



CHỦ ĐỀ 6



GIAO DIỄN VÀ MỘT SỐ LỚP TRONG .NET FRAMEWORK

GVGD: PHẠM THỊ KIM NGOAN

Email: ptkngoan@gmail.com

Nội dung



Giao diện (interface)

- Interface là một khuôn mẫu mà mọi lớp thực thi nó đều phải tuân theo.
- Các Interface chỉ chứa khai báo của các thành phần: phương thức, thuộc tính đóng gói, sự kiện, chỉ mục. Các lớp dẫn xuất phải định nghĩa các thành phần được thực thi từ Interface.
- Interface xác định các class phải làm gì chứ không quy định làm thế nào.
- Các thành phần của interface là public và abstract.

Giao diện (interface)

- Interface không có các thuộc tính (attribute, field)
- Interface có thể được kế thừa từ nhiều interface cơ sở (base interface).
- Interface không có constructor cũng không có destructor.
- Các class có thể thực thi cùng lúc nhiều interface.

Giao diện (interface)

➤ Mục đích sử dụng interface:

- C# không hỗ trợ đa kế thừa nên interface như là 1 giải pháp cho việc đa kế thừa.
- Trong 1 hệ thống việc trao đổi thông tin giữa các thành phần cần được đồng bộ và có những thống nhất chung. Vì thế dùng interface sẽ giúp đưa ra những quy tắc chung mà bắt buộc các thành phần trong hệ thống này phải làm theo mới có thể trao đổi với nhau được.

Giao diện (interface)

➤ Cú pháp

```
<access> interface <interface_name>
{
    methods, properties, events, indexes
}
```

Giao diện (interface)

➤ Ví dụ

```
interface IAnimal
{
    //properties
    string Name { get ; set ; }
    byte Age { get ; set ; }

    //Methods
    void Speak();
    void Eat();

}
```

Giao diện (interface)

- Giao diện kế thừa từ giao diện khác

```
<access> interface <interface_name>: <interface_1>, ...  
{  
    methods, properties, events, indexes  
}
```

Một số interface thường dùng

- ICloneable
- IComparable
- IComparer
- ...

ICloneable

- Ví dụ: Sao chép đối tượng

```
public class Person: ICloneable
{
    string id, name;
    bool male;

    public Object Clone(){
        return this.MemberwiseClone();
    }

    ...
}

class Program{
    Person p = new Person();
    Person p1= (Person)p.Clone();
    p1.Output();
}
```

IComparable

- Ví dụ: sắp xếp tăng dần theo age

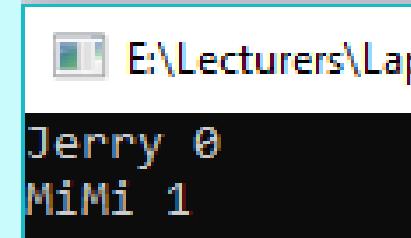
```
public class Cat:IAnimal, IComparable <Cat>
{
    private string name;
    private byte age;

    ...
    public Cat(string n="",byte a = 0)
    {
        name = n;age = a;    }
    public override string ToString()
    {
        return this.name + " " + this.age ;
    }
    public int CompareTo(Cat c)
    {
        if (age > c.age) return 1;
        else if (age == c.age) return 0;
        else return -1;
    }
}
```

IComparable

- Ví dụ: Thực hiện sắp xếp tăng dần

```
class Program
{
    static void Main(string[] args)
    {
        List<Cat> ls = new List<Cat>()
        {
            new Cat {Name ="MiMi",Age=1},
            new Cat { Name = "Jerry", Age = 0 }
        };
        ls.Sort();
        foreach (Cat c in ls)
            Console.WriteLine (c.ToString());
        Console.ReadKey();
    }
}
```



IComparer

using System.Collections;

- Ví dụ: Sắp xếp theo chiều tăng dần

```
class Cat_Age_Asc : IComparer<Cat>
{
    public int Compare(Cat c1, Cat c2)
    {
        if (c1.Age > c2.Age) return 1;
        else if (c1.Age == c2.Age) return 0;
        else return -1;
    }
}
```

IComparer

using System.Collections;

