: trả về giá trị Value của phần tử hiện tại.

3.6. Ví dụ

Mình thử dùng foreach duyệt 1 Hashtable và in ra giá trị Key – Value của mỗi phần tử:

3.6.1. code

Tạo một Hashtable đơn giản với 3 phần tử:

Hashtable hash = new Hashtable();

hash.Add(“T”,”Trieu”);

hash.Add(“A”,”Anhtrieu”);

hash.Add(“NG”,”Nguyen Trieu”);

foreach (DictionaryEntry item in hash)

{

Console.WriteLine(item.Key + “\t” + item.Value);

}

**3.6.2. kết quả**

A Anhtrieu

NG Nguyen Trieu

T Trieu

***4.Queue***

4.1. Định nghĩa

Hàng đợi (Queue) là một cấu trúc dữ liệu trừu tượng, là một cái gì đó tương tự như hàng đợi trong đời sống hàng ngày (xếp hàng).

Khác với ngăn xếp, hàng đợi là mở ở cả hai đầu. Một đầu luôn luôn được sử dụng để chèn dữ liệu vào (hay còn gọi là sắp vào hàng) và đầu kia được sử dụng để xóa dữ liệu (rời hàng). Cấu trúc dữ liệu hàng đợi tuân theo phương pháp First-In-First-Out, tức là dữ liệu được nhập vào đầu tiên sẽ được truy cập đầu tiên.

Trong đời sống thực chúng ta có rất nhiều ví dụ về hàng đợi, chẳng hạn như hàng xe ô tô trên đường một chiều (đặc biệt là khi tắc xe), trong đó xe nào vào đầu tiên sẽ thoát ra đầu tiên. Một vài ví dụ khác là xếp hàng học sinh, xếp hàng mua vé, …

Chú ý: Do Queue cũng là 1 Collections nên để sử dụng ta cần thêm thư viện System.Collections bằng câu lệnh:

Using System.Collections;

4.2. Công dụng:

 Khử đệ quy tổ chức lưu vết các quá trình tìm kiếm chiều rộng và quay lui,vét cạn, tổ chức quản lý và phân phối tiến trình trong các hệ điều hành, tổ chức bộ đệm bàn phím.

4.3. Đặc điểm:

Là một danh sách lưu trữ các đối tượng nhưng không thể truy cập các phần tử thông qua chỉ số phần tử được.

Hành động thêm phần tử vào Queue được gọi là Enqueue (xếp hàng).

Hành động lấy phần tử ra khỏi Queue được gọi là Dequeue (ra khỏi hàng). Và luôn luôn lấy ra phần tử được thêm vào đầu tiên.

4.4 Một số thuộc tính và phương thức được hỗ trợ sẵn trong Queue:

4.4.1. Thuộc tính:

|  |  |
| --- | --- |
| Tên thuộc tính | Ý nghĩa |
| Count | Trả về 1 số nguyên là số phần tử hiện có trong Queue |

4.4.2 Phương thức:

|  |  |
| --- | --- |
| Tên phương thức | Ý nghĩa |
| Clear() | Xóa tất cả các phần tử trong Queue. |
| Clone() | Tạo 1 bản sao từ Queue hiện tại. |
| Contains(object Value) | Kiểm tra đối tượng Value có tồn tại trong Queue hay không. |
| Copy To(Array array, int Index) | Thực hiện sao chép tất cả phần tử trong Queue sang mảng một chiều array từ vị trí Index của array. |
| Peek() | Trả về gái trị của đối tượng tại vị trí đầu trong Queue(phần tử được thêm vào đầu tiên ) nhưng không xóa phần tử khỏi Queue. |
| Dequeue() | Trả về gái trị của đối tượng tại vị trí đầu trong Queue(phần tử được thêm vào đầu tiên ) nhưng đồng thời xóa phần tử khỏi Queue. |
| Enqueue(object Value) | Thêm một phần tử có giá trị Value vào đầu Queue. |
| ToArray() | Tạo ra 1 mảng các object chứa tất cả các phần tử trong Queue và trả về mảng đó. |

4.5. Một số ví dụ:

4.5.1. Code

using System;

using System.Collections;

namespace ConsoleApp3

{

class Program

{

static void Main(string[] args)

{

Queue MyQueue4 = new Queue();

MyQueue4.Enqueue(" AnhTrieu ");

MyQueue4.Enqueue(" KTDL ");

MyQueue4.Enqueue(" HuongDoiTuong ");

Console.WriteLine(" So phan tu hien tai cua Queue la: {0}", MyQueue4.Count);

Console.WriteLine(" Phan tu dau cua Queue la: {0}", MyQueue4.Peek());

Console.WriteLine(" So phan tu cua Queue sau khi goi ham Peek: {0}", MyQueue4.Count);

Console.WriteLine(" Popping...");

int Length = MyQueue4.Count;

for (int i = 0; i < Length; i++)

{

Console.Write(" " + MyQueue4.Dequeue());

}

Console.WriteLine();

Console.WriteLine(" So phan tu cua Queue sau khi Pop la: {0}", MyQueue4.Count);

}

}

}

5.SORTEDLIST

5.1. Định nghĩa

Sortlist là một Collections lưu trữ dữ liệu dưới dạng cặp Key – Value. Key đại diện cho 1 khoá giống như chỉ số phần tử của mảng và Value chính là giá trị tương ứng của khoá đó.

5.2. Đặc điểm

Là một Hashtable nhưng các giá trị sắp xếp theo Key. Việc sắp xếp này được thực hiện một cách tự động mỗi khi thêm 1 phần tử mới vào **SortedList**.

Có thể truy xuất đến các phần tử trong **SortedList**thông qua **Key**(như **Hashtable**) hoặc thông qua chỉ số phần tử (như **ArrayList**).

**SortedList**chính là sự kết hợp giữa **ArrayList**với **Hashtable**.

5.3. Một số thuộc tính và phương thức hỗ trợ sẵn trong Sortlist.

5.4. Một số ví du:

using System;

using System.Collections;

public class SamplesSortedList {

public static void Main() {

SortedList mySL = new SortedList();

mySL.Add("Third", "!");

mySL.Add("Second", "World");

mySL.Add("First", "Hello");

Console.WriteLine( "mySL" );

Console.WriteLine( " Count: {0}", mySL.Count );

Console.WriteLine( " Capacity: {0}", mySL.Capacity );

Console.WriteLine( " Keys and Values:" );

PrintKeysAndValues( mySL );

}

public static void PrintKeysAndValues( SortedList myList ) {

Console.WriteLine( "\t-KEY-\t-VALUE-" );

for ( int i = 0; i < myList.Count; i++ ) {

Console.WriteLine( "\t{0}:\t{1}", myList.GetKey(i), myList.GetByIndex(i) );

}

Console.WriteLine();

}

}

6.STACK

6.1. Định nghĩa

Stack (hay còn gọi là ngăn xếp) là một cấu trúc dữ liệu hoạt động theo nguyên lý lifo. Người ta hay gọi nó là ngăn xếp bởi vì nó hoạt động giống như ngăn xếp trong thực tế vậy.

Vì C# đã hỗ trợ sẵn cấu trúc dữ liệu Stack rồi nên chúng ta chỉ tìm hiểu cách sử dụng nó thôi. Còn cách tổ chức nó như thế nào sẽ được trình bày trong serial về cấu trúc dữ liệu và giải thuật.

* 1. Đặc điểm
* Là một danh sách lưu trữ các đối tượng nhưng không thể truy cập các phần tử thông qua chỉ số phần tử được.
* Hành động thêm phần tử vào Stack được gọi là Push(đẩy vào).
* Hành động lấy phần tử ra khỏi Stack được gọi là Pop(đẩy ra). Và luôn luôn lấy ra phần tử được thêm vào cuối cùng.
  1. Một số thuộc tính và phương thức hỗ trợ sẵn
  2. Một số ví dụ

using System;

using System.Collections;

public class SamplesStack {

public static void Main() {

Stack myStack = new Stack();

myStack.Push("Hello");

myStack.Push("World");

myStack.Push("!");

Console.WriteLine( "myStack" );

Console.WriteLine( "\tCount: {0}", myStack.Count );

Console.Write( "\tValues:" );

PrintValues( myStack );

}

public static void PrintValues( IEnumerable myCollection ) {

foreach ( Object obj in myCollection )

Console.Write( " {0}", obj );

Console.WriteLine();

}

}

**7.List**

**7.1. Định nghĩa**

Là một Collections giúp lưu trữ và quản lý một danh sách các đối tượng theo kiểu mảng (truy cập các phần tử bên trong thông qua chỉ số index).

