Hướng đối tượng trong C#

Lập trình trực quan

TS. Cao Thị Luyên
luyenct@utc.edu.vn
0912403345

Nội dung

- Lớp và đối tượng
- Các hàm đặc biệt của lớp
- Overloading
- Thừa kế và đa hình
- Lớp trừu tượng và giao diện
- Xử lý ngoại lệ

Lớp

- lớp là một kiểu mẫu mô tả các đối tượng có chung đặc điểm (thuộc tính, hành vi, cử chỉ…)
- Cú pháp:

```
[phạm vi truy cập] [thuộc tính] class <tên lớp>
{
    //Khai báo các trường -fields
      //Khai báo các thuộc tính properties
      //Khai báo các phương thức methods
}
```

[**phạm vi truy cập**]: khả năng truy nhập thành phần dữ liệu (public, private, internal, protected, internal protected)

internal: giới hạn truy cập trong cùng một project

protected internal: giới hạn truy cập trong project hoặc lớp dẫn xuất

[**thuộc tính**]: có thể là static

Đối tượng

Đối tượng là một thể hiện (đại diện, đại diện) của một lớp

- Cú pháp: Sử dụng từ khoá **new**

<tên lớp> <tên đối tượng>;

<tên đối tượng> = new <tên lớp>([các giá trị khởi tạo nếu có]);

Hoặc:

<tên lớp> <tên đối tượng> = new <tên lớp>([các giá trị khởi tạo nếu có]);

- Chú ý
  - Sau khi khai báo biến đối tượng thì biến đó chỉ là một con trỏ
  - Sau khi cấp phát bằng từ khoá **new** thì con trỏ trỏ tới một đối tượng thực sự

Truy cập thuộc tính/phương thức

- Truy cập thuộc tính
  - <tên đối tượng>.<tên thuộc tính>
- Truy cập phương thức
  - <tên đối tượng>.<tên phương thức>([danh sách đối số nếu có])
- Ví dụ:
  - Tạo đối tượng hình chữ nhật h

```
HCN h;
h = new HCN();
h.Nhap();
h.Xuat();
```

Ví dụ - Lớp hình chữ nhật

```
class HCN
{       protected float Dai, Rong;
        public float ChuVi()
        {
                return (Dai + Rong )*2;
        }
        public float DienTich()
        {
                return Dai* Rong;
        }
        public void Nhap()
        {
                Console.WriteLine("Nhap chieu dai: ");
                Dai = float.Parse(Console.ReadLine());
                Console.WriteLine("Nhap chieu rong: ");
                Rong = float.Parse(Console.ReadLine());
        }
}
```

Ví dụ về lớp Số phức

- Field : phần thực a, phần ảo b
- Properties: A, B để đọc và ghi dữ liệu cho phần thực a và phần ảo b
- Methods:
  - Hàm tạo; khởi gán giá trị cho các fields a,b
  - Tính tổng 2 số phức
  - Tính hiệu, tích, thương
  - Nhập giá trị một số phức
  - Hàm in 1 số phức ra màn hình
  - Ứng dụng lớp Số phức để thực hiện chương trình nhập vào 3 số phức c1,c2,c3. Tính (c1+c2)/(c1-c2)*c3. In các số phức ra màn hình

Phạm vi truy nhập

public	Có thể được truy xuất bởi bất cứ phương thức của lớp nào khác
private	Chỉ có thể truy xuất bởi các phương thức của chính lớp đó
protected	Có thể được truy xuất bởi các phương thức của chính lớp đó và các lớp dẫn xuất (derived) từ nó
internal	Có thể được truy xuất bởi các phương thức của các lớp trong cùng khối kết hợp (assembly)
internal protected	Có thể được truy xuất bởi các phương thức của lớp đó, lớp dẫn xuất từ lớp đó và các lớp trong cùng khối kết hợp (assembly) với nó

Chú ý

- Các thành phần dữ liệu xem như biến toàn cục đối với các phương thức của lớp (các phương thức của lớp có quyền truy cập đến các thành phần này mà không cần khai báo lại)
- Mặc định, mức độ truy cập là **private**
- Các lớp thuộc cùng một project có thể xem là cùng một khối kết hợp (**assembly**)

2. Phương thức Tạo và huỷ đối tượng

- Khởi tạo đối tượng → Gọi Phương thức tạo của lớp (constructor)
  - Được gọi đến 1 cách tự động khi đối tượng của lớp được tạo ra để khởi tạo giá trị đầu cho các thành phần dữ liệu của đối tượng.
  - Nếu không xây dựng thì phương thức tạo mặc định được gọi.
  - Trước khi Phương thức tạo của lớp được chạy, đối tượng chưa thực sự tồn tại trong bộ nhớ, sau khi tạo lập hoàn thành, bộ nhớ lưu trữ một thể hiện của lớp.

Phương thức tạo của lớp mặc định

- Mặc định tạo đối tượng của lớp
- Các thuộc tính được khởi tạo giá trị mặc định

Kiểu dữ liệu	Giá trị mặc định
Numeric (int, long,…)	0
bool	false
char	'\0' (null)
reference	null

Xây dựng Phương thức tạo của lớp

- Phương thức tạo của lớp có tên trùng với tên lớp, không có kiểu trả về, phạm vi truy cập thường là public
- Có thể có nhiều Phương thức tạo của lớp trong cùng lớp
- Phương thức tạo của lớp có thể có tham số hoặc không
- Cú pháp;

```
public class_name()
public class_name(danh sách tham số)
```

Ví dụ - Phương thức tạo của lớp

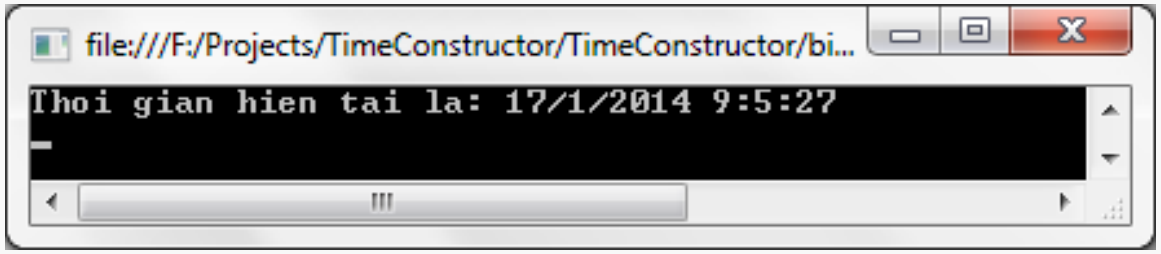
- Xây dựng lớp Time

```
public class Time
    {   //Các thuộc tính
        private int Year, Month, Date; //Năm,Tháng,Ngày
        private int Hour,Minute,Second; //Giờ, Phút, Giây
        //Các phương thức
        //Phương thức tạo (khởi tạo giá trị)
        public Time(DateTime dt)
        {
            Year = dt.Year;
            Month = dt.Month;
            Date = dt.Day;
            Hour = dt.Hour;
            Minute = dt.Minute;
            Second = dt.Second;
        }
        public void HienThiThoiGian()
        {
Console.WriteLine("Thoi gian hien tai la: {0}/{1}/{2} {3}:{4}:{5}", Date,
Month, Year, Hour, Minute, Second);
        }
    }
```

Ví dụ - Phương thức tạo của lớp (tiếp)

```
class Program
    {
        static void Main(string[] args)
        {
            System.DateTime ThoiGianHienTai;
            ThoiGianHienTai = System.DateTime.Now;
            //Tạo đối tượng t thuộc lớp Time
            Time t = new Time(ThoiGianHienTai);
            t.HienThiThoiGian();
            Console.ReadLine();
        }
    }
```

- Kết quả khi chạy chương trình:



Phương thức tạo sao chép

- Phương thức tạo sao chép giúp tạo ra một đối tượng mới và khởi tạo giá trị cho đối tượng mới bằng cách sao chép dữ liệu của đối tượng đã tồn tại
- Cú pháp
  - Khai báo

```
public <tên lớp>(<tên lớp>  <đối tượng 1>)
{
      thuộc tính 1=<đối tượng 1>.thuộc tính 1;
      thuộc tính 2=<đối tượng 1>.thuộc tính 2;
}
```

  - Sử dụng

```
<tên lớp> <đối tượng 2> = new <tên lớp>(đối tượng 1)
```

Ví dụ - Phương thức tạo sao chép

- Thêm Phương thức tạo sao chép lớp Time

```
//Phương thức tạo của lớp sao chép. Sao chép lại các giá trị
//của đối tượng dt
public Time(Time dt)
        {   Year = dt.Year;
            Month = dt.Month;
            Date = dt.Date;
            Hour = dt.Hour;
            Minute = dt.Minute;
            Second = dt.Second;
        }
static void Main(string[] args)
        {   System.DateTime ThoiGianHienTai;
            ThoiGianHienTai = System.DateTime.Now;
             Time t1 = new Time(ThoiGianHienTai);
            t1.HienThiThoiGian();
            //Tạo đối tượng t2 cùng với t1
            Time t2 = new Time(t1);
            t2.HienThiThoiGian();
            Console.ReadLine();
        }
```

Từ khoá this

- Từ khoá ***this*** trỏ đến thể hiện hiện tại (current instance) của đối tượng
- Từ khoá this rất hữu ích trong một số trường hợp

Ví dụ 1: Dùng tham chiếu ***this*** với mục đích tránh xung đột tên của tham số với tên biến dữ liệu của đối tượng.

```
public class Date
{
    private int Year;
    private int Month;
    private int Day;

    public Date(int Day, int Month, int Year)
    {
        Console.WriteLine("Constructor co 3 tham
so!");
        this.Year = Year;
        this.Month = Month;
        this.Day = Day;
    }
```

Từ khoá this

- Dùng làm tham số cho phương thức của đối tượng khác, cho phép phương thức đó có thể tác động đến các thành phần của đối tượng hiện tại
- Ví dụ:

```
class myClass
{
public void Foo(OtherClass otherObject)
{
    otherObject.Bar(this);
}
}
```

- Gọi tường minh các phương thức, thuộc tính của lớp
- Ví dụ:

```
public void MyMethod(int y)
{ …. this.Draw(); }
```

Phương thức huỷ (destructor)

- Dùng để giải phóng vùng nhớ đã cấp phát cho đối tượng khi mà đối tượng không còn được tham chiếu đến.
- Cú pháp: ~<tên lớp>()

```
public classname
{
    public classname()
    {
        // code of constructor
        // các công việc cần thực hiện
    }
    ~ classname()
    {
        // code of descontructor
        // các công việc cần thực hiện
    }
}
```

Phương thức huỷ (destructor)

- C# có cơ chế tự động thu gom rác (garbage collector) → người lập trình không phải huỷ đối tượng một cách tường minh
- Bộ thu gom rác tự động gọi phương thức huỷ

3. Truyền tham số

- Trong C# có thể truyền tham số cho phương thức theo kiểu tham chiếu hoặc tham trị.
- Tham số truyền cho phương thức theo kiểu kiểu tham trị:
  - Một bản sao của tham số đó được tạo ra và bị huỷ khi kết thúc phương thức
  - Giá trị của tham số được truyền không thay đổi sau khi kết thúc phương thức

Truyền tham chiếu

- C# hỗ trợ truyền tham chiếu, sử dụng các từ khoá:
  - **ref**: truyền tham chiếu, biến được tham chiếu phải được khởi tạo trước khi truyền
  - **out**: truyền tham chiếu, biến được tham chiếu không cần khởi gán trước khi truyền. Trong phương thức phải có lệnh gán giá trị cho các biến tham chiếu này.

Ví dụ - truyền tham chiếu, từ khoá ref

```
static void Swap2(ref int a, ref int b)
    {
      int temp;
      temp = a;
      a = b;
      b = temp;
      Console.WriteLine("Trong phuong thuc: a={0}, b={1},",a,b);
    }
static void Main(string[] args)
    {
    int n,m;
    n = 30; m = 40;
    Console.WriteLine("Truoc khi goi swap2: n = {0}, m={1},", n, m);
    Swap2(ref n, ref m);
    Console.WriteLine("Sau khi goi swap2: n = {0}, m = {1},", n, m);
    Console.ReadLine();
    }
```

Ví dụ - truyền tham chiếu, từ khoá out

```
//Phương thức thay đổi giá trị, sử dụng truyền tham chiếu, từ khoá out
static void Change(out int a, out int b)
{
   a = 100;
   b = 200;
   Console.WriteLine("Trong phuong thuc: a={0},b={1},", a,b);
}
static void Main(string[] args)
{
  int n,m;
  Change (out n, out m);
  Console.WriteLine("Sau khi goi Change: n={0}, m={1},",n, m);
  Console.ReadLine();
}
```

Hãy thử 2 ví trên nếu bỏ từ khóa ref, out

4. Nạp chồng hàm

- Khi muốn có nhiều phương thức (hàm) cùng tên, nhiều hàm với tham số đầu vào khác nhau → sử dụng kĩ thuật nạp chồng hàm (overloading)
- Để phân biệt được các hàm với nhau, căn cứ vào một trong hai yếu tố:
  - Khác tên
  - Khác tham số hoặc kiểu dữ liệu của tham số

Ví dụ:

```
void myMethod(int p1);
void myMethod(int p1, int p2);
void myMethod(double p1, double s1);
```

Ví dụ - Nạp chồng hàm

- Xây dựng lớp PhanSo

```
class PhanSo
    {
        int Tu, Mau;
//nạp chồng các phương thức khởi tạo
        public PhanSo()
        {
            Tu = 0;
            Mau = 1;
        }
        public PhanSo(int x)
        {
            Tu = x;
            Mau = 1;
        }
        public PhanSo(int t, int m)
        {
            Tu = t;
            Mau = m;
        }
```

```
public void InPhanSo()
    {

Console.WriteLine("{0}/{1} ",
Tu, Mau);
    }
public PhanSo Cong(PhanSo PS2)
   {
      int TS = Tu * PS2.Mau +
Mau * PS2.Tu;
      int MS = Mau * PS2.Mau;
//Gọi phương thức tạo 2 tham số
      PhanSo KetQua = new
PhanSo(TS, MS);
      return KetQua;
   }
```

Ví dụ - Nạp chồng hàm

```
static void Main(string[] args)
        {
            PhanSo p1 = new PhanSo();
            Console.Write("Phan so p1 = ");
            p1.InPhanSo();
            PhanSo p2 = new PhanSo(5);
            Console.Write("Phan so p2 = ");
            p2.InPhanSo();
            int ts, ms;
            Console.Write("Nhap tu so: ");
            ts = int.Parse(Console.ReadLine());
            Console.Write("Nhap mau so: ");
            ms = int.Parse(Console.ReadLine());
            PhanSo p3 = new PhanSo(ts, ms);
            Console.Write("Phan so p3 = ");
            p3.InPhanSo();
            //p1=p2+p3
            p1 = p2.Cong(p3);
            Console.Write("Phan so p1 = p2 + p3 = ");
            p1.InPhanSo();
            Console.ReadLine();
        }
```

Properties

-Thành phần của lớp cho phép truy cập dữ liệu (Fields) của class

Có thể tạo Properties có thể chỉ đọc, chỉ ghi hoặc vừa đọc vừa ghi bằng cách tạo các setter và getter.

Ví dụ tạo setter và getter của dữ liệu _height của lớp HCN:

```
public float Height //Tạo thuộc tính Height để truyền giá trị và lấy giá trị cho _height
  {  //Lấy giá trị
     get
     {
        return _height;
     }
     //Thiết lập giá trị
     set
     {
        _height = value;
     }
  }
}
```

```
access_modifier return_type
Property_name
{ get { return attribute; }
set { attribute = value; }
}
```

Properties và indexers trong C# – Indexers

Trong lập trình C#, indexer cho phép chúng ta làm việc với một class như mảng. Và indexers cũng tương tự như properties ở 2 phương thức get và set nhưng khác properties ở chỗ, trong indexers chấp nhận tham số.

Cú pháp tạo indexer trong C#

```
access_modifier return_type this [parameter]
{
 get {
   //return value base on index
 }
 set {
  //set value base on index
 }
}
```

Xây dựng lớp dẫn xuất

- Khái niệm: là lớp được xây dựng từ lớp ban đầu bằng cách bổ sung thêm các thành phần fields, properties, methods. Lớp ban đầu gọi là lớp cha (lớp cơ sở); lớp mới xây dựng được gọi là lớp con (lớp lớp dẫn xuất).

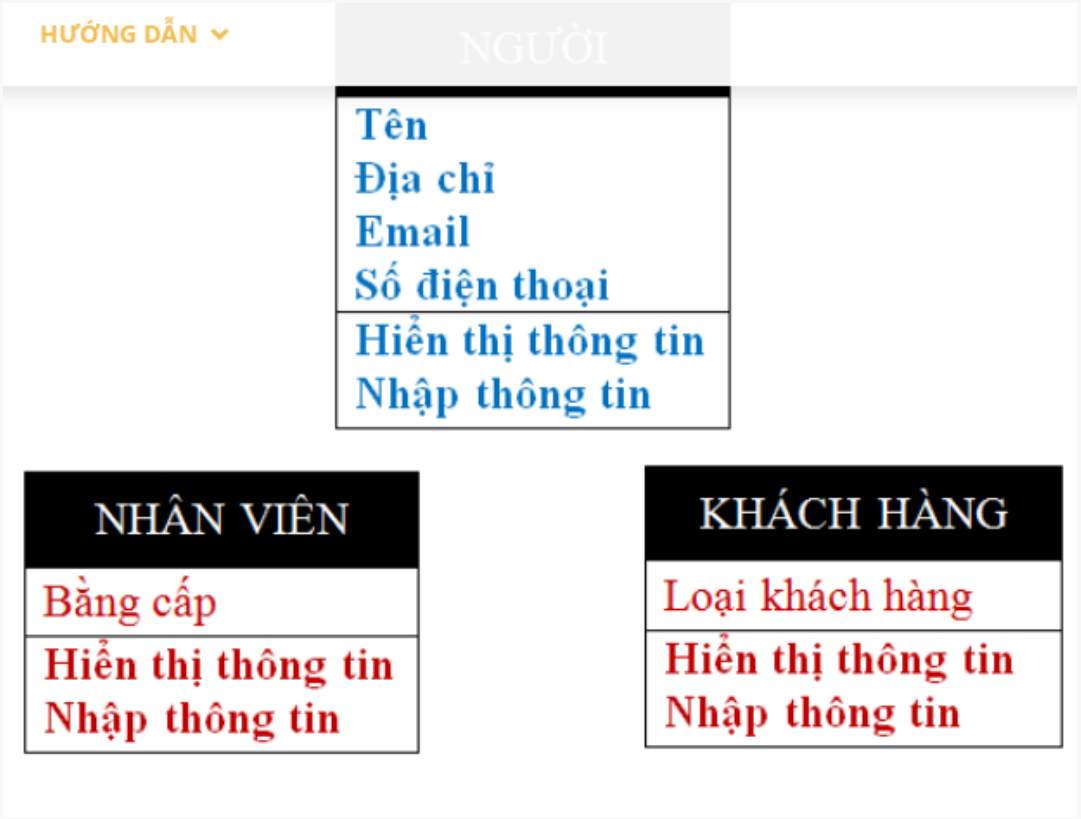
Cú pháp:

Phạm vi tên_lớp_DX: lớp_Cơ_Sở{

//khai báo thành viên riêng của lớp DX

}

HƯỚNG DẪN


NGƯỜI
Tên
Địa chỉ
Email
Số điện thoại
Hiển thị thông tin
Nhập thông tin
NHÂN VIÊN
Bằng cấp
Hiển thị thông tin
Nhập thông tin
KHÁCH HÀNG
Loại khách hàng
Hiển thị thông tin
Nhập thông tin

```
class Nguoi// xay dung lop co so
   {//kb truong dl fields
      string ten, diachi, sodienthoai;
      //khai bao ham tao: can it nhat ham tao khogn doi
      public Nguoi()
      {
         ten = "Toan";diachi = "Hai Phong";          sodienthoai = "00001111";
      }
      public string Ten { get => ten; set => ten = value; }
      public string Diachi { get => diachi; set => diachi = value; }
      public string Sodienthoai { get => sodienthoai; set => sodienthoai = value; }
      //khai bao thuoc tinh properties
      public void xuat1()
      {
         Console.Write(ten + " " + diachi + " " + sodienthoai);
      }
   }
```

```
class NhanVien:Nguoi
   {
      string bangcap;

      public NhanVien():base ()//goi ham tao lop cs
      {
         bangcap = "12/12";
      }
      public  void xuat()
      {
         base.xuat1();//goi ham xuat cua lop cs
         Console.WriteLine(" " + bangcap); ;
      }
   }
```

Lớp trừu tượng và giao diện

- Lớp trừu tượng
  - Là những lớp không hoàn thiện
  - Thiết lập như là lớp cơ sở cho những lớp dẫn xuất
- Phương thức trừu tượng
  - Là phương thức không hoàn thiện (chỉ có nguyên mẫu, không có phần mô tả cài đặt chi tiết)
  - Không có sự thực thi
- Cú pháp

  abstract public class <tên lớp>

  abstract public void <tên phương thức>();

  (có dấu chấm phẩy **;** sau tên phương thức)

Lớp trừu tượng

- Ví dụ
  - Xây dựng lớp HinhHoc gồm 2 phương thức: Tính chu vi, diện tích là phương thức trừu tượng.
  - Xây dựng lớp TamGiac, HinhChuNhat kế thừa từ lớp HinhHoc, xây dựng phương thức tính chu vi, diện tích

```
//Lớp trừu tượng
    abstract public class HinhHoc
    {
        abstract public void Nhap();
        abstract public double  ChuVi();
        abstract public double  DienTich();
    }
```

Ví dụ lớp trừu tượng

```
public class TamGiac:HinhHoc
    {   private double a, b, c;
        public override void Nhap()
        {   Console.Write("Nhap canh a: ");
            a=Convert.ToDouble(Console.ReadLine());
            Console.Write("Nhap canh b: ");
            b=Convert.ToDouble(Console.ReadLine());
            Console.Write("Nhap canh c: ");
            c=Convert.ToDouble(Console.ReadLine());
        }
        public override double ChuVi()
        { return a + b + c;
        }
        public override double DienTich()
        {   double p = (a + b + c) / 2;
            return Math.Sqrt(p * (p - a) * (p - b) * (p - c));
        }
    }
```

Ví dụ lớp trừu tượng

```
//Lớp hình chữ nhật
public class HinhChuNhat : HinhHoc
    {   private double a, b;
        public override void Nhap()
        {
            Console.Write("Nhap chieu dai: ");
            a = Convert.ToDouble(Console.ReadLine());
            Console.Write("Nhap chieu rong: ");
            b = Convert.ToDouble(Console.ReadLine());
        }
        public override double  ChuVi()
        {
            return (a + b) * 2;
        }
        public override double  DienTich()
        {
            return a * b;
        }
    }
```

Ví dụ lớp trừu tượng

```
class Program
    {
        static void Main(string[] args)
        {
            HinhHoc H1;
            TamGiac TG1 = new TamGiac();
            TG1.Nhap();
            Console.WriteLine("Thong tin ve tam giac: ");
            Console.WriteLine("Chu vi la : {0}", TG1.ChuVi());
            Console.WriteLine("Dien tich la : {0,8:f2}", TG1.DienTich());
            HinhChuNhat  HCN1 = new HinhChuNhat();
            HCN1.Nhap();
            Console.WriteLine("Thong tin ve hinh chu nhat: ");
            H1 = HCN1 ;
            Console.WriteLine("Chu vi la : {0}", H1.ChuVi());
            Console.WriteLine("Dien tich la : {0,8:f2}", H1.DienTich());
            Console.ReadLine();
        }
    }
```

Giao diện (interface)

- Giao diện
  - Là một dạng của lớp trừu tượng
  - Sử dụng với mục đích hỗ trợ tính đa hình
  - Chỉ có nguyên mẫu của phương thức, chỉ mục, thuộc tính (Lớp kế thừa từ giao diện phải có cài đặt cụ thể)
  - Lớp kế thừa giao diện được gọi là lớp thực thi (implement) giao diện

Giao diện

- Cú pháp

[Mức độ truy cập] **interface** <Tên giao diện}[:Giao diện cơ sở]

{

Nội dung

}

[Mức độ truy cập] : public hoặc internal

[Giao diện cơ sở] : interface khác mà nó kế thừa

Tên giao diện bắt đầu bằng chữ I

- Chú ý
  - Các thành phần trong giao diện mặc định đều là public
  - Mỗi lớp có thể kế thừa một lớp khác đồng thời kế thừa nhiều giao diện

Ví dụ Giao diện

- Xây dựng giao diện Inguoi gồm các phương thức Nhập, Xuất, thuộc tính Tuoi

```
//Giao diện Inguoi
public interface INguoi
    {
        void Nhap();
        void Xuat();
        int Tuoi
        {
            get;
        }
    }
```

Ví dụ Giao diện

```
//Lớp SinhVien thực thi giao diện INguoi
public class SinhVien : INguoi
    {   private string HoTen;
        private int NamSinh;
        public void Nhap()
        {  Console.Write("Nhap ho ten: ");
            HoTen = Console.ReadLine();
            Console.Write("Nhap nam sinh: ");
            NamSinh = Convert.ToInt16(Console.ReadLine());
        }
        public int Tuoi
        {   get
            { return System.DateTime.Today.Year - NamSinh;
            }
        }
        public void Xuat()
        {   Console.WriteLine("Ho ten: " + HoTen);
            Console.WriteLine("Tuoi: " + Tuoi );
        }
    }
```

Ví dụ Giao diện

```
class Program
    {
        static void Main(string[] args)
        {
            SinhVien SV1 = new SinhVien();
            SV1.Nhap();
            SV1.Xuat();

            INguoi N1 = (INguoi)SV1; //Ép kiểu
            N1.Xuat();
            Console.ReadLine();
        }
    }
```

Selead class/method

- Selead class: được dùng để ngăn thừa kế hay không thể được thừa kế bởi các class khác Sử dụng từ khoá selead
- Selead Method: các phương thức không thể viết chồng (overriden) trong lớp dẫn xuất.
- Virtual method có thể viết chồng trong lớp dẫn xuất nhưng Virtual Sealed Method thì không

Static class/method

- Cả hai loại Static và Sealed đều không thể thừa kế
  Sự khác nhau giữu Static Class và Selead Class:
- Có thể tạo được một object (instance) từ Selead Class, còn Static Class thì không thể.
- Trong Static Class, chỉ có Static members là được phép, không thể viết các method không là static

Xử lý ngoại lệ

- Giới thiệu về ngoại lệ
- Xử lý ngoại lệ
- Cấu trúc try … catch
- Cấu trúc try …catch …finally
- Ném ra ngoại lệ
- Ngoại lệ do người sử dụng định nghĩa

Giới thiệu về ngoại lệ

- Trong lập trình có thể gặp các lỗi sau:
  - Lỗi cú pháp
  - Lỗi logic thuật toán
  - Lỗi thực thi
- Ngoại lệ: các trường hợp hoạt động không bình thường
- Xử lý ngoại lệ như thế nào
  - Làm thế nào để có thể tiếp tục thực hiện

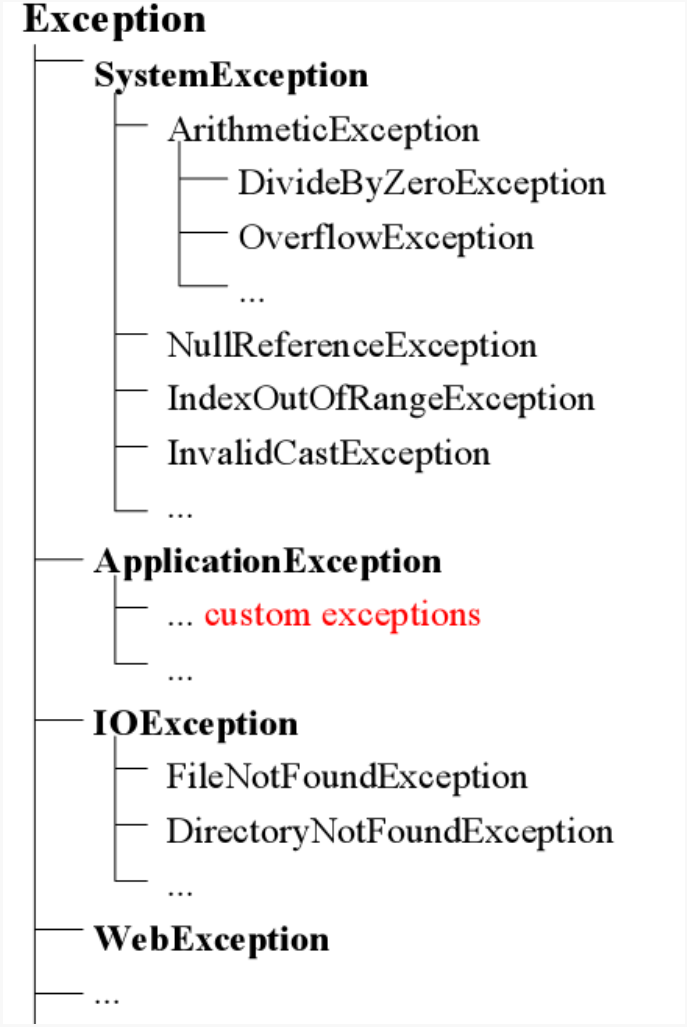
Cách xử lý lỗi truyền thống

- **Cài đặt mã xử lý tại nơi phát sinh ra lỗi**
  - Làm cho chương trình trở lên khó hiểu
  - Không phải lúc nào cũng đầy đủ thông tin để xử lý
  - Không nhất thiết phải xử lý
- **Truyền trạng thái lên mức trên**
  - Thông qua tham số, giá trị trả lại
  - Dễ nhầm, vẫn còn khó hiểu
- **Khó kiểm soát được hết các trường hợp**
  - Lỗi số học, lỗi bộ nhớ, …
- **Lập trình viên thường quên không xử lý lỗi**
  - Bản chất con người
  - Thiếu kinh nghiệm, cố tình bỏ qua

Xử lý ngoại lệ trong C#

- Được kế thừa từ C++, Java
- Dựa trên cơ chế ném và bắt ngoại lệ
  - Ném ngoại lệ: Dừng chương trình và chuyển điều khiển lên mức trên (nơi bắt ngoại lệ)
  - Bắt ngoại lệ: xử lý ngoại lệ
- Ngoại lệ: là đối tượng mang thông tin về lỗi đã xảy ra
  - Ngoại lệ được ném tự động
  - Ngoại lệ được ném tường minh

Phả hệ ngoại lệ trong C#



Ưu điểm của ném bắt ngoại lệ

- Dễ sử dụng
  - Chuyển điều khiển đến nơi có khả năng xử lý ngoại lệ
  - Có thể ném nhiều ngoại lệ
- Tách xử lý ngoại lệ khỏi thuật toán
  - Tách mã xử lý
  - Sử dụng cú pháp khác
- Không bỏ sót ngoại lệ
- Làm chương trình dễ đọc hơn, an toàn hơn

Cấu trúc try … catch

- **Việc phân tách đoạn chương trình thông thường và phần xử lý ngoại lệ được thể hiện thông qua cú pháp `try – catch`**
  - Khối lệnh `try {…}`: khối lệnh có khả năng ném ngoại lệ
  - Khối lệnh `catch() {…}`: bắt và xử lý với ngoại lệ

```
try {
// throw an exception
}
catch (TypeOfException e) {
    exception-handling statements
}
```

Ví dụ bắt ngoại lệ

```
class Program
    {
        static void Main(string[] args)
        {   int a, b;
            Console.Write("Nhap so nguyen a: ");
            a = int.Parse(Console.ReadLine());
            Console.Write("Nhap so nguyen b: ");
            b = int.Parse(Console.ReadLine());
            try
            {
                int thuong = a / b;
                Console.WriteLine("Thuong la: {0}", thuong);
            }
            catch (Exception e)
            {
                Console.WriteLine("Error: " + e.Message);
            }
            Console.ReadLine();
        }
    }
```

Cấu trúc try …catch … finally

- Có thể bắt nhiều loại ngoại lệ khác nhau bằng cách sử dụng nhiều khối lệnh **`catch`** đặt kế tiếp
  - khối lệnh `catch` sau không thể bắt ngoại lệ là lớp dẫn xuất của ngoại lệ được bắt trong khối lệnh `catch` trước
- Khối lệnh **`finally`** có thể được đặt cuối cùng để thực hiện các công việc "dọn dẹp" cần thiết
  - `finally` luôn được thực hiện dù ngoại lệ có được bắt hay không
  - `finally` được thực hiện cả khi không có ngoại lệ được ném ra

Cú pháp try … catch … finally

```
try {
…
}
catch(Exception1 e1) {
…
}
catch(Exception2 e2) {
…
}
finally {
…
}
```

Ví dụ

```
FileStream s = null;
try {
    s = new FileStream(curName, FileMode.Open);
    ...
} catch (FileNotFoundException e) {
    Console.WriteLine("file {0} not found", e.FileName);
} catch (IOException) {
    Console.WriteLine("some IO exception occurred");
} catch {
    Console.WriteLine("some unknown error occurred");
} finally {
    if (s != null) s.Close();
}
```

Ném ra ngoại lệ

- Để ném ra ngoại lệ, dùng throw theo cú pháp:

throw Expresstion;

Ví dụ:

```
if (minute < 1 || minute >= 60) {
  string fault = minute + "is not a valid minute";
  throw new InvalidTimeException(fault);
}
```

Ví dụ ném ra ngoại lệ

```
using System;
class TinhGiaTri
    {
        private int a, b;
        public void TinhToan()
        {   Console.Write("Nhap so nguyen a: ");
            a = int.Parse(Console.ReadLine());
            Console.Write("Nhap so nguyen b: ");
            b = int.Parse(Console.ReadLine());
            try
            {
                int thuong = a / b;
                Console.WriteLine("Thuong la: {0}", thuong);
            }
            catch (DivideByZeroException e)
            {
      throw new DivideByZeroException("Lỗi chia cho 0 rồi nhé! ",e);
            }
        }
    }
```

Ví dụ ném ra ngoại lệ

```
class Program
    {
        static void Main(string[] args)
        {
            TinhGiaTri GT1 = new TinhGiaTri();
            try
            {
                GT1.TinhToan();
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
            Console.ReadLine();
        }
    }
```

Ngoại lệ do người dùng tự định nghĩa

- Khi không tìm được lớp ngoại lệ phù hợp chúng ta có thể tự định nghĩa lớp ngoại lệ bằng cách kế thừa từ lớp Exception

```
throw new FunnyException(10);

class FunnyException : ApplicationException {
    public int errorCode;
    public FunnyException(int x) { errorCode = x; }
}
```

Bài tập về nhà

- Bài 1. Quản lý sinh viên
  - Xây dựng lớp sinh viên quản lý Họ tên, Ngày sinh, Điểm thi môn Lập trình, Cơ sở dữ liệu, Thiết kế web.
  - Xây dựng lớp danh sách gồm N sinh viên
  - Đưa ra số lượng sinh viên được làm khoá luận tốt nghiệp; Số lượng sinh viên làm chuyên đề tốt nghiệp với các điều kiện:
    - Làm khoá luận nếu điểm Trung bình >= 8 và không môn nào dưới 5
    - Làm chuyên đề tốt nghiệp nếu Không có môn nào dưới 5

Bài tập về nhà

- Bài 1. Xây dựng chương trình làm việc với phân số
  - Thuộc tính:

```
        private int TuSo; //Tử số
    private int MauSo; //Mẫu số
}
```

  - Các phương thức
    - Hàm tạo (khởi tạo tử số = 0, mẫu số = 1)
    - Nhập phân số
    - In phân số
    - Rút gọn phân số
    - Tính tổng/hiệu/tích/thương 2 phân số

Bài tập về nhà

- Bài 2. Xây dựng chương trình làm việc với các điểm trong không gian 2 chiều
  - Thuộc tính:

    private double x; //Hoành độ

    private double y; //Tung độ
  - Các phương thức
    - Hàm tạo không tham số: khởi tạo điểm toạ độ (0,0)
    - Hàm tạo 2 tham số x, y: khởi tạo điểm có toạ độ (x,y)
    - Nhập toạ độ
    - In toạ độ điểm ra màn hình
    - Tính khoảng cách giữa 2 điểm

Bài tập về nhà

- Bài 3. Chương trình làm việc với mảng 1 chiều
  - Thuộc tính

    private int n; //Số phần tử của mảng

    int[ ] a; //Mảng 1 chiều
  - Phương thức
    - Hàm tạo Mang1Chieu(int n) để khởi tạo mảng gồm n phần tử
    - Nhập mảng
    - In mảng ra màn hình
    - Sắp xếp mảng

      public void sapxep(int thutu) //thutu = 0: tăng dần, thutu=1: giảm dần
    - Tìm kiếm

      public int timkiem(int m) //Trả về -1 nếu không thấy, trả về vị trí nếu tìm thầy

Bài tập về nhà

- Bài 4. Xây dựng chương trình làm việc với ma trận
  - Thuộc tính

```
private int m; //Số dòng
private int n; //Số cột
private int[ , ] a; //Mảng 2 chiều
```

  - Phương thức
    - Hàm tạo MaTran(int m, int n) để khởi tạo ma trận m dòng, n cột
    - Nhập ma trận
    - In ma trận ra màn hình
    - Cộng 2 ma trận

Bài tập về nhà

– Xây dựng các phương thức trong lớp MaTran
  - Tính hiệu 2 ma trận
  - Tính tích 2 ma trận
  - Tìm ma trận chuyển vị
  - Kiểm tra ma trận có phải ma trận vuông hay không?

Bài tập về nhà

- Bài 5. Xây dựng lớp NhanVien
  - Các thành phần dữ liệu
    - Họ tên, Năm sinh, Địa chỉ, Lương cơ bản, Hệ số, Phụ cấp, Tổng tiền
  - Các phương thức
    - Hàm tạo không tham số
    - Nhập nhân viên
    - Tính lương: Tổng tiền = Lương cơ bản x Hệ số + Phụ cấp
    - In nhân viên ra màn hình

Bài tập về nhà

- Bài 6. Chương trình quản lý Sinh viên
  - Xây dựng lớp SinhVien
    - Các thành phần dữ liệu:
      - Mã sinh viên, Họ tên, Năm sinh, Điểm lập trình, Điểm CSDL , Điểm TB (trong đó: Điểm TB=Điểm Lập trình + Điểm CSDL)/2
    - Các hàm tạo
      - Hàm tạo không tham số
      - Hàm tạo có 5 tham số (Họ tên, năm sinh, quê quán, Điểm lập trình, Điểm CSDL)
    - Các phương thức
      - Nhập thông tin sinh viên
      - In thông tin sinh viên ra màn hình

Bài tập về nhà

– Xây dựng lớp DanhSach

- Thuộc tính

  private int n; //Số lượng sinh viên

  private SinhVien[ ] DS; //Mảng chứa danh sách Sinh viên

  }

- Các phương thức
  - Nhập danh sách sinh viên
  - In thông tin các sinh viên có trong danh sách
  - Liệt kê những sinh viên có điểm TB>8.0
  - Sắp xếp danh sách theo Mã sinh viên

Demo Xây dựng lớp, lớp dx (ConsoleA &WindowA)

- Xây dựng lớp Sách(mã sách, tên sacnh, tên tg, số lượng)
- Xây dựng lớp Sách mới thừa kế từ lớp Sánh bổ sung mã Qrcode.
- Viết chương trình nhập thông tin danh sach đầu sách. Cho biết sách có qrcode cho trước còn trong cửa hàng không? Nếu còn thì cho biết số lượng tồn.
- Tìm sách theo tên sách và tác giả cho trước

Trân trọng cảm ơn!