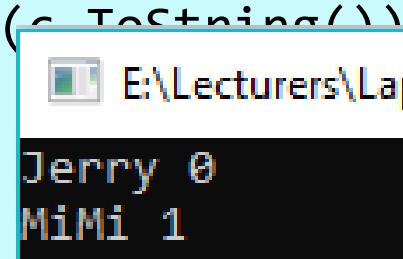
- Ví dụ: Sắp xếp theo chiều giảm dần

```
class Cat_Age_Dsc : IComparer<Cat>
{
    public int Compare(Cat c1, Cat c2)
    {
        if (c1.Age < c2.Age) return 1;
        else if (c1.Age == c2.Age) return 0;
        else return -1;
    }
}
```

IComparer

- Ví dụ: Thực hiện sắp xếp

```
class Program
{
    static void Main(string[] args)
    {
        List<Cat> ls = new List<Cat>()
        {
            new Cat {Name ="MiMi", Age=1},
            new Cat { Name = "Jerry", Age = 0 }
        };
        Cat_Age_Asc Asc = new Cat_Age_Asc();
        // Cat_Age_Dsc Dsc = new Cat_Age_Dsc();
        ls.Sort(Asc); //ls.Sort(Dsc);
        foreach (Cat c in ls)
            Console.WriteLine (c.ToString());
        Console.ReadKey();
    }
}
```



Nội dung



Lớp String

- Sử dụng từ khóa **string** để khai báo một biến chuỗi.
- Từ khóa **string** là một bí danh (alias) của lớp **System.String** trong C#.
- Khai báo chuỗi:

```
string s1 = "Hello"; // tạo chuỗi dùng từ khóa string
String s2 = "every body"; // tạo chuỗi dùng Lớp String
System.String s3 = "Hi"; // tạo chuỗi dùng Lớp String
```

Lớp String

- Phương thức thiết lập của lớp **String**.

```
class Program
{
    static void Main(string[] args)
    {
        char[] letters = { 'H', 'e', 'l', 'l', 'o' };
        String greetings = new String(letters);
        Console.WriteLine(greetings);
        Console.ReadKey();
    }
}
```

Lớp String

- Truy xuất ký tự trong chuỗi: Chuỗi là mảng ký tự nên có thể truy xuất thông qua chỉ số.

```
class Program
{
    static void Main(string[] args)
    {
        char[] letters = { 'H', 'e', 'l', 'l', 'o' };
        String greetings = new String(letters);
        Console.WriteLine(greetings[1]);
        Console.ReadKey();
    }
}
```

Lớp String

- Thuộc tính **Length** của **String**: Lấy số ký tự của đối tượng String hiện tại.

```
string s1 = "Hello"; // tạo chuỗi dùng từ khóa string
s1.Length; // có giá trị là 5
```

- Toán tử nối chuỗi: +

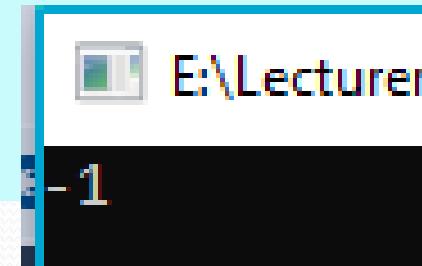
```
string s1 = "Hello"; // tạo chuỗi dùng từ khóa string
String s2 = "every body"; // tạo chuỗi dùng Lớp String
System.String s3 = s1 + s2;
```

Lớp String

- Một số phương thức thông dụng của **String**

int CompareTo(string s1): so sánh với chuỗi s1. Nếu hai chuỗi bằng nhau trả về 0; chuỗi > s1 trả về 1; chuỗi < s1 trả về -1.

```
class Program
{
    static void Main(string[] args)
    {
        string s1 = "Hello";
        string s2 = "Hi every body";
        Console.WriteLine(s1.CompareTo(s2).ToString ());
        Console.ReadKey();
    }
}
```



Lớp String

- Một số phương thức static của **String**

static int Compare(string strA, string strB): so sánh hai xâu strA và strB. Nếu strA bằng strB trả về 0; strA > strB – trả về 1; strA < strB – trả về -1.

```
class Program
{
    static void Main(string[] args)
    {
        string s1 = "Hello";
        string s2 = "Hi every body";
        Console.WriteLine(String.Compare(s1,s2).ToString());
        Console.ReadKey();
    }
}
```



Lớp String

- Một số phương thức static của **String**.

static string Format(string format, Object arg0): giúp tạo ra xâu “động” từ giá trị của các biến.

- Các ký tự định dạng:

C hoặc c là định dạng số chuẩn dành cho kiểu tiền tệ (currency).

D, d - Decimal; F, f - Fixed point; G, g - General;

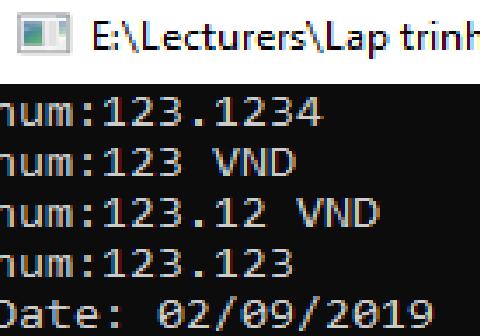
X, x - Hexadecimal; N, n - Number;

P, p - Percent; R, r - Round-trip;

E, e - Scientific.

Lớp String

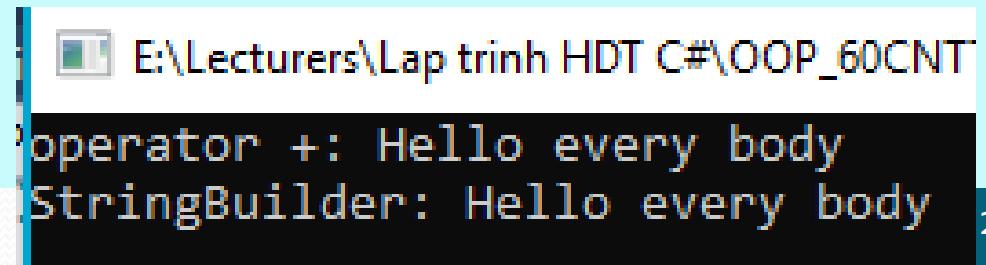
```
class Program
{
    static void Main(string[] args)
    {//định dạng
        float num = 123.1234f;
        Console.WriteLine(string.Format("num:{0}", num));
        Console.WriteLine(string.Format("num:{0:c}", num));
        Console.WriteLine(string.Format("num:{0:c2}", num));
        Console.WriteLine(string.Format("num:{0:f3}", num));
        var day = new DateTime(2019, 9, 2);
        Console.WriteLine(string.Format("Date: {0:d}", day));
        Console.ReadKey();
    }
}
```



```
num:123.1234
num:123 VND
num:123.12 VND
num:123.123
Date: 02/09/2019
```

Lớp StringBuilder

```
class Program
{
    static void Main(string[] args)
    {
        string s1 = "Hello";
        string s2 = "every body";
        s1 = s1 +" "+ s2;
        Console.WriteLine("operator +: "+ s1);
        StringBuilder s = new StringBuilder ("Hello");
        s.Append(" ");
        s.Append("every body");
        Console.WriteLine("StringBuilder: "+ s);
        Console.ReadKey();
    }
}
```



```
E:\Lecturers\Lap trinh HDT C#\OOP_60CNT
operator +: Hello every body
StringBuilder: Hello every body
```

Nội dung



Lớp List

System.Collections.Generic

- List trong C# là một Generic 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**).
- List có thể thêm và bỏ các phần tử tại một vị trí đã xác định, chính nó có thể tự điều chỉnh kích cỡ một cách tự động.
- List cho phép cấp phát bộ nhớ động, thêm, tìm kiếm và sắp xếp các phần tử trong một danh sách.

