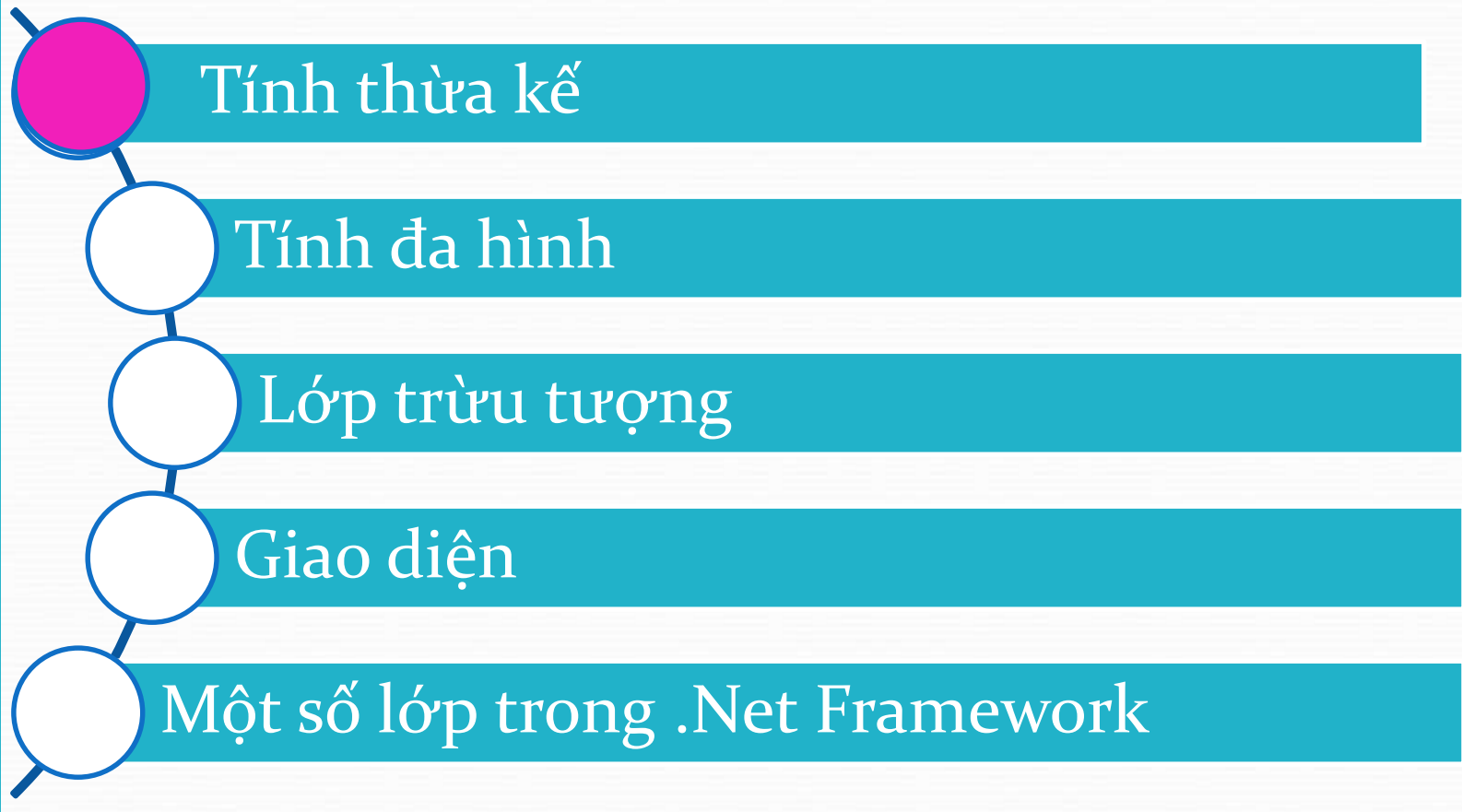


CHỦ ĐỀ 5



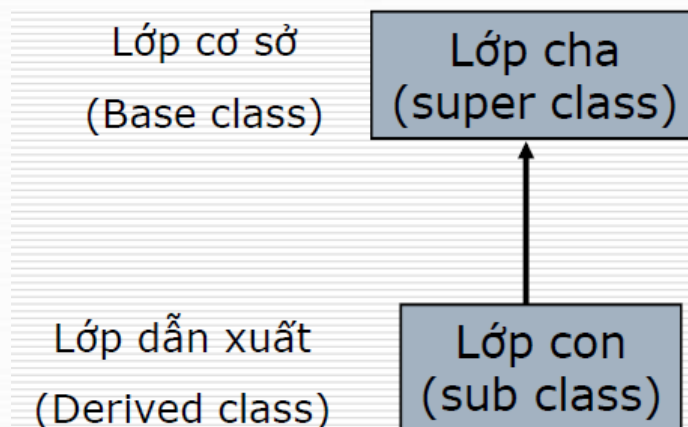
THỪA KẾ VÀ ĐA HÌNH

Nội dung



Lớp cơ sở, lớp dẫn xuất

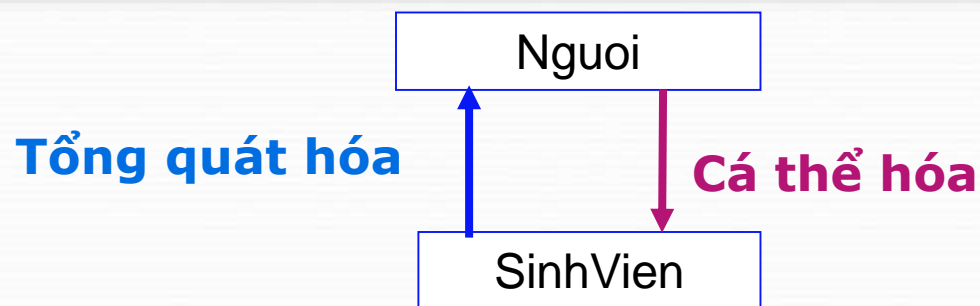
- Tính kế thừa là một khái niệm nền tảng cho phép tái sử dụng mã đang tồn tại.
- Các lớp (class) có thể kế thừa từ lớp khác (định nghĩa class mới dựa trên class đã có). Lớp mới được gọi là lớp dẫn xuất (lớp con), lớp đã có là lớp cơ sở (lớp cha)



Lớp cơ sở, lớp dẫn xuất

- Lớp dẫn xuất thừa kế các thành phần của lớp cơ sở + các thành phần mới.
- Ngầm định một lớp được tạo ra sẽ kế thừa từ lớp `System.Object`.
- Một lớp chỉ được phép kế thừa tối đa 1 lớp, trừ khi được khai báo `sealed`.
- C# cho phép kế thừa phân cấp.

Lớp cơ sở, lớp dẫn xuất



- Lớp cơ sở (lớp cha): Nguoi
- Lớp dẫn xuất (lớp con): SinhVien

Chú ý: Lớp dẫn xuất không thể bỏ đi các thành phần đã được khai báo trong lớp cơ sở.

Tính thừa kế

- Hai điểm mạnh nhất trong lập trình hướng đối tượng là **tính kế thừa** và **đa hình**
- Cú pháp

```
<Phạm vi> class <Tên lớp dẫn xuất>:[tầm vực]<Tên  
lớp cơ sở>  
{  
    - các thành phần dữ liệu  
    + các thành phần phương thức  
    ...  
}
```

Tính thừa kế

(1) Quyền truy xuất tp đó ở lớp cha (lớp cơ sở):

(2) Kiểu dẫn xuất

(1) \ (2)	Thừa kế private	Thừa kế protected	Thừa kế public
private	-	-	-
protected	private	protected	protected
public	private	protected	public

Quyền truy xuất ở lớp con

Tính thừa kế

➤ Ví dụ:

```
class Nguoi
{
    string hoten;
    DateTime ngaysinh;
    bool gioitinh;
    ...
}
class SinhVien: Nguoi
{
    string maso;
    float dtb;
    ...
}
```


Định nghĩa lại phương thức ở lớp con

```
class SinhVien: Người
{
    string maso;
    float dtb;
    public new void Nhap()
    {
        base.Nhap();
        Console.Write("nhap ma so sinh vien:");
        maso = Console.ReadLine();
        Console.Write("nhap diem trung binh:");
        dtb = float.Parse(Console.ReadLine());
    }
}
```

Phương thức khởi tạo không tham số

```
class Nguoi
{
    string hoten;
    DateTime ngaysinh;
    bool gioitinh;
    public Nguoi()
    { hoten = ""; ngaysinh =new DateTime(); gioitinh=true; }
    ...
}
class SinhVien: Nguoi
{
    string maso;
    float dtb;
    public SinhVien ():base()
    { maso = ""; dtb = 0; }
    ...
}
```

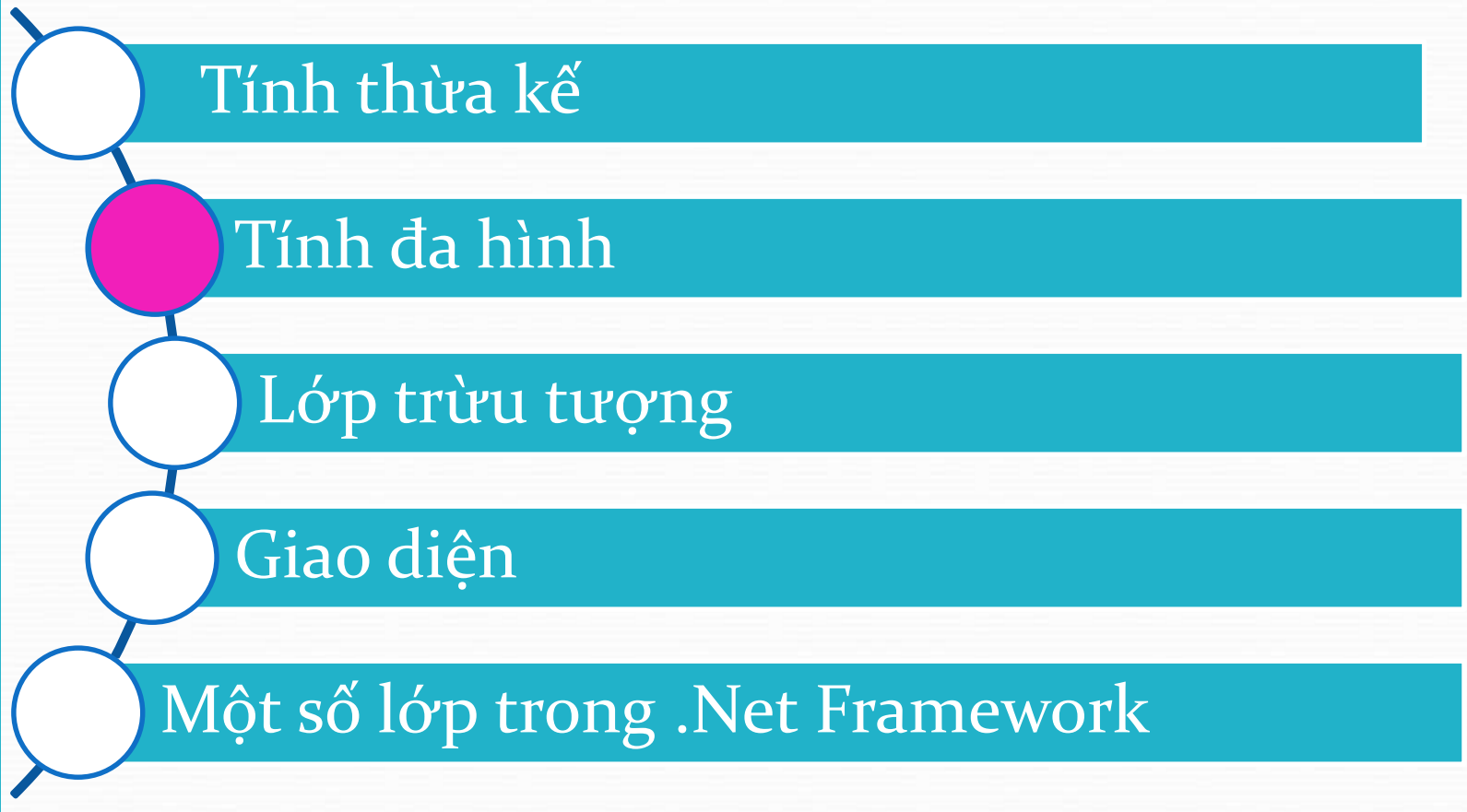
Phương thức khởi tạo có tham số

```
class Nguoi
{
    string hoten;
    DateTime ngaysinh;
    bool gioitinh;
    public Nguoi(string ht, DateTime ns, bool gt)
    { hoten = ht; ngaysinh= ns; gioitinh = gt; }
    ...
}
class SinhVien: Nguoi
{
    string maso;
    float dtb;
    public SinhVien (string ht, DateTime ns, bool gt, string m,
float d):base (ht, ns, gt)
    { maso = m; dtb = d; }
    ...
}
```