Rất giống mảng các object nhưng có thể thêm hoặc xoá các phần tử một cách linh hoạt và có thể tự điều chỉnh kích cỡ một cách tự động.

List là một lớp nên khi sử dụng ta cần sử dụng toán tử new:

List<int> Tên list = new List<int>();

Chỉ định sức chứa (CapaciNgoài ra bạn cũng có thể khởi tạo 1 List chứa các phần tử được sao chép từ một Generic Collections khác (lưu ý là có cùng kiểu dữ liệu truyền vào)

Ngoài ra bạn cũng có thể khởi tạo 1 List chứa các phần tử được sao chép từ một Generic Collections khác (lưu ý là có cùng kiểu dữ liệu truyền vào):

\* Khởi tạo 1 List số nguyên có kích thước bằng với MyList2.

\* Sao chép toàn độ phần tử trong MyList2 vào MyList3.

7.2. Một số thuộc tính và Phương thức

7.2.1 Thuộc tính

|  |  |
| --- | --- |
| Thuộc tính | Miêu tả |
| Capacity | Trả về 1 số nguyên cho biết số phần tử mà list có thể chứa. Nếu số phần tử thêm vào chạm sức chứa này thì hệ thống tự động tăng lên. Ngoài ra ta có thể gán 1 sức chứa bất kỳ cho List |
| Count | Trả về một số nguyên là số phần tử hiện có trong list |
| Item | Lấy hoặc thiết lập phần tử tại chỉ mục đã xác định |

7.2.2. Phương thức

|  |  |
| --- | --- |
|  | Phương thức |
| 1 | public virtual int Add(object value);  Thêm một đối tượng vào phần cuối của List |
| 2 | public virtual void AddRange(ICollection c);  Thêm các phần tử của một ICollection vào phần cuối của List |
| 3 | public virtual void Clear();  Gỡ bỏ các phần tử từ List đó |
| 4 | public virtual bool Contains(object item);  Xác định có hay không một phần tử là nằm trong List |
| 5 | public virtual void Insert(int index, object value);  Chèn một phần tử vào List tại chỉ mục đã xác định |
| 6 | public virtual void RemoveAt(int index);  Gỡ bỏ phần tử tại chỉ mục đã xác định của List |
| 7 | public virtual void Reverse();  Đảo ngược thứ tự phần tử trong List |
| 8 | public virtual void SetRange(int index, ICollection c);  Sao chép các phần tử của một collection qua một dãy các phần tử trong List |
| 9 | public virtual void Sort();  Sắp xếp các phần tử trong List |

8.IDictionary

8.1. Định nghĩa

Dictionary trong C# là một Collections lưu trữ dữ liệu dưới dạng cặp Key-Value. Key đại diện cho 1 khoá giống như chỉ số phần tử của mảng và Value chính là giá trị tương ứng của khoá đó. Ta sẽ dử dụng Key để truy cập đến Value tương ứng.

Do Dictionary là 1 Generic Collection nên để sử dụng ta cần thêm thư viện System.Collection.Generic bằng câu lệnh:

using System.Collection.Generic;

8.2. một số thuộc tính và phương thức:

8.2.1. thuộc tính

|  |  |
| --- | --- |
| Tên thuộc tính | Ý nghĩa |
| Count | Trả về 1 số nguyên là số phần tử hiện có trong Dictionary. |
| Keys | Trả về 1 danh sách chứa các Key trong Dictionary. |
| Values | Trả về 1 danh sách chứa các Value trong Dictionary. |

8.2.2. Một số phương thức:

|  |  |
| --- | --- |
| Tên phương thức | Ý nghĩa |
| Add(TKey key,TValue Value) | Thêm 1 cặp Key-Value vào Dictionary. |
| Clear() | Xóa tất cả các phần tử trong Dictionary. |
| ContainsValue(TKey Key) | Kiểm tra đối tượng Key có tồn tại trong Dictionary hay không. |
| ContainsValue(TValue Value) | Kiểm tra đối tượng Value có tồn tại trong Dictionary hay không. |
| Remove(TKey Key) | Xóa đối tượng có Key xuất hiện đầu tiên trong Dictionary. |
| TryGetValue(TKey Key,Tvalue Value) | Kiểm tra Key có tồn tại hay không.Nếu có sẽ trả về True đồng thời trả về giá trị Value tương ứng qua biến Value.  Ngược lại trả về false. |

8.3. Một số ví dụ:

8.3.1. Code

using System;

namespace ConsoleApp2

{

class Program

{

static void Main(string[] args)

{

Hashtable hash = new Hashtable();

hash.Add(“T”,”Trieu”);

hash.Add(“A”,”Anhtrieu”);

hash.Add(“NG”,”Nguyen Trieu”);

foreach (DictionaryEntry item in hash)

{

Console.WriteLine(item.Key + “\t” +item.Value);

}

}

}

}

8.3.2. Kết quả

T Trieu

A AnhTrieu

NG Nguyen Trieu

**9. IEnumerable**

**9.1. Khái niệm**

IEnumerable nằm trong namespace System.Collections

IEnumerable có thể duyệt cac phần tử chỉ 1 chiều tiến lên, nó không thể duyệt ngược lại giữa các phần tử.

IEnumerable tốt nhất khi truy vấn từ một collection in-memory tức là trong bộ nhớ RAM như List, Array…

Khi truy vấn dữ liệu từ database, IEnumerable thực thi câu lệnh select trên server sau đó tải toàn bộ dữ liệu về client rồi mới lọc dữ liệu.

IEnumerable phù hợp với Linq to Object và Linq to XML

IEnumerable không hỗ trợ custom query

IEnumerable không hỗ trợ lazy loading vì thế không phù hợp với trường hợp phân trang.

**9.2.Công dụng**

Mục đích của interface IEnumerable là cho phép chúng ta có thể sử dụng từ khóa foreach trên đối tượng của class cài đặt interface này. Một ví dụ về class chúng ta thường dùng có cài đặt IEnumerable là List. Trong trường hợp chúng ta sử dụng Generics List<> thì interface được cài đặt là IEnumerable<>

**9.3. một số ví dụ**

using System;

using System.Collections;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApp1

{

public class Person

{

public Person(string fName, string lName)

{

this.firstName = fName;

this.lastName = lName;

}

public string firstName;

public string lastName;

}

public class People : IEnumerable

{

private Person[] \_people;

public People(Person[] pArray)

{

\_people = new Person[pArray.Length];

for (int i = 0; i < pArray.Length; i++)

{

\_people[i] = pArray[i];

}

}

IEnumerator IEnumerable.GetEnumerator()

{

return (IEnumerator)GetEnumerator();

}

public PeopleEnum GetEnumerator()

{

return new PeopleEnum(\_people);

}

}

public class PeopleEnum : IEnumerator

{

public Person[] \_people;

int position = -1;

public PeopleEnum(Person[] list)

{

\_people = list;

}

public bool MoveNext()

{

position++;

return (position < \_people.Length);

}

public void Reset()

{

position = -1;

}

object IEnumerator.Current

{

get

{

return Current;

}

}

public Person Current

{

get

{

try

{

return \_people[position];

}

catch (IndexOutOfRangeException)

{

throw new InvalidOperationException();

}

}

}

}

class App

{

static void Main()

{

Person[] peopleArray = new Person[2]

{

new Person("Anh", "Truong"),

new Person("Xuan", "Tran"),

};

People peopleList = new People(peopleArray);

foreach (Person p in peopleList)

Console.WriteLine(p.firstName + " " + p.lastName);

}

}

}

**10.IEnumerator**

**10.1. khái niệm**

Interface IEnumerator<> bao gồm 2 phương thức quan trọng: **MoveNext**, **Reset** và một Property là **Current**.

Property **Current** trả về phần tử hiện tại đang được duyệt tới trong danh sách.

**MoveNext** dùng để đi đến phần tử tiếp theo trong danh sách (hay nói cách khác thay đổi giá trị của Property Current). Phương thức này trả về giá trị true nếu như việc di chuyển đến đối tượng tiếp theo thành công, trả về false nếu thất bại (trong trường hợp đã đến cuối danh sách). Khi **MoveNext** trả về false thì Current sẽ có giá trị không xác định.

