Xây dựng lớp và giao diện

ThS. Nguyễn Duy Hải

Nội dung

- Khai báo lớp
- Constructor & destructor
- Hàm thành viên
- Thuộc tính
- Đa hình trong C#
- Down cast – up cast
- Abstract class
- Sealed class, nested class
- Interface

Tạo lớp trong C#

- Khai báo lớp

```
[access modifier] class <class name>  [: base class]
{
        // class body
}
```

- Access modifier:
  - public, protected, internal, protected internal, private
- Nếu ko khai báo lớp cơ sở thì C# mặc định xem lớp cơ sở là object
- Lớp luôn là kiểu dữ liệu tham chiếu trong C#

Khóa truy xuất cho class

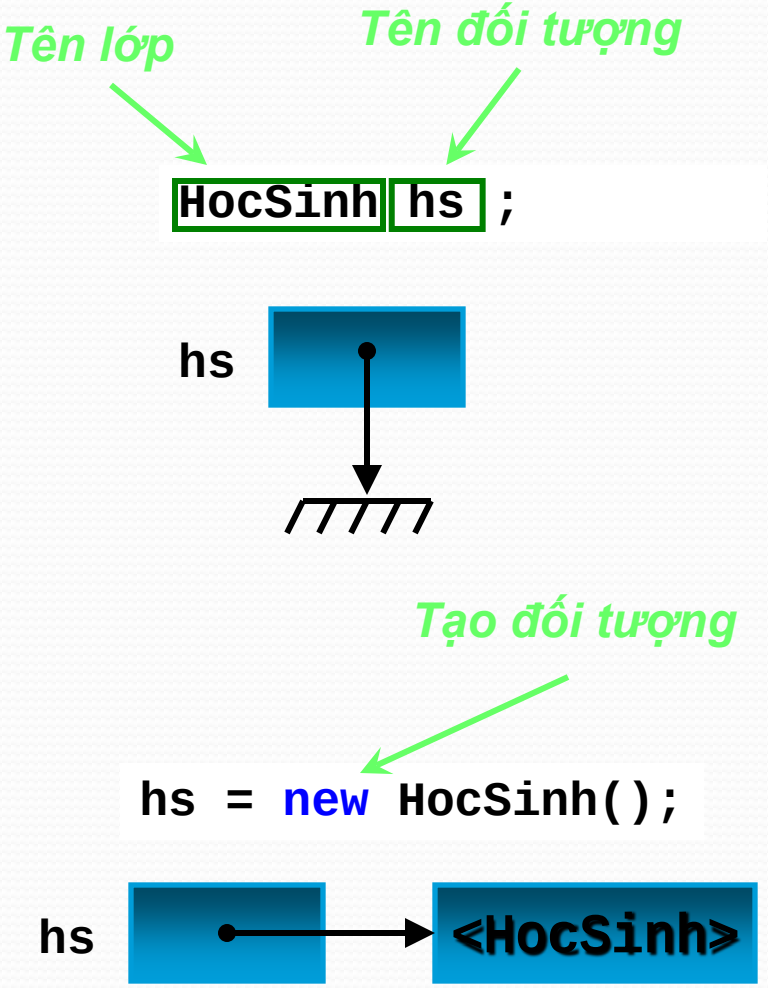
- Một class chứa trong namespace chỉ có 2 khóa truy xuất
  - Public: cho phép bên ngoài assembly truy xuất
  - Internal: chỉ cho phép sử dụng bên trong assembly
- Assembly là tập mã đã được biên dịch sang .NET
  - Một assembly chứa nội dung thực thi chương trình hay thư viện động
  - Assembly có thể chứa trong nhiều file

Các thành phần của class

- Lớp có thể chứa các phần sau
  - Constructor và destructor
  - Field và constant
  - Method
  - Property
  - Indexer
  - Event
  - Chứa các kiểu khác (nested): class, struct, enumeration, interface và delegate

Tạo đối tượng

- Khai báo
  - Trong thân lớp
    - Giống như thuộc tính
  - Trong thân phương thức
    - Tương tự như biến
- Khởi tạo
  - Bằng lệnh new



Constructor

- Được gọi tự động khi tạo đối tượng
- Cùng tên với lớp
- Constructor ko tham số sẽ được tạo mặc định khi không có bất cứ constructor nào
- Cho phép overload constructor để tạo ra nhiều cách khởi tạo đối tượng
- Static constructor: ko tham số, ko access modifier,

Constructor

- Constructor mặc định
  - Không có tham số
  - Khởi tạo thể hiện (đối tượng) khi chưa biết thông tin gì về nó
- Constructor sao chép
  - Tham số vào là đối tượng cùng lớp
  - Tạo ra obj như bản sao của obj đầu vào
- Constructor khác
  - Có một hay nhiều tham số vào
  - Tạo obj khi biết một số thông tin nào về nó

Constructor

```
class HocSinh
{
    //...
    public HocSinh()
    {
        hoTen = “unknown";
        namSinh = 1990;
        diemVan = diemToan = 0;
    }
    public HocSinh(HocSinh hs)
    {
        hoTen = hs.hoTen;
        namSinh = hs.namSinh;
        diemVan = hs.diemVan;
        diemToan = hs.diemToan;
    }
    public HocSinh(string ht)
    {
        hoTen = ht;
    }
}
```

Constructor mặc định

Constructor sao chép

Constructor khác
(tạo học sinh khi biết họ tên)

Constructor

- Khai báo private cho constructor sẽ ko cho phép tạo đối tượng

```
public class MyString
{
    private MyString()
    {
    }
    public static string Concat(string s1, string s2)
    {
        ...
    }
    public static int Compare(string s1, string s2)
    {
        ...
    }
}
public class Tester
{
    public static int Main()
    {

        MyString str = new MyString();
        return 0;
    }
}
```

Ko thể tạo thể hiện/obj

Destructor

- Thực hiện nhiệm vụ "clean" khi đối tượng bị hủy
  - Trùng tên lớp và có dấu "~" phía trước
  - Không có tham số và access modifier
- Mỗi lớp chỉ có 1 destructor

```
class HocSinh
{
    //...
    ~HocSinh()
    {
        siSo--;
    }
}
```

Method

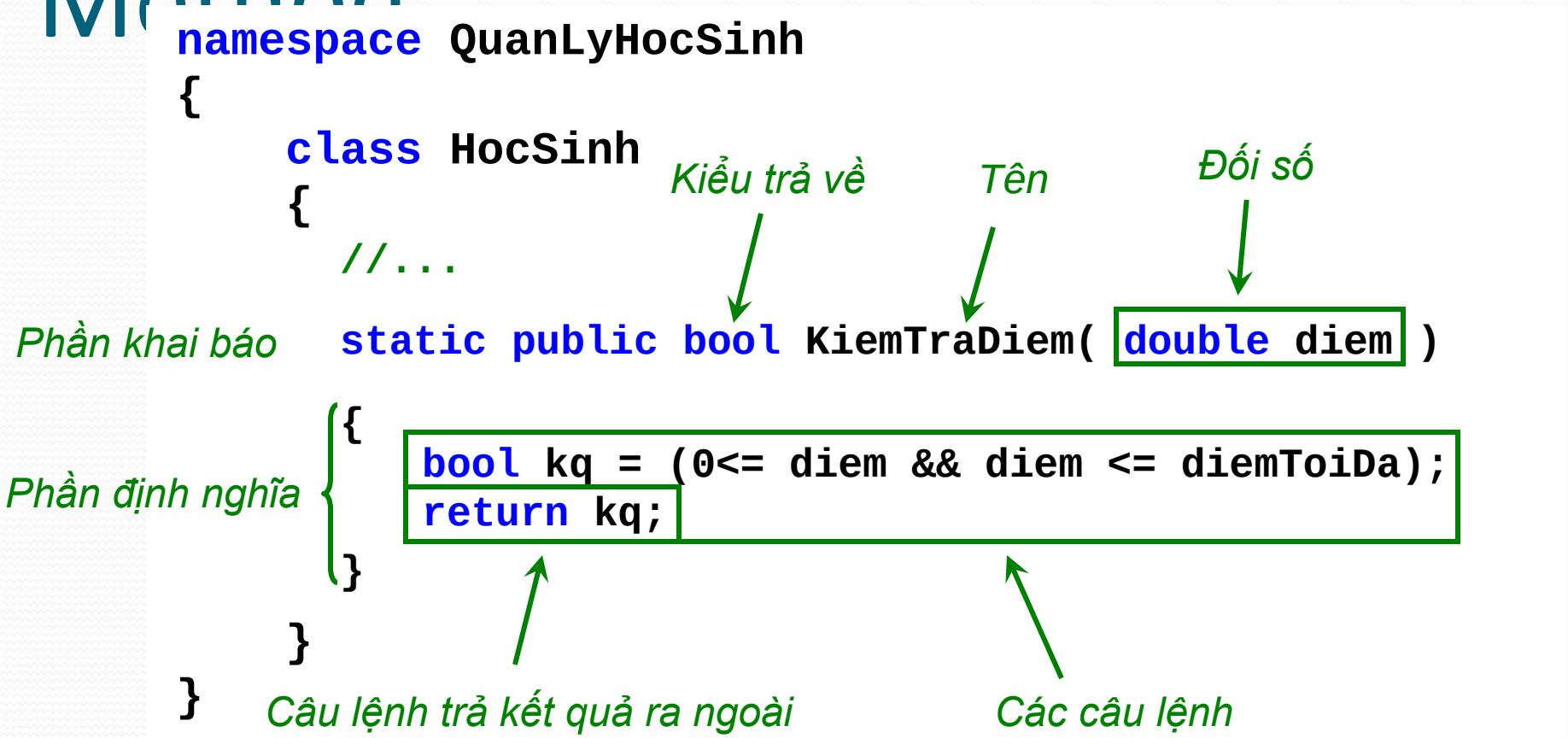
- Hàm, thủ tục khai báo trong class
- Hành vi giao tiếp với bên ngoài
- Static và non static

```
public class CSharp
{
      public CSharp ( )      { . . .}
      public static void StaticMethod( ) { . . .}
    public void NonStaticMethod( ) { . . .}
}
public class Tester() {
      CSharp cs = new CSharp( );
      cs.NonStaticMethod( );
      CSharp.StaticMethod( );
}
```

Truy cập qua thể hiện: cs

Truy cập qua tên lớp: CSharp

Method


namespace QuanLyHocSinh
{
class HocSinh
{
//...
static public bool KiemTraDiem(double diem)
{
bool kq = (0<= diem && diem <= diemToiDa);
return kq;
}
}
}
Kiểu trả về
Tên
Đối số
Phần khai báo
Phần định nghĩa
Câu lệnh trả kết quả ra ngoài
Các câu lệnh

Method

```
namespace QuanLyHocSinh
{
    class HocSinh
    {
        //...

        public void Xuat( )

        {
            Console.WriteLine("Ho ten   : "+hoTen);
            Console.WriteLine("Nam sinh : "+namSinh);
            Console.WriteLine("Diem van : "+diemVan);
            Console.WriteLine("Diem toan: "+diemToan);
        }

    }
}
```

Không có đối số

Kiểu trả về

Các câu lệnh

```
static void ThongBao( double d )
{
    Console.WriteLine("Day la ThongBao(double)");
}
static void ThongBao( int i )
{
    Console.WriteLine("Day la ThongBao(int)");
}
static void ThongBao( int i1, int i2 )
{
    Console.WriteLine("Day la ThongBao(int, int)");
}
static void ThongBao( HocSinh hs )
{
    Console.WriteLine("Day la ThongBao(HocSinh)");
}
```

Các phương thức cùng có tên là ThongBao

Các phương có tham số đầu vào khác nhau

Method - overload

```
ThongBao(40);
```
```
Day la ThongBao(int)
```

```
ThongBao(6.8);
```
```
Day la ThongBao(double)
```

```
ThongBao(new HocSinh());
```
```
Day la ThongBao(HocSinh)
```

```
ThongBao(9,5);
```
```
Day la ThongBao(int, int)
```

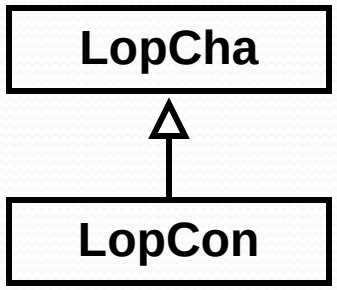
Method – virtual method

Tên lớp con

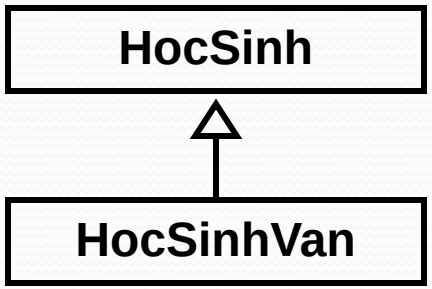
Tên lớp cha

```
class LopCon:LopCha
{

}
```




```
class HocSinhVan: HocSinh
{
    //...
}
```



Tạo ra lớp `HocSinhVan` (học sinh chuyên văn) kế thừa từ lớp `HocSinh`

Method – virtual method

- Phương thức ảo:
  - Khai báo từ khoá virtual
  - Cho phép lớp con có thể thay thế (override)
- Đây chính là thực thi tính đa hình
  - Một phương thức của lớp cơ sở (lớp cha) có thể được thực thi khác nhau ở lớp dẫn xuất (lớp con)

Method – virtual method

- Phương thức tính điểm trung bình của lớp HocSinh

```
class HocSinh
{
    //...
    public virtual float TinhDiemTrungBinh()
    {
        float kq = (diemVan + diemToan) / 2;
        return kq;
    }
}
```

Method – virtual method

- Lớp HocSinhVan phủ quyết lại cách tính điểm trung bình của lớp HocSinh

```
class HocSinhVan:HocSinh
{
    //...
    public override double TinhDiemTrungBinh()
    {
        double kq = (diemVan * 2 + diemToan) / 3;
        return kq;
    }
}
```

Method – virtual method

```
HocSinh          new HocSinh(6,7
float       hs1
HocSinhVan           new HocSinhVan
float       hs2


float       hs1
```

VD Polymorphism

```
public class Shape
{
    public virtual void Draw()
    {
        Console.WriteLine("Shape draw!");
    }
}
public class Line : Shape
{
    public override void Draw()
    {
        Console.WriteLine("Line draw!");
    }
}
public class Circle : Shape
{
    public new void Draw()
    {
        Console.WriteLine("Circle draw!");
    }
}
```

Phủ quyết hàm Draw của Shape

Hàm mới cùng tên Draw với hàm Draw của lớp cơ sở

Polymorphism

```
Shape s1 = new Shape();
s1.Draw();
```

Shape draw!

```
Shape s2 = new Line();
s2.Draw();
```

Line draw!

```
Shape s3 = new Circle();
s3.Draw();
```

Shape draw!

```
Circle c = (Circle)s3;
c.Draw();
```

Circle draw!

Property

- Getter/Setter là các phương thức:
  - Getter: Cho phép đối tượng cung cấp giá trị của thuộc tính ra bên ngoài
  - Setter: Cho phép bên ngoài thay đổi giá trị của thuộc tính của đối tượng *một cách có kiểm soát*
- Property:
  - Được bổ sung vào C# để thay thế cách dùng getter/setter truyền thống

Property

- Đặt vấn đề
  - Lớp **HocSinh** có thuộc tính **diemVan** (điểm văn)
  - Giá trị của **diemVan** phải từ **0** tới **10**
  - Bên ngoài có thể thấy và đổi giá trị của **diemVan**
  - Chỉ cho phép đưa giá trị mới (**diemMoi**) vào **diemVan** nếu giá trị mới là hợp lệ (từ **0** tới **10**)

Property

Dùng getter/setter

```
class HocSinh
{
    protected  double diemVan=0;
    //…
    public double GetDiemVan()
    {
        return diemVan;
    }
    public void SetDiemVan( double diemMoi)
    {
        if (0<=diemMoi&&diemMoi<=10)
            diemVan=diemMoi;
    }
}
```

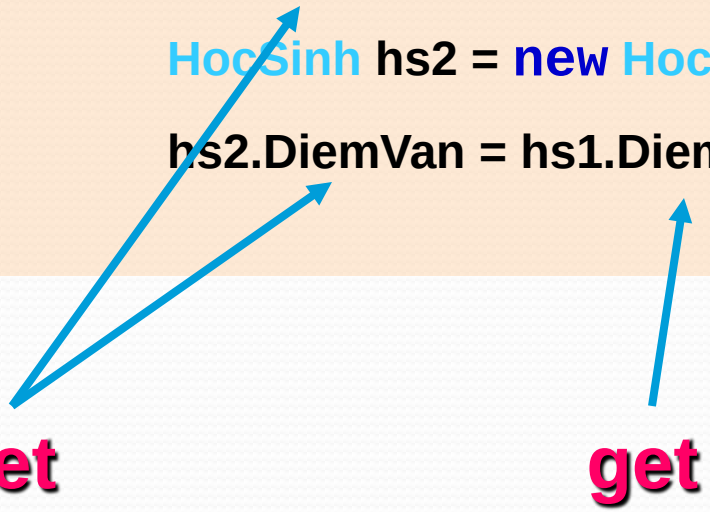
Property Cài đặt bằng property

```
class HocSinh
{
    protected  double diemVan=0;
    //…
    public double DiemVan   {
        get {
            return diemVan;
        }
        set {
            if (0 <= value && value <= 10)
                diemVan = value;
        }
    }
}
```

Property

- Sử dụng property

```
class Tester
{
        HocSinh hs1 = new HocSinh("Nguyen Ha My Tien");
        hs1.DiemVan = 5;
        HocSinh hs2 = new HocSinh("Nguyen Ha Thanh Tung");
        hs2.DiemVan = hs1.DiemVan;
}
```



Up-cast và down-cast

- Up-cast
  - Ép kiểu từ handle con lên handle cha
  - Luôn thành công
  - Thực hiện tự động (implicit)
- Down-cast
  - Ép kiểu từ handle cha xuống handle con
  - Tùy trường hợp
  - Phải chỉ định rõ (explicit)

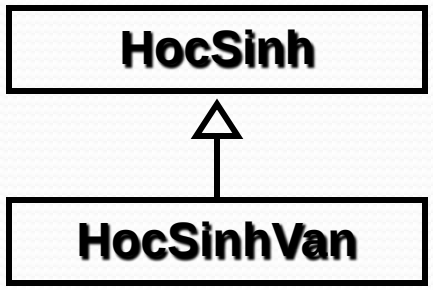
Ví dụ up-cast

```
HocSinh hs = new HocSinhVan();
```

HocSinhVan *là lớp con của* ***HocSinh***

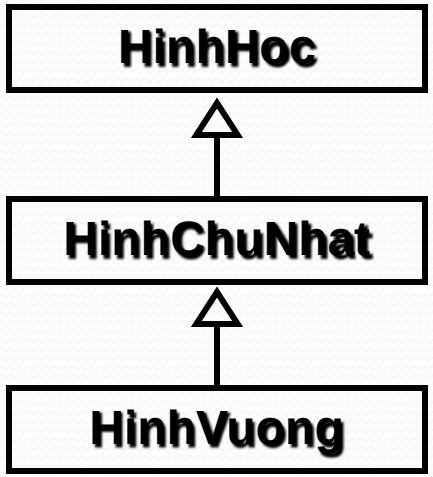
```
HocSinhVan hsv  = new HocSinhVan();
Object o = hsv;
```

HocSinhVan *là lớp con (gián tiếp) của* ***Object***




```
HinhHoc hh = new HinhVuong();
```

HinhVuong *là lớp con của* ***HinhChuNhat***
HinhChuNhat *là lớp con của* ***HinhHoc***



Ví dụ down-cast

```
((HinhVuong)hh).Width = 8;
```

*Được vì handle **hh** đang giữ một đối tượng **HinhVuong***

```
((HinhChuNhat)hh).Width = 8;
```

*Được vì handle **hh** đang giữ một đối tượng **HinhVuong**,*
*mà lớp **HinhChuNhat** là cha của lớp **HinhVuong***

```
((HinhChuNhat)o).Width = 8;
```

*Không được vì handle **o** đang giữ một đối tượng **HocSinhVan**,*
*mà lớp **HocSinhVan** và lớp **HinhChuNhat** không có quan hệ cha-con*
→ cần có user-defined cast

Kiểm tra trước khi down-cast

- Để đảm bảo down-cast thành công, cần kiểm tra xem handle có phải đang giữ đối tượng phù hợp hay không
- Từ khóa: **`is`**

```
if (o is HinhChuNhat)
    ((HinhChuNhat)o).Width = 8;
else
    Console.WriteLine("o khong la HinhChuNhat");
```

Abstract class

- Lớp trừu tượng ko cho phép tạo thể hiện của lớp đó
- Sử dụng polymorphism đòi hỏi
  - khai báo các phương thức là virtual hay abstract trong lớp cơ sở trừu tượng
  - Override chúng trong lớp dẫn xuất (lớp con)
- Bất cứ lớp nào có một phương thức trừu tượng thì phải khai báo lớp là lớp trừu tượng

Abstract class

Abstract Method	Virtual Method
từ khoá: abstract	từ khoá: virtual
Chỉ có phần khai báo method và kết thúc là dấu “;”, Không cần có phần thực thi cho phương thức abstract ở lớp abstract	Có phần thực thi cho phương thức virtual ở lớp cơ sở
Bắt buộc lớp dẫn xuất phải override lại	Không bắt buộc lớp dẫn xuất phải override
từ khoá override trước phương thức ở lớp con	từ khoá override trước phương thức ở lớp con

Abstract class - example

Abstract class:

```
public abstract class AbstractClass
{
        public AbstractClass()
        {
        }
        public abstract int AbstractMethod();
        public virtual int VirtualMethod()
        {
            return 0;
        }
}
```

Abstract class - example

Derived class:

```
public class DerivedClass : AbstractClass
{
        public DerivedClass()
        { }
        public override int AbstractMethod()
        {
                return 0;
        }
        public override int VirtualMethod()
        {
                return base.VirtualMethod ();
        }
}
```

Bắt buộc phải có

Sealed Class

- Còn gọi là lớp niêm phong, không cho phép lớp khác kế thừa nó, ngược với lớp abstract.

```
using System;
sealed class MyClass
{
        public int x;
        public int y;
}
class MainClass {
        public static void Main()
        {
                MyClass mC = new MyClass();
                mC.x = 110; mC.y = 150;
                Console.WriteLine("x = {0}, y = {1}", mC.x, mC.y);
        }
 }
```

S

```
using System;
class MyClass1 {
        public int x;  public int y;
        public virtual void Method() {
            Console.WriteLine("virtual method");
        }
 }
class MyClass : MyClass1 {
        public override sealed void Method() {
            Console.WriteLine("sealed method");
         }
 }
class MainClass {
        public static void Main() {
                MyClass1 mC = new MyClass();
                mC.x = 110; mC.y = 150;
                Console.WriteLine("x = {0}, y = {1}", mC.x, mC.y);
                mC.Method();
        }
 }
```

Sử dụng sealed trước phương thức để ngăn ko cho lớp dẫn xuất override

Nested class

- Lớp được khai báo bên trong thân của lớp khác gọi là: inner class hay nested class, lớp kia gọi là outer class
- Lớp nội có thể truy cập tất cả thành viên của lớp ngoài, kể cả private
- Lớp nội nằm trong lớp ngoài nên nó có thể là private với lớp ngoài
- Khi lớp nội khai báo là public thì có thể truy xuất thông qua tên của lớp ngoài: outer_class.inner_class

Nested class

```
public class Fraction
{
    private int numerator;
    private int denominator;
    public Fraction(int numerator, int denominator)    {
        this.numerator = numerator;
        this.denominator = denominator;
    }
    public override string ToString()    {
            string str = numerator.ToString() + "/" + denominator.ToString();
            return  s;
    }
    public class FractionArtist
    {
        public void Draw(Fraction f)        {
            Console.WriteLine("Drawing the numerator {0}", f.numerator);
            Console.WriteLine("Drawing the denominator {0}", f.denominator);
        }
    }
}
```

Truy xuất được thành phần Private của lớp outer

Lớp nested class

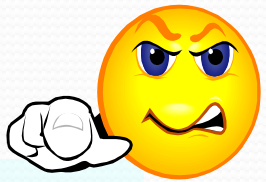
Nested class

```
class Tester
{
    static void Main()
    {
        Fraction f1 = new Fraction(3, 4);
        Console.WriteLine("f1: {0}", f1.ToString());
        Fraction.FractionArtist fa = new Fraction.FractionArtist();
        fa.Draw(f1);
    }
}
```

Truy xuất lớp inner qua lớp outer, lớp outer tương tự như manespace

Interface

- Interface quy định các chức năng nhưng không mô tả cụ thể chúng
- Phương thức trong interface
  - Chỉ khai báo, không định nghĩa
  - Không có từ khóa phạm vi, luôn là **`public`**
- Interface không thể chứa thuộc tính
- Từ khóa: **`interface`**



Có thể xem interface như một bản hợp đồng, nếu lớp nào sử dụng (kế thừa) nó thì phải thực thi đầy đủ các mô tả (phương thức) trong hợp đồng (interface)

Interface

[access modifier] interface <interface name> [: base interface list]

- Một interface có thể kế thừa từ nhiều interface khác
- Một lớp có thể kế thừa từ nhiều interface
  - Phải định nghĩa tất cả phương thức mà các interface "cha" quy định
- Sự kế thừa interface phải đặt sau sự kế thừa lớp
- Các interface thường được đặt tên với tiền tố là "I"
  - IFile, IComparable, IDisposable, IStorable, ICloneable…

Interface - example

```
public interface IStudent
{
        int StudentID
        {
                get;
                set;
        }
        void AddSubject(string subjectName);
}
```

Khai báo property StudentID gồm hàm get ,set

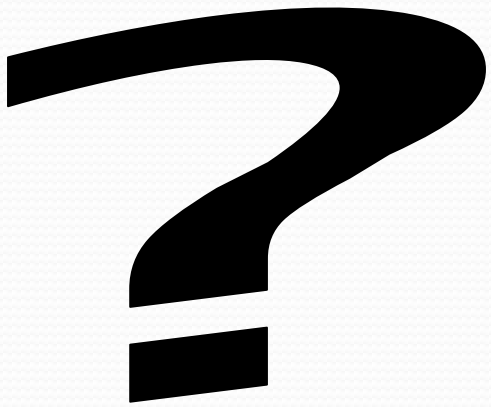
Khai báo phương thức AddSubject

Phải định nghĩa {get,set} của StudentID và AddSubject ở lớp thực thi interface

Interface - example

```
public class Student : IStudent
{
    private int studentID = 0;
    private ArrayList subjects = null;
    public Student()    {}
    public int StudentID
    {
        get {    return studentID;   }
        set {    studentID = value;  }
    }
    public void AddSubject(string subjectName)
    {
        subjects.Add(subjectName);
    }
}
```

Bắt buộc lớp student phải định nghĩa property Student và hàm AddSubject



Xin cảm ơn!