Phương thức khởi tạo sao chép

```
class Nguoi
{
    string hoten;
    DateTime ngaysinh;
    bool gioitinh;
    public Nguoi(Nguoi ng)
    {
        hoten = ng.hoten; ngaysinh= ng.ngaysinh;
        gioitinh = ng.gioitinh; }
    ...
}
class SinhVien: Nguoi
{
    string maso;
    float dtb;
    public SinhVien (SinhVien sv):base ((Nguoi) sv)
    {
        maso = sv.maso; dtb = sv.dtb; }
    ...
}
```

Nội dung



Tính đa hình

- **Đa hình** là ý tưởng “sử dụng một giao diện chung cho nhiều phương thức khác nhau”, dựa trên phương thức ảo (virtual method) và cơ chế liên kết muộn (late binding).
- **Tính đa hình** là cơ chế các đối tượng thuộc các lớp khác nhau có thể hiểu cùng 1 thông điệp theo các cách khác nhau.
- Phân loại:
 - **Đa hình tĩnh**: Hàm cụ thể được gọi là phần tử nhận thông điệp lúc **khai báo**.
 - **Đa hình động**: Hàm cụ thể được gọi là phần tử nhận thông điệp lúc **chạy chương trình**.

Tính đa hình tĩnh

- Để thể hiện được tính đa hình tĩnh:
 - Cho phép khai báo các phương thức trùng tên nhau nhưng có tham số khác nhau trong class.
 - Dùng nạp chồng phương thức (overloading).
- Các điểm cần lưu ý khi thực hiện nạp chồng phương thức:
 - Tên của các phương thức phải trùng nhau.
 - Số lượng tham số phải khác nhau.
 - Kiểu dữ liệu của các tham số và thứ tự các tham số phải khác nhau.

Tính đa hình tĩnh

```
class Sinhvien
{
    string ma, ten;
    float dtb;
    public Sinhvien()
    {
        ma = "";ten = "";dtb = 0;
    }
    public Sinhvien (string m, string t, float d)
    {
        ma = m; ten = t; dtb = d;
    }
    ...
}
```


Tính đa hình động

- Để thể hiện được tính đa hình động:
 - Các lớp phải có quan hệ kế thừa với cùng 1 lớp cha nào đó.
 - Phương thức đa hình phải được ghi đè (override) ở các lớp con.

Tính đa hình động

- **Ghi đè (overriding):** được dùng để định nghĩa lại phương thức của lớp cơ sở (lớp cha) trong lớp dẫn xuất (lớp con kế thừa)
- Các điểm cần lưu ý khi thực hiện ghi đè:
 - Phương thức ở lớp cơ sở và lớp dẫn xuất phải có cùng dạng hàm và kiểu dữ liệu trả về
 - Phương thức lớp cơ sở phải được khai báo với từ khóa **virtual**
 - Phương thức lớp dẫn xuất phải được khai báo với từ khóa **override**

Tính đa hình động

```
class Ngươi
{
    string hoten;
    DateTime ngaysinh;
    bool gioitinh;
    public virtual void Xuat()
    {
        if(gioitinh)
            Console.WriteLine("{0}\t Nam \t{1}", hoten,
                               ngaysinh.ToShortDateString());
        else
            Console.WriteLine("{0}\t Nu \t{1}", hoten,
                               ngaysinh.ToShortDateString());
    }
}
```

Tính đa hình động

```
class Sinhvien: Ngươi
{
    string maso;    float dtb;
    public override void Xuat()
    {
        Console.WriteLine("{0}\t", maso);
        base.Xuat();
        Console.WriteLine("\t{0}", dtb);
    }
}
```

Tính đa hình động

```
class program
{
    static void Main()
    {
        DateTime d=DateTime.parse(Console.ReadLine());
        Ngươi ng = new Ngươi("N.V.A", true, d);
        ng.Xuat();
        Ngươi sv = new SinhVien("60165511", "nguyen
                                van An", true, d, 8);
        sv.Xuat();
    }
}
```

Nội dung

- Tính thừa kế
- Tính đa hình
- Lớp trừu tượng
- Giao diện
- Một số lớp trong .Net Framework

Lớp trừu tượng (Abstract Classes)

- **Lớp trừu tượng** là class cơ sở (base class) mà các class khác có thể được dẫn xuất từ nó
- class trừu tượng có thể có cả hai loại phương thức: phương thức trừu tượng và phương thức cụ thể
- Khi nào thì sử dụng class trừu tượng?
 - Nếu muốn tạo các class mà các class này sẽ chỉ là các class cơ sở, và **không** muốn bất cứ ai **tạo các đối tượng** của các kiểu class này.
 - class trừu tượng thường được dùng để biểu thị rằng nó là class không đầy đủ và rằng nó được dự định sẽ chỉ được dùng như là một class cơ sở.

Lớp trừu tượng (abstract)

- class trừu tượng có chứa phương thức trừu tượng
- Phương thức trừu tượng: phương thức **không có phần thân**.

Lớp trừu tượng

➤ Cú pháp

```
<phạm vi> abstract class tên_class  
{  
    Các thành viên cụ thể của class  
    public abstract <kiểu DL> <tênphươngthức>([ds đối số]);  
}
```

➤ Ví dụ

```
abstract class Người  
{  
    string hoten;  
    DateTime ngaysinh;  
    bool gioitinh;  
    public abstract void Xuat(); //phương thức trừu tượng  
}
```

Lớp trừu tượng

- Trong lớp dẫn xuất sẽ phải **override** các phương thức trừu tượng ở lớp cơ sở.

```
abstract class Nguoi
{
    string hoten;
    DateTime ngaysinh;
    bool gioitinh;
    public abstract void Xuat(); //phương thức trừu tượng
}
class SinhVien: Nguoi
{
    string maso;
    float dtb;
    public override void Xuat()
    {
        ...
    }
}
```

Lớp trừu tượng

- Không được tạo ra đối tượng từ lớp trừu tượng

```
class SinhVien: Nguoi
{
    string maso;
    float dtb;
    public void Xuat()
    {
        base.Xuat();
        Console.WriteLine("\t{0}", dtb);
    }
}
```

```
class Program
{
    public static void Main()
    {
        Nguoi ng = new Nguoi(-);
    }
}
```

Lớp niêm phong (Sealed)

- Từ khóa **sealed** được sử dụng để biểu thị khi khai báo một class nhằm ngăn ngừa sự dẫn xuất từ một class, điều này cũng giống như việc ngăn cấm một class nào đó có class con.
- Một **class sealed** cũng không thể là một class trừu tượng.
- Các structs trong C# được ngầm định sealed. Do vậy, chúng không thể được thừa kế