Lớp List

- Khai báo và khởi tạo lớp **List**

```
List<Data type> List_name = new List<Data type>();
```

- Các phương thức khởi tạo

```
List<Data type> List_name = new List<Data type>(n);
```

//n - số phần tử

```
List<Data type> List_name = new List<Data type>(ls);
```

//ls - một danh sách đã có

Lớp List

➤ Một số thuộc tính trong List

Count: số phần tử hiện có trong list

Capacity: sức chứa của list, nếu thêm phần tử chạm sức chứa thì hệ thống tự động tăng lên

➤ Các phương thức thường dùng:

Add(object o): thêm phần tử o vào cuối list

Clear(): xóa tất cả các phần tử trong list

Contains(T v): kiểm tra đối tượng v có trong list không

Insert(int i, T v): chèn đối tượng v vào vị trí i trong list.

Remove(T v): xóa đối tượng v xuất hiện đầu tiên trong list.

Lớp List

➤ Ví dụ

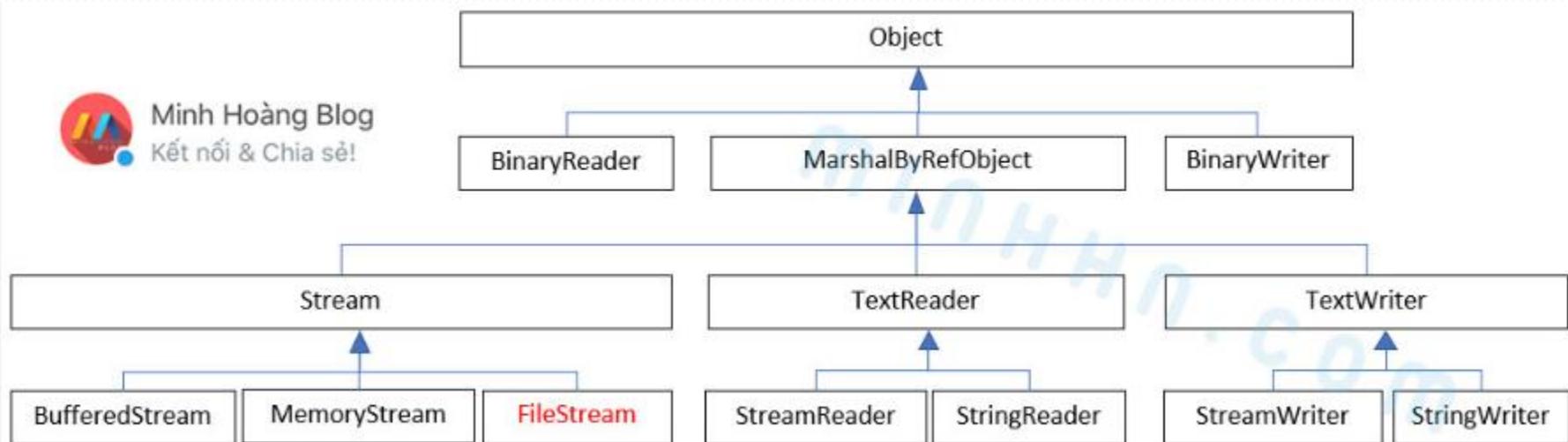
```
class Program
{
    static void Main(string[] args)
    { List<int> ls = new List<int>();
        int n,i=1;
        do //nhập các số đến 0 thì dừng
        { Console.WriteLine("phan tu thu {0}: ", i);
            n = int.Parse(Console.ReadLine());
            ls.Add(n);
            i++;
        } while (n != 0);
        ls.Sort(); //sắp xếp
        foreach (int i = 0;i<ls.Count; i++)
            Console.Write(ls[i].ToString() + "\t");
    }
}
```

Nội dung



Thao tác trên File (FileStream)

- Sơ đồ các class trong System.IO



- Lớp **FileStream** trong giúp đọc, ghi và đóng các File

Thao tác trên File (FileStream)

- Cú pháp để tạo một đối tượng **FileStream**

```
FileStream <object_name> = new FileStream(<file_name>,  
< FileMode Enumerator>, < FileAccess Enumerator>,  
< FileShare Enumerator>);
```

Trong đó:

- **FileMode** enumerator định nghĩa các phương thức khác nhau để mở file.
- **FileAccess** mở file để đọc hay ghi.
- **FileShare** : Dùng để xử lý việc chiếm dụng tài nguyên trong trường hợp có nhiều tiến trình chạy đồng thời.

