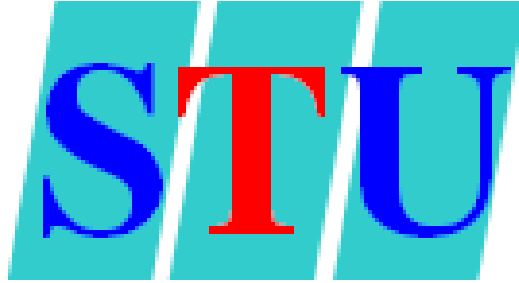


Trường Đại Học Công Nghệ Sài Gòn
Khoa Công Nghệ Thông Tin
---oOo---



Giáo trình thực hành:

LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

(Lưu hành nội bộ)

Năm 2021

BÀI 1: LÀM QUEN VỚI NGÔN NGỮ LẬP TRÌNH C#

I. MỤC TIÊU

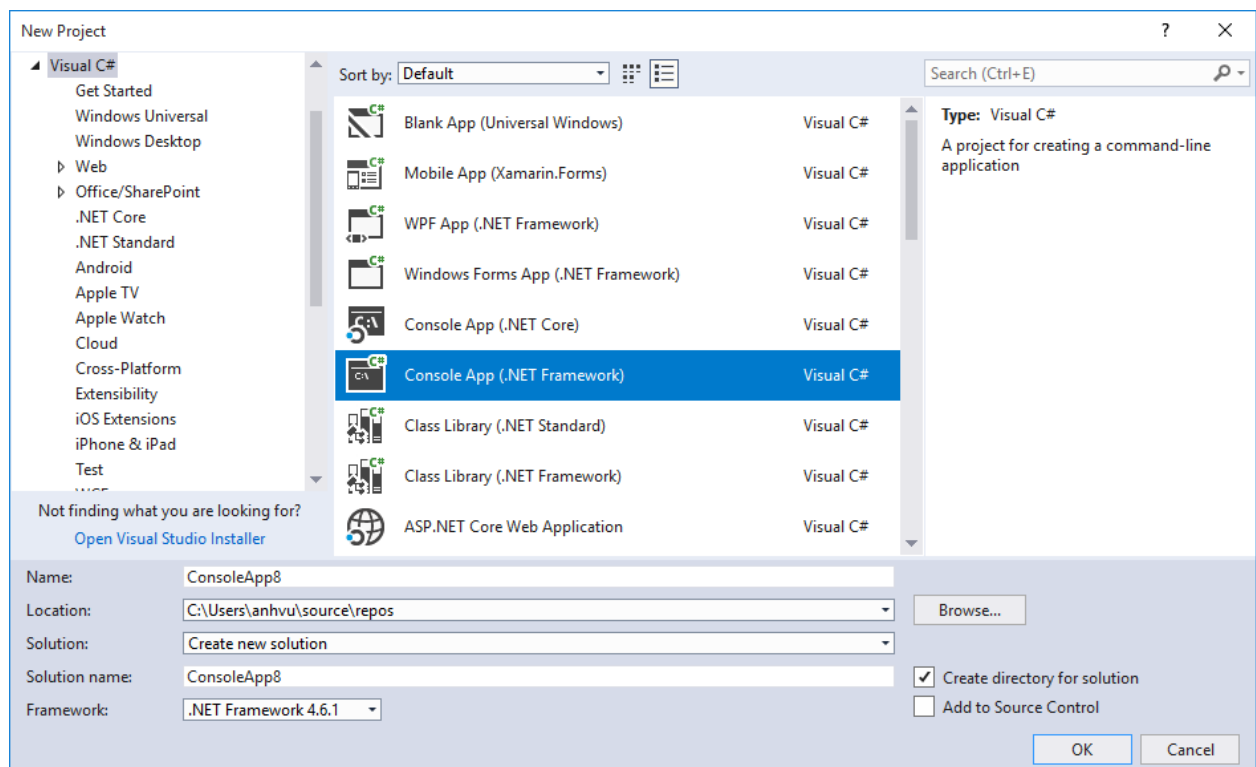
Sau khi thực hành xong bài này, sinh viên phải biết cách:

- Tạo ứng dụng dạng Console trên C#.
- Nhập/xuất dữ liệu từ bàn phím.
- Viết hàm và các cách truyền tham số.
- Sử dụng các kiểu string, DateTime, mảng.

II. TÓM TẮT LÝ THUYẾT

1. Hướng dẫn tạo Project File:

Từ menu **File | New | Project**. Cửa sổ New project xuất hiện như hình dưới đây.



- Trong hình trên:
 - Bên vùng Project Types: Chọn Visual C#

- Bên vùng Templates: Chọn Console App (.NET Framework)

- Đặt tên cho Project cần tạo trong mục Name.
- Chọn thư mục cần lưu trữ trong Location. Sau đó nhấp Ok.

2. Tạo chương trình C# đầu tiên

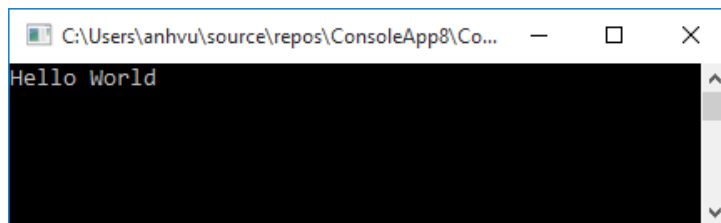
Hướng dẫn cách tạo chương trình HelloWorld sử dụng ngôn ngữ C# và môi trường phát triển MS VS.NET

- Tạo một empty project mới đặt tên là HelloWorld và lưu nó vào 1 vị trí tùy ý.
- Viết một chương trình để hiển thị một chuỗi ký tự lên màn hình

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

namespace ConsoleApp8
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World");
            Console.ReadLine();
        }
    }
}
```

- Build chương trình bằng cách **Build -> Build Solution**. Hoặc sử dụng phím tắt Ctrl+Shift+B.
- Chạy chương trình sử dụng **Debug->Start Without Debugging**. Hoặc có thể sử dụng phím tắt Ctrl+F5.
- Kết quả hiển thị ra màn hình như sau:



Chú ý: Trong C# phân biệt chữ hoa và chữ thường.

3. Kiểu dữ liệu cơ bản và cách chuyển đổi giữa các kiểu dữ liệu.**a. Kiểu dữ liệu cơ bản:**

C# chia thành hai tập hợp kiểu dữ liệu chính: Kiểu xây dựng sẵn mà ngôn ngữ cung cấp và kiểu do người dùng định nghĩa.

- Bảng các kiểu dữ liệu xây dựng sẵn:

Kiểu dữ liệu	Mô tả - cách khai báo
object	Kiểu dữ liệu cơ bản của tất cả các kiểu khác: Ví dụ: <i>object obj = null;</i>
string	Được sử dụng để lưu trữ những giá trị kiểu chuỗi cho biến: <i>string str = "Hello";</i>
int	Sử dụng để lưu trữ giá trị kiểu số nguyên: <i>int ival = 12</i>
byte	Sử dụng để lưu trữ giá trị kiểu byte: <i>byte val = 12</i>
float	Sử dụng để lưu trữ số thực: <i>float val = 1.23F</i>
bool	Cho phép biến được lưu trữ giá trị đúng hoặc sai: <i>bool val1 = false ; bool val2 = true</i>
char	Cho phép một biến lưu trữ ký tự: <i>char cval = 'A';</i>

b. Kiểu string

- ❖ Chuỗi trong C# là một kiểu dựng sẵn như int , long ... có đầy đủ tính chất mềm dẻo , mạnh mẽ và dễ dùng .
 - Để khai báo một đối tượng chuỗi ta sử dụng từ khóa **string**.
- ❖ Tạo chuỗi mới :
 - Cú pháp : `string = "` Khai báo và gán một chuỗi `"`
 - Một số ký tự đặc biệt như : `"\n"`, `"\\"`, `"\t"`... đại diện cho ký tự xuống dòng , dấu xuyệt `"\"`

, dấu tab.

- Ví dụ khai báo : `duongdan = "c:\\Windows\\system32";`

=> Sẽ có giá trị là : `c:\\Windows\\system32`

- C# cũng cung cấp cho ta cách khai báo theo đúng đường dẫn gốc bằng cách thêm ký tự `@`

- Ví dụ : `duongdan = @"c:\\Windows\\system32"`

❖ Phương thức `ToString ()` :

- Dùng để chuyển đổi một đối tượng bất kỳ sang kiểu chuỗi

Ví dụ : `int n = 5 ;`

`string s = n.ToString();`

Bây giờ chuỗi `s` sẽ có giá trị là `5` . Bằng cách này ta cũng có thể tạo ra một chuỗi mới .

❖ Các thao tác chuỗi :

1 . Các hàm thành viên :

- **Empty** : biến thành viên tính đại diện cho một chuỗi rỗng

- **Compare()** : phương thức tính dùng so sánh hai chuỗi

- **CompareOrdinal ()** : So sánh hai chuỗi không quan tâm đến ngôn ngữ

- **Concat ()** : tạo chuỗi mới từ nhiều chuỗi

- **Copy ()** : tạo một bản sao

- **Equals ()** : so sánh hai chuỗi có giống nhau

- **Join ()** : ghép nối nhiều chuỗi

- **Chars ()** : Indexer của chuỗi

- **Length** : chiều dài chuỗi

- **Insert()** : chèn một chuỗi khác vào chuỗi

- **SubString ()** : Lấy một chuỗi con

- **ToLower ()** : tạo chuỗi chữ thường

- **ToUpper ()** : tạo chuỗi chữ hoa

- **Trim ()** : Cắt bỏ khoảng trắng hai đầu chuỗi

VD 1. Ghép chuỗi

```
string a = " Xin ";
string b = " Chào ";
string c = a + " " + b ; // Ket qua Xin Chào
```

VD 2 : Lấy ký tự

```
string s = " XIN CHAO " ;
char c = s[1] ; // kết quả là X
```

VD3 : Lấy chuỗi con :

```
string s = "XIN CHAO".Substring(3); // kết quả s = "CHAO"  
string s = "XIN CHAO".Substring(4,3) ; // kết quả s = "CHA"
```

c. Kiểu DateTime

Khai báo kiểu dữ liệu: *DateTime dt*

Có các định dạng tùy chỉnh sau đây: y (năm), M (tháng), d (ngày), h (giờ 12), H (giờ 24), m (phút), s (thứ hai), f (phần nhỏ của giây), F (phần nhỏ của giây, không bao gồm số 0 sau cùng), t (PM hoặc AM) và z (múi giờ).

```
//Khởi tạo thời gian date time 2008-03-09 16:05:07.123  
DateTime dt = new DateTime(2008, 3, 9, 16, 5, 7, 123);
```

```
String.Format("{0:y yy yyy yyyy}", dt); // "8 08 008 2008" năm  
String.Format("{0:M MM MMM MMMM}", dt); // "3 03 Mar March" tháng  
String.Format("{0:d dd ddd dddd}", dt); // "9 09 Sun Sunday" ngày  
String.Format("{0:h hh H HH}", dt); // "4 04 16 16" giờ 12/24  
String.Format("{0:m mm}", dt); // "5 05" phút  
String.Format("{0:s ss}", dt); // "7 07" giây  
String.Format("{0:f ff fff ffff}", dt); // "1 12 123 1230" phần nhỏ  
của giây  
String.Format("{0:F FF FFF FFFF}", dt); // "1 12 123 123" phần nhỏ  
của giây(không có số 0)  
String.Format("{0:t tt}", dt); // "P PM" A.M. or  
P.M.  
String.Format("{0:z zz zzz}", dt); // "-6 -06 -06:00" múi giờ
```

Sử dụng dấu phân cách "/" , ":". Các ký tự này sẽ được chèn vào kết quả đầu ra, để xác định thời gian.

```
//Dấu phân cách trong định dạng "/"  
String.Format("{0:d/M/yyyy HH:mm:ss}", dt); // "9/3/2008 16:05:07"  
//Dưới đây là một vài ví dụ về tùy chỉnh định dạng kiểu DateTime  
//Định dạng tháng/ngày/năm với ký tự "0" và không có ký tự "0"  
String.Format("{0:M/d/yyyy}", dt); // "3/9/2008"  
String.Format("{0:MM/dd/yyyy}", dt); // "03/09/2008"  
  
//Lấy tên của ngày/tháng  
String.Format("{0:ddd, MMM d, yyyy}", dt); // "Sun, Mar 9, 2008"  
String.Format("{0:dddd, MMMM d, yyyy}", dt); // "Sunday, March 9, 2008"  
  
//Lấy năm với 2, 4 ký tự  
String.Format("{0:MM/dd/yy}", dt); // "03/09/08"  
String.Format("{0:MM/dd/yyyy}", dt); // "03/09/2008"
```

Định dạng Datetime với các kiểu mẫu được định sẵn.

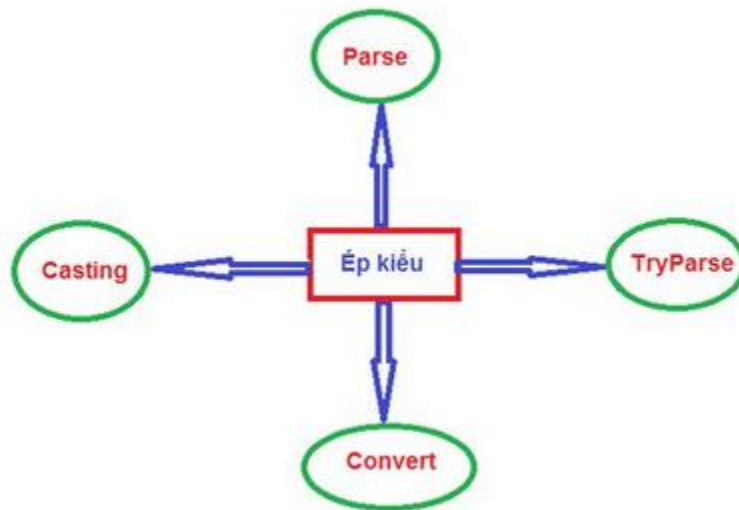
Trong `DateTimeFormatInfo` có các kiểu mẫu định sẵn cho các culture hiện hành. Ví dụ `ShortTimePattern` định dạng `h:mm tt` cho **en-US** culture và định dạng `HH:mm` cho **de-DE** culture. Bảng sau cho thấy các kiểu mẫu định sẵn trong `DateTimeFormatInfo` được định theo **en-US** culture. Cột đầu tiên chỉ ra các kiểu mẫu được định sẵn.

Kiểu mẫu	DateTimeFormatInfo	Giá trị định sẵn (en-US culture)
t	ShortTimePattern	h:mm tt
d	ShortDatePattern	M/d/yyyy
T	LongTimePattern	h:mm:ss tt
D	LongDatePattern	dddd, MMMM dd, yyyy
f	(combination of D and t)	dddd, MMMM dd, yyyy h:mm tt
F	FullDateTimePattern	dddd, MMMM dd, yyyy h:mm:ss tt
g	(combination of d and t)	M/d/yyyy h:mm tt
G	(combination of d and T)	M/d/yyyy h:mm:ss tt
m, M	MonthDayPattern	MMMM dd
y, Y	YearMonthPattern	MMMM, yyyy
r, R	RFC1123Pattern	ddd, dd MMM yyyy HH':'mm':'ss 'GMT'(*)
s	SortableDateTimePattern	yyyy'-'MM'-'dd'T'HH':'mm':'ss(*)
u	UniversalSortableDateTimePattern	yyyy'-'MM'-'dd HH':'mm':'ss'Z'(*)
		(*) = không phụ thuộc vào culture

Ví dụ sau minh họa cách sử dụng kiểu mẫu định sẵn.

```
String.Format("{0:t}", dt); // "4:05 PM" ShortTime
String.Format("{0:d}", dt); // "3/9/2008" ShortDate
String.Format("{0:T}", dt); // "4:05:07 PM" LongTime
String.Format("{0:D}", dt); // "Sunday, March 09, 2008" LongDate
String.Format("{0:f}", dt); // "Sunday, March 09, 2008 4:05 PM"
LongDate+ShortTime
String.Format("{0:F}", dt); // "Sunday, March 09, 2008 4:05:07 PM"
FullDateTime
String.Format("{0:g}", dt); // "3/9/2008 4:05 PM"
ShortDate+ShortTime
String.Format("{0:G}", dt); // "3/9/2008 4:05:07 PM"
ShortDate+LongTime
String.Format("{0:m}", dt); // "March 09" MonthDay
String.Format("{0:y}", dt); // "March, 2008" YearMonth
String.Format("{0:r}", dt); // "Sun, 09 Mar 2008 16:05:07 GMT" RFC1123
String.Format("{0:s}", dt); // "2008-03-09T16:05:07"
SortableDateTime
```

d. Chuyển đổi kiểu dữ liệu



- **Parse:** Phương thức Parse là phương thức được sử dụng khá phổ biến khi chúng ta muốn chuyển đổi một chuỗi sang một kiểu dữ liệu tương ứng. Mỗi kiểu dữ liệu cơ bản trong C# đều có phương thức Parse để chuyển đổi sang kiểu dữ liệu đó. Một số ví dụ các câu lệnh minh họa cho việc chuyển đổi sử dụng phương thức Parse:

```
int a = Int32.Parse("123"); //a sẽ mang giá trị 123
float b = Float.Parse("20.7"); //b sẽ mang giá trị 20.7
bool c = Boolean.Parse("true"); //c sẽ mang giá trị true
```

Nếu như chuỗi chúng ta truyền vào là rỗng, không đúng định dạng hoặc vượt quá giá trị cho phép thì chúng ta sẽ nhận được các Exception tương ứng. Ví dụ:

```
int a = Int32.Parse("Hello"); //sai định dạng, FormatException
byte b = Byte.Parse("10000000000"); //quá giới hạn, OverflowException
bool c = Boolean.Parse(null); //tham số là null, ArgumentNullException
```

- **TryParse:** Giống như Parse, TryParse cũng là phương thức được tích hợp sẵn trong các lớp kiểu dữ liệu cơ bản của C#. Tuy nhiên, cú pháp của TryParse có phần khác với Parse. Cụ thể, tham số thứ nhất của TryParse là chuỗi cần chuyển đổi và tham số thứ hai là biến sẽ chứa giá trị đã được chuyển đổi, biến thứ hai này phải được đánh dấu là out . Một số ví dụ minh họa

```
int a;
```



```
Int32.TryParse("123", out a); //a sẽ mang giá trị 123
bool b;
Boolean.TryParse("false", out b); //b sẽ mang giá trị false
```

Điểm khác biệt thứ hai của TryParse so với Parse là phương thức TryParse không ném ra các ngoại lệ như Parse mà sẽ trả về các giá trị true (chuyển đổi thành công) hoặc false (chuyển đổi thất bại, biến mang giá trị mặc định).

```
int a;
Int32.TryParse("hello",out a);//trả về giá trị false, a=0;
bool b;
Boolean.TryParse("", out b); //trả về giá trị false, b=False
```

Chú ý: Ngoài ra, phương thức TryParse sẽ thực thi nhanh hơn phương thức Parse vì TryParse không ném ra ngoại lệ

- **Convert:** Lớp Convert là một lớp tiện ích trong C# cung cấp cho chúng ta rất nhiều phương thức tĩnh khác nhau để chuyển đổi từ một kiểu dữ liệu này sang kiểu dữ liệu khác. Tham số mà các phương thức trong Convert nhận không nhất thiết phải là chuỗi mà có thể ở nhiều kiểu dữ liệu khác nhau (int, bool, double...). Ví dụ:

```
int a = Convert.ToInt32("123");//chuyển chuỗi 123 sang số nguyên
bool b = Convert.ToBoolean(13);//chuyển số 13 sang kiểu bool
```

Các phương thức trong lớp Convert sẽ trả về giá trị mặc định nếu như tham số truyền vào là null. Còn trong các trường hợp sai định dạng hoặc vượt quá giới hạn thì các phương thức đó sẽ ném ra các ngoại lệ tương tự như phương thức Parse. Ví dụ:

```
bool a = Convert.ToBoolean("khoaaimon"); //FormatException
int b = Convert.ToInt32("123456787654"); //OverflowException
double d = Convert.ToDouble(null); //trả về giá trị mặc định
```

- **Casting(Ép kiểu):** Ép kiểu là cách chúng ta có thể sử dụng khi muốn chuyển đổi giữa các kiểu dữ liệu có tính chất tương tự nhau (thường là số). Ví dụ :

```
int a = 100;
float b = a; //chuyển đổi ngầm định, b = 100
int c = (int)b; //chuyển đổi rõ ràng, c = 100
int a = 100;
```

Ép kiểu chỉ được sử dụng khi chúng ta biết rõ rằng đối tượng đó chứa kiểu dữ liệu tương ứng với kiểu mà ta cần chuyển tới. Ví dụ như các trường hợp sau sẽ là các lỗi cú pháp trong lập trình:

```
string a = "1234";  
int b = (int)a; //lỗi, không thể ép kiểu chuỗi sang kiểu số  
bool c = true;  
double d = (double)c; //lỗi, không thể ép kiểu bool sang kiểu  
double
```

4. Mảng một chiều

- **Cách khai báo:** Để khai báo một mảng trong C# chúng ta sử dụng cú pháp như sau:

<DataType>[] <ArrayName>

Trong đó:

DataType: kiểu dữ liệu như int, float....

ArrayName: tên mảng (hãy đặt tên nào đó có ý nghĩa và mang tính chất gợi mở nhé).

[]: cặp dấu ngoặc vuông này chỉ ra cho trình biên dịch biết chúng ta đang khai báo một mảng.

Vd: `int [] x;`

- Bây giờ là vấn đề khởi tạo mảng tức là quá trình khai báo số lượng các phần tử của mảng và gán giá trị cho các phần tử đó. Có 4 cách như sau để khởi tạo giá trị của mảng:

C1: Khai báo và chỉ ra số phần tử của mảng.

Cú pháp:

[DataType] [] [ArrayName] = new [DataType] [number of elements];

number of elements: số phần tử của mảng.

VD: khai báo một mảng số nguyên lưu trữ 5 phần tử.

```
int[] a = new int[5]
```

C2: Khai báo sau đó mới khởi tạo số phần tử của mảng.

Cú pháp

[DataType] [] [ArrayName];

[ArrayName] = new [DataType] [number of elements];

VD: khai báo một mảng số nguyên lưu trữ 5 phần tử.

```
int [] a;
```

```
a = new int[5];
```

C3: Khai báo, chỉ ra số lượng các phần tử mảng và gán các giá trị ban đầu cho các phần tử mảng.

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

C4: Khai báo, không chỉ ra số lượng các phần tử mảng và gán giá trị cho các phần tử của mảng.

- Truy cập các phần tử mảng: một mảng là một danh sách các phần tử có cùng kiểu dữ liệu, các phần tử đó được đánh số thứ tự bắt đầu từ 0 đến n-1 (Trong đó n là số phần tử của mảng).
Như vậy để truy cập đến 1 phần tử của mảng thì chúng ta sử dụng một số nguyên để chỉ ra số thứ tự của phần tử đó trong mảng, phần tử nguyên này được gọi là chỉ số (index).
Cú pháp tổng quát để truy cập đến phần tử thứ i của mảng là:

[tên mảng] [i-1];

VD: gán giá trị 5 cho phần tử thứ 3 của mảng. `a[2] = 5;`

5. Mảng hai chiều

Để khai báo một mảng trong C# chúng ta sử dụng cú pháp như sau:

`<DataType>[,] <ArrayName>`

Trong đó:

DataType: kiểu dữ liệu như int, float....

ArrayName: tên mảng (hãy đặt tên nào đó có ý nghĩa và mang tính chất gợi mở nhé).

[,]: cặp dấu ngoặc vuông và dấu phẩy này chỉ ra cho trình biên dịch biết chúng ta đang khai báo một mảng 2 chiều.

- Bây giờ là vấn đề khởi tạo ma trận tức là quá trình khai báo số lượng các phần tử của ma trận và gán giá trị cho các phần tử đó. Có 4 cách như sau để khởi tạo giá trị của ma trận:

C1: Khai báo và chỉ ra số phần tử của ma trận.

Cú pháp:

```
[DataType] [, ] [ArrayName] = new [DataType] [number of elements];
```

number of elements: số phần tử của mảng.

VD: khai báo một ma trận 2 dòng, 2 cột như sau:

```
int[,] a = new int[2, 2]
```

C2: Khai báo sau đó mới khởi tạo số phần tử của ma trận.

Cú pháp

```
[DataType] [, ] [ArrayName];
```

```
[ArrayName] = new [DataType] [number of elements];
```

VD: khai báo một ma trận số nguyên lưu trữ 2 dòng, 2 cột.

```
Int [,] a;
```

```
a = new int[2, 2];
```

C3: Khai báo, chỉ ra số lượng các phần tử ma trận và gán các giá trị ban đầu cho các phần tử của ma trận

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

C4: Khai báo, không chỉ ra số lượng các phần tử ma trận và gán giá trị cho các phần tử của ma trận.

- Truy cập các phần tử ma trận: Cú pháp tổng quát để truy cập đến phần tử thứ i,j của ma trận là:

```
[tên mảng] [i-1, j-1];
```

VD: gán giá trị 5 cho phần tử thứ 1,2 của mảng. `a[1, 2] = 5;`

6. Cách truyền tham số

- Khai báo phương thức (hàm)
- Truyền tham số dạng in (ø)
- Truyền tham số dạng out
- Truyền tham số dạng ref

a. Khai báo phương thức

```
[modifiers] return_type MethodName([parameters])  
{  
    // Than phuong thuc  
}
```

Vì dụ:

```
public static void Xuat(StrHocSinh hs)
{
    Console.WriteLine("Ma so: {0}. Ho ten: {1}", hs.MaSo, hs.HoTen);
    //Cau lenh xuat hoc sinh
}
```

b. Phương thức dạng in

- Thân phương thức chỉ tham khảo giá trị của tham số không thay đổi giá trị của tham số
- Ví dụ:

```
public static void Xuat(StrHocSinh hs)
{
    Console.WriteLine("Ma so: {0}. Ho ten: {1}", hs.MaSo, hs.HoTen);
    //Cau lenh xuat hoc sinh
}
Gọi hàm trong hàm Main:
Xuat(hs);
```

c. Phương thức dạng out

- Ra khỏi hàm giá trị của tham số sẽ thay đổi, bên trong hàm phải cấp phát (khởi tạo) giá trị cho tham số.
- Ví dụ:

```
public static void Nhap(out StrHocSinh hs)
{
    hs = new StrHocSinh();
    //Cau lenh nhap hoc sinh
}
• Gọi trong hàm Main:
Nhap(out hs);
```

d. Phương thức dạng ref

- Thân phương thức cấp phát (khởi tạo) giá trị của tham số trước khi sử dụng. Ra khỏi hàm giá trị của tham số có thể thay đổi.
- Ví dụ:

```
public static void TinhDiemTrungBinh(ref StrHocSinh hs)
{
    hs.DTB = (hs.Toan+ hs.Van)/2;
}
• Gọi trong hàm Main:
TinhDiemTrungBinh(ref hs);
```

III. NỘI DUNG THỰC HÀNH**1. Chương trình mẫu (3 điểm):**

Cho đoạn mã như sau:

STT	MÃ LỆNH	GHI CHÚ
1	public static void GetNumber(ref int x, ref int y)	
2	{	
3	x = 5;	
4	y = 10;	
5	}	
6	static void Main(string[] args)	
7	{	
8	int a = 0, b = 0;	
9	GetNumber(a, b);	
10	Console.WriteLine("a={0}\n b={1}", a, b);	
11	Console.ReadLine();	
12	}	

Yêu cầu:

- Đoạn mã trên có lỗi, hãy biên dịch và sửa lỗi. Cho biết dòng lệnh nào gây ra lỗi.
- Đoạn mã sau cho biết khai báo một kiểu dữ liệu DateTime

STT	MÃ LỆNH	GHI CHÚ
1	static void Main(string[] args)	
2	{	
3	DateTime value = new DateTime(2014, 11, 20);	
4	Console.WriteLine(value);	
5	Console.WriteLine(value == DateTime.Today);	
6	// Khai báo kiểu DateTime	
7	string simpleTime = "20/1/2000";	
8	DateTime time = DateTime.Parse(simpleTime);	
9	Console.WriteLine(time);	
10	Console.ReadLine();	
11	}	

Yêu cầu:

- Đọc hiểu từng câu lệnh
- Xuất ra hôm nay là ngày bao nhiêu?

2. Bài tập vận dụng (4 điểm)

Đoạn mã sau cho biết nhập, xuất một mảng 2 chiều như sau:

Bài 1: LÀM QUEN VỚI NGÔN NGỮ LẬP TRÌNH C#

STT	MÃ LỆNH	GHI CHÚ
1	<code>class Program</code>	
2	<code>{</code>	
3	<code>static void NhapMang(int[,] a, out int n)</code>	
4	<code>{</code>	
5	<code>Console.Write("Nhap n:");</code>	
6	<code>n = Convert.ToInt32(Console.ReadLine());</code>	
9	<code>for (int i = 0; i < n; i++)</code>	
10	<code>for (int j = 0; j < n; j++)</code>	
11	<code>{</code>	
12	<code>Console.Write("Nhap a[{0},{1}] = ", i, j);</code>	
13	<code>a[i, j] = Convert.ToInt32(Console.ReadLine());</code>	
14	<code>}</code>	
15	<code>}</code>	
16	<code>static void XuatMang(int[,] a, int n)</code>	
17	<code>{</code>	
18	<code>for (int i = 0; i < n; i++)</code>	
19	<code>{</code>	
20	<code>for (int j = 0; j < m; j++)</code>	
22	<code>Console.Write(a[i, j] + "\t");</code>	
23	<code>Console.WriteLine();</code>	
24	<code>}</code>	
25	<code>}</code>	
26	<code>static int KTNT(int n)</code>	
27	<code>{</code>	
28	<code>if(n<2)</code>	
29	<code>return 0;</code>	
30	<code>else</code>	
31	<code>{</code>	
32	<code>for(int i=2; i <= n/2; i++)</code>	
33	<code>if(n % i == 0)</code>	
34	<code>return 0;</code>	
35	<code>return 1;</code>	
36	<code>}</code>	
36	<code>}</code>	
38	<code>static void Main(string[] args)</code>	
39	<code>{</code>	
40	<code>const int MAX = 10;</code>	
41	<code>int n, m;</code>	
42	<code>int[,] a = new int[MAX, MAX];</code>	
43	<code>NhapMang(a, out n);</code>	
44	<code>XuatMang(a, n);</code>	
45	<code>Console.Read();</code>	

46	}	
47	}	

Yêu cầu:

- Build và chạy thử chương trình
- Đoạn mã trên bị lỗi. Yêu cầu sửa lỗi.
- Ở dòng thứ 42, bỏ từ khóa new được không? Cho biết ý nghĩa của từ khóa.
- Viết hàm INCC: Xuất ra các phần tử trên đường chéo chính
- Xây dựng hàm xuất ra các số nguyên tố
- Xây dựng hàm **Tính tổng** các phần tử trong mảng

3. Bài tập tổng hợp (3 điểm)

Viết chương trình cho phép nhập vào một ma trận kích thước $n \times n$, mỗi phần tử là một số nguyên. Sau đó thực hiện các công việc sau:

- Xuất ma trận vừa nhập ra màn hình
- Xuất ra tổng các số nguyên tố.
- Xuất ra tổng các phần tử trên dòng k của ma trận.
- Viết hàm đếm các phần tử âm, hàm đếm phần tử dương trong ma trận.
- Viết hàm đếm số lần xuất hiện của phần tử x trong ma trận.
- Xuất ra màn hình các số hoàn thiện (số hoàn thiện là số có tích các ký số = tổng các ký số. Ví dụ số 123 là số hoàn thiện vì $1*2*3 = 1+2+3$)
- Nhập vào một số nguyên n. Cho biết chỉ số vị trí các phần tử có giá trị bằng n trong ma trận.

4. Bài tập làm thêm

- Viết hàm đếm các phần tử nhỏ hơn x trong ma trận.
- Viết hàm tính tổng các phần tử nằm ở dòng chẵn trong ma trận.
- Viết hàm kiểm tra xem 1 ma trận vuông có phải là ma trận đơn vị không ?

Gợi ý : ma trận đơn vị là ma trận các phần tử trên đường chéo chính = 1, các phần tử khác = 0.

- Viết hàm kiểm tra 1 ma trận vuông có đối xứng không. (ma trận đối xứng khi và chỉ khi $a_{ij} = a_{ji}$ với mọi i,j)

- e) Tính cộng hai ma trận.
- f) Tính nhân hai ma trận.
- g) Viết hàm hoán đổi giá trị các phần tử nằm đối xứng trên đường chéo chính và phụ của ma trận vuông. VD :

Ma trận :

1	2	3
4	5	6
7	8	9

→ Kết quả : ($1 \leftrightarrow 3$, $7 \leftrightarrow 9$)

3	2	1
4	5	6
9	8	7

- h) Viết chương trình nhập vào mảng một chiều các số nguyên. Xuất ra giá trị của mảng . Tính tổng các phần tử trong mảng đó.
- i) Viết chương trình nhập vào bán kính đường tròn. Hãy tính chu vi, diện tích hình tròn, sau đó xuất kết quả ra màn hình.
- j) Viết chương trình cho phép nhập vào giờ, phút, giây. Hãy đổi sang giây, và xuất kết quả ra màn hình.
- k) Viết chương trình nhập vào 4 số nguyên a, b, c, d. Tính trung bình cộng của 4 số, xuất kết quả ra màn hình.
- l) Viết chương trình nhập vào mảng các số thực và xuất ra các phần tử âm có trong mảng đó.
- m) Viết chương trình nhập mảng các số nguyên xuất ra các phần tử lẻ.
- n) Viết chương trình nhập vào một mảng một chiều các số nguyên và xuất ra màn hình các phần tử là số chính phương nằm tại những vị trí lẻ trong mảng.

BÀI 2: XÂY DỰNG LỚP VÀ SỬ DỤNG ĐỐI TƯỢNG

I. MỤC TIÊU

- Xây dựng lớp: khai báo các thành phần dữ liệu, viết phương thức tạo (constructor), các thành phần cập nhật dữ liệu và trả về dữ liệu (property), các phương thức (method), và các phép toán (operator).
- Sử dụng đối tượng: viết chương trình chính có sử dụng những đối tượng của lớp đã được xây dựng.

II. TÓM TẮT LÝ THUYẾT

1. Đối tượng và lớp đối tượng

Đối tượng (*object*) là một thực thể phần mềm bao gồm dữ liệu và những xử lý trên dữ liệu đó. Thành phần dữ liệu được gọi là thuộc tính (*attribute*), thành phần cập nhật và trả về dữ liệu (Property), thành phần xử lý được gọi là phương thức (*method*).

Thuộc tính nên được xây dựng như một thành phần chứa bên trong đối tượng mà các đối tượng khác bên ngoài không thể truy xuất đến được. Để có thể cập nhật hoặc lấy dữ liệu trên thuộc tính, ta phải sử dụng thành phần cập nhật và trả về dữ liệu. Phương thức được hiểu như các hàm, có tác dụng xử lý, tính toán dựa trên những thuộc tính bên trong đối tượng

Trong thành phần cập nhật & trả về dữ liệu, ta sử dụng 2 phương thức con là **get** và **set**.

Get: Trả kết quả về. Cấu trúc:

```
get { return <tên thuộc tính>; }
```

Set: Cập nhật giá trị cho thuộc tính. Cấu trúc:

```
set { <tên thuộc tính> = value; }
```

Tập hợp các đối tượng có cùng kiểu thuộc tính và phương thức tạo thành lớp đối tượng (class).

Trong C#, đối tượng được khai báo như sau:

```
class <Tên lớp>
{
    <Scope> + khai báo các thành phần
}
```

Lưu ý: 1 số quy ước đặt tên

Bài 2: XÂY DỰNG LỚP VÀ SỬ DỤNG ĐỐI TƯỢNG

Thông thường, người ta thường quy ước 1 số cách đặt tên các lớp, phương thức, biến cho dễ nhớ như sau:

- Tên lớp: thường bắt đầu bằng chữ C. Cho biết đối tượng khai báo là 1 Class.
 - o Ví dụ: CPhanSo, CDiem, CDaThuc, ...
- Tên thuộc tính: thường bắt đầu bằng chữ quy định kiểu dữ liệu của thuộc tính đó
 - o Ví dụ: int iDiem, float fDiem, double dDiem, bool bDiem,...
- Tên các phương thức cập nhật & trả về dữ liệu thuộc tính: Thường được viết như tên thuộc tính, nhưng bỏ đi các chữ cái đầu tiên quy định kiểu dữ liệu.
 - o Ví dụ: int Diem { } , float Diem { } , double Diem { } , bool Diem { }...
- Tên các phương thức: Được đặt ngắn gọn, dễ gọi nhớ
 - o Ví dụ: void Xuat(), void Nhap(), void KTNguyenTo(), ...

Việc đặt tên đôi khi rất quan trọng, nó giúp chúng ta dễ dàng kiểm soát được các dòng code. Đặt tên rõ ràng sẽ giúp cho việc gọi nhớ tốt hơn, nâng cao khả năng lập trình hơn.

Ví dụ xây dựng lớp đối tượng PhanSo:

```
class CPhanSo
{
    //Các thuộc tính - Attributes
    private int iTuSo;
    private int iMauSo;
    //Các thành phần cập nhật & trả về dữ liệu - properties
    public int TuSo
    {
        get { return iTuSo; }
        set { iTuSo = value; }
    }
    public int MauSo
    {
        get { return iMauSo; }
        set { iMauSo = value; }
    }
    //Các phương thức - Method
    private void Function1()
    {
        //Viết xu li cho phuong thuc Function1
    }
}
```

```
    }  
    public void Function2()  
    {  
        //Viết xử lý cho phương thức Function2  
    }  
}
```

2. Phạm vi (Scope)

Các thành phần bên trong lớp đối tượng đều có phạm vi hoạt động riêng biệt. Có 3 loại phạm vi: private, protected, public. Trong phạm vi bài này, chúng ta chỉ xét đến 2 loại phạm vi private, public.

Private:

Các thành phần lớp đối tượng có phạm vi là private chỉ được sử dụng bên trong lớp đó.

Public:

Các thành phần lớp đối tượng có phạm vi là public được sử dụng cả bên trong và bên ngoài lớp đó.

Ví dụ:

```
class CPhanSo  
{  
    private void Function1()  
    {  
        //Viết xử lý cho phương thức Function1  
    }  
    public void Function2()  
    {  
        //Viết xử lý cho phương thức Function2  
    }  
}
```

```
class Program  
{  
    static void Main(string[] args)  
    {  
        CPhanSo phanso = new CPhanSo();  
        phanso.Function1(); // hàm không thể sử dụng  
        phanso.Function2(); // hàm có thể sử dụng  
    }  
}
```

```
}  
}
```

Trong ví dụ trên, ta khởi tạo 1 đối tượng CphanSo trong class Program. Do phương thức Function1 được khai báo private nên nó chỉ được sử dụng chỉ bên trong lớp CphanSo. Do đó, ta không thể sử dụng Function1 trong lớp Program, mà chỉ có thể sử dụng phương thức Function2

3. Con trỏ this

Con trỏ **this** được sử dụng trong phạm vi của lớp đối tượng. Nó đại diện cho con trỏ đang trỏ đến đối tượng gọi thực hiện phương thức đó.

Toán tử New

Trong C#, toán tử new dùng để cấp phát bộ nhớ cho 1 biến con trỏ thuộc 1 kiểu dữ liệu bất kì.

Ví dụ

```
CPhanSo phanso = new CPhanSo();
```

Khác với C++, để giải phóng & thu hồi bộ nhớ của biến con trỏ sau khi sử dụng xong, ta cần phải dùng toán tử delete. Tuy nhiên, trong C#, hệ thống tự động thu hồi & giải phóng bộ nhớ khi chúng không còn được sử dụng nữa thông qua Garbage Collector. Do đó, trong C# không có toán tử delete.

4. Hàm dựng và hàm hủy

a. Hàm dựng

Hàm dựng (constructor) của một lớp đối tượng là 1 phương thức đặc biệt được gọi tự động ngay khi đối tượng đó được cấp phát bộ nhớ.

Tính chất

- Gọi tự động khi đối tượng với được cấp phát bộ nhớ
- Có thể có tham số đầu vào
- Không có giá trị trả về
- Một lớp đối tượng có thể có nhiều hàm dựng với các tham số khác nhau
- Hàm dựng có tên trùng với tên lớp

Bất kì lớp đối tượng nào cũng đều có hàm dựng. Mặc định, nếu không định nghĩa & khai báo hàm dựng. Hàm dựng mặc định (default constructor) - không có tham số đầu vào, bên trong hàm không thực hiện bất cứ công việc gì - sẽ được tự động thêm vào.

Ví dụ

```
class CPhanSo  
{
```

```
<cac thuc tinh>
...
public CPhanSo() //default contrusctor
{

}

public CPhanSo(int tu, int mau)
{

}

public CPhanSo(CPhanSo a)
{

}

}

static void Main(string[] args)
{
    CPhanSo phanso = new CPhanSo();
    CPhanSo phanso1 = new CPhanSo(1, 2); // phanso1 được khởi tạo từ 2 tham
số kiểu int (tử, mẫu)
    CPhanSo phanso2 = new CPhanSo(phanso1); // phanso2 được khởi tạo từ 1
tham số kiểu phanso khác
}
```

b. Hàm hủy

Hàm hủy (destructor) của một lớp đối tượng là 1 phương thức đặc biệt được gọi tự động ngay khi đối tượng đó được hủy & giải phóng bộ nhớ

Tính chất

- Gọi tự động khi đối tượng với bị hủy và không cần được gọi.
- Không có tham số đầu vào
- Không có giá trị trả về
- Một lớp đối tượng chỉ có duy nhất 1 hàm hủy
- Hàm hủy có tên trùng với tên lớp kèm theo dấu ~ ở phía trước. Ngoài ra hàm hủy không có Scope ở phía đầu khi khai báo.

Bài 2: XÂY DỰNG LỚP VÀ SỬ DỤNG ĐỐI TƯỢNG

Bất kì lớp đối tượng nào cũng đều có hàm hủy. Mặc định, nếu không định nghĩa & khai báo hàm hủy. Hàm hủy mặc định (default destructor) - không có tham số đầu vào, bên trong hàm không thực hiện bất cứ công việc gì - sẽ được tự động thêm vào.

Ví dụ

```
class CPhanSo
{
    <cac thuoc tinh>
    ...
    ~ CPhanSo () //destructor
    {
    }
}
```

```
static void Main(string[] args)
{
    CPhanSo phanso = new CPhanSo();
    phanso = null;
}
```

Trong hàm main, ngay khi ta gán `phanso = null;` hàm destructor của con trỏ `phanso` đã được gọi tự động sau khi hệ thống giải phóng & thu hồi bộ nhớ mà ta không cần khai báo.

5. Định nghĩa toán tử

Toán tử là công cụ dùng để thao tác dữ liệu. Một toán tử là một ký hiệu dùng để đại diện cho một thao tác cụ thể nào đó được thực hiện trên dữ liệu.

Cũng như các phương thức đã xây dựng ở bài 03, chúng ta được phép định nghĩa các phép toán cho lớp đối tượng. Để có thể hiểu rõ hơn các phép toán trong lớp đối tượng, ta xét ví dụ định nghĩa toán tử + và - cho đối tượng `CPhanSo` đã xây dựng ở bài 03.

Ví dụ

Phương pháp không sử dụng toán tử:

Giả sử ta cần cộng 2 phân số. Ta định nghĩa hàm cộng như sau:

```
public CPhanSo tong(CPhanSo a)
{
    CPhanSo res = new CPhanSo();
}
```

Bài 2: XÂY DỰNG LỚP VÀ SỬ DỤNG ĐỐI TƯỢNG

```
res.TuSo = this.TuSo * a.MauSo + this.MauSo * a.TuSo;
res.MauSo = this.MauSo * a.MauSo;
return res;
}
```

Trong chương trình chính, ta gọi hàm này như sau:

```
static void Main(string[] args)
{
    CPhanSo a = new CPhanSo(2,1);
    CPhanSo b = new CPhanSo(2, 2);
    c = b.tong(a);
    c.Xuat();
}
```

Nhận xét: Để có thể thực hiện các phương thức cộng, trừ, nhân, chia,... ta phải viết :

CPhanSo tổng = số_thứ_nhất.cộng(số_thứ_hai)

Cách sử dụng như vậy mang nặng tính chất thể hiện phương thức cộng, chứ không phải là phép cộng. Do đó, nó không thể hiện đầy đủ về mặt ý nghĩa. Ta xét cách viết thứ 2 như sau

Phương pháp sử dụng toán tử

Giả sử lớp CPhanSo được định nghĩa 2 hàm cộng trừ như sau

```
public static CPhanSo cong(CPhanSo a, CPhanSo b)
{
    //Xây dựng hàm cộng 2 phân số, kết quả trả về
    // là 1 phân số mới là tổng của 2 phân số.
    CPhanSo result = new CPhanSo();
    result.TuSo = a.TuSo * b.MauSo + a.MauSo * b.TuSo;
    result.MauSo = a.MauSo * b.MauSo;
    return result;
}

public static CPhanSo tru(CPhanSo a, CPhanSo b)
{
    //Xây dựng hàm trừ 2 phân số, kết quả trả về
    // là 1 phân số mới là hiệu của 2 phân số
    CPhanSo result = new CPhanSo();
    result.TuSo = a.TuSo * b.MauSo - a.MauSo * b.TuSo;
    result.MauSo = a.MauSo * b.MauSo;
    return result;
}
```


Bài 2: XÂY DỰNG LỚP VÀ SỬ DỤNG ĐỐI TƯỢNG

```
}
```

Dựa vào 2 hàm cộng trừ đã được định nghĩa, ta xây dựng 2 phép toán + và – áp dụng cho lớp phân số như sau:

```
public static CPhanSo operator +(CPhanSo a, CPhanSo b)
{
    return cong(a,b);
}
public static CPhanSo operator -(CPhanSo a,CPhanSo b)
{
    return tru(a,b);
}
```

Trong chương trình chính, ta gọi như sau:

```
static void Main(string[] args)
{
    CPhanSo a = new CPhanSo(2,1);
    CPhanSo b = new CPhanSo(2, 2);
    CPhanSo c = new CPhanSo();
    c = a - b;
    c.Xuat();
}
```

Tổng quát:

```
CPhanSo tổng = số_thứ_nhất + số_thứ_hai;
```

Rõ ràng, đối với cách viết này, ta hiểu rõ hơn bản chất của phép toán, chứ không mang nặng ý nghĩa là phương thức.

6. Cách sử dụng

Trong C#, các toán tử là các phương thức tĩnh, kết quả trả về là giá trị biểu diễn kết quả của 1 phép toán ứng với các tham số là các toán hạng. Khi tạo toán tử cho 1 lớp đối tượng, ta phải nạp chồng toán tử. Cú pháp nạp chồng toán tử như sau

```
public static <tên_lớp_đối_tượng> operator <kiểu_operator>
(<các_tham_số_toán_hạng>)
```

7. Phân loại

C# cung cấp rất nhiều các loại toán tử, trong phạm vi bài học, chúng ta chỉ xét 1 số loại toán cơ bản

a) Toán tử số học

C# cung cấp 5 loại toán tử số học bao gồm:

- Cộng (+)
- Trừ (-)
- Nhân (*)
- Chia (/)
- Lấy dư (%)

Các phép toán này không thể thiếu trong bất kì ngôn ngữ lập trình nào. Với các kiểu dữ liệu số nguyên, số thực,.. các phép toán này được xây dựng sẵn mà chúng ta thường sử dụng trước đây. Tuy nhiên đối với các kiểu dữ liệu hoặc các lớp đối tượng tự định nghĩa, đôi khi chúng ta phải xây dựng lại các phép toán này cho phù hợp với kiểu dữ liệu tự định nghĩa đó.

Lưu ý: Khi xây dựng các phép toán số học này, các phép toán +=, -=, *=, /= tự động phát sinh, không cần xây dựng lại

b) Toán tử quan hệ

Toán tử quan hệ dùng để kiểm tra mối quan hệ giữa 2 biến, hoặc giữa 1 biến và 1 hằng. Các toán tử quan hệ điển hình:

- So sánh lớn hơn (>)
- So sánh nhỏ hơn (<)
- So sánh bằng (==)
- So sánh lớn hơn hoặc bằng (>=)
- So sánh nhỏ hơn hoặc bằng (<=)
- So sánh khác (!=)

Kết quả trả về thường mang giá trị bool. True hoặc False.

c) Toán tử tăng giảm

Toán tử ++ được gọi là toán tử tăng

Toán tử -- được gọi là toán tử giảm

Các loại toán tử tăng giảm**i. Tăng giảm tiền tố**

Cú pháp: <operator> tên_biến;

Ví dụ: ++a; --a;

ii. Tăng giảm hậu tố

Cú pháp: tên_biến<operator>;

Ví dụ: a++; a--;

Sự khác biệt

Xét 2 trường hợp sau

TH1:

```
CPhanSo a = new CPhanSo(5,1);
```

```
a++;
```

```
a.Xuat();
```

```
++a;
```

```
a.Xuat();
```

```
a--;
```

```
a.Xuat();
```

```
--a;
```

```
a.Xuat();
```

Kết quả

```
6/1
```

```
7/1
```

```
6/1
```

```
5/1
```

TH2:

```
CPhanSo a = new CPhanSo(5,1);
```

```
CPhanSo b = new CPhanSo();
```

```
b= a++;
```

```
a.Xuat();
```

```
b.Xuat();
```

```
b=++a;
```

```
a.Xuat();
```

```
b.Xuat();
```

```
b=a--;
```

```
a.Xuat();
```

```
b.Xuat();
```

```
b=--a;
```

```
a.Xuat();
```

```
b.Xuat();
```

Kết quả:

a: 6/1

b: 5/1

a: 7/1

b: 7/1

a: 6/1

b: 7/1

a: 5/1

b: 5/1

Trong trường hợp 1, không có sự khác biệt của tiền tố và hậu tố. Tuy nhiên ở trường hợp 2, sự khác biệt nằm ở biến b được gán sau hoặc trước khi thực hiện toán tử tăng giảm. Ta có thể rút ra kết luận như sau:

```
y = x++;
```

tương đương:

```
y = x;
```

```
x = x+1;
```

```
y = ++x;
```

tương đương:

```
x = x+1;
```

```
y=x;
```

Cần phân biệt được sự khác biệt này để sử dụng cho đúng.

Khi xây dựng toán tử dạng này (--,++) C# tự động phát sinh cả 2 dạng tiền tố và hậu tố.

d) Toán tử ép kiểu

Bài 2: XÂY DỰNG LỚP VÀ SỬ DỤNG ĐỐI TƯỢNG

Trong bất kì loại ngôn ngữ nào, khi chuyển đổi kiểu dữ liệu (ép kiểu) từ kiểu dữ liệu thấp lên cao luôn luôn thành công, nhưng từ cao xuống thấp đôi khi sẽ gây mất thông tin. Xét ví dụ sau:

TH1:

```
int so1 = 123456;  
long so2 = (long)so1;
```

TH2:

```
long so1 = 1234567890123;  
int so2 = (int)so1;
```

- Trong trường hợp 1, việc ép kiểu thành công và không bị mất mát dữ liệu, nhưng trong trường hợp 2, việc ép kiểu sẽ gây mất thông tin do kiểu dữ liệu int có kích thước thấp hơn kiểu dữ liệu long. Do đó khi ép kiểu sẽ gây sai sót. Do đó, khi chuyển kiểu dữ liệu từ cao xuống thấp, ta cần phải chuyển tường minh.

- Trong C#, để chuyển kiểu dữ liệu từ thấp lên cao, ta dùng từ khóa implicit. Để chuyển kiểu dữ liệu từ cao xuống thấp, ta dùng từ khóa explicit.

Xét ví dụ chuyển đổi kiểu dữ liệu Phân số thành int và ngược lại như sau

Xét lớp CPhanSo

```
class CPhanSo  
{  
    public CPhanSo(int a, int b)  
    {  
        this.TuSo = a;  
        this.MauSo = b;  
    }  
    public static implicit operator CPhanSo(int a)  
    {  
        return new CPhanSo(a, 1);  
    }  
    public static explicit operator int(CPhanSo a)  
    {  
        return (int)(a.TuSo / a.MauSo);  
    }  
}
```

e) Lợi ích của việc sử dụng toán tử

Bài 2: XÂY DỰNG LỚP VÀ SỬ DỤNG ĐỐI TƯỢNG

Việc sử dụng các toán tử giúp cho việc trình bày các đoạn mã dễ nhìn, dễ quản lí và rõ ràng hơn. Tuy nhiên, cần sử dụng đúng mục đích, thích hợp để tránh gây nhầm lẫn, khó hiểu.

II. NỘI DUNG THỰC HÀNH:

1/ Chương trình mẫu: (3đ)

File CPhanSo.cs

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

namespace Vidu
{
    class CPhanSo
    {
        #region Attributes
        private int iTuSo;
        private int iMauSo;
        #endregion
        #region Properties
        public int TuSo
        {
            get { return iTuSo; }
            set { iTuSo = value; }
        }
        public int MauSo
        {
            get { return iMauSo; }
            set { iMauSo = value; }
        }
        #endregion
        #region Methods
        //Construction
        public CPhanSo()
        {
        }
        //destruction
        ~CPhanSo()
        {
        }

        public CPhanSo(int a, int b)
        {

```

```
        this.TuSo = a;
        this.MauSo = b;
    }
    public static CPhanSo operator-- (CPhanSo b)
    {
        return new CPhanSo(b.TuSo-b.MauSo,b.MauSo);
    }
    public static CPhanSo cong(CPhanSo a,CPhanSo b)
    {
        //Xây dựng hàm cộng 2 phân số, kết quả trả về
        // là 1 phân số mới là tổng của 2 phân số.
        CPhanSo result = new CPhanSo();
        result.TuSo = a.TuSo * b.MauSo + a.MauSo * b.TuSo;
        result.MauSo = a.MauSo * b.MauSo;
        return result;
    }
    public static CPhanSo operator +(CPhanSo a, CPhanSo b)
    {
        return cong(a,b);
    }

    public void Xuat()
    {
        Console.Write(this.iTuSo.ToString() + " / " +
this.iMauSo.ToString());
    }
    #endregion
}
}
```

File Program.cs

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

namespace Vidu
```

```
{
    class Program
    {
        static void Main(string[] args)
        {
            CPhanSo a = new CPhanSo(5,1);
            CPhanSo b = new CPhanSo(7,2);
            CPhanSo c = a + b;
            c.Xuat();
            c = a--;
            c.Xuat();
            a.Xuat();
        }
    }
}
```

Yêu cầu:

- Build và chạy thử chương trình
- Đọc hiểu từng đoạn code.
- Phân tích dòng nào là thuộc tính, phương thức cập nhật dữ liệu, các hàm trong lớp đối tượng?
- Trong file Program.cs. Ta thay dòng lệnh `CPhanSo a = new CPhanSo(5,1);` thành `CPhanSo a;` có được không, vì sao?
- Có thể viết code cho operator `+` trực tiếp mà không cần gọi hàm `cong(a,b)` bên trong đó được không?
- Trong file Program.cs. Ta thay dòng lệnh `c = a--;` thành `c = --a;` Ghi lại kết quả thay đổi.

2/ Chỉnh sửa số liệu theo yêu cầu (5đ):

Từ chương trình mẫu trên, sinh viên viết thêm các yêu cầu sau:

- Tạo thêm hàm dựng (Constructor) với các tham số đầu vào là 1 biến có kiểu `CPhanSo`.
`public CPhanSo(CPhanSo a)`
- Trong file Program.cs. Sử dụng hàm dựng vừa tạo ở trên để tạo đối tượng mới với các tham số đầu vào tương ứng.

- Xây dựng hàm kiểm tra xem phân số đã tối giản hay chưa. Kết quả trả về True hoặc False.
- Xây dựng hàm tối giản phân số. Kết quả trả về là 1 phân số đã được tối giản.
- Xây dựng các hàm trừ nhân chia 2 phân số. Kết quả trả về có kiểu dữ liệu là 1 phân số mới
- Viết các phép toán trừ, nhân, chia 2 phân số.
- Viết lại toán tử giảm cho 1 phân số, giá trị giảm là 1/1.
- Viết toán tử tăng cho 1 phân số. Giá trị tăng là 1/1.
- Viết toán tử ép kiểu từ float lên kiểu CPhanSo và ngược lại, từ kiểu CPhanSo về kiểu float.
- Trong file Program.cs. Viết chương trình chính gọi các toán tử đã xây dựng, xuất ra kết quả trên màn hình console sau mỗi bước gọi phép toán.

3/ Bài tập tổng hợp (2đ)

Sinh viên xây dựng 1 lớp đối tượng kiểu dữ liệu là đa thức bậc 2 có dạng $ax^2 + bx + c=0$. Được mô tả như sau:

- Khai báo 3 thuộc tính a, b, c kiểu số nguyên có scope là private.
- Các phương thức cập nhật & trả dữ liệu trên thuộc tính
- Hàm dựng (Constructor) 1 đa thức với 3 tham số đầu vào kiểu int.
- Hàm in đa thức theo dạng $ax^2 + bx + c = 0$.
- Hàm kiểm tra đa thức có nghiệm không, kết quả trả về số nghiệm của đa thức. Nếu đa thức không có nghiệm, kết quả trả về -1.
- Hàm tính delta = căn ($b^2 - 4ac$). Kết quả trả về là 1 số thực.
- Hàm in các nghiệm của đa thức. Nếu đa thức không có nghiệm, in ra câu “Đa thức không có nghiệm”.
- Viết các toán tử cộng, trừ 2 đa thức
- Viết toán tử so sánh bằng giữa 2 đa thức. Quy tắc so sánh bằng như sau:
 - o 2 Đa thức được gọi là bằng nhau nếu các hệ số a, b, c của chúng bằng nhau từng đôi một $a1=a2 \ \&\& \ b1=b2 \ \&\& \ c1=c2$
- Viết toán tử so sánh khác giữa 2 đa thức. Quy tắc so sánh khác như sau
 - o 2 đa thức được gọi là khác nhau nếu chúng xuất hiện ít nhất 1 cặp hệ số khác nhau $a1 \neq a2 \ || \ b1 \neq b2 \ || \ c1 \neq c2$

- Xây dựng 1 lớp CDaThuc2 là 1 đa thức bậc 1 $bx+c$ có các thuộc tính và phương thức cập nhật & trả dữ liệu tương tự như lớp CDaThuc. Lưu file dưới dạng CDaThuc2.cs
- Giả sử lớp CDaThuc có dạng $a_1x^2 + b_1x + c_1 = 0$, lớp CDaThuc2 có dạng $b_2x + c_2 = 0$.
 - o Viết toán tử ép kiểu CDaThuc thành CDaThuc2 với quy tắc sau:
 - $b_2 = b_1$
 - $c_2 = c_1$
 - o Viết toán tử ép kiểu CDaThuc2 thành CDaThuc với quy tắc sau:
 - $a_1 = 0$
 - $b_1 = b_2$
 - $c_1 = c_2$
- Trong file Program.cs. Xây dựng chương trình minh họa đầy đủ phép toán đã xây dựng, xuất ra kết quả trên màn hình console sau mỗi bước gọi phép toán.

4/ Bài tập làm thêm:

Tạo lớp CComplex theo yêu cầu như sau:

- Mỗi đối tượng của lớp lưu trữ một số phức. Bao gồm: Phần thực và phần ảo là những số thực kiểu double.
- Hàm tạo không tham số đầu vào và hàm tạo hai tham số đầu vào kiểu số nguyên.
- Phương thức cập nhật: đặt lại giá trị của phần thực, phần ảo, số phức, và trả về phần thực, phần ảo.
- Phương thức Norm trả về modul của số phức.
- Phương thức Print dùng để xuất số phức ra một chuỗi.
- Toán tử cộng, trừ, nhân, chia 2 số phức
- Toán tử so sánh bằng, so sánh khác 2 số phức
- Toán tử ép kiểu từ kiểu int thành kiểu số phức và ngược lại
- Toán tử ép kiểu từ kiểu float thành kiểu số phức và ngược lại
- Toán tử tăng giảm 1 số phức, quy tắc tăng giảm: chỉ tăng giảm phần thực 1 đơn vị

Bài 3 - 5: XÂY DỰNG ỨNG DỤNG QUẢN LÝ DANH MỤC DỮ LIỆU

I. MỤC TIÊU

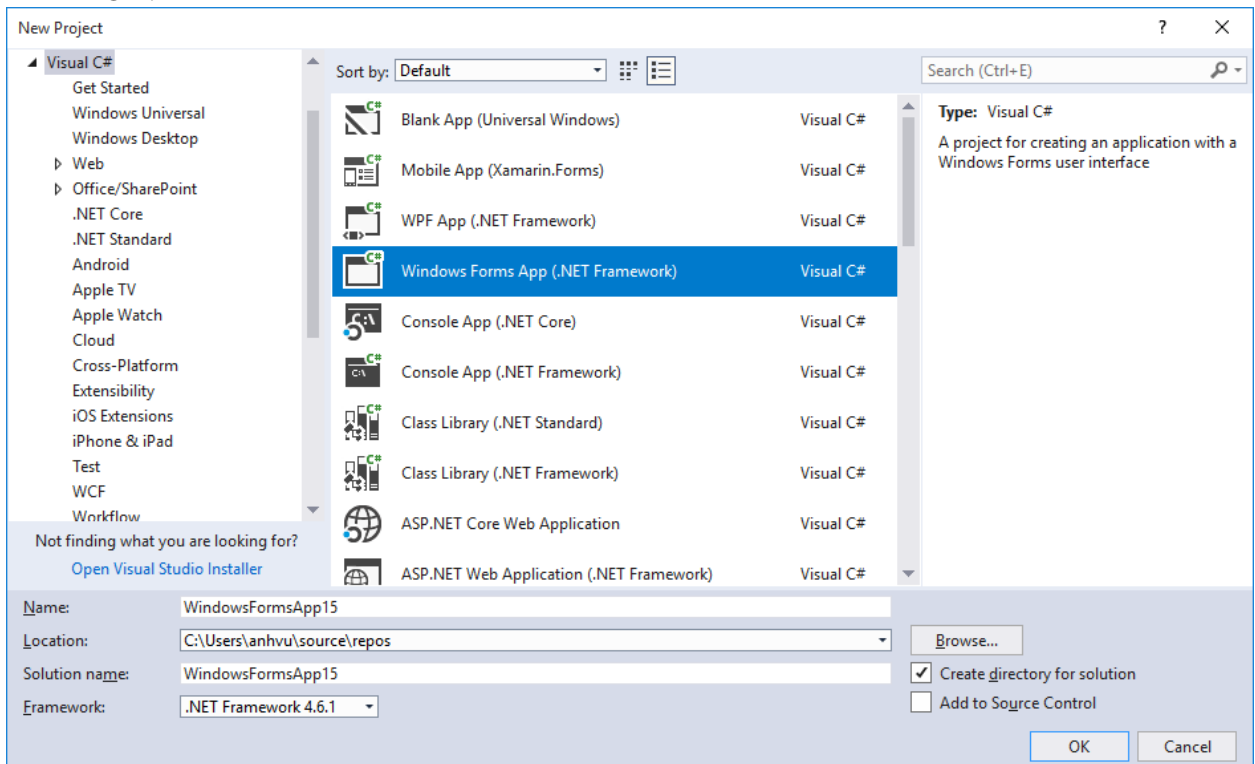
Hoàn tất bài thực hành này, sinh viên có thể :

- Xây dựng lớp đối tượng, tạo đối tượng và sử dụng đối tượng ở mức thành thạo.
- Sử dụng tập đối tượng thông qua các lớp đối tượng đã có trong C#: List<T>, Dictionary<TKey,TValue>, ArrayList, Hashtable, ...
- Thao tác trên File nhị phân thông qua các lớp có trong C#: File, FileStream, BinaryFormatter.
- Nắm bắt lỗi trong chương trình.
- Lập trình WinForm cơ bản.
- Xây dựng ứng dụng theo mẫu 1-Tiers, 2-Tiers.

II. TÓM TẮT LÝ THUYẾT

1. Tạo ứng dụng windows

- a. Khởi động Microsoft Visual Studio .Net => File => New Project => chọn ngôn ngữ Visual C# => chọn Windows Form App (.NET Framework) => nhập tên ứng dụng => OK.



- b. Thực thi ứng dụng : menu Debug => Start Debug (hoặc nhấn F5).
- c. Các thuộc tính (Property) thường dùng :
- Text : chuỗi hiển thị trên control.

- Enabled (True/False) : cho phép/không cho phép control hoạt động.
- Visible (True/False) : cho phép/không cho phép control hiển thị.
- Font : chọn Font cho control.
- Location : tọa độ của control trên Form.
- d. Các sự kiện (Event) thường dùng :
 - Click : xảy ra khi click vào control.
 - DoubleClick : xảy ra khi nhấp đôi chuột vào đối tượng.
 - TextChanged : xảy ra khi thay đổi chuỗi hiển thị trên TextBox.
- 2. Tìm hiểu các điều khiển :** SV xem thêm thông tin về các điều khiển này bằng MSDN.
- e. Label Control :
 - Mục đích: Dùng để xuất dữ liệu.
 - Một đối tượng tương ứng với Label Control thuộc lớp Label.
 - Lớp Label có Property thường dùng: Text.
- f. TextBox Control :
 - Mục đích: Dùng để nhập/xuất dữ liệu.
 - Một đối tượng tương ứng với TextBox Control thuộc lớp TextBox.
 - Lớp TextBox có Property thường dùng: Text.
- g. Button Control :
 - Mục đích: Dùng để thực hiện một công việc tính toán nào đó.
 - Một đối tượng tương ứng với Button Control thuộc lớp Button.
 - Lớp Button có Property thường dùng: Text.
- h. ComboBox Control :
 - Mục đích: là sự kết hợp của TextBox, ListBox và Button. Phần lớn các thành phần cơ bản của TextBox, ListBox và Button đều sử dụng được cho ComboBox.
 - Một đối tượng tương ứng với ComboBox Control thuộc lớp ComboBox.
 - ComboBox có ba kiểu (Property DropDownStyle): Simple, DropDown và DropDownList.
- i. CheckBox Control:
 - Mục đích: dùng để lựa chọn hay không lựa chọn một dữ liệu đã liệt kê.
 - Một đối tượng tương ứng với CheckBox Control thuộc lớp CheckBox.
 - Lớp CheckBox có Property thường dùng: Text và Checked.
- j. RadioButton Control :
 - Mục đích: dùng để lựa chọn một trong các mục chọn dữ liệu đã liệt kê.
 - Một đối tượng tương ứng với RadioButton Control thuộc lớp RadioButton.
 - Lớp TextBox có Property thường dùng: Text và Checked.
- k. GroupBox Control :
 - Mục đích: dùng để phân vùng hiển thị trên Form.
 - Một đối tượng tương ứng với GroupBox Control thuộc lớp GroupBox.
 - Lớp GroupBox có Property thường dùng: Text.
- l. DateTimePicker Control :
 - Mục đích: dùng để lựa chọn ngày và giờ.
 - Một đối tượng tương ứng với Date Time Picker Control thuộc lớp DateTimePicker.
 - Lớp DateTimePicker có Property thường dùng: **Value**. Một giá trị tương ứng với Date Time Picker Control thuộc kiểu DateTime.

- Khi ta trừ hai dữ liệu thuộc kiểu ngày (**DateTime**), kết quả của phép toán là một biến thuộc kiểu **TimeSpan**.
- Ta có thể cộng (trừ) một dữ liệu thuộc kiểu ngày với một số nguyên (thông qua việc cộng một biến thuộc kiểu DateTime với một biến thuộc kiểu TimeSpan), kết quả là một dữ liệu thuộc kiểu ngày (một biến thuộc kiểu DateTime).

3. DataGridView Control: dùng để nhập/xuất dữ liệu dạng bảng (gồm nhiều dòng và nhiều cột)

- Thuộc tính Columns: tập hợp các tiêu đề của DataGridView. Có thể thêm/xóa các tiêu đề. Các thuộc tính cần quan tâm: Name, HeaderText, DataPropertyName, ColumnType.
- Thuộc tính DataSource: nguồn dữ liệu cần hiển thị.
- Thuộc tính AutoGenerateColumns: tự động lấy các cột nếu bằng true, ngược lại lấy đúng cột đã khai báo.
- Thuộc tính AllowUserToAddRows: cho (true)/không cho (false) thêm dòng trực tiếp trên lưới.
- Thuộc tính AllowUserToDeleteRows: cho (true)/không cho (false) xóa dòng trực tiếp trên lưới.

4. Lớp List<T> : là một kiểu danh sách các đối tượng có thể được truy cập bởi chỉ số.

- Property Count : trả về số lượng phần tử trong List.
- Property Item : trả về hoặc thiết lập phần tử được xác định bởi chỉ số.
- Phương thức Add : thêm một phần tử vào List.
- Phương thức Clear : xóa tất cả các phần tử trong List.
- Phương thức Contains : xác định xem một phần tử có trong List không.
- Phương thức Insert : chèn thêm một phần tử vào List.
- Phương thức Remove : xóa phần tử ra khỏi List.
- Phương thức RemoveAt : xóa phần tử ra khỏi List theo chỉ số.
- Phương thức Sort : sắp xếp các phần tử trong List.
- Phương thức ToArray : chuyển List<T> thành một mảng.
- Ví dụ :

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

public class Example
{
    public static void Demo(System.Windows.Controls.TextBlock outputBlock)
    {
        List<string> dinosaurs = new List<string>();

        outputBlock.Text += String.Format("\nCapacity: {0}",
            dinosaurs.Capacity) + "\n";

        dinosaurs.Add("Tyrannosaurus");
        dinosaurs.Add("Amargasaurus");
        dinosaurs.Add("Mamenchisaurus");
        dinosaurs.Add("Deinonychus");
        dinosaurs.Add("Compsognathus");
    }
}
```

```
outputBlock.Text += "\n";
foreach (string dinosaur in dinosaurs)
{
    outputBlock.Text += dinosaur + "\n";
}

outputBlock.Text += String.Format("\nCapacity: {0}",
dinosaurs.Capacity) + "\n";
outputBlock.Text += String.Format("Count: {0}", dinosaurs.Count) +
"\n";

outputBlock.Text += String.Format("\nContains(\"Deinonychus\"): {0}",
dinosaurs.Contains("Deinonychus")) + "\n";
}
}
/* Đây là kết quả của đoạn chương trình trên:

Capacity: 0

Tyrannosaurus
Amargasaurus
Mamenchisaurus
Deinonychus
Compsognathus

Capacity: 8
Count: 5

Contains("Deinonychus"): True
*/
```

SV có thể xem tham khảo thêm trong MSDN.

5. Lớp Dictionary : là loại kiểu dữ liệu tập hợp lưu trữ các cặp khóa/giá trị (key/value) . Thường được sử dụng để tra cứu thông qua việc ánh xạ khóa (key) sang giá trị (value) cần tìm. Các khóa (key) không được trùng nhau. (SV có thể xem thêm thông tin về lớp Dictionary trong MSDN bằng từ khóa “*Dictionary(Of TKey, TValue) class*”).

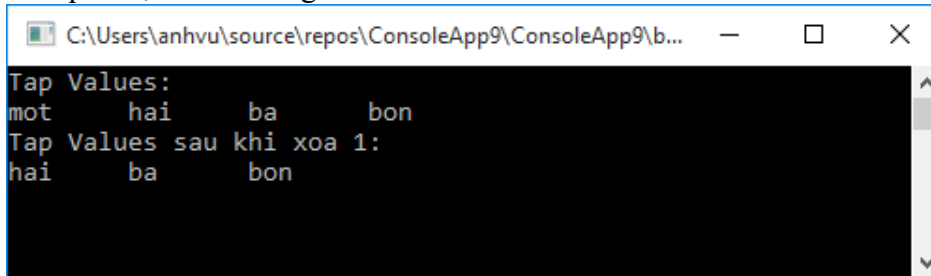
- Count : lấy tổng số phần tử trong Dictionary.
- Item : lấy/thiết lập phần tử trong Dictionary bằng chỉ số.
- Add : thêm phần tử vào Dictionary.
- Remove : xóa phần tử trong Dictionary.
- Clear : xóa tất cả phần tử.
- Keys: tập các khóa trong Dictionary.
- Values: tập các giá trị trong Dictionary.

Ví dụ :

```
static void Main(string[] args)
{
    Dictionary<int, string> dic1 = new Dictionary<int, string>();
    dic1.Add(1, "mot");
    dic1.Add(2, "hai");
    dic1.Add(3, "ba");
    dic1.Add(4, "bon");
    string skQ = "";
    Console.WriteLine("Tap Values:");
```

```
        foreach (string s in dic1.Values)
            SKQ += s + "\t";
        Console.WriteLine(SKQ);
        dic1.Remove(1);
        SKQ = "";
        Console.WriteLine("Tap Values sau khi xoa 1:");
        foreach (string s in dic1.Values)
            SKQ += s + "\t";
        Console.WriteLine(SKQ);
        Console.ReadLine();
    }
```

Kết quả thực thi chương trình :



```
Tap Values:
mot    hai    ba    bon
Tap Values sau khi xoa 1:
hai    ba    bon
```

6. Lớp ArrayList : (SV có thể xem thêm thông tin về lớp ArrayList trong MSDN bằng từ khóa “ArrayList class”).

- Count : lấy tổng số phần tử trong ArrayList.
- Item : lấy/thiết lập phần tử trong ArrayList bằng chỉ số.
- Reverse : đảo ngược thứ tự các phần tử trong ArrayList.
- Sort : sắp xếp các phần tử trong ArrayList.
- Add : thêm phần tử vào ArrayList.
- RemoveAt : xóa phần tử theo chỉ số.
- Clear : xóa tất cả phần tử.

Ví dụ :

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Collections;

namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        public Form1 ()
        {
            InitializeComponent();
        }
    }
}
```

```
public static void PrintValues(TextBox txt, IEnumerable
myList, char mySeparator)
{
    foreach (Object obj in myList)
        txt.Text += mySeparator + obj.ToString();
    txt.Text += "\r\n";
}

private void Form1_Load(object sender, EventArgs e)
{
    // khởi tạo đối tượng ArrayList.
    ArrayList myAL = new ArrayList();
    myAL.Add("The");
    myAL.Add("quick");
    myAL.Add("brown");
    myAL.Add("fox");

    // khởi tạo đối tượng Queue.
    Queue myQueue = new Queue();
    myQueue.Enqueue("jumped");
    myQueue.Enqueue("over");
    myQueue.Enqueue("the");
    myQueue.Enqueue("lazy");
    myQueue.Enqueue("dog");

    // hiển thị ArrayList và Queue.
    txtXuat.Text="ArrayList chứa các giá trị :\r\n";
    PrintValues(txtXuat, myAL, '\t');
    txtXuat.Text += "Queue chứa các giá trị :\r\n";
    PrintValues(txtXuat, myQueue, '\t');

    // chép các phần tử từ Queue sang ArrayList.
    myAL.AddRange(myQueue);

    // hiển thị ArrayList.
    txtXuat.Text += " ArrayList chứa các giá trị :\r\n";
    PrintValues(txtXuat, myAL, '\t');
}
}

/*
Kết quả xuất ra TextBox :
ArrayList chứa các giá trị :
    The    quick    brown    fox
Queue chứa các giá trị :
    jumped    over    the    lazy    dog
ArrayList chứa các giá trị :
    The    quick    brown    fox    jumped    over    the    lazy
dog
*/
```


7. Lớp HashTable : đây là lớp cấu trúc dữ liệu bảng băm (hashtable) được xây dựng sẵn. (SV có thể xem thêm thông tin bằng từ khóa “*Hashtable class*”).

- Phương thức Add : thêm 1 phần tử vào Hashtable.
- Phương thức Remove : xóa 1 phần tử ra khỏi Hashtable.
- Phương thức Clear : xóa tất cả phần tử trong Hashtable.
- Thuộc tính Count : trả về số phần tử có trong Hashtable.
- Thuộc tính Item : truy xuất phần tử có khóa (key) xác định.
- Thuộc tính Value : lấy một ICollection chứa giá trị trong Hashtable.

Ví dụ :

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Collections;

namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        public static void PrintKeysAndValues(TextBox txt, Hashtable
myHT)
        {
            txt.Text+="\t-KEY-\t-VALUE-\r\n";
            foreach (DictionaryEntry de in myHT)
            {
                txt.Text += "\t" + de.Key + ":\t" + de.Value;
                txt.Text += "\r\n";
            }
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            // khởi tạo Hashtable.
            Hashtable myHT = new Hashtable();
            myHT.Add("one", "The");
            myHT.Add("two", "quick");
            myHT.Add("three", "brown");
            myHT.Add("four", "fox");

            // hiển thị Hashtable.
            txtXuat.Text="Hashtable chứa các giá trị :\r\n";
            PrintKeysAndValues(txtXuat, myHT);
        }
    }
}
```

```

    }
}
/*
Kết quả xuất ra TextBox :
Hashtable chứa các giá trị :
    -KEY-    -VALUE-
    one:     The
    three:   brown
    four:    fox
    two:     quick

```

8. Nắm bắt lỗi (Exception) : Chúng ta phát hiện và xử lý lỗi với cấu trúc :
try ... catch ... finally

- Khối try : chứa các lệnh có khả năng gây ra lỗi.
- Khối catch : chứa các dòng lệnh để bắt và xử lý lỗi phát sinh trên khối try. Khối này gồm một loạt các lệnh bắt đầu với từ khóa catch, biến kiểu Exception ứng với một kiểu Exception muốn bắt và các lệnh xử lý. Dĩ nhiên, ta có thể dùng một lệnh catch cho tất cả các Exception.
- Khối finally : là khối tùy chọn, sau khi chạy qua các khối try và catch nếu không có chỉ định nào khác, khối finally sẽ được thực hiện bất kể lỗi có xảy ra hay không.

Cú pháp chung cho cấu trúc xử lý lỗi như sau:

```

try
{
    //Khối lệnh có thể phát sinh lỗi
}
catch (khai báo ngoại lệ 1)
{ các câu lệnh xử lý ngoại lệ 1 }
...
catch (khai báo ngoại lệ n)
{ các câu lệnh xử lý ngoại lệ n }
finally
{ các câu lệnh xử lý khối finally }

```

Ví dụ:

```

double d = 0; int i;
try
{
    Console.WriteLine("Hay nhap mot so nguyen:");
    i = int.Parse(Console.ReadLine());
    d = 42 / i;
    Console.WriteLine("d = {0}", d);
}
catch (DivideByZeroException ex)
{
    Console.WriteLine("Khong the chia cho so khong.");
}

```

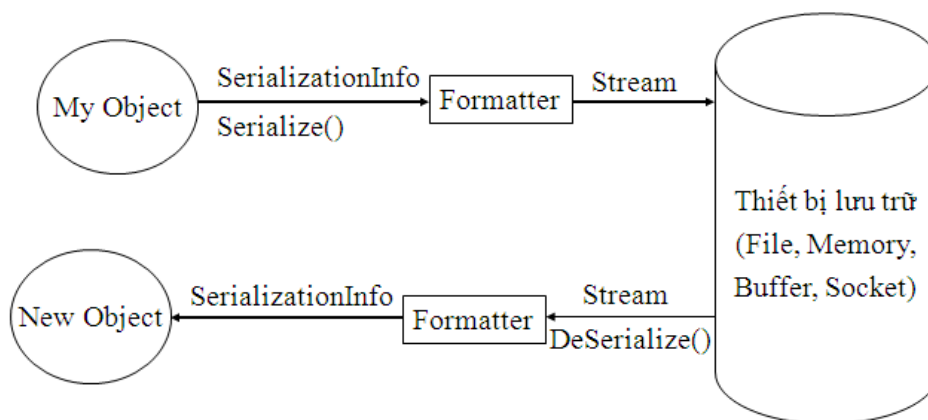
```
}  
catch(FormatException ex)  
{  
    Console.WriteLine("Xin nhap so nguyen.");  
}
```

Bẫy lỗi với kiểu Exception chung:

```
double d = 0; int i;  
try  
{  
    Console.WriteLine("Hay nhap mot so nguyen:");  
    i = int.Parse(Console.ReadLine());  
    d = 42 / i;  
    Console.WriteLine("d = {0}", d);  
}  
catch (Exception ex)  
{  
    Console.WriteLine("Khong the chia.");  
}
```

9. Đọc/Ghi danh sách lên file nhị phân (sử dụng kỹ thuật Serialize) :

- Serialize có nghĩa là sắp theo thứ tự. Khi ta muốn lưu một đối tượng xuống tập tin trên đĩa từ để lưu trữ, ta phải định ra trình tự lưu trữ của dữ liệu trong đối tượng. Khi cần tái tạo lại đối tượng từ thông tin trên tập tin đã lưu trữ, ta sẽ nạp đúng theo trình tự đã định trước đó. Đây gọi là quá trình Serialize.
- Các đối tượng cơ sở đều có khả năng Serialize. Để đối tượng của ta có thể Serialize, trước tiên cần khai báo attribute [Serialize] cho lớp đối tượng đó. Nếu đối tượng có chứa các đối tượng khác thì đối tượng đó phải có khả năng Serialize.
- Namespace System.Runtime.Serialization.Formatters.Binary : xử lý file nhị phân bằng kỹ thuật Serialize (SV xem thêm thông tin trong MSDN). Ta có khái quát tiếng trình Serialize như sau :



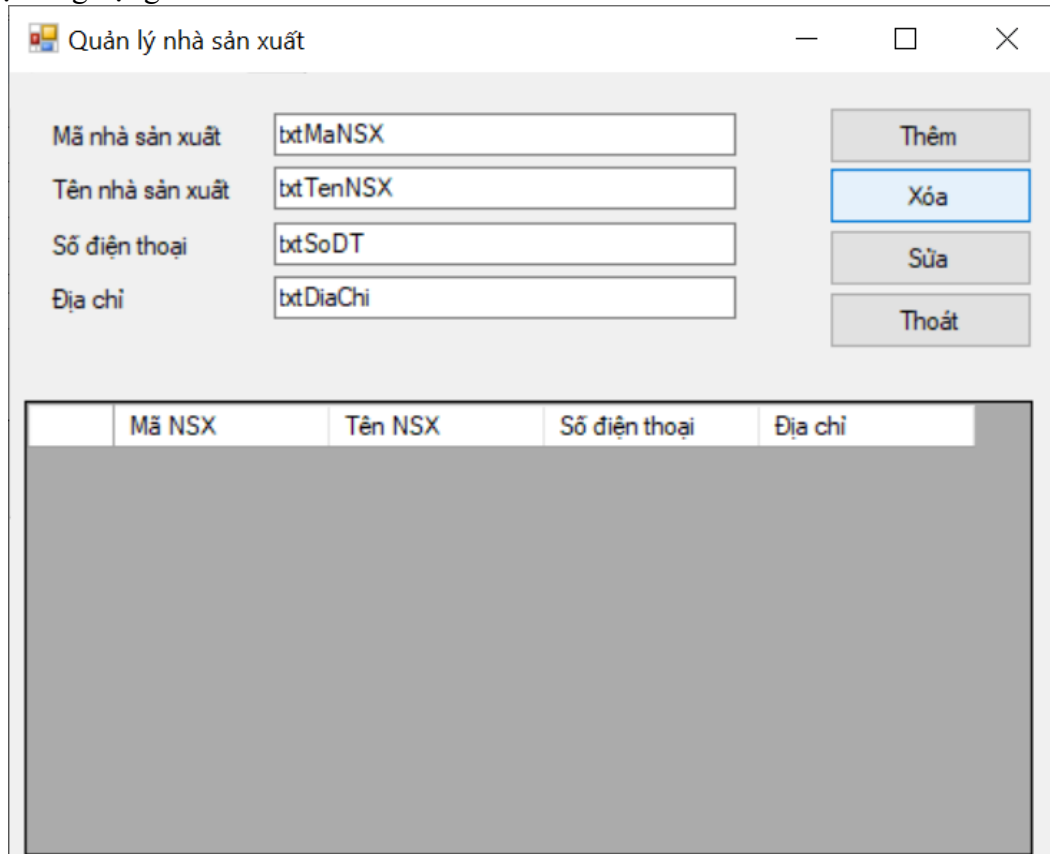
THÀNH VIÊN	Ý NGHĨA
BinaryFormatter	

Deserialize()	Thôi nối tiếp hóa một luồng byte thành một biểu đồ đối tượng.
Serialize()	Nối tiếp hóa một đối tượng hoặc biểu đồ các đối tượng đã nối thành một luồng.

III. NỘI DUNG THỰC HÀNH BÀI 3

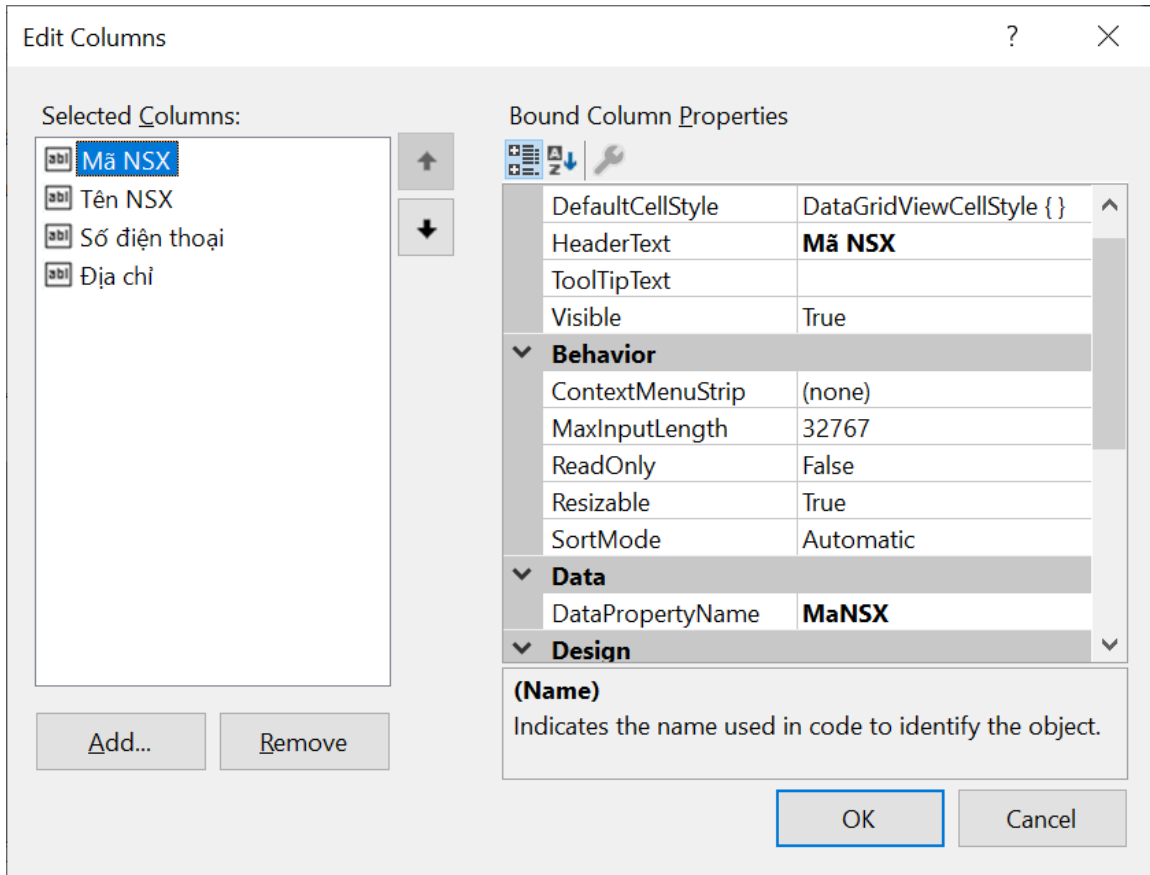
1. Chương trình mẫu (4đ)

a. Tạo ứng dụng sau :



Trong ứng dụng có :

- Bốn TextBox dùng để nhập xuất dữ liệu. Lần lượt đặt tên (thuộc tính Name) là **txtMaNSX**, **txtTenNSX**, **txtSoDT**, **txtDiaChi**.
- Bốn Button dùng để thực hiện các chức năng thêm, xóa, sửa, thoát. Lần lượt đặt tên là **btnThem**, **btnXoa**, **btnSua**, **btnThoat**.
- Một DataGridView dùng để hiển thị danh sách nhà sản xuất đặt tên là **dgvNSX**.
Hướng dẫn thêm các cột vào DataGridView:
 - Chọn thuộc tính Columns. Hiển thị hộp thoại:



- Lần lượt nhấn Add để thêm 4 cột:
 - Cột 1: HeaderText: Mã NSX, DataPropertyName: MaNSX. (Lưu ý: DataPropertyName là thuộc tính chỉ ra tên thuộc tính sẽ liên kết để hiển thị dữ liệu. Trong trường hợp này, cột “Mã NSX” liên kết với thuộc tính MaNSX sẽ được viết trong lớp CNhaSanXuat).
 - Cột 2: HeaderText: Tên NSX, DataPropertyName: TenNSX.
 - Cột 3: HeaderText: Số điện thoại, DataPropertyName: SoDT.
 - Cột 4: HeaderText: Địa chỉ, DataPropertyName: DiaChi.

- b. Viết lớp CNhaSanXuat, mỗi đối tượng thuộc lớp CNhaSanXuat sẽ lưu trữ thông tin của nhà sản xuất gồm mã nhà sản xuất, tên nhà sản xuất, số điện thoại, địa chỉ. Lớp CNhaSanXuat được viết code như sau:

```
public class CNhaSanXuat
{
    private string m_mansx, m_tennsx, m_sodt, m_diachi;
    public string MaNSX
    {
        get { return m_mansx; }
        set { m_mansx = value; }
    }
    public string TenNSX
    {
        get { return m_tennsx; }
```

```
        set { m_tennsx = value; }
    }
    public string SoDT
    {
        get { return m_sodt; }
        set { m_sodt = value; }
    }
    public string DiaChi
    {
        get { return m_diachi; }
        set { m_diachi = value; }
    }
    public CNhaSanXuat()
    {
        m_mansx = "";
        m_tennsx = "";
        m_sodt = "";
        m_diachi = "";
    }
    public CNhaSanXuat(string ma, string ten, string sodt, string dc)
    {
        m_mansx = ma;
        m_tennsx = ten;
        m_sodt = sodt;
        m_diachi = dc;
    }
}
```

- c. Viết code cho Form, ta sử dụng đối tượng thuộc lớp Dictionary để lưu danh sách nhà sản xuất

```
public partial class frmNSX : Form
{
    private Dictionary<string, CNhaSanXuat> dsNSX;
    private void hienDSNhaSanXuat()
    {
        dgvNSX.DataSource = dsNSX.Values.ToList();
    }
    private CNhaSanXuat timNSX(string ma)
    {
        try
        {
            return dsNSX[ma];
        }
        catch
        {
            return null;
        }
    }
    public frmNSX()
    {
        InitializeComponent();
    }
}
```

```
{
    InitializeComponent();
}

private void frmNSX_Load(object sender, EventArgs e)
{
    dsNSX = new Dictionary<string, CNhaSanXuat>();
}

private void btnThem_Click(object sender, EventArgs e)
{
    CNhaSanXuat n = new CNhaSanXuat();
    n.MaNSX = txtMaNSX.Text;
    n.TenNSX = txtTenNSX.Text;
    n.SoDT = txtSoDT.Text;
    n.DiaChi = txtDiaChi.Text;
    if (timNSX(n.MaNSX) == null)
    {
        dsNSX.Add(n.MaNSX, n);
        hienDSNhaSanXuat();
    }
    else
        MessageBox.Show("Mã nhà sản xuất " + n.MaNSX + " đã tồn tại.  
Không thêm được.");
}

private void dgvNSX_RowEnter(object sender, DataGridViewCellEventArgs e)
{
    txtMaNSX.Text =
dgvNSX.Rows[e.RowIndex].Cells[0].Value.ToString();
    txtTenNSX.Text =
dgvNSX.Rows[e.RowIndex].Cells[1].Value.ToString();
    txtSoDT.Text = dgvNSX.Rows[e.RowIndex].Cells[2].Value.ToString();
    txtDiaChi.Text =
dgvNSX.Rows[e.RowIndex].Cells[3].Value.ToString();
}

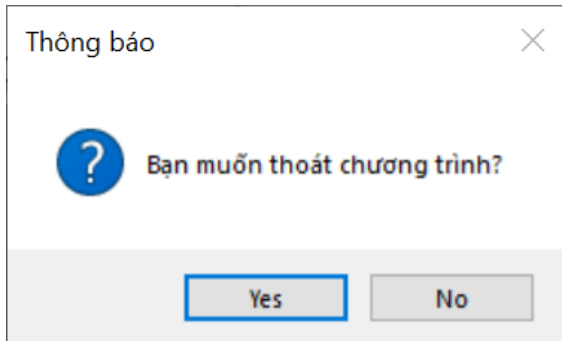
private void btnThoat_Click(object sender, EventArgs e)
{
    Close();
}
}
```

2. Vận dụng: (3đ)

SV hãy viết code theo yêu cầu sau:

- Nút Xóa: dùng để xóa nhà sản xuất đang chọn ra khỏi danh sách nhà sản xuất.
- Nút Sửa: dùng để sửa thông tin nhà sản xuất đang chọn.

- Khi đóng Form sẽ xuất hiện hộp thông báo như sau:



Nếu nhấn Yes sẽ thoát chương trình, nếu nhấn No sẽ không thoát chương trình.

3. Bài tập tổng hợp: (3đ)

- 1) Trong Form Quản lý Nhà sản xuất đã thực hiện ở phần trên, SV hãy đổi đối tượng thuộc lớp Dictionary<TKeys, TValues> thành đối tượng thuộc lớp List<T> để lưu danh sách nhà sản xuất.
- 2) Thiết kế ứng dụng như sau:

- Viết lớp CVatTu, mỗi đối tượng thuộc lớp CVatTu dùng để lưu thông tin của vật tư gồm mã vật tư, tên vật tư, đơn vị tính. SV hãy viết các thuộc tính và phương thức khởi tạo của lớp CVatTu.
- SV hãy viết code cho form :

- Sử dụng đối tượng thuộc lớp Dictionary để lưu danh sách vật tư.
 - Viết code cho các chức năng thêm, xóa, sửa, hiển thị danh sách vật tư.
- 3) Thực hiện lại form Quản lý vật tư nhưng sử dụng đối tượng thuộc lớp List<T> để lưu danh sách vật tư.

IV. NỘI DUNG THỰC HÀNH BÀI 4

1. Chương trình mẫu (2đ)

a. Tạo ứng dụng sau :

	Mã số học sinh	Họ tên	Ngày sinh	Phái	Địa chỉ
	hs0001	Nguyễn Văn A	10/02/2011	True	180 Cao Lỗ
▶	hs0002	Trần Thị B	02/04/2011	False	180 Cao Lỗ

Trong ứng dụng có :

- Ba TextBox dùng để nhập xuất dữ liệu.
- Một DateTimePicker dùng để nhập xuất ngày sinh.
- Hai RadioButton để chọn phái.
- Bốn Button Thêm, Xóa, Sửa, Lưu File.
- Viết lớp CHocSinh gồm các thông tin : mã số học sinh, họ tên, ngày sinh, phái, địa chỉ. Trong lớp có :
 - Các phương thức khởi tạo.
 - Các thuộc tính (property) thiết lập hoặc trả về dữ liệu : mã số học sinh, họ tên, ngày sinh, phái, địa chỉ.

```
public class CHocSinh
{
    private string m_mshs;
    private string m_hoten;
    private DateTime m_ngaysinh;
    private bool m_phai;
    private string m_diachi;
    1 reference
    public string MSHS...
    1 reference
    public string HoTen...
    1 reference
    public DateTime NgaySinh...
    1 reference
    public bool Phai...
    1 reference
    public string DiaChi...
    1 reference
    public CHocSinh()...
    0 references
    public CHocSinh(string ma,string ht,DateTime ns,bool p,string dc)...
```

- Sử dụng đối tượng thuộc lớp List<T> để lưu trữ danh sách các học sinh thuộc lớp CHocSinh. Trong đó :
 - Nút “Thêm” : lưu dữ liệu của học sinh đang nhập vào đối tượng lớp List<T>.
 - DataGridView hiển thị danh sách học sinh. Khi chọn mẫu tin trong DataGridView, mẫu tin được chọn sẽ hiển thị lên các control TextBox, DateTimePicker trên form.

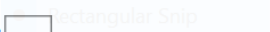
2. Thay đổi số liệu/yêu cầu: (2đ)

- Trong DataGridView, cột Phái hiển thị Nam/Nữ thay cho True/False.
- Khi thêm học sinh: nếu mã số học sinh đã tồn tại trong danh sách học sinh thì hiển thị thông báo “Mã số học sinh đã tồn tại trong danh sách học sinh. Không thêm được.”. Ngược lại, cho phép thêm thông tin học sinh mới vào danh sách học sinh.


3. Vận dụng: thực hiện thêm các chức năng: (2đ)

- Nút “Xóa”: xóa học sinh đang chọn khỏi danh sách học sinh.
- Nút “Sửa”: sửa thông tin học sinh đang chọn trong danh sách học sinh.
- Nút “Lưu File”: lưu danh sách học sinh vào file nhị phân.
- Sự kiện Load của Form: đọc thông tin danh sách học sinh từ file nhị phân và hiển thị danh sách học sinh lên DataGridView.


```
public partial class frmHocSinh : Form
{
    private List<CHocSinh> dsHS;
    1 reference
    private void hienDSHocSinh()...
    1 reference
    public frmHocSinh()...
    1 reference
    private void frmHocSinh_Load(object sender, EventArgs e)...
```




```
    1 reference
    private void btnThem_Click(object sender, EventArgs e)...
```




```
    1 reference
    private void dgvHocSinh_RowEnter(object sender, DataGridViewCellEventArgs e)...
```




```
    1 reference
    private void btnXoa_Click(object sender, EventArgs e)...
```



```
    1 reference
    private void btnSua_Click(object sender, EventArgs e)...
```



```
    1 reference
    private void btnLuuFile_Click(object sender, EventArgs e)...
```




```
}
```

Thiết kế của Form frmHocSinh


4. Bài tập tổng hợp (4đ)

Xây dựng ứng dụng quản lý học sinh (như các yêu cầu trong phần trên) nhưng sử dụng lớp Dictionary<Tkey,Tvalue> để lưu trữ danh sách học sinh và lớp frmHocSinh được thiết kế như sau:


```
public partial class frmHocSinh : Form
{
    private Dictionary<string,CHocSinh> dsHS;
    1 reference
    private void hienDSHocSinh()...
```




```
    1 reference
    public frmHocSinh()...
```




```
    1 reference
    private void frmHocSinh_Load(object sender, EventArgs e)...
```




```
    1 reference
    private void btnThem_Click(object sender, EventArgs e)...
```




```
    1 reference
    private void dgvHocSinh_RowEnter(object sender, DataGridViewCellEventArgs e)...
```




```
    1 reference
    private void btnXoa_Click(object sender, EventArgs e)...
```



```
    1 reference
    private void btnSua_Click(object sender, EventArgs e)...
```



```
    1 reference
    private void btnLuuFile_Click(object sender, EventArgs e)...
```



```
}
```

V. NỘI DUNG THỰC HÀNH BÀI 5

1. Chương trình mẫu (4đ)

Bài 3 - 5: XÂY DỰNG ỨNG DỤNG QUẢN LÝ DANH MỤC DỮ LIỆU

- a) Xây dựng ứng dụng quản lý thuê phòng khách sạn (Mỗi phiếu thuê phòng chỉ có một phòng thuê). Thông tin phiếu thuê phòng bao gồm: Mã số phiếu thuê, ngày bắt đầu thuê, ngày kết thúc thuê, họ tên khách hàng và loại phòng thuê (có 4 loại phòng là: A, B, C, D).

- Số ngày thuê được tính theo công thức sau:
$$[\text{số ngày thuê}] = [\text{ngày kết thúc thuê}] - [\text{ngày bắt đầu thuê}] + 1.$$
- Tiền thuê phòng được tính như sau:

Biết: $[\text{tiền thuê}] = [\text{số ngày thuê}] * [\text{đơn giá}]$, Trong đó: $[\text{đơn giá}]$ được tính như sau:

[Loại phòng thuê]	A	B	C	D
[đơn giá]	250 ngàn đồng/ngày	400 ngàn đồng/ngày	600 ngàn đồng/ngày	900 ngàn đồng/ngày

- b) Viết lớp CPhieuThue gồm các thông tin : Mã số phiếu thuê, ngày bắt đầu thuê, ngày kết thúc thuê, họ tên khách hàng và loại phòng thuê. Trong lớp có :

- Các phương thức khởi tạo
- Các thuộc tính (property): Mã số phiếu thuê, ngày bắt đầu thuê, ngày kết thúc thuê, họ tên khách hàng, loại phòng thuê, số ngày thuê, tiền thuê.

Lớp CPhieuThue chứa thông tin phiếu thuê phòng được thiết kế như sau:

```
public enum KieuLoaiPhong { A, B, C, D}
6 references
public class CPhieuThue
{
    private string m_mapt;
    private DateTime m_ngaybd;
    private DateTime m_ngaykt;
    private string m_tenk;
    private KieuLoaiPhong m_loaiphong;
    1 reference
    public CPhieuThue()...
    0 references
    public CPhieuThue(string mapt,DateTime ngaybd,DateTime ngaykt,
        string tenkh,KieuLoaiPhong loaiphong)...
    2 references
    public string MaPT...
    1 reference
    public DateTime NgayBD...
    1 reference
    public DateTime NgayKT...
    1 reference
    public string TenKH...
    4 references
    public KieuLoaiPhong LoaiPhong...
    4 references
    public int SoNgayThue...
    0 references
    public int TienThue...
}
```

Bài 3 - 5: XÂY DỰNG ỨNG DỤNG QUẢN LÝ DANH MỤC DỮ LIỆU

Lưu ý: SoNgayThue và TienThue là 2 thuộc tính (property) chỉ đọc (readonly).

Thiết kế giao diện form như hình sau :

	Mã phiếu thuê	Ngày bắt đầu	Ngày kết thúc	Tên khách hàng	Loại phòng	Số ngày thuê	Tiền thuê
*							

Lưu ý:

- Hai DateTimePicker thiết kế thuộc tính Format là “Custom” và thuộc tính CustomFormat là “dd/mm/yyyy H:mm”.
- Khi thiết kế các cột của DataGridView thì thuộc tính DataPropertyName của các cột trùng với tên Property của lớp cần hiển thị dữ liệu. VD: cột “Mã phiếu thuê” có thuộc tính DataPropertyName là “MaPT”.

2. Vận dụng: (4đ)

- Sử dụng đối tượng thuộc lớp Dictionary để lưu trữ danh sách các phiếu thuê.
- DataGridView hiển thị danh sách phiếu thuê. Khi chọn mẫu tin trong DataGridView, mẫu tin được chọn sẽ hiển thị lên các control trên form.
- Nút “Thêm”: thêm phiếu thuê mới vào danh sách phiếu thuê.
- Nút “Xóa”: xóa phiếu thuê đang chọn ra khỏi danh sách phiếu thuê.
- Nút “Sửa”: sửa thông tin phiếu thuê đang chọn trong danh sách phiếu thuê.
- Khi thêm phiếu thuê: nếu mã số phiếu thuê đã tồn tại trong danh sách phiếu thuê thì hiển thị thông báo “Mã số phiếu thuê đã tồn tại trong danh sách phiếu thuê. Không thêm được.”. Ngược lại, cho phép thêm thông tin phiếu thuê mới vào danh sách phiếu thuê.
- Nút “Lưu File”: lưu danh sách phiếu thuê vào file nhị phân.
- Sự kiện Load của Form: đọc thông tin danh sách phiếu thuê từ file nhị phân và hiển thị danh sách phiếu thuê lên DataGridView.

3. Bài tập tổng hợp: (2đ)

Bài 3 - 5: XÂY DỰNG ỨNG DỤNG QUẢN LÝ DANH MỤC DỮ LIỆU

Ta thực hiện lại bài trên theo mô hình 2-Tiers như sau:

- SV sao chép project phần vận dụng trên ra thành 1 project mới và thực hiện tiếp trên project mới này.
- Thiết kế lớp CXuLyPhieuThue như sau:

```
public class CXuLyPhieuThue
{
    private Dictionary<string, CPhieuThue> dsPT;
    0 references
    public CXuLyPhieuThue()...
    0 references
    public List<CPhieuThue> layDSPhieuThue()...
    0 references
    public CPhieuThue tim(string ma)...
    0 references
    public void them(CPhieuThue pt)...
    0 references
    public void xoa(string ma)...
    0 references
    public void sua(CPhieuThue pt)...
    0 references
    public bool ghiFile(string tenfile)...
    0 references
    public bool docFile(string tenfile)...
}
```

- Sử dụng đối tượng thuộc lớp Dictionary để lưu danh sách phiếu thuê.
- Phương thức khởi tạo không đối số.
- Phương thức layDSPhieuThue() để lấy danh sách phiếu thuê.
- Các phương thức tim(), them(), xoa(), sua() để tìm, thêm, xóa, sửa phiếu thuê.
- Phương thức ghiFile() để ghi danh sách phiếu thuê lên file nhị phân.
- Phương thức docFile() để đọc danh sách phiếu thuê từ file nhị phân.
- SV sử dụng lớp CXuLyPhieuThue để xử lý dữ liệu phiếu thuê trong form. Thiết kế code form như sau:

```
public partial class frmPhieuThue : Form
{
    private CXuLyPhieuThue x1PT;
    1 reference
    private void hienDSPhieuThue()...
    1 reference
    public frmPhieuThue()...
    1 reference
    private void frmPhieuThue_Load(object sender, EventArgs e)...
    1 reference
    private void btnThem_Click(object sender, EventArgs e)...
    1 reference
    private void dgvPT_RowEnter(object sender, DataGridViewCellEventArgs e)...
    1 reference
    private void btnXoa_Click(object sender, EventArgs e)...
    1 reference
    private void btnSua_Click(object sender, EventArgs e)...
    1 reference
    private void btnLuuFile_Click(object sender, EventArgs e)...
    1 reference
}
```

- Khai báo đối tượng thuộc lớp CXuLyPhieuThue.
- Phương thức hienDSPhieuThue(): hiển thị danh sách phiếu thuê lên DataGridView.
- Sự kiện frmPhieuThue_Load(): đọc danh sách phiếu thuê từ file nhị phân và hiển thị lên DataGridView.
- Các chức năng thêm, xóa, sửa, lưu file, sự kiện dgvPT_RowEnter tương tự như các mô tả của phần Vận dụng.

4. Bài tập về nhà

- a. SV thực hiện lại phần Bài tập tổng hợp nhưng sử dụng đối tượng thuộc lớp List<T> để lưu danh sách phiếu thuê.
- b. SV hãy thực hiện lại form nhập liệu cho học sinh tương tự Bài 4 nhưng viết theo mô hình 2-Tiers (tương tự phần Bài tập tổng hợp) bằng cả 2 cách:
 - Sử dụng đối tượng thuộc lớp List<T> để lưu danh sách học sinh.
 - Sử dụng đối tượng thuộc lớp Dictionary để lưu danh sách học sinh.

Bài 6, 7: TÍNH KẾ THỪA VÀ TÍNH ĐA HÌNH

I. MỤC TIÊU:

- Xây dựng các lớp kế thừa, phương thức tạo lập, phương thức tĩnh, phương thức ảo và sử dụng các thành phần trong chương trình.
- Xây dựng lớp cơ sở trừu tượng, interface, phương thức thuần ảo, tính đa hình và sử dụng trong chương trình.

II. TÓM TẮT LÝ THUYẾT:

1. Tính kế thừa

- Kế thừa là cơ chế cho phép định nghĩa một lớp mới (còn gọi là lớp dẫn xuất, derived class) dựa trên một lớp đã có sẵn (còn gọi là lớp cơ sở, base class).
- Lớp dẫn xuất có hầu hết các thành phần giống như lớp cơ sở (bao gồm tất cả các phương thức và biến thành viên của lớp cơ sở, trừ 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).

❖ Khai báo :

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

❖ Sử dụng lớp kế thừa :

- 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).
- 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):base(ThamSốLớpCha)  
{  
    // Khởi tạo giá trị cho các thành phần của lớp con  
}
```

❖ Phương thức hủy :

- Khi một đối tượng của lớp dẫn xuất được giải phóng (bị hủy), thì các đối tượng thành phần và các đối tượng thừa kế từ các lớp cơ sở cũng bị giải phóng theo. Do đó các phương thức hủy tương ứng sẽ được gọi đến.
- Như vậy khi xây dựng phương thức hủy của lớp dẫn xuất, chúng ta chỉ cần quan tâm đến các thành phần dữ liệu khai báo thêm trong lớp dẫn xuất mà thôi.

Ví dụ:

```
~PhuongThucHuyLopCha()  
{  
    //giải phóng các vùng nhớ của lớp cha  
}  
...  
~PhuongThucHuyLopCon()  
{  
    //giải phóng các vùng nhớ của lớp con  
}
```

2. Tính đa hình:

- 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 gọi 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.

a. Quá tải thành viên và tính đa hình:

Quá tải thành viên là khả năng cho phép một lớp có nhiều thuộc tính, phương thức cùng tên nhưng với các tham số khác nhau về kiểu dữ liệu, cũng như về số lượng. Khi được gọi, dựa vào tham số truyền vào, phương thức tương ứng sẽ được thực hiện.

b. Phương thức ảo và tính đa hình:

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

- Lớp cơ sở đánh dấu phương thức ảo bằng từ khóa **virtual**.
- Các lớp dẫn xuất định nghĩa lại phương thức ảo này (đánh dấu bằng từ khóa **override**).
- Vì tham chiếu thuộc lớp cơ sở có thể trỏ đến một đối tượng thuộc lớp dẫn xuất và có thể truy cập phương thức 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.

3. Lớp trừu tượng (abstract class):

- Trong 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.

4. Interface :

- Interface (nhiều tài liệu gọi là giao diện hoặc lớp giao tiếp) là 1 tập các thành phần chỉ có khai báo mà không có phần định nghĩa (giống phương thức ảo). Các thành phần này có thể là phương thức, thuộc tính, sự kiện.
- So sánh lớp trừu tượng và interface :
 - ❖ Giống nhau:
 - Điều không thể khởi tạo đối tượng bên trong được.
 - Điều có thể khai báo các phương thức nhưng không thực hiện chúng.
 - Điều bao gồm các phương thức abstract.
 - Điều được thực thi từ các class dẫn xuất.
 - Điều có thể kế thừa từ nhiều interface.

❖ Khác nhau:

Lớp trừu tượng	Interface
Cho phép khai báo thành phần dữ liệu	Không cho phép
Các phương thức có thể có thân hàm hoặc không có thân hàm.	Chỉ khai báo không có thân hàm

Bài 6, 7: TÍNH KẾ THỪA VÀ TÍNH ĐA HÌNH

Lớp dẫn xuất chỉ kế thừa được từ 1 lớp trừu tượng và nhiều interface.	Lớp triển khai có thể triển khai nhiều interface.
Có chứa constructor	Không có
Các phương thức có từ khóa access modifier (public, protected,...)	Không có

III. NỘI DUNG THỰC HÀNH:

1/ Chương trình mẫu: (2đ) (Bài này thực hiện trên ứng dụng Console)

- b. Xây dựng lớp **CThiSinh** sao cho mỗi đối tượng của lớp chứa dữ liệu của 1 thí sinh gồm mã thí sinh (biến `m_MaTS`), họ tên (`m_Hoten`) được thiết kế như sau :

```
public enum KieuKetQua { Dat, KhongDat}
public class CThiSinh
{
    protected string m_MaTS;
    protected string m_Hoten;
    public string MaTS
    {
        get { return m_MaTS; }
        set { m_MaTS = value; }
    }
    public string Hoten
    {
        get { return m_Hoten; }
        set { m_Hoten = value; }
    }
    public CThiSinh()
    {
        m_MaTS = "";
        m_Hoten = "";
    }
    public CThiSinh(string ma, string ht)
    {
        m_MaTS = ma;
        m_Hoten = ht;
    }
    public virtual double TongDiem()
    {
        return 0;
    }
    public virtual KieuKetQua KetQua()
    {
        return KieuKetQua.KhongDat;
    }
}
```

2/ Vận dụng : (6đ)

- a) Xây dựng lớp Khối A là **CKhoiA** được kế thừa từ lớp **CThiSinh** sao cho mỗi đối tượng của lớp được kế thừa từ lớp **CThiSinh**, ngoài ra còn có điểm toán, điểm lý và điểm hóa được thiết kế như sau :

```
public class CKhoiA:CThiSinh
{
    protected double m_DiemToan;
    protected double m_DiemLy;
    protected double m_DiemHoa;
    public double DiemToan...
    public double DiemLy...
    public double DiemHoa...
    public CKhoiA()...
    public CKhoiA(string ma, string ht,
        double toan, double ly, double hoa)...
    public override double TongDiem()...
    public override KieuKetQua KetQua()...
    public override bool laKhoiA()
    {
        return true;
    }
    public override bool laKhoiNangkhieu()
    {
        return false;
    }
}
```

- SV hãy viết các phương thức khởi tạo và thuộc tính của lớp.
 - Viết phương thức ảo tính tổng điểm TongDiem() biết tổng điểm = điểm toán + điểm lý + điểm hóa.
 - Viết phương thức ảo tính kết quả KetQua(): nếu tổng điểm ≥ 18.5 thì kết quả là đạt, ngược lại, kết quả là không đạt.
- b) Xây dựng lớp Khôi Năng Khiếu là **CKhoiNangkhieu** được kế thừa từ lớp **CThiSinh** sao cho mỗi đối tượng của lớp được kế thừa từ lớp **CThiSinh**, ngoài ra còn có điểm toán và điểm năng khiếu được thiết kế như sau :

```
class CKhoiNangkhieu:CThiSinh
{
    protected double m_DiemToan;
    protected double m_DiemNangKhieu;
    public double DiemToan[...]
    public double DiemNangKhieu[...]
    public CKhoiNangkhieu() [...]
    public CKhoiNangkhieu(string ma, string ht,
        double toan, double nangkhieu) [...]
    public override double TongDiem() [...]
    public override KieuKetQua KetQua() [...]
    public override bool laKhoiA()
    {
        return false;
    }
    public override bool laKhoiNangkhieu()
    {
        return true;
    }
}
```

- SV hãy viết các phương thức khởi tạo và thuộc tính của lớp.
- Viết phương thức ảo tính tổng điểm TongDiem() biết tổng điểm = điểm toán + điểm năng khiếu * 2.
- Viết phương thức ảo tính kết quả KetQua(): nếu tổng điểm ≥ 17 thì kết quả là đạt, ngược lại, kết quả là không đạt.

3/ Bài tập tổng hợp (4đ)

Trong lớp Program của ứng dụng, định nghĩa các phương thức (không được sử dụng toán tử **is** của ngôn ngữ C#):

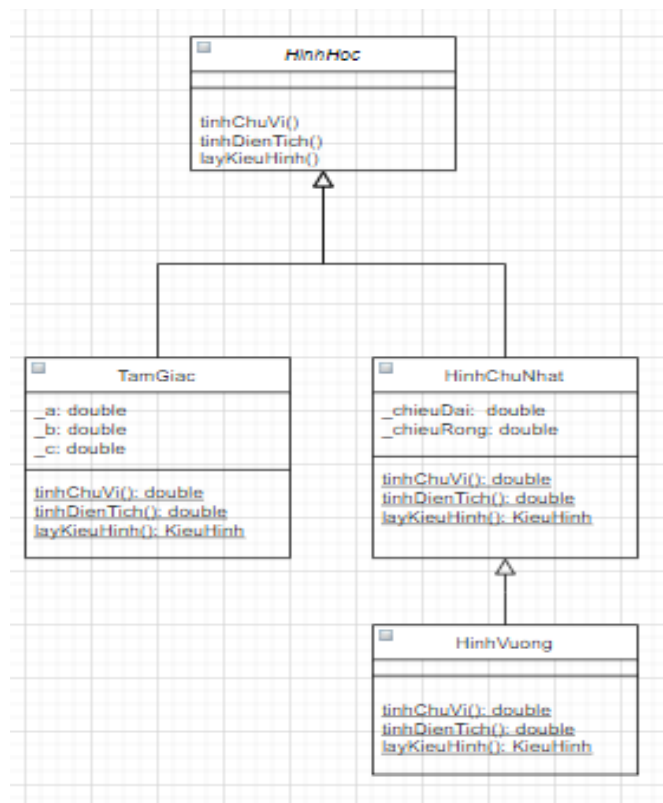
- Trả về số thí sinh thi khối năng khiếu trong danh sách thí sinh ds. Biết:
`static int getSoThiSinhKhoiNangkhieu(List<CThiSinh> ds)`
- Trả về danh sách thí sinh thi khối A có trong danh sách thí sinh ds. Biết:
`static List<CThiSinh> getDSThiSinhKhoiA(List<CThiSinh> ds)`
- Trả về điểm thi lớn nhất của khối A có trong danh sách thí sinh ds. Biết:
`static double getDiemLonNhatKhoiA(List<CThiSinh> ds)`
- Trả về danh sách tất cả các thí sinh có kết quả đạt có trong danh sách thí sinh ds. Biết:
`static List<CThiSinh> getDSThiSinhDat(List<CThiSinh> ds)`
- Trả về danh sách tất cả các thí sinh năng khiếu có kết quả là không đạt có trong danh sách thí sinh ds. Biết:
`static List<CKhoiNangkhieu> getDSThiSinhNangKhieuKhongDat(List<CThiSinh> ds)`
- Trả về danh sách tất cả các thí sinh khối A bị điểm liệt (nếu có 1 môn bất kỳ bị điểm nhỏ hơn hoặc bằng 1). Biết:
`static List<CKhoiA> getDSThiSinhKhoiABiDiemLiet(List<CThiSinh> ds)`

IV. NỘI DUNG THỰC HÀNH BÀI 7:**1. Mô tả bài toán 1****1.1. Mô tả**

Xây dựng lớp Hình học là lớp cơ sở trừu tượng. Trong đó có các lớp Tam giác, Hình chữ nhật, Hình vuông kế thừa từ lớp Hình học. Mỗi lớp đối tượng sẽ có các thuộc tính tương ứng như sau: lớp tam giác sẽ có độ dài 3 cạnh; lớp hình chữ nhật sẽ có chiều dài và chiều rộng; lớp hình vuông kế thừa từ lớp hình chữ nhật, sẽ kế thừa luôn thuộc tính chiều dài và chiều rộng của hình chữ nhật.

Trong hàm main sẽ gọi thực hiện các yêu cầu sau:

- Đếm số lượng hình chữ nhật có trong danh sách hình học đã nhập ban đầu
- Tìm hình chữ nhật có diện tích nhỏ nhất
- Tìm hình chữ nhật có chu vi lớn nhất
- Tìm danh sách hình vuông.

1.2. Sơ đồ lớp:**1.3. Chương trình mẫu****1.3.1. Lớp hình học**

Dòng	Nội dung
1	<code>public enum KieuHinh { HìnhHoc, TamGiac, HìnhChuNhat, HìnhVuong }</code>
2	<code>public abstract class HìnhHoc</code>
3	<code>{</code>
4	<code> public abstract double tinhChuVi();</code>
5	<code>}</code>

6	<code>public abstract double tinhDienTich();</code>
7	
8	<code>public abstract KieuHinh layKieuHinh();</code>
9	<code>}</code>

1.3.2. Lớp Tam giác

Dòng	Nội dung
1	<code>class TamGiac: HinhHoc</code>
2	<code>{</code>
3	<code>protected double _a;</code>
4	<code>protected double _b;</code>
5	<code>protected double _c;</code>
6	
7	<code>public double a</code>
8	<code>{</code>
9	<code>get { return this._a; }</code>
10	<code>set { this._a = value; }</code>
11	<code>}</code>
12	<code>public double b</code>
13	<code>{</code>
14	<code>get { return this._b; }</code>
15	<code>set { this._b = value; }</code>
16	<code>}</code>
17	<code>public double c</code>
18	<code>{</code>
19	<code>get { return this._c; }</code>
20	<code>set { this._c = value; }</code>
21	<code>}</code>
22	<code>public TamGiac()</code>
23	<code>{</code>
24	<code>this._a = 0;</code>
25	<code>this._b = 0;</code>
26	<code>this._c = 0;</code>
27	<code>}</code>
28	<code>public TamGiac(double _a,double _b,double _c)</code>
29	<code>{</code>
30	<code>this._a = _a;</code>
31	<code>this._b = _b;</code>
32	<code>this._c = _c;</code>
33	<code>}</code>
34	<code>public override double tinhChuVi()</code>
35	<code>{</code>
36	<code>return this._a + this._b + this._c;</code>
37	<code>}</code>
38	<code>public override double tinhDienTich()</code>
39	<code>{</code>
40	<code>double p = tinhChuVi() / 2;</code>
41	<code>return Math.Sqrt(p * (p - this._a) * (p - this._b) * (p -</code>
42	<code>this._c));</code>
43	<code>}</code>

```
44         public override KieuHinh layKieuHinh()
45         {
46             return KieuHinh.TamGiac;
47         }
48     }
```

1.3.3. Lớp Hình chữ nhật

Dòng	Nội dung
1	<code>class HìnhChuNhat:HinhHoc</code>
2	<code>{</code>
3	<code> protected double _chieuDai;</code>
4	<code> protected double _chieuRong;</code>
5	
6	<code> public double chieuDai</code>
7	<code> {</code>
8	<code> get { return this._chieuDai; }</code>
9	<code> set { this._chieuDai = value; }</code>
10	<code> }</code>
11	<code> public double chieuRong</code>
12	<code> {</code>
13	<code> get { return this._chieuRong; }</code>
14	<code> set { this._chieuRong = value; }</code>
15	<code> }</code>
16	<code> public HìnhChuNhat()</code>
17	<code> {</code>
18	<code> this._chieuDai = 0;</code>
19	<code> this._chieuRong = 0;</code>
20	<code> }</code>
21	<code> public HìnhChuNhat(double _chieuDai,double _chieuRong)</code>
22	<code> {</code>
23	<code> this._chieuDai = _chieuDai;</code>
24	<code> this._chieuRong = _chieuRong;</code>
25	<code> }</code>
26	<code> public override double tinhChuVi()</code>
27	<code> {</code>
28	<code> return (this._chieuDai + this._chieuRong) * 2;</code>
29	<code> }</code>
30	<code> public override double tinhDienTich()</code>
31	<code> {</code>
32	<code> return this._chieuDai * this._chieuRong;</code>
33	<code> }</code>
34	<code> public override KieuHinh layKieuHinh()</code>
35	<code> {</code>
36	<code> return KieuHinh.HinhChuNhat;</code>
37	<code> }</code>
38	
39	<code>}</code>

1.3.4. Lớp Hình vuông

Dòng	Nội dung
1	<code>class HìnhVuong:HìnhChuNhat</code>
2	<code>{</code>
3	<code> public HìnhVuong():base()</code>
4	<code> {</code>
5	<code> }</code>
6	<code> public HìnhVuong(double a) :base(a,a)</code>
7	<code> {</code>
8	<code> }</code>
9	<code> public override double tínhChuVi()</code>
10	<code> {</code>
11	<code> return base.tínhChuVi();</code>
12	<code> }</code>
13	<code> public override double tínhDienTich()</code>
14	<code> {</code>
15	<code> return base.tínhDienTich();</code>
16	<code> }</code>
17	<code> public override KieuHinh layKieuHinh()</code>
18	<code> {</code>
19	<code> return KieuHinh.HìnhVuong;</code>
20	<code> }</code>
21	<code>}</code>
22	
23	
24	

1.3.5. Class Program (chứa hàm main)

Dòng	Nội dung
1	<code>class Program</code>
2	<code>{</code>
3	<code> private static List<HìnhHoc> dsHinh = new List<HìnhHoc>();</code>
4	
5	<code> public static void khaiTaoDanhSachHinh()</code>
6	<code> {</code>
7	<code> HìnhHoc hìnhTamGiac1 = new TamGiac(1, 2, 3);</code>
8	<code> HìnhHoc hìnhTamGiac2 = new TamGiac(4, 5, 6);</code>
9	<code> HìnhHoc hìnhChuNhat1 = new HìnhChuNhat(7, 6);</code>
10	<code> HìnhHoc hìnhChuNhat2 = new HìnhChuNhat(9, 8);</code>
11	<code> HìnhHoc hìnhChuNhat3 = new HìnhChuNhat(11, 10);</code>
12	<code> HìnhHoc hìnhChuNhat4 = new HìnhChuNhat(13, 12);</code>
13	<code> HìnhHoc hìnhVuong1 = new HìnhVuong(4);</code>
14	<code> HìnhHoc hìnhVuong2 = new HìnhVuong(5);</code>
15	<code> HìnhHoc hìnhVuong3 = new HìnhVuong(6);</code>
16	
17	<code> dsHinh.Add(hìnhVuong1);</code>
18	<code> dsHinh.Add(hìnhChuNhat1);</code>
19	<code> dsHinh.Add(hìnhTamGiac1);</code>
20	<code> dsHinh.Add(hìnhTamGiac2);</code>
21	<code> dsHinh.Add(hìnhChuNhat2);</code>

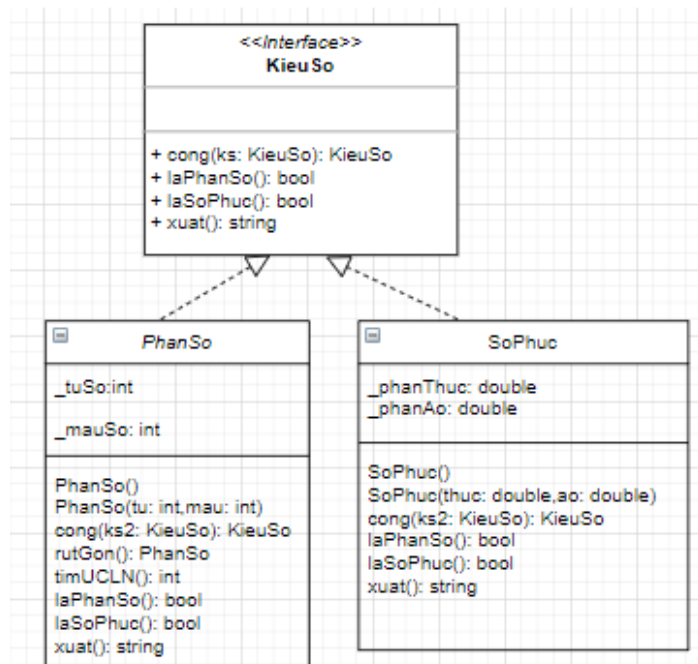
```
22         dsHinh.Add(hinhVuong2);
23         dsHinh.Add(hinhChuNhat3);
24         dsHinh.Add(hinhVuong3);
25         dsHinh.Add(hinhChuNhat4);
26
27     }
28     private static int demHinhChuNhat(List<HinhHoc> ds)
29     {
30         int soLuongHinhChuNhat = 0;
31         foreach(HinhHoc hinh in ds)
32         {
33             if (hinh.layKieuHinh().Equals(KieuHinh.HinhChuNhat))
34             {
35                 soLuongHinhChuNhat++;
36             }
37         }
38         return soLuongHinhChuNhat;
39     }
40     private static HinhHoc timHinhCoDienTichNhoNhat(List<HinhHoc> ds)
41     {
42
43     }
44     private static HinhHoc timHinhChuNhatCoChuViLonNhat(List<HinhHoc> ds)
45     {
46
47     }
48     private static List<HinhHoc> timDanhSachHinhVuong(List<HinhHoc> ds)
49     {
50         List<HinhHoc> dsHinhVuong = new List<HinhHoc>();
51         foreach(HinhHoc hinh in ds)
52         {
53             if (hinh.layKieuHinh().Equals(KieuHinh.HinhVuong))
54             {
55                 dsHinhVuong.Add(hinh);
56             }
57         }
58         return dsHinhVuong;
59     }
60     static void Main(string[] args)
61     {
62         khoiTaoDanhSachHinh();
63         int soHinhChuNhat = demHinhChuNhat(dsHinh);
64         Console.WriteLine("So hinh chu nhat la:" + soHinhChuNhat);
65
66         Console.ReadLine();
67     }
68 }
69
70
71
```

1.4. Yêu cầu:

- Tìm hình có diện tích nhỏ nhất
- Tìm hình chữ nhật có chu vi lớn nhất

2. Mô tả bài toán 2**2.1. Mô tả**

Thiết kế chương trình cho phép người dùng nhập vào 2 phân số và 2 số phức. Tính tổng 2 phân số; tính tổng 2 số phức đó; sau đó in kết quả ra màn hình.

2.2. Sơ đồ lớp:**2.3. Chương trình mẫu****2.3.1. Interface Kiểu số**

Dòng	Nội dung
1	<code>interface KieuSo</code>
2	<code>{</code>
3	<code> KieuSo cong(KieuSo a);</code>
4	<code> bool laPhanSo();</code>
5	<code> bool laSoPhuc();</code>
6	<code> string xuất();</code>
7	<code>}</code>

2.3.2. Lớp Phân số

Dòng	Nội dung
1	<code>class PhanSo:KieuSo</code>
2	<code>{</code>
3	<code> private int _tuSo;</code>

```
4     private int _mauSo;
5     public int tuSo
6     {
7         get { return this._tuSo; }
8         set { this._tuSo = value; }
9     }
10    public int mauSo
11    {
12        get { return this._mauSo; }
13        set { this._mauSo = value; }
14    }
15    public PhanSo()
16    {
17        this._tuSo = 1;
18        this._mauSo = 1;
19    }
20    public PhanSo(int tuSo, int mauSo)
21    {
22        this._tuSo = tuSo;
23        this._mauSo = mauSo;
24    }
25    //1   3   1*4 + 2*3
26    //- + - = -----
27    //2   4       2*4
28    public KieuSo cong(KieuSo a)
29    {
30        PhanSo tong = new PhanSo();
31        tong._tuSo = this._tuSo * ((PhanSo)a).mauSo + this._mauSo *
32        ((PhanSo)a)._tuSo;
33        tong._mauSo = this._mauSo * ((PhanSo)a)._mauSo;
34        PhanSo ketQua = new PhanSo();
35        ketQua = rutGon(tong);
36        return ketQua;
37    }
38    public int timUCLN(PhanSo a)
39    {
40        int tu = Math.Abs( a._tuSo);
41        int mau = Math.Abs(a._mauSo);
42        while (tu != mau)
43        {
44            if (tu > mau)
45                tu = tu - mau;
46            else
47                mau = mau - tu;
48        }
49        return tu;
50    }
51    public PhanSo rutGon(PhanSo a)
52    {
53        PhanSo ketQua = new PhanSo();
54        int uocChungLonNhat = timUCLN(a);
55        ketQua._tuSo = a._tuSo / uocChungLonNhat;
56        ketQua._mauSo = a._mauSo / uocChungLonNhat;
57        return ketQua;
58    }
59    public bool laPhanSo()
60    {
61        return true;
62    }
```

Bài 6, 7: TÍNH KẾ THỪA VÀ TÍNH ĐA HÌNH

55	}
56	public bool laSoPhuc()
57	{
58	return false;
59	}
59	public string xuat()
60	{
61	return this._tuSo + "/" + this._mauSo;
62	}
	}

2.3.3. Lớp số phức

Dòng	Nội dung
1	class SoPhuc:KieuSo
2	{
3	private double _phanThuc;
4	private double _phanAo;
5	
6	public double phanThuc
7	{
8	get { return this._phanThuc; }
9	set { this._phanThuc = value; }
10	}
11	public double phanAo
12	{
13	get { return this._phanAo; }
14	set { this._phanAo = value; }
15	}
16	public SoPhuc()
17	{
18	this._phanThuc = 1;
19	this._phanAo = 1;
20	}
21	public SoPhuc(double thuc,double ao)
22	{
23	this._phanThuc = thuc;
24	this._phanAo = ao;
25	}
26	public KieuSo cong(KieuSo a)
27	{
28	SoPhuc tong = new SoPhuc();
29	tong._phanThuc = this._phanThuc + ((SoPhuc)a)._phanThuc;
30	tong._phanAo = this._phanAo + ((SoPhuc)a)._phanAo;
31	return tong;
32	}
33	
34	public bool laPhanSo()
35	{
36	return false;
37	}
38	

39	<code>public bool laSoPhuc()</code>
40	<code>{</code>
41	<code> return true;</code>
42	<code>}</code>
43	<code>public string xuat()</code>
44	<code>{</code>
45	<code> return this._phanThuc + "+" + this._phanAo + "i";</code>
46	<code>}</code>
47	<code>}</code>

2.3.4. Hàm main

Dòng	Nội dung
1	<code>class Program</code>
2	<code>{</code>
3	<code> static void Main(string[] args)</code>
4	<code> {</code>
5	<code> PhanSo ps1 = new PhanSo(1, 2);</code>
6	<code> PhanSo ps2 = new PhanSo(3, 4);</code>
7	
8	<code> KieuSo tong = new PhanSo();</code>
9	<code> tong = ps1.cong(ps2);</code>
10	
11	<code> string ketQua = tong.xuat();</code>
12	<code> Console.WriteLine("Tong 2 phan so la:" + ketQua);</code>
13	
14	<code> SoPhuc sp1 = new SoPhuc(3, 2);</code>
15	<code> SoPhuc sp2 = new SoPhuc(5, 8);</code>
16	<code> KieuSo tong2SoPhuc = new SoPhuc();</code>
17	<code> tong2SoPhuc = sp1.cong(sp2);</code>
18	<code> string ketQuaCongSoPhuc = tong2SoPhuc.xuat();</code>
19	<code> Console.WriteLine("Tong 2 so phuc la:" + ketQuaCongSoPhuc);</code>
20	
21	
22	<code> Console.ReadLine();</code>
23	<code> }</code>
24	<code>}</code>

2.4. Yêu cầu

- Viết phương thức trừ, nhân, chia hai phân số.
- Viết phương thức trừ, nhân hai số phức.

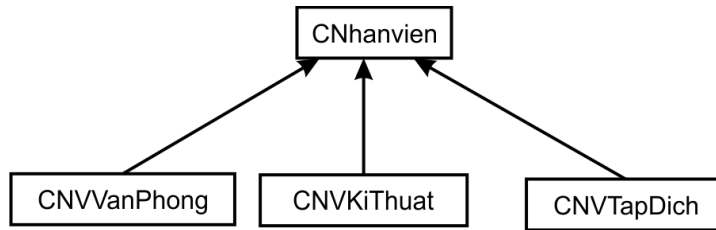
2.5. Bài tập tổng hợp

Trong lớp Program, SV hãy thực hiện:

- Viết phương thức nhập 1 dãy các phân số và số phức.
- Viết phương thức tính tổng tất cả các phân số.
- Viết phương thức tìm phân số lớn nhất.

V. BÀI TẬP LÀM THÊM

Xây dựng ứng dụng console theo mô hình như sau :

**Mô tả:**

Lớp CNhanVien bao gồm các đặc điểm sau:

Thuộc tính

- Họ tên
- Ngày sinh
- Địa chỉ

Phương thức

- Nhập
- Xuất
- Tính lương

Các lớp CNVVanPhong, CNVKiThuat, CNVTapDich kế thừa từ đối tượng CNhanVien. Tuy nhiên, chúng sẽ có những thuộc tính riêng và các phương thức Nhập, Xuất, Tính lương cũng khác nhau như sau :

Lớp CNVVanPhong

Kế thừa lớp CNhanVien, ngoài ra còn có thêm

Thuộc tính

- Hệ số lương
- Lương cơ bản

Phương thức tính lương được tính như sau :

- $Lương = Hệ\ số\ lương * lương\ cơ\ bản$

Lớp CNVKiThuat

Kế thừa lớp CNhanVien, ngoài ra còn có thêm :

Thuộc tính

- Tổng số sản phẩm sản xuất được

Phương thức tính lương được tính như sau :

- $Lương = Tổng\ số\ sản\ phẩm * 100.000$

Lớp CNVTapDich

Kế thừa từ lớp CNhanVien, ngoài ra còn có thêm

Thuộc tính

- Số ngày công

Phương thức tính lương được tính như sau :

- $Lương = \text{số ngày công} * 50000$

Hàm nhập, xuất của các đối tượng được mô tả như sau :

Cho phép người dùng nhập thông tin từ bàn phím, ứng với mỗi loại nhân viên sẽ nhập các thông tin khác nhau tùy thuộc vào số thuộc tính của loại nhân viên đó. Hàm nhập và xuất không chứa tham số đầu vào.

Minh họa

void Nhap() ;

void Xuat() ;

Yêu cầu

Xây dựng hệ thống quản lý thông tin từng nhân viên, cho phép nhập, xuất thông tin 1 nhân viên, tính tổng lương của tất cả nhân viên sử dụng

▪ Không sử dụng cơ chế đa hình

- Sử dụng 3 danh sách để quản lý 3 loại nhân viên khác nhau

List <CNVVanPhong> lstNVVP ;

List <CNVTapDich> lstNVTD ;

List <CNVKiThuat> lstNVKT ;

- Để tính tổng lương của tất cả nhân viên, phải tính tổng lương của từng loại danh sách trước, sau đó, tính tổng của 3 danh sách.

▪ Có sử dụng cơ chế đa hình

- Chỉ cần sử dụng 1 danh sách để quản lý 3 loại nhân viên khác nhau

List <CNhanVien> lstDSNV ;

- Xây dựng hàm ảo Nhập, Xuất, Tính lương ở lớp CNhanVien, đồng thời cài đặt các hàm ảo này ở lớp kế thừa tương ứng với từng loại nhân viên
- Để tính tổng lương, chỉ cần duyệt qua từng nhân viên trong danh sách, tính tổng hàm tính lương.

I. MỤC TIÊU

- Xây dựng ứng dụng hướng đối tượng: xác định các chức năng, xây dựng sơ đồ lớp xác định CTDL của ứng dụng, xây dựng sơ đồ lớp xác định kiến trúc của ứng dụng, thiết kế các thành phần tương ứng và viết chương trình.
- Cài đặt mẫu thiết kế hướng đối tượng Singleton.
- Cài đặt mối quan hệ bao hàm (độc lập và phụ thuộc).
- Cài đặt mối quan hệ phụ thuộc.
- Xây dựng ứng dụng theo mẫu 3-Tiers.
- Hiện thực các ràng buộc dữ liệu: ràng buộc khóa chính, ràng buộc khóa ngoại, ...
- Lập trình WinForm cơ bản.

II. TÓM TẮT LÝ THUYẾT

1. Các loại quan hệ:

1.1 Giới thiệu:

Khi phân tích và thiết kế lớp đối tượng cho một bài toán cụ thể, ngoài việc xác định các thành phần dữ liệu và danh sách các phương thức cần thiết cho lớp đó, chúng ta cũng cần phải xác định mối liên hệ giữa các lớp đối tượng và cách thức cài đặt tương ứng cho những mối liên hệ này.

Trong lập trình hướng đối tượng, chúng ta thường sử dụng 2 loại quan hệ cơ bản sau giữa các lớp đối tượng:

Quan hệ bao hàm/bộ phận (has-a, part-of)

Quan hệ tổng quát hóa/đặc biệt hóa (is-a)

1.2 Quan hệ bao hàm/bộ phận (has-a, part-of)

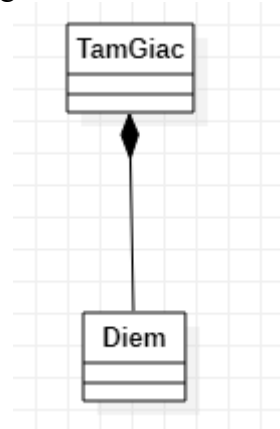
Trong mỗi quan hệ giữa 2 lớp đối tượng A và B, quan hệ bao hàm cho biết rằng lớp A có chứa đối tượng của lớp B (hay nói cách khác, lớp B là một bộ phận của lớp A).

Đối với quan hệ bao hàm, chúng ta có 2 biến thể khác nhau:

a. Quan hệ bao hàm phụ thuộc (composition)

Đối với loại quan hệ bao hàm phụ thuộc này, chu kỳ sống đối tượng của lớp B sẽ ngắn hơn hoặc bằng chu kỳ sống đối tượng của lớp A. Khi đối tượng của lớp A bị hủy, các đối tượng của lớp B trong lớp A cũng bị hủy theo.

Chẳng hạn, trong thiết kế lớp tam giác, mỗi tam giác có 3 đỉnh, khi đó chúng ta có lớp TamGiac chứa 3 điểm (3 đỉnh của tam giác). Quan hệ bao hàm của lớp TamGiac và lớp Diem được thể hiện bằng UML như sau:



✧ Cài đặt các loại quan hệ

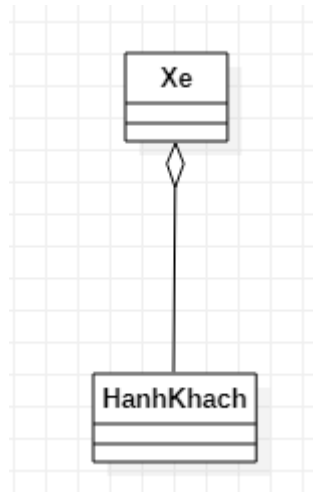
```

class Diem
{
    private double _x;
    private double _y;
    0 references
    public Diem()...
    0 references
    public Diem(double x,double y)...
    1 reference
    public double x...
    0 references
    public double y...
    0 references
    public double tinhKhoangCach(Diem d)...
}

class TamGiac
{
    private Diem _A;
    private Diem _B;
    private Diem _C;
    0 references
    public TamGiac()...
    0 references
    public TamGiac(Diem A,Diem B,Diem C)...
    0 references
    public Diem A...
    0 references
    public Diem B...
    0 references
    public Diem C...
    0 references
    public double tinhChuVi()...
    0 references
    public double tinhDienTich()...
}
    
```

b. Quan hệ bao hàm độc lập (aggregation)

Đối với loại quan hệ bao hàm độc lập, lớp B vẫn là một phần của lớp A. Tuy nhiên, không có ràng buộc về chu kỳ sống của lớp B với lớp A. Mỗi quan hệ này thường được cài đặt bên trong lớp A bằng con trỏ hay tham chiếu đến lớp B. Chẳng hạn, khi thiết kế mối quan hệ giữa lớp xe và hành khách, mỗi xe có thể chở được nhiều hành khách. Tuy nhiên, các hành khách không phụ thuộc vào một xe nào đó.



✧ Cài đặt các loại quan hệ

```

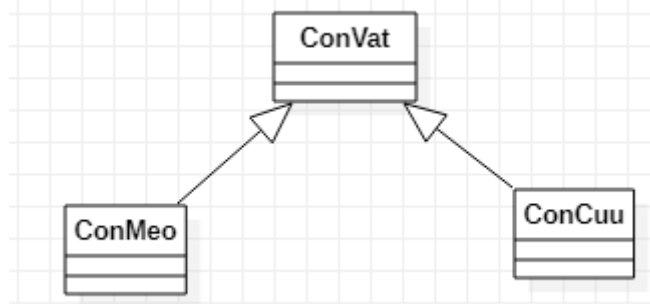
class HanhKhach
{
    private string _hoTen;
    private string _soDienThoai;
    0 references
    public HanhKhach()...
}

class Xe
{
    private List<HanhKhach> _arrKhach;
    0 references
    public Xe()...
    0 references
    public Xe(List<HanhKhach> ds) ...
}
    
```

1.3 Quan hệ tổng quát hóa/đặc biệt hóa (is-a)

Quan hệ tổng quát hóa/đặc biệt hóa được sử dụng cho mối liên hệ giữa 2 lớp đối tượng A và B trong trường hợp lớp này là một trường hợp tổng quát hay đặc biệt của lớp kia.

Ví dụ như lớp ConVat là trường hợp tổng quát hóa của lớp ConMeo và lớp ConCuu.



Trong ví dụ trên, chúng ta xem “ConMèo là một ConVat” hay “ConCuu là một ConVat” nên chúng sẽ thừa hưởng tất cả các tính chất như một ConVat.

✧ Cài đặt các loại quan hệ

```
class ConVat
{
    2 references
    public virtual void keu()
    {
    }
}
class Cuu:ConVat
{
    2 references
    public override void keu()
    {
        Console.WriteLine("Be be...");
    }
}
class Meo:ConVat
{
    2 references
    public override void keu()
    {
        Console.WriteLine("Meo meo");
    }
}
```

2. Một số loại quan hệ khác

Bên cạnh 2 loại quan hệ chính thường được sử dụng ở trên, chúng ta còn có 1 số loại quan hệ khác giữa 2 lớp đối tượng:

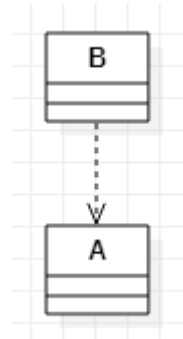
Quan hệ bạn

Quan hệ phụ thuộc

2.1 quan hệ bạn

Trong mỗi quan hệ giữa các lớp đối tượng, quan hệ bạn là quan hệ cho phép lớp bạn của mình được quyền truy xuất đến toàn bộ nội dung (dữ liệu hay phương thức) của mình, kể cả các thành phần private.

Khi lớp B là bạn của lớp A: thì lớp B có quyền truy xuất và sử dụng tất cả các thành phần (kể cả thành phần private) của lớp A.



Lưu ý: khác với 2 loại quan hệ bao hàm và quan hệ đặc biệt hóa, quan hệ bạn không tự hủy đối tượng của lớp bạn khi lớp này kết thúc chu kỳ sống.

2.2 quan hệ phụ thuộc

Quan hệ phụ thuộc cho biết khi có sự thay đổi trên lớp chính thì có thể gây ra sự thay đổi trên lớp phụ thuộc. Tuy nhiên, lớp chính hoàn toàn độc lập đối với các thay đổi từ lớp phụ thuộc.

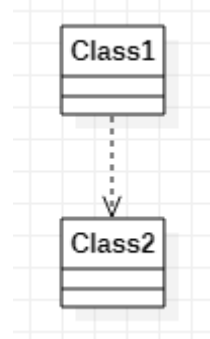
Quan hệ phụ thuộc thường xuất hiện trong các trường hợp sau:

Lớp phụ thuộc sử dụng một biến toàn cục là lớp chính

Lớp phụ thuộc sử dụng lớp chính là một tham số trong hành động/phương thức của nó

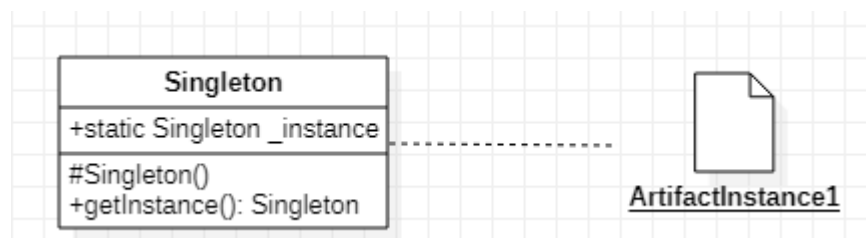
Lớp phụ thuộc sử dụng biến cục bộ là lớp chính

Lớp phụ thuộc gửi thông điệp đến cho lớp chính.



3. Lớp Singleton

Mẫu thiết kế Singleton (Singleton Design Pattern) là một mẫu thiết kế trong lập trình hướng đối tượng. Áp dụng mẫu Singleton lên một lớp sẽ giúp đảm bảo lớp này chỉ có nhiều nhất một đối tượng được tạo ra trong suốt chương trình. Sơ đồ UML của mẫu Singleton được thể hiện như sau:



Ý tưởng cài đặt của mẫu Singleton: để đảm bảo chỉ có một đối tượng duy nhất của lớp được tạo ra trong suốt chương trình, chúng ta cần giải quyết các vấn đề sau:

- Không cho phép người dùng tự ý tạo ra một đối tượng Singleton, ví nếu người dùng được tạo thoải mái thì họ sẽ có thể tạo nhiều hơn một đối tượng trong chương trình. Để làm được điều này, ta sẽ quản lý hàm khởi tạo sao cho người dùng không thể tự ý dùng hàm này để tạo đối tượng mới được bằng cách đặt các hàm khởi tạo của lớp Singleton trong phạm vi private.
- Chúng ta đảm bảo chỉ có tối đa một đối tượng Singleton được tạo ra trong suốt chương trình bằng cách tạo sẵn một đối tượng ở dạng static. Đối tượng này sẽ là đối tượng Singleton duy nhất (nếu có) và người dùng sẽ chỉ được phép sử dụng đối tượng đã có sẵn này.

Ví dụ (một đoạn code)

```

9 references
class Singleton
{
    private static Singleton _instance;
    1 reference
    private Singleton()
    {

    }
    1 reference
    public static Singleton getInstance()
    {
        if (_instance == null)
            _instance = new Singleton();
        return _instance;
    }
}

```

III. NỘI DUNG THỰC HÀNH 1

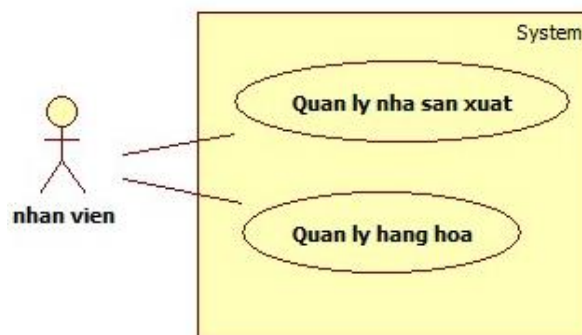
1. Mô tả bài toán

Xây dựng ứng dụng quản lý thông tin hàng hóa. Mỗi loại hàng hóa có 1 nhà sản xuất.

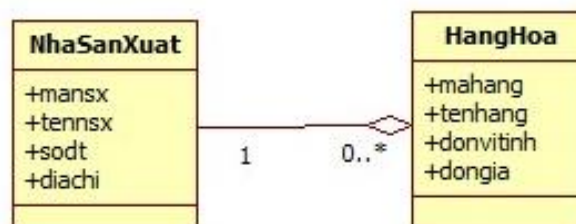
Trong bài thực hành này, chúng ta sẽ cài đặt các mối quan hệ giữa các lớp đối tượng như: quan hệ bao hàm, quan hệ phụ thuộc,...

Trong đó, khi phân tích chúng ta sẽ thấy mỗi hàng hóa sẽ có một nhà sản xuất nào đó. Vì vậy mối quan hệ giữa lớp đối tượng hàng hóa và lớp đối tượng nhà sản xuất là quan hệ bao hàm độc lập.

2. Sơ đồ Usecase



3. Cấu trúc dữ liệu



4. Giao diện ứng dụng

4.1. Quản lý nhà sản xuất

Form Nhà sản xuất

Mã nhà sản xuất

ximang001

Tên nhà sản xuất

Nhà máy xi măng Hà Tiên

Số điện thoại

0878.834.555

Địa chỉ

348 Xa lộ Hà Nội

Thêm

Xóa

Sửa

	Mã nhà sản xuất	Tên nhà sản xuất	Số điện thoại	Địa chỉ
▶	ximang001	Nhà máy xi măng Hà Tiên	0878.834.555	348 Xa lộ Hà Nội
	sat001	Công ty Hoa Sen	0437.876.221	876 Trần Hưng Đạo
	gach001	Công ty Prime	0879.334.553	360 Lê Lai
*				

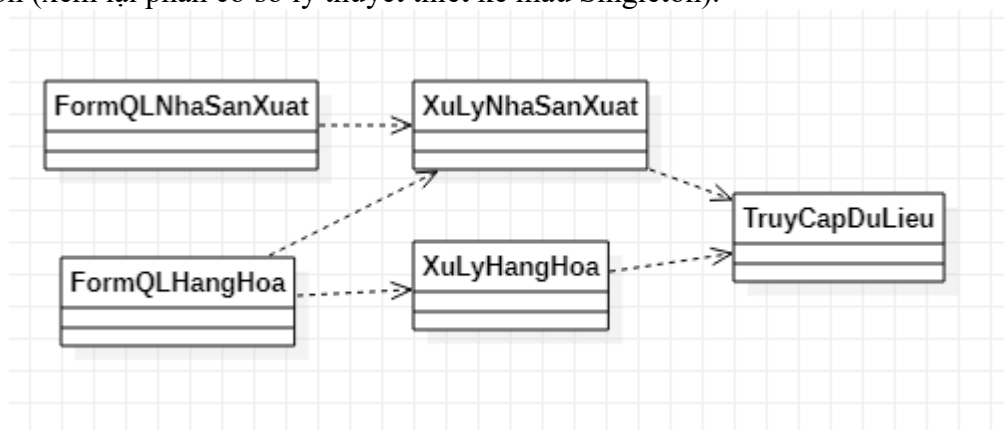
4.2. Quản lý hàng hóa

	Mã hàng	Tên hàng	Đơn vị tính	Đơn giá	Mã nhà sản xuất
▶	mh001	Sắt Thái Nguyên phi 10	cây	72	sat001
	mh002	Xi măng Hà tiên p300	bao	280	ximang001

5. Xây dựng ứng dụng trên 3-tier

5.1. Sơ đồ lớp

Khi cài đặt bằng mô hình 3-tier thì các tầng sẽ đóng vai trò cụ thể như sau: giao diện là các form sẽ giữ vai trò tương tác với người dùng, nhận các điều khiển, yêu cầu; sau khi nhận các yêu cầu sẽ đến tầng thứ 2 xử lý các yêu cầu cụ thể của từng lớp đối tượng; nếu có yêu cầu thao tác với file sẽ đến tầng thứ 3 (cài đặt các hàm thao tác dữ liệu). Chúng ta cài đặt tầng thứ 3 theo mẫu Singleton (xem lại phần cơ sở lý thuyết thiết kế mẫu Singleton).



5.2. Chương trình mẫu

5.2.1. Lớp truy cập dữ liệu


```
[Serializable]
11 references
class TruyCapDuLieu
{
    private static TruyCapDuLieu instance = null;
    private List<NhaSanXuat> dsNhaSanXuat;
    private List<HangHoa> dsHangHoa;
    1 reference
    private TruyCapDuLieu()...
    2 references
    public static TruyCapDuLieu khoiTao()...
    1 reference
    public List<HangHoa> getDanhSachHangHoa()...
    1 reference
    public List<NhaSanXuat> getDanhSachNhaSanXuat()...
    1 reference
    public static bool docFile(string tenFile)...
    1 reference
    public static bool ghiFile(string tenFile)...
}
```

5.2.2. Lớp Nhà sản xuất

```
[Serializable]
26 references
class NhaSanXuat
{
    private string _maNhaSanXuat;
    private string _tenNhaSanXuat;
    private string _dienThoai;
    private string _diaChi;

    8 references
    public string maNhaSanXuat...
    5 references
    public string tenNhaSanXuat...
    1 reference
    public string dienThoai...
    1 reference
    public string diaChi...
    3 references
    public NhaSanXuat()...
    1 reference
    public NhaSanXuat(string ma,string ten,string dienThoai,string diaChi)...
}
```

5.2.3. Lớp Hàng hóa

[Serializable]

25 references

class HangHoa

```
{
    private string _maHang;
    private string _tenHang;
    private string _donViTinh;
    private double _donGia;
    private NhaSanXuat _nhaSanXuat;

    5 references
    public string maHang[...]
    1 reference
    public string tenHang[...]
    1 reference
    public string donViTinh[...]
    1 reference
    public double donGia[...]
    4 references
    public NhaSanXuat nhaSanXuat[...]
    1 reference
    public HangHoa()[...]
    2 references
    public HangHoa(string ma,string ten,string donViTinh,double donGia,NhaSanXuat nhaSanXuat)...
```

5.2.4. Lớp Xử lý nhà sản xuất

class XuLyNhaSanXuat

```
{
    private List<NhaSanXuat> _dsNhaSanXuat;
    1 reference
    public XuLyNhaSanXuat()...
    2 references
    public List<NhaSanXuat> getDanhSachNhaSanXuat()...
    1 reference
    public void them(NhaSanXuat nsx)...
    0 references
    public void xoa(string maNsx)...
    0 references
    public void sua(NhaSanXuat nsx)...
    0 references
    public NhaSanXuat tim(string maNsx)...
```

5.2.5. Lớp Xử lý hàng hóa

```
class XuLyHangHoa
{
    private List<HangHoa> dsHangHoa;
    private List<NhaSanXuat> dsNhaSanXuat;
    1 reference
    public XuLyHangHoa()...
    4 references
    public List<HangHoa> getDanhSachHangHoa()...
    1 reference
    public List<NhaSanXuat> getDanhSachNhaSanXuat()...

    1 reference
    public void them(HangHoa hh)...
    1 reference
    public void xoa(string maHang)...
    1 reference
    public void sua(HangHoa hh)...
    0 references
    public HangHoa tim(string maHang)...
}
```

5.2.6. Giao diện quản lý nhà sản xuất

```
public partial class frmQuanLyNhaSanXuat : Form
{
    private XuLyNhaSanXuat xuLy;
    1 reference
    public frmQuanLyNhaSanXuat()...
    1 reference
    private void frmQuanLyNhaSanXuat_Load(object sender, EventArgs e)...
    1 reference
    private void btnThem_Click(object sender, EventArgs e)...
    2 references
    private void hienThiDanhSachNhaSanXuat(DataGridView dgv, List<NhaSanXuat> ds)...
```

5.2.7. Giao diện quản lý hàng hóa

```
public partial class frmQuanLyHangHoa : Form
{
    private XuLyHangHoa xuLyHang;
    1 reference
    public frmQuanLyHangHoa()...
    1 reference
    private void frmQuanLyHangHoa_Load(object sender, EventArgs e)...
    1 reference
    private void btnThem_Click(object sender, EventArgs e)...
    1 reference
    private void btnXoa_Click(object sender, EventArgs e)...
    1 reference
    private void btnSua_Click(object sender, EventArgs e)...
    1 reference
    private void hienThiDanhSachNhaSanXuat(ComboBox cbo, List<string> ds)...
    4 references
    private void hienThiDanhSachHangHoa(DataGridView dgv, List<HangHoa> ds)...
}

```

5.2.8. Giao diện chính

```
public partial class Form1 : Form
{
    1 reference
    public Form1()...
    1 reference
    private void quảnLyNhàSảnXuấtToolStripMenuItem_Click(object sender, EventArgs e)...
    1 reference
    private void quảnLyHàngHóaToolStripMenuItem_Click(object sender, EventArgs e)...
    1 reference
    private void ghiDữLiệuToolStripMenuItem_Click(object sender, EventArgs e)...
    1 reference
    private void Form1_Load(object sender, EventArgs e)...
}

```

5.3. Cài đặt

5.4. Lớp Truy cập dữ liệu

Dòng	Nội dung
1	<code>class TruyCapDuLieu</code>
2	<code>{</code>
3	<code>private static TruyCapDuLieu instance = null;</code>
4	<code>private List<NhaSanXuat> dsNhaSanXuat;</code>
5	<code>private List<HangHoa> dsHangHoa;</code>
6	<code>private TruyCapDuLieu()</code>
7	<code>{</code>
8	<code>dsNhaSanXuat = new List<NhaSanXuat>();</code>
9	<code>dsHangHoa = new List<HangHoa>();</code>
10	<code>}</code>
11	<code>public static TruyCapDuLieu khaiTao()</code>
12	<code>{</code>
13	<code>if (instance == null)</code>
14	<code>instance = new TruyCapDuLieu();</code>
15	<code>return instance;</code>
16	<code>}</code>
17	<code>}</code>
18	
19	

```

20     public List<HangHoa> getDanhSachHangHoa()
21     {
22         return dsHangHoa;
23     }
24     public List<NhaSanXuat> getDanhSachNhaSanXuat()
25     {
26         return dsNhaSanXuat;
27     }
28     public static bool docFile(string tenFile)
29     {
30         try
31         {
32             FileStream fs = new FileStream(tenFile,
33 FileMode.Open);
34             BinaryFormatter bf = new BinaryFormatter();
35             instance = (TruyCapDuLieu)bf.Deserialize(fs);
36             fs.Close();
37             return true;
38         }
39         catch (Exception err)
40         {
41             return false;
42         }
43     }
44     public static bool ghiFile(string tenFile)
45     {
46         try
47         {
48             FileStream fs = new FileStream(tenFile,
49 FileMode.Create);
50             BinaryFormatter bf = new BinaryFormatter();
51             bf.Serialize(fs, instance);
52             fs.Close();
53             return true;
54         }
55         catch (Exception err)
56         {
57             return false;
58         }
59     }
60 }

```

5.5. Lớp Nhà sản xuất

Dòng	Nội dung
1	<code>class NhaSanXuat</code>
2	<code>{</code>
3	<code>private string _maNhaSanXuat;</code>
4	<code>private string _tenNhaSanXuat;</code>
5	<code>private string _dienThoai;</code>
6	<code>private string _diaChi;</code>
7	
8	<code>public string maNhaSanXuat</code>
9	<code>{</code>
10	<code>get { return this._maNhaSanXuat; }</code>
11	<code>set { this._maNhaSanXuat = value; }</code>
12	<code>}</code>
13	<code>public string tenNhaSanXuat</code>
14	<code>{</code>
15	<code>get { return this._tenNhaSanXuat; }</code>
16	<code>set { this._tenNhaSanXuat = value; }</code>
17	<code>}</code>
18	<code>public string dienThoai</code>
19	<code>{</code>
20	<code>get { return this._dienThoai; }</code>
21	<code>set { this._dienThoai = value; }</code>
22	<code>}</code>
23	<code>public string diaChi</code>
24	<code>{</code>
25	<code>get { return this._diaChi; }</code>
26	<code>set { this._diaChi = value; }</code>
27	<code>}</code>
28	<code>public NhaSanXuat()</code>
29	<code>{</code>
30	<code>this._maNhaSanXuat = null;</code>
31	<code>this._tenNhaSanXuat = null;</code>
32	<code>this._dienThoai = null;</code>
33	<code>this._diaChi = null;</code>
34	<code>}</code>
35	<code>public NhaSanXuat(string ma,string ten,string dienThoai,string</code>
36	<code>diaChi)</code>
37	<code>{</code>
38	<code>this._maNhaSanXuat = ma;</code>
39	<code>this._tenNhaSanXuat = ten;</code>
40	<code>this._dienThoai = dienThoai;</code>
41	<code>this._diaChi = diaChi;</code>
42	<code>}</code>
43	<code>}</code>

5.6. Lớp Hàng hóa

Dòng	Nội dung
1	<code>class HangHoa</code>
2	<code>{</code>
3	<code>private string _maHang;</code>
4	<code>private string _tenHang;</code>
5	<code>private string _donViTinh;</code>
6	<code>private double _donGia;</code>
7	<code>private NhaSanXuat _nhaSanXuat;</code>
8	<code>public string maHang</code>
9	<code>{</code>
10	<code>get { return this._maHang; }</code>
11	<code>set { this._maHang = value; }</code>
12	<code>}</code>
13	<code>public string tenHang</code>
14	<code>{</code>
15	<code>get { return this._tenHang; }</code>
16	<code>set { this._tenHang = value; }</code>
17	<code>}</code>
18	<code>public string donViTinh</code>
19	<code>{</code>
20	<code>get { return this._donViTinh; }</code>
21	<code>set { this._donViTinh = value; }</code>
22	<code>}</code>
23	<code>public double donGia</code>
24	<code>{</code>
25	<code>get { return this._donGia; }</code>
26	<code>set { this._donGia = value; }</code>
27	<code>}</code>
28	<code>public NhaSanXuat nhaSanXuat</code>
29	<code>{</code>
30	<code>get { return this._nhaSanXuat; }</code>
31	<code>set { this._nhaSanXuat = value; }</code>
32	<code>}</code>
33	<code>public HangHoa()</code>
34	<code>{</code>
35	<code>this._maHang = null;</code>
36	<code>this._tenHang = null;</code>
37	<code>this._donViTinh = null;</code>
38	<code>this._donGia = 0;</code>
39	<code>this._nhaSanXuat = null;</code>
40	<code>}</code>
41	<code>public HangHoa(string ma,string ten,string donViTinh,double</code>
42	<code>donGia,NhaSanXuat nhaSanXuat)</code>
	<code>{</code>
	<code>this._maHang = ma;</code>
	<code>this._tenHang = ten;</code>
	<code>this._donViTinh = donViTinh;</code>
	<code>this._donGia = donGia;</code>
	<code>this._nhaSanXuat = nhaSanXuat;</code>

43	}
44	}

5.7. Lớp xử lý nhà sản xuất

Dòng	Nội dung
1	<code>class XuLyNhaSanXuat</code>
2	<code>{</code>
3	<code> private List<NhaSanXuat> _dsNhaSanXuat;</code>
4	<code> public XuLyNhaSanXuat()</code>
5	<code> {</code>
6	<code> TruyCapDuLieu duLieu = TruyCapDuLieu.khoiTao();</code>
7	<code> this._dsNhaSanXuat = duLieu.getDanhsachNhaSanXuat();</code>
8	<code> }</code>
9	<code> public List<NhaSanXuat> getDanhsachNhaSanXuat()</code>
10	<code> {</code>
11	<code> return this._dsNhaSanXuat;</code>
12	<code> }</code>
13	<code> public void them(NhaSanXuat nsx)</code>
14	<code> {</code>
15	<code> this._dsNhaSanXuat.Add(nsx);</code>
16	<code> }</code>
17	<code> public void xoa(string maNsx)</code>
18	<code> {</code>
19	<code> foreach (NhaSanXuat nsx in this._dsNhaSanXuat)</code>
20	<code> {</code>
21	<code> if (nsx.maNhaSanXuat.Equals(maNsx))</code>
22	<code> {</code>
23	<code> this._dsNhaSanXuat.Remove(nsx);</code>
24	<code> break;</code>
25	<code> }</code>
26	<code> }</code>
27	<code> }</code>
28	<code> public void sua(NhaSanXuat nsx)</code>
29	<code> {</code>
30	<code> NhaSanXuat ketQuaTim = tim(nsx.maNhaSanXuat);</code>
31	<code> if(ketQuaTim !=null)</code>
32	<code> {</code>
33	<code> ketQuaTim.tenNhaSanXuat = nsx.tenNhaSanXuat;</code>
34	<code> ketQuaTim.diaChi = nsx.diaChi;</code>
35	<code> ketQuaTim.dienThoai = nsx.dienThoai;</code>
36	<code> }</code>
37	<code> }</code>
38	<code> public NhaSanXuat tim(string maNsx)</code>
39	<code> {</code>
40	<code> // ...</code>
41	<code> }</code>
42	<code>}</code>
43	<code></code>
44	<code></code>

45	
46	}
47	
48	}
49	

5.8. Lớp xử lý hàng hóa

Dòng	Nội dung
1	<code>class XuLyHangHoa</code>
2	<code>{</code>
3	<code> private List<HangHoa> dsHangHoa;</code>
4	<code> private List<NhaSanXuat> dsNhaSanXuat;</code>
5	<code> public XuLyHangHoa()</code>
6	<code> {</code>
7	<code> dsHangHoa = new List<HangHoa>();</code>
8	<code> TruyCapDuLieu duLieu = TruyCapDuLieu.khoiTao();</code>
9	<code> this.dsHangHoa = duLieu.getDanhsachHangHoa();</code>
10	<code> this.dsNhaSanXuat = duLieu.getDanhsachNhaSanXuat();</code>
11	<code> }</code>
12	<code> public List<HangHoa> getDanhsachHangHoa()</code>
13	<code> {</code>
14	<code> return dsHangHoa;</code>
15	<code> }</code>
16	<code> public List<NhaSanXuat> getDanhsachNhaSanXuat()</code>
17	<code> {</code>
18	<code> return dsNhaSanXuat;</code>
19	<code> }</code>
20	<code> public void them(HangHoa hh)</code>
21	<code> {</code>
22	<code> }</code>
23	<code> public void xoa(string maHang)</code>
24	<code> {</code>
25	<code> }</code>
26	<code> }</code>
27	<code> public void sua(HangHoa hh)</code>
28	<code> {</code>
29	<code> }</code>
30	<code> }</code>
31	<code> public HangHoa tim(string maHang)</code>
32	<code> {</code>
33	<code> }</code>
34	<code> }</code>
35	<code>}</code>
36	
37	
38	
39	
40	

5.9. Lớp giao diện quản lý nhà sản xuất

Dòng	Nội dung
1	<code>public partial class frmQuanLyNhaSanXuat : Form</code>
2	<code>{</code>
3	<code> private XuLyNhaSanXuat xuLy;</code>
4	<code> public frmQuanLyNhaSanXuat()</code>
5	<code> {</code>
6	<code> InitializeComponent();</code>
7	<code> xuLy = new XuLyNhaSanXuat();</code>
8	<code> }</code>
9	<code> private void frmQuanLyNhaSanXuat_Load(object sender,</code>
10	<code>EventArgs e)</code>
11	<code> {</code>
12	<code> List<NhaSanXuat> dsNhaSanXuat =</code>
13	<code> xuLy.getDanhsachNhaSanXuat();</code>
14	<code> hienThiDanhsachNhaSanXuat(dgvDanhsachNhaSanXuat</code>
15	<code> ,dsNhaSanXuat);</code>
16	<code> }</code>
17	<code> private void btnThem_Click(object sender, EventArgs e)</code>
18	<code> {</code>
19	<code> NhaSanXuat nsx = new</code>
20	<code> NhaSanXuat(txtMaNhaSanXuat.Text, txtTenNhaSanXuat.Text,</code>
21	<code> txtDienThoaiNhaSanXuat.Text, txtDiaChiNhaSanXuat.Text);</code>
22	<code> xuLy.them(nsx);</code>
23	<code> hienThiDanhsachNhaSanXuat(dgvDanhsachNhaSanXuat,</code>
24	<code> xuLy.getDanhsachNhaSanXuat());</code>
25	<code> }</code>
26	<code> private void hienThiDanhsachNhaSanXuat(DataGridView</code>
27	<code> dgv,List<NhaSanXuat> ds)</code>
28	<code> {</code>
29	<code> dgv.DataSource = null;</code>
30	<code> dgv.DataSource = ds;</code>
31	<code> dgv.Refresh();</code>
32	<code> }</code>
33	<code>}</code>
34	
35	
36	
37	
38	
39	
40	
41	
42	
43	

5.10. Lớp giao diện quản lý hàng hóa

Dòng	Nội dung
1	<code>public partial class frmQuanLyHangHoa : Form</code>
2	<code>{</code>
3	<code>private XuLyHangHoa xuLyHang;</code>
4	<code>public frmQuanLyHangHoa()</code>
5	<code>{</code>
6	<code>InitializeComponent();</code>
7	<code>}</code>
8	<code>private void frmQuanLyHangHoa_Load(object sender,</code>
9	<code>EventArgs e)</code>
10	<code>{</code>
11	<code>xuLyHang = new XuLyHangHoa();</code>
12	<code></code>
13	<code>//Hiển thị danh sách nhà sản xuất lên combox</code>
14	<code>hienThiDanhSachNhaSanXuat(cboTenNhaSanXuat,</code>
15	<code>xuLyHang.getDanhsachNhaSanXuat());</code>
16	<code></code>
17	<code>//Hiển thị danh sách hàng hóa lên datagridview</code>
18	<code>hienThiDanhSachHangHoa(dgvDanhSachHangHoa,</code>
19	<code>xuLyHang.getDanhsachHangHoa());</code>
20	<code>}</code>
21	<code>private void btnThem_Click(object sender, EventArgs e)</code>
22	<code>{</code>
23	<code>NhaSanXuat nsx = new NhaSanXuat();</code>
24	<code>nsx.tenNhaSanXuat = cboTenNhaSanXuat.Text;</code>
25	<code>nsx.maNhaSanXuat =</code>
26	<code>cboTenNhaSanXuat.SelectedValue.ToString();</code>
27	<code></code>
28	<code>HangHoa hh = new HangHoa(txtMaHang.Text,</code>
29	<code>txtTenHang.Text, txtDonViTinh.Text,</code>
30	<code>double.Parse(txtDonGia.Text), nsx);</code>
31	<code>xuLyHang.them(hh);</code>
32	<code>hienThiDanhSachHangHoa(dgvDanhSachHangHoa,</code>
33	<code>xuLyHang.getDanhsachHangHoa());</code>
34	<code>}</code>
35	<code>private void btnXoa_Click(object sender, EventArgs e)</code>
36	<code>{</code>
37	<code>string maHang = txtMaHang.Text;</code>
38	<code>xuLyHang.xoa(maHang);</code>
39	<code>hienThiDanhSachHangHoa(dgvDanhSachHangHoa,</code>
40	<code>xuLyHang.getDanhsachHangHoa());</code>
41	<code>}</code>
42	<code>}</code>
43	<code>}</code>
44	<code>}</code>
45	<code>}</code>
46	<code>}</code>
47	<code>}</code>
48	<code>}</code>
49	<code>}</code>

50	}
51	private void btnSua_Click(object sender, EventArgs e)
52	{
53	NhaSanXuat nsx = new NhaSanXuat();
54	nsx.tenNhaSanXuat = cboTenNhaSanXuat.Text;
55	nsx.maNhaSanXuat =
56	cboTenNhaSanXuat.SelectedValue.ToString();
57	
58	
59	HangHoa hh = new HangHoa(txtMaHang.Text, txtTenHang.Text,
60	txtDonViTinh.Text, double.Parse(txtDonGia.Text), nsx);
61	xuLyHang.sua(hh);
62	hienThiDanhSachHangHoa(dgvDanhSachHangHoa,
63	xuLyHang.getDanhsachHangHoa());
64	}
65	private void hienThiDanhSachNhaSanXuat(ComboBox cbo,
66	List<string> ds)
67	{
68	cbo.DisplayMember = "tenNhaSanXuat";
69	cbo.ValueMember = "maNhaSanXuat";
70	cbo.DataSource = ds;
71	}
72	private void hienThiDanhSachHangHoa(DataGridView dgv,
73	List<HangHoa> ds)
74	{
75	dgv.DataSource = null;
76	dgv.DataSource = ds;
77	dgv.Refresh();
78	}
79	}
80	
81	
82	
83	
84	

5.11. Lớp giao diện chính

Dòng	Nội dung
1	public partial class Form1 : Form
2	{
3	public Form1()
4	{
5	InitializeComponent();
6	}
7	private void Form1_Load(object sender, EventArgs e)
8	{
9	TruyCapDuLieu.docFile("HangHoa.dat");
10	}
11	

```
12         }
13         private void
14 quảnLýNhàSảnXuấtToolStripMenuItem_Click(object sender, EventArgs
15 e)
16     {
17         frmQuanLyNhaSanXuat frm = new frmQuanLyNhaSanXuat();
18         frm.ShowDialog();
19     }
20     private void quảnLýHàngHóaToolStripMenuItem_Click(object
21 sender, EventArgs e)
22     {
23         frmQuanLyHangHoa frm = new frmQuanLyHangHoa();
24         frm.ShowDialog();
25     }
26     private void ghiDữLiệuToolStripMenuItem_Click(object
27 sender, EventArgs e)
28     {
29         bool ketQuaGhiFile =
30 TruyCapDuLieu.ghiFile("HangHoa.dat");
31         if (ketQuaGhiFile == true)
32             MessageBox.Show("Đã ghi file!", "Thông báo",
33 MessageBoxButtons.OK, MessageBoxIcon.Information);
34         else
35             MessageBox.Show("Ghi file thất bại!", "Thông
36 báo", MessageBoxButtons.OK, MessageBoxIcon.Information);
37     }
38 }
39
40
41
42
43
44
45
46
47
48
49
```

5.12. Yêu cầu

- Viết hàm xóa nhà sản xuất
- Viết hàm sửa thông tin nhà sản xuất
- Viết hàm thêm hàng hóa
- Viết hàm sửa thông tin hàng hóa
- Viết hàm xóa hàng hóa

IV. NỘI DUNG THỰC HÀNH 2

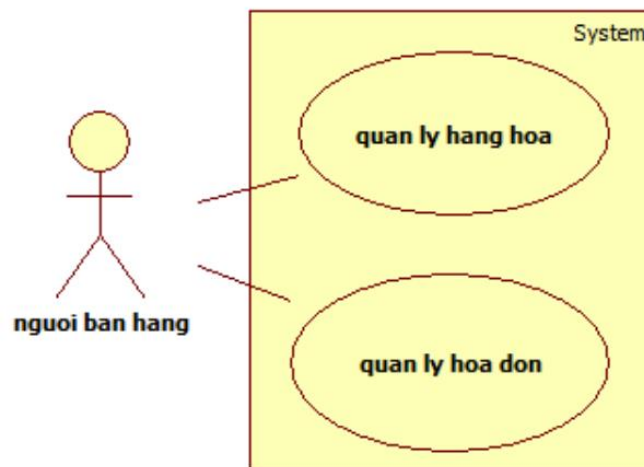
1. Mô tả bài toán

Quản lý hóa đơn bán hàng. Người dùng sẽ chọn món hàng cần mua, nhấn nút “chọn hàng hóa” để đưa món hàng mình chọn vào danh sách các mặt hàng. Khi người dùng nhấn chọn 1 mặt hàng thì đơn vị tính và giá tiền của món hàng đó sẽ tự động lấy dữ liệu để điền vào các ô tương ứng. Người dùng chỉ việc nhập số lượng cần mua, chương trình sẽ tự động tính thành tiền món hàng đó.

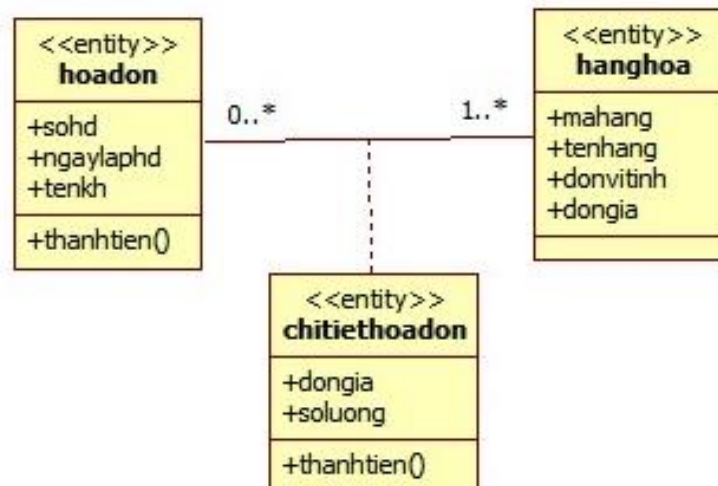
Trong bài thực hành này, chúng ta sẽ cài đặt các mối quan hệ giữa các lớp đối tượng như: quan hệ bao hàm, quan hệ phụ thuộc,...

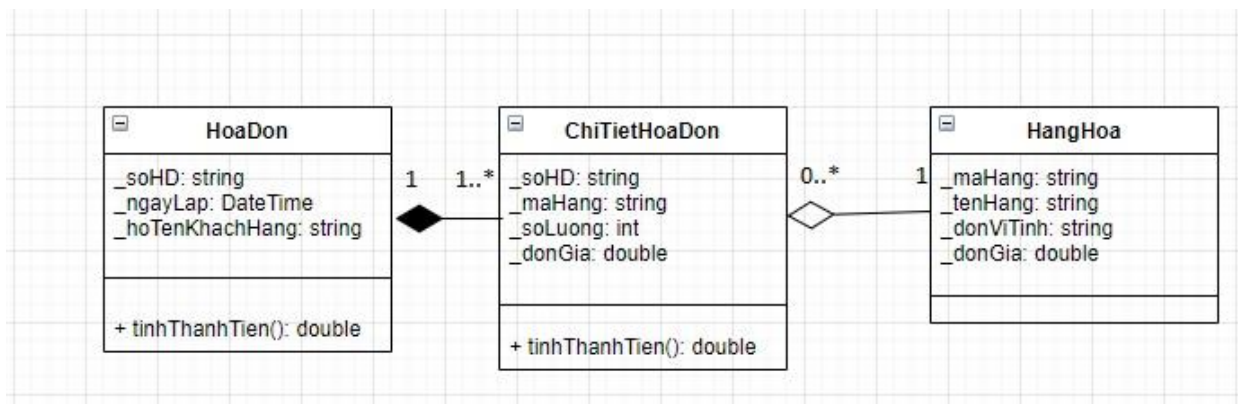
Mỗi hóa đơn sẽ có chi tiết hóa đơn, liệt kê danh sách các mặt hàng đã chọn. Vì thế chúng ta có mối quan hệ giữa lớp hóa đơn và lớp chi tiết hóa đơn là bao hàm phụ thuộc; chi tiết hóa đơn sẽ gồm các mặt hàng nào hoặc chưa có mặt hàng nào, vì thế chúng ta có mối quan hệ giữa lớp chi tiết hóa đơn và lớp mặt hàng là bao hàm độc lập.

2. Sơ đồ Usecase



3. Cấu trúc dữ liệu





4. Giao diện ứng dụng

4.1. Quản lý hàng hóa

The screenshot shows the 'FormHanghoa' application window. It contains input fields for 'Mã hàng' (001), 'Tên hàng' (Gạo thơm Xuân hồng), 'Đơn vị tính' (túi 5kg), and 'Đơn giá' (110). To the right are buttons for 'Thêm', 'Xóa', and 'Sửa'. Below these is a table with columns 'Mã hàng', 'Tên hàng', and 'Đơn vị tính'.

	Mã hàng	Tên hàng	Đơn vị tính
▶	001	Gạo thơm Xuân hồng	túi 5kg
	002	Nước mắm Phú quốc	chai
	003	Mì gói Cung đình	gói
	004	Gạo ST25 Sóc trắng	túi 5kg
*			

4.2. Quản lý hóa đơn

The screenshot shows a Windows application window titled "frmHoaDon". It contains several input fields and buttons for creating an invoice. At the top, there are fields for "Số hóa đơn" (Invoice Number), "Ngày lập hóa đơn" (Invoice Date) with a calendar icon, and "Tên khách hàng" (Customer Name). Below these are fields for "Mã hàng" (Item Code), "Tên hàng" (Item Name) with a dropdown arrow, "Đơn vị tính" (Unit), "Đơn giá" (Unit Price), and "Số lượng" (Quantity). Buttons "Lập hóa đơn" (Create Invoice) and "Chọn hàng hóa" (Select Goods) are also present. On the right, there is a "Danh mục hóa đơn" (Invoice Category) section with a "Xem hóa đơn" (View Invoice) button and a table with columns "Số hóa đơn", "Ngày lập", and "Họ tên khách hàng". At the bottom, there is a large table with columns "Mã hàng", "Tên hàng", "Đơn vị tính", "Đơn giá", "Số lượng", and "Thành tiền".

Số hóa đơn	Ngày lập	Họ tên khách hàng
*		

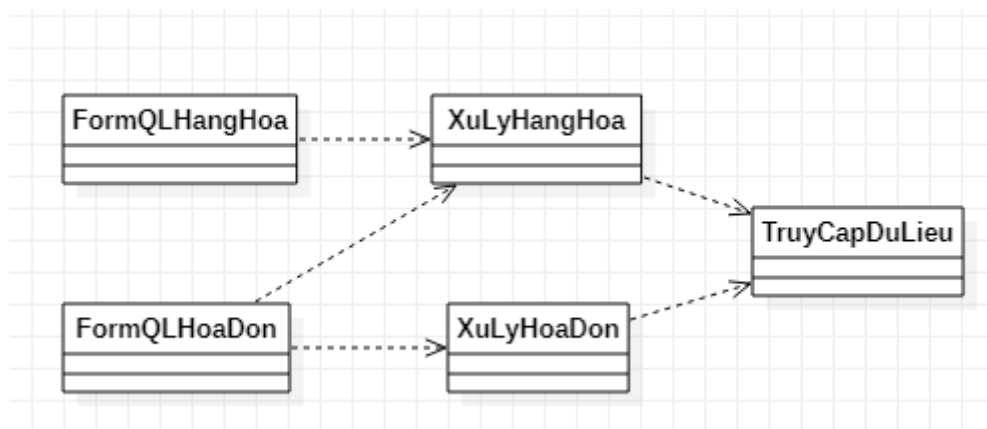
Mã hàng	Tên hàng	Đơn vị tính	Đơn giá	Số lượng	Thành tiền
*					

4.3. Giao diện chính của ứng dụng

The screenshot shows a Windows application window titled "Form Main". It has a menu bar with three options: "Quản lý hàng hóa" (Manage Goods), "Quản lý hóa đơn" (Manage Invoices), and "Ghi dữ liệu" (Save Data). The main area of the window is a large, empty gray rectangle.

5. Xây dựng ứng dụng trên 3-tier

5.1. Sơ đồ lớp



5.2. Chương trình mẫu

5.2.1. Lớp truy cập dữ liệu

```

[Serializable]
11 references
class TruyCapDuLieu
{
    private static TruyCapDuLieu instance = null;
    private List<HoaDon> dsHoaDon;
    private List<HangHoa> dsHangHoa;
    1 reference
    private TruyCapDuLieu()...
    2 references
    public static TruyCapDuLieu khaiTao()...
    2 references
    public List<HangHoa> getDanhSachHangHoa()...
    1 reference
    public List<HoaDon> getDanhSachHoaDon()...
    1 reference
    public static bool docFile(string tenFile)...
    1 reference
    public static bool ghiFile(string tenFile)...
}
    
```

5.2.2. Lớp Hàng hóa

```
[Serializable]
25 references
class HangHoa
{
    private string _maHang;
    private string _tenHang;
    private string _donViTinh;
    private double _donGia;
    5 references
    public string maHang...
    1 reference
    public string tenHang...
    1 reference
    public string donViTinh...
    1 reference
    public double donGia...
    1 reference
    public HangHoa()...
    2 references
    public HangHoa(string ma,string ten,string donViTinh,double donGia)...
```

5.2.3. Lớp hóa đơn

```
[Serializable]
12 references
class HoaDon
{
    private string _soHoaDon;
    private DateTime _ngayLap;
    private string _hoTenKhachHang;
    private List<ChiTietHoaDon> _chiTietHoaDon;
    1 reference
    public string soHoaDon...
    1 reference
    public DateTime ngayLap...
    1 reference
    public string hoTenKhachHang...
    0 references
    public List<ChiTietHoaDon> chiTietHoaDon...
    0 references
    public HoaDon()...
    1 reference
    public HoaDon(string soHD,DateTime ngayLap,string khachHang)...
```

5.2.4. Lớp chi tiết hóa đơn

[Serializable]

14 references

class ChiTietHoaDon

```

{
    private int _soLuong;
    private double _donGia;
    private HangHoa _hangHoa;
    2 references
    public int soLuong[...]
    0 references
    public double donGia[...]
    5 references
    public HangHoa hangHoa[...]
    0 references
    public ChiTietHoaDon()[...]
    1 reference
    public ChiTietHoaDon(int soLuong, double donGia, HangHoa hangHoa) [...]
    1 reference
    public double tinhThanhTien() [...]
}

```

5.2.5. Lớp Xử lý hàng hóa

5 references

class XuLyHangHoa

```

{
    private List<HangHoa> dsHangHoa;
    2 references
    public XuLyHangHoa() [...]
    4 references
    public List<HangHoa> getDanhSachHangHoa() [...]
    1 reference
    public void them(HangHoa hh) [...]
    1 reference
    public void sua(HangHoa hh) [...]
    1 reference
    public void xoa(string maHang) [...]
}

```

5.2.6. Lớp xử lý hóa đơn

3 references

```
class XuLyHoaDon
{
    private List<HoaDon> dsHoaDon = null;
    private List<ChiTietHoaDon> dsChiTietHoaDon = null;
    private List<HangHoa> dsHangHoa = null;
    1 reference
    public XuLyHoaDon()...
    1 reference
    public List<HoaDon> getDanhSachHoaDon()...
    2 references
    public List<ChiTietHoaDon> getDanhSachChiTietHoaDon()...
    2 references
    public List<HangHoa> getDanhSachHangHoa()...
    1 reference
    public void them(HoaDon hd)...
    1 reference
    public void them(ChiTietHoaDon chiTietHD)...
}
```

5.2.7. Giao diện quản lý hàng hóa

```
public partial class frmHangHoa : Form
```

```
{
    private XuLyHangHoa xuLyHang;
    1 reference
    public frmHangHoa()...
    1 reference
    private void frmHangHoa_Load(object sender, EventArgs e)...
    1 reference
    private void btnThem_Click(object sender, EventArgs e)...
    1 reference
    private void btnSua_Click(object sender, EventArgs e)...
    1 reference
    private void btnXoa_Click(object sender, EventArgs e)...
    4 references
    private void hienThiDanhSachHangHoa(DataGridView dgv, List<HangHoa> ds)...
    1 reference
    private void dgvDanhSachHangHoa_CellContentClick(object sender, DataGridViewCellEventArgs e)...
}
```

5.2.8. Giao diện quản lý hóa đơn

```

public partial class frmHoaDon : Form
{
    private XuLyHoaDon xuLyHoaDon;
    private XuLyHangHoa xuLyHang;
    1 reference
    public frmHoaDon()...
    1 reference
    private void frmHoaDon_Load(object sender, EventArgs e)...
    1 reference
    private void btnXemHoaDon_Click(object sender, EventArgs e)...
    1 reference
    private void btnChonHangHoa_Click(object sender, EventArgs e)...
    1 reference
    private void hienThiDanhSachChiTietHoaDon(List<ChiTietHoaDon> ds)...
    1 reference
    private void btnLapHoaDon_Click(object sender, EventArgs e)...
    1 reference
    private void khoiTaoDanhDachHangVaoComboboxTenHang(ComboBox cbo)...
    1 reference
    private void cboTenHang_SelectedIndexChanged(object sender, EventArgs e)...
}

```

5.2.9. Giao diện chính

```

public partial class FormMain : Form
{
    1 reference
    public FormMain()...
    1 reference
    private void FormMain_Load(object sender, EventArgs e)...
    1 reference
    private void ghiToolStripMenuItem_Click(object sender, EventArgs e)...
    1 reference
    private void hanghoaToolStripMenuItem_Click(object sender, EventArgs e)...
    1 reference
    private void hoadonToolStripMenuItem_Click(object sender, EventArgs e)...
}

```

6. Yêu cầu

6.1. Cài đặt Cấu trúc dữ liệu cho ứng dụng bao gồm các lớp:

1. Lớp Hàng hóa
2. Lớp Chi tiết hóa đơn
3. Lớp Hóa đơn

6.2. Cài đặt lớp Truy cập dữ liệu

6.3. Chức năng quản lý hàng hóa thực hiện các tác vụ như: hiển thị danh sách hàng hóa. Thêm, xóa và sửa thông tin hàng hóa. Cài đặt các lớp cho chức năng quản lý hàng hóa bao gồm:

1. Lớp truy cập dữ liệu (đã cài đặt trong mục 6.2)
2. Lớp xử lý hàng hóa
3. Lớp giao diện quản lý hàng hóa

6.4. Chức năng quản lý hóa đơn thực hiện các tác vụ như: chọn hàng hóa đưa vào hóa đơn, lập hóa đơn, xem danh sách hoá đơn và xem thông tin chi tiết của hóa đơn. Cài đặt các lớp cho chức năng quản lý hóa đơn bao gồm:

1. Lớp truy cập dữ liệu (đã cài đặt trong mục 6.2)
2. Lớp xử lý hóa đơn
3. Lớp giao diện quản lý hóa đơn

6.5. Cài đặt lớp giao diện chính bao gồm các thành phần: Quản lý hàng hóa, Quản lý hóa đơn và ghi dữ liệu lên file.

7. Bài tập làm thêm

7.1. Xây dựng lớp **LopHoc**, mỗi đối tượng tượng trưng cho 1 lớp học gồm các thông tin : mã lớp, khoa, hệ đào tạo. Hãy viết các thuộc tính và phương thức khởi tạo cho lớp LopHoc.

-

LopHoc
+sMalop +sKhoa +sHeDaoTao +LopHoc()

7.2. SV hãy thiết kế form như sau:

	Mã lớp	Khoa	Hệ đào tạo
▶	D20_TH01	CNTT	DH
	D20_TH02	CNTT	DH

Trong ứng dụng có:

- Ba TextBox hiển thị thông tin lớp học gồm mã lớp, khoa, hệ đào tạo.
- Ba Button Thêm/Xóa/Sửa.
- Một DataGridView để hiển thị danh sách lớp học.
- Dùng đối tượng thuộc lớp List<T> để lưu danh sách lớp học.
- Dữ liệu danh sách các lớp học được lưu và đọc từ file bằng kỹ thuật Serialize.
- Ứng dụng được xây dựng theo mô hình 3-tầng (3-tier).

7.3. SV thiết kế form chính là MDI Form như sau:

Danh mục Type Here

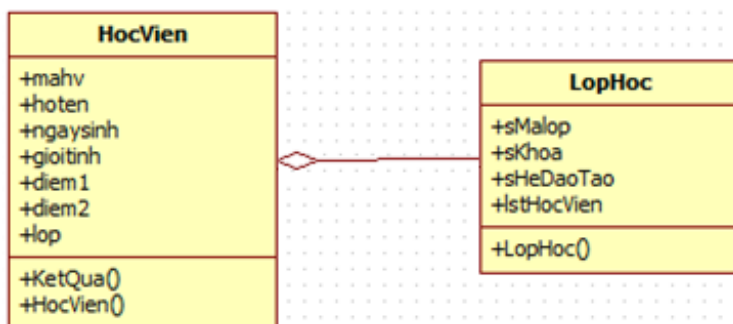
- Lớp học
- Học viên
- Thoát

Type Here

- Khi nhấn vào menu “Lớp học” sẽ mở form “Quản lý lớp học” như phần 7.2.
- Khi nhấn vào menu “Học viên” sẽ mở form “Quản lý học viên” như sau:

- SV hãy viết lớp HocVien như các bài thực hành trước và xây dựng form “Quản lý học viên” theo mô hình 3-tầng (3-tier) tương tự phần 7.2. Lưu ý: dùng control DataGridView để hiển thị danh sách học viên.

7.4. SV tạo mối quan hệ giữa 2 lớp LopHoc và HocVien như sau (một đối tượng học viên chứa một đối tượng lớp học):



Trong đó:

- Lớp LopHoc :


```
[Serializable]
18 references
class LopHoc
{
    private string sMalop;
    private string sKhoa;
    private string sHeDaoTao;

    "Properties và Phương thức khởi tạo"
}
```

- Lớp HocVien có 1 thuộc tính “lop” có kiểu LopHoc là lớp học mà học viên đó đang học:

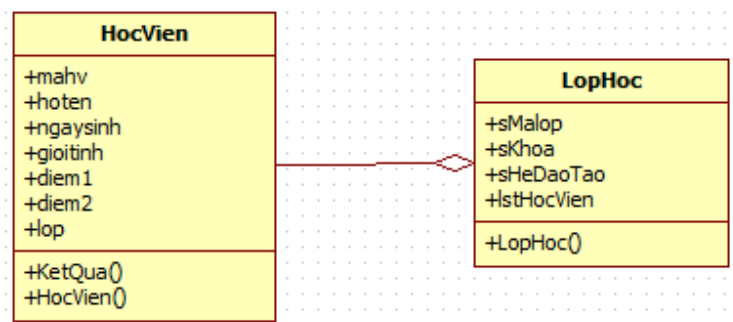
```
[Serializable]
3 references
class HocVien
{
    private string mahv;
    private string hoten;
    private DateTime ngaysinh = new DateTime();
    private bool gioitinh;
    private float diem1;
    private float diem2;
    private LopHoc lop;

    "Properties và Phương thức khởi tạo"
}
```

- Thiết kế lại form “Quản lý học viên” như sau:

Yêu cầu: Tất cả các mã lớp sẽ được hiển thị trong ComboBox “Lớp”.

7.5. SV tạo mối quan hệ giữa 2 lớp LopHoc và HocVien như sau (một đối tượng lớp học chứa nhiều đối tượng học viên):



Trong đó:

- Lớp LopHoc có thuộc tính lstHocVien chứa danh sách học viên của lớp tương ứng:

```
[Serializable]
17 references
class LopHoc
{
    private string sMalop;
    private string sKhoa;
    private string sHeDaoTao;
    private List<HocVien> lstHocVien;
    "Properties và Phương thức khởi tạo"
}
```

- Lớp HocVien có 1 thuộc tính “lop” có kiểu LopHoc là lớp học mà học viên đó đang học:

```
[Serializable]
3 references
class HocVien
{
    private string mahv;
    private string hoten;
    private DateTime ngaysinh = new DateTime();
    private bool gioitinh;
    private float diem1;
    private float diem2;
    private LopHoc lop;
    "Properties và Phương thức khởi tạo"
}
```

- Thiết kế lại form “Quản lý lớp học” như sau:

Trong đó :

- DataGridView “Danh sách lớp học” : sẽ hiển thị danh sách tất cả các lớp học.
- DataGridView “Danh sách học viên trong lớp” : sẽ hiển thị danh sách tất cả các học viên thuộc lớp đang hiển thị.