TRƯỜNG ĐẠI HỌC ĐÀ LẠT
KHOA TOÁN - TIN HỌC

PHẠM QUANG HUY

LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

(Bài Giảng Tóm Tắt)

-- Lưu hành nội bộ --

Đà Lạt 2008

BÀI GIẢNG LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

LỜI MỞ ĐẦU

Lập trình theo phương pháp hướng đối tượng là một phương pháp lập trình tương đối mới (xuất hiện từ những năm 1990) và được hầu hết các ngôn ngữ lập trình hiện nay hỗ trợ. Giáo trình này sẽ giới thiệu các đặc trưng của phương pháp lập trình hướng đối tượng như tính đóng gói, tính kế thừa và tính đa hình. Chúng tôi chọn ngôn ngữ **C#** để minh họa, vì đây là ngôn ngữ lập trình hướng đối tượng dễ học và phổ dụng nhất hiện nay. Sau khi hoàn tất giáo trình này, sinh viên sẽ biết được cách mô hình hóa các lớp đối tượng trong thế giới thực thành các lớp đối tượng trong C# và cách phối hợp các đối tượng này để giải quyết vấn đề đang quan tâm.

Trước khi tìm hiểu chi tiết về phương pháp lập trình hướng đối tượng, sinh viên nên đọc trước phần phụ lục ***A-Cơ bản về ngôn …….ngữ C#*** để làm quen với các kiểu dữ liệu, các cấu trúc điều khiển trong ngôn ngữ C#. Sau khi đã nắm bắt được phương pháp lập trình hướng đối tượng, sinh viên nên đọc thêm phần phụ lục ***B-Biệt lệ*** để có thể viết chương trình có tính dung thứ lỗi cao hơn.

MỤC LỤC

I. Giới thiệu lập trình hướng đối tượng

I.1. Lập trình hướng thủ tục (Pascal, C, …)

Trong phương pháp lập trình thủ tục, chương trình là một hệ thống các thủ tục, hàm. Tức là, khi viết chương trình, ta phải xác định chương trình làm những công việc (thao tác) nào? Mỗi thao tác gồm những thao tác con nào? Từ đó mỗi thao tác sẽ tương ứng với một hàm. Như vậy, lập trình theo phương pháp thủ tục là xác định các hàm, định nghĩa các hàm và gọi các hàm này để giải quyết vấn đề được đặt ra.
Một trong những nhược điểm của phương pháp này là mọi hàm đều có thể truy cập biến toàn cục hoặc dữ liệu có thể phải truyền qua rất nhiều hàm trước khi đến được hàm thực sự sử dụng hoặc thao tác trên nó. Điều này dẫn đến sự khó kiểm soát khi chương trình quá lớn và khi phát triển, sửa đổi chương trình.
Một khó khăn nữa đó là việc nhớ các hàm xây dựng sẵn khi số lượng hàm quá nhiều.

I.2. Lập trình hướng đối tượng (Object-oriented programming)

Phương pháp này lấy đối tượng làm nền tảng để xây dựng chương trình. Đối tượng là sự gắn kết giữa dữ liệu của đối tượng và các hàm (còn gọi là phương thức) thao tác trên các dữ liệu này.

Đối tượng = Dữ liệu + Phương thức

Khi viết chương trình theo phương pháp hướng đối tượng ta phải trả lời các câu hỏi:

- Chương trình liên quan tới những lớp đối tượng nào?
- Mỗi đối tượng cần có những dữ liệu và thao tác nào?
- Các đối tượng quan hệ với nhau như thế nào trong chương trình?

Từ đó ta thiết kế các lớp đối tượng và tổ chức trao đổi thông tin giữa các đối tượng, ra lệnh để đối tượng thực hiện các nhiệm vụ thích hợp.

Ví dụ :

- Đối tượng chuỗi :
  - Dữ liệu: mảng các kí tự.
  - Thao tác: tính chiều dài, nối hai chuỗi...
- Đối tượng stack :
  - Dữ liệu: số nguyên hay kí tự , hay một kiểu dữ liệu đã định nghĩa.
  - Thao tác: tạo lập stack, đưa một phần tử vào đỉnh, loại bỏ phần tử ở đỉnh…

Các ngôn ngữ lập trình hướng đối tượng đều có ba đặc điểm chung là tính đóng gói (encapsulation), tính kế thừa (inheritance) và tính đa hình (polymorphism).

I.2.1. Tính đóng gói

Tính đóng gói là kỹ thuật ràng buộc dữ liệu và phương thức thao tác trên dữ liệu đó vào trong lớp để dễ kiểm soát, làm tăng tính trừu tượng của dữ liệu. Lớp đối tượng chỉ cung cấp một số phương thức để giao tiếp với môi trường bên ngoài, che dấu đi cài đặt thực sự bên trong của lớp.

I.2.2. Tính kế thừa

Tính kế thừa là quá trình định nghĩa một lớp đối tượng (gọi là lớp dẫn xuất) dựa trên lớp khác đã định nghĩa gọi là lớp cơ sở nhằm tận dụng các đoạn mã chương trình đã có. Lớp mới chỉ việc bổ sung các thành phần riêng của chính nó hoặc định nghĩa lại các hàm của lớp cơ sở không còn phù hợp với nó.

I.2.3. Tính đa hình

Tính đa hình là ý *tưởng "sử dụng một giao diện chung cho nhiều phương thức khác nhau"*, dựa trên cơ chế liên kết muộn. Tức là phương thức cụ thể sẽ được xác định vào lúc chạy chương trình, tùy thuộc vào đối tượng đang thực thi giao diện đó. Điều này làm giảm đáng kể độ phức tạp của chương trình.

I.2.4. Ưu điểm của phương pháp lập trình hướng đối tượng

- Tính đóng gói làm giới hạn phạm vi sử dụng của các biến, nhờ đó việc quản lý giá trị của biến dễ dàng hơn, việc sử dụng mã an toàn hơn.
- Phương pháp này làm cho tốc độ phát triển các chương trình mới nhanh hơn vì mã được tái sử dụng và cải tiến dễ dàng, uyển chuyển.
- Phương pháp này tiến hành tiến trình phân tích, thiết kế chương trình thông qua việc xây dựng các đối tượng có sự tương hợp với các đối tuợng thực tế. Điều này làm cho việc sửa đổi dễ dàng hơn khi cần thay đổi chương trình.
- …

II. Lớp và đối tượng

Chương trình là một hệ thống các đối tượng. Xây dựng một chương trình là định nghĩa các lớp đối tượng, sau đó khai báo các đối tượng và tổ chức để các đối tượng thực thi nhiệm vụ của mình.

II.1. Định nghĩa lớp

Một lớp là một kiểu cấu trúc mở rộng, đó là một kiểu mẫu chung cho các đối tượng thuộc cùng một loại. Như vậy, thành phần của lớp gồm cấu trúc dữ liệu mô tả các đối tượng trong lớp và các phương thức (còn gọi là hàm, hành vi, thao tác) mà mỗi biến đối tượng của lớp đều có. Các phương thức này thao tác trên các thành phần dữ liệu được khai báo trong lớp.

Việc định nghĩa lớp thể hiện tính đóng gói của phương pháp lập trình hướng đối tượng.

Cú pháp định nghĩa lớp:

[MứcĐộTruyCập] class TênLớp [:LớpCơSở]

{

- Khai báo các thành phần dữ liệu (khai báo biến)

- Định nghĩa các phương thức, thuộc tính của lớp

}

Chú ý:

- *Dữ liệu và phương thức của lớp được gọi chung là thành phần của lớp.*
- *Các thành phần dữ liệu được xem như biến toàn cục đối với các phương thức của lớp, tức là các phương thức của lớp có quyền truy cập đến các thành phần dữ liệu này mà không cần phải khai báo lại trong từng phương thức.*

Mức độ truy cập

Thông thường, mức độ truy cập (***access-modifiers***) của một lớp là ***public***. Ngoài ra các thành phần của lớp cũng có mức độ truy cập riêng. Mức độ truy cập của một thành phần cho biết loại phương thức nào được phép truy cập đến nó, hay nói cách khác nó mô tả phạm vi mà thành phần đó được nhìn thấy.

Bảng sau liệt kê các kiểu mức độ truy cập của các thành phần trong một lớp:

Mức độ truy cập	**Ý nghĩa**
public	Thành viên được đánh dấu public được nhìn thấy bởi bất kỳ phương thức nào của lớp khác.
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

Chú ý:

- *Mặc định, khi không chỉ cụ thể mức độ truy cập thì thành viên của lớp được xem là có mức độ truy cập private.*

- *Mức độ truy cập internal cho phép các phương thức của các lớp trong cùng một khối kết hợp (assembly) với lớp đang định nghĩa có thể truy cập. Các lớp thuộc cùng một project có thể xem là cùng một khối kết hợp.*

II.2. Tạo đối tượng

Lớp mô tả cấu trúc chung của một nhóm đối tượng nào đó, ngược lại, một đối tượng là một trường hợp cụ thể của một lớp (còn gọi là một thể hiện của một lớp).

Vì đối tượng là một kiểu tham chiếu nên dữ liệu thực sự được tạo trên vùng nhớ Heap và ta phải dùng toán tử ***new*** để cấp phát cho đối tượng. Kể từ lúc đối tượng được cấp phát bộ nhớ, ta có thể gán các giá trị cho các biến thành viên, gọi thi hành các phương thức của đối tượng này.

Thường thì ta chỉ việc khai báo và cấp phát đối tượng, việc hủy vùng nhớ mà đối tượng chiếm giữ khi đối tượng đó mất hiệu lực sẽ do bộ dọn rác của trình biên dịch đảm nhiệm.

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);

Chú ý:

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

Ví dụ:

Chương trình nhập chiều dài, chiều rộng của hình chữ nhật và xuất ra diện tích, chu vi của hình chữ nhật.

```
using System;
namespace LopDoiTuongHCN
{
     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());
        }
        public void Xuat()
        {
            Console.WriteLine("Hinh chu nhat: Dai = {0},
            Rong = {1}", Dai, Rong);
        }
    }

    class Application
    {
        static void Main(string[] args)
        {
            HCN h;
            h = new HCN();
            h.Nhap();
            h.Xuat();
            Console.WriteLine("Chu vi hinh chu nhat:
            {0}", h.ChuVi());
            Console.WriteLine("Dien tich hinh chu nhat:
            {0}", h.DienTich());
            Console.ReadLine();
        }
    }
}
```

Trong ví dụ trên, ta định nghĩa một lớp các hình chữ nhật (HCN), mỗi đối tượng thuộc lớp này có thành phần dữ liệu là chiều dài và chiều rộng và có các phương thức như: *nhap(), xuat(), DienTich(), ChuVi()*. Sau đó, trong hàm *Main()* ta khai báo một đối tượng hình chữ nhật tên là *h*, cấp phát vùng nhớ cho đối tượng này và gọi thực hiện các phương thức của nó.

Chú ý:

Nếu ta bỏ đi từ khóa *public* đứng trước mỗi phương thức của lớp HCN thì hàm *Main()* sẽ không thể truy cập đến các phương thức của đối tượng *h* và trình biên dịch sẽ báo lỗi vì khi đó các phương thức này có mức độ truy cập là *private*.

Bài tập 1: *xây dựng lớp hình chữ nhật với thành phần dữ liệu là tọa độ góc trên bên trái (x1, y1), tọa độ góc dưới bên phải (x2, y2) và các phương thức tính chiều dài, chiều rộng, diện tích, chu vi của hình chữ nhật và phương thức vẽ hình chữ nhật bằng các ký tự '*' ra màn hình.*

Bài tập 2: *viết chương trình xây dựng lớp phân số và các thao tác trên phân số như +, -, *, /, tìm ước số chung lớn nhất của tử và mẫu, rút gọn, cộng phân số với một số nguyên.*

Gợi ý:

```
class PhanSo
{
      int Tu, Mau;    // private members
      public void NhapPhanSo()
      {
            // Đoạn mã nhập tử số và mẫu số.
      }
      public void GanGiaTri(int TuSo, int MauSo)
      {
            // Đoạn mã gán giá trị cho tử số và mẫu số.
      }
      public void XuatPhanSo()
      {
            // Đoạn mã xuất tử số và mẫu số ở dạng (a/b)

      }
      public PhanSo Cong(PhanSo PS2)
//cộng phân số hiện hành với phân số PS2 và trả về một phân
số
      {
            PhanSo KetQua = new PhanSo();
            KetQua.TS = Tu * PS2.Mau + Mau* PS2.Tu;
            KetQua.MS = Mau * PS2.Mau;
            return KetQua;
      }
      public PhanSo Tru(PhanSo PS2)
      {
      // Đoạn mã tru phân số hiện hành với phân số PS2
      và trả về một phân số

      }
```

… các phương thức khác

II.3. Phương thức tạo lập (constructor) của một đối tượng

Phương thức tạo lập của một đối tượng có các tính chất sau:

- Được gọi đến một cách tự động khi một đối tượng của lớp được tạo ra. Dùng để khởi động các giá trị đầu cho các thành phần dữ liệu của đối tượng thuộc lớp.
- Tên phương thức giống với tên lớp và có mức độ truy cập là public.
- Không có giá trị trả về.

- Trước khi phương thức tạo lập chạy, đối tượng chưa thực sự tồn tại trong bộ nhớ, sau khi tạo lập hoàn thành, bộ nhớ lưu trữ một thể hiện hợp lệ của lớp.
- Khi ta không định nghĩa một phương thức tạo lập nào cho lớp, trình biên dịch sẽ tự động tạo một phương thức tạo lập mặc định cho lớp đó và khởi tạo các biến bằng các giá trị mặc định.

Thông thường ta nên định nghĩa một phương thức tạo lập cho lớp và cung cấp tham số cho phương thức tạo lập để khởi tạo các biến cho đối tượng của lớp.

Chú ý rằng, nếu lớp có phương thức tạo lập có tham số thì khi khởi tạo đối tượng (bằng toán tử new) ta phải truyền tham số cho phương thức tạo lập theo cú pháp:

TênBiếnĐốiTượng = new TênLớp(DanhSáchĐốiSố);

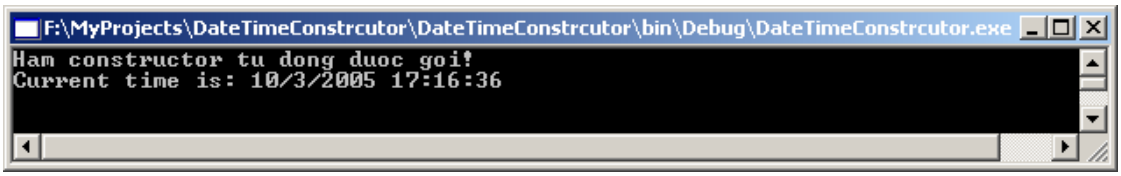
Ví dụ:

Ví dụ sau 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 gán cho các thành phần dữ liệu của đối tượng thuộc lớp ***Time***.

```
using System;
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}",Month, Date, Year, Hour, Minute,
          Second);
     }

     public Time(System.DateTime dt)// constructor
     {
          Console.WriteLine("Ham constructor tu dong duoc
goi!");
          Year = dt.Year;
          Month = dt.Month;
          Date = dt.Day;
          Hour = dt.Hour;
          Minute = dt.Minute;
     }
}
```

```
class DateTimeConstrcutorApp
{
      static void Main( )
      {
            System.DateTime            currentTime            =
System.DateTime.Now;
            Time t = new Time(currentTime);
            t.DisplayCurrentTime( );
            Console.ReadLine();
      }
}
```

Kết quả của chương trình:



Hãy nhấn **F11** chạy debug để hiểu rõ hơn quá trình khởi tạo đối tượng *t,* gọi thực hiện hàm constructor của *t*.

Chú ý rằng, ta cố tình gán giá trị mặc định là 30 cho biến *Second* để biến *Second* của mọi đối tượng thuộc lớp *Time* khi mới được tạo ra đều có giá trị là 30.

II.4. Phương thức tạo lập sao chép (copy constructor)

Phương thức tạo lập sao chép khởi gán giá trị cho đối tượng mới bằng cách sao chép dữ liệu của đối tượng đã tồn tại (cùng kiểu). Ví dụ, ta muốn truyền một đối tượng *Time t1* để khởi gán cho đối tượng *Time t2* mới với mục đích làm cho *t2* có giá trị giống t1, ta sẽ xây dựng phương thức tạo lập sao chép của lớp *Time* như sau:

```
public Time(Time existingTimeObject)
{
      Year        = existingTimeObject.Year;
      Month       = existingTimeObject.Month;
      Date        = existingTimeObject.Date;
      Hour        = existingTimeObject.Hour;
      Minute      = existingTimeObject.Minute;
      Second      = existingTimeObject.Second;
}
```

Khi đó cú pháp khai báo *t2* là:

Time t2 = new Time(t1).

Khi đó hàm copy constructor được gọi và gán giá trị của *t1* cho *t2*.

Bài tập 1: *Xây dựng lớp HocSinh (họ tên, điểm toán, điểm văn) với các phương thức: khởi tạo, xuất, tính điểm trung bình.*

Bài tập 2: *Xây dựng lại lớp PhanSo phần trước với phương thức khởi tạo gồm 2 tham số.*

Bài tập 3: *Xây dựng lớp ngăn xếp Stack lưu trữ dữ liệu số nguyên bằng mảng với các thao tác cơ bản như: Push, Pop, kiểm tra tràn stack, kiểm tra stack rỗng…Dữ liệu của một đối tượng thuộc lớp Stack gồm: Data (mảng số nguyên), Size (kích thước của mảng Data), Top (chỉ số của phần tử nằm trên đỉnh Stack).*

Bài tập 4: *Xây dựng lớp hàng đợi Queue lưu trữ dữ liệu số nguyên bằng mảng với các thao tác trên hàng đợi.*

II.5. Quá tải hàm

Quá tải hàm là định nghĩa các hàm cùng tên nhưng khác tham số hoặc kiểu trả về. Khi chạy chương trình, tùy tình huống mà hàm thích hợp nhất được gọi.

Ví dụ 1:

Minh họa việc quá tải phương thức tạo lập để linh động trong cách tạo đối tượng. Lớp *Date* có 3 phương thức tạo lập có tác dụng lần lượt như sau:

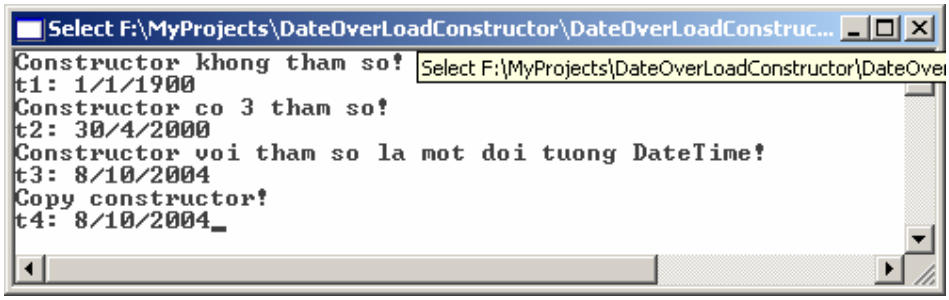
- *public Date():* khởi tạo đối tượng thuộc lớp *Date* với giá trị mặc định là 1/1/1900.
- *public Date(int D, int M, int Y)*: khởi tạo các giá trị *Day, Month, Year* của đối tượng thuộc lớp *Date* bằng ba tham số *D, M, Y*.
- *public Date(Date ExistingDate)*: đây là hàm copy constructor, khởi tạo đối tượng mới thuộc lớp *Date* bằng một đối tượng cùng kiểu đã tồn tại.
- *public Date(System.DateTime dt)*: khởi tạo đối tượng thuộc lớp *Date* bằng dữ liệu của đối tượng thuộc lớp *System.DateTime* (có sẵn).

```
using System;

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

   public void Display( )
   {
        Console.Write("{0}/{1}/{2}", Day, Month, Year);
   }

   // constructors without argument --> set date to
1/1/1900
   public Date()
```

```
    {
        Console.WriteLine("Constructor khong tham so!");
        Year = 1900;
        Month = 1;
        Day  = 1;
    }

    // constructors with DateTime argument
    public Date(System.DateTime dt)
    {
        Console.WriteLine("Constructor voi tham so la mot
        doi tuong DateTime!");
        Year = dt.Year;
        Month = dt.Month;
        Day  = dt.Day;
    }
    // constructors with 3 int arguments
    public Date(int D, int M, int Y)
    {
        Console.WriteLine("Constructor co 3 tham so!");
        Year = Y;
        Month = M;
        Day = D;
    }
    // copy constructors
    public Date(Date ExistingDate)
    {
        Console.WriteLine("Copy constructor!");
        Year = ExistingDate.Year;
        Month = ExistingDate.Month;
        Day = ExistingDate.Day;
    }
}

class DateOverLoadConstructorApp
{
    static void Main(string[] args)
    {
        System.DateTime          currentTime          =
System.DateTime.Now;
        Date t1 = new Date();// t1 = 1/1/1900
        Console.Write("t1: ");
        t1.Display();
        Console.WriteLine();

        Date t2 = new Date(30, 4, 2000);
        Console.Write("t2: ");
        t2.Display( );
        Console.WriteLine();

        Date t3 = new Date(currentTime);
```

```
            Console.Write("t3: ");
            t3.Display( );
            Console.WriteLine();

            Date t4 = new Date(t3);
            Console.Write("t4: ");
            t4.Display( );
            Console.ReadLine();
        }
    }
```

Kết quả của chương trình:



Ví dụ 2:

Quá tải hàm khởi tạo của lớp phân số để linh động khi tạo ra các đối tượng phân số. *(Xem cách trả về của hàm* ***public PhanSo Cong(PhanSo PS2)).***

```
class PhanSo
{
    int Tu, Mau;
// Hàm khởi tạo gán giá trị cố định
    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 XuatPhanSo()
    {
            Console.Write("({0}/{1})", Tu, Mau);
   }
   public PhanSo Cong(PhanSo PS2)
   {
```

```
        int TS = Tu * PS2.Mau + Mau * PS2.Tu;
        int MS = Mau * PS2.Mau;
        //Gọi hàm khởi tạo 2 tham số
          PhanSo KetQua = new PhanSo(TS, MS);
          return KetQua;
      }
}
 class Program
 {
      static void Main(string[] args)
      {
         p1.XuatPhanSo(); Console.WriteLine();

         PhanSo p2 = new PhanSo(3); // p2 = 3/1
         p2.XuatPhanSo(); Console.WriteLine();

         Console.WriteLine("Nhap tu so: ");
         int Ts = int.Parse(Console.ReadLine());
         Console.WriteLine("Nhap mau so: ");
         int Ms = int.Parse(Console.ReadLine());
         PhanSo p3 = new PhanSo(Ts, Ms);
         p3.XuatPhanSo(); Console.WriteLine();

         p1 = p2.Cong(p3);
         p1.XuatPhanSo();
         Console.ReadLine();
     }
}
```

Ta có thể định nghĩa phương thức quá tải chỉ khác nhau ở từ khóa ***ref*** hoặc ***out*** nhưng không thể có hai phương thức chỉ khác nhau ở hai từ khóa ***ref*** và ***out***.

Chẳng hạn, việc quá tải như sau là hợp lệ:

```
class MyClass
{
   public void MyMethod(int i) {i = 10;}
   public void MyMethod(ref int i) {i = 10;}
}
```

nhưng việc quá tải như sau là không hợp lệ:

```
class MyClass
{
   public void MyMethod(out int i) {i = 10;}
   public void MyMethod(ref int i) {i = 10;}
}
```

II.6. Sử dụng các thành viên tĩnh

Dữ liệu và phương thức của một lớp có thể là thành viên thuộc thể hiện của lớp (đối tượng) hoặc thành viên tĩnh (có từ khóa ***static*** đứng trước). Thành viên thể hiện được kết hợp riêng với từng đối tượng của lớp. Như vậy, trong cùng một lớp,

các đối tượng khác nhau có những biến dữ liệu cùng tên, cùng kiểu nhưng được cấp phát ở các vùng nhớ khác nhau và giá trị của chúng cũng có thể khác nhau. Trong khi đó, thành viên tĩnh (biến, phương thức) được coi là phần chung của các đối tượng trong cùng một lớp. Mọi đối tượng thuộc lớp đều có thể truy cập thành viên tĩnh. Nói cách khác, các thành viên thể hiện được xem là toàn cục trong phạm vi từng đối tượng còn thành viên tĩnh được xem là toàn cục trong phạm vi một lớp.

Việc truy cập đến thành viên tĩnh phải thực hiện thông qua tên lớp (không được truy cập thành viên tĩnh thông qua đối tượng) theo cú pháp:

TênLớp.TênThànhViênTĩnh

Chú ý:

- *Phương thức tĩnh thao tác trên các dữ liệu tĩnh và không thể truy cập trực tiếp các thành viên không tĩnh.*

Ngoài ra, ta có thể định nghĩa một phương thức tạo lập tĩnh, phương thức này dùng để khởi gán giá trị cho biến tĩnh của lớp và sẽ chạy trước khi thể hiện của đầu tiên lớp được tạo. Phương thức tạo lập tĩnh hữu dụng khi chúng ta cần cài đặt một số công việc mà không thể thực hiện được thông qua chức năng khởi dựng và công việc cài đặt này chỉ được thực hiện duy nhất một lần.

Ví dụ: Biến thành viên tĩnh được dùng với mục đích theo dõi số thể hiện hiện tại của lớp.

```
using System;
public class Cat
{
     private static int SoMeo = - 6; // bien tinh
     private string TenMeo ;

     // Phuong thuc tao lap cua doi tuong
     public Cat( string T)
     {
          TenMeo = T ;
          Console.WriteLine("WOAW!!!! {0} day!", TenMeo);
          SoMeo++;
     }

     // Phuong thuc tao lap tinh
     static Cat( )
     {
          Console.WriteLine("Bat dau lam thit meo !!!!");
          SoMeo = 0;
     }

     public static void HowManyCats( )
     {
          Console.WriteLine("Dang   lam   thit   {0}   con
meo!",SoMeo);
     }
```

```
}

public class Tester
{
      static void Main( )
      {
            Cat.HowManyCats( );
            Cat tom = new Cat("Meo Tom" );
            Cat.HowManyCats( );
            Cat muop = new Cat("Meo Muop");
            Cat.HowManyCats( );
            //Tom.HowManyCats( ); ---> Error
            Console.ReadLine();
      }
}
```

Trong ví dụ này, ta xây dựng lớp ***Cat*** với một biến tĩnh ***SoMeo*** để đếm số thể hiện (số mèo) hiện có và một biến thể hiện ***TenMeo*** để lưu tên của từng đối tượng mèo. Như vậy, mỗi đối tương ***tom, muop*** đều có riêng biến ***TenMeo*** và chúng dùng chung biến ***SoMeo***.

Ban đầu biến ***SoMeo*** được khởi gán giá trị -6, nhưng khi đối tượng ***tom*** (đối tượng đầu tiên của lớp ***Cat***) được tạo ra, phương thức tạo lập tĩnh ***static Cat()*** tự động thực hiện và gán lại giá trị 0 cho biến tĩnh này.

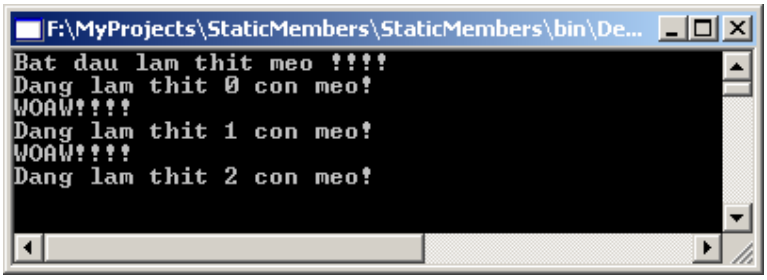
Mỗi khi một đối tượng thuộc lớp ***Cat*** được tạo ra thì phương thức tạo lập của đối tượng này truy cập đến biến đếm ***SoMeo*** và tăng giá trị của biến này lên một đơn vị. Như vậy, khi đối tượng ***tom*** được tạo ra, giá trị của biến này tăng lên thành 1, khi đối tượng ***muop*** được tạo ra, giá trị của biến này tăng lên thành 2. Phương thức tĩnh ***HowManyCats()*** thực hiện nhiệm vụ xuất biến tĩnh ***SoMeo*** thông qua tên lớp bằng câu lệnh:

```
Cat.HowManyCats( );
```

Nếu ta gọi lệnh sau thì trình biên dịch sẽ báo lỗi:

```
tom.HowManyCats( );
```

Kết quả chạy chương trình:



Bài tập: *Xây dựng lớp MyDate lưu trữ các giá trị ngày, tháng, năm với các phương thức: contructor với 3 tham số, xuất, kiểm tra năm nhuận, tính số ngày của tháng theo tháng và năm, xác định ngày kế tiếp của đối tượng ngày/ tháng/ năm hiện hành.*

Gợi ý: *để tính số ngày của tháng ta có thể dùng một mảng tĩnh {31, 28, 31, 30, 31, 30, 31, 31, 30,31, 30,31} để lưu số ngày tương ứng*

với từng tháng. Tuy nhiên với tháng 2 thì tùy năm có nhuận hay không mà ta tính ra giá trị tương ứng.

II.7. Tham số của phương thức

Trong C#, ta có thể truyền tham số cho phương thức theo kiểu tham chiếu hoặc tham trị. Khi truyền theo kiểu tham trị sẽ xảy ra việc sao chép giá trị từ đối số (tham số thực) sang tham số (tham số hình thức). Còn khi truyền theo kiểu tham chiếu thì đối số và tham số đều là một.

C# cung cấp từ khóa ***ref*** để truyền đối số theo kiểu tham chiếu và từ khóa ***out*** để truyền đối số vào trong phương thức theo kiểu tham chiếu mà không cần khởi gán giá trị đầu cho đối số. Tuy nhiên, khi dùng từ khóa ***out*** thì trong phương thức phải có lệnh gán giá trị cho tham chiếu này.

Đối với những dữ liệu kiểu giá trị (**int, long, float, char,**…), muốn thay đổi giá trị của chúng thông qua việc truyền tham số cho hàm, phương thức ta phải truyền theo kiểu tham chiếu một cách tường minh bằng từ khóa ***ref*** hoặc ***out***.

Đối với những dữ liệu kiểu tham chiếu (đối tượng, mảng), khi dùng chúng để truyền đối số mà không có từ khóa ***ref*** hoặc ***out***, ta chỉ có thể làm thay đổi giá trị của vùng nhớ trong **heap** mà chúng trỏ tới nhưng bản thân tham chiếu (địa chỉ vùng nhớ) không bị thay đổi. Nếu muốn thay đổi vùng nhớ mà chúng trỏ tới ta phải dùng từ khóa ***ref*** hoặc ***out*** một cách tường minh.

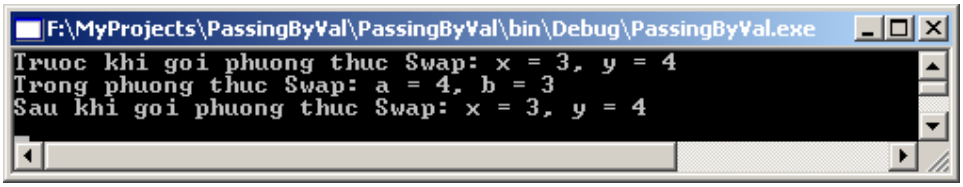
II.7.1. Truyền tham trị bằng tham số kiểu giá trị

Trong ví dụ sau đây, *a* và *b* là hai tham số dạng tham trị của phương thức *Swap* 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.

```
using System;
class PassingByVal
{
     static void Swap(int a, int b)
     {
          int Temp = a;
          a = b;
          b = Temp;
          Console.WriteLine("Trong phuong thuc Swap: a =
          {0}, b =  {1}", a, b);
     }

     static void Main(string[] args)
     {
          int x = 3, y = 4;
          Console.WriteLine("Truoc khi goi phuong thuc
          Swap: x =      {0}, y = {1}", x, y);
          Swap(x,y);
          Console.WriteLine("Sau khi goi phuong thuc Swap:
          x =  {0}, y = {1}", x, y);
```

```
            Console.ReadLine();
        }
}
```

Kết quả chương trình:



II.7.2. Truyền tham chiếu bằng tham số kiểu giá trị với từ khóa ref

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

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

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

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

Một phương thức chỉ có thể trả về một giá trị, do đó khi muốn phương thức trả về nhiều giá trị, chúng ta dùng cách thức truyền tham chiếu. Ví dụ, phương thức *GetTime* sau đây trả về các giá trị *Hour, Minute, Second*.

```
using System;

public class Time
{
      private int Hour;
      private int Minute;
      private int Second;

      public void Display( )
      {
            Console.WriteLine("{0}:{1}:{2}",   Hour,   Minute,
Second);
      }

      public void GetTime(ref int h, ref int m, ref int s)
      {
            h = Hour;
            m = Minute;
            s = Second;
      }

      public Time(System.DateTime dt)
      {
            Hour = dt.Hour;
            Minute = dt.Minute;
```

```
            Second = dt.Second;
      }
}

public class PassingParameterByRef
{
      static void Main( )
      {
            DateTime currentTime = DateTime.Now;
            Time t = new Time(currentTime);
            t.Display( );

            int theHour = 0;
            int theMinute = 0;
            int theSecond = 0;

            t.GetTime(ref   theHour,ref      theMinute,   ref
theSecond);
            Console.WriteLine("Current   time:   {0}:{1}:{2}",
            theHour, theMinute, theSecond);
            Console.ReadLine();
      }
}
```

II.7.3. Truyền tham chiếu với tham số kiểu giá trị bằng từ khóa out

Mặc định, C# quy định tất các biến phải được gán giá trị trước khi sử dụng, vì vậy, trong ví dụ trên, nếu chúng ta không khởi tạo các biến *theHour, theMinute* bằng giá trị 0 thì trình biên dịch sẽ thông báo lỗi. Từ khóa ***out*** cho phép ta sử dụng tham chiếu mà không cần phải khởi gán giá trị đầu. Trong ví dụ trên, ta có thể sửa phương thức *GetTime* thành:

public void GetTime(out int h, out int m, out int s)

Và hàm *Main()* được sửa lại như sau:

```
static void Main( )
      {
            DateTime currentTime = DateTime.Now;
            Time t = new Time(currentTime);
            t.Display( );
            /*int theHour = 0;
            int theMinute = 0;
            int theSecond = 0;*/
            int theHour;
            int theMinute;
            int theSecond;
            t.GetTime(out   theHour,out      theMinute,   out
theSecond);
            Console.WriteLine("Current   time:   {0}:{1}:{2}",
            theHour, theMinute, theSecond);
```

```
            Console.ReadLine();
      }
```

II.7.4. Truyền tham trị với tham số thuộc kiểu tham chiếu

Khi truyền tham số theo cách này ta chỉ có thể thực hiện các thao tác làm thay đổi các dữ liệu thành phần của đối số. Các thao tác làm thay đổi toàn bộ đối số không có tác dụng.

Ví dụ 1:

Xét hàm ***NghichDao2(PhanSo p)*** trong ví dụ dưới đây. Phân số p là dữ liệu thuộc kiểu tham chiếu và được truyền cho hàm theo kiểu tham trị, vì vậy, thao tác gán

p = p2

không có thác dụng.

```
class PhanSo
{
      int Tu, Mau;

      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 XuatPhanSo()
      {
            Console.Write("({0}/{1})", Tu, Mau);
      }

      public PhanSo Cong(PhanSo PS2)
      {
            int TS = Tu * PS2.Mau + Mau * PS2.Tu;
            int MS = Mau * PS2.Mau;

            return  new PhanSo(TS, MS);
      }
```

```
        public static void NghichDao1(PhanSo p)
        {
              int t = p.Tu;
              p.Tu = p.Mau;
              p.Mau = t;

              }
        public static void NghichDao2(PhanSo p)
        {
              PhanSo p2 = new PhanSo();
              p2.Mau = p.Tu;
              p2.Tu = p.Mau;
              p  = p2;
        }
        public static void NghichDao3(ref PhanSo p)
        {

              PhanSo p2 = new PhanSo();
              p2.Mau = p.Tu;
              p2.Tu = p.Mau;
              p = p2;
        }
 }
class Program
{
 static void Main(string[] args)
 {
        PhanSo p1 = new PhanSo(); // p1 = 0/1
        p1.XuatPhanSo(); Console.WriteLine();

        PhanSo p2 = new PhanSo(3); // p2 = 3/1
        p2.XuatPhanSo(); Console.WriteLine();

        Console.WriteLine("Nhap tu so: ");
        int Ts = int.Parse(Console.ReadLine());
        Console.WriteLine("Nhap mau so: ");
        int Ms = int.Parse(Console.ReadLine());

        p3 = new PhanSo(Ts, Ms);
        p3.XuatPhanSo(); Console.WriteLine();

        p1 = p2.Cong(p3);
        p1.XuatPhanSo();

        PhanSo.NghichDao1(p1);
        p1.XuatPhanSo(); Console.WriteLine();

        PhanSo.NghichDao2(p1);
        p1.XuatPhanSo(); Console.WriteLine();
        PhanSo.NghichDao3(ref p1);
```

```
        p1.XuatPhanSo();
        Console.ReadLine();
    }
}
```

Ví dụ 2:

Ví dụ sau đây minh họa việc truyền tham số là tham chiếu (*myArray*) cho phương thức *Change* theo kiểu tham trị. Đối số *myArray* là một dữ liệu tham chiếu, nó trỏ tới một mảng số nguyên được cấp phát trong **Heap**. Khi truyền *myArray* theo kiểu tham trị cho phương thức *Change*, ta chỉ có thể làm thay đổi các giá trị của vùng nhớ trong **Heap**; tức là, ta chỉ có thể thay đổi giá trị của một ô nhớ trong mảng *myArray,* mọi thao tác cấp phát lại vùng nhớ cho *myArray* thực hiện trong phương thức này sẽ không ảnh hưởng tới mảng gốc được cấp phát trong hàm ***Main().***

```
using System;

class PassingRefByVal
{
        static void Change(int[] arr)
        {
        // Lệnh nay thay lam thay doi gia tri cua vung nho
trong     heap.
            arr[0]= 888;
        // Cap phat lai arr, lam thay doi dia chi cua arr.
Lenh nay  chi co tac dung cuc bo. Sau khi goi ham Change,
gia tri cua    Arr[0] van khong thay doi.
            arr   =   new   int[5]   {-3,   -1,   -2,   -3,   -4};
            Console.WriteLine("Trong phuong thuc Change, phan
            tu dau tien la: {0}", arr[0]);
        }

        public static void Main()
        {
            int[] myArray = {1,4,5};
            Console.WriteLine("Trong ham Main,truoc khi goi
            phuong thuc Change, phan tu dau tien la: {0}",
            myArray [0]);
            Change(myArray);
            Console.WriteLine("Trong ham Main, sau khi goi
            phuong thuc Change, phan tu dau tien la:  {0}",
            myArray [0]);
            Console.ReadLine();
        }
}
```

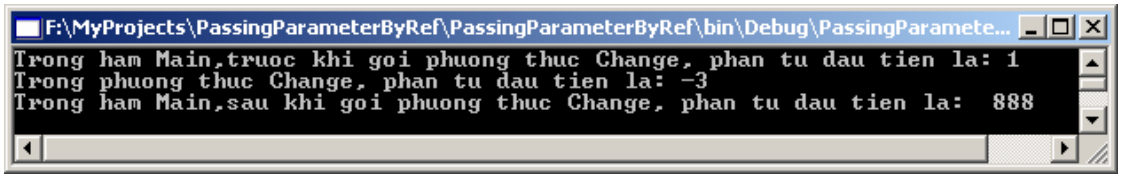
Kết quả chương trình cho thấy: trong hàm ***Change()***, ta chỉ có thể thay đổi giá trị của vùng nhớ do *myArray* trỏ tới (được cấp phát trong **heap**) bằng lệnh:

```
arr[0]= 888;
```

mà không thay đổi được bản thân tham chiếu *myArray* bằng lệnh cấp phát lại:

```
arr = new int[5] {-3, -1, -2, -3, -4};
```

Vì nếu thay đổi được tham chiến *myArray* thì sau khi gọi phương thức *Change* giá trị của phần từ đầu tiên trong mảng phải bằng -3.



II.7.5. Truyền tham chiếu với tham số thuộc kiểu dữ liệu tham chiếu

Với cách truyền tham số này, những câu lệnh làm thay đổi tham số sẽ có hiệu lực.

Trong ví dụ 2 phần II.7.4 nếu ta khai báo lại nguyên mẫu của phương thức *Change* như sau:

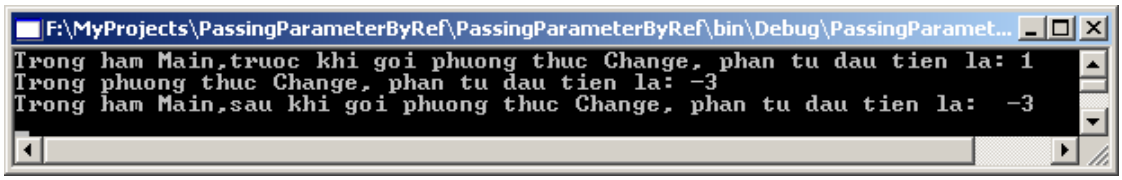
```
static void Change(ref int[] arr)
```

và trong hàm Main() ta thay câu lệnh `Change(myArray);` bằng câu lệnh:

```
Change(ref myArray);
```

thì mọi thao tác làm thay đổi giá trị của vùng nhớ trong **heap** do *myArray* trỏ tới hoặc mọi thao tác cấp phát lại vùng nhớ cho *myArray* thực hiện trong phương thức này sẽ làm thay đổi mảng gốc được cấp phát trong hàm *Main()*.

Kết quả chạy chương trình:



Điều này chứng tỏ ta có thể thay đổi đối số thuộc kiểu tham chiếu.
Ta có thể lý giải tương tự cho hàm `NghichDao3(ref PhanSo p)` trong ví dụ 1 phần II.7.4.

> ***Bài tập 1:*** *Hãy viết phương thức để hoán vị hai dữ liệu kiểu chuỗi (string). Hãy giải thích tại sao khi dùng phương thức này ta cần phải truyền tham chiếu cho phương thức dù string là dữ liệu thuộc kiểu tham chiếu.*
>
> ***Bài tập 2:*** *Viết chương trình nhập một lớp gồm N học sinh, mỗi học sinh có các thông tin như: họ, tên, điểm văn, điểm toán, điểm trung bình.*
> *Tính điểm trung bình của từng học sinh theo công thức: (điểm văn + điểm toán)/2.*
> - *Tính trung bình điểm văn của cả lớp.*
> - *Tính trung bình điểm toán của cả lớp.*
> - *Sắp xếp học sinh trong lớp theo thứ tự họ tên.*
> - *Sắp xếp học sinh trong lớp theo thứ tự không giảm của điểm trung bình, nếu điểm trung bình bằng nhau thì sắp xếp theo tên.*

II.8. 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.

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;
     }
… các phương thức khác
}
```

Ví dụ 2: bài toán tháp Hà Nội, minh họa dùng tham chiếu *this* trong đệ qui.

```
using System;

class Cot
{
     uint [] Disks;
     uint Size, Top = 0;
     char Ten;

     public Cot(uint n, char TenCot)
     {
          Size = n;
          Ten = TenCot;
          Disks = new uint[Size];
     }

     public void ThemDia(uint DiskID)
     {
```

```
            Disks[Top] = DiskID;
            Top++;
      }

//    ~Cot()     {Console.WriteLine("Freeing {0}", Ten);}

// Chuyen mot dia tren dinh tu cot hien hanh sang cot C
// Ham nay sư dung tham chieu this de chuong trinh duoc ro
rang hon
      public void Chuyen1Dia(Cot C)
      {
            this.Top--;
            Console.WriteLine("Chuyen  dia  {0}  tu  {1}  sang
            {2}",     this.Disks[Top], this.Ten, C.Ten);

            //C.ThemDia(this.Disks[this.Top]);
            C.Disks[C.Top] = this.Disks[this.Top];
            C.Top++;
      }
// Chuyen N dia (SoDia) tu cot hien hanh sang cot C, lay
cot B lam trung //gian. Ham nay sư dung tham chieu this voi
muc dich de qui
      public void ChuyenNhieuDia(uint SoDia, Cot B, Cot C)
      {
            // Chuyen mot dia sang C
            if( SoDia == 1) Chuyen1Dia(C);
            else
            {
                  ChuyenNhieuDia(SoDia-1,C, B);
                  Chuyen1Dia(C);
                  B.ChuyenNhieuDia(SoDia -1,this, C);
            }
      }
}

class This_ThapHaNoiApp
{
      static void Main()
      {
            Cot A, B, C;
            uint SoDia = 4;
            A = new Cot(SoDia, 'A');
            B = new Cot(SoDia, 'B');
            C = new Cot(SoDia, 'C');

            //Nap N dia vao cot A
            uint i = SoDia;
            while(i>0)
            {
                  A. ThemDia(i);
                  i--;
```

```
            }

//Chuyen N dia tu cot A sang cot C, lay cot B lam trung
gian
            A.ChuyenNhieuDia(SoDia,B, C);
            Console.ReadLine();
      }
}
```

II.9. Đóng gói dữ liệu với thuộc tính (property)

Thuộc tính là một đặc tính mới được giới thiệu trong ngôn ngữ **C#** làm tăng sức mạnh của tính đóng gói. Thuộc tính đơn giản là các phương thức lấy giá trị (***get***) và gán giá trị (***set***). Nó cho phép truy cập đến các thành phần dữ liệu của đối tượng ở mức độ đọc hoặc ghi hoặc cả hai và che dấu cài đặt thực sự bên trong lớp. Một thuộc tính thường quản lý một biến dữ liệu của lớp và thuộc tính đó có thể là:

- Chỉ đọc (*read-only*): chỉ có phương thức ***get***. Ta chỉ được đọc giá trị của thuộc tính.
- Chỉ ghi (*write-only*): chỉ có phương thức ***set***. Ta chỉ được gán (ghi dữ liệu) giá trị cho thuộc tính.
- Vừa đọc vừa ghi (*read/write*): có cả hai phương thức ***get*** và ***set***. Được phép đọc và ghi giá trị.

Để khai báo một thuộc tính, chúng ta viết kiểu thuộc tính và tên theo sau bởi cặp {}. Bên trong cặp {} chúng ta có thể khai báo các phương thức ***get*** hay ***set***.

Cú pháp định nghĩa một thuộc tính:

```
public KiểuTrảVề  TênThuộcTính
{
        // phương thức lấy giá trị của thuộc tính
        get
        {
                //…các câu lệnh
                return BiểuThứcTrảVề; // trả về một giá trị
        }

        //phương thức gán giá trị cho thuộc tính
        set
        {
                //…các câu lệnh
                BiếnThànhViên = value;
        }
}
```

Phần thân của phương thức *get* của thuộc tính tương tự như phần thân phương thức của lớp. Chúng trả về giá trị (hoặc tham chiếu) nào đó, thường là trả về giá

trị của biến thành viên mà thuộc tính quản lý. Khi ta truy cập đến thuộc tính thì phương thức ***get*** được gọi thực hiện.

Phương thức ***set*** của thuộc tính dùng để gán giá trị cho biến thành viên mà thuộc tính quản lý. Khi định nghĩa phương thức ***set***, ta phải sử dụng từ khóa ***value*** để biểu diễn cho giá trị dùng để gán cho biến thành viên. Khi gán một giá trị cho thuộc tính thì phương thức ***set*** tự động được gọi và tham số ẩn ***value*** chính là giá trị dùng để gán.

Ví dụ:

```
using System;

class Student
{
//Chú ý rằng tên của các biến có dấu "_" còn tên của các
thuộc tính tương ứng thì không có dấu "_".
     string _Ten ;
     float _DiemToan, _DiemTin;
     float _DiemTB;

     // constructor
     public Student()//
     {
          _Ten = "";
          _DiemToan = 0;
          _DiemTin = 0;
          _DiemTB = 0;
     }

     // thuoc tinh ten - (read/write)
     public string Ten
     {
          get {return _Ten;}
          set {_Ten = value;}
     }

     //Thuoc tinh diem toan - (read/write)
     public float DiemToan
     {
          get {return _DiemToan;}
          set
          {
               _DiemToan = value;
               _DiemTB = (_DiemToan + DiemTin)/2;
          }
     }

     //Thuoc tinh diem tin - (read/write)
     public float DiemTin
     {
```

```
            get {return _DiemTin;}
            set
            {
                  _DiemTin = value;
                  _DiemTB = (_DiemToan + DiemTin)/2;
            }
      }

      //Thuoc tinh diem tin - (read only)
      public float DiemTrungBinh
      {
            get {return _DiemTB;}
      }
}

class Student_PropertyApp
{
      static void Main(string[] args)
      {
            Student s1 = new Student();
            s1.Ten = "Hoa";
            s1.DiemToan = 5;
            s1.DiemTin = 7;
            //s1.DiemTrungBinh = 6;  ---> loi
            Console.WriteLine("Ten: {0}, diem Toan: {1}, diem
            Tin:  {2},  diem  trung  binh:  {3}",  s1.Ten,
            s1.DiemToan, s1.DiemTin, s1.DiemTrungBinh);

            Console.ReadLine();
      }
}
```

Trong ví dụ trên, ***Ten, DiemToan, DiemTin*** là các thuộc tính vừa đọc vừa ghi và ***DiemTrungBinh*** là thuộc tính chỉ đọc. Chúng phủ lên các thành phần dữ liệu tương ứng là ***_Ten, _DiemToan, _DiemTin, _DiemTB***, giúp người thiết kế che dấu cài đặt thực sự bên trong lớp. Đặc biệt là các thuộc tính ***DiemToan, DiemTin, DiemTrungBinh,*** chúng không cho phép người dùng gán giá trị cho biến ***_DiemTB*** và che dấu cách tính điểm trung bình.

Khi ta gọi các lệnh:

```
s1.Ten = "Hoa";
s1.DiemToan = 5;
s1.DiemTin = 7;
```

thì phương thức ***set*** của các thuộc tính ***Ten, DiemToan, DiemTin*** tương ứng được gọi và tham số ẩn ***value*** lần lượt là ***"Hoa", 5,*** *7*.

Khi ta gọi các lệnh:

```
Console.Write("Ten:  {0},  diem  Toan:  {1},  ",  s1.Ten,
s1.DiemToan);
```

```
Console.WriteLine(" diem Tin: {0}, diem TB: {1}",
s1.DiemTin, s1.DiemTrungBinh);
```

thì phương thức ***get*** của các thuộc tính ***Ten, DiemToan, DiemTin, DiemTrungBinh*** tương ứng được gọi.

Nếu ta gọi lệnh:

```
s1.DiemTrungBinh = 6;
```

thì trình biên dịch sẽ báo lỗi vì thuộc tính ***DiemTrungBinh*** không cài đặt phương thức ***set***.

Bài tập 1: *Viết chương trình xây dựng lớp TamGiac với dữ liệu là 3 cạnh của tam giác. Xây dựng các thuộc tính (property) ChuVi, DienTich và các phương thức kiểm tra kiểu của tam giác (thường, vuông, cân, vuông cân, đều).*

Bài tập 2: *Viết chương trình xây dựng lớp HinhTruTron (hình trụ tròn) với dữ liệu chiều cao và bán kính. Xây dựng các thuộc tính (property) DienTichDay (diên tích mặt đáy), DienTichXungQuanh (diên tích mặt xung quanh), TheTich (thể tích).*

II.10. Toán tử (operator)

Trong C#, *toán tử* là một phương thức tĩnh dùng để quá tải một phép toán nào đó trên các đối tượng. Mục đích của toán tử là để viết mã chương trình gọn gàng, dễ hiểu hơn, thay vì phải gọi phương thức.

Ta có thể quá tải các toán tử sau:

- Toán học: +, -, *, /, %.
- Cộng trừ một ngôi: ++, --, -.
- Quan hệ so sánh: ==, !=, >, <, >=, <=.
- Ép kiểu: ().
- …

Cú pháp khai báo nguyên mẫu của một toán tử T:

public static KiểuTrảVề operator T (CácThamSố)

{

///các câu lệnh trong thân toán tử

}

Chú ý:

- Tham số của toán tử phải là tham trị (không dùng các từ khóa *ref, out*).
- Không được quá tải toán tử = (gán), && , || (and, or logic), ?: (điều kiện), checked, unchecked, new, typeof, as, is.

- [] không được xem là một toán tử.
- Khi quá tải các toán tử dạng: +, -, *, / , % thì các toán tử +=, -=, *=, /= , %= cũng tự động được quá tải.
- Khi quá tải toán tử thì nên quá tải theo cặp đối ngẫu. Chẳng hạn, khi quá tải toán tử == thì quá tải thêm toán tử !=…
- Khi quá tải toán tử ==, != thì nên phát triển thêm các phương thức *Equals()*, *GetHashCode()* để đảm bảo luật *"hai đối tượng bằng nhau theo toán tử == hoặc phương thức Equals sẽ có cùng mã băm"*.
- Khi định nghĩa toán tử ép kiểu ta phải chỉ ra đây là toán tử ép kiểu ngầm định (implicit) hay tường minh (explicit).

Cú pháp định nghĩa toán tử ép kiểu:

public static [implicit | explicit] operator KiểuTrảVềT (Type V)

trong đó ***Type V*** là biến cần ép sang kiểu ***KiểuTrảVềT***.

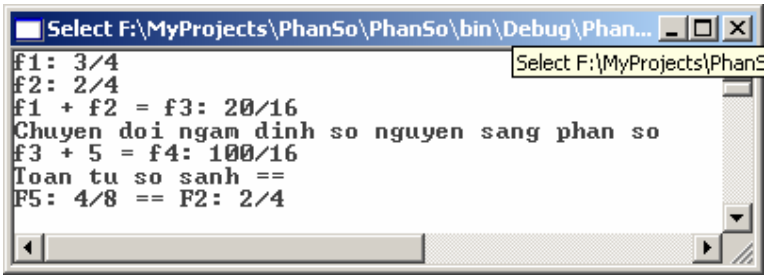
Ví dụ: xây dựng lớp phân số và quá tải các phép toán trên phân số.

```
using System;

class PhanSo
{
     int Tu, Mau;    // private members

     //constructor
     public PhanSo(int TuSo, int MauSo)
     {
          Tu = TuSo;
          Mau = MauSo;
     }

     //constructor
     public PhanSo(int HoleNumber)
     {
          Tu = HoleNumber;
          Mau = 1;
     }

     //constructor
     public PhanSo()
     {
          Tu = 0;
          Mau = 1;
     }

     //Chuyen doi ngam dinh tu so nguyen sang phan so
     public  static  implicit  operator  PhanSo(int  theInt)

     {
```

```
            Console.WriteLine("Chuyen doi ngam dinh tu so
            nguyen sang phan so");
            return new PhanSo(theInt);
        }

        //Chuyen doi tuong minh phan so sang so nguyen;
        public static explicit operator int(PhanSo PS)
        {
            return PS.Tu/PS.Mau;
        }

        //toan tu so sanh ==
        public static bool operator==(PhanSo PS1, PhanSo PS2)
        {
            return (PS1.Tu * PS2.Mau == PS2.Tu * PS1.Mau);
        }

        // Toan tu so sanh !=;
        public static bool operator!=(PhanSo PS1, PhanSo PS2)
        {
            return !(PS1 == PS2);
        }

        // phong thuc so sanh 2 phan so co bang nhau hay khong
        public override bool Equals(object o)
        {
            Console.WriteLine("Phuong thuc Equals");
            if (! (o is PhanSo) )    return false;
            return this == (PhanSo) o;
        }

        //Toan tu cong hai phan so
         public static PhanSo operator+(PhanSo PS1, PhanSo
PS2)
        {
            int MauMoi = PS1.Mau * PS2.Mau ;
            int TuMoi = PS2.Mau * PS1.Tu + PS1.Mau * PS2.Tu;
            return new PhanSo(TuMoi, MauMoi);
        }

        // Tang phan so them mot don vi!
        public static PhanSo operator++(PhanSo PS)
        {
            PS.Tu = PS.Mau +  PS.Tu;
            return PS;
        }

        //ep phan so ve gia tri True, false de tra loi cau
doi: day co phai la mot phan so hop le hay khong
        public static implicit operator bool(PhanSo PS)
        {
```

```
            return PS.Mau !=0;
      }

      //Phuong thuc doi phan so thanh chuoi
      public override string ToString()
      {
            String s = Tu.ToString( ) + "/" + Mau.ToString(
);
            return s;
      }
}
class PhanSoApp
{
      static void Main( )
      {
            PhanSo f1 = new PhanSo(3,4);
            Console.WriteLine("f1: {0}", f1.ToString( ));

            PhanSo f2 = new PhanSo(2,4);
            Console.WriteLine("f2: {0}", f2.ToString( ));

            PhanSo f3 = f1 + f2;
            Console.WriteLine("f1   +   f2   =   f3:   {0}",
f3.ToString());

            PhanSo f4 = f3 + 5;
            Console.WriteLine("f3   +   5   =   f4:   {0}",
f4.ToString( ));

            PhanSo f5 = new PhanSo(4,8);
            if (f5 == f2)
            {
                  Console.WriteLine("F5:  {0}  ==  F2:  {1}",
                        f5.ToString( ), f2.ToString( ));
            }
            Console.ReadLine();
      }
}
```

Kết quả của chương trình:



Bài tập 1: *Xây dựng lớp SoPhuc (số phức) với các phương thức và các toán tử +, -, *, / , ép sang kiểu số thực…*
Bài tập 2: *Xây dựng lớp MaTranVuong (ma trận vuông) với các phương thức và các toán tử +, -, *, / , ép sang kiểu số thực (trả về định thức của ma trận)…*

II.11. Indexer (Chỉ mục)

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]
{
      //Hàm đọc
      get
      {
            //thân hàm đọc
      }
      // Hàm ghi
      set
      {
            //thân hàm ghi
      }
}
```

Ví dụ 1: *Định nghĩa lớp mảng và dùng indexer để truy cập trực tiếp đến các phần tử của mảng:*

```
using System;
class ArrayIndexer
{
      int [] myArray;
      int Size;

      public ArrayIndexer (int n)
      {
            Size = n;
            myArray = new int[Size];
      }

// chi muc cho phép truy cập đến phần tử thứ index trong
mảng.
      public int this [int index]        {
            get
            {
                  // Kiểm tra ngoài mảng
```

```
                if (index < 0 || index >= Size)return 0;
                //throw                                    new
IndexOutOfRangeException();
                else return myArray[index];
            }
            set
            {
                if (!(index < 0 || index >= Size))
                    myArray[index] = value;
            }
        }
}

public class MainClass
{
    public static void Main()
    {
        ArrayIndexer b = new ArrayIndexer(10);
        b[3] = 256;
        b[4] = 1024;
        for (int i=0; i<5; i++)
        {
            Console.WriteLine("b[{0}] = {1}", i, b[i]);
        }
        Console.ReadLine();
    }
}
```

Trong lớp ***ArrayIndexer*** trên ta định nghĩa một chỉ mục trả về kiểu ***int*** và có một tham số ***index*** là chỉ số của phần tử cần truy cập trong mảng.

```
public int this [int index]
```

Phương thức ***get*** lấy giá trị của phần tử thứ ***index*** , phương thức ***set*** gán giá trị cho phần tử thứ ***index*** trong mảng.

Ví dụ 2:

Giả sử ta cần định nghĩa một lớp có nhiệm vụ thao tác trên file như thao tác trên một mảng các byte. Lớp này hữu dụng khi cần thao tác trên một file rất lớn mà không thể đưa toàn bộ file vào bộ nhớ chính tại một thời điểm, nhất là khi ta chỉ muốn thay đổi một vài byte trong file. Ta định nghĩa lớp **`FileByteArray`** có chức năng như trên. Chỉ mục

```
public byte this[long index]
{
    //khối kệnh truy cập đến phần tử thứ index
}
```

có chức năng truy cập tới byte thứ **index**.

Lớp **`ReverseApp`** dùng lớp **`FileByteArray`** để đảo ngược một file.

```
using System;
using System.IO;

// Lop cho phep truy cap den file nhu la mot mang cac bye
public class FileByteArray
{
     Stream st; //stream dung de truy cap den file

     //Phuong thuc khoi tao: mo fileName va lien ket luong
stream toi file
     public FileByteArray(string fileName)
     {
          //Mo file
          st = new FileStream(fileName, FileMode .Open);
     }

     // Dong luong (stream) sau khi da thao tac xong tren
file
     public void Close()
     {
          st.Close();
          st = null;
     }

     // Indexer de truy cap (doc/ghi) den byte thu index
cua file
     public byte this[long index]
     {
          // Doc mot byte tai vi tri index
          get
          {
               //Tao vung dem de doc ghi
               byte[] buffer = new byte[1];

               //Dich chuyen con tro den vi tri index trong
file
               st.Seek(index, SeekOrigin .Begin);

               //Doc 1 byte, tai vi tri hien hanh cua con
tro doc  (offset = 0, count = 1)
               st.Read(buffer, 0, 1);
               return buffer[0];
          }

          // Ghi mot byte và file tai vi tri index .
          set
          {
               //Gan gia tri can ghi vao buffer
               byte[] buffer = new byte[1] {value};
```

```
                //Dich chuyen den vi tri can ghi trong luong
                st.Seek(index, SeekOrigin.Begin);

                //Ghi du lieu vao file
                st.Write(buffer, 0, 1);
            }
        }

        // Lay chieu dai cua file (so byte)
        public long Length
        {
            get
            {
                return st.Seek(0, SeekOrigin.End);
                // Ham seek tra ve vị trí hiện tại của con
            trỏ.
            }
        }
}

public class DaoNguocFileApp
{
        public static void Main()
        {
            FileByteArray          file          =          new
FileByteArray("F:\\data.txt");
            long len = file.Length;
            long i;
            // dao nguoc file
            for (i = 0; i < len / 2; ++i)
            {
                byte t;
                t = file[i];
                file[i] = file[len - i - 1];
                file[len - i - 1] = t;
            }
            //Xuat file
            for (i = 0; i < len; ++i)
            {
                Console.Write((char)file[i]);
            }
            file.Close();
            Console.ReadLine();
        }
}
```

Trong ví dụ trên ***chỉ mục*** trả về kiểu ***byte*** và có một tham số ***index*** là vị trí cần truy cập trong file (kiểu long).

```
public byte this[long index]
```

Phương thức ***get*** định nghĩa các dòng lệnh để đọc một byte từ file, phương thức ***set*** định nghĩa các dòng lệnh để ghi một byte vào file.

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).

II.12. Lớp lồng nhau

Lớp định nghĩa bên trong một lớp khác gọi là *inner class*, lớp nằm ngoài gọi là *outer class*.

Các phương thức của lớp nằm trong có thể truy cập đến các thành phần private của lớp nằm ngoài (nhưng phải thông qua một đối tượng nào đó).

II.13. Câu hỏi ôn tập

1. *Từ khoá nào được sử dụng trong khai báo dữ liệu của lớp?*
2. *Sự khác nhau giữa thành viên được khai báo là public và các thành viên không được khai báo là public?*
3. *Lớp mà chúng ta xây dựng thuộc kiểu dữ liệu nào?*
4. *Có phải tất cả những dữ liệu thành viên luôn luôn được khai báo là public để bên ngoài có thể truy cập chúng?*
5. *Có thể tạo phương thức bên ngoài của lớp hay không?*
6. *Sự khác nhau giữa một lớp và một đối tượng của lớp?*
7. *Thành viên nào trong một lớp có thể được truy cập mà không phải tạo thể hiện của lớp?*
8. *Khi nào thì phương thức khởi tạo được gọi?*
9. *Khi nào thì phương thức khởi tạo tĩnh được gọi?*
10. *Phương thức tĩnh có thể truy cập được thành viên nào và không truy cập được thành viên nào trong một lớp?*
11. *Có thể truyền biến chưa khởi tạo vào một hàm được không?*
12. *Sự khác nhau giữa truyền biến tham chiếu và truyền biến tham trị vào một phương thức?*
13. *Làm sao để truyền tham chiếu với biến kiểu giá trị vào trong một phương thức?*
14. *Từ khóa **this** được dùng làm gì trong một lớp?*

15. Cú pháp định nghĩa một thuộc tính?

16. Cú pháp định nghĩa một toán tử?

II.14. Bài tập tổng hợp

1. *Xây dựng một lớp đường tròn lưu giữ bán kính và tâm của đường tròn. Tạo các phương thức để tính chu vi, diện tích của đường tròn.*
2. *Thêm thuộc tính BanKinh vào lớp được tạo ra từ bài tập 1.*
3. *Viết lớp giải phương trình bậc hai. Lớp này có các thuộc tính a, b, c và nghiệm x1, x2. Lớp cho phép bên ngoài xem được các nghiệm của phương trình và cho phép thiết lập hay xem các giá trị a, b, c.*
4. *Xây dựng lớp đa thức với các toán tử +, -, *, / và chỉ mục để truy cập đến hệ số thứ i của đa thức.*
5. ***X**ây dựng lớp ma trận với các phép toán +, -, *, / và chỉ mục để truy cập đến phần tử bất kỳ của ma trận.*
6. ***X**ây dựng lớp NguoiThueBao (số điện thoại, họ tên), từ đó xây dựng lớp DanhBa (danh bạ điện thoại) với các phương thức như nhập danh bạ điện thoại, xuất danh bạ điện thoại, tìm số điện thoại theo tên (chỉ mục), tìm tên theo số điện thoại (chỉ mục).*

III. Kế thừa (inheritance) và đa hình (polymorphism)

Phần I, II trình bày cách tạo một kiểu dữ liệu mới bằng các định nghĩa lớp. Việc định nghĩa một lớp thể hiện tính đóng gói của phương pháp lập trình hướng đối tượng. Trong phần này ta tìm hiểu mối quan hệ giữa các đối tượng trong thế giới thực và cách thức mô hình hóa những quan hệ này trong mã chương trình dựa trên khái niệm kế thừa. Các mối quan hệ này được biểu diễn thông qua tính kế thừa và tính đa hình.

III.1. Quan hệ chuyên biệt hóa và tổng quát hóa

Các lớp và thể hiện của lớp không tồn tại trong một không gian độc lập, chúng tồn tại trong một mạng các quan hệ và phụ thuộc qua lại lẫn nhau.

Quan hệ tổng quát hóa và chuyên biệt hóa là quan hệ phân cấp và tương hỗ lẫn nhau (tương hỗ vì chuyên biệt hóa là mặt đối lập với tổng quát hóa). Và những quan hệ này là phân cấp vì chúng tạo ra cây quan hệ. Chẳng hạn, quan hệ *is-a* (*là một*) là một sự chuyên biệt hóa. Ví dụ, khi ta nói "*Sơn dương là một loài động vật, đại bàng cũng là một loài động vật* ", thì có nghĩa là: "*Sơn dương và đại bàng là những loại động vật chuyên biệt, chúng có những đặc điểm chung của động vật và ngoài ra chúng có những đặc điểm phân biệt nhau*". Và như vậy, ***động vật*** là tổng quát hóa của ***sơn dương*** và ***đại bàng***; ***sơn dương*** và ***đại bàng*** là chuyên biệt hóa của ***động vật***.

Trong C#, quan hệ chuyên biệt hóa, tổng quát hoá thường được thể hiện thông qua sự kế thừa. Bởi vì, thông thường, khi hai lớp chia sẻ chức năng, dữ liệu với nhau, ta trích ra các phần chung đó và đưa vào lớp cơ sở chung để có thể nâng cao khả năng sử dụng lại các mã nguồn chung, cũng như dễ dàng quản lý mã nguồn.

III.2. Kế thừa

Kế thừa là cơ chế cho phép định nghĩa một lớp mới (còn gọi là lớp dẫn xuất, drived 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 các phương thức private, phương thức khởi tạo, phương thức hủy và phương thức tĩnh). Nói cách khác, lớp dẫn xuất sẽ kế thừa hầu hết các thành viên của lớp cơ sở.

Một điều cần chú ý rằng, lớp dẫn xuất vẫn được kế thừa các thành phần dữ liệu private của lớp cơ sở nhưng không được phép truy cập trực tiếp (truy cập gián tiếp thông qua các phương thức của lớp cơ sở).

Cú pháp định nghĩa lớp dẫn xuất:

```
class TênLớpCon : TênLớpCơSở
{
        // Thân lớp dẫn xuất
}
```

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

```
using System;

//Lop co so Point2D
class Point2D
{
	public int x,y;
	public void Xuat2D()
	{
		Console.WriteLine("({0}, {1})", x, y);
	}
}

//Lop dan xuat Point3D ke thua tu lop Point2D
class Point3D:Point2D
{
	public int z;
	public void Xuat3D()
	{
		Console.WriteLine("({0}, {1}, {2})", x, y, z);
	}
}

class PointApp
{
	public static void Main()
	{
		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();
		Console.ReadLine();
	}
}
```

Một thực thể (đối tượng) của lớp dẫn xuất có tất cả các trường (biến) được khai báo trong lớp dẫn xuất và các trường đã được khai báo trong các lớp cơ sở mà nó kế thừa. Ở ví dụ trên, rõ ràng trong lớp ***Point3D*** ta không khai báo các biến ***x, y*** nhưng trong phương thức ***Xuat3D()*** ta vẫn có thể truy cập ***x, y***. Thậm chí trong hàm ***Main()***, ta 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***.

Chú ý:

- Lớp dẫn xuất không thể bỏ đi các thành phần đã được khai báo trong lớp cơ sở.
- Các hàm trong lớp dẫn xuất không được truy cập trực tiếp đến các thành viên có mức độ truy cập là ***private*** trong lớp cơ sở.

Ví dụ:

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

ClassA.x is inaccessible due to its protection level.

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

class ClassB: ClassA
{
      public void GiamX()
      {
            x = x - 1;      // Loi.
      }
}
```

Nếu sửa lại khai báo ***int x = 5;*** thành ***protected int x = 5;*** hoặc ***public int x = 5;*** thì sẽ không còn lỗi trên vì thành phần *protected* hoặc *public* của lớp cơ sở có thể được truy cập trực tiếp trong lớp dẫn xuất (nhưng không được truy cập trong một phương thức không thuộc lớp cơ sở và lớp dẫn xuất).

III.3. Gọi phương thức tạo lập của lớp cơ sở

Vì lớp dẫn xuất không thể kế thừa phương thức tạo lập của lớp cơ sở nên một lớp dẫn xuất phải thực thi phương thức tạo lập riêng của mình. Nếu lớp cơ sở có một phương thức tạo lập mặc định (tức là không có phương thức tạo lập hoặc phương thức tạo lập không có tham số) thì phương thức tạo lập của lớp dẫn xuất được định nghĩa như cách thông thường. Nếu lớp cơ sở có phương thức tạo lập có tham số thì lớp dẫn xuất cũng phải định nghĩa phương thức tạo lập có tham số theo cú pháp sau:

```
TênLớpCon(ThamSốLớpCon): TênLớpCơSở(ThamSốLớpCha)
{
      // Khởi tạo giá trị cho các thành phần của lớp dẫn xuất
}
```

Chú ý: ***ThamSốLớpCon*** phải bao ***ThamSốLớpCha.***

Ví dụ:

```
using System;

//Lop co so
class Point2D
{
     public int x,y;
     //phuong thuc tao lap cua lop co so co tham so
     public Point2D(int a, int b)
     {
          x = a; y = b;
     }
     public void Xuat2D()
     {
          Console.Write("({0}, {1})", x, y);
     }
}

//Lop dan xuat
class Point3D:Point2D
{
     public int z;

     //Vi phuong thuc tao lap cua lop co so co tham so nen
phuong thuc tao lap cua lop dan xuat cung phai co tham so
     public Point3D(int a, int b, int c):base (a,b)
     {
          z = c;
     }
     public void Xuat3D()
     {
          Console.Write("({0}, {1}, {2})", x, y, z);
     }
}

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

Trong ví dụ trên, vì phương thức tạo lập của lớp ***Point2D*** có tham số:

```
public Point2D(int a, int b)
```

nên khi lớp ***Point3D*** dẫn xuất từ lớp ***Point2D,*** phương thức tạo lập của nó cần có ba 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***. Phương thức tạo lập có nguyên mẫu như sau:

```
public Point3D(int a, int b, int c):base (a, b)
```

Phương thức tạo lập này gọi phương thức tạo lập của lớp cơ sở ***Point2D*** bằng cách đặt dấu ":" sau danh sách tham số và từ khoá ***base*** với các đối số tương ứng với phương thức tạo lập của lớp cơ sở.

III.4. Định nghĩa phiên bản mới trong lớp dẫn xuất

Qua những phần trên chúng ta có nhận xét rằng, 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.

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 '"
```

bởi vì, 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. Điều này có tác dụng che dấu thành phần kế thừa đó đối với các phương thức bên ngoài lớp dẫn xuất. 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.

Ví dụ:

```
using System;

class XeHoi
{
     //Cac thanh phan nay la protected de phuong thuc Xuat
     cua lop XeXat va XeHoi co the truy cap duoc
```

```
        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.Write("Xe: {0}, Bien so: {1}, Toc do: {2}
            kmh",HangSX, BienSo, TocDo);
        }
}

class  XeCar: XeHoi
{
        int SoHanhKhach;

        public XeCar(int td, string BS, string HSX, int SHK):
        base(td, BS, HSX)
        {
            SoHanhKhach = SHK;
        }

        //Tu khoa new che dau phuong thuc Xuat cua lop XeHoi
vi phuong thuc Xuat cua lop XeHoi khong con phu hop voi lop
XeCar.
        public new void Xuat()
        {
        // Goi phuong thuc xuat cua lop co so thong qua tu
khoa base
            base.Xuat();
            Console.WriteLine(", {0} cho ngoi", SoHanhKhach);
        }
}

class  XeTai: XeHoi
{
        int TrongTai;

        public XeTai(int td, string BS, string HSX, int TT):
        base(td, BS, HSX)
        {
            TrongTai = TT;
        }

//Tu khoa new che dau phuong thuc Xuat cua lop XeHoi vi
phuong thuc Xuat cua lop XeHoi khong con phu hop voi lop
XeCar nua.
```

```
     public new void Xuat()
     {
          base.Xuat(); // Goi phuong thuc xuat cua lop co
               Console.WriteLine(",  trong  tai  {0}  tan",
TrongTai);
     }
}

public class Test
{
     public static void Main()
     {
          XeCar  c  =  new  XeCar(150,"49A-4444",  "Toyota",
24);
          c.Xuat();
          XeTai t = new XeTai(150,"49A-5555", "Benz", 12);
          t.Xuat();
          Console.ReadLine();
     }
}
```

Trong ví dụ trên, lớp ***XeHoi*** có một phương thức ***Xuat( )*****,** tuy nhiên phương thức này chỉ xuất ra các thông tin như **BienSo**, **TocDo**, **HangSX** nên không còn phù hợp với hai lớp ***XeTai*** và ***XeCar*** nữa. Do đó hai lớp dẫn xuất này định nghĩa một phiên bản mới của phương thức ***Xuat()*** theo cú pháp sau:

```
public new void Xuat( )
{
     …
}
```

Việc thêm vào từ khoá ***new*** chỉ ra rằng người lập trình muốn tạo ra một phiên bản mới của phương thức này trong các lớp dẫn xuất nhằm che dấu phương thức đã kế thừa từ lớp cơ sở ***XeHoi***. Như vậy, trong hàm **Main(),** khi gọi:

```
c.Xuat();
```

hoặc

```
t.Xuat();
```

trình biên dịch sẽ hiểu rằng đây là phương thức ***Xuat()*** của lớp ***XeTai*** hoặc lớp ***XeCar***.

Hơn nữa, trong phương thức ***Xuat()*** của lớp ***XeTai*** và ***XeCar*** ta vẫn có thể gọi phương thức ***Xuat()*** của lớp ***XeHoi*** bằng câu lệnh:

```
base.Xuat();
```

III.5. 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. Với các lớp ***XeHoi***, ***XeCar*** như trên, ta có thể định nghĩa hàm ***Main()*** như sau:

```
public static void Main()
{
      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 thuoclop dan xuat XeCar");
      Console.WriteLine("Nhung chi co the goi ham xuat tuong
      ung voi XeHoi");
      XeHoi h = c;
      h.Xuat();
      Console.ReadLine();
}
```

Kết quả chạy chương trình:

Khi gọi lệnh

```
c.Xuat();
```

trình biên dịch gọi phương thức ***Xuat()*** của lớp ***XeCar*** và xuất các thông tin: *hàng sản xuất (Toyota), biển số (49A-444), tốc độ tối đa (150 km/h), 24 chỗ ngồi*.

Sau đó một đối tượng *h* thuộc lớp cơ sở *XeHoi* trỏ đến đối tượng *c* thuộc lớp dẫn xuất :

```
XeHoi h = c;
```

Khi gọi lệnh

```
h.Xuat();
```

trình biên dịch sẽ thực hiện phương thức *Xuat()* của lớp ***XeHoi*** nên chỉ xuất các thông tin: *hàng sản xuất (Toyota), biển số (49A-444), tốc độ tối đa (150 km/h)*.

> ***Bài tập 1:*** *Xây dựng lớp Stack và lớp Queue (cài đặt theo kiểu danh sách liên kết) bằng cách đưa những thành phần dữ liệu và phương chung của hai lớp này vào một lớp cơ sở SQ và từ đó xây dựng các lớp Stack, Queue kế thừa từ lớp SQ.*

Bài tập 2: *Xây dựng lớp hình tròn với các thuộc tính (properties): bán kính, đường kính, diện tích.*

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*R2*
- *Thể tích hình cầu tính bán kính R được tính theo công thức 4/3*PI*R^3*

Bài tập 3: *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.*

III.6. Phương thức ảo (virtual method) và tính đa hình (polymorphism)

Hai đặc điểm mạnh nhất của kế thừa đó là khả năng sử dụng lại mã chương trình và đa hình (polymorphism). Đa hình là ý tưởng "sử dụng một giao diện chung cho nhiều phương thức khác nhau", dựa trên phương thức ảo (virtual method) và cơ chế liên kết muộn (late binding). Nói cách khác, đây là cơ chế cho phép gởi một loại thông điệp tới nhiều đối tượng khác nhau mà mỗi đối tượng lại có cách xử lý riêng theo ngữ cảnh tương ứng của chúng.

Đây là một kịch bản thực hiện tính đa hình:

Lớp của các đối tượng nút nhấn (*button*), nhãn (*label*), nút chọn (*option button*), danh sách sổ xuống (*combobox*)… đều kế thừa từ lớp *Window* và đều có các phương thức vẽ (hiển thị) riêng của mình lên form. Một form có thể có nhiều đối tượng như trên và được lưu trong một danh sách (không cần biết các đối tượng trong danh sách là *ListBox* hay *Button*... miễn là đối tượng đó là một thể hiện *Window*). Khi form được mở, nó có thể yêu cầu mỗi đối tượng *Window* tự vẽ lên form bằng cách gởi thông điệp vẽ đến từng đối tượng trong danh sách và các đối tượng này sẽ thực hiện chức năng vẽ tương ứng. Khi đó ta muốn *form* xử lý tất cả các đối tượng *Window* theo đặc trưng đa hình.

Để thực hiện được đa hình ta phải thực hiện các bước sau:

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.

Nếu tham chiếu này trỏ tới đối tượng thuộc lớp dẫn xuất thì phương thức ảo đã được lớp dẫn xuất định nghĩa lại được thực hiện.

Ví dụ:

```
using System;
public class MyWindow
{
     protected int top, left; //Toa do goc tren ben trai
     public MyWindow(int t, int l)
     {
          top = t;
          left = l;
     }

     // Phuong thuc ao
     public virtual void DrawWindow( )
     {
          Console.WriteLine("...dang ve Window tai toa do
          {0}, {1}",     top, left);
     }
}

public class MyListBox : MyWindow
{
     string listBoxContents;
     public  MyListBox(int  top,int  left,string  contents):
     base(top, left)
     {
          listBoxContents = contents;
     }

     public override void DrawWindow( )
     {
          Console.WriteLine  ("...dang  ve  listbox  {0}  tai
          toa do: {1},{2}", listBoxContents, top, left);
     }
}

public class MyButton : MyWindow
{
     public MyButton(int top,int left):base(top, left) {}
     public override void DrawWindow( )
     {
          Console.WriteLine ("...dang ve button tai toa do:
          {0},{1}", top, left);
     }
}

public class Tester
{
     static void Main( )
```

```
    {
        Random R = new Random();
        int t;
        string s = "";
        MyWindow[] winArray = new MyWindow[4];
        for (int i = 0;i < 4; i++)
        {
            t = R.Next() % 3;
            switch(t)
            {
                case 0:
                    winArray[i] =new MyWindow( i * 2,
                    i * 4); break;
                case 1:
                    s = "thu " +  (i+1).ToString();
                    winArray[i] = new MyListBox(i*3, i
* 5, s);
                    break;
                case 2:
                    winArray[i] =new MyButton(i * 10,
                    i * 20); break;
            }
        }

    for (int i = 0; i < 4; i++)
        {
            winArray[i].DrawWindow( );
        }
        Console.ReadLine();
    }
}
```

Trong ví dụ này ta xây dựng một lớp ***MyWindow*** có một phương thức ảo:

```
public virtual void DrawWindow( )
```

Các lớp ***MyListBox, MyButton*** kế thừa từ lớp ***MyWindow*** và định nghĩa lại **(`override`) phương thức** `DrawWindow()` theo cú pháp:

```
public override void DrawWindow( )
```

Sau đó trong hàm *Main ()* ta khai báo và tạo một mảng các đối tượng ***MyWindow***. Vì mỗi phần tử thuộc mảng này là một tham chiếu thuộc lớp ***MyWindow*** nên nó có thể trỏ tới bất kỳ một đối tượng nào thuộc các lớp kế thừa lớp ***MyWindow***, chẳng hạn lớp ***MyListBox*** hay lớp ***MyButton***.

Vòng lặp for đầu tiên tạo ngẫu nhiên các đối tượng thuộc một trong các lớp ***MyWindow***, ***MyListBox***, ***MyButton***, vì vậy, tại thời điểm biên dịch chương trình, trình biên dịch không biết đối tượng thứ i thuộc lớp nào và do đó chưa thể xác định được đoạn mã của phương thức ***`DrawWindow()`*** cần gọi. Tuy nhiên, tại thời điểm chạy chương trình, sau vòng lặp for đầu tiên, mỗi ***`winArray[i]`*** tham chiếu tới một loại đối tượng cụ thể nên trình thực thi sẽ tự động xác định được phương

thức ***DrawWindow()*** cần gọi. (Như vậy ta đã sử dụng một giao diện chung là ***DrawWindow()*** cho nhiều phương thức khác nhau).

Chú ý rằng nếu phương thức **DrawWindow()** trong các lớp *MyListBox, MyButton,..* không có từ khóa ***override*** như cú pháp:

```
public override void DrawWindow( )
```

thì trình biên dịch sẽ báo lỗi.

Ví dụ 2:

Một điểm dịch vụ cần quản lý các thông tin cho thuê xe đạp và xe máy. Với xe đạp cần lưu họ tên người thuê, số giờ thuê. Tiền thuê xe đạp được tính như sau: 10000 (đồng) cho giờ đầu tiên, 80000 cho mỗi giờ tiếp theo. Với mỗi xe máy cần lưu họ tên người thuê, số giờ thuê, loại xe (100 phân khối, 250 phân khối), biển số. Tiền thuê xe máy được tính như sau: Đối với giờ đầu tiên, loại xe 100 phân khối tính 15000; loại xe 250 phân khối tính 20000. Đối với những giờ tiếp theo tính 10000 cho cả hai loại xe máy.

Viết chương trình xây dựng các lớp cần thiết sau đó nhập danh sách các thông tin thuê xe đạp và xe máy, sau đó xuất ra các thông tin sau:

- Xuất tất cả các thông tin thuê xe (cả số tiền thuê tương ứng).
- Tính tổng số tiền cho thuê xe đạp và xe máy.
- Xuất tất cả các thông tin liên quan đến việc thuê xe đạp.
- Tính tổng số tiền cho thuê xe máy loại 250 phân khối.

Giải:

```
using System;
public class XE
{
     protected string hoten;
     protected int giothue;
     public virtual int TienThue
     {
          get {return 0;}
     }
     public virtual void Xuat()     {     }
     public virtual string ID()
     {
          return "X";
     }
}
public class XEDAP:XE
{
     public XEDAP(string ht,int gt)
     {
          hoten = ht;
          giothue = gt;
     }
```

```
	public override string ID()
	{
		return "XD";
	}
	public override int TienThue
	{
		get
		{
			int T = 10000;
			if (giothue > 0)    T = T+ 8000*(giothue-1);
			return T;
		}
	}
	public override void Xuat()
	{
		Console.WriteLine(hoten + "\t" + giothue +"\t" +
TienThue);
	}
}
public class XEMAY:XE
{
	string loaixe;
	string bienso;
	public XEMAY(string ht,int gt,string lx,string bs)
	{
		hoten = ht;
		giothue = gt;
		loaixe=lx;
		bienso=bs;
	}
	public override string ID()
	{
		if (loaixe=="100")return "XM_100";
		else return "XM_250";
	}
	public override int TienThue
	{
		get
		{
			int T;
			if (loaixe == "100")T = 15000;
			else T = 20000;
			(if (giothue > 0)    T = T + 10000*(giothue-
		1);

			return T;
		}
	}
	public override void Xuat()
	{
```

```
          Console.WriteLine("Xe may: " + hoten + "\t" +
giothue +"\t" + loaixe + "\t" + bienso +"\t"+ TienThue);
     }
}
public class CUAHANG
{
     public int n;
     XE []XT;  //Xe cho thue

     public CUAHANG(int size)
     {
          n=size;
          XT= new XE[size];
     }
     public int menu()
     {
          int chon;
          do
          {
               Console.WriteLine("******  Bang  Chon  Nhap
********");
               Console.WriteLine("1. Nhap Xe Dap");
               Console.WriteLine("2. Nhap Xe May");
               chon=int.Parse(Console.ReadLine());
          }while((chon!=1) && (chon!=2));
          return chon;
     }
     public void NhapDSXeChoThue()
     {
          int chon;
          for(int i=0;i<n;i++)
          {
               chon=menu();
               if(chon==1)
               {
                    Console.WriteLine("***** Ban chon nhap
xe dap *******");
                    Console.WriteLine("Ho     ten     nguoi
thue?");
                    string ht=Console.ReadLine();
                    Console.WriteLine("So gio thue?");
                    int gt=int.Parse(Console.ReadLine());
                    XT[i]=new XEDAP(ht,gt);
               }
               else
               {
                    Console.WriteLine("***** Ban chon nhap
xe may ********");
                    Console.WriteLine("Nguoi thue?");
                    string ht=Console.ReadLine();
                    Console.WriteLine("So gio thue?");
```

```
                    int gt=int.Parse(Console.ReadLine());
                    Console.WriteLine("Ban  nhap  vao  loai
xe(100 hoac 125 )phan khoi:" );
                    string lx=Console.ReadLine();
                    Console.WriteLine("Bien so:");
                    string bs=Console.ReadLine();
                    XT[i]=new XEMAY(ht,gt,lx,bs);
                }
            }
        }
        public void XuatDSXeThue()
        {
            for(int i=0;i<n;i++)
            {
                XT[i].Xuat();
            }
        }
        public long TongTienChoThue()
        {
            long ts=0;
            for(int i=0;i<n;i++)
            {
                ts=ts+XT[i].TienThue;
            }
            return ts;
        }
        public void XuatXeDap()
        {
            for(int i=0;i<n;i++)
            {
                if (XT[i].ID() =="XD")XT[i].Xuat();
            }
        }
        public long TongTienXeMay250()
        {
            long T = 0;
            for(int i=0;i<n;i++)
            {
                if    (XT[i].ID()    =="XM_250")    T    +=
XT[i].TienThue;
            }
            return T;
        }
}
public class App
{
        public static void Main()
        {
            CUAHANG ch=new CUAHANG(3);
            ch.NhapDSXeChoThue();
```

```
            Console.WriteLine("Xuat tat ca nhung thong tin
thue xe");
            ch.XuatDSXeThue();

            Console.WriteLine("Tong tien thue xe " +
ch.TongTienChoThue());
            Console.WriteLine("Thong tin ve xe dap cho
thue:");
            ch.XuatXeDap();

            Console.WriteLine("Tong tien thue xe may 250 phan
khoi" + ch.TongTienXeMay250());
            Console.ReadLine();
      }
}
```

III.7. Lớp Object

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.

Trong những ví dụ trước ta đã thực hiện việc ghi đè lên phương thức *ToString()* và *Equals()* của lớp ***Object***.

III.8. Lớp trừu tượng(abstract)

Trong các ví dụ về phương thức ảo trên, nếu lớp dẫn xuất không định nghĩa lại phương thức ảo của lớp cơ sở thì nó được kế thừa như một phương thức thông thường. Tức là lớp dẫn xuất không nhất thiết phải định nghĩa lại phương thức ảo. Để bắt buộc tất cả các lớp dẫn xuất phải định nghĩa lại (***override***) một phương thức của lớp cơ sở ta phải đặt từ khóa ***abstract*** trước phương thức đó và phương thức đó được gọi là phương thức trừu tượng. Trong phần thân của phương thức trừu tượng không có câu lệnh nào, nó chỉ tạo một tên phương thức và đánh dấu rằng nó phải được thi hành trong lớp dẫn xuất. Lớp chứa phương thức trừu tượng được gọi là lớp trừu tượng.

Cú pháp khai báo phương thức trừu tượng:

abstract public void TênPhươngThức( );

Phương thức trừu tượng phải được đặt trong lớp trừu tượng. Lớp trừu tượng có từ khóa ***abstract*** đứng trước.

Ví dụ: Xây dựng lớp ***HinhHoc*** với phương thức tính chu vi, diện tích là phương thức trừu tượng hoặc phương thức ảo. Sau đó định nghĩa các lớp ***HinhChuNhat*** (hình chữ nhật), ***HinhTron*** (hình tròn) kế thừa từ lớp ***HinhHọc*** với các thành phần dữ liệu và phương thức tính chu vi, diện tích cụ thể của từng loại đối tượng.

```
// lop hinh hoc (truu tuong)
abstract public class HinhHoc
{
      abstract public double DienTich();
      virtual public double ChuVi() { return 0;}
}

// lop hinh tron ke thua tu lop hinh hoc
public class HinhTron : HinhHoc
{
      double _bankinh;
      public double BanKinh
      {
            get{ return _bankinh;}
            set{ _bankinh = value;}
      }

      public override double DienTich()
      {
            return _bankinh*_bankinh*3.1416;
      }

      public override double ChuVi()
      {
            return _bankinh*2*3.1416;
      }
}

// lop hinh chu nhat ke thua tu lop hinh hoc
public class HinhChuNhat : HinhHoc
{
      double _dai, _rong;

      public double ChieuDai
      {
            get{ return _dai;}
            set{ _dai = value;}
      }

      public double ChieuRong
      {
            get{ return _rong;}
            set{ _rong = value;}
```

```
    }

    public override double DienTich(){ return _dai*_rong;}
    public  override  double  ChuVi(){return  (_dai+_rong
)*2;}
}

class Tester
{
    static void Main(string[] args)
    {
        HinhHoc h;
        HinhTron t = new HinhTron();
        t.BanKinh = 5;
        Console.WriteLine("Thong tin ve hinh tron");
        h = t;
        Console.WriteLine("Chu  vi  hinh  tron:  {0}  ",
h.ChuVi());
        Console.WriteLine("Dien  tich  hinh  tron:{0}  ",
h.DienTich());

        HinhChuNhat n = new HinhChuNhat();
        n.ChieuDai = 4;
        n.ChieuRong = 3;
        h =  n;
        Console.WriteLine("Thong tin ve hinh chu nhat ");
        Console.WriteLine("Chu  vi  hinh  chu  nhat:{0}",
h.ChuVi());
        Console.WriteLine("Dien tich hinh chu nhat:{0}",
h.DienTich());
        Console.ReadLine();
    }
}
```

Trong lớp ***HinhHoc*** ở ví dụ trên, ta khai báo phương thức tính diện tích là một phương thức trừu tượng (không có phần cài đặt mã của phương thức vì chưa biết đối tượng hình thuộc dạng nào nên không thể tính diện tích của nó). Như vậy, lớp ***HinhHoc*** là một lớp trừu tượng.

```
abstract public double DienTich();
```

Trong lớp trên, ta cũng khai báo một phương thức tính chu vi là phương thức ảo với mục đích minh họa rằng trong lớp trừu tượng có thể có phương thức ảo (hoặc bất cứ một thành phần nào khác của một lớp). Vì đây là một phương thức không trừu tượng nên phải có phần cài đặt mã bên trong thân phương thức.

Các lớp ***HinhChuNhat***, ***HinhTron*** kế thừa từ lớp ***HinhHoc*** nên chúng buộc phải cài đặt lại phương thức tính diện tích.

Trong hàm *Main()*, ta tạo ra đối tượng ***n*** thuộc lớp ***HinhChuNhat*** và đối tượng ***t*** thuộc lớp ***HinhTron.*** Khi tham chiếu ***h*** thuộc lớp ***HinhHoc*** trỏ tới đối tượng ***n*** (tương tự với đối tượng ***t***), nó có thể gọi được phương thức tính diện tích của lớp

HinhChuNhat (tương tự lớp ***HinhTron*****)**, điều này chứng tỏ phương thức trừu tượng cũng có thể dùng với mục đích đa hình.

Chú ý: Phân biệt giữa từ khóa new và override

- *Từ khóa **override** dùng để định nghĩa lại (ghi đè) phương thức ảo **(virtual)** hoặc phương thức trừu tượng **(abstract)** của lớp cơ sở, nó được dùng với mục đích đa hình.*
- *Từ khóa **new** để che dấu thành viên của lớp cơ sở trùng tên với thành viên của lớp dẫn xuất.*

III.9. Giao diện (interface)

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.

III.9.1. Thực thi giao diện

Ta dùng từ khóa ***interface*** để khai báo một giao diện với cú pháp sau:

[MứcĐộTruyCập] **interface** ***TênGiaoDiện [:CácGiaoDiệnCơSở]***
{

Nội dung
}

MứcĐộTruyCập: public hoặc internal.

CácGiaoDiệnCơSở: danh sách các interface khác mà nó kế thừa.

Về mặt cú pháp, một giao diện giống như một lớp chỉ có phương thức trừu trượng. Nó có thể chứa khai báo của phương thức, thuộc tính, chỉ mục, sự kiện (events) nhưng không được chứa biến dữ liệu. Khi kế thừa một giao diện ta phải thực thi mọi phương thức, thuộc tính,… của giao diện.

Chú ý:

- *Mặc định, tất cả các thành phần khai báo trong giao diện đều là public. Nếu có từ khóa public đứng trước sẽ bị báo lỗi. Các thành phần trong giao diện chỉ là các khai báo, không có phần cài đặt mã.*
- *Một lớp có thể kế thừa một lớp khác đồng thời kế thừa nhiều giao diện.*

Ví dụ: Mọi chiếc xe hơi hoặc xe máy đều có hành động (phương thức) khởi động và dừng. Ta có thể dùng định nghĩa một giao diện kèm thêm một thuộc tính để cho biết chiếc xe đã khởi động hay chưa:

```
public interface IDrivable
```

```
    {
        void KhoiDong();
        void Dung();
        bool DaKhoiDong
        {
            get;
        }
    }
```

Thuộc tính đã khởi động (*DaKhoiDong*) chỉ có phương thức ***get*** vì khi gọi phương thức ***KhoiDong()*** thì thuộc tính này sẽ có giá trị ***true***, khi gọi phương thức ***Dung()*** thì thuộc tính này sẽ có giá trị ***false***.

Khi đó, một lớp *Car* thực thi giao diện này phải cài đặt các phương thức và thuộc tính đã khai báo trong giao diện ***Idrivable*** trên:

```
public interface IDrivable
{
    void KhoiDong();
    void Dung();
    bool DaKhoiDong
    {
        get;
    }
}

public class Car:IDrivable
{
    public bool Started = false;
    public void KhoiDong()
    {
        Console.WriteLine("Xe ca khoi dong");
        Started = true;
    }
    public void Dung()
    {
        Console.WriteLine("Xe ca dung");
        Started = false;
    }
    public bool DaKhoiDong
    {
        get{return Started;}
    }
}
public class Tester
{
    public static void Main()
    {
        Car c = new Car();
        c.KhoiDong();
        Console.WriteLine("c.DaKhoiDong = " +
c.DaKhoiDong);
```

```
            c.Dung();
            Console.WriteLine("c.DaKhoiDong      =      "      +
c.DaKhoiDong);
            Console.ReadLine();
      }
}
```

III.9.2. Hủy đối tượng

Ngôn ngữ C# cung cấp *cơ chế thu dọn rác tự động* (garbage collection) và do vậy không cần phải khai báo tường minh các phương thức hủy. Tuy nhiên, khi làm việc với các đoạn mã không được quản lý thì cần phải khai báo tường minh các phương thức hủy để giải phóng các tài nguyên. C# cung cấp ngầm định một phương thức để thực hiện điều khiển công việc này, phương thức đó là ***Finalize()*** hay còn gọi là bộ kết thúc. Phương thức ***Finalize()*** này sẽ được gọi một cách tự động bởi *cơ chế thu dọn* khi đối tượng bị hủy.

Phương thức ***Finalize()*** chỉ giải phóng các tài nguyên mà đối tượng nắm giữ, và không tham chiếu đến các đối tượng khác. Nếu với những đoạn mã bình thường tức là chứa các tham chiếu kiểm soát được thì không cần thiết phải tạo và thực thi phương thức ***Finalize()***. Ta chỉ làm điều này khi xử lý các tài nguyên không kiểm soát được.

Ta không bao giờ gọi một phương thức ***Finalize()*** của một đối tượng một cách trực tiếp, ngoại trừ gọi phương thức này của lớp cơ sở khi ở bên trong phương thức ***Finalize()*** của lớp đang định nghĩa. Trình thu dọn sẽ tự động thực hiện việc gọi ***Finalize()*** cho chúng ta.

a) Cách Finalize thực hiện

Bộ thu dọn duy trì một danh sách những đối tượng có phương thức ***Finalize()***. Danh sách này được cập nhật mỗi lần khi đối tượng cuối cùng được tạo ra hay bị hủy. Khi một đối tượng trong danh sách kết thúc của *bộ thu dọn* được chọn đầu tiên. Nó sẽ được đặt vào hàng đợi (queue) cùng với những đối tượng khác đang chờ kết thúc. Sau khi phương thức ***Finalize()*** của đối tượng thực thi bộ thu dọn sẽ gom lại đối tượng và cập nhật lại danh sách hàng đợi, cũng như là danh sách kết thúc đối tượng.

b) Bộ hủy của C#

Ta khai báo một phương thức hủy trong C# như sau:

```
~Class1()
 {
}
```

Tuy nhiên, trong ngôn ngữ C# thì cú pháp khai báo trên là một shortcut liên kết đến một phương thức kết thúc ***Finalize()*** được kết với lớp cơ sở, do vậy khi viết

```
~Class1()
{
```

```
        // Thực hiện một số công việc
    }
```

Cũng tương tự như viết :

```
    Class1.Finalize()
    {
        // Thực hiện một số công việc
        base.Finalize();
    }
```

Do sự tương tự như trên nên khả năng dẫn đến sự lộn xộn nhầm lẫn là không tránh khỏi, nên ta phải tránh viết các phương thức hủy và viết các phương thức ***Finalize()*** tường minh nếu có thể được.

c) Phương thức Dispose

Như chúng ta đã biết thì việc gọi một phương thức kết thúc ***Finalize()*** trong C# là không hợp lệ, vì phương thức này được thực hiện một cách tự động bởi *bộ thu dọn rác*. Nếu muốn xử lý các tài nguyên không kiểm soát (như xử lý các handle của tập tin), đóng hay giải phóng nhanh chóng bất cứ lúc nào thì ta nên thực thi giao diện ***IDisposable***. Giao diện ***IDisposable*** yêu cầu những lớp thực thi nó định nghĩa một phương thức tên là ***Dispose()*** để thực hiện công việc dọn dẹp. Ý nghĩa của phương thức ***Dispose()*** là cho phép chương trình thực hiện các công việc dọn dẹp hay giải phóng tài nguyên mong muốn mà không phải chờ cho đến khi phương thức ***Finalize()*** được gọi.

Khi chúng ta cung cấp một phương thức ***Dispose()*** thì phải ngưng bộ thu dọn gọi phương thức ***Finalize()*** trong đối tượng của chúng ta. Để ngưng bộ thu dọn, chúng ta gọi một phương thức tĩnh của lớp GC (garbage collector) là ***GC.SuppressFinalize()*** và truyền tham số là tham chiếu **this** của đối tượng. Và sau đó phương thức ***Finalize()*** sử dụng để gọi phương thức ***Dispose()*** như đoạn mã sau:

```
    public void Dispose()
    {
        // Thực hiện công việc dọn dẹp
        // Yêu cầu bộ thu dọn GC trong thực hiện kết thúc
        GC.SuppressFinalize( this );
    }
    public override void Finalize()
    {
        Dispose();
        base.Finalize();
    }
```

Ví dụ:

```
using System;
     public class Mang: IDisposable
     {
          int  []Data;
          public int Size;

          public  Mang(int n)
          {
               Data = new int[n];

               Size = n;
          }
          public void Dispose()
          {
               // Thuc hien viec don dep
               Console.WriteLine("Huy mang");
               Data =new int [0];
               Size  = 0;
               // yeu cau bo thu don GC thuc hien ket thuc
               GC.SuppressFinalize( this );
          }
          public void Nhap()
          {
               for( int i =0; i < Size; i++)
               {
                    Console.WriteLine("nhap phan tu thu {0}
",i);
                    Data[i]                                   =
int.Parse(Console.ReadLine());
               }
          }

          public void Xuat()
          {
               for( int i = 0; i < Size; i++)
               {
                    Console.Write("{0} \t",Data[i]);

               }
          }
     }
     class App
     {
          /// <summary>
          /// The main entry point for the application.
          /// </summary>
          [STAThread]
          static void Test()
          {
               Mang A = new Mang(5);
               A.Nhap();
```

```
            A.Xuat();
            A.Dispose();
        }
        static void Main(string[] args)
        {
            Test();
            Console.ReadLine();
        }
}
```

d) Phương thức Close

Khi xây dựng các đối tượng, chúng ta muốn cung cấp cho người sử dụng phương thức ***Close()***, vì phương thức ***Close()*** có vẻ tự nhiên hơn phương thức ***Dispose()*** trong các đối tượng có liên quan đến xử lý tập tin. Ta có thể xây dựng phương thức ***Dispose()*** với thuộc tính là **private** và phương thức ***Close()*** với thuộc tính **public**. Trong phương thức ***Close()*** đơn giản là gọi thực hiện phương thức ***Dispose()***.

e) Câu lệnh using

Khi xây dựng các đối tượng chúng ta không thể chắc chắn được rằng người sử dụng có thể gọi hàm ***Dispose()***. Và cũng không kiểm soát được lúc nào thì bộ thu dọn GC thực hiện việc dọn dẹp. Do đó để cung cấp khả năng mạnh hơn để kiểm soát việc giải phóng tài nguyên thì C# đưa ra cú pháp chỉ dẫn **using**, cú pháp này đảm bảo phương thức ***Dispose()*** sẽ được gọi sớm nhất có thể được. Ý tưởng là khai báo các đối tượng với cú pháp **using** và sau đó tạo một phạm vi hoạt động cho các đối tượng này trong khối được bao bởi dấu ({}). Khi khối phạm vi này kết thúc, thì phương thức ***Dispose()*** của đối tượng sẽ được gọi một cách tự động.

```
using System.Drawing;
class Tester
{
    public static void Main()
    {
        using ( Font Afont = new Font("Arial",10.0f))
        {
            // Đoạn mã sử dụng AFont
            .....
        }// Trình biên dịch sẽ gọi Dispose để giải phóng AFont

        Font TFont = new Font("Tahoma",12.0f);
        using (TFont)
        {
            // Đoạn mã sử dụng TFont
            .....
        }// Trình biên dịch gọi Dispose để giải phóng TFont
    }
}
```

Trong phần khai báo đầu của ví dụ thì đối tượng Font được khai báo bên trong câu lệnh **using**. Khi câu lệnh **using** kết thúc, thì phương thức ***Dispose()*** của đối tượng Font sẽ được gọi.

Còn trong phần khai báo thứ hai, một đối tượng Font được tạo bên ngoài câu lệnh **using**. Khi quyết định dùng đối tượng này ta đặt nó vào câu lệnh **using**. Và cũng tương tự như trên khi khối câu lệnh **using** thực hiện xong thì phương thức ***Dispose()*** của font được gọi.

III.9.3. Thực thi nhiều giao diện

Các lớp có thể thực thi nhiều giao diện, đây là cách để thực hiện đa kế thừa trong C#.

Ví dụ: Tạo một giao diện tên là ***Istoreable*** với các phương thức ***Write()*** để lưu nội dung của đối tượng vào file và phương thức ***Read()*** để đọc dữ liệu từ file. Sau đó ta tạo lớp ***Document*** thực thi giao diện ***Istorable*** để các đối tượng thuộc lớp này có thể đọc từ cơ sở dữ liệu hoặc lưu trữ vào cơ sở dữ liệu. Việc mở file được thực hiện thông qua đối tượng ***fs*** thuộc lớp ***FileStream***, việc ghi và đọc file thông qua đối tượng thuộc các lớp ***StreamWriter*** và ***StreamReader***. Đồng thời lớp ***Document*** cũng thực thi một giao diện khác tên là `Iencryptable`, giao diện này có hai phương thức là mã hóa (`Encrypt( )`)và giải mã (`Decrypt( )`):

```
using System;
using System.IO;

// Khai bao giao dien
interface IStorable
{
     // mac dinh cac khai bao phuong thuc la public. Khong
cai dat gi
     void Read(string FileName);
     void Write(string FileName);
     string Data { get; set; }
}

interface IEncryptable
{
     void Encrypt( );
     void Decrypt( );
}

// Lop Document thuc thi giao dien IStorable
public class Document : IStorable, IEncryptable
{
     string S;
     public Document(string str)
     {
          S = str;
     }
     // thuc thi phuong thuc Read cua giao dien IStorable
```

```
     public void Read( string FileName)
     {
          //Mo fiel de doc
          FileStream            fs            =            new
FileStream(FileName,FileMode.Open);
          //tao luong doc du lieu
          StreamReader sr = new StreamReader(fs);
          //Doc tung dong cho den khi gia tri doc duoc la
null
          string text;
          S= "";
          while((text = sr.ReadLine())!=null)
          {
               S = S + text;
          }
          //Dong luong va dong file
          sr.Close();
          fs.Close();
     }
     // thuc thi phuong thuc Write cua giao dien IStorable
     public void Write(string FileName)
     {
          //Mo file de ghi du lieu
          FileStream fs ;
          fs                         =                          new
     fileStream(FileName,FileMode.OpenOrCreate);
          //Tao luong ghi du lieu vao file
          StreamWriter sw = new StreamWriter(fs);
          //ghi chuoi S ra file
          sw .WriteLine(S);
          //dong luong va dong file
          sw .Close();
          fs.Close();
     }
     // thuc thi thuoc tinh Data cua giao dien IStorable
     public string Data
     {
          get  {    return S; }
          set  {    S = value;     }
     }
     //  thuc  thi  phuong  thuc  Encrypt  cua  giao  dien
IEncryptable
     public void  Encrypt()
     {
          string KQ ="";
          for (int i = 0 ; i< S.Length; i++)
               KQ = KQ + (char)((int)S[i] + 5);
          S = KQ;
     }
     //  thuc  thi  phuong  thuc  Encrypt  cua  giao  dien
IEncryptable
```

```
      public void  Decrypt()
      {
            string KQ ="";
            for (int i = 0 ; i< S.Length; i++)
                  KQ= KQ + (char)((int)S[i] - 5);
            S = KQ;
      }
}
// Thu nghiem chuong chinh
public class Tester
{
      static void Main( )
      {
            string FileName = "F:\\datafile.txt";
            Document doc = new Document("THIEU NU la viet tat
            cua tu THIEU NU TINH");
            doc.Write(FileName);
            doc.Read(FileName);
            Console.WriteLine("Du  lieu  trong  file:  {0}",
doc.Data);
            Console.WriteLine("Du lieu sau khi ma hoa:");
            doc.Encrypt();
            Console.WriteLine(doc.Data);
            Console.WriteLine("Du lieu sau khi giai ma:");
            doc.Decrypt();
            Console.WriteLine(doc.Data);
            Console.ReadLine();
      }
}
```

Vì giao diện là một dạng lớp cơ sở nên ta có thể cho một tham chiếu kiểu giao diện trỏ tới đối tượng thuộc lớp thực thi giao diện và gọi những phương thức cần thiết. Chẳng hạn, trong hàm *Main()* trên ta có thể thay câu lệnh:

```
doc.Read(FileName);
```

bằng hai câu lệnh sau:

```
IStorable isDoc = doc; //(IStorable) doc;
isDoc.Read(FileName);
```

III.9.4. Mở rộng giao diện

Ta cũng có thể mở rộng giao diện bằng cách bổ sung những thành viên hay phương thức mới. Chẳng hạn, ta có thể mở rộng giao tiếp ***IStorable*** thành ***IStorableAndCompressible*** bằng cách kế thừa từ giao tiếp ***Istorable*** bổ sung các phương thức nén file và giải nén file:

```
interface IStorableAndCompressible : IStorable
{
      // bo sung them phuong thuc nen va giai nen
      void Compress( );
      void Decompress( );
```

```
}
```

III.9.5. Kết hợp giao diện

Thay vì lớp ***Document*** thực thi hai giao diện ***IStorable, IEncryptable*** , ta có thể tạo kết hợp hai giao diện này thành một giao diện mới là ***IStorableAndEncryptable***. Sau đó lớp ***Document*** thực thi giao diện mới này:

```
interface IstorableAndEncryptable: IStorable,
IEncryptable
{
//Có thể bổ sung thêm các phương thức, thuộc tính… mới
}
public class Document : IStorableAndEncryptable
{
…
}
```

III.9.6. Kiểm tra đối tượng có hỗ trợ giao diện hay không bằng toán tử *is*

Ta có thể hỏi một đối tượng có hỗ trợ giao tiếp hay không để gọi các phương thức thích hợp bằng cách dùng toán tử ***is***. Nếu đối tượng không hỗ trợ giao diện mà ta cố tình ép kiểu thì có thể xảy ra biệt lệ (lỗi khi thực thi chương trình).

Ví dụ: kiểm tra đối tượng ***doc*** có hỗ trợ giao diện ***IStorable*** hay không, nếu có thì ép sang kiểu ***IStorable*** và gọi phương thức ***Read().***

```
static void Main( )
{
     Document doc = new Document("Test Document");
     // chi ep sang kieu Istorable neu doi tuong co ho tro
     if (doc is IStorable)
     {
          IStorable isDoc = (IStorable) doc;
          isDoc.Read( );
     }
}
```

III.9.7. Các giao diện Icomparer, IComparable (giao diện so sánh) và ArrayList

- Thuộc name space System.Conllections
- Khi thi công giao diện IComparer cần cài đặt hàm mô tả sự so sánh hai đối tượng obj1 và obj2:

  int Compare(Objerct obj1, Object obj2)

- Khi thi công giao diện IComparable cần cài đặt hàm mô tả sự so sánh của đối tượng hiện tại với một đối tượng obj2 khác.

  int CompareTo(Object obj2)

- ArrayList có các chức năng của cả mảng và danh sách liên kết như:

Tên	Loại	Ý nghĩa
Count	Thuộc tính	Số phần tử hiện có
Add	Hàm	Thêm một phần tử vào cuối
BinarySearch	Hàm	Tìm nhị phân
Contains	Hàm	Kiểm một phần tử có thuộc Array List hay không
Clear	Hàm	Xóa các phần tử
IndexOf	Hàm	Trả về vị trí đầu tiên của phần tử cần tìm
LastIndexOf	Hàm	Trả về vị trí cuối cùng của phần tử cần tìm
Insert	Hàm	Chèn 1 phần tử vào 1 vị trí nào đó
Remove	Hàm	Loại bỏ 1 phần tử
RemoveAt	Hàm	Loại bỏ 1 phần tử tại một vị trí nào đó
Reverse	Hàm	Đảo chiều
Sort	Hàm	Sắp xếp
…	...	

- ArrayList cho phép sắp xếp bất cứ đối tượng nào cài đặt giao diện Icomparable.

Ví dụ 1:

- Khai báo và cấp phát đối tượng ArrayList AL:

```
ArrayList AL = new ArrayList();
```

- Thêm một phần tử có giá trị 5 vào cuối AL:

```
AL.Add(5);
```

- Chèn giá trị 6 vào đầu AL:

```
AL.Insert(0, 6);
```

- Xuất phần tử thứ 2 trong AL:

```
Console.WriteLine("{0}", AL[1]);
```

…

Ví dụ 2: sắp xếp trên ArrayList theo tên của nhân viên.

```
using System;
using System.Collections;

class NhanVien: IComparable
{
```

```
	string Ten;
	long Luong;
	public NhanVien(string T, long L)
	{
		Ten = T;
		Luong = L;
	}
	public override string ToString()
	{
		return Ten + ", " + Luong;
	}
	//Thuc thi ham cua giao dien IComparable
	//Ham nay uy quyen viec so sanh 2 doi tuong cho ham so
sanh
	//ten cua doi tuong hien hanh voi ten cua doi tuong
khac
	public int CompareTo(Object rhs)
	{
		NhanVien r = (NhanVien) rhs;
		return Ten.CompareTo(r.Ten);
	}
	public void Xuat()
	{
		Console.WriteLine(Ten + ", " + Luong);
	}
}

class App
{
	[STAThread]
	static void Main(string[] args)
	{
		string []AT = {"Anh Hai", "Chi Hai","Anh Ba","Chi
Ba"};
		long []AL = {500, 600, 700, 800};

		ArrayList empArray = new ArrayList();
		Random r = new Random(10);
		//r.Next(X):---> Tao so nguyen <= X
		string T; long L;
		//Toa cac nhan vien nhan Ten va Luong ngau nhien
		for (int i =0 ; i<5; i ++)
		{
			T= AT[r.Next(4)];
			L= AL[r.Next(4)];
			empArray.Add(new NhanVien(T, L));
		}

	Console.WriteLine("================================");
		Console.WriteLine("Cac nhan vien");
		for (int i =0 ; i<5; i ++)
```

```
            {
                Console.WriteLine(empArray[i].ToString());
            }
            Console.WriteLine("Tat ca co {0} nhan vien",
empArray.Count);

            //Chen 2 nhan vien vao vi tri thu 2 và 4
            NhanVien nv = new NhanVien("Anh Tu", 900);
            empArray.Insert(1,nv);
            nv = new NhanVien("Chi Tu", 300);
            empArray.Insert(3, nv);

            //Xuat

      Console.WriteLine("================================");
            Console.WriteLine("Them 2 nhan vien");
            foreach (NhanVien x in  empArray)
            {
                x.Xuat();
            }
            Console.WriteLine("Tat ca co {0} nhan vien",
empArray.Count);

            //Xoa mot nhan vien cuoi cung va sap xep theo ten

      Console.WriteLine("================================");
            empArray.RemoveAt(3);
            empArray.Sort();
            Console.WriteLine("Sau khi xoa nhan vien thu 4 va
sap xep theo luong:");
            foreach (NhanVien x in  empArray)
            {
                x.Xuat();
            }
            Console.WriteLine("Tat ca co {0} nhan vien",
empArray.Count);

            Console.ReadLine();
      }
}
```

Nếu mỗi đối tượng trong ArrayList có thể có nhiều tiêu chí để so sánh ta cũng có thể sắp xếp theo tiêu chí tương ứng. Chẳng hạn ta có thể sắp xếp nhân viên theo tên nhưng cũng có thể sắp xếp theo lương.

Khi đó cách cài đặt như sau:

a) Xây dựng một lớp chính kế thừa giao diện IComparable. Thực thi 2 hàm sau:
int CompareTo(Object obj) của giao diện IComparable .

int CompareTo(Object obj, ThamSốQuiĐịnhTieuChíSoSánh *comp*).

```
class NhanVien //LopChinh
{
      …
      int CompareTo(Object obj) {…}.
      int CompareTo(Object obj, ThamSốQuiĐịnhTiêuChíSoSánh
      comp)
      {
      //các câu lệnh so sánh tương ứng với tiêu chí comp
      }
      …

}
```

b) Xây dựng một lớp kế thừa giao diện Icomparer

Lớp này là một lớp nội tại của lớp chính (inner class-lớp định nghĩa bên trong một lớp khác) đang xây dựng. (Trong ví dụ này, lớp chính là lớp NhanVien, lớp nội tại là lớp EmpComparer chính). Đối tượng của lớp nội tại sẽ quản lý việc so sánh hai đối tượng thuộc lớp chính, nó sẽ gọi hàm so sánh của lớp chính theo tiêu chí cần so sánh.

```
public class EmpComparer: IComparer
{
      ThamSốQuiĐịnhTiêuChíSoSánh com;
      public int Compare(Object lhs, Object rhs)
      {
            NhanVien r = (NhanVien) rhs;
            NhanVien l = (NhanVien) lhs;
            return l.CompareTo(r, _whichComparison);
      }
}
```

c) Trước khi gọi hàm Sort() của đối tượng ArrayList ta cần thực hiện các việc sau:

- Tạo một đối tượng (tạm gọi là *comp*) thuộc lớp nội tại.
- Qui định tiêu chí so sánh thông qua đối tượng *comp* này.
- Gọi hàm Sort của ArrayList với tham số là đối tượng *comp* này.

Ví dụ 3: sắp xếp các nhân viên trong ArrayList theo nhiều tiêu chí khác nhau.

```
using System;
using System.Collections;

class NhanVien: IComparable
{
      string Ten;
      long Luong;
      public NhanVien(string T, long L)
      {
            Ten = T;
```

```
            Luong = L;
        }
        public override string ToString()
        {
            return Ten + " , " + Luong;
        }
        public static EmpComparer GetComparer()
        {
            return new NhanVien.EmpComparer();
        }

        // Mac dinh se sap xep theo ten.
        //So sanh ten voi ten cua doi tuong khac
        public int CompareTo(Object rhs)
        {
            NhanVien r = (NhanVien) rhs;
            return Ten.CompareTo(r.Ten);
        }

        // So sanh ten voi ten cua doi tuong khac
        public int CompareTo(NhanVien rhs,
EmpComparer.ComparisonType whichComparison)
        {
            switch (whichComparison)
            {
                case EmpComparer.ComparisonType.Ten:
                    return Ten.CompareTo(rhs.Ten);
                case EmpComparer.ComparisonType.Luong:
                    return Luong.CompareTo(rhs.Luong);
                default:
                    return 0;
            }

        }
        // Lop noi tai (inner class) dung de thiet lap cac
thong tin so sanh hai doi tuong cung lop nhan vien
        public class EmpComparer:System.Collections.IComparer
        {
            ComparisonType _whichComparison;
            public enum ComparisonType
            {
                Ten,Luong
            }
            public int Compare(Object lhs, Object rhs)
            {
                NhanVien r = (NhanVien) rhs;
                NhanVien l = (NhanVien) lhs;
                return l.CompareTo(r, _whichComparison);
            }
            public ComparisonType WhichComparison
            {
```

```
                get {return _whichComparison;}
                set {_whichComparison =value;}
            }
        }
}
class App
{
        [STAThread]
        static void Main(string[] args)
        {
            string []AT = {"Anh Hai", "Chi Hai","Anh Ba","Chi
Ba"};
            long []AL = {500, 600, 700, 800};
            ArrayList empArray = new ArrayList();
            Random r = new Random(10);
            //r.Next(X):---> Tao so nguyen <= X
            string T; long L;
            for (int i =0 ; i<5; i ++)
            {
                T= AT[r.Next(4)];
                L= AL[r.Next(4)];
                empArray.Add(new NhanVien(T, L));
            }
            Console.WriteLine("======================");
            Console.WriteLine("Truoc khi sap xep");
            for (int i =0 ; i<5; i ++)
                Console.WriteLine(empArray[i].ToString());

            //Tao doi tuong quan ly viec so sanh
            NhanVien.EmpComparer c = NhanVien.GetComparer();
            //Qui dinh tieu chi so sanh la theo luong
            c.WhichComparison                                    =
NhanVien.EmpComparer.ComparisonType.Luong;
            //Goi ham Sort, tham so la doi tuong quan ly viec
so sanh
            empArray.Sort(c);
            Console.WriteLine("======================");
            Console.WriteLine("Sau khi sap xep theo luong:");
            for (int i =0 ; i<5; i ++)
                Console.WriteLine(empArray[i].ToString());

            Console.WriteLine("======================");
            //Qui dinh lai tieu chi so sanh la theo ten
            c.WhichComparison                                    =
NhanVien.EmpComparer.ComparisonType.Ten;
            //Goi ham Sort voi tham so la doi tuong quna ly
viec so sanh
            empArray.Sort(c);
            Console.WriteLine("Sau khi sap xep theo ten:");
            for (int i =0 ; i<5; i ++)
                Console.WriteLine(empArray[i].ToString());
```

```
            Console.ReadLine();
        }
}
```

Giao diện *IComparable* có tác dụng thiết lập tiêu chí sắp xếp mặc định. Vì vậy, lớp chính (NhanVien) không nhất thiết phải kế thừa giao diện *IComparable*, tuy nhiên khi đó mỗi lần gọi hàm Sort() của ArrayList ta luôn phải xác định trước tiêu chí sắp xếp.

III.9.8. Câu hỏi ôn tập

1. *Sự chuyên biệt hóa được sử dụng trong C# thông qua tính gì?*
2. *Khái niệm đa hình là gì? Khi nào thì cần sử dụng tính đa hình?*
3. *Hãy xây dựng cây phân cấp các lớp đối tượng sau: Xe_Toyota, Xe_Dream, Xe_Spacy, Xe_BMW, Xe_Fiat, Xe_DuLich, Xe_May, Xe?*
4. *Từ khóa new được sử dụng làm gì trong các lớp?*
5. *Một phương thức ảo trong lớp cơ sở có nhất thiết phải được phủ quyết (định nghĩa lại) trong lớp dẫn xuất hay không?*
6. *Lớp trừu tượng có cần thiết phải xây dựng hay không? Hãy cho một ví dụ về một lớp trừu tượng cho một số lớp.*
7. *Lớp Object cung cấp những phương thức nào mà các lớp khác thường xuyên kế thừa để sử dụng.*

III.9.9. Bài tập tổng hợp

1. *Hãy xây dựng các lớp đối tượng trong câu hỏi 3, thiết lập các quan hệ kế thừa dựa trên cây kế thừa mà bạn xây dựng. Mỗi đối tượng chỉ cần một thuộc tính là myNane để cho biết tên của nó (như Xe_Toyota thì myName là "Toi la Toyota"...). Các đối tượng có phương thức Who() cho biết giá trị myName của nó. Hãy thực thi sự đa hình trên các lớp đó. Cuối cùng tạo một lớp Tester với hàm Main() để tạo một mảng các đối tượng Xe, đưa từng đối tượng cụ thể vào mảng đối tượng Xe, sau đó cho lặp từng đối tượng trong mảng để nó tự giới thiệu tên (bằng cách gọi hàm Who() của từng đối tượng).*
2. *Xây dựng các lớp đối tượng hình học như: điểm, đoạn thẳng, đường tròn, hình chữ nhật, hình vuông, tam giác, hình bình hành, hình thoi. Mỗi lớp có các thuộc tính riêng để xác định được hình vẽ biểu diễn của nó như đoạn thẳng thì có điểm đầu, điểm cuối.... Mỗi lớp thực thi một phương thức Draw() phủ quyết phương thức Draw() của lớp cơ sở gốc của các hình mà nó dẫn xuất. Hãy xây dựng lớp cơ sở của các lớp trên và thực thi đa hình với phương thức Draw(). Sau đó tạo lớp Tester cùng với hàm Main() để thử nghiệm đa hình giống như bài tập 1 ở trên.*

PHỤ LỤC
Phụ lục A - CƠ BẢN VỀ NGÔN NGỮ C#

I. Tạo ứng dụng trong C#

Ví dụ dưới đây là một ứng dụng dòng lệnh (console application) đơn giản, ứng dụng này giao tiếp với người dùng thông qua bàn phím, màn hình DOS và không có giao diện đồ họa người dùng, giống như các ứng dụng thường thấy trong Windows. Khi xây dựng các ứng dụng nâng cao trên Windows hay Web ta mới cần dùng các giao diện đồ họa, còn để tìm hiểu về ngôn ngữ C# thuần tuý thì cách tốt nhất là ta viết các ứng dụng dòng lệnh.

Ứng dụng dòng lệnh trong ví dụ sau sẽ xuất chuỗi "Hello World" ra màn hình.

Ví dụ:

```
using System;  //Khai báo thư viện
class HelloWorld    //Khai báo lớp
{
     static void Main( ) //Định nghĩa hàm Main
     {
          //Xuất câu thông báo "Hello ra màn hình"
          System.Console.WriteLine("Hello World");

          //Chờ người dùng gõ một phím bất kỳ để dừng
chương trình
          System.Console.ReadLine();
     }
}
```

Chương trình trên khai báo một kiểu đơn giản: lớp ***HelloWorld*** bằng từ khóa *class*, được bao bởi dấu {}, trong đó có một phương thức (hàm) tên là ***Main().*** Vì khi chương trình thực thi, CLR gọi hàm ***Main()*** đầu tiên nên mỗi chương trình phải có một và chỉ một hàm ***Main().***

Các lớp có các thuộc tính dữ liệu và các hành vi của chúng. Thuộc tính dữ liệu là các thành phần dữ liệu mà mọi đối tượng thuộc lớp đều có. Hành vi là phương thức của lớp (còn gọi là hàm thành viên), đó là các hàm thao tác trên các thành phần dữ liệu của lớp. Trong ví dụ này, lớp ***HelloWorld*** không có thuộc tính dữ liệu và hành vi nào (trừ hàm **Main()** bắt buộc phải có).

Trong ví dụ này, ta sử dụng đối tượng ***Console*** để thao tác với bàn phím và màn hình. Đối tượng ***Console*** thuộc không gian (name space, thư viện) *System* vì vậy ta sử dụng chỉ thị ***using System*** ở đầu chương trình.

Để truy cập đến một thành phần của lớp hoặc của đối tượng ta dùng toán tử chấm ".". Lệnh ***`System.Console.WriteLine("Hello World");`*** có

nghĩa là gọi hàm ***WriteLine*** của đối tượng ***Console*** trong thư viện ***System*** để xuất một chuỗi ***"Hello World"*** ra màn hình.

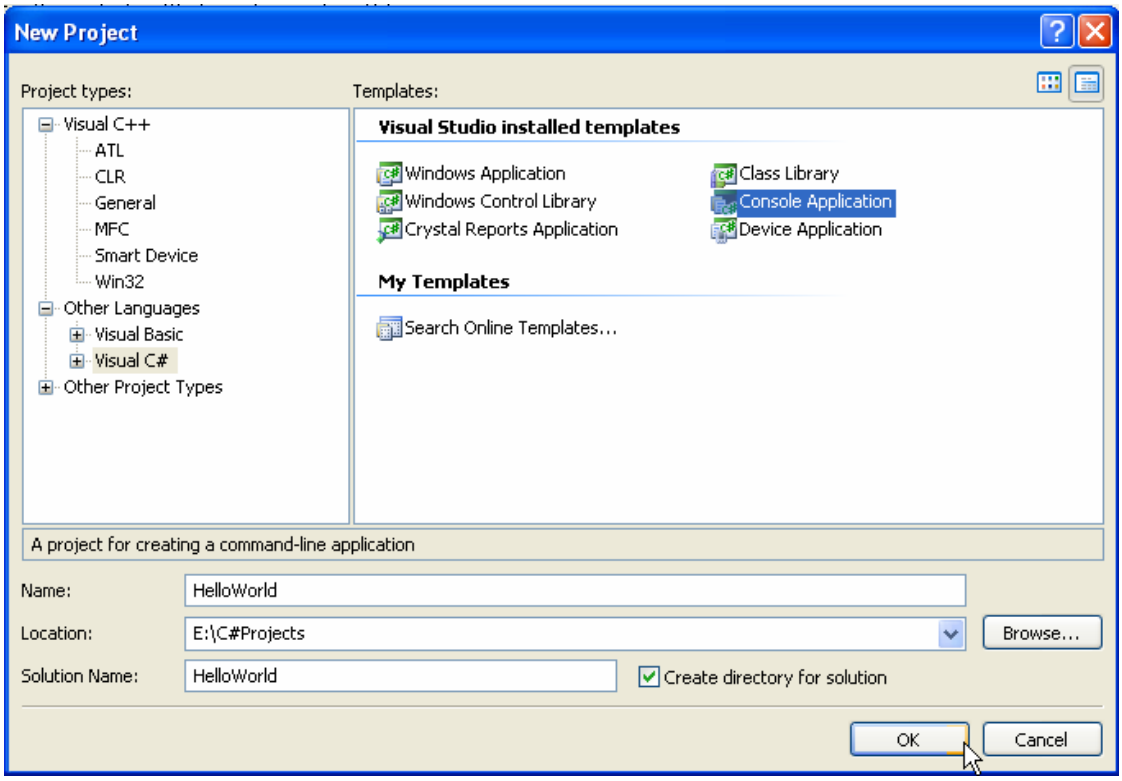
Lệnh `System.Console.ReadLine();` thực chất dùng để nhập một chuỗi từ bàn phím. Trong trường hợp này nó có tác dụng chờ người dùng nhấn phím Enter để kết thúc chương trình.

Chú ý:

Phần 1 này trình bày các chương trình theo phương pháp lập trình thủ tục truyền thống nhằm làm quen với ngôn ngữ C#.

I.1. Soạn thảo chương trình "Hello World"

- Khởi động MS Visual Studio .Net 2003 qua các bước sau: **Start \ Programs \ Chọn MS Visual Studio .Net 2003 \ MS Visual Studio .Net 2003**.
- Tạo ứng dụng dòng lệnh tên là Hello World qua các bước sau: **File \ New \ Project. Chọn Visual C# Project** trong ô **Project Types** và chọn **Console Application** trong ô **Templates** như hình dưới đây. Nhập vào tên dự án là ***HelloWorld*** vào ô **Name** và đường dẫn để lưu trữ dự án vào ô **Location** (ví dụ, E:\C#Projects).



Hình I-1: Tạo ứng dụng C# console trong Visual Studio .NET.

- Sau đó đưa lệnh sau vào trong phương thức Main().

```
System.Console.WriteLine("Hello World");
```

I.2. Biên dịch và chạy chương trình "Hello World"

- Chọn Ctrl+Shift+B hay Build→Build.
- Nhấn Ctrl + F5 để chạy chương trình mà không thực hiện dò lỗi.

II. Cơ sở của ngôn ngữ C#

Trong phần trước chúng ta đã tìm hiểu cách tạo một chương trình C# đơn giản. Phần này sẽ trình bày các kiến thức cơ bản cho việc lập trình trong ngôn ngữ C# như: hệ thống kiểu dữ liệu xây dựng sẵn (như int, bool, string…), hằng, cấu trúc liệt kê, chuỗi, mảng, biểu thức và cậu lệnh, các cấu trúc điều khiển như **if, switch**, **while**, **do**...**while**, **for**, và **foreach**... Nắm vững phần này sẽ giúp ta hiểu phương pháp lập trình hướng đối tượng một cách nhanh chóng, dễ dàng.

II.1. Kiểu dữ liệu

Kiểu dữ liệu trong C# có thể được phân làm 2 loại:

- Kiểu dựng sẵn: int, long ...
- Kiểu người dùng tạo ra: lớp, struct…

Tuy nhiên, người lập trình thường quan tâm tới cách phân loại sau:

- Kiểu giá trị: giá trị thật sự của nó được cấp phát trên stack. Ví dụ: int, long ..., struct.
- Kiểu tham chiếu: địa chỉ của kiểu tham chiếu được lưu trữ trên stack nhưng dữ liệu thật sự lưu trữ trong heap. Ví dụ: lớp, mảng…

***Chú ý*:**

Tất cả các kiểu dữ liệu xây dựng sẵn là kiểu dữ liệu giá trị ngoại trừ các đối tượng, mảng và chuỗi. Và tất cả các kiểu do người dùng định nghĩa ngoại trừ kiểu cấu trúc đều là kiểu dữ liệu tham chiếu.

II.1.1. Các kiểu xây dựng sẵn trong C#:

Ngôn ngữ C# đưa ra các kiểu dữ liệu xây dựng sẵn rất hữu dụng, phù hợp với một ngôn ngữ lập trình hiện đại. Mỗi kiểu nguyên thủy của C# được ánh xạ đến một kiểu dữ liệu được hỗ trợ bởi hệ thống xác nhận ngôn ngữ chung (Common Language Specification: CLS) trong MS.NET. Việc ánh xạ này đảm bảo các đối tượng được tạo ra trong C# có thể được sử dụng đồng thời với các đối tượng được tạo bởi bất cứ ngôn ngữ khác được biên dịch bởi .NET, chẳng hạn VB.NET.

Kiểu trong C#	Kích thước (byte)	Kiểu tương ứng trong .Net	Mô tả

byte	1	Byte	Không dấu 0 → 255
char	1	Char	Ký tự unicode
bool	1	Boolean	True hay false
sbyte	1	Sbyte	Có dấu(-128 →127)
short	2	Int16	Có dấu -32.768 → 32.767
ushort	2	Uint16	Không dấu (0 → 65353)
int	4	Int32	Có dấu -2,147,483,647 → 2,147,483,647.
uint	4	Uint32	Không dấu 0 → 4,294,967,295.
float	4	Single	$+/-1.5 * 10^{-45}$ → $+/-3.4 * 10^{38}$
double	8	Double	$+/-5.0 * 10^{-324}$ → $+/-1.7 * 10^{308}$
decimal	8	Decimal	Lên đến 28 chữ số.
long	8	Int64	-9,223,372,036,854,775,808 → 9,223,372,036,854,775,807
ulong	8	Uint64	0 to 0xffffffffffffffff.

C# đòi hỏi các biến phải được khởi gán giá trị trước khi truy xuất giá trị của nó.

II.1.2. Hằng

Cú pháp:

const kiểu tên_biến = giá trị.

Ví dụ II.1.2: Khai báo hai hằng số **DoDongDac, DoSoi** (nhiệt độ đông đặc và nhiệt độ sôi).

```
using System;
class Values
{
      const int DoDongDac = 32; // Nhiệt độ đông đặc
      const int DoSoi = 212;   //Độ sôi
      static void Main( )
      {
         Console.WriteLine("Nhiệt độ đông đặc của nước: {0}",
         DoDongDac);
         Console.WriteLine("Nhiệt độ sôi của nước: {0}",
         DoSoi);
```

```
    }
}
```

II.1.3. Kiểu liệt kê

Kiểu liệt kê là một kiểu giá trị rời rạc, bao gồm một tập các hằng có liên quan với nhau. Mỗi biến thuộc kiểu liệt kê chỉ có thể nhận giá trị là một trong các hằng đã liệt kê. Chẳng hạn, thay vì khai báo dài dòng và rời rạc như sau:

```
const int DoDongDac = 32; // Nhiệt độ đông đặc
const int DoSoi = 212;     //Độ sôi
const int HoiLanh = 60;
const int AmAp = 72;
```

ta có thể định nghĩa kiểu liệt kê có tên là `NhietDo` như sau:

```
enum NhietDo
{
    DoDongDac = 32; // Nhiệt độ đông đặc
    DoSoi = 212;   //Độ sôi
    HoiLanh = 60;
    AmAp = 72;
}
```

Mỗi kiểu liệt kê có thể dựa trên một kiểu cơ sở như int, short, long..(trừ kiểu char). Mặc định là kiểu int.

Ví dụ II.1.3.1: Xây dựng kiểu liệt kê mô tả kích cỡ của một đối tượng nào đó dựa trên kiểu số nguyên không dấu:

```
enum KichCo: uint
{
    Nho = 1;
    Vua = 2;
    Rong = 3;
}
```

Ví dụ II.1.3.2: Ví dụ minh họa dùng kiểu liệt kê để đơn giản mã chương trình:

```
using System;
enum NhietDo
{
    GiaBuot = 0, DongDac = 32, AmAp = 72, NuocSoi = 212
}

class Enum
{

    static void Main(string[] args)
    {
        Console.WriteLine("Nhiệt độ đông đặc của nước:
        {0}", NhietDo.DoDongDac);
```

```
            Console.WriteLine("Nhiệt độ sôi của nước: {0}",
            NhietDo.DoSoi);
      }
}
```

Mỗi hằng trong kiểu liệt kê tương ứng với một giá trị. Nếu chúng ta không chỉ ra giá trị, giá trị mặc định là 0 và tăng thứ tự với các phần tử tiếp theo.

Ví dụ II.1.3.3

```
enum SomeValues
{
          First,    Second, Third = 20, Fourth
}
```

Khi đó: `First = 0, Second = 2, Third = 20, Fourth = 21.`

II.1.4. Kiểu chuỗi

Đối tượng *string* lưu trữ một chuỗi các ký tự. Chuỗi là một mảng các ký tự nên ta có thể truy cập đến từng ký tự tương tự như cách truy cập đến một phần tử của mảng. Ta khai báo một biến *string* sau đó gán giá trị cho biến string hoặc vừa khai báo vừa khởi gán giá trị.

string myString = "Hello World";

Chú ý:

Ta có thể gán (toàn bộ) một giá trị mới cho một biến kiểu string nhưng không thể thay đổi từng ký tự trong chuỗi.

Ví dụ II.1.4.1: Có thể thực hiện các lệnh sau:

string S1 = "Hello World";

S1 = "how are you?";

Ví dụ II.1.4.2: Không thể thực hiện các lệnh sau:

string S1 = "Hello World";

S1[0] =' h';

II.2. Lệnh rẽ nhánh

II.2.1. Lệnh if

Ví dụ II.2.1: Nhập một số nguyên, kiểm tra số vừa nhập là chẵn hay lẻ.

```
using System;
namespace IfExample
{
      class IF
```

```
    {
        static void Main(string[] args)
        {
            int Value;
            Console.WriteLine("Nhap mot so nguyen!");

            //Nhập một số nguyên từ bàn phím và gán cho
        value
            Value = Int32.Parse(Console.ReadLine());
            if (Value % 2 == 0)
                Console.WriteLine("Ban nhap so chan!");
            else
                Console.WriteLine("Ban nhap so le!");
            Console.Read();
        }
    }
}
```

II.2.2. Lệnh switch

Cú pháp:

switch (Biểu thức)
{
case* hằngsố_1*:
Các câu lệnh
Lệnh nhảy
case* hằngsố_2*:
Các câu lệnh
Lệnh nhảy
…
[default: các câu lệnh]
}

Ví dụ II.2.2 Hiện một thực đơn và yêu cầu người dùng chọn một

```
using System;
enum ThucDon:int
{
    Xoai,Oi,Coc
}

//Minh hoa leänh switch
class Switch
{
    static void Main(string[] args)
    {
        ThucDon Chon;
        NhapLai:
        Console.WriteLine("Chon mot mon!");
```

```
            Console.WriteLine("{0} - Xoai",
(int)ThucDon.Xoai);
            Console.WriteLine("{0} - Oi", (int)ThucDon.Oi);
            Console.WriteLine("{0} - Coc", (int)ThucDon.Coc);
            Chon=(ThucDon) int.Parse(Console.ReadLine());
            if (Chon < 0) goto NhapLai;

            switch(Chon)
            {
                  case ThucDon.Xoai:
                        Console.WriteLine("Ban chon an 1 qua
xoai!");
                        break;
                  case ThucDon.Oi:
                        Console.WriteLine("Ban chon an 1 mieng
oi!");
                        break;
                  case ThucDon.Coc:
                        Console.WriteLine("Ban chon an 1 con
coc!");
                        break;
                  default:
                        Console.WriteLine("Nha hang chua co mon
nay!");
                        break;
            }
            Console.WriteLine("Chuc ban ngon mieng!");
            Console.ReadLine();
      }
}
```

Ghi chú: Trong C#, ta không thể tự động nhảy xuống một trường hợp **case** tiếp theo nếu câu lệnh **case** hiện tại không rỗng.

II.2.3. Lệnh goto

Ví dụ II.2.3: Xuất các số từ 0 đến 9

```
      using System;
      public class Tester
      {
            public static int Main( )
            {
                  int i = 0;
                  LapLai: // nhãn
                  Console.WriteLine("i: {0}",i);
                  i++;
                  if (i < 10)     goto LapLai; // nhảy tới nhãn
      LapLai
```

```
            return 0;
        }
    }
```

II.2.4. Lệnh lặp while

Ví dụ II.2.4: Phân tích số nguyên dương N ra thừa số nguyên tố.

```
using System;
class While
{
    static void Main(string[] args)
    {
        int N, M, i;
        Console.WriteLine("Nhap so nguyen duong
(>1): ");
        N= int.Parse(Console.ReadLine());
        if (N <2)
        {
            Console.WriteLine("So khong hop le ");
            return;
        }
        string KetQua;
        KetQua = "";
        i=2;
        M=N;
        while(M>1)
        {
            if (M%i==0)
            {
                M=M/i;
                if (KetQua.Equals(""))KetQua =
KetQua + i;
                else KetQua = KetQua + "*"+i;
            }
            else i = i +1;
        }
        Console.WriteLine("So {0} o dang thua so
        nguyen to la:{1}", N, KetQua);
        Console.ReadLine();
    }
}
```

II.2.5. Lệnh do…while

Cú pháp: ***do <lênh> while <biểu_thức>.***

Vòng lặp **do …while** thực hiện ít nhất 1 lần.

Ví dụ II.2.5: Kết quả của đoạn lệnh sau là gì?

```
using System;
public class Tester
{
      public static int Main( )
      {
            int i = 11;
            do
            {
                  Console.WriteLine("i: {0}",i);
                  i++;
            } while (i < 10);
            return 0;
      }
}
```

II.2.6. Lệnh for

Cú pháp: ***for (khởi tạo; điều kiện dừng; lặp) lệnh*;**

Ví dụ II.2.6: Kiểm tra số nguyên tố

```
using System;
class NguyenTo
{
      static void Main(string[] args)
      {
            int N, i;
            Console.WriteLine("Nhap so nguyen duong (>1): ");
            N= int.Parse(Console.ReadLine());
            if (N <2)
            {
                  Console.WriteLine("So khong hop le ");
                  return;
            }
            bool KetQua;
            KetQua = true;
            for ( i = 2; i<= Math.Sqrt(N); i++)
            {
                  if (N%i==0)
                  {
                        KetQua = false;
                        break;
                  }
            }
            if (KetQua)
            Console.WriteLine("{0} la so nguyen to",N);
                  else
                        Console.WriteLine("{0} khong la so nguyen
                        to",N);
            Console.ReadLine();

      }
```

```
}
```

II.2.7. Lệnh foreach

Vòng lặp **foreach** cho phép tạo vòng lặp duyệt qua một tập hợp hay một mảng. Câu lệnh **foreach** có cú pháp chung như sau:

foreach (<kiểu thành phần> <tên truy cập thành phần > in < tên tập hợp>)
<Các câu lệnh thực hiện>

Ví dụ II.2.7: Xuất các kí tự trong chuỗi.

```
using System;
using System.Collections.Generic;
using System.Text;

public class UsingForeach
{
    public static int Main()
    {
        string S = "He no";
        string[] MonAn;

     MonAn = new string[3] {"Ga", "Vit","Ngan"};

        foreach (char item in S)
        {
            Console.Write("{0} ", item);
        }
        Console.WriteLine();
        foreach (string Si in MonAn)
        {
            Console.Write("{0} \n", Si);
        }
        System.Console.Read();
        return 0;
    }
}
```

II.2.8. Lệnh continue và break

Thỉnh thoảng chúng ta muốn quay lại vòng lặp mà không cần thực hiện các lệnh còn lại trong vòng lặp, chúng ta có thể dùng lệnh ***continue.***

Ngược lại, nếu chúng ta muốn thoát ra khỏi vòng lặp ngay lập tức chúng ta có thể dùng lệnh *break;*

Ví dụ II.2.8: Một chương trình liên tục ghi dữ liệu vào file cho đến khi chương trình nhận được tín hiệu "X". Nếu việc ghi dữ liệu thành công thì chương trình nhận được tín hiệu "0", nếu không thành công thì nhận được tín hiệu "A".

```
using System;
```

```
public class Tester
{
     public static int Main( )
     {
          string signal = "0"; // initialize to neutral
          while (signal != "X") // X indicates stop
          {
               Console.WriteLine("Dang ghi du lieu”);
               Console.WriteLine("Nhap mot chuoi tuong
               trung cho ket qua ghi file: ");
               Console.WriteLine("0 – Khong co loi!");
               Console.WriteLine("A – Co loi!");
               Console.WriteLine("X – Khong duoc ghi du
               lieu nua!");

               signal = Console.ReadLine( );
               if (signal == "0")
               {
                    Console.WriteLine("OK!\n");
                    continue;
               }
               if (signal == "A")
               {
                    Console.WriteLine("Loi! Huy bo thao tac
ghi!\n");
                    break;
               }
               //Thu hien viec huy bo du lieu ghi khong
thanh cong
               Console.WriteLine("{0} – Dang huy du lieu
               khong ghi duoc!\n", signal);
          }
          return 0;
     }
}
```

II.3. Mảng

Mảng thuộc loại dữ liệu tham chiếu (dữ liệu thực sự được cấp phát trong Heap).

II.3.1. Mảng một chiều

- Cú pháp khai báo mảng 1 chiều:

  Kiểu [] Ten_bien;

- Cú pháp cấp phát cho mảng bằng từ khóa new:

  Ten_bien = new Kiểu [Kích Thước];

Ví dụ II.3.1: Nhập một mảng số nguyên, sắp xếp và xuất ra màn hình.

```
using System;
class Array
{
     public static void NhapMang(int[] a, uint n)
     {
          int i;
          for ( i = 0; i<n; i++)
          {
               Console.WriteLine("Nhap phan tu thu {0}",i);
               a[i] = Int32.Parse(Console.ReadLine());

          }
     }
     public static void XuatMang(int[] a, uint n)
     {
          int i;
          for ( i = 0; i<n; i++)
                    Console.Write("{0} ",a[i]);

     }
     public static void SapXep(int[] a, uint n)
     {
          int i, j, temp;
          for ( i = 0; i<n-1; i++)
          {
               for ( j= i+1; j<n; j++)
               {
                    if (a[i]>a[j])
                    {
                         temp=a[i];     a[i]=a[j];
     a[j]=temp;
                    }
               }
          }
     }

     public static void Main()
     {
          int[] A;
          uint n;
          Console.WriteLine("Nhap kich thuoc mang: ");
          n = uint.Parse(Console.ReadLine());

          A= new int[n];
          NhapMang(A,n);

          Console.WriteLine("Mang vua nhap");
          XuatMang(A,n);
```

```
            SapXep(A,n);
            Console.WriteLine("Mang sau khi sap xep");
            XuatMang(A,n);
            Console.ReadLine();
      }
}
```

II.3.2. Mảng nhiều chiều

- Cú pháp khai báo mảng 2 chiều:

  Kiểu [][] Ten_bien;

Vì mảng 2 chiều là mảng mà mỗi phần tử lại là một mảng con nên ta thực hiện việc cấp phát mảng các mảng trước, sau đó lần lượt cấp phát cho các mảng con. Cú pháp cấp phát cho mảng 2 chiều như sau:

- Cấp pháp một mảng các mảng:

  Ten_bien = new Kiểu [Kích Thước][];

- Cấp phát cho từng mảng con thứ i:

  Ten_bien [i] = new Kiểu [kich thuoc mảng thứ i];

Ví dụ II.3.2: Nhập, xuất ma trận số thực.

```
using System;
class Matrix
{
      public static void NhapMaTran(float[][]a, uint n, uint
m)
      {
            int i, j;
            for ( i = 0; i < n; i++)
            {
                  for ( j = 0; j < m; j++)
                  {
                        Console.Write("Nhap phan tu thu
[{0},{1}]",i,j);
                        a[i][j] = float.Parse(Console.ReadLine());

                  }
            }
      }

      public static void XuatMaTran(float[][]a, uint n, uint
m)
      {
```

```
            int i, j;
            for ( i = 0; i < n; i++)
            {
                  for ( j = 0; j < m; j++)
                        Console.Write("     {0}",     a[i][j]);
                  Console.WriteLine();
            }
      }

      public static void Main()
      {
            float[][] A;
            uint n, m;
            Console.WriteLine("Nhap kich thuoc cua ma tran:
");
            Console.WriteLine("Nhap so dong cua ma tran: ");
            n = uint.Parse(Console.ReadLine());
            Console.WriteLine("Nhap so cot cua ma tran: ");
            m = uint.Parse(Console.ReadLine());

            A = new float[n][];

            int i;
            for( i = 0; i < n; i++) A[i] = new float[m];
            NhapMaTran(A,n, m);

            Console.WriteLine("Ma tran vua nhap");
            XuatMaTran(A,n,m);
            Console.ReadLine();
      }
}
```

II.3.3. Một số ví dụ về mảng nhiều chiều

Sau đây là một số ví dụ về mảng nhiều chiều:

- Khai báo mảng 3 chiều kiểu số nguyên với kích thước mỗi chiều là 4, 2 và 3:

```
int[,,] myArray = new int [4,2,3];
```

- Khai báo mảng 2 chiều, cấp phát và khởi gán giá trị cho mảng:

```
int[,] myArray = new int[,] { {1,2}, {3,4}, {5,6}, {7,8} };
```

hoặc

```
int[,] myArray = {{1,2}, {3,4}, {5,6}, {7,8}};
```

- Gán giá trị cho một phần tử trong mảng 2 chiều:

```
myArray[2,1] = 25;
```

II.4. Không gian tên (namespace)

Có thể hiểu không gian tên như là thư viện. Sử dụng không gian tên giúp ta tổ chức mã chương trình tốt hơn, tránh trường hợp hai lớp trùng tên khi sử dụng các thư viện khác nhau. Ngoài ra, không gian tên được xem như là tập hợp các lớp đối tượng, và cung cấp duy nhất các định danh cho các kiểu dữ liệu và được đặt trong một cấu trúc phân cấp. Việc sử dụng không gian tên trong lập trình là một thói quen tốt, bởi vì công việc này chính là cách lưu các mã nguồn để sử dụng về sau. Ngoài thư viện (namespace) do MS.NET và các hãng thứ ba cung cấp, ta có thể tạo riêng cho mình các không gian tên .

C# đưa ra từ khóa **using** để khai báo sử dụng không gian tên trong chương trình:

using < Tên namespace >

Trong một không gian tên ta có thể định nghĩa nhiều lớp và không gian tên .

Để tạo một không gian tên ta dùng cú pháp sau:

```
namespace <Tên namespace>
{
     < Định nghĩa lớp A>
     < Định nghĩa lớp B >
     .....
}
```

Ví dụ II.4.1 : Định nghĩa lớp Tester trong namespace `Programming_C_Sharp`.

```
namespace Programming_C_Sharp
{
     using System;
     public class Tester
     {
          public static int Main( )
          {
               for (int i=0;i<10;i++)
               {
                    Console.WriteLine("i: {0}",i);
               }
               return 0;
          }
     }
}
```

Ví dụ II.4.2: Khai báo không gian tên lồng nhau:

```
namespace Programming_C_Sharp
{
     namespace Programming_C_Sharp_Test1
     {
          using System;
          public class Tester
          {
               public static int Main( )
               {
                    for (int i=0;i<10;i++)
                    {
                         Console.WriteLine("i: {0}",i);
                    }
                    return 0;
               }
          }
     }
namespace Programming_C_Sharp_Test2
     {

          public class Tester
          {
               public static int Main( )
               {
                    for (int i=0;i<10;i++)
                    {
                         Console.WriteLine("i: {0}",i);
                    }
                    return 0;
               }
          }
     }
}
```

Phụ lục B - BIỆT LỆ (NGOẠI LỆ)

- Là các dạng lỗi gặp phải khi chạy chương trình (lúc biên dịch chương trình không phát hiện được). Thường là do người dùng gây ra lúc chạy chương trình.
- Kết thúc bởi từ khoá Exception.

I. Ném ra biệt lệ

Để báo động sự bất thường của chương trình.

Cú pháp:

throw [biểu thức tạo biệt lệ];

Sau khi ném ra một ngoại lệ, các đoạn lệnh sau lệnh **throw** sẽ bị bỏ qua. Chương trình thực hiện việc bắt ngoại lệ hoặc dừng.

II. Bắt ngoại lệ

- Để chương trình có tính dung thứ lỗi cao hơn, cho phép vẫn chạy chương trình đối với những lỗi không quá quan trọng. Chẳng hạn khi nhập một giá trị nguyên, người dùng vô tình nhập một ký tự, khi đó không nhất thiết phải dừng chương trình mà chỉ thông báo lỗi và cho phép người dùng nhập lại.
- Để chương trình thân thiện hơn đối với người sử dụng. Thông báo lỗi cụ thể, thay vì dạng thông báo lỗi mang tính kỹ thuật khó hiểu của hệ thống.

Việc bắt ngoại lệ được thực hiện thông qua khối ***try { } catch { }*** như sau:

try

{

Các câu lệnh có thể gây ra biệt lệ.

}

catch (khai báo biệt lệ 1) {các câu lệnh xử lý biệt lệ 1}

…

catch (khai báo biệt lệ n) {các câu lệnh xử lý biệt lệ n}

- Nếu không có khai báo biệt lệ nào trong khối ***catch*** thì khi đó ta bắt tất cả các dạng ngoại lệ do khối ***try*** gây ra.

Ví dụ: Xét đoạn chương trình

```
using System;
class Class1
{
   static void Main(string[] args)
   {
```

```
            int x=0;
            Console.WriteLine("Nhap mot so nguyen");
            x=int.Parse(Console.ReadLine());
            Console.WriteLine("So nguyen vua nhap {0}",x);
            Console.ReadLine();
        }
    }
```

- Khi chạy chương trình, nếu ta nhập một số nguyên chương trình sẽ chạy tốt. Nếu ta (vô tình) nhập một dữ liệu không phải là số nguyên (chẳng hạn nhập ký tự 'r'), chương trình sẽ dừng và báo lỗi **runtime** sau:

  An unhandled exception of type 'System.FormatException' occurred in mscorlib.dll

  Additional information: Input string was not in a correct format.

Vì vậy, để chương trình có tính dung thứ lỗi (vì đây có thể là lỗi vô tình của người sử dụng) ta cần viết lại như sau để cho người dùng nhập lại:

```
using System;
class Class1
{
      static void Main(string[] args)
      {
            int x=0;
            Console.WriteLine("Nhap mot so nguyen");
            NHAPLAI:
            try
            {
                  x=int.Parse(Console.ReadLine());
            }
            //catch(System.Exception e)
            catch(System.FormatException e)
            {
                  Console.WriteLine("Loi : " + e.ToString());
                  Console.WriteLine("Khong duoc nhap loai du
lieu khac. Hay nhap lai");
                  goto NHAPLAI;
            }
            Console.WriteLine("So nguyen vua nhap {0}",x);
            Console.ReadLine();
      }

}
```

Vì đoạn mã

```
x=int.Parse(Console.ReadLine());
```

có thể gây ra biệt lệ

```
System.FormatException
```

nên ta đặt nó trong khối ***try*** và khối ***catch*** bắt biệt lệ này. Sau đó xuất thông báo lỗi, nhưng không dừng chương trình mà cho phép nhập lại bằng lệnh nhảy tới nhãn *NHAPLAI*:

```
goto NHAPLAI;
```

Vì mọi loại biệt lệ đều dẫn xuất từ ***System.Exception*** nên ta có thể xem mọi biệt lệ là một ***System.Exception***. Do vậy, nếu ta không biết loại biệt lệ là gì ta thay lệnh

```
catch(FormatException e)
```

bằng lệnh:

```
catch(Exception e)
```

Nếu muốn bắt mọi ngoại lệ nhưng không thông báo lỗi ta có thể sử dụng khối catch rỗng như sau:

```
catch
  {
      Console.WriteLine("Khong duoc nhap loai du
      lieu khac. Hay nhap lai");
      goto NHAPLAI;
  }
```

Khi đó chương trình cho phép nhập lại nhưng không thông báo cho người dùng lỗi là gì.

Ví dụ 2: Bắt nhiều ngoại lệ

```
using System;
class Class1
{
      static void Main(string[] args)
      {
            byte x=0;
            NHAPLAI:
            Console.WriteLine("Nhap mot so nguyen");
                        try
            {
                  x=byte.Parse(Console.ReadLine());
            }
            //catch(System.Exception e)
            catch(FormatException e1)
            {
                  Console.WriteLine("Loi : " +e1.ToString());
                  Console.WriteLine("Khong duoc nhap loai du lieu
khac. Hay nhap lai");
                  goto NHAPLAI;
            }
            catch(OverflowException e2)
            {
                  Console.WriteLine("Loi : " +e2.ToString());
```

```
                Console.WriteLine("So phai thuoc doan [0..256].
Hay nhap lai");
                goto NHAPLAI;
            }
            Console.WriteLine("So nguyen vua nhap {0}",x);
            Console.ReadLine();
        }
}
```

Khi một ngoại lệ phát sinh, chương trình sẽ nhảy ngay tới khối ***catch*** gần nhất có thể bắt ngoại lệ hoặc dừng nếu không có khối catch nào có thể bắt ngoại lệ này.

III. Khối finally

Khi đặt khối ***finally*** sau các khối catch thì cho dù có biệt lệ hay không chương trình vẫn không dừng mà sẽ thực hiện khối ***finally***. (Nếu bắt được ngoại lệ thì chương trình sẽ thực hiện khối catch tương ứng trước khi thực hiện khối ***finally***).

Hãy thử thêm đoạn lệnh sau vào ví dụ trên.

```
finally
{
    Console.WriteLine("So nguyen vua nhap {0}",x);
}
```

IV. Một số ngoại lệ khác:

- ***System.OutOfMemoryException:*** Lỗi không thể cấp phát bộ nhớ.
- ***System.StackOverflowException:*** Lỗi tràn stack. Thường là do gọi đệ qui quá sâu hoặc gọi đệ qui bị lặp vô tận.
- ***System.NullReferenceException***: Lỗi xảy ra khi truy cập tới một tham chiếu trỏ tới null trong khi cần một tham chiếu trỏ tới một đối tượng thực sự hiện hữu.
- ***System.TypeInitializationException:*** Hàm constructor ném ra một ngoại lệ nhưng không có ngoại lệ catch nào bắt ngoại lệ này.
- ***System.InvalidCastException:*** Xảy ra khi không thể thực hiện việc ép kiểu tường minh từ một kiểu cơ sở hoặc một giao diện sang một kiểu dẫn xuất.
- ***System.ArrayTypeMismatchException:*** Kiểu của giá trị cần lưu vào mảng không hợp kiểu với kiểu của mảng.
- ***System.IndexOutOfRangeException:*** Truy cập ngoài mảng.
- ***System.MulticastNotSupportedException***: Lỗi liên quan tới việc không thể kết hợp 2 delegate không null vì kiểu trả về của delegate không phải là void.
- ***System.ArithmeticException***: Lỗi số học. Chẳng hạn chia cho 0, tràn dữ liệu.

- ***System.DivideByZeroException:*** Lỗi chia cho 0
- ***System.OverflowException***: Tràn dữ liệu. Chẳng hạn gán dữ liệu quá lớn cho một biến kiểu byte.
- ...

V. Một số ví dụ khác

Nên bắt biệt lệ cụ thể trước, biệt lệ tổng quát

```
using System;
public class Test
{
      public static void Main( )
      {
            Test t = new Test( );
            t.CanAChiaB(4,-5 );
            Console.ReadLine();
      }
      public void CanAChiaB(int a,int b)
      {
            try
            {
                  if (b == 0)
                        throw new DivideByZeroException( );
                        if (a*b<= 0)
                            throw new ArithmeticException( );
                  else
                  {
                        double kq = Math.Sqrt(a/b);
                        Console.WriteLine ("Ket qua = {0}",kq );
                  }
            }
            // Bat ngoai le cu the truoc
            catch (DivideByZeroException)
            {
                  Console.WriteLine("Loi chia cho 0!");
            }
            //Bat ngoai le tong quat sau;
            catch (ArithmeticException)
            {
                  Console.WriteLine("Co   loi   so   hoc   gi   gi
                  do...hehe!");
            }
      }
}
```

TÀI LIỆU THAM KHẢO

1) Phạm Hữu Khang, *C# 2005 cơ bản*, Nxb Lao Động Xã Hội, 2006.
2) Phạm Hữu Khang, *C# 2005 Tập 2-Lập trình Windows Form*, Nxb Lao Động Xã Hội, 2006.
3) Dương Quang Thiện, *Lập trình Visual C# như thế nào? Tập 1,2,3*, Nxb Tổng hợp Tp HCM, 2005.
4) Ths.Nguyễn Cẩn, *Tự học ngôn ngữ lập trình C++,*Nbx Đồng Nai,1996 .
5) Jesse Liberty, *Programming C#, 2nd Edition*, tr 1-320 ,Nxb OReilly.
6) Ben Albahari, *CSharp Essentials, 2nd Edition*, tr 1-88, Nxb OReilly.
7) VN-Guide, Lập trình Java, Nxb Thống Kê, 2000.