

## Bài 5. Lập trình hướng đối tượng (tiếp)

### 4 tính chất của LTHĐT:

- ✓ Tính đóng gói
- ✓ Tính kế thừa
- ✓ Tính đa hình
- ✓ Tính trừu tượng

### 1. Lớp trong lập trình hướng đối tượng (class)

### 2. Các thành phần của lớp

### 3. Lớp và phương thức tĩnh (Static)

#### ✓ Tạo một lớp tĩnh:

```
[Phạm vi] static class <tên_lớp>
{
    <Khai_báo_các_thành_phần_tĩnh: biến tĩnh, phương thức tĩnh>
}
```

- Các lớp tĩnh không thể kế thừa và không thể khởi tạo 1 đối tượng cho lớp tĩnh.
- Các lớp tĩnh chỉ có các biến thành viên tĩnh và các phương thức tĩnh.
- Có thể có phương thức khởi tạo tĩnh không tham số để khởi tạo các thành viên tĩnh.
- Lớp tĩnh thường được dùng với mục đích khai báo một lớp tiện ích chứa các hàm tiện ích hoặc hằng số.

#### ✓ Tạo một phương thức tĩnh:

```
[Phạm vi] static <kiểu_trả_về> <tên_PT>([DSTS])
{
}
```

- Phương thức tĩnh là một phương thức dùng chung của lớp, **được gọi thông qua tên lớp và không cần khởi tạo bất kỳ đối tượng nào**, từ đó tránh việc lãng phí bộ nhớ.
- Hỗ trợ trong việc viết các hàm tiện ích của thư viện để sử dụng lại.

#### ✓ Gọi phương thức tĩnh:

*Tên\_lớp\_tĩnh.Tên\_phương\_thức\_tĩnh(danh sách tham số)*

#### Ví dụ:

```
static class Rectangle
{
    public static int width;
    public static int height;
    static Rectangle()
    {
        width = 0; height = 0;
    }
    public static int Area()
    { return width * height; }
```

```

public static int Perimeter()
{ return (width * height)*2; }

    public static void Input()
    {
        Console.WriteLine("Width:");
        width = int.Parse(Console.ReadLine());
        Console.WriteLine("Height:");
        height = int.Parse(Console.ReadLine());
    }

    public static void Output()
    {
        Console.WriteLine("Rectangle: width=" + width + ", height=" + height);
    }
}

class Program
{
    static void Main(string[] args)
    {
        //Rectangle r = new Rectangle(); => Lỗi
        // r.Input(); => lỗi
        //Rectangle.Input(); => đúng
        Rectangle.width = 9;
        Rectangle.height= 8;
        Rectangle.Output();
        Console.WriteLine("Area is: " + Rectangle.Area());
        Console.WriteLine("Perimeter is: " + Rectangle.Perimeter());
        Console.ReadLine();
    }
}

```

## 4. Kế thừa

### 4.1. Khái niệm:

Kế thừa là một trong các đặc điểm của lập trình hướng đối tượng. Nhờ sự kế thừa mà ta có thể tạo ra các lớp mới từ lớp cơ sở đã có. Khi sử dụng kế thừa ta có thể tái sử dụng mã chương trình, tạo ra một giao diện chung mà các đối tượng thừa kế từ giao diện này có thể truy cập. Lớp mới kế thừa từ lớp cơ sở có thể thừa kế từ các thuộc tính và phương thức của lớp cơ sở (base class).

### 4.2. Các lưu ý khi viết chương trình có kế thừa

#### a. Khi định nghĩa lớp kế thừa, dùng cú pháp:

*[phạm vi] class <tên lớp kế thừa>: <tên lớp cơ sở>*

```

{    ...    }

```

Ví dụ:

```

class SINHVIEN: NGUOI
{    ...    }

```

#### b. Khi định nghĩa hàm tạo (Constructor) ở lớp kế thừa.

*[private/public] <tên lớp kế thừa>([DSTS]):base([DSTS])*

```

{    ...    }

```

Ví dụ:

```
public SINHVIEN(): base()  
{ ... }
```

c. Phải định nghĩa phương thức trong lớp cơ sở với từ khóa **virtual** thì phương thức này mới được gọi lại (**override**) trong lớp kế thừa.

```
[private/public] virtual void<kiểu><tên PT>([DSTS])  
{ ... }
```

Ví dụ:

```
public virtual void Input()  
{ ... }
```

d. Khi ở trong phương thức của một lớp kế thừa, muốn gọi phương thức ở lớp cơ sở.

```
base.<tên PT>([DSTS]);
```

Ví dụ:

```
base.Input(); //Input() là pt ở lớp cơ sở
```

e. Khi khai báo đối tượng trong hàm Main(), ta không cần quan tâm tới lớp cơ sở.

Ví dụ:

```
SINHVIEN SV1=new SINHVIEN();
```

## Bài tập

**Bài 1:** Xây dựng lớp cơ sở **NGUOI** với các thuộc tính: họ tên, địa chỉ, tuổi và các phương thức: **constructor**, **Input()**, **Show()** nhập xuất dữ liệu tương ứng với các thuộc tính đã có. Xây dựng lớp **SINHVIEN** kế thừa từ lớp **NGUOI** có thêm các thuộc tính và phương thức sau:

- Thuộc tính thêm : điểm trung bình.
- Phương thức: **constructor**, **Input()**, **Show()**.
- Viết mã trong phương thức Main ở lớp Program để gọi đủ các phương thức đã viết.
- Viết mã trong phương thức Main ở lớp Program nhập vào một danh sách các sinh viên. In danh sách các sinh viên lên màn hình. (dùng **List**)

**Giải:**

```
class NGUOI  
{  
    // khai báo thuộc tính  
    public string hoten { get; set; }  
    public int tuoi { get; set; }  
    public string diachi { get; set; }  
    public NGUOI()  
    { tuoi = 0; hoten = ""; diachi = ""; }  
  
    public NGUOI(string ht, int t, string dc)  
    { hoten = ht; tuoi = t; diachi = dc; }  
  
    public virtual void Input()  
    {  
        Console.OutputEncoding = Encoding.UTF8;
```

```

        Console.Write("Vào họ tên:"); hoten = Console.ReadLine();
        Console.Write("Vào tuổi:"); tuoi = Int32.Parse(Console.ReadLine());
        Console.Write("Vào địa chỉ:"); diachi = Console.ReadLine();
    }
    public virtual void Show()
    {
        Console.WriteLine("{0,-20}{1,-10}{2,-30}", hoten, tuoi, diachi);
    }
}

class SINHVIEN : NGUOI
{
    public double dtb { get; set; }
    public SINHVIEN() : base()
    { dtb = 0; }
    public SINHVIEN(string ht, int t, string dc, double tb) : base(ht, t, dc)
    { dtb = tb; }
    public override void Input()
    {
        //Console.OutputEncoding = Encoding.UTF8;
        base.Input();
        Console.Write("Vào ĐTB:");
        dtb = Double.Parse(Console.ReadLine());
    }
    public override void Show()
    {
        base.Show();
        Console.WriteLine("{0,-10}", dtb);
    }
}

class Program
{
    static void Main(string[] args)
    {
        //sử dụng constructor không ts
        /* SINHVIEN sv1 = new SINHVIEN();
        sv1.Input();
        sv1.Show();
        Console.ReadLine(); */
        /*-----*/
        //sử dụng constructor có ts
        /* SINHVIEN sv2 = new SINHVIEN("Mai", 18, "Hà Nội", 8);
        sv2.Show();
        Console.ReadLine(); */
        /*-----*/

        //Dùng List
        List<SINHVIEN> DSSV = new List<SINHVIEN>();
        int n;
        Console.Write("Nhập số lượng sinh viên: ");
        n = int.Parse(Console.ReadLine());
        Console.WriteLine("\n=====Nhập danh sách Sinh viên =====");
        for (int i = 0; i < n; i++)
        {
            Console.WriteLine("Nhập sinh viên thu {0}: ", i + 1);

```

```

        SINHVIEN sv= new SINHVIEN();
        sv.Input();
        DSSV.Add(sv);
    }
    Console.WriteLine("\n====Danh sanh sv vua nhap la =====");
    Console.WriteLine("{0,-20}{1,-10}{2,-30}{3,-10}", "Ho ten", "Tuoi", "Dia chi", "Diem TB");
    foreach (SINHVIEN item in DSSV)
        item.Show();
    Console.ReadLine();
}
}

```

**Bài 2:** Xây dựng lớp **PERSON** gồm có các thuộc tính: họ tên, giới tính, năm sinh và các phương thức: **constructor, nhập, xuất** dữ liệu tương ứng với các thuộc tính đã có.

Xây dựng lớp **CONGNHAN** kế thừa từ lớp **PERSON** có thêm các thuộc tính và phương thức sau:

- Thuộc tính thêm : tên công ty, hệ số lương.
- Phương thức: **constructor, nhập, xuất, Thu nhập** ( $\text{thu nhập} = \text{hệ số lương} * 850000$ )
- Viết mã trong phương thức **Main** ở lớp **Program** để:
  - + Gọi đủ các phương thức đã viết.
  - + Nhập vào một danh sách các công nhân. In danh sách các công nhân lên màn hình (cả thu nhập)
  - + In danh sách các công nhân có hệ số lương cao nhất.

## 5. Giao diện (Interface)

- Một lớp dẫn xuất trong C# không thể kế thừa hai hoặc nhiều lớp cơ sở vì C# không hỗ trợ đa kế thừa.  
Để khắc phục nhược điểm này, các giao diện đã được giới thiệu.
- Một lớp trong C# có thể triển khai nhiều giao diện.
- Một lớp thực thi nhiều giao diện phải triển khai tất cả các phương thức được khai báo trong các giao diện.
- Từ khóa ghi đề không được sử dụng trong khi triển khai các phương thức trừu tượng của một giao diện.



<pre> class A { } class B { } class C: A, B { }                 </pre>	<pre> interface A { } interface B { } class C : A, B { }                 </pre>
--	---

- Một giao diện chỉ chứa các thành viên trừu tượng mà không thể có mã thực hiện bất kỳ phương thức nào.
- Một giao diện không thể được khởi tạo mà chỉ có thể được kế thừa bởi các lớp hoặc các giao diện khác.
- Trong C #, theo mặc định, tất cả các thành viên được khai báo trong một giao diện là **public**.

## Khai báo:

```
interface Tên_interface
{
    Các thuộc tính;
    Các phương thức;
}
```

## Ví dụ 1: tạo interface thao tác với file.

```
interface IFile
{
    void ReadFile();
    void WriteFile(string text);
}
```

## Ví dụ 2: tạo một giao diện và thực thi.

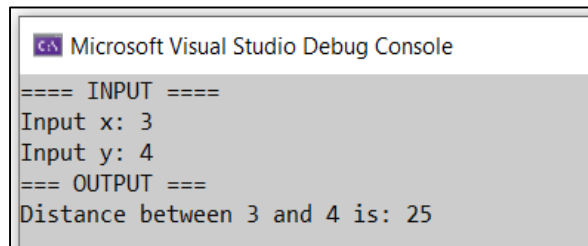
```
//Tạo giao diện
interface IPoint
{
    int x{ get; set; }
    int y{ get; set; }
    double distance{ get; }
    void InputXy();
    void OutputXy();
}
```

Chỉ có tên PT

```
//Thực thi giao diện
class Point : IPoint
{
    public int x { get; set; }
    public int y { get; set; }
    public double distance
    {
        get { return (x * x + y * y); }
    }
    public void InputXy()
    {
        Console.WriteLine("==== INPUT ===");
        Console.Write("Input x: "); x = int.Parse(Console.ReadLine());
        Console.Write("Input y: "); y = int.Parse(Console.ReadLine());
    }
    public void OutputXy()
    {
        Console.WriteLine("=== OUTPUT===");
        Console.WriteLine("Distance between " + x + " and " + y + " is: " + distance);
    }
}

//Sử dụng giao diện
class Program
{
    static void Main(string[] args)
    {
        Point p = new Point();
        p.InputXy();
        p.OutputXy();
        Console.ReadLine();
    }
}
```

Trong lớp kế thừa từ giao diện:  
sẽ khai báo các phương thức và thuộc  
tính cùng tên với khai báo trong giao diện.

A screenshot of the Microsoft Visual Studio Debug Console window. The title bar reads "Microsoft Visual Studio Debug Console". The console content shows the following text:

```
==== INPUT ====  
Input x: 3  
Input y: 4  
=== OUTPUT ===  
Distance between 3 and 4 is: 25
```

==== INPUT ====  
Input x: 3  
Input y: 4  
=== OUTPUT ===  
Distance between 3 and 4 is: 25