**Reset** dùng để đưa con trỏ hiện tại về vị trí ban đầu. Vị trí ban đầu này là vị trí nằm ngày trước phần tử đầu tiên trong danh sách. Phương thức này có thể không cần cài đặt, nó chỉ được dùng để tương thích với các ứng dụng COM.

Chúng ta truyền vào cho constructor của Enumerator một danh sách các phần tử để duyệt. Tham số truyền vào có thể là mảng, danh sách, đối tượng dạng tập hợp… miễn sao phương thức cài đặt tương ứng trong lớp thỏa mãn việc duyệt qua danh sách đó

10.2. một số ví dụ

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace List

{

class Program

{

static void Main(string[] args)

{

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

Name.Add("Truong Hung Anh");

Console.WriteLine(" So luong phan tu trong List la: {0}", Name.Count);

foreach (string item in Name)

{

Console.Write(" " + item);

}

Console.WriteLine();

Name.Insert(0, "Tran Thi Le Xuan");

Console.WriteLine(" So luong phan tu trong List sau khi chen la: {0}", Name.Count);

foreach (string item in Name)

{

Console.Write(" " + item);

}

Console.WriteLine();

bool isExists = Name.Contains("Nguyen Minh Sang");

if (isExists == false)

{

Console.WriteLine(" Khong ton tai");

}

}

}

}

**10.3. Phần kết luận**

Cả hai đều là giao diện

Mã IEnountable là rõ ràng và có thể được sử dụng trong vòng lặp foreach

IEnumerator sử dụng While, MoveNext, current để lấy bản ghi hiện tại

IEnumerable không nhớ trạng thái

IEnumerator vẫn tồn tại trạng thái có nghĩa là hàng nào đang đọc

IEnumerator không thể được sử dụng trong vòng lặp foreach

IEnumerable định nghĩa một phương thức GetEnumerator trả về IEnumerator

IEnumerator cho phép truy cập chỉ đọc vào một bộ sưu tập

11. ILIST

11.1 Thuộc tính

|  |  |
| --- | --- |
| [Count](https://docs.microsoft.com/en-us/dotnet/api/system.collections.icollection.count?view=netframework-4.8#System_Collections_ICollection_Count) | Gets the number of elements contained in the  (Inherited from [ICollection](https://docs.microsoft.com/en-us/dotnet/api/system.collections.icollection?view=netframework-4.8)) |
| [IsFixedSize](https://docs.microsoft.com/en-us/dotnet/api/system.collections.ilist.isfixedsize?view=netframework-4.8#System_Collections_IList_IsFixedSize) | Gets a value indicating whether the [IList](https://docs.microsoft.com/en-us/dotnet/api/system.collections.ilist?view=netframework-4.8) has a fixed size. |
| [IsReadOnly](https://docs.microsoft.com/en-us/dotnet/api/system.collections.ilist.isreadonly?view=netframework-4.8#System_Collections_IList_IsReadOnly) | Gets a value indicating whether the [IList](https://docs.microsoft.com/en-us/dotnet/api/system.collections.ilist?view=netframework-4.8) is read-only. |
| [IsSynchronized](https://docs.microsoft.com/en-us/dotnet/api/system.collections.icollection.issynchronized?view=netframework-4.8#System_Collections_ICollection_IsSynchronized) | Gets a value indicating whether access to the [ICollection](https://docs.microsoft.com/en-us/dotnet/api/system.collections.icollection?view=netframework-4.8) is synchronized (thread safe).  (Inherited from [ICollection](https://docs.microsoft.com/en-us/dotnet/api/system.collections.icollection?view=netframework-4.8)) |
| [Item[Int32]](https://docs.microsoft.com/en-us/dotnet/api/system.collections.ilist.item?view=netframework-4.8#System_Collections_IList_Item_System_Int32_) | Gets or sets the element at the specified index. |
| [SyncRoot](https://docs.microsoft.com/en-us/dotnet/api/system.collections.icollection.syncroot?view=netframework-4.8#System_Collections_ICollection_SyncRoot) | Gets an object that can be used to synchronize access to the [ICollection](https://docs.microsoft.com/en-us/dotnet/api/system.collections.icollection?view=netframework-4.8).  (Inherited from [ICollection](https://docs.microsoft.com/en-us/dotnet/api/system.collections.icollection?view=netframework-4.8)) |

11.2. Methods

|  |  |
| --- | --- |
| [Add(Object)](https://docs.microsoft.com/en-us/dotnet/api/system.collections.ilist.add?view=netframework-4.8#System_Collections_IList_Add_System_Object_) | Adds an item to the [IList](https://docs.microsoft.com/en-us/dotnet/api/system.collections.ilist?view=netframework-4.8). |
| [Clear()](https://docs.microsoft.com/en-us/dotnet/api/system.collections.ilist.clear?view=netframework-4.8#System_Collections_IList_Clear) | Removes all items from the [IList](https://docs.microsoft.com/en-us/dotnet/api/system.collections.ilist?view=netframework-4.8). |
| [Contains(Object)](https://docs.microsoft.com/en-us/dotnet/api/system.collections.ilist.contains?view=netframework-4.8#System_Collections_IList_Contains_System_Object_) | Determines whether the [IList](https://docs.microsoft.com/en-us/dotnet/api/system.collections.ilist?view=netframework-4.8) contains a specific value. |
| [CopyTo(Array, Int32)](https://docs.microsoft.com/en-us/dotnet/api/system.collections.icollection.copyto?view=netframework-4.8#System_Collections_ICollection_CopyTo_System_Array_System_Int32_) | Copies the elements of the [ICollection](https://docs.microsoft.com/en-us/dotnet/api/system.collections.icollection?view=netframework-4.8) to an [Array](https://docs.microsoft.com/en-us/dotnet/api/system.array?view=netframework-4.8), starting at a particular [Array](https://docs.microsoft.com/en-us/dotnet/api/system.array?view=netframework-4.8) index.  (Inherited from [ICollection](https://docs.microsoft.com/en-us/dotnet/api/system.collections.icollection?view=netframework-4.8)) |
| [GetEnumerator()](https://docs.microsoft.com/en-us/dotnet/api/system.collections.ienumerable.getenumerator?view=netframework-4.8#System_Collections_IEnumerable_GetEnumerator) | Returns an enumerator that iterates through a collection.  (Inherited from [IEnumerable](https://docs.microsoft.com/en-us/dotnet/api/system.collections.ienumerable?view=netframework-4.8)) |
| [IndexOf(Object)](https://docs.microsoft.com/en-us/dotnet/api/system.collections.ilist.indexof?view=netframework-4.8#System_Collections_IList_IndexOf_System_Object_) | Determines the index of a specific item in the [IList](https://docs.microsoft.com/en-us/dotnet/api/system.collections.ilist?view=netframework-4.8). |
| [Insert(Int32, Object)](https://docs.microsoft.com/en-us/dotnet/api/system.collections.ilist.insert?view=netframework-4.8#System_Collections_IList_Insert_System_Int32_System_Object_) | Inserts an item to the [IList](https://docs.microsoft.com/en-us/dotnet/api/system.collections.ilist?view=netframework-4.8) at the specified index. |
| [Remove(Object)](https://docs.microsoft.com/en-us/dotnet/api/system.collections.ilist.remove?view=netframework-4.8#System_Collections_IList_Remove_System_Object_) | Removes the first occurrence of a specific object from the [IList](https://docs.microsoft.com/en-us/dotnet/api/system.collections.ilist?view=netframework-4.8). |
| [RemoveAt(Int32)](https://docs.microsoft.com/en-us/dotnet/api/system.collections.ilist.removeat?view=netframework-4.8#System_Collections_IList_RemoveAt_System_Int32_) | Removes the [IList](https://docs.microsoft.com/en-us/dotnet/api/system.collections.ilist?view=netframework-4.8) item at the specified index. |