Lớp niêm phong

➤ Cú pháp

```
<phạm vi> sealed class tên_class  
{  
    Các thành viên cụ thể của class  
}
```

➤ Ví dụ

```
sealed class Nguoi  
{  
    string hoten;  
    DateTime ngaysinh;  
    bool gioitinh;  
    public void Xuat()  
    { Console.WriteLine("{0}\t{1}", hoten, ngaysinh);  
    }  
}
```

```
class SinhVien: Nguoi  
{  
    ...  
}
```

Nội dung

- Tính thừa kế
- Tính đa hình
- Lớp trừu tượng
- Giao diện
- Một số lớp trong .Net Framework

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

```
<Phạm vi> interface <Tên giao diện>
{
    khai báo các thành phần (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

```
<Phạm vi> interface <Tên giao diện>: <giao diện 1>,...  
{  
    khai báo các thành phần (method, properties,  
                                event, indexes)  
}
```

Một số interface

- Cloneable
- Comparable
- Comparer
- ...

ICloneable

➤ Ví dụ: Sao chép đối tượng

```
public class SinhVien: ICloneable
{
    string ma, hoten;
    float dtb;
    public Object Clone(){
        return this.MemberwiseClone();
    }
    ...
}

class Program{
    SinhVien sv = new SinhVien();
    SinhVien sv1= (SinhVien)sv.Clone();
    sv1.Info();
}
```

IComparable

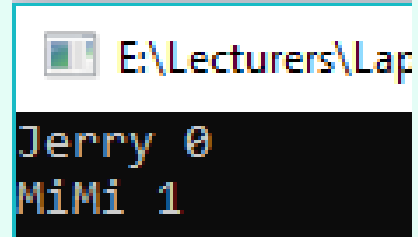
- Ví dụ: sắp xếp tăng dần theo age

```
public class Cat:IAAnimal, 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();
    }
}
```



```
E:\Lecturers\Lap
Jerry 0
MiMi 1
```


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;

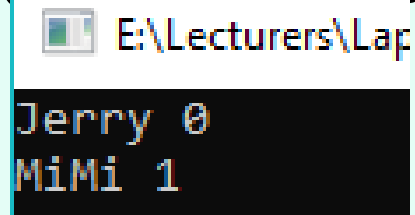
- Ví dụ: Sắp xếp theo chiều giảm dần

```
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();
    }
}
```



E:\Lecturers\Lap

Jerry 0
MiMi 1

Nội dung

- Tính thừa kế
- Tính đa hình
- Lớp trừu tượng
- Giao diện
- Một số lớp trong .Net Framework

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: +

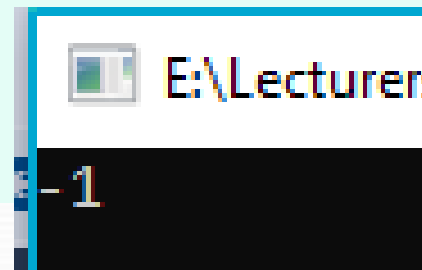
```
string s1 = "Hello"; // tạo chuỗi dùng từ khóa string  
String s2 = "every body"; // tạo chuỗi dùng Lớp String  
System.String s3 = s1 + s2;
```


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();
    }
}
```



E:\Lectures

-1

Lớp String

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

`static string Format(string format, Object arg0):` giúp tạo ra chuỗi “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

➤ Ví dụ:

```
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();
    }
}
```

E:\Lecturers\Lap trinh

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

Lớp StringBuilder

➤ Ví dụ:

```
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
```

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<kiểu dữ liệu> Tên_List = new List<kiểu dữ liệu>();
```

- Các phương thức khởi tạo

```
List<kiểu dữ liệu> Tên_List = new List<kiểu dữ liệu>(n);
```

//n - số phần tử

```
List<kiểu dữ liệu> Tên_List = new List<kiểu dữ liệu>(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.Write("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");
```

```
    }
```

```
}
```

Kết thúc chủ đề 5