Thao tác trên File (FileStream)

Trong đó:

- **FileMode** enumerator định nghĩa các phương thức khác nhau để mở file.
 - **Append**: Mở một file đã có và đặt con trỏ tại phần cuối của file, hoặc tạo File, nếu file đó chưa tồn tại.
 - **Create**: Nếu file chưa tồn tại thì tạo file mới.
 - **CreateNew**: Luôn luôn tạo file mới.
 - **Open**: Mở một file đang tồn tại.
 - **OpenOrCreate**: Mở một file nếu nó tồn tại, nếu chưa tồn tại thì sẽ tạo một file mới.
 - **Truncate**: Mở một file đang tồn tại và truncate (cắt) kích cỡ của nó về 0 byte.

Thao tác trên File (FileStream)

Trong đó:

- **FileAccess:**
 - **Read:** Mở file chỉ để đọc
 - **ReadWrite:** Mở file để đọc và ghi
 - **Write:** Mở file chỉ để ghi
- **FileShare:**
 - **Inheritable:** Cho phép một file truyền tính kế thừa tới các tiến trình con.
 - **None:** Từ chối việc chia sẻ file hiện tại.
 - **Read:** Cho phép mở file chỉ để đọc.
 - **ReadWrite:** Cho phép mở file để đọc và ghi.
 - **Write:** Cho phép mở file chỉ để ghi.

Thao tác trên File (FileStream)

```
class Program
{
    static void Main(string[] args)
    {   FileStream f = new FileStream("E:\\binary.txt",
                                    FileMode.OpenOrCreate, FileAccess.ReadWrite);
        for (int i = 1; i <= 20; i++)
        {
            f.WriteByte((byte)i);
        }
        f.Position = 0;
        for (int i = 0; i <= 20; i++)
        {
            Console.Write(f.ReadByte() + " ");
        }
        f.Close();
    }
}
```

Thao tác trên File (StreamReader)

- **StreamReader** để đọc dữ liệu từ file trong C#
- Một số phương thức thường dùng:

Close() Nó đóng đối tượng StreamReader.

Peek() Trả về ký tự có sẵn tiếp theo nhưng không hủy nó.

Read() Đọc ký tự tiếp theo từ Input Stream.

ReadLine() Đọc một dòng.

ReadToEnd() Đọc hết file.

Thao tác trên File (StreamReader)

```
class Program
{
    static void Main(string[] args)
    { try
        {
            FileStream f = new FileStream("E:\\data.txt",
                FileMode.Open);
            StreamReader rd = new StreamReader(f, Encoding.UTF8);
            string line;
            // doc va hien thi cac dong cho toi cuoi file
            while ((line = rd.ReadLine()) != null)
            {
                Console.WriteLine(line);
            }
        }catch
        {
            Console.WriteLine("Khong the doc file da cho: ");
        }
    }
}
```

Thao tác trên File (StreamWriter)

- **StreamWriter** để ghi dữ liệu vào file trong C#
- Một số phương thức thường dùng:

Close() Nó đóng đối tượng StreamWriter.

Flush() Xóa tất cả buffer cho Writer hiện tại.

Write(string value) Ghi một string tới Stream.

Write(int value) Ghi biểu diễn text của một giá trị signed integer 4 byte tới Text string hoặc stream.

Write(double value) Ghi biểu diễn text của một giá trị số thực 8 byte tới Text string hoặc stream.

WriteLine() Ghi một line terminator tới Text string hoặc stream.

Thao tác trên File (StreamWriter)

```
class Program
{
    static void Main(string[] args)
    { try
        {
            FileStream f = new FileStream("E:\\data.txt", FileMode.Create);
            StreamWriter sw = new StreamWriter(f, Encoding.UTF8);
            Console.Write("nhap ho ten sv:");
            string name = Console.ReadLine();
            sw.WriteLine(name);//ghi vào file
            sw.Flush();
            Console.Write("nhap nam sinh:");
            int year =int.Parse (Console.ReadLine());
            sw.WriteLine(year);//ghi vào file
            sw.Flush();
            f.Close();
        }
        catch
        { Console.WriteLine("Khong the ghi du lieu vao file da cho! "); }
    }
}
```

Kết thúc chủ đề 6