11.3. Extension Methods

|  |  |
| --- | --- |
| [Cast<TResult>(IEnumerable)](https://docs.microsoft.com/en-us/dotnet/api/system.linq.enumerable.cast?view=netframework-4.8#System_Linq_Enumerable_Cast__1_System_Collections_IEnumerable_) | Casts the elements of an [IEnumerable](https://docs.microsoft.com/en-us/dotnet/api/system.collections.ienumerable?view=netframework-4.8) to the specified type. |
| [OfType<TResult>(IEnumerable)](https://docs.microsoft.com/en-us/dotnet/api/system.linq.enumerable.oftype?view=netframework-4.8#System_Linq_Enumerable_OfType__1_System_Collections_IEnumerable_) | Filters the elements of an [IEnumerable](https://docs.microsoft.com/en-us/dotnet/api/system.collections.ienumerable?view=netframework-4.8) based on a specified type. |
| [AsParallel(IEnumerable)](https://docs.microsoft.com/en-us/dotnet/api/system.linq.parallelenumerable.asparallel?view=netframework-4.8#System_Linq_ParallelEnumerable_AsParallel_System_Collections_IEnumerable_) | Enables parallelization of a query. |
| [AsQueryable(IEnumerable)](https://docs.microsoft.com/en-us/dotnet/api/system.linq.queryable.asqueryable?view=netframework-4.8#System_Linq_Queryable_AsQueryable_System_Collections_IEnumerable_) | Converts an [IEnumerable](https://docs.microsoft.com/en-us/dotnet/api/system.collections.ienumerable?view=netframework-4.8) to an [IQueryable](https://docs.microsoft.com/en-us/dotnet/api/system.linq.iqueryable?view=netframework-4.8). |

11.3. Một số ví dụ

using System;

using System.Collections;

class Program

{

static void Main()

{

var test = new SimpleList();

Console.WriteLine("Populate the List");

test.Add("one");

test.Add("two");

test.Add("three");

test.Add("four");

test.Add("five");

test.Add("six");

test.Add("seven");

test.Add("eight");

test.PrintContents();

Console.WriteLine();

Console.WriteLine("Remove elements from the list");

test.Remove("six");

test.Remove("eight");

test.PrintContents();

Console.WriteLine();

Console.WriteLine("Add an element to the end of the list");

test.Add("nine");

test.PrintContents();

Console.WriteLine();

Console.WriteLine("Insert an element into the middle of the list");

test.Insert(4, "number");

test.PrintContents();

Console.WriteLine();

Console.WriteLine("Check for specific elements in the list");

Console.WriteLine($"List contains \"three\": {test.Contains("three")}");

Console.WriteLine($"List contains \"ten\": {test.Contains("ten")}");

}

}

class SimpleList : IList

{

private object[] \_contents = new object[8];

private int \_count;

public SimpleList()

{

\_count = 0;

}

public int Add(object value)

{

if (\_count < \_contents.Length)

{

\_contents[\_count] = value;

\_count++;

return (\_count - 1);

}

return -1;

}

public void Clear()

{

\_count = 0;

}

public bool Contains(object value)

{

for (int i = 0; i < Count; i++)

{

if (\_contents[i] == value)

{

return true;

}

}

return false;

}

public int IndexOf(object value)

{

for (int i = 0; i < Count; i++)

{

if (\_contents[i] == value)

{

return i;

}

}

return -1;

}

public void Insert(int index, object value)

{if ((\_count + 1 <= \_contents.Length) && (index < Count) && (index >= 0))}

\_count++;

for (int i = Count - 1; i > index; i--)

{

\_contents[i] = \_contents[i - 1];

}

\_contents[index] = value;

}

}

public bool IsFixedSize

{

get

{

return true;

}

}

public bool IsReadOnly

{

get

{

return false;

}

}

public void Remove(object value)

{

RemoveAt(IndexOf(value));

}

public void RemoveAt(int index)

{

if ((index >= 0) && (index < Count))

{

for (int i = index; i < Count - 1; i++)

{

\_contents[i] = \_contents[i + 1];

}

\_count--;

}

}

public object this[int index]

{

get

{

return \_contents[index];

}

set

{

\_contents[index] = value;

}

}

public void CopyTo(Array array, int index)

{

for (int i = 0; i < Count; i++)

{

array.SetValue(\_contents[i], index++);

}

}

public int Count

{

get

{

return \_count;

}

}

public bool IsSynchronized

{get{ return false;}}

public object SyncRoot

{ get{ return this;} }

public IEnumerator GetEnumerator()

{

throw new NotImplementedException("The method or operation is not implemented.");

}

public void PrintContents()

{

Console.WriteLine($"List has a capacity of {\_contents.Length} and currently has {\_count}elements.");

Cosole.Write("List contents:");

fo (int i = 0; i < Count; i++)

Console.Write($" {\_contents[i]}");

}

Console.WriteLine();