NGÔN NGỮ LẬP TRÌNH C#

Nguyễn Thị Hồng Anh
Bộ môn: Kỹ thuật phần mềm

Chương 2. Lập trình hướng đối tượng với C#

Nội dung

- Lớp và đối tượng
- Property
- Tính kế thừa
- Tính đa hình
- Tính trừu tượng
- Interface
- Partial
- Namespace

Lớp và đối tượng

<access modifier> class <tên lớp>
{
- Khai báo các thành phần dữ liệu của lớp
- Khai báo và định nghĩa các hàm thiết lập của lớp
- Khai báo và định nghĩa các hàm hủy bỏ của lớp
- Khai báo và định nghĩa các hàm của lớp
}

Access modifier

Access modifier	Phạm vi
public	Không hạn chế phạm vi
private	Chỉ có trong phạm vi lớp (default cho các thành phần của lớp)
protected	có phạm vi ở lớp và lớp con
internal	giới hạn phạm vi trong cùng project (assembly) hiện tại - default cho class
protected internal	giới hạn phạm vi trong assembly hiện tại và lớp con

Ví dụ

```
namespace ConsoleApp1{
    public class HinhTron
    {
        double bk;
        public void nhap()       {
            Console.WriteLine("Moi nhap bkinh:");
            bk=double.Parse(Console.ReadLine());
        }
        public double chuVi()
        {
            return 2*3.14*bk;
        }
    }
}
```

```
public class Program
    {
        static void Main(string[] args)
        {
            HinhTron ht = new HinhTron();
            ht.bk = 8;
            double cv = ht.chuVi();
            Console.WriteLine("chu
vi={0}", cv);
        }
    }
```

Cho biết kết quả đoạn chương trình trên?

Types of variables

- Local variable
- Instance variable/ non-static variable
- static variable/ class variable
- constant variable
- read only variable

Types of variable

- **instance variable:**
  - Không có từ khóa static
  - khai báo trong lớp, ngoài phương thức
  - Ví dụ:

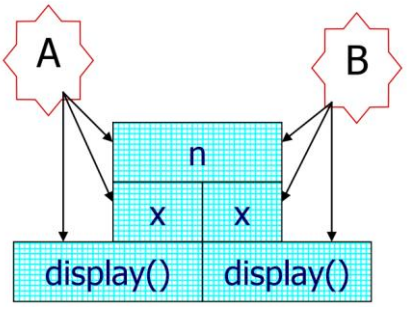
```
class Marks {

    // These variables are instance variables.
    // These variables are in a class and
    // are not inside any function
    int engMarks;
    int mathsMarks;
    int phyMarks;

    // Main Method
    public static void Main(String[] args)
    {
```


Types of variable

- Static variable
  - có từ khóa static
  - Khai báo trong lớp ngoài phương thức

```
class abc{
   private static int n;
   private float x;
   public void display(){
}
```

Khai báo: abc A, B;



- Constant variable
  - Khai báo với từ khóa const
  - Phải gán giá trị ngay khi khai báo
  - Hoạt động giống static variable
  - Ví dụ:
    - const int x; // error
    - const float y=6;


```
internal class Program
{
    int x;
    int a = 10;
    static int b = 20;
    const float max = 50;
    0 references
    public static void Main()
    {
        Program obj = new Program();
        Console.WriteLine("The value of a is = " + obj.a);
        Console.WriteLine("The value of b is = " + Program.b);
        Console.WriteLine("The value of max is = " + Program.max);
    }
}
```

- read only variable
  - Được khai báo sau từ khóa readonly
  - Chỉ thiết lập được giá trị khi khai báo hoặc trong các constructor
  - Hoạt động giống instance variable
  - Ví dụ:


```
internal class Program
{
    int x;
    int a = 10;
    static int b = 20;
    const float max = 50;
    readonly int k ;

    1 reference
    public Program()
    {
        k = 9; ;
    }
    0 references
    public void thu()
    {
        k = 7; // error
    }
    0 references
    public static void Main()
    {
        Program obj = new Program();
        Console.WriteLine("The value of a is = " + obj.a);
        Console.WriteLine("The value of b is = " + Program.b);
        Console.WriteLine("The value of max is = " + Program.max);
        Console.WriteLine("k=" + obj.k);
    }
}
```

Constructor

- Có tên trùng tên lớp
- Không có giá trị trả về và không cần khai báo void
- Hàm thiết lập có thể xây dựng hàm chồng
- Khi không xây dựng hàm thiết lập thì sử dụng hàm thiết lập ngầm định (không tham số)

Destructor - Hàm hủy bỏ

- Được gọi đến khi một đối tượng của lớp được hủy khỏi bộ nhớ với chức năng giải phóng bộ nhớ đã được cấp cho các thành phần dữ liệu của đối tượng.
- Một số qui định:
  - Tên hàm hủy bỏ bắt đầu bằng dấu ~, tiếp theo tên lớp
  - Hàm hủy bỏ không có kiểu dữ liệu trả về và không cần khai báo kiểu void
  - Không cần khai báo tham số, mỗi lớp chỉ cần 1 hàm hủy bỏ
  - Nếu lớp không định nghĩa hàm hủy bỏ khi biên dịch chương trình tự động sinh 1 hàm hủy, nhưng hàm này không làm gì cả.

Getter/ Setter/ Property

- Setter/ Getter/ Properties: Thiết lập và lấy giá trị các thuộc tính có kiểu private
- Cú pháp properties:

```
public  <kiểu_trả_về> <Tên_properties>
{  get{
   // các câu lệnh
   return giá_trị;
   }
   set{
   // các câu lệnh xử lý value
   tên thuộc tính = value;
}}
```



```
namespace LopThu3
{
    2 references
    internal class GetSet
    {
        private int id;
        2 references
        public int Id
        {
            get
            {
                return id;
            }
            set
            {
                if (value < 0)
                {
                    Console.WriteLine("id khong duoc de so am");
                }
                else
                    id = value;
            }
        }

    }
}
```

```
using LopThu3;

0 references
internal class Program
{
    0 references
    public static void Main()
    {
        GetSet ob = new GetSet();
        ob.Id = -1;
        Console.WriteLine("ID=" + ob.Id);
    }
}
```

Bài tập

1. Xây dựng lớp NhanVien chứa các thông tin:

- Mã số, họ tên (kiểu chuỗi)
- Số ngày công (số nguyên) (giá trị>0)
- Xếp loại (char (A,B,C)). Kết quả xếp loại thi đua dựa vào qui đinh:

+ Số ngày công >26 : loại A

+ 26 > = Số ngày công > = 22 : loại B

+ Số ngày công <22 : loại C

- Lương ngày (200.000 đồng): Áp dụng cho tất cả các nhân viên

Bài tập

Hãy xây dựng thêm các phương thức sau:

- Property cho từng thuộc tính có kiểm tra ràng buộc theo yêu cầu nêu trên
- 3 hàm khởi tạo
- Hàm hủy
- Nhập/xuất thông tin nhân viên
- Hàm tính lương (số ngày công * lương ngày)
- Hàm tính thưởng: nếu xếp loại A thì thưởng 5% lương, loại B thưởng 2% lương và loại C không thưởng

Bài tập

2. Xây dựng lớp (TinhTienDien) để tính số điện đã sử dụng và tiền điện (TĐ) cho các hộ gia đình trong một khu dân cư. Thông tin gồm họ tên chủ hộ, mã số công tơ điện, chỉ số cũ, chỉ số mới. Tiền điện của các hộ gia đình sẽ được tính theo công thức sau:

- Nếu số điện sử dụng (SĐSD) nhỏ hơn định mức (ĐM) thì TĐ = SĐSD* đơn giá trong định mức (ĐGTĐM).

- Ngược lại giá tiền mỗi số điện vượt định mức sẽ được tính gấp rưỡi so với ĐGTĐM. ĐM và ĐGTĐM áp dụng chung cho tất cả các hộ và có thể thay đổi theo thời gian.

Viết chương trình cho phép người dùng nhập vào thông tin và các chỉ số điện của 1 hộ gia đình, sau đó xuất ra thông tin và tiền điện của chủ

nhà.

Lưu ý: Giá trị của định mức và đơn giá định mức áp dụng cho tất cả các hộ gia đình, lấy theo giá thực tế và có thể thay đổi theo thời gian.

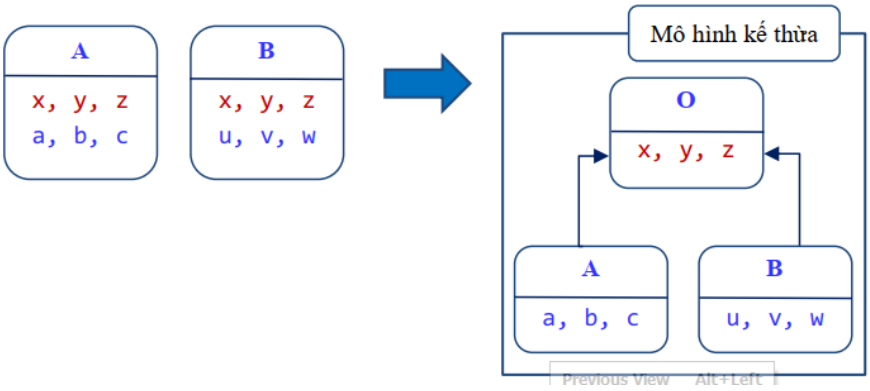
Tính kế thừa

- Cho phép định nghĩa các lớp mới dựa trên các lớp đã có
- Lớp được kế thừa là lớp cha (base, super)
- Lớp kế thừa lớp khác gọi là lớp con (derrived, sub)
- C# không có đa kế thừa

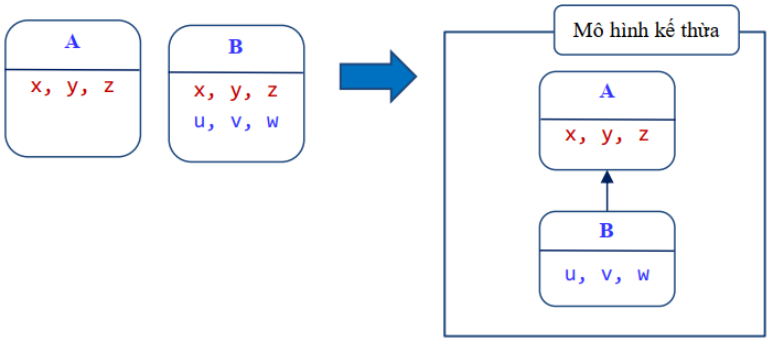
Cú pháp xây dựng lớp kế thừa

```
[<Quyền truy cập>] class <Tên_Lớp_Cha>
{
// Nội dung lớp cha
}

[<Quyền truy cập>] class <Lớp_Con> : <Lớp_Cha>
{
// Nội dung lớp con
}
```


Mô hình kế thừa



Mô hình kế thừa



Truy cập các thành phần của lớp cha

- <lớp cha>.<tên PT>(danh sách các tham số)
- Gọi phương thức của lớp cha:
  - base.tên_PT(tham số)
- Không dùng từ khóa base cho các thành phần static

```
using System;
  public class Employee
   {
      public float salary = 40000;
   }
  public class Programmer: Employee
   {
      public float bonus = 10000;
   }
  class TestInheritance{
      public static void Main(string[] args)
       {
          Programmer p1 = new Programmer();

          Console.WriteLine("Salary: " + p1.salary);
          Console.WriteLine("Bonus: " + p1.bonus);

       }
   }
```


Hàm khởi tạo trong kế thừa

- Phương thức khởi tạo của lớp cha luôn luôn được gọi mỗi khi có một đối tượng thuộc lớp con khởi tạo. Và được gọi trước phương thức khởi tạo của lớp con.
- Cú pháp tạo phương thức khởi tạo ở lớp con

```
public <Tên_Lớp_Con>(<ds_tham_số>):base(<danh_sách_tham_số>)
{
  //gán giá trị cho các thành phần dữ liệu bổ sung của lớp con
}
```

Override

- Khai báo phương thức ở lớp cha

```
public virtual <type> <tên PT> ([danh sách tham số])
{...}
```

