Lập trình cấu trúc

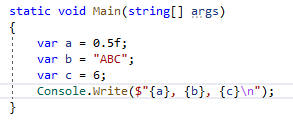
# Kiểu dữ liệu và biến

## Kiểu dữ liệu

|  |  |  |  |
| --- | --- | --- | --- |
| **Nhóm** | **Kiểu dữ liệu** | **Kích thước**  **(bytes)** | **Ý nghĩa** |
| Kiểu số nguyên | byte | 1 | Số nguyên dương**không dấu** có giá trị từ **0** đến **255** |
| sbyte | 1 | Số nguyên có dấu có giá trị từ**-128**đến **127** |
| short | 2 | Số nguyên có dấu có giá trị từ**-32,768**đến **32,767** |
| ushort | 2 | Số nguyên **không dấu**có giá trị từ **0** đến **65,535** |
| int | 4 | Số nguyên có dấu có giá trị từ **-2,147,483,647** đến **2,147,483,647** |
| uint | 4 | Số nguyên**không dấu**có giá trị từ **0** đến **4,294,967,295** |
| long | 8 | Số nguyên có dấu có giá trị từ -9**,223,370,036,854,775,808** đến  **9,223,370,036,854,775,807** |
| ulong | 8 | Số nguyên **không dấu**có giá trị từ **0**đến **18,446,74**4,0**73,709,551,615** |
| Kiểu ký tự | char | 2 | Chứa một ký tự Unicode |
| Kiểu logic | bool | 1 | Chứa 1 trong 2 giá trị logic là **true**hoặc **false** |
| Kiểu số thực | float | 4 | Kiểu số thực dấu chấm động có giá trị dao động  từ **3.4E – 38**đến **3.4E + 38**, với 7 chữ số có nghĩa |
| double | 8 | Kiểu số thực dấu chấm động có giá trị dao động từ**1.7E – 308**đến**1.7E + 308**, với 15, 16 chữ số có nghĩa |
| decimal | 8 | Có độ chính xác đến 28 con số và giá trị thập phân, được dùng trong tính toán tài chính |

### Kiểu var

* Là kiểu mà ta không cần phải khai báo kiểu dữ liệu thông thường như int, float, string,… mà var sẽ tự ép kiểu cho biến của nó.
* Kiểu var giống như cách khai báo biến trong ngôn ngữ PHP
* Nếu dùng var ta bắt buộc phải gán giá trị cho biến ngay khi ta khởi tạo.



* Không nên lạm dụng var vì sẽ làm cho code khó hiểu. Ta phải hạn chế dùng var tới mức tối đa
* Áp dụng trong trường hợp ta không biết trước kiểu của biến đó.

## Biến

* Cú pháp: // Type a = Value;

namespace HelloCsharp

{

class Program

{

int a = 20;

static void Main(string[] args)

{

int b = 16;

}

}

}

* Biến cục bộ: là biến **b** trên ví dụ. Với biến cục bộ ta phải gán giá trị cho nó. Nhưng nếu muốn khai báo biến cục bộ là rỗng thì ta khai báo giá trị default



* Biến toàn cục: là biến **a** trên ví dụ. Biến toàn cục có thể không cần gán giá trị.
* VD:

namespace HelloCsharp

{

class Program

{

static int a = 25, b=10;

static void Main(string[] args)

{

int sum = a + b;

float val = (float)a / b;

Console.WriteLine("a = "+a+", b ="+b);

Console.WriteLine("a+b="+sum+", a/b="+val);

}

}

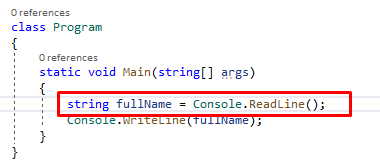
}

## Hằng số

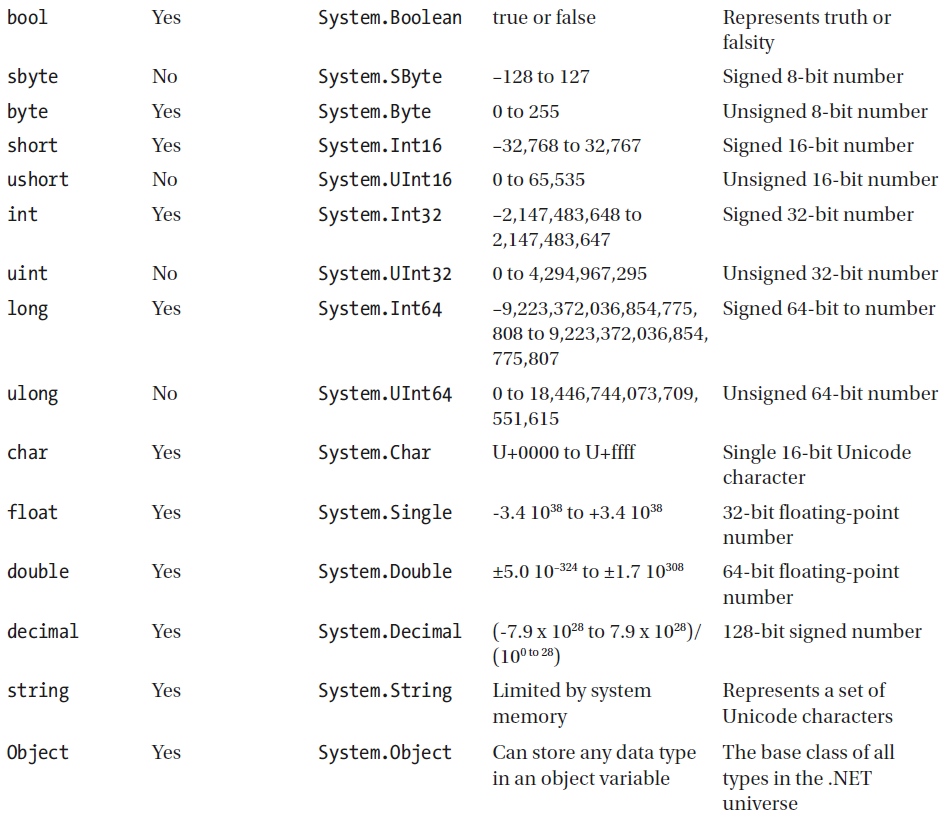
* Là biến biết trước giá trị mà biến đó không được phép thay đổi
* Cú pháp khai báo: public const float PI = 3.14f;
* Cách dùng: Program.PI => (Class name).(const)

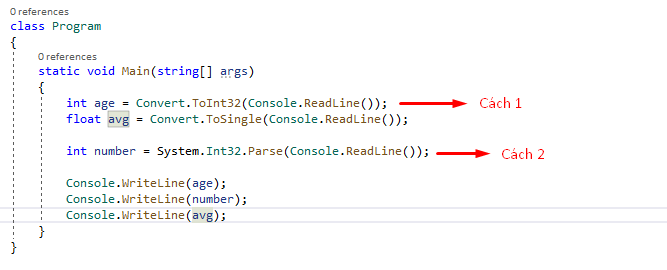
## Nhập dữ liệu vào từ bàn phím

### Nhập một chuỗi String

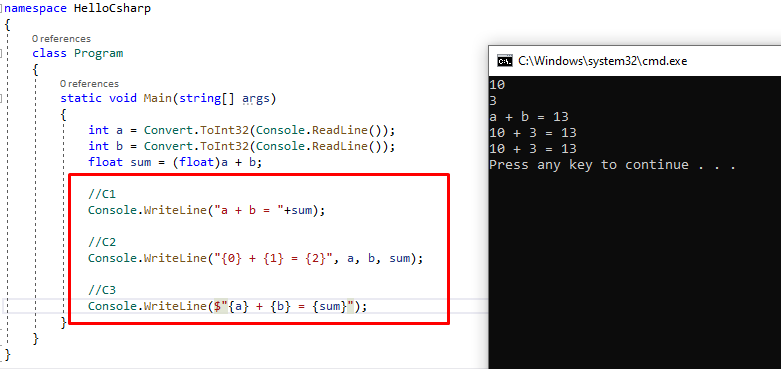


### Nhập một số





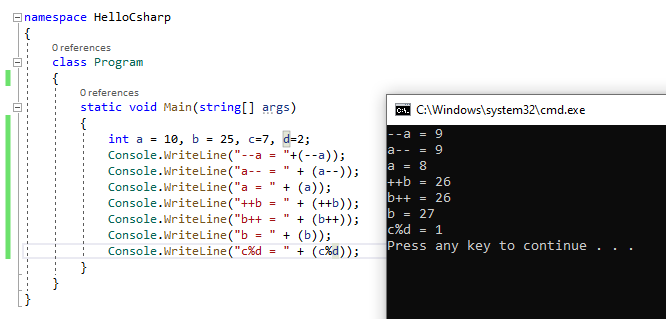
## Hiển thị dữ liệu ra màn hình



# Toán tử

## Toán tử số học

* (+, -, \*, /, %, ++, --)
* BT:



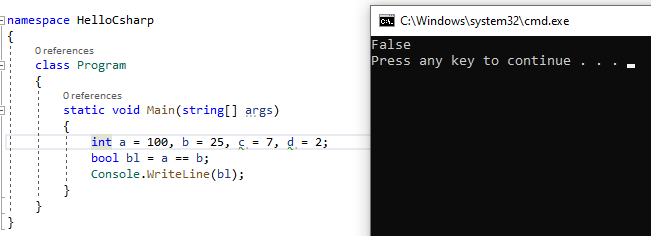
## Toán tử gán

* (=, +=, -=, \*=, /=, %=)
* VD:



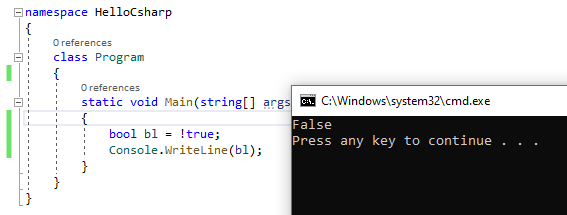
## Toán tử so sánh

* (==, !=, >, <, >=, <=)



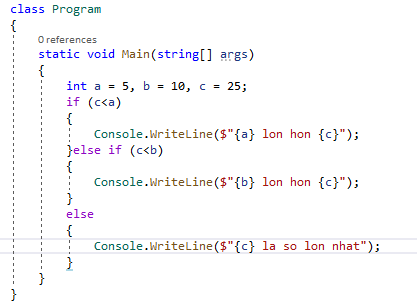
## Toán tử logic

* (!, &&, ||)

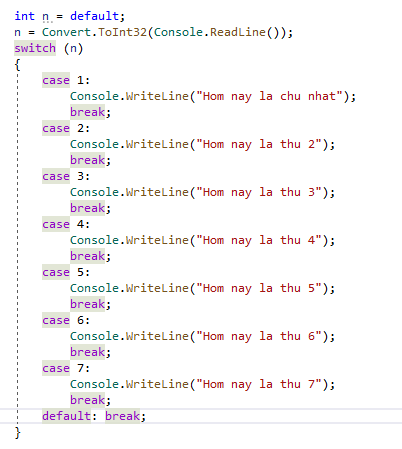


# Cấu trúc rẽ nhánh

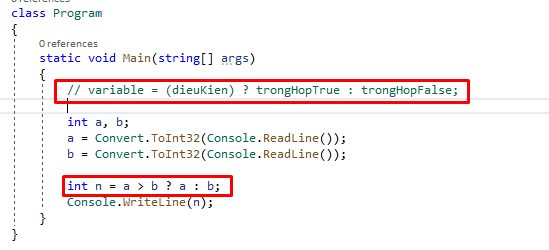
## If-else



## Switch-case

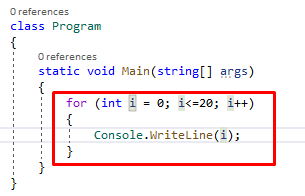


## Toán tử 3 ngôi

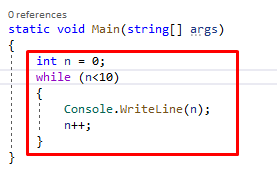


# Vòng lặp (Loop)

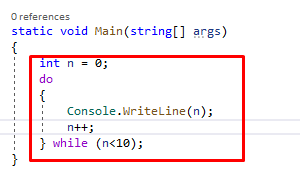
## For



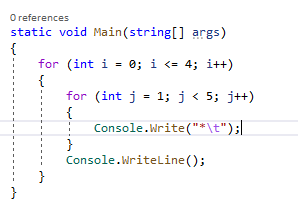
## While



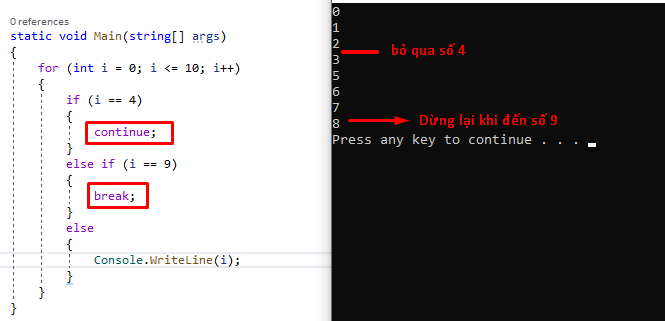
## Do-while



## Vòng lặp lồng nhau



## Break, Continue



# Mảng

## Mảng 1 chiều

### Khai báo mảng

* Cú pháp: int[] arrays;
* Cấp phát mảng có 200 phần tử: int[] arrays = new int[200];
* Số lượng phần tử mảng không thể thay đổi khi đã cấp phát.
* Cú pháp khởi tạo giá trị 1: int[] arrays = { 1, 2, 3 };
* Cú pháp khởi tạo giá trị 2: int[] arrays = new int[]{ 1, 2, 3 };
* Cú pháp khởi tạo giá trị 3: int[] arrays = new int[3]{ 1, 2, 3 };
* Cú pháp xem độ dài mảng: arrays.Length
* Cú pháp gọi một phần tử của mảng: arrays[1] => Kq: 2
* Cú pháp thay đổi giá trị trong mảng: arrays[1] = 5; => Kq: 2 thay bằng 5

### Duyệt mảng

#### For

* Với cách duyệt mảng này ta có thể duyệt mảng hoặc thay đổi giá trị của mảng

for (int i = 0; i<arrays.Length;i++)

{

Console.Write($"{arrays[i]}\t");

}

#### Vòng lặp Foreach

* Với cách duyệt mảng này ta chỉ có thể duyệt mảng mà thôi.

foreach (var array in arrays)

{

Console.WriteLine(array);

}

### Các phương thức lớp aray

#### Cú pháp sắp xếp mảng

* Sắp xếp tăng: Array.Sort(arrays);
* Sắp xếp giảm: Array.Reverse(arrays);

#### Reset các phần tử mảng về giá trị mặc định

* Array.Clear(names, 0, names.Length);

#### Tìm kiếm giá trị trong mảng

* int indexOf = Array.IndexOf(names, "Lan");

## Mảng 2 chiều

### Khai báo mảng

* Cú pháp khai bảo mảng vuông: int[,] arrays;
* Cú pháp khai bảo mảng zikzak: int[][] number;
* Cú pháp cấp phát mảng vuông: int[,] arrays = new int[10,8];
* Cú pháp cấp phát mảng zikzak:

int[][] number = new int[6][];

for (int i = 0; i < number.Length; i++)

{

number[i] = new int[i+1];

}

### Duyệt mảng

* Duyệt mảng zikzak:

for (int i = 0; i < number.Length; i++)

{

for (int j = 0; j < number[i].Length; j++)

{

Console.Write($"{i},{j}\t");

}

Console.WriteLine();

}

* Duyệt mảng vuông:

for (int i = 0; i < arrays.GetLength(0); i++)

{

for (int j = 0; j < arrays.GetLength(1); j++)

{

Console.Write($"{i},{j}\t");

}

Console.WriteLine();

}

* GetLength(0): Chiều dài cột

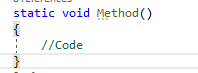
GetLength(1): Chiều dài hàng

## Mảng động – List

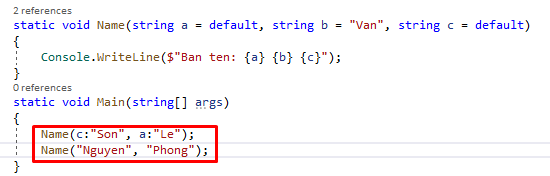
* Gọi thư viện: using System.Collections.Generic;
* Khai báo list: List<int> list = new List<int>();
* Xem độ dài của list: list.Count
* Add dữ liệu vào mảng: list.Add(10)
* Lấy dữ liệu theo index = 1: list[1]

# Phương thức – Method

* Làm cho code được tái sử dụng lại
* Tiết kiệm được việc viết nhiều đoạn code trùng lặp nhau.
* Cú pháp:



* Các loại phương thức:
  + Phương thức không có tham số
  + Phương thức có tham số truyền vào



* Nạp chồng phương thức là các phương thức có cùng tên nhưng khác nhau về kiểu trả về và bộ tham số truyền vào.

## Từ khóa Out, Ref

### Out

* Không cần khởi tạo giá trị ban đầu cho biến.
* Bắt buộc phải gán giá trị cho biến được sử dụng trong thân hàm. Nếu không gán giá trị thì nó sẽ bị lỗi.
* Từ khóa out dùng khi ta muốn truyền giá trị của biến từ một hàm riêng biệt và sử dụng hàm đó trong các hàm khác.

static void Main(string[] args)

{

string name;

int age;

float mark;

Infomation(out name, out age, out mark);

ShowInfo(name, age, mark);

}

// Alt + Enter

private static void ShowInfo(string name, int age, float mark)

{

Console.WriteLine("Ho va ten: "+name);

Console.WriteLine("Tuoi: "+age);

Console.WriteLine("Diem so: "+mark);

}

static void Infomation(out string name, out int age, out float mark)

{

Console.WriteLine("Nhap ten: ");

name = Console.ReadLine();

Console.WriteLine("Nhap tuoi: ");

age = Convert.ToInt32(Console.ReadLine());

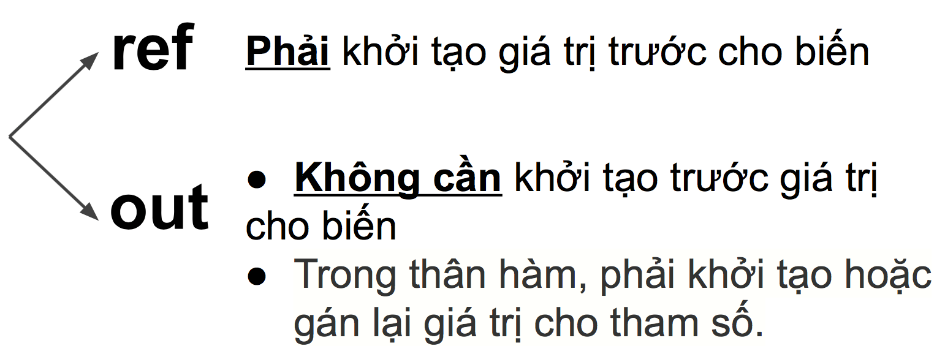
Console.WriteLine("Nhap diem: ");

mark = Convert.ToSingle(Console.ReadLine());

}

### Ref

* Cách gọi và công dụng giống như từ khóa out.
* Khi sử dụng từ khóa ref bắt buộc phải khởi tạo giá trị ban đầu cho biến.
* Không cần gán giá trị cho biến được sử dụng trong thân hàm.



## Từ khóa Params

* Khi ta có các biến cùng một kiểu và muốn truyền nhiều biến vào trong hàm thì ta dùng từ khóa params.
* Các tham số truyền vào có thể là mảng 1 chiều, các biến số cụ thể.

static void Main(string[] args)

{

float [] mark = {10, 20, 30, 40 };

Infomation(mark);

Infomation(15, 25, 35, 45);

}

// Alt + Enter

static void Infomation(params float [] marks)

{

for (int i = 0; i < marks.Length; i++)

{

Console.Write(marks[i]+"\t");

}

}

Lập trình hướng đối tượng

# Thuật ngữ cơ bản

## Lớp

class Program

{

//code

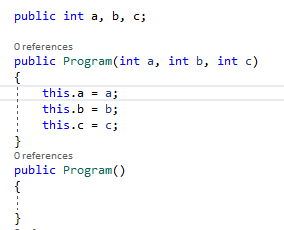
}

## Đối tượng

Cat cat = new Cat();

## Contructor

* Khởi tạo giá trị ban đầu cho đối tượng.
* Contructor nó không có kiểu trả về
* Tên trùng với tên của lớp.
* Trong mỗi lớp luôn có một contructor mặc định nhằm giúp ta có thể gọi đối tượng của lớp đó.
* Khi ta tạo một contructor mới thì contructor mặc định sẽ bị ẩn đi.

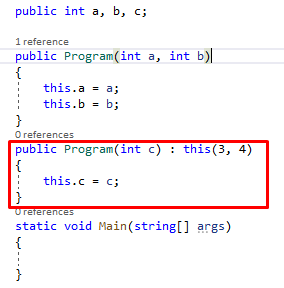


* Viết tắt nhanh: ctor > tab > tab

### Từ khóa this

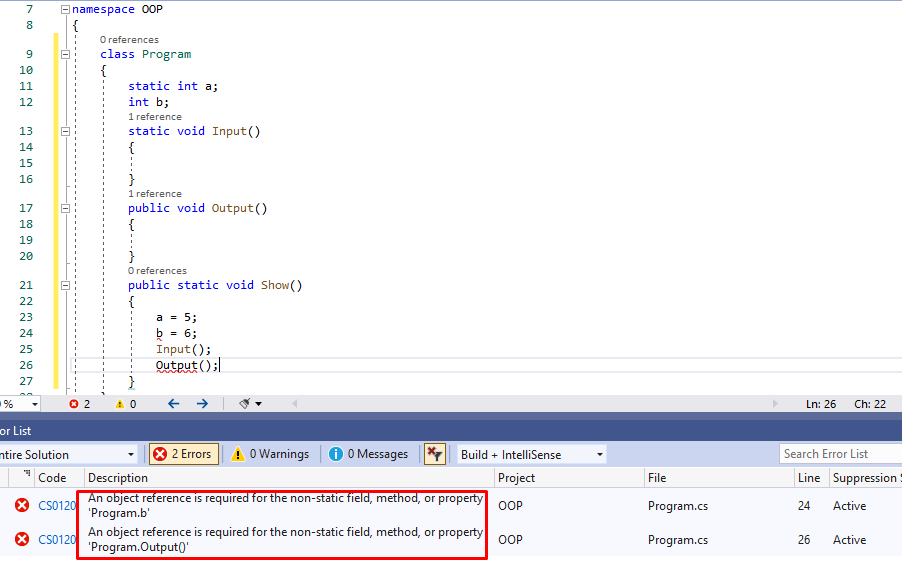
* This đại diện cho đối tượng hiện thời của lớp đó.
* Trường hợp sử dụng this khác. Khi chúng ta gọi contructor vào một contructor khác.

VD:



### Thành phần Static

* Static được sử dụng cho lớp, trường dữ liệu, phương thức,..
* Những thành phần static chỉ tương tác với những thành phần static khác.



### Static Contructor

* Tương tự hàm contructor nhưng chỉ sử dụng trong các thành phần static.
* Static contructor không overload
* Không chứa access modifier
* Không nhận tham số
* Chỉ được thực thi duy nhất một làn cho dù có tạo bao nhiêu đối tượng đi nữa.
* Ứng dụng cho việc khởi tạo database

### Lớp static và import static

* Lớp Math là một ví dụ cho lớp static
* Để import các thành phần static thì ta dùng: using static System.Math;
* Chúng ta không nên lạm dụng import vì nó sẽ làm code bị rối.

## Access Modifier

### Public

* Không giới hạn truy cập

### Private

* Chỉ được sử dụng trong class chứa nó

### Protected

* Chỉ được sử dụng trong class chứa nó và các class con của nó

# Tính đóng gói

* Dữ liệu của đối tượng thì không nên được truy cập trực tiếp bên ngoài lớp bởi chính bản thân nó mà việc này cần thực hiện gián tiếp thông qua các thành phần public khác.

## Getter và Setter

public void SetAge(int age)

{

if (age < 0 || age > 200)

{

Console.WriteLine("Tuoi khong hop le");

}

Else

{

this.age = age;

}

}

public int GetAge()

{

return this.age;

}

## .Net properties

public int Age

{

get { return age; }

set { if (value<0||value>200) {

Console.WriteLine("Khong hop le.!");

}

Else

{

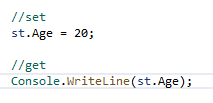
age = value;

}

}

}

* Thay vì ta phải viết getter và setter thì ta chỉ cần viết 1 properties Age. Nó sẽ làm code bạn dễ hiểu và rõ ràng, không rời rạc.
* Sử dụng properties:



* Ta có thể thay get { return age; } thành get => age;. Như vậy trong hàm contructor thay vì gọi trực tiếp đến thuộc tính this.age = age;. Thì ta chỉ cần gọi Age = age;. Tức là gọi đến properties Age cho nó xử lý data truyền vào thuộc tính.
* Phím tắt khi sử dụng trong VS là: propfull

## Read only, Write only, Automatic properties

* Read only: Chỉ đọc 1 lần gống như đọc số chứng minh thư
* Write only: Chỉ viết 1 lần giống như nhập mật khẩu
* Automatic properties: Bắt buộc phải có khối get

### Read only

public string EmtyID

{

get => emtyID;

}

### Write only

public string CkeckPass

{

Set

{

password = value;

}

}

### Automatic properties

* Khai báo: public string Email { get; set; } = "ABC@gmail.com";
* Sử dụng:



# Tính kế thừa

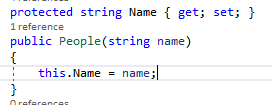
* Tính kế thừa cho phép lập trình viên tạo ra một lớp mới kế thừa một lớp đã tồn tại. Nghĩa là lớp được kế thừa sẽ có tất cả những thuộc tính và phương thức với quyền truy cập không phải private của lớp cha.
* Mục đích của tính kề thừa là để tái sử dụng những thuộc tính và phương thức phổ biến trong một lớp mà không cần phải tạo ra chúng

## Kế thừa lớp

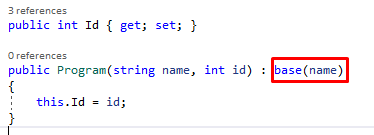


## Kế thừa phương thức khởi tạo

* Trong lớp cha:

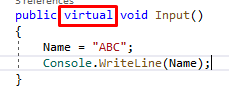


* Trong lớp con kế thừa từ lớp cha:

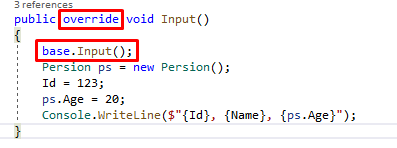


## Ghi đè phương thức

* Trong lớp cha:

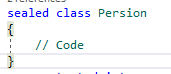


* Trong lớp con kế thừa từ lớp cha:



## Từ khóa Sealed

* Một lớp được chỉ định với từ khoá sealed là một lớp không cho phép kế thừa. Một phương thức được chỉ định với từ khoá sealed là một phương thức không cho overriding.
* Sealed class:

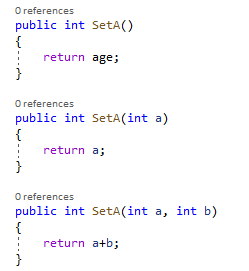


* Sealed method sử dụng ở lớp con:

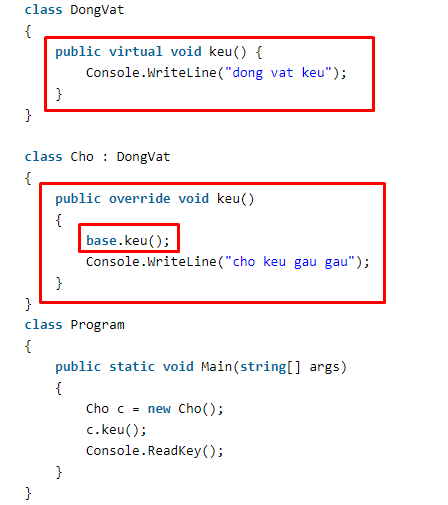


# Tính đa hình

## Overloading – Nạp chồng

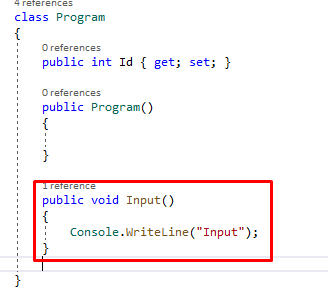


## Overriding – Ghi đè

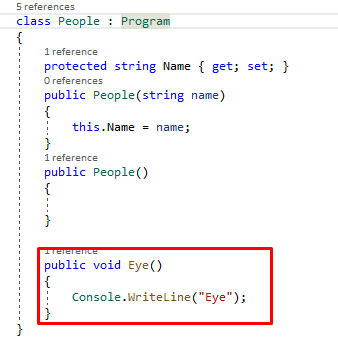


## Tính đa hình của đối tượng

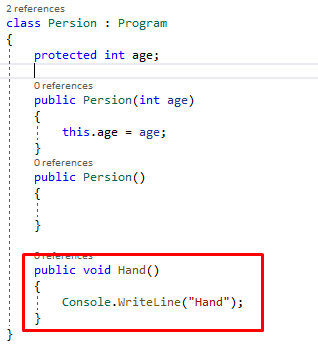
* Tại lớp cha:



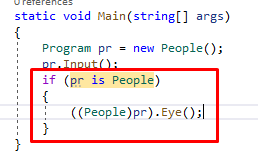
* Tại lớp con thứ nhất:



* Tại lớp con thứ hai:



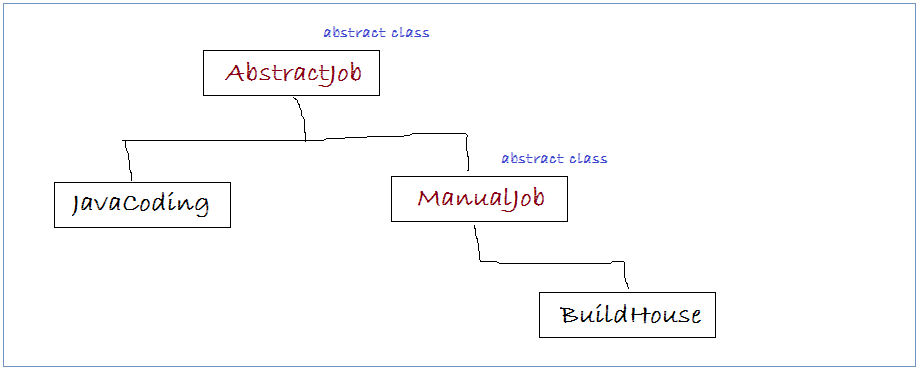
* Sử dụng:



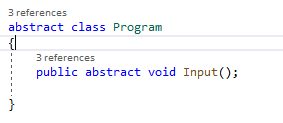
# Tính trừu tượng

## Abstract Class

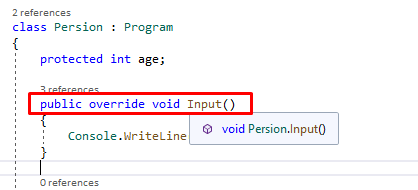
* Nó được khai báo với từ khóa abstract.
* Nó có thể khai báo 0, 1 hoặc nhiều method trừu tượng bên trong.
* Bạn không thể khởi tạo 1 đối tượng trực tiếp từ một class trừu tượng.
* Khi 1 lớp kế thừa từ abstract class thì nó bắt buộc phải khai báo method có trong abstract class đó.



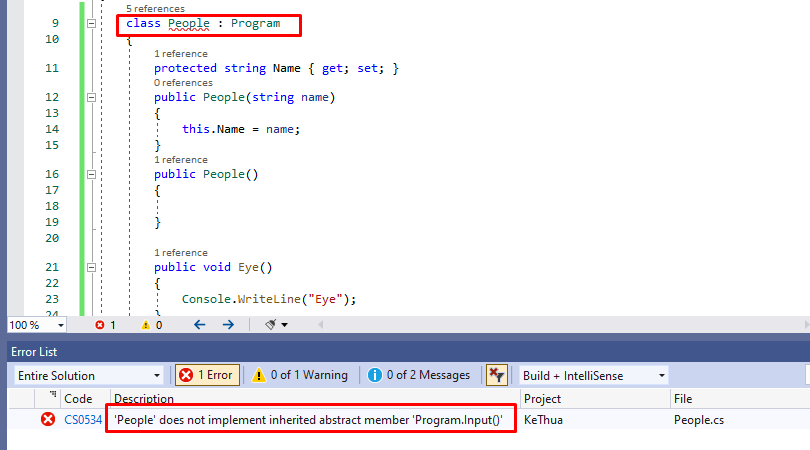
* Abstract class:



* Class con 1:

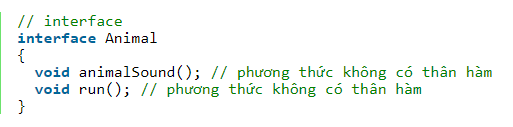


* Class con 2:

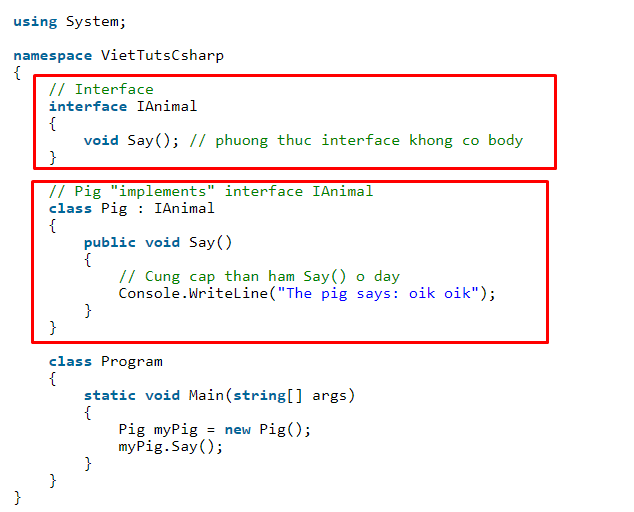


## Interface

* Một Interface trong C# là một bản thiết kế của một lớp. Nó chỉ có các phương thức và thuộc tính trừu tượng. Interface là một kỹ thuật để thu được tính trừu tượng hoàn toàn và đa kế thừa trong C#.



VD:



* Implement nhiều Interface:

