

BÀI GIẢNG LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

CHƯƠNG 3

LỚP VÀ ĐỐI TƯỢNG

TRẦN THỊ THU THẢO

BỘ MÔN TIN HỌC QUẢN LÝ, KHOA THỐNG KÊ – TIN HỌC

TRƯỜNG ĐẠI HỌC KINH TẾ, ĐẠI HỌC ĐÀ NẴNG

THAOTRAN@DUE.EDU.VN

NỘI DUNG

1. Lớp

2. Đối tượng

3. Phương thức

4. Phương thức khởi tạo

5. Phương thức hủy

LỚP (CLASS)

□ KHÁI NIỆM

Một **lớp (class)** trong lập trình hướng đối tượng đại diện cho tập hợp các đối tượng có cùng các đặc điểm, hành vi, phương thức hoạt động.

Trong đó:

- Đặc điểm của đối tượng được thể hiện ra bên ngoài là các thuộc tính (property) của lớp;
- Các hành vi hay phương thức hoạt động của đối tượng gọi là phương thức của lớp;
- Các thuộc tính và phương thức được gọi chung là thành viên của lớp.

LỚP (CLASS)

❑ KHAI BÁO LỚP:

- Khai báo một lớp bắt đầu bằng từ khóa **class** theo sau là tên của lớp, phần thân của lớp được đánh dấu bằng cặp dấu mở đóng ngoặc nhọn **{}**.

- **Cú pháp**

```
class <tên lớp>
```

```
{
```

```
    <Phạm vi truy cập> <Các thành phần của lớp>;
```

```
}
```

- **<tên lớp>** là tên do người dùng đặt và tuân theo quy tắc đặt tên
- **<Phạm vi truy cập>** bao gồm các từ khoá như **public**, **protected**, **private**, **static**,...
- **<Các thành phần của lớp>** bao gồm các biến, phương thức của lớp.

LỚP (CLASS)

❑ KHAI BÁO LỚP

■ Phạm vi truy cập

Mức độ truy cập	Ý nghĩa
public	Không hạn chế. Thành phần mang thuộc tính này có thể được truy cập ở bất kỳ nơi nào
private	Chỉ có các phương thức của lớp A mới được phép truy cập đến thành phần được đánh dấu private trong các lớp A.
protected	Chỉ có các phương thức của lớp A hoặc của lớp dẫn xuất từ A mới được phép truy cập đến thành phần được đánh dấu protected trong lớp A.
internal	Các thành viên internal trong lớp A được truy xuất trong các phương thức của bất kỳ lớp trong khối kết hợp (assembly) của A
protected internal	Tương đương với protected or internal

ĐỐI TƯỢNG

❑ TẠO ĐỐI TƯỢNG

- **Sử dụng từ khóa “new”**

Là một đối tượng trong thế giới thực, có thuộc tính và hành vi. Trong C#, đối tượng là một thể hiện của lớp. Sử dụng từ khóa **new** để khởi tạo một đối tượng của lớp.

- **Cú pháp khai báo đối tượng và cấp phát vùng nhớ cho đối tượng:**

<TênLớp> <TênBiếnĐốiTượng>;

TênBiếnĐốiTượng = new TênLớp(DanhSáchĐốiSố);

hoặc

TênLớp TênBiếnĐốiTượng = new TênLớp(DanhSáchĐốiSố);

ĐỐI TƯỢNG

❑ TẠO ĐỐI TƯỢNG

- Ví dụ

```
HìnhChuNhat h;  
h = new HìnhChuNhat();
```

Hoặc

```
HìnhChuNhat h = new HìnhChuNhat();
```

PHƯƠNG THỨC

□ KHÁI NIỆM VÀ CÚ PHÁP

- Phương thức (hàm thành viên) là một nhóm lệnh của C# cùng nhau thực hiện một tác vụ
- **Cú pháp khai báo một phương thức**
<mức độ truy cập> <kiểu dữ liệu thuộc tính> Tên phương thức(<tham số>)

```
{  
    // thân phương thức  
}
```

Ví dụ:

```
public float Chuvi()  
{  
    return (Dai+Rong)*2;  
}
```


PHƯƠNG THỨC

□ KHÁI NIỆM VÀ CÚ PHÁP

- Cú pháp gọi một phương thức của đối tượng

<Tên_Đối_Tượng>.<Tên_PhươngThức>(<các tham số nếu có>);

Ví dụ:

```
h.Nhap();  
h.Xuat();  
h.Chuvi();
```

PHƯƠNG THỨC

Ví dụ 1: Viết chương trình tính chu vi hình chữ nhật

```
class HìnhChuNhat  
{
```

HìnhChuNhat là tên của lớp

```
    public float Dai;  
    public float Rong;  
    public void Nhap()  
{
```

Dai, Rong là 2 thuộc tính

```
        Console.WriteLine("Nhap chieu dai hình chu nhật :");  
        Dai = float.Parse(Console.ReadLine());  
        Console.WriteLine("Nhap chieu rong hình chu nhật :");  
        Rong = float.Parse(Console.ReadLine());  
    }
```

Nhap(), Xuat(), Chuvi()
là phương thức

```
    public void Xuat()  
{
```

Kiểu trả về là **void** nên thân phương thức **không chứa lệnh return giá trị**

```
        Console.WriteLine("Hình chu nhật: Dai = {0}, Rong = {1}",  
        Dai, Rong);  
    }
```

```
    public float Chuvi()  
{
```

Kiểu trả về là **float (int/ double/..)** nên thân phương thức **phải chứa lệnh return theo đúng kiểu đã quy định**

```
        return (Dai+Rong)*2;  
    }
```

PHƯƠNG THỨC

Ví dụ 1: Viết chương trình tính chu vi hình chữ nhật

```
class Program
{
    static void Main(string[] args)
    {
        HìnhChuNhat h = new HìnhChuNhat();
        h.Nhap();
        h.Xuat();
        Console.WriteLine("Chu vi HCN: {0}", h.Chuvi());
        Console.ReadLine();
    }
}
```

Tạo đối tượng

Gọi phương thức của đối tượng

TERMINAL

PROBLEMS

OUTPUT

DEBUG CONSOLE

PS E:\OOP> cd ViduClass

PS E:\OOP\ViduClass> dotnet run "e:\OOP\ViduClass\Program.cs"

Nhap chieu dai hình chu nhật :10.2

Nhap chieu rong hình chu nhật :3.5

Hình chu nhật: Dai = 10.2, Rong = 3.5

Chu vi hình chu nhật: 27.4

PHƯƠNG THỨC

□ KHÁI NIỆM

▪ Phương thức khởi tạo (Constructor)

Đây là một phương thức đặc biệt của lớp. Nó được thực thi khi khởi tạo đối tượng.

Đặc điểm của phương thức khởi tạo:

- Có tên trùng với tên lớp.
- Không có kiểu trả về.
- Được tự động gọi khi 1 đối tượng thuộc lớp được khởi tạo.
- Nếu như bạn không khai báo bất kỳ phương thức khởi tạo nào thì hệ thống sẽ tự tạo ra phương thức khởi tạo mặc định không đối số và không có nội dung gì.
- Có thể có nhiều constructor bên trong 1 lớp.

PHƯƠNG THỨC KHỞI TẠO

□ KHÁI NIỆM

▪ Phương thức khởi tạo (Constructor)

Có 2 loại phương thức khởi tạo:

• Phương thức khởi tạo **không đối số**:

- Là phương thức khởi tạo không có bất kỳ tham số truyền vào nào.
- Thường dùng để khởi tạo các giá trị mặc định cho các thuộc tính bên trong class khi khởi tạo đối tượng (giá trị mặc định này do người lập trình quyết định).

PHƯƠNG THỨC KHỞI TẠO

❑ CÚ PHÁP

▪ Phương thức khởi tạo (Constructor)

• Phương thức khởi tạo có đối số:

- Là phương thức khởi tạo có tham số truyền vào. Và khi khởi tạo đối tượng để phương thức này được gọi ta cần truyền đầy đủ các tham số.
- Thường dùng để khởi tạo các giá trị cho các thuộc tính bên trong class khi khởi tạo đối tượng (các giá trị này do người khởi tạo đối tượng truyền vào).
- Cú pháp:

TênBiếnĐốiTượng = *new* *TênLớp(DanhSáchĐốiSố);*

PHƯƠNG THỨC KHỞI TẠO

□ VÍ DỤ

- **Ví dụ 2:** Viết chương trình xây dựng một lớp *Time*, trong đó có một phương thức tạo lập nhận tham số có kiểu *DateTime* (kiểu xây dựng sẵn của trình biên dịch) làm tham số khởi tạo gán cho các thành phần dữ liệu của đối tượng thuộc lớp *Time*

PHƯƠNG THỨC KHỞI TẠO

❑ VÍ DỤ

▪ Ví dụ 2:

```
using System;
namespace C3Vd2
{
    public class Time
    {
        // private member variables
        int Year;
        int Month;
        int Date;
        int Hour;
        int Minute;
        int Second = 30;
        public void DisplayCurrentTime( )
        {
            Console.WriteLine("Current time is: {0}/{1}/{2} {3}:{4}:{5}",Date,
Month, Year, Hour, Minute, Second);
        }
    }
}
```


PHƯƠNG THỨC KHỞI TẠO

❑ VÍ DỤ

```
public Time(System.DateTime dt)
{
```

Phương thức khởi tạo có đối số (constructor)

```
    Console.WriteLine("Ham constructor tu dong duoc goi!");
    Year = dt.Year;
    Month = dt.Month;
    Date = dt.Day;
    Hour = dt.Hour;
    Minute = dt.Minute;
```

Khởi tạo các giá trị cho các thuộc tính bên trong class

```
    }
}
class DateTimeConstructorApp
{
```

```
    static void Main()
    {
```

```
        System.DateTime currentTime = System.DateTime.Now;
        Time t = new Time(currentTime);
        t.DisplayCurrentTime( );
        Console.ReadLine();
    }
```

Truyền giá trị vào main

```
    }
}
```

```
PS E:\OOP\C3Vd2> dotnet run
Ham constructor tu dong duoc goi!
Current time is: 22/5/2022 14:55:30
```

PHƯƠNG THỨC KHỞI TẠO

❑ VÍ DỤ

- **Ví dụ 3:** Viết chương trình mô tả số đo của học sinh.

```
using System;
namespace C3Vd3
{
    class Hocsinh
    {
        public double Weight;
        public double Height;
        public void Info()
        {
            Console.WriteLine(" Học sinh có cân nặng là: "
+ Height + " và chiều cao là: " + Weight);
        }
    }
}
```

PHƯƠNG THỨC KHỞI TẠO

❑ VÍ DỤ

- **Ví dụ 3:** Viết chương trình mô tả số đo của học sinh.

```
public Hocsinh() //Constructor không có tham số
{
    Weight = 40;
    Height = 150;
}

public Hocsinh(int w, int h) //Constructor có tham số
{
    Weight = w;
    Height = h;
}
}
```

PHƯƠNG THỨC KHỞI TẠO

❑ VÍ DỤ

- **Ví dụ 3:** Viết chương trình mô tả số đo của học sinh.

```
class SodoHocsinh
{
    static void Main()
    {
        Hocsinh hs1 = new Hocsinh();
        hs1.Info();

        Hocsinh hs2 = new Hocsinh(48, 160);
        hs2.Info();
    }
}
```

```
PS E:\OOP\C3Vd3> dotnet run
```

```
Hoc sinh co can nang la: 150 va chieu cao la: 40
```

```
Hoc sinh co can nang la: 160 va chieu cao la: 48
```

PHƯƠNG THỨC HỦY

□ KHÁI NIỆM

▪ Phương thức hủy (destructor)

Đây là phương thức đặc biệt được gọi đến trước khi 1 đối tượng bị thu hồi.

Đặc điểm của phương thức hủy:

- Có tên trùng với tên lớp nhưng để phân biệt với constructor thì ta thêm dấu “~” vào trước tên lớp.
- Không có kiểu trả về.
- Được tự động gọi khi 1 đối tượng thuộc lớp kết thúc “vòng đời” của nó thông qua bộ thu dọn rác tự động GC (Garbage Collection).
- Nếu bạn không khai báo destructor thì C# sẽ tự động tạo ra 1 destructor mặc định và không có nội dung gì.
- Chỉ có 1 destructor duy nhất trong 1 lớp.


PHƯƠNG THỨC HỦY

❑ VÍ DỤ

▪ Phương thức hủy (destructor)

Ví dụ:

```
class Hocsinh
{
    public double Weight;
    public double Height;
    public void Info()
    public Hocsinh(int w, int h) //Constructor có tham số
    {
        Weight = w;
        Height = h;
    }
    ~Hocsinh()
    {
        Console.WriteLine("Doi tuong Hocsinh da bi huy.");
    }
}
```



BÀI TẬP KẾT THÚC CHƯƠNG 3

❑ **Bài 1:** Xây dựng lớp Sach gồm:

- Thuộc tính: Tensach, Tacgia, NamXB, Soluong
- Phương thức:
 - Hàm Khởi tạo, Hàm hủy
 - Hàm Nhập, Xuất

Xuất ra thông tin các loại sách đã nhập ra màn hình

❑ **Bài 2:** Xây dựng lớp Meo gồm:

- Thuộc tính: Ten, Giong, Gioitinh, Tuoi
- Phương thức:
 - Hàm Khởi tạo, Hàm hủy
 - Hàm Nhập, Xuất, Đếm

a) Xuất ra danh sách thông tin mèo đã nhập ra màn hình

b) Đếm số lượng mèo có trong danh sách

BÀI TẬP KẾT THÚC CHƯƠNG 3

❑ **Bài 3:** Xây dựng lớp **Organization** gồm:

- Thuộc tính: **Name, Country, City, Address, Telephone** là Tên Doanh nghiệp, Quốc gia, Thành phố, Địa chỉ, Số điện thoại
- Phương thức:
 - Hàm Khởi tạo, Hàm hủy
 - Hàm Nhập, Xuất – Để nhập và xuất thông tin

Nhập và xác nhận các thông tin đầu vào trong đó yêu cầu:

- Thông tin đầu vào các thuộc tính không được để trống
- Thông tin về Số điện thoại phải đảm bảo đủ 10 chữ số và không được để trống

Xuất ra màn hình thông tin của Doanh nghiệp nếu tất các thông tin nhập vào là hợp lệ.

Trường hợp thông tin không hợp lệ thì thông báo “Nhập thông tin lỗi” và Dừng chương trình.

BÀI TẬP KẾT THÚC CHƯƠNG 3

❑ **Bài 4:** Xây dựng lớp **Triangle** gồm:

- Thuộc tính: **a, b, c** (kiểu int) là 3 cạnh của một tam giác
- Phương thức:
 - Hàm Khởi tạo, Hàm hủy
 - Hàm Nhập, Xuất – Để nhập và xuất thông tin
 - Hàm Tính để tính chu vi và diện tích hình tam giác

Tạo một **Menu** để thực hiện lặp lại các chức năng cho đến khi thoát:

Bấm 1: Nhập các cạnh a,b,c của tam giác

Bấm 2: Tính chu vi và diện tích hình tam giác

Bấm 3: Xuất ra các giá trị a,b,c.

Trường hợp các cạnh tạo được một tam giác thì Xuất ra giá trị Chu vi và Diện tích của tam giác. Trường hợp không tạo được tam giác thì xuất ra màn hình “**Không hợp lệ**”

Bấm 0: Thoát khỏi chương trình

BÀI GIẢNG LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

CHƯƠNG 4 ĐÓNG GÓI

TRẦN THỊ THU THẢO

BỘ MÔN TIN HỌC QUẢN LÝ, KHOA THỐNG KÊ – TIN HỌC
TRƯỜNG ĐẠI HỌC KINH TẾ, ĐẠI HỌC ĐÀ NẴNG
THAOTRAN@DUE.EDU.VN

NỘI DUNG

1. Khái niệm Đóng gói
2. Các chỉ thị truy cập
3. Phương thức truy vấn và phương thức cập nhật
4. Thành viên tĩnh của lớp
5. Tham số của phương thức
6. Tham chiếu this
7. Chỉ mục indexer

ĐÓNG GÓI

□ KHÁI NIỆM

Tính đóng gói là khả năng che dấu thông tin của đối tượng với môi trường bên ngoài. Tính chất này giúp đảm bảo tính toàn vẹn và bảo mật của đối tượng.

Tính đóng gói được thể hiện thông qua các từ khóa xác định phạm vi sử dụng:

- **public**: cho phép truy cập khi đứng ở ngoài lớp
- **private**: chỉ cho phép truy cập khi đứng ở trong lớp
- **protected**: chỉ cho phép truy cập khi đứng ở trong lớp hoặc ở lớp con
- **internal**: được truy cập từ các lớp nằm trong cùng một assembly

CHỈ THỊ TRUY CẬP public

□ KHÁI NIỆM

Chỉ thị truy cập public cho phép một lớp đưa ra các thuộc tính và các phương thức thành viên của nó cho các phương thức và đối tượng khác. Bất kỳ thành viên nào cũng có thể được truy cập từ bên ngoài lớp

CHỈ THỊ TRUY CẬP public

Ví dụ 1:

Xây dựng một lớp Hình chữ nhật gồm:

Thuộc tính: Chiều dài và chiều rộng.

Phương thức:

- Hàm khởi tạo
- Hàm nhập, xuất
- Hàm tính diện tích (Tinhdientich)

Tính diện tích hình chữ nhật rồi in kết quả lên màn hình

CHỈ THỊ TRUY CẬP public

Ví dụ 1:

```
class HìnhChuNhat
```

```
{
```

```
    public double Dai { get; set; }
```

```
    public double Rong { get; set; }
```

```
    public double Tinhdientich()
```

```
    {
```

```
        return Dai * Rong;
```

```
    }
```

```
    public void Xuat()
```

```
    {
```

```
        Console.WriteLine("Chieu dai: {0}", Dai);
```

```
        Console.WriteLine("Chieu rong: {0}", Rong);
```

```
        Console.WriteLine("Dien tich HCN: {0}", Tinhdientich());
```

```
    }
```

```
}
```

Các thuộc tính đều mang phạm vi là public

Các phương thức mang phạm vi là public

CHỈ THỊ TRUY CẬP public

Ví dụ 1:

```
class Program
{
    static void Main(string[] args)
    {
        HìnhChuNhat hcn = new HìnhChuNhat();
        hcn.Dai = 4.5;
        hcn.Rong = 3.5;
        hcn.Xuat();
        Console.ReadLine();
    }
}
```

Thuộc tính Dai, Rong, phương thức Xuat(), Tinhdientich() có thể truy cập từ phương thức Main() bằng các tạo một thể hiện của lớp Hìnhchunhat

Chieu dai: 4.5
Chieu rong: 3.5
Dien tich HCN: 15.75

CHỈ THỊ TRUY CẬP **private**

□ KHÁI NIỆM

Chỉ thị truy cập **private** cho phép lớp ẩn các trường và các phương thức thành viên khỏi các phương thức và đối tượng khác.

Chỉ các thành viên trong cùng một lớp mới có thể truy cập các thành viên **private** của nó. Ngay cả thể hiện của lớp cũng không thể truy cập các thành viên **private** của nó

Nếu một trường hoặc phương thức không có chỉ thị truy cập thì nó sẽ được gán chỉ thị truy cập mặc định là **private**

CHỈ THỊ TRUY CẬP **private**

Ví dụ 2:

```
namespace C4Vd2
{
    class Hinhchunhat
    {
        private double _dai;
        private double _rong;
        public Hinhchunhat(double dai, double rong)
        {
            _dai = dai;
            _rong = rong;
        }
        public double Tinhdientich()
        {
            return _dai * _rong;
        }
    }
}
```

Các thuộc tính đều mang phạm vi là **private** → Không thể truy cập từ phương thức **Main()**

Các phương thức mang phạm vi là **public** → có thể truy cập từ phương thức **Main()** bằng các sử dụng thể hiện **hcn** của lớp **Hinhchunhat**

CHỈ THỊ TRUY CẬP private

Ví dụ 2:

```
public void Xuat()  
{  
    Console.WriteLine("Chieu Dai: {0}", _dai);  
    Console.WriteLine("Chieu Rong: {0}", _rong);  
    Console.WriteLine("Dien tich HCN: {0}", Tinhdientich());  
}  
}  
class Program  
{  
    static void Main(string[] args)  
    {  
        Hinhchunhat hcn = new Hinhchunhat(5.5, 3.5);  
        hcn.Xuat();  
        Console.ReadLine();  
    }  
}
```

```
Chieu Dai: 5.5  
Chieu Rong: 3.5  
Dien tich HCN: 19.25
```

CHỈ THỊ TRUY CẬP **protected**

❑ KHÁI NIỆM

Chỉ thị truy cập **protected** cho phép lớp ẩn các trường và các phương thức thành viên khỏi các phương thức và đối tượng khác. Nó chỉ cho phép một lớp con truy cập các trường và các phương thức thành viên này của lớp cha (hoặc lớp cơ sở)

Các thành viên trong class cha và class con mới có thể truy cập các thành viên **protected** của lớp cha. Ngay cả thể hiện của lớp con hay lớp cha cũng không thể truy cập các thành viên của **protected** class cha

Chỉ thị truy cập **protected** tương tự chỉ thị truy cập **private**, chỉ khác là nó cho phép class con truy cập các trường và phương thức thành viên của class cha có chỉ thị **protected**.

CHỈ THỊ TRUY CẬP **protected**

Ví dụ 3:

```
namespace C4Vd3 {  
    class Shape {  
        protected int _rong;  
        protected int _dai;  
        public void SetRong(int rong)  
        {  
            _rong = rong;  
        }  
        public void SetDai(int dai)  
        {  
            _dai = dai;  
        }  
        public void Xuat()  
        {  
            Console.WriteLine("rong = {0}, dai = {1}", _rong, _dai);  
        }  
    }  
}
```

Các thuộc tính đều mang phạm vi là **protected**, nên không thể được truy cập từ phương thức **Main()**

Các phương thức **SetRong**, **SetDai** và **Xuat()** của class **Shape** là **public** nên chúng được class hcn kế thừa và có thể truy cập ở phương thức **Main()**

CHỈ THỊ TRUY CẬP protected

Ví dụ 3:

```
class hcnTester
{
    static void Main(string[] args)
    {
        Shape shape = new Shape();
        shape.SetRong(4);
        shape.SetDai(6);
        shape.Xuat();
    }
}
```

CHỈ THỊ TRUY CẬP `protected`

Ví dụ 3:

```
class hcn: Shape
{
    public int TinhDientich()
    {
        return (_rong * _dai);
    }
}

class hcnTester {
    static void Main(string[] args)
    {
        hcn hcn = new hcn();
        hcn.SetRong(5);
        hcn.SetDai(7);
        hcn.Xuat();
        Console.WriteLine("Total area: {0}", hcn.TinhDientich());
        Console.ReadKey();
    }
}
```

Lớp `hcn` kế thừa lớp `Shape` nên có thể truy cập được trường `_dai` và `_rong` để tính diện tích trong phương thức `TinhDientich()`

```
rong = 4, dai = 6
rong = 5, dai = 7
Total area: 35
```

CHỈ THỊ TRUY CẬP **internal**

❑ KHÁI NIỆM

Chỉ thị truy cập **internal** cho phép lớp đưa ra các thuộc tính và phương thức thành viên của nó cho các phương thức và đối tượng khác trong cùng assembly

Chỉ thị truy cập **internal** tương tự như chỉ thị **public**, nó chỉ khác ở chỗ chỉ thị giới hạn phạm vi truy cập **internal** trong cùng một assembly còn chỉ thị **public** thì không có bất kỳ giới hạn nào

CHỈ THỊ TRUY CẬP **internal**

Ví dụ 3:

```
namespace C4Vd3 {  
    public class HìnhChuNhat  
    {  
        internal double Dai;  
        internal double Rong;  
        double TinhDientich() // private method  
        {  
            return Dai * Rong;  
        }  
        public void Xuat()  
        {  
            Console.WriteLine("Dai: {0}", Dai);  
            Console.WriteLine("Rong: {0}", Rong);  
            Console.WriteLine("Dien tich: {0}", TinhDientich());  
        }  
    }  
}
```

CHỈ THỊ TRUY CẬP **internal**

Ví dụ 3:

```
class Program
{
    static void Main(string[] args)
    {
        HìnhChuNhat hcn = new HìnhChuNhat();
        hcn.Dai = 4.5;
        hcn.Rong = 3.5;
        hcn.Xuat();
        Console.ReadLine();
    }
}
```

Dai: 4.5
Rong: 3.5
Dien tích: 15.75

PHƯƠNG THỨC TRUY VẤN VÀ CẬP NHẬT

□ KHÁI NIỆM

▪ Phương thức truy vấn

Phương thức truy vấn là phương thức giúp người dùng có thể xem được dữ liệu của 1 thuộc tính nào đó. Cụ thể, phương thức truy vấn chỉ cần trả về giá trị của thuộc tính tương ứng là đủ.

▪ Phương thức cập nhật

Phương thức cập nhật là phương thức giúp người dùng có thể thay đổi giá trị cho 1 thuộc tính nào đó. Cụ thể, phương thức cập nhật chỉ cần thực hiện cập nhật giá trị mới cho thuộc tính tương ứng (có thể kiểm tra tính đúng đắn của dữ liệu trước khi truyền vào).

PHƯƠNG THỨC TRUY VẤN VÀ CẬP NHẬT

□ QUY ƯỚC

- **Một số quy ước nhỏ về cách đặt tên của phương thức truy vấn và phương thức cập nhật**
 - Những phương thức truy vấn nên bắt đầu bằng từ khoá **get** và kèm theo sau là tên thuộc tính tương ứng.
Ví dụ: `getHoTen()`, `getDiemToan()`, ...
 - Những phương thức cập nhật nên bắt đầu bằng từ khoá **set** và kèm theo sau là tên thuộc tính tương ứng.
Ví dụ: `setDiemToan()`, `setHoTen()`, ...
- Nếu thuộc tính kiểu luận lý (**bool**) thì tên phương thức truy vấn nên bắt đầu bằng từ khoá **is** và kèm theo sau là tên thuộc tính tương ứng.
- Phương thức truy vấn sẽ có kiểu trả về trùng với kiểu dữ liệu của thuộc tính tương ứng và không có tham số truyền vào.
- Phương thức cập nhật sẽ có kiểu trả về là **void** và có 1 tham số truyền vào có kiểu dữ liệu trùng với kiểu dữ liệu của thuộc tính tương ứng.

PHƯƠNG THỨC TRUY VẤN VÀ CẬP NHẬT

Ví dụ 4: Viết chương trình quản lý điểm CSLT của Sinh viên

```
class Sinhvien
```

```
{
```

```
    private string MASV;  
    private double DiemCSLT;
```

```
    public string getMASV()
```

```
{
```

```
        return MASV;
```

```
}
```

```
    public void setDiemCSLT(int diemcslt)
```

```
{
```

```
        DiemCSLT = diemcslt;
```

```
}
```

```
}
```

Các thuộc tính đều mang phạm vi là private

Phương thức truy vấn giá trị thuộc tính MASV

Phương thức cập nhật giá trị thuộc tính DiemCSLT.

Vì thế phương thức có 1 tham số truyền vào kiểu double trùng với kiểu của DiemCSLT

PHƯƠNG THỨC TRUY VẤN VÀ CẬP NHẬT

❑ Ví dụ 4:

```
class Sinhvien
{
    public void ShowInfo()
    {
        MASV = "1234";
        Console.WriteLine("SV Ma so {0} co diem CSLT la: {1} ", MASV,
DiemCSLT);
    }
}
class DiemHP
{
    static void Main(string[] args)
    {
        Sinhvien SV1 = new Sinhvien();
        SV1.getMASV();
        SV1.setDiemCSLT(8);
        SV1.ShowInfo();
    }
}
```

SV Ma so 1234 co diem CSLT la: 8

□ TỪ KHÓA **get** VÀ **set**

- Trong C#, phương thức truy xuất và phương thức cập nhật đã được nâng cấp lên thành 1 cấu trúc mới ngắn gọn hơn và tiện dụng hơn đó là **property**
- Sử dụng property giúp ta có thể thao tác dữ liệu tự nhiên hơn nhưng vẫn đảm bảo tính đóng gói của lập trình hướng đối tượng.

get & set

□ TỪ KHÓA **get** VÀ **set**

▪ CÚ PHÁP:

```
<kiểu dữ liệu> <tên property>
{
    get { return <tên thuộc tính>; }
    set { <tên thuộc tính> = value; }
}
```

Trong đó:

- **<kiểu dữ liệu>** là kiểu dữ liệu của property. Thường sẽ trùng với kiểu dữ liệu của thuộc tính **private** tương ứng bên trong lớp.
- **<tên property>** là tên do người dùng đặt và tuân theo quy tắc đặt tên.
- **get, set, value** là từ khoá có ý nghĩa:
 - Từ khoá **get** tương đương với phương thức truy vấn.
 - Từ khoá **set** tương đương với phương thức cập nhật.
 - Từ khoá **value** đại diện cho giá trị mà người gán vào property (tương đương với tham số truyền vào của phương thức cập nhật).
- **<tên thuộc tính>** là tên thuộc tính thực sự bên trong lớp.

get & set

□ TỪ KHÓA **get** VÀ **set**

- Trong C#, phương thức truy xuất và phương thức cập nhật đã được nâng cấp lên thành 1 cấu trúc mới ngắn gọn hơn và tiện dụng hơn đó là **property**
- Sử dụng property giúp ta có thể thao tác dữ liệu tự nhiên hơn nhưng vẫn đảm bảo tính đóng gói của lập trình hướng đối tượng.

get & set

□ TỪ KHÓA get VÀ set

- **Ví dụ 4:** Viết chương trình quản lý điểm CSLT của sinh viên

```
class Sinhvien
{
    private string MASV;
    private double DiemCSLT;

    public double diemcslt
    {
        get {return DiemCSLT;}
        set {DiemCSLT = value;}
    }
    public string masv
    {
        get {return MASV;}
        set {MASV = value;}
    }
}
```

□ TỪ KHÓA get VÀ set

▪ Ví dụ 4:

```
class DiemHP
{
    static void Main(string[] args)
    {
        Sinhvien SV1 = new Sinhvien();
        SV1.masv = "5678";
        SV1.diemcslt = 9;
        Console.WriteLine("SV Ma so {0} co diem CSLT la: {1} ",
SV1.masv, SV1.diemcslt);
    }
}
```

SV Ma so 5678 co diem CSLT la: 9

THÀNH VIÊN TĨNH CỦA LỚP

□ KHÁI NIỆM

- Thuộc tính hoặc phương thức trở thành dạng tĩnh nếu có từ khóa `static` trước phần khai báo thuộc tính hoặc phương thức. Nó là thành viên của lớp chứ không phải của đối tượng.
- Thành viên tĩnh được khởi tạo khi khai báo lớp. Thành viên tĩnh được sử dụng để lưu trữ và chia sẻ giá trị dùng chung giữa tất cả đối tượng được tạo từ lớp.

THÀNH VIÊN TĨNH CỦA LỚP

- **Đặc điểm của thành viên tĩnh:**
 - Được khởi tạo **1 lần duy nhất** ngay khi biên dịch chương trình.
 - Có thể dùng chung cho mọi đối tượng.
 - **Được gọi thông qua tên lớp.**
 - Được huỷ khi kết thúc chương trình.
- **Có 4 loại thành viên tĩnh chính:**
 - Biến tĩnh (static variable).
 - Phương thức tĩnh (static method).
 - Lớp tĩnh (static class).
 - Phương thức khởi tạo tĩnh (static constructor).

THÀNH VIÊN TĨNH CỦA LỚP

- **Biến tĩnh:**
 - **Cú pháp:**

<phạm vi truy cập> static <kiểu dữ liệu> <tên biến> = <giá trị khởi tạo>;

Trong đó:

- **<phạm vi truy cập>** là các phạm vi truy cập của biến
- **static** là từ khoá để khai báo thành viên tĩnh.
- **<kiểu dữ liệu>** là kiểu dữ liệu của biến
- **<tên biến>** là tên biến do người dùng đặt và tuân thủ các quy tắc đặt tên biến
- **<giá trị khởi tạo>** là giá trị ban đầu mà biến tĩnh này chứa. Nếu bạn không khai báo giá trị này thì C# thì tự gán giá trị mặc định và đưa ra 1 cảnh báo khi bạn biên dịch chương trình.

THÀNH VIÊN TĨNH CỦA LỚP

- **Ví dụ 5:** Quản lý số lượng học sinh đang có (Giả sử 1 đối tượng được tạo ra là 1 học sinh)

```
class Hocsinh
{
    private string ten;
    public string TEN
    {
        get {return ten;}
        set {ten = value;}
    }
    public static int Count = 0;
    public Hocsinh()
    {
        Console.WriteLine("Nhap ho ten hoc sinh: ");
        ten = Console.ReadLine();
        Count++;
    }
}
```

THÀNH VIÊN TĨNH CỦA LỚP

▪ Biến tĩnh:

- **Ví dụ 5:** Quản lý số lượng học sinh đang có (Giả sử 1 đối tượng được tạo ra là 1 học sinh)

```
class SodoHocsinh
{
    static void Main()
    {
        Console.WriteLine("So luong hoc sinh ban dau:
"+Hocsinh.Count);
        Hocsinh hs1 = new Hocsinh();
        Console.WriteLine("So luong hoc sinh ban dau:
"+Hocsinh.Count);
        Hocsinh hs2 = new Hocsinh();
        Console.WriteLine("So luong hoc sinh ban dau:
"+Hocsinh.Count);
    }
}
```

```
So luong hoc sinh ban dau: 0
Nhap ho ten hoc sinh: Ha
So luong hoc sinh ban dau: 1
Nhap ho ten hoc sinh: Van
So luong hoc sinh ban dau: 2
```


THÀNH VIÊN TĨNH CỦA LỚP

- Phương thức tĩnh:
 - Cú pháp:

```
<phạm vi truy cập> static <kiểu trả về> <tên phương thức>
{
    // nội dung phương thức
}
```

Trong đó:

- **<phạm vi truy cập>** là các phạm vi truy cập của phương thức
- **static** là từ khoá để khai báo thành viên tĩnh.
- **<kiểu trả về>** là kiểu trả về của phương thức
- **<tên phương thức>** là tên biến do người dùng đặt và tuân thủ các quy tắc đặt tên

THÀNH VIÊN TĨNH CỦA LỚP

- **Phương thức tĩnh:**

- **Hàm tĩnh:** được sử dụng với 2 mục đích chính:
- Hàm tĩnh là 1 hàm 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 để sử dụng lại.

THÀNH VIÊN TĨNH CỦA LỚP

- Ví dụ 6: Viết một Lớp *Tinhtoan* tính lũy thừa của một số nguyên

```
class Tinhtoan
{
    public static long LuyThua(int CoSo, int SoMu)
    {
        long KetQua = 1;
        for (int i = 0; i < SoMu; i++)
        {
            KetQua *= CoSo;
        }
        return KetQua;
    }
}
```

THÀNH VIÊN TĨNH CỦA LỚP

- Ví dụ 6: Viết một Lớp *Tinhtoan* tính lũy thừa của một số nguyên

```
class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine(TinhToan.LuyThua(3, 3));
    }
}
```

```
PS E:\OOP\C3Vd6> dotnet run
27
```

THÀNH VIÊN TĨNH CỦA LỚP

- **Lớp tĩnh:**

- **Cú pháp:**

```
<phạm vi truy cập> static class <tên lớp>
{
    // các thành phần của lớp
}
```

Trong đó:

- **<phạm vi truy cập>** là các phạm vi truy cập của lớp
- **static** là từ khoá để khai báo thành viên tĩnh.
- **class** là từ khoá để khai báo lớp.
- **<tên lớp>** là tên do người dùng đặt và tuân thủ các quy tắc đặt tên

THÀNH VIÊN TĨNH CỦA LỚP

▪ Lớp tĩnh:

• Đặc điểm

- Chỉ chứa các thành phần tĩnh (biến tĩnh, phương thức tĩnh).
- **Không thể** khai báo, khởi tạo 1 đối tượng thuộc lớp tĩnh.

Với 2 đặc điểm trên có thể thấy lớp tĩnh thường được dùng với mục đích khai báo 1 lớp tiện ích chứa các hàm tiện ích hoặc hằng số vì:

- Ràng buộc các thành phần bên trong lớp phải là **static**.
- Không cho phép tạo ra các đối tượng dư thừa làm lãng phí bộ nhớ.
- Mọi thứ đều được truy cập thông qua tên lớp.
- **Ví dụ: Trong ví dụ 6, có thể sử dụng lớp tĩnh, phương thức tĩnh để khai báo. Ví dụ lớp Math**

Lớp Math chứa:

- Các hằng số như **PI**, **E**.
- Các phương thức hỗ trợ tính toán như: **sin, cos, tan, sqrt, exp, ...**

THÀNH VIÊN TĨNH CỦA LỚP

- **Lớp tĩnh:**

- **Ví dụ: sử dụng lớp Math tính lũy thừa**

```
Console.WriteLine("Tính luy thừa: ");  
Console.WriteLine(Math.Pow(3,3));
```

Tính luy thừa:
27

THAM SỐ CỦA PHƯƠNG THỨC

- Ví dụ 7: Xây dựng lớp Hoanvi để hoán vị 2 số a và b

```
class Hoanvi
{
    static void Hoan_vi(int a, int b)
    {
        int temp=a;
        a=b;
        b=temp;
        Console.WriteLine("Trong phuong thuc Hoan_vi:
a={0}, b={1}",a,b);
    }
}
```


THAM SỐ CỦA PHƯƠNG THỨC

- Ví dụ 7: Xây dựng lớp Hoanvi để hoán vị 2 số a và b

```
static void Main(string[] args)
{
    int x=3; int y=4;
    Console.WriteLine("Truoc khi gọi phương thức  
Hoan_vị: x={0}, y={1}",x,y);
    Hoan_vị(x,y);
    Console.WriteLine("Sau khi gọi phương thức  
Hoan_vị: x={0}, y={1}",x,y);
}
```

THAM SỐ CỦA PHƯƠNG THỨC

- Truyền tham trị bằng tham số kiểu giá trị
Ở ví dụ 7, ở lớp `Hoanvi`, `a`, `b` là hai tham số dạng tham trị của phương thức `Hoan_vi`, nên mọi sự thay đổi chỉ diễn ra trong thân phương thức này mà không ảnh hưởng đến đối số `x`, `y` được truyền vào.

Khi chạy chương trình, ta nhận được kết quả:

```
Truoc khi gọi phương thức Hoan_vi: x=3, y=4  
Trong phương thức Hoan_vi: a=4, b=3  
Sau khi gọi phương thức Hoan_vi: x=3, y=4
```

THAM SỐ CỦA PHƯƠNG THỨC

- Truyền tham chiếu bằng tham số kiểu giá trị với từ khóa **ref** hoặc **out**

Để phương thức **Hoan_vi** cho ra kết quả như mong muốn ta phải sửa lại tham số a, b theo kiểu tham chiếu như sau:

```
static void Hoan_vi(ref int a, ref int b)
```

Khi đó ta gọi phương thức **Hoan_vi** với hai đối số x, y theo cú pháp:

```
Hoan_vi(ref x, ref y);
```

Trước khi gọi phương thức **Hoan_vi**: x=3, y=4

Trong phương thức **Hoan_vi**: a=4, b=3

Sau khi gọi phương thức **Hoan_vi**: x=4, y=3

THAM CHIẾU **this**

Khi một đối tượng đang thực thi một phương thức của thể hiện (không phải là phương thức tĩnh), tham chiếu **this** tự động trỏ đến đối tượng này. Mọi phương thức của đối tượng đều có thể tham chiếu đến các thành phần của đối tượng thông qua tham chiếu **this**. Có 3 trường hợp sử dụng tham chiếu **this**:

- Tránh xung đột tên khi tham số của phương thức trùng tên với tên biến dữ liệu của đối tượng.
- Dùng để truyền đối tượng hiện tại làm tham số cho một phương thức khác (chẳng hạn gọi đệ qui)
- Dùng với mục đích chỉ mục.

THAM CHIẾU **this**

Ví dụ 8: Dùng tham chiếu **this** với mục đích tránh xung đột tên của tham số với tên biến dữ liệu của đối tượng.

```
class People
{
    int old; string name; double height;
    public People(int old, string name, double height)
    {
        Console.WriteLine("\n---Goi ham xay dung co 3  
tham so---");
        this.old = old;
        this.name = name;
        this.height = height;
    }
}
```

THAM CHIẾU **this**

Ví dụ 8: Xây dựng lớp People quản lý sức khỏe

```
class People
```

```
{  
    int old; string name; double height;  
    public People(string name, int old, double height)  
    {  
        this.name = name;  
        this.old = old;  
        this.height = height;  
    }  
    public void Show()  
    {  
        Console.WriteLine("Name: " + name + "\nOld: " +  
old + "\nHeight: " + height);  
    }  
}
```

Gọi hàm xây dựng với 3 tham số.
Dùng tham chiếu **this** với mục đích
tránh xung đột tên của tham số với
tên biến dữ liệu của đối tượng

THAM CHIẾU **this**

Ví dụ 8: Xây dựng bài toán quản lý sức khỏe con người với các thuộc tính: Name, Old, Height
Xuất ra màn hình thông tin của người đó.

```
class Program
{
    static void Main(string[] args)
    {
        People p1 = new People("Nguyen Van A", 20, 180);
        p1.Show();
    }
}
```

Name: Nguyen Van A
Old: 20
Height: 180

CHỈ MỤC (INDEXER)

Việc định nghĩa chỉ mục cho phép tạo các đối tượng của lớp hoạt động giống như một mảng ảo. Tức là các đối tượng có thể sử dụng toán tử [] để truy cập đến một thành phần dữ liệu nào đó. Việc định nghĩa chỉ mục tương tự như việc định nghĩa một thuộc tính.

Cú pháp tạo chỉ mục:

```
public KiểuTraVề this [DanhSáchThamSố]  
{  
get  
{//thân hàm đọc}  
set  
{//thân hàm ghi}  
}
```


CHỈ MỤC (INDEXER)

Ví dụ 9: Xây dựng lớp `IndexedNames` và dùng chỉ mục để truy cập trực tiếp đến các phần tử của `IndexedNames`

```
using System;
namespace C3Vd9
{
    class IndexedNames
    {
        //khai báo một mảng chuỗi
        private string[] namelist = new string[size];
        static public int size = 10;
        public IndexedNames()
        {
            for (int i = 0; i < size; i++)
                namelist[i] = "None";
        }
    }
}
```

CHỈ MỤC (INDEXER)

```
public string this[int index]
{
    get
    {
        string tmp;
        if( index >= 0 && index <= size-1 ){
            tmp = namelist[index];
        }
        else{
            tmp = "";
        }
        return ( tmp );
    }
    set
    {
        if( index >= 0 && index <= size-1 ){
            namelist[index] = value;
        }
    }
}
```

CHỈ MỤC (INDEXER)

```
static void Main(string[] args)
{
    IndexedNames names = new IndexedNames();
    names[0] = "Nguyen Van An";
    names[1] = "Tran Van Bao";
    names[2] = "Le Van Chinh";
    names[3] = "Ngo Van Danh";
    names[4] = "Nguyen Thi Hoa";
    names[5] = "Ho Truc Linh";
    names[6] = "Pham Ha Trang";
    for ( int i = 0; i < IndexedNames.size; i++ )
    {
        Console.WriteLine(names[i]);
    }
}
```

CHỈ MỤC (INDEXER)

Khi biên dịch và thực thi chương trình, tạo ra các kết quả như sau:

```
Nguyen Van An  
Tran Van Bao  
Le Van Chinh  
Ngo Van Danh  
Nguyen Thi Hoa  
Ho Truc Linh  
Pham Ha Trang  
None  
None  
None
```

CHỈ MỤC (INDEXER)

Chú ý:

- Một ***chỉ mục*** phải có ít nhất một tham số, và tham số có thể có kiểu bất kỳ.
- ***chỉ mục*** có thể chỉ có phương thức ***get***.
- Mặc dù ***chỉ mục*** là một đặc điểm thú vị của **C#** nhưng cần phải sử dụng đúng mục đích (sử dụng để đối tượng có thể hoạt động như mảng, mảng nhiều chiều).
- Một lớp có thể có nhiều ***chỉ mục*** nhưng chúng phải có các dấu hiệu phân biệt với nhau (tương tự như quá tải phương thức).

BÀI TẬP KẾT THÚC CHƯƠNG 4

❑ **Bài 1:** Xây dựng lớp Phanso (Phân số) gồm:

- Thuộc tính: Tuso, Mauso (Tử số, Mẫu số)
- Phương thức:
 - Hàm Khởi tạo không Tham số, Hàm hủy
 - Hàm Nhập, xuất
 - Hàm Cong(), Tru(), Nhan(), Chia()

Tính Tổng, Hiệu, Tích, Thương 2 phân số A và B rồi in ra kết quả trên màn hình.

BÀI TẬP KẾT THÚC CHƯƠNG 4

□ **Bài 2:** Xây dựng lớp SoPhuc (Số phức) gồm:

- Thuộc tính: PhanThuc, PhanAo (Phần thực, Phần ảo)
- Phương thức:
 - Hàm Khởi tạo không Tham số, Hàm hủy
 - Hàm Nhập, xuất
 - Hàm Cong(), Tru(), Nhan(), Chia()

Tính Tổng, Hiệu, Tích, Thương 2 số phức A và B rồi in ra kết quả trên màn hình.

BÀI TẬP KẾT THÚC CHƯƠNG 4

❑ **Bài 3:** Xây dựng lớp Tamgiac (Tam giác) gồm:

- Thuộc tính: Cạnh a, Cạnh b, Cạnh c
- Phương thức:
 - Hàm Khởi tạo không Tham số, Hàm hủy
 - Hàm Nhập, xuất
 - Hàm Kiemtra()

Xuất ra màn hình kiểu của tam giác (Tam giác thường, tam giác vuông, tam giác cân, tam giác vuông cân, tam giác đều)

BÀI TẬP KẾT THÚC CHƯƠNG 4

- ❑ **Bài 4:** Xây dựng một ứng dụng quản lý điểm học phần OOP của sinh viên có chứa nội dung như sau:
- Lớp thông tin sinh viên (**Info**) bao gồm: **ID, Hoten, Group** lần lượt là Mã Sinh viên, Họ Tên sinh viên, Lớp sinh hoạt
 - Lớp điểm thành phần (**DiemTP**) bao gồm **TP1, TP2, TP3**

Các phương thức yêu cầu trong bài:

- Nhập mã sinh viên, Họ và tên sinh viên và Lớp SH
- Xuất số lượng sinh viên có trong lớp học phần OOP
- Nhập điểm TP1, TP2, TP3 cho mỗi sinh viên
- Tính điểm TB = $TP1 * 0.1 + TP2 * 0.3 + TP3 * 0.6$
- Xuất ra thông tin của sinh viên và điểm trung bình môn OOP

BÀI TẬP KẾT THÚC CHƯƠNG 4

Bài 5: Xây dựng lớp **Doanhnghiep** gồm các thuộc tính: **TenDN**, **MST**, **Diachi** lần lượt là Tên doanh nghiệp, Mã số thuế, Địa chỉ của doanh nghiệp đó.

Từ đó xây dựng lớp **DanhsachDN** (**Danh sách doanh nghiệp**) với các phương thức:

- Nhập danh sách doanh nghiệp
- Xuất danh sách doanh nghiệp
- Tìm mã số thuế theo tên doanh nghiệp (chỉ mục)
- Tìm tên doanh nghiệp và địa chỉ doanh nghiệp theo mã số thuế (chỉ mục)

BÀI GIẢNG LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

CHƯƠNG 5 KẾ THỪA

TRẦN THỊ THU THẢO

BỘ MÔN TIN HỌC QUẢN LÝ, KHOA THỐNG KÊ – TIN HỌC

TRƯỜNG ĐẠI HỌC KINH TẾ, ĐẠI HỌC ĐÀ NẴNG

THAOTRAN@DUE.EDU.VN

NỘI DUNG

1. Khái niệm Kế thừa

2. Phạm vi Kế thừa

3. Gọi phương thức khởi tạo của lớp cơ sở

4. Định nghĩa phiên bản mới của lớp dẫn xuất

5. Tham chiếu thuộc lớp cơ sở

6. Thực thi trên C#



KHÁI NIỆM KẾ THỪA

□ KHÁI NIỆM

Tính kế thừa trong lập trình là cách 1 lớp có thể thừa hưởng lại những thuộc tính, phương thức từ 1 lớp khác và sử dụng chúng như là của bản thân mình. Hay nói cách khác, kế thừa là cơ chế cho phép định nghĩa một lớp mới (còn gọi là lớp dẫn xuất – derived class, dựa trên một lớp đã có sẵn (còn gọi là lớp cơ sở - base class). Lớp dẫn xuất có hầu hết các thành phần giống như lớp cơ sở (bao gồm tất cả các phương thức và biến thành viên của lớp cơ sở, trừ phương thức private, phương thức khởi tạo, phương thức hủy và phương thức tĩnh)

KHÁI NIỆM KẾ THỪA

□ KHÁI NIỆM

Cú pháp định nghĩa lớp dẫn xuất:

class <tên lớp con> : <tên lớp cơ sở>

```
{  
    // Thân lớp dẫn xuất  
}
```


Trong đó:

- **class** là từ khoá để khai báo lớp.
- <tên lớp con> là tên do người dùng đặt và tuân theo các quy tắc đặt tên <tên lớp cơ sở> là tên lớp mà ta muốn kế thừa các đặc tính của nó.

KHÁI NIỆM KẾ THỪA

Ví dụ 1: Xây dựng lớp Point2D (tọa độ trong không gian 2 chiều), từ đó mở rộng cho lớp Point3D.

```
using System;
namespace C5Vd1
{
    class Point2D
    {
        public int x,y;
        public void Xuat2D()
        {
            Console.WriteLine("{0},{1}",x,y);
        }
    }
}
```



KHÁI NIỆM KẾ THỪA

Ví dụ 1:

Lớp dẫn xuất Point3D kế thừa từ lớp Point2D

```
class Point3D: Point2D
{
    public int z;
    public void Xuat3D()
    {
        Console.WriteLine("{0},{1},{2}",x,y,z);
    }
}
```

Lớp dẫn xuất Point3D, không cần khai báo các biến x, y nhưng trong phương thức Xuat3D(), vẫn truy cập x, y được. Thậm chí, trong hàm Main(), có thể sử dụng đối tượng p3 để gọi phương thức Xuat2D() của lớp cơ sở. Điều này chứng tỏ Point3D được kế thừa các biến x,y từ Point2D

KHÁI NIỆM KẾ THỪA

Ví dụ 1:

```
class PointApp
{
    static void Main(string[] args)
    {
        Point2D p2 = new Point2D();
        p2.x = 1;
        p2.y = 2;
        p2.Xuat2D();
        Point3D p3 = new Point3D();
        p3.x = 4;
        p3.y = 5;
        p3.z = 6;
        p3.Xuat3D();
        p3.Xuat2D();
    }
}
```

1,2
4,5,6
4,5

PHẠM VI KẾ THỪA

Trong C#, không hỗ trợ đa kế thừa (1 lớp kế thừa từ nhiều lớp) nhưng lại hỗ trợ thực thi nhiều interface. Các thành phần của lớp cơ sở có được kế thừa xuống lớp dẫn xuất hay không là do phạm vi truy cập của thành phần đó là gì.

- Thành phần có phạm vi là **private** thì không được kế thừa.
- Thành phần có phạm vi là **protected**, **public** thì được phép kế thừa.

Phương thức khởi tạo và phương thức huỷ bỏ không được kế thừa.

PHẠM VI KẾ THỪA

Ví dụ 2: Nếu ta định nghĩa lớp *ClassA* và *ClassB* kế thừa từ *ClassA* như sau thì câu lệnh $x = x - 1$ sẽ bị báo lỗi

```
namespace C5Vd2
```

```
{  
    class ClassA  
    {  
        int x = 5;  
        public void XuatX()  
        {  
            Console.WriteLine("{0}",x);  
        }  
    }  
    class ClassB: ClassA  
    {  
        public void GiamX()  
        {  
            x = x - 1  
        }  
    }  
}
```

Muốn sửa lỗi phải khai báo lại thành

public int x = 5 hoặc

protected int x = 5

GỌI PHƯƠNG THỨC KHỞI TẠO CỦA LỚP CƠ SỞ

□ **Phương thức khởi tạo** mặc định của lớp cơ sở luôn luôn được gọi mỗi khi có 1 đối tượng thuộc lớp dẫn xuất khởi tạo. Và được gọi trước phương thức khởi tạo của lớp dẫn xuất. Nếu như lớp cơ sở có phương thức khởi tạo có tham số thì đòi hỏi lớp dẫn xuất phải có phương thức khởi tạo tương ứng và thực hiện gọi phương thức khởi tạo của lớp cơ sở thông qua từ khoá **base**.

□ Khi đối tượng của lớp con bị huỷ thì **phương thức huỷ bỏ** của nó sẽ được gọi trước sau đó mới gọi phương thức huỷ bỏ của lớp cơ sở để huỷ những gì lớp con không huỷ được.

GỌI PHƯƠNG THỨC KHỞI TẠO CỦA LỚP CƠ SỞ

□ CÚ PHÁP

```
public <tên lớp>(<danh sách tham số của lớp con>) : base(<danh  
sách tham số>)  
{  
    // Khởi tạo giá trị cho các thành phần của lớp dẫn xuất  
}
```

Trong đó:

- <tên lớp> là tên lớp con (lớp dẫn xuất).
- <danh sách tham số của lớp con> là danh sách tham số của constructor của lớp con.
- base là từ khoá để gọi đến constructor của lớp cơ sở.
- <danh sách tham số> là danh sách tham số tương ứng với constructor của lớp cha.

GỌI PHƯƠNG THỨC KHỞI TẠO CỦA LỚP CƠ SỞ

□ CÚ PHÁP

```
public <tên lớp>(<danh sách tham số của lớp con>) : base(<danh  
sách tham số>)  
{  
    // Khởi tạo giá trị cho các thành phần của lớp dẫn xuất  
}
```

Trong đó:

- <tên lớp> là tên lớp con (lớp dẫn xuất).
- <danh sách tham số của lớp con> là danh sách tham số của constructor của lớp con.
- base là từ khoá để gọi đến constructor của lớp cơ sở.
- <danh sách tham số> là danh sách tham số tương ứng với constructor của lớp cha.

GỌI PHƯƠNG THỨC KHỞI TẠO CỦA LỚP CƠ SỞ

Ví dụ 3: Xây dựng lớp Point2D, từ đó mở rộng cho lớp Point3D dùng phương thức tạo lập của lớp cơ sở

```
using System;  
namespace C5Vd3
```

```
{  
    class Point2D  
    {  
        public int x,y;  
        public Point2D(int a, int b)  
        {  
            x = a; y = b;  
        }  
        public void Xuat2D()  
        {  
            Console.WriteLine("{0},{1}",x,y);  
        }  
    }  
}
```

Phương thức tạo lập của lớp Point2d có tham số

GỌI PHƯƠNG THỨC KHỞI TẠO CỦA LỚP CƠ SỞ

Ví dụ 3: Xây dựng lớp Point2D, từ đó mở rộng cho lớp Point3D dùng phương thức tạo lập của lớp cơ sở

```
class Point3D: Point2D
{
    public int z;
    public Point3D(int a, int b, int c):base(a,b)
    {
        z = c;
    }
    public void Xuat3D()
    {
        Console.WriteLine("{0},{1},{2}",x,y,z);
    }
}
```

Lớp Point3D dẫn xuất từ lớp Point2D, phương thức tạo lập cũng cần có tham số

Hai tham số đầu tiên dùng để khởi gán cho các biến x, y kế thừa từ lớp Point2D, tham số còn lại dùng để khởi gán cho biến thành viên z của lớp Point3D

GỌI PHƯƠNG THỨC KHỞI TẠO CỦA LỚP CƠ SỞ

Ví dụ 3: Xây dựng lớp Point2D, từ đó mở rộng cho lớp Point3D dùng phương thức tạo lập của lớp cơ sở

```
class PointApp
{
    static void Main(string[] args)
    {
        Point2D p2 = new Point2D(1,2);
        Console.WriteLine("Toa do cua diem 2D : ");
        p2.Xuat2D();
        Point3D p3 = new Point3D(4,5,6);
        Console.WriteLine("Toa do cua diem 3D : ");
        p3.Xuat3D();
    }
}
```

```
Toa do cua diem 2D :1,2
Toa do cua diem 3D :4,5,6
```

ĐỊNH NGHĨA PHIÊN BẢN MỚI TRONG LỚP DẪN XUẤT

Nhận xét chung:

Khi cần định nghĩa hai lớp mà chúng có chung một vài đặc trưng, chức năng thì những thành phần đó nên được đặt vào một lớp cơ sở. Sau đó hai lớp này sẽ kế thừa từ lớp cơ sở đó và bổ sung thêm các thành phần của riêng chúng. Ngoài ra, lớp dẫn xuất còn có quyền định nghĩa lại các phương thức đã kế thừa từ lớp cơ sở nhưng không còn phù hợp với nó nữa.

ĐỊNH NGHĨA PHIÊN BẢN MỚI TRONG LỚP DẪN XUẤT

Lớp dẫn xuất kế thừa hầu hết các thành viên của lớp cơ sở vì vậy trong bất kỳ phương thức nào của lớp dẫn xuất ta có thể truy cập trực tiếp đến các thành viên này (mà không cần thông qua một đối tượng thuộc lớp cơ sở). Tuy nhiên, nếu lớp dẫn xuất cũng có một thành phần **X** (biến hoặc phương thức) nào đó trùng tên với thành viên thuộc lớp cơ sở thì trình biên dịch sẽ có cảnh báo dạng như sau:

```
"keyword new is required on 'LớpDẫnXuất.X' because  
it hides inherited member on 'LớpCơSở.X' "
```

ĐỊNH NGHĨA PHIÊN BẢN MỚI TRONG LỚP DẪN XUẤT

Trong lớp dẫn xuất, khi khai báo một thành phần trùng tên lớp thành phần trong lớp cơ sở thì trình biên dịch hiểu rằng người dùng muốn che dấu các thành viên của lớp cơ sở và yêu cầu người dùng đặt từ khóa **new** ngay câu lệnh khai báo thành phần đó. Nếu phương thức của lớp dẫn xuất muốn truy cập đến thành phần **X** của lớp cơ sở thì phải sử dụng từ khóa **base** theo cú pháp: **base.X**

Cú pháp:

```
public new void <Tên phương thức>
{
    ....
}
```

ĐỊNH NGHĨA PHIÊN BẢN MỚI TRONG LỚP DẪN XUẤT

Ví dụ 4: Viết chương trình quản lý thông tin Xe

```
using System;
namespace C5Vd4
{
    class XeHoi
    {
        protected int TocDo;
        protected string BienSo;
        protected string HangSX;
        public XeHoi(int td, string BS, string HSX)
        {
            TocDo = td; BienSo = BS; HangSX = HSX;
        }
        public void Xuat()
        {
            Console.WriteLine("Xe: {0}, Bien so: {1}, Tocdo: {2}
            kmh", HangSX, BienSo, TocDo);
        }
    }
}
```

Lớp XeHoi có phương thức Xuat() xuất ra các thông tin như Biển số, Tốc độ, Hãng sản xuất

ĐỊNH NGHĨA PHIÊN BẢN MỚI TRONG LỚP DẪN XUẤT

Ví dụ 4: Viết chương trình quản lý thông tin Xe

```
class XeCar: XeHoi
{
    int SoHanhKhach;
    public XeCar(int td, string BS, string HSX, int SHK):
base(td,BS,HSX)
    {
        SoHanhKhach = SHK;
    }
    public new void Xuat()
    {
        base.Xuat();
        Console.WriteLine(",{0} cho ngoi", SoHanhKhach);
    }
}
```

Phương thức Xuat() của lớp XeHoi không còn phù hợp với lớp XeCar. Cần định nghĩa một phiên bản mới của phương thức Xuat()

ĐỊNH NGHĨA PHIÊN BẢN MỚI TRONG LỚP DẪN XUẤT

Ví dụ 4: Viết chương trình quản lý thông tin Xe

```
class XeTai: XeHoi
{
    int TrongTai;
    public XeTai(int td, string BS, string HSX, int TT):
base(td,BS,HSX)
    {
        TrongTai = TT;
    }
    public new void Xuat()
    {
        base.Xuat();
        Console.WriteLine(", trong tai {0} tan", TrongTai);
    }
}
```

Sử dụng từ khóa base để đại diện cho lớp cơ sở và gọi đến các thành phần của lớp cơ sở

ĐỊNH NGHĨA PHIÊN BẢN MỚI TRONG LỚP DẪN XUẤT

Ví dụ 4: Viết chương trình quản lý thông tin Xe

```
public class Program
{
    static void Main(string[] args)
    {
        XeCar c = new XeCar(150, "43A-45235", "Toyota", 24);
        c.Xuat();
        XeTai t = new XeTai(150, "43A-98235", "Benz", 12);
        t.Xuat();
    }
}
```

Xe: Toyota, Bien so: 43A-45235, Tocdo: 150 kmh, 24 cho ngoi
Xe: Benz, Bien so: 43A-98235, Tocdo: 150 kmh, trong tai 12 tan

THAM CHIẾU THUỘC LỚP CƠ SỞ

Một tham chiếu thuộc lớp cơ sở có thể trỏ đến một đối tượng thuộc lớp dẫn xuất nhưng nó chỉ được phép truy cập đến các thành phần được khai báo trong lớp cơ sở.

Ở Ví dụ 4, với các lớp XeHoi, XeCar như trên, ta có thể định nghĩa hàm Main() như sau:

THAM CHIẾU THUỘC LỚP CƠ SỞ

```
static void Main(string[] args)
{
    XeCar c = new XeCar(150, "49A-4444", "Toyota", 24);
    c.Xuat();
    Console.WriteLine();
    Console.WriteLine("Tham chieu cua lop co so XeHoi  
co the tro den doi tuong thuoc lop dan xuat XeCar");
    Console.WriteLine("Nhưng chỉ có thể gọi hàm xuất  
tuong ung voi XeHoi");
    XeHoi h = c;
    h.Xuat();
}
```

Xe: Toyota, Bien so: 49A-4444, Tocdo: 150 kmh, 24 cho ngoi

Tham chieu cua lop co so XeHoi co the tro den doi tuong thuoc lop dan xuat XeCar
Nhưng chỉ có thể gọi hàm xuất tuong ung voi XeHoi
Xe: Toyota, Bien so: 49A-4444, Tocdo: 150 kmh

BÀI TẬP ÔN TẬP CHƯƠNG

Bài 1. Xây dựng lớp hình tròn với các thuộc tính (properties): **bán kính, đường kính, diện tích.**

a. Xây dựng lớp hình cầu kế thừa từ lớp hình tròn. Lớp này che dấu đi các thuộc tính: diện tích (dùng từ khóa new) đồng thời bổ sung thêm thuộc tính: **thể tích.**

- Diện tích hình cầu tính bán kính R được tính theo công thức

$$4 * PI * R^2$$

- Thể tích hình cầu tính bán kính R được tính theo công thức

$$4/3 * PI * R^3$$

b. Tương tự, xây dựng lớp hình trụ tròn kế thừa từ lớp hình tròn với các thuộc tính: chu vi mặt đáy, diện tích mặt đáy, diện tích xung quanh, diện tích toàn phần, thể tích.

BÀI TẬP ÔN TẬP CHƯƠNG

Bài 2a. Xây dựng Lớp **People** gồm:

- Thuộc tính: **ID**, **Hoten**, **Tuoi**, **Diachi** để lưu lần lượt các giá trị mã số, tên, tuổi và địa chỉ
- Phương thức trong lớp **People** gồm: hàm **Nhap()** và **Xuat()**, **hàm khởi tạo (có hoặc không có tham số)** và **hàm hủy** (nếu có)

❖ Tạo Lớp **Students** Kế thừa từ lớp **People**

Lớp **Students** sẽ có thêm:

- Thuộc tính **Term** để lưu tên học phần, **TP1**, **TP2**, **TP3** lần lượt là điểm thành phần 1, 2, 3
 - Phương thức **GPA()** để tính điểm trung bình môn học và xếp loại theo thang điểm tín chỉ của môn học đó.
- ❖ **Xuất ra màn hình tất cả thông tin của sinh viên**

BÀI TẬP ÔN TẬP CHƯƠNG

❖ Bài 2b. Tạo Lớp Lecture Kế thừa từ lớp People

Lớp Lecture sẽ có thêm:

- Thuộc tính **Kinhnghiem** để lưu số năm kinh nghiệm, **Hocvi**, **Chucvu** để lưu học vị (ThS/TS/...), chức vụ (Trưởng BM, Trưởng Khoa, ...) của Giảng viên
 - Phương thức **Sapxep()** để sắp xếp giảng viên theo số năm kinh nghiệm giảng dạy
- ❖ Xuất ra màn hình tất cả thông tin của các Giảng viên được sắp xếp theo số năm kinh nghiệm từ nhiều đến ít.

BÀI GIẢNG LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

CHƯƠNG 6

ĐA HÌNH & GIAO DIỆN

TRẦN THỊ THU THẢO

BỘ MÔN TIN HỌC QUẢN LÝ, KHOA THỐNG KÊ – TIN HỌC

TRƯỜNG ĐẠI HỌC KINH TẾ, ĐẠI HỌC ĐÀ NẴNG

THAOTRAN@DUE.EDU.VN

NỘI DUNG

1. Khái niệm đa hình

2. Lớp trừu tượng

3. Giao diện

4. So sánh giữa giao diện và lớp trừu tượng

5. Thực thi trên C#

KHÁI NIỆM ĐA HÌNH

□ KHÁI NIỆM

Tính đa hình là hiện tượng các đối tượng thuộc các lớp khác nhau có thể hiểu cùng 1 thông điệp theo các cách khác nhau.

Ví dụ về đa hình trong thực tế. Ta có 3 con vật: chó, mèo, lợn. Cả 3 con vật này đều là động vật. Nhưng khi ta bảo cả 3 động vật kêu thì con chó sẽ kêu gâu gâu, con mèo sẽ kêu meo meo và con heo sẽ kêu ừn ừn. Trong ví dụ trên 3 con vật: chó, mèo, lợn xem như là các đối tượng. Việc ta bảo 3 động vật kêu chính là thông điệp. Rõ ràng cả 3 con vật có thể hiểu cùng 1 thông điệp là kêu theo các cách khác nhau.



KHÁI NIỆM ĐA HÌNH

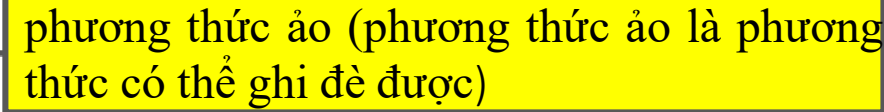
❑ CÁC BƯỚC THỰC HIỆN ĐA HÌNH

1. Lớp cơ sở đánh dấu phương thức ảo bằng từ khóa **virtual** hoặc **abstract**.
2. Các lớp dẫn xuất định nghĩa lại phương thức ảo này (đánh dấu bằng từ khóa **override**).
3. Vì tham chiếu thuộc lớp cơ sở có thể trỏ đến một đối tượng thuộc lớp dẫn xuất và có thể truy cập hàm ảo đã định nghĩa lại trong lớp dẫn xuất nên khi thực thi chương trình, tùy đối tượng được tham chiếu này trỏ tới mà phương thức tương ứng được gọi thực hiện. Nếu tham chiếu này trỏ tới đối tượng thuộc lớp cơ sở thì phương thức ảo của lớp cơ sở được thực hiện.

KHÁI NIỆM ĐA HÌNH

Ví dụ 1: Ta có 3 lớp `Animal`, `Cat`, `Dog`. Trong đó `Cat` và `Dog` kế thừa từ lớp `Animal`. Trong các lớp đều có phương thức `Speak()`.

```
class Animal
{
    public virtual void Speak()
    {
        Console.WriteLine("Animal is speaking. . .");
    }
}
```



Virtual là từ khoá dùng để khai báo 1 phương thức ảo (phương thức ảo là phương thức có thể ghi đè được)

KHÁI NIỆM ĐA HÌNH

Ví dụ 1:

```
class Cat : Animal
{
    public override void Speak()
    {
        Console.WriteLine(" Cat is speaking Meo Meo");
    }
}
class Dog : Animal
{
    public override void Speak()
    {
        Console.WriteLine(" Dog is speaking Gau Gau");
    }
}
```

override là từ khoá dùng để đánh dấu phương thức ghi đè lên phương thức của lớp cơ sở

KHÁI NIỆM ĐA HÌNH

Ví dụ 1:

```
class Program
{
    static void Main(string[] args)
    {
        Animal cat = new Cat();
        Animal dog = new Dog();
        cat.Speak();
        dog.Speak();
    }
}
```

Cat is speaking Meo Meo
Dog is speaking Gau Gau

LỚP ĐỐI TƯỢNG

Tất cả các lớp trong C# đều được dẫn xuất từ lớp Object. Lớp Object là lớp gốc trên cây phân cấp kế thừa, nó cung cấp một số phương thức mà các lớp con có thể ghi đè (override) như:

Phương thức	Ý nghĩa
Equals()	Kiểm tra hai đối tượng có tương đương nhau không
GetHashCode()	Cho phép đối tượng cung cấp hàm Hash riêng để sử dụng trong kiểu tập hợp
GetType()	Trả về kiểu đối tượng
ToString()	Trả về chuỗi biểu diễn đối tượng
Finalize()	Xóa đối tượng trong bộ nhớ
MemberwiseClone()	Tạo copy của đối tượng

LỚP TRỪU TƯỢNG

Phương thức thuần ảo là 1 phương thức ảo và không có định nghĩa bên trong.

Lớp trừu tượng là lớp chứa phương thức thuần ảo.

Abstract là từ khoá dùng để khai báo 1 lớp trừu tượng hoặc 1 phương thức thuần ảo.

LỚP TRỪU TƯỢNG

Xét ví dụ 1: Ở đây phương thức `Speak()` của lớp `Animal`, phần định nghĩa của phương thức này chỉ là hình thức sau đó cũng sẽ bị các lớp kế thừa ghi đè lên.

Khi sử dụng **abstract** để nhấn mạnh 2 điều:

- Phương thức `Speak()` có thể ghi đè (**override**).
- Phương thức `Speak()` không có định nghĩa gì bên trong.

LỚP TRỪU TƯỢNG

Để khai báo lớp trừu tượng và phương thức thuần ảo ta chỉ cần thêm khoá **abstract** vào trước tên lớp và tên phương thức.

```
abstract class <Tên lớp cơ sở>  
{
```

// Khai báo phương thức thuần ảo nên không cần định nghĩa nội dung

```
abstract public void <Tên phương thức>;
```

```
}
```


LỚP TRỪU TƯỢNG

Ví dụ 1: Thay đổi cách viết ở hàm cơ sở, kết quả ta nhận được vẫn như lúc đầu.

```
abstract class Animal
{
    abstract public void Speak();
}
```

Cat is speaking Meo Meo
Dog is speaking Gau Gau

GIAO DIỆN (interface)

□ KHÁI NIỆM

Giao diện là một dạng của lớp trừu tượng được sử dụng với mục đích hỗ trợ tính đa hình. Trong giao diện không có bất cứ một cài đặt nào, chỉ có nguyên mẫu của các phương thức, chỉ mục, thuộc tính mà một lớp khác khi kế thừa nó thì phải có cài đặt cụ thể cho các thành phần này (tức là lớp kế thừa giao diện tuyên bố rằng nó hỗ trợ các phương thức, thuộc tính, chỉ mục được khai báo trong giao diện).

Khi một lớp kế thừa một giao diện ta nói rằng lớp đó thực thi (implement) giao diện.

GIAO DIỆN (interface)

❑ CÚ PHÁP

```
interface <tên interface>
{
    // Khai báo các thành phần bên trong interface
}
```

Trong đó:

- **Interface** là từ khoá dùng để khai báo 1 **interface**.
- **<tên interface>** là tên do người dùng đặt và tuân theo các quy tắc đặt tên
- **Lưu ý** là để tránh nhầm lẫn với lớp kế thừa thì khi đặt tên **interface** người ta thường thêm tiền tố “I” để nhận dạng.

ĐẶC ĐIỂM CỦA `interface`

- Chỉ chứa khai báo không chứa phần định nghĩa (giống phương thức thuần ảo). Mặc dù giống phương thức thuần ảo nhưng bạn không cần phải khai báo từ khoá `abstract`.
- Việc ghi đè 1 thành phần trong `interface` cũng không cần từ khoá `override`.
- Không thể khai báo phạm vi truy cập cho các thành phần bên trong `interface`. Các thành phần này sẽ mặc định là `public`.

ĐẶC ĐIỂM CỦA **interface**

- **Interface** không chứa các thuộc tính (các biến) dù là hằng số hay biến tĩnh vẫn không được.
- **Interface** không có constructor cũng không có destructor.
- Các lớp có thể thực thi nhiều **interface** cùng lúc (ở 1 góc độ nào đó có thể nó là phương án thay thế đa kế thừa).
- Một **interface** có thể kế thừa nhiều **interface** khác nhưng không thể kế thừa bất kỳ lớp nào.

ĐẶC ĐIỂM CỦA interface

Ví dụ 2: Mọi con vật đều có tiếng kêu. Ta có thể dùng định nghĩa một giao diện kèm thêm một thuộc tính để mô tả tiếng kêu của con vật.

```
interface ISpeak
{
    void Speak();
}
class Animal:ISpeak
{
    public void Speak()
    {
        Console.WriteLine("Animal is speaking...");
    }
}
```

ĐẶC ĐIỂM CỦA interface

Ví dụ 2:

```
class Cat : ISpeak
{
    public void Speak()
    {
        Console.WriteLine("Cat is speaking Meo Meo");
    }
}
class Dog : ISpeak
{
    public void Speak()
    {
        Console.WriteLine("Dog is speaking Gau Gau");
    }
}
```

ĐẶC ĐIỂM CỦA interface

```
class Program
{
    static void Main(string[] args)
    {
        Animal animal = new Animal();
        Dog dog = new Dog();
        Cat cat = new Cat();
        animal.Speak();
        dog.Speak();
        cat.Speak();
    }
}
```

Animal is speaking. . .
Dog is speaking Gau Gau
Cat is speaking Meo Meo

SO SÁNH GIỮA **interface** & **abstract**

Những điểm giống nhau giữa **interface** và **abstract class**:

- Điều có thể chứa phương thức thuần ảo.
- Điều không thể khởi tạo đối tượng.

SO SÁNH GIỮA **interface** & **abstract**

Những điểm khác nhau giữa **interface** và **abstract class**:

Interface	Abstract class
Chỉ có thể kế thừa nhiều interface khác	Có thể kế thừa 1 lớp và nhiều interface
Chỉ chứa các khai báo và không có phần nội dung. Không thể chứa biến	Có thể chứa các thuộc tính (biến) và các phương thức bình thường bên trong
Không cần dùng từ khóa override để ghi đè	Sử dụng từ khóa override để ghi đè
Không có constructor và cũng không có destructor	Có constructor và cũng không có destructor
Không có phạm vi truy cập. Mặc định luôn là public	Có thể khai báo phạm vi truy cập

SO SÁNH GIỮA **interface** & **abstract**

Những điểm khác nhau giữa **interface** và **abstract class**:

Interface	Abstract class
Dùng để định nghĩa 1 khuôn mẫu hoặc quy tắc chung	Dùng để định nghĩa cốt lõi của lớp, thành phần chung của lớp và sử dụng cho nhiều đối tượng cùng kiểu
Cần thời gian để tìm phương thức thực tế tương ứng với lớp dẫn đến thời gian chậm hơn	Nhanh hơn so với interface
Khi cần thêm mới 1 khai báo. Cần phải tìm hết tất cả những lớp có thực thi interface này để định nghĩa nội dung cho phương thức mới	Đối với abstract class, khi định nghĩa một phương thức mới, hoàn toàn có thể định nghĩa nội dung phương thức là rỗng hoặc những thực thi mặc định nào đó. Vì thế toàn bộ hệ thống vẫn chạy bình thường

SO SÁNH GIỮA **interface** & **abstract**

Những điểm khác nhau giữa **interface** và **abstract class**:

Interface	Abstract class
Dùng để định nghĩa 1 khuôn mẫu hoặc quy tắc chung	Dùng để định nghĩa cốt lõi của lớp, thành phần chung của lớp và sử dụng cho nhiều đối tượng cùng kiểu
Cần thời gian để tìm phương thức thực tế tương ứng với lớp dẫn đến thời gian chậm hơn	Nhanh hơn so với interface

Nguồn: <https://howkteam.vn/course/lap-trinh-oop-voi-c/interface-trong-lap-trinh-huong-doi-tuong-1396>

BÀI TẬP ÔN TẬP CHƯƠNG

Bài 1. Xây dựng lớp **Hìnhhoc** (Hình học) với phương thức tính **chu vi**, **diện tích** là phương thức trừu tượng hoặc phương thức ảo. Sau đó định nghĩa các lớp **HìnhChuNhat** (Hình chữ nhật), **Hinhtron** (Hình tròn), **Hinhtamgiac** (Hình tam giác), **Hinhvuong** (Hình vuông) kế thừa từ lớp **Hìnhhoc** với các thành phần dữ liệu và phương thức tính chu vi, diện tích cụ thể của từng loại đối tượng.

BÀI TẬP ÔN TẬP CHƯƠNG

Bài 2. Mọi thiết bị (**Máy quạt, Điều hòa, Tivi**) đều có tính năng **Mở (ON)** và **tắt (OFF)**. Hãy dùng định nghĩa một giao diện kèm theo một thuộc tính để cho biết, máy quạt, điều hòa, tivi có đang mở hay không?

BÀI TẬP ÔN TẬP CHƯƠNG

Bài 3. Để quản lý danh mục sách, tạo một lớp **Edition** sử dụng phương thức trừu tượng hoặc phương thức ảo.

- Trong lớp **Edition**, tạo phương thức **CompareTo** để sắp xếp danh mục các ấn phẩm theo tên của Tác giả
- Tạo các lớp dẫn xuất gồm:
 - Book** (title, author, year, publisher)
 - Article** (title, author, journal, year)
 - OnlineResource** (title, author, link, abstract)
- Trong phương thức **Main()**, tạo một mảng gồm n ấn phẩm, hiển thị thông tin đầy đủ từng danh mục, sắp xếp danh mục ấn phẩm theo tên ấn phẩm (**title**) hoặc theo tên tác giả (**author**) và thực hiện chức năng tìm kiếm ấn phẩm theo tên của tác giả (**author**).

BÀI TẬP ÔN TẬP CHƯƠNG

Bài 4. Một khách sạn 5 sao cần quản lý các thông tin cho thuê phòng (Phòng **Standard** và Phòng **VIP**). Với Phòng Standard cần lưu **tên khách hàng, số CMND, Số ngày thuê**. Tiền thuê phòng cho phòng Standard 500\$/ ngày nếu ở dưới 5 ngày, 400\$/ngày nếu ở trên 5 ngày. Với mỗi phòng VIP cần lưu **tên khách hàng, số CMND, Số ngày thuê, loại phòng (phòng **Luxury**, phòng **President**)**. Nếu thuê dưới 5 ngày, phòng Luxury sẽ có giá thuê là 1100\$/ngày và phòng President có giá thuê là 1300\$/ngày. Thuê từ 6 ngày trở lên thì tính 1000\$ cho cả hai loại phòng. Viết chương trình xây dựng các lớp cần thiết, sau đó nhập danh sách thông tin thuê phòng (phòng Standard và phòng VIP), sau đó xuất ra các thông tin sau:

- Xuất ra tất cả các thông tin thuê phòng (kể cả doanh số tương ứng)
- Tính tổng số tiền cho thuê phòng Standard và phòng VIP
- Xuất tất cả các thông tin liên quan đến việc thuê phòng Standard
- Tính tổng số tiền cho thuê phòng Luxury