- Khai báo phương thức override ở lớp con

```
public override <type> <tên PT> ([danh sách tham số])
{...}
```

- **Lưu ý:** *virtual methods không được để private*

Bài tập

1. Một đơn vị sản xuất gồm có các cán bộ là công nhân, kỹ sư, nhân viên.

+ Mỗi cán bộ cần quản lý lý các thuộc tính: Họ tên, năm sinh, giới tính, địa chỉ (Giới tính phải ghi là Nam hoặc Nữ, độ tuổi phải >=18)

+ Các công nhân cần quản lý: Bậc (công nhân bậc 3/7, bậc 4/7 ...)

+ Các kỹ sư cần quản lý: Ngành đào tạo

+ Các nhân viên phục vụ cần quản lý thông tin: công việc

a. Xây dựng các lớp NhanVien, CongNhan, KySu kế thừa từ lớp CanBo

b. Xây dựng các hàm để truy nhập, hiển thị thông tin và kiểm tra về các thuộc tính của các lớp.

c. Tạo 3 đối tượng có kiểu lần lượt là công nhân, kỹ sư, nhân viên. Thực hiện nhập và xuất thông tin các cán bộ vừa tạo

Tính đa hình

```
internal class LopCon : Cha
{
    protected string info;
    1 reference
    public string Info...
    0 references
    public LopCon()...
    1 reference
    public LopCon(string name, string info)...
    3 references
    public override void Display()
    {
        base.Display();
        Console.WriteLine($"- {info}");
    }
}
```

```
internal class Cha
{
    protected string name;
    1 reference
    public string Name...
    0 references
    public Cha() ...
    1 reference
    public Cha(string name) ...
    3 references
    public virtual void Display() {
    Console.Write($"{ name}");
    }
}
```

Tính đa hình

- Cho biết kết quả đoạn chương trình sau:

```
internal class Cha
{
    protected string name;
    0 references
    public string Name{...}
    0 references
    public Cha(){...}
    2 references
    public Cha(string name){...}

    1 reference
    public void Display()
    {
        Console.Write("Display lop cha");
    }
```

```
internal class LopCon : Cha
{
    protected string info;
    0 references
    public string Info{...}
    0 references
    public LopCon(){...}
    2 references
    public LopCon(String info){...}
    0 references
    public LopCon(string name, string info) {...}

    0 references
    public void Display()
    {
        Console.WriteLine("Đay la display lop con");
    }
```

```
internal class Test
{
    0 references
    public static void Main(String[] args)
    {
        Cha cha = new Cha("Tran An");
        cha.Display();
        Cha cha1 = new LopCon("khong override");
        cha1.Display();

    }
}
```


Tính đa hình

```
internal class Cha
{
    protected string name;
    0 references
    public string Name{...}
    0 references
    public Cha(){...}
    2 references
    public Cha(string name){...}

    2 references
    public void Display(){...}
    1 reference
    public virtual void Display1()
    {
        Console.Write("Đây là display 1 lop cha"
    }
```

```
internal class LopCon : Cha
{
    protected string info;
    0 references
    public string Info{...}
    0 references
    public LopCon(){...}
    1 reference
    public LopCon(String info){...}
    0 references
    public LopCon(string name, string info) {...}

    0 references
    public void Display()
    {
        Console.WriteLine("Đay la display lop con");
    }
    1 reference
    public override void Display1()
    {

        Console.WriteLine("Display lop con 1");
    }
}
```

```
internal class Test
{
    0 references
    public static void Main(String[] args)
    {
        Cha cha = new Cha("Tran An");
        cha.Display1();
        Cha cha1 = new LopCon("khong override");
        cha1.Display1();

    }
}
```

Tính trừu tượng

- Mỗi lớp trừu tượng chứa ít nhất 1 phương thức trừu tượng
- Khai báo phương thức trừu tượng

```
<access modifier> abstract <type> name([th.s.]);
```

- Khai báo lớp trừu tượng

```
<access modifier> abstract class Name{...}
```

- Khai báo phương thức thực thi phương thức trừu tượng ở lớp con:

```
<access modifier> override <type> name{...}
```


Tính trừu tượng

- **Lưu ý:**
  - Một lớp thừa kế từ lớp trừu tượng phải hiện thực tất cả các phương thức trừu tượng hoặc lớp đó cũng phải là lớp trừu tượng.
  - Phương thức trừu tượng không được khai báo sử dụng từ khóa virtual. Vì bản thân từ khóa **abstract** đã bao hàm khái niệm **virtual**.
  - Phương thức trừu tượng không thể là phương thức tĩnh (static)
  - Lớp trừu tượng không tạo được **instance**
  - *abstract method không được để **private***

Ví dụ

```
internal abstract class Shape
{
    4 references
    public abstract void tinhDienTich();

}

internal class HinhTron: Shape

{
    2 references
    public override void tinhDienTich()
    {
        Console.WriteLine("Day la tinh dien tich hinh tron");
    }
}
```

```
public static void Main(String[] args)
{
    Shape shape = new Shape();// error
    HinhChuNhat hcn = new HinhChuNhat();
    hcn.tinhDienTich();
    HinhTron ht= new HinhTron();
    ht.tinhDienTich();

}

internal class HinhChuNhat:Shape
{
        2 references
        public override void tinhDienTich()
    {
        Console.WriteLine("day la tinh dien tich hc
    }
}
```


Bài tập

Mỗi môn học đều có các thông tin là mã môn, tên môn, số tín chỉ và khoa phụ trách, điểm. Hãy xây dựng chương trình tính học phí, tính điểm tích lũy cho một sinh viên trong một học kỳ biết rằng mỗi sinh viên trong một học kỳ có thể học rất nhiều môn khác nhau.

Các môn học trong trường được chia làm 3 loại như sau:

- Môn lý thuyết: Môn lý thuyết là môn có 3 cột điểm, điểm tiểu luận, điểm giữa kỳ và điểm cuối kỳ với hệ số là 0.2, 0.3 và 0.5. Học phí của môn lý thuyết sẽ được tính với đơn giá là 250000/ 1 tín chỉ.

Bài tập

- Môn thực hành: có 4 cột điểm kiểm tra và điểm trung bình sẽ bằng trung bình cộng của bốn bài kiểm tra. Môn thực hành cũng được tính với đơn giá 350000/ 1 tín chỉ cộng với 100000 tiền cơ sở vật chất.
- Môn đồ án: sẽ có điểm của GVHD và GVPB. Điểm của môn học là trung bình cộng của điểm GVHD và GVPB với điểm của GVHD tính hệ số 2. Mỗi môn đồ án đều có học phí là 2000000.

Interface

- Cú pháp khai báo interface:

```
public interface <Ten_interface>
{
    //noi dung interface
}
```

- Cú pháp khai báo lớp thực thi interface

```
public class <TenLopCon>:<LopCha>, interfce1, interface2,..
{
    //cai dat lop
}
```

Interface

- Một interface có thể mở rộng từ các interface khác:

```
public interface <Ten_interface>:<interface1>,<interface2>,...
{
    //khai bao interface
}
```

- Phương thức override các phương thức từ interface:

public <type> name([tham số]){...}

Interface

- Lưu ý:
  - Phương thức implement các phương thức của các interface ở các lớp không có từ khóa **override**
  - Phương thức trừ tượng: có thể có hoặc kng có từ khóa **abstract**
  - Có thể khai báo body cho các PT của interface
  - Dữ liệu của interface: static hoặc const

Ví dụ

```
internal interface interface1
{
        2 references
        public void tinhToan();
}
internal class HinhChuNhat : Shape, interface1
{
    2 references
    public override void tinhDienTich()
    {
        Console.WriteLine("day la tinh dien tich hcn");
    }
    2 references
    public void tinhToan()
    {
        Console.WriteLine("Phuong thuc interface");
    }
}
```

```
public static void Main(String[] args)
{
    HinhChuNhat hcn = new HinhChuNhat();
    hcn.tinhDienTich();
    hcn.tinhToan();
}
```


Bài tập

Trong một ứng dụng có quản lý bất động sản của công ty ABC có 4 đối tượng là: đất trống, nhà ở, biệt thự và khách sạn. Biết rằng tất cả các đối tượng này đều có các thông tin sau: mã số, chiều dài, chiều rộng và phương thức tính giá trị. Biết rằng giá trị được tính như sau:

- Đất trống: giá bán = diện tích * 10000
- Nhà ở có thêm thông tin về số lầu: giá bán = diện tích *10000 + số lầu * 100000
- Khách sạn có thêm thông tin về số sao: giá bán = 100000 + số sao*50000

Bài tập

- Biệt thự: giá bán = diện tích *400000

Tuy nhiên trong 4 loại bất động sản trên thì có 2 loại sản phẩm khi mua khách hàng phải đóng thêm phí kinh doanh là biệt thự và khách sạn. Biết rằng phí kinh doanh sẽ được tính như sau:

- Biệt thự: Diện tích *1000
- Khách sạn: Chiều rộng*5000

Hãy xây dựng ứng dụng quản lý bất động sản.

Partial

- C# cho phép tách các class, method, interface thành nhiều file .cs bằng cách sử dụng từ khóa **partial.**
- Khi biên dịch chương trình sẽ có nhiệm vụ gom các file này với nhau để thực thi.

EmployeeProps.cs

```
public partial class Employee
{
    public int EmpId { get; set; }
    public string Name { get; set; }
}
```

EmployeeMethods.cs

```
public partial class Employee
{
    //constructor
    public Employee(int id, string name){
        this.EmpId = id;
        this.Name = name;
    }

    public void DisplayEmpInfo() {
        Console.WriteLine(this.EmpId + " " this.Name);
    }
}
```

Partial

- Quy tắc của các partial class:
  - Tất cả các lớp phải được đặt cùng namespce và assembly
  - Tất cả các phần phải có cùng quyền truy cập: private, public, ...
  - Nếu có phần nào khai báo abstract, seal, kiểu cơ sở thì tất cả các lớp phải khai báo cùng kiểu
  - Các phần khác nhau có các kiểu cơ sở khác nhau và lớp cuối cùng sẽ thừa kế tất cả các kiểu cơ sở
  - Từ khóa **partial** chỉ có thể xuất hiện ngay phía trước các từ khóa class, interface,...

Partial method

- Partial method: phải sử dụng từ khóa partial và phải trả về kiểu void
- Tham số của các PT partial có thể có in, ref, nhưng ko có out
- partial method không thể dạng virtual
- partial method có thể dạng static

EmployeeProps.cs

```
public partial class Employee
{
    public Employee() {
        GenerateEmpId();
    }
    public int EmpId { get; set; }
    public string Name { get; set; }

    partial void GenerateEmployeeId();
```

EmployeeMethods.cs

```
public partial class Employee
{
    partial void GenerateEmployeeId()
    {
        this.EmpId = random();
    }
}
```


https://www.tutorialsteacher.com/csharp/csharp-partial-class

Namespace

- Một namespace trong C# được thiết kế để phân nhóm toàn bộ các kiểu dữ liệu theo một cấu trúc phân cấp.
- Nhờ có namespace kiểu dữ liệu được quản lí tốt hơn và tránh được hiện tượng xung đột tên.
- Các tên được khai báo trong một namespace không xung đột với cùng tên đó nhưng được khai báo ở một namespace khác.

Namespace

- Khai báo

```
namespace tên_namespace
{
        // Khai báo code
}
```

Example: Namespace

```
namespace School
{
    class Student
    {

    }

    class Course
    {

    }
}
```

Namespace

- Để sử dụng các namespace sử dụng toán tử dot (.) sau tên namespace

Example: Refer a Class with Namespace

```
namespace CSharpTutorials
{
    class Program
    {
        static void Main(string[] args)
        {
            School.Student std = new School.Student();

            School.Course cs = new School.Course();
        }
    }
}
```


Namespace

- Sử dụng từ khóa **using** để khai báo sử dụng các lớp trong namespace.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.CompilerServices;
using System.Text;
using System.Threading.Tasks;

namespace LopThu3
{
    3 references
    partial class HinhChuNhat : Shape, interface1
    {
```