

C# 6 – Lớp và Phương thức

Giảng viên: **ThS. Lê Thiện Nhật Quang**

Email: quangln.dotnet.vn@gmail.com

Website: <http://dotnet.edu.vn>

Điện thoại: **0868.917.786**



MỤC TIÊU

- Giải thích class và object
- Tìm hiểu các phương thức - methods
- Danh sách các chỉ định truy cập – access modifiers
- Giải thích phương thức nạp chồng – overloading
- Tìm hiểu hàm dựng – constructor và hàm hủy - destructor

1.1. LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

- ❑ Lập trình hướng đối tượng (Object-oriented programming) là kỹ thuật lập trình được thiết kế để giải quyết các chương trình bằng cách tạo ra các đối tượng mô phỏng các thực thể bên ngoài của vấn đề cần giải quyết.
- ❑ Lập trình hướng đối tượng sẽ định nghĩa các đối tượng bao gồm thuộc tính (data) và phương thức (thao tác với dữ liệu).
- ❑ Điểm mạnh của lập trình hướng đối tượng là tái sử dụng lại code, cung cấp một cấu trúc các module của chương trình một cách rõ ràng, che dấu được dữ liệu bên trong, giúp lập trình viên đơn giản hóa độ phức tạp của bài toán.

1.2. TÍNH CHẤT

Tính trừu tượng Abstraction

Là khả năng ẩn chi tiết triển khai, chỉ cung cấp thông tin tính năng tới người dùng. Tính trừu tượng được thể hiện qua abstract class và interface

Tính Bao đóng Encapsulation

Là khả năng che giấu thông tin của đối tượng với môi trường bên ngoài. Việc cho phép môi trường bên ngoài tác động lên các dữ liệu nội tại của đối tượng hoàn toàn tùy thuộc vào người viết mã.

Tính thừa kế Inheritance

Là khả năng tạo một lớp mới dựa trên một lớp có sẵn. Lớp có sẵn là lớp cha (hoặc lớp cơ sở), lớp mới là lớp con (hoặc lớp dẫn xuất) và lớp con thừa kế các thành viên được định nghĩa ở lớp cha.

Tính đa hình Polymorphism

Là khả năng một đối tượng có thể thực hiện một tác vụ theo nhiều cách khác nhau. Đa hình gồm 2 loại là đa hình tĩnh và đa hình động.

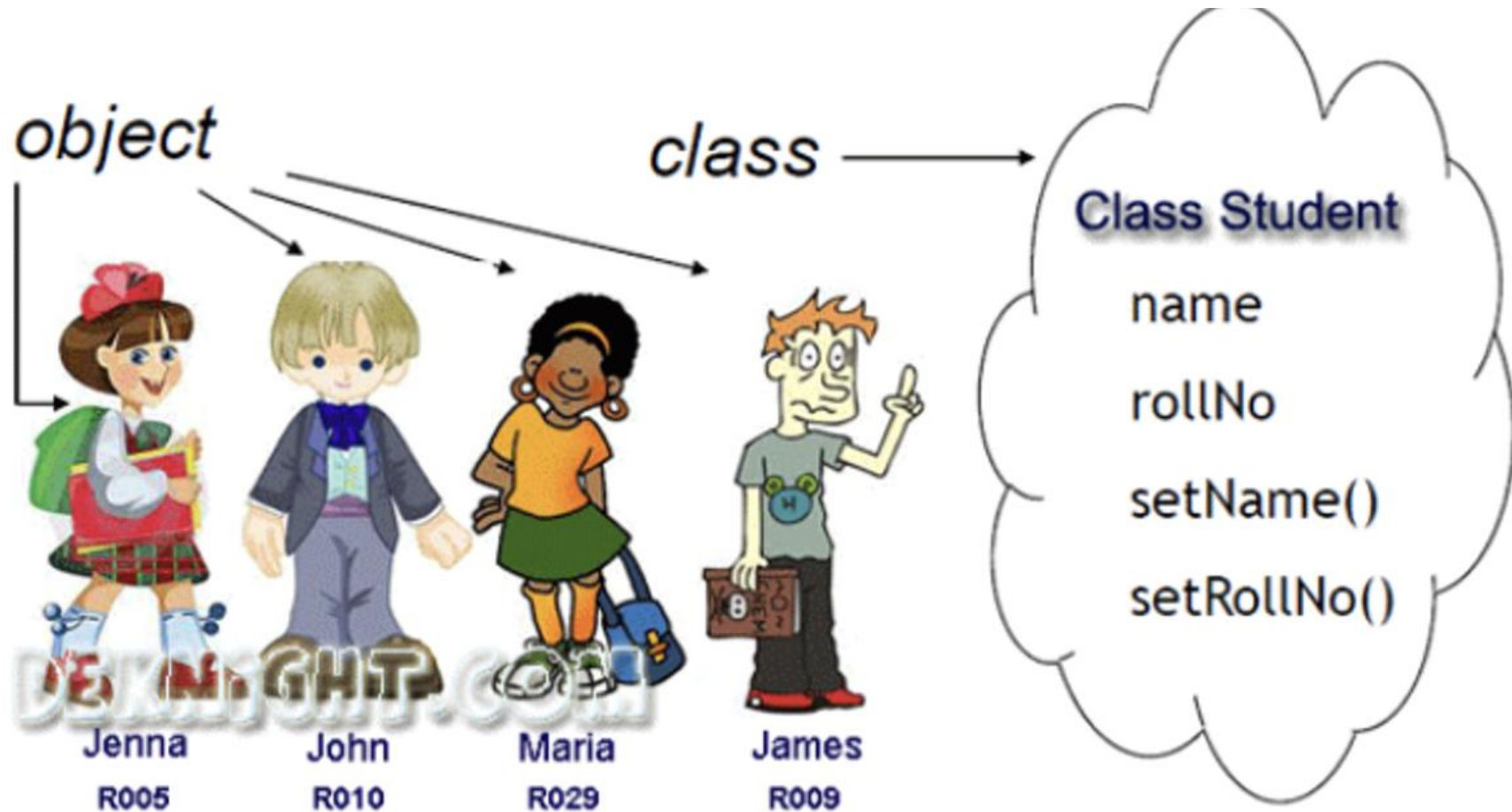
1.3. KHÁI NIỆM VỀ OBJECT – ĐỐI TƯỢNG

- ❑ Biểu diễn đối tượng trong thế giới thực
- ❑ Mỗi đối tượng được đặc trưng bởi các thuộc tính và các hành vi riêng của nó



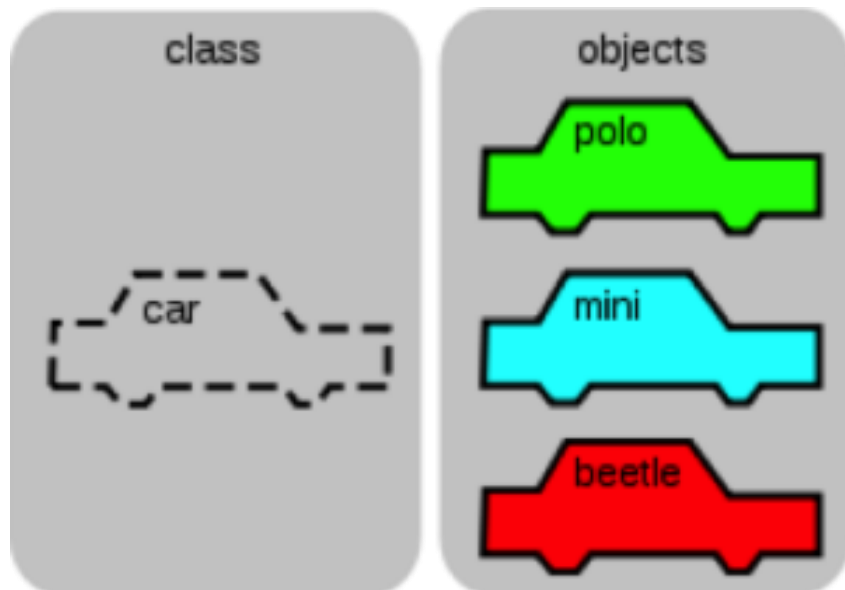
1.4. KHÁI NIỆM VỀ CLASS – LỚP

- ❑ Lớp là một khuôn mẫu được sử dụng để mô tả các đối tượng cùng loại.
- ❑ Lớp bao gồm các thuộc tính (trường dữ liệu) và các phương thức (hàm thành viên)



1.5. CLASS - OBJECT

- **Class** trong OOP định nghĩa trừu tượng các đặc tính của đối tượng, chúng ta hiểu nôm na giống như bản thiết kế hay khuôn mẫu của đối tượng nào đó, ví dụ như bản thiết kế của ô tô, bản vẽ của toà nhà, các control như TextBox, Button, Label.
- **Object** là một sản phẩm được tạo ra từ bản thiết kế của **Class**, ví dụ bản thiết kế ô tô đó được mang đi sản xuất, bản vẽ toà nhà được dùng để xây dựng, TextBox, Button ... được kéo vào Form thì lúc đó sản phẩm tạo từ các bản thiết kế đó gọi là **Object**.



1.6. TẠO CLASS

Khi tạo một class có nghĩa là bạn đang tạo một bản kế hoạch chi tiết cho một kiểu dữ liệu. Điều này thực sự không định nghĩa bất kỳ dữ liệu nào, nhưng nó xác định tên lớp có nghĩa là gì, các đặc tính (thuộc tính) và hành vi (phương thức) của lớp.

Đối tượng là các thể hiện của một lớp. Các phương thức và thuộc tính của lớp được gọi là thành viên của lớp.

Class nên bắt đầu với từ khoá “*Class*” và tiếp theo là tên của Class

Cú pháp

```
{  
//class members  
}
```



```
public class HìnhTron
{
    public double bankinh;
    0 references
    public double getChuvi()
    {
        return 2 * Math.PI * this.bankinh;
    }
    0 references
    public double getDientich()
    {
        return Math.PI * Math.Pow(this.bankinh,2);
    }
    0 references
    public void Xuat()
    {
        |
    }
}
```

- ✓ Lớp HìnhTron
- ✓ Trường bankinh
- ✓ 3 phương thức
 - getChuVi()
 - getDienTich()
 - Xuat()

1.7. HÌNH THỨC CHUNG CỦA CLASS

```
<access specifier> class ClassName {  
    // member properties  
    <access specifier> <data type> Property1 { get; set; }  
    <access specifier> <data type> Property2 { get; set; }  
  
    // constructors  
    <access specifier> ClassName(){  
        // constructor body  
    }  
    <access specifier> ClassName(parameter_list){  
        // constructor body  
    }  
  
    // member methods  
    <access specifier> <return type> Method1(parameter_list) {  
        // method body  
    }  
    <access specifier> <return type> Method2(parameter_list) {  
        // method body  
    }  
}
```

- **access specifier**: là chỉ thị truy cập xác định các quy tắc truy cập cho các thành viên cũng như chính lớp đó. Có các từ khóa chỉ thị truy cập sau: public, internal, protected, private. Nếu không được chỉ định thì chỉ thị truy cập mặc định cho lớp là internal. Còn chỉ thị truy cập mặc định cho các thành viên của lớp là private.
- **data type**: xác định kiểu dữ liệu của thuộc tính.
- **return type**: xác định kiểu dữ liệu trả về của phương thức, nếu phương thức không trả về dữ liệu thì sử dụng kiểu void.
- **constructor**: là phương thức khởi tạo của class. Mặc định khi bạn không khai báo constructor thì khi biên dịch hệ thống sẽ tự tạo một constructor mặc định - là constructor không có đối số.
- Tên lớp, thuộc tính, phương thức hợp lệ chỉ gồm ký tự a-z, A-Z, số 0-9 và ký tự _ và không được bắt đầu bằng số 0-9. Quy ước đặt tên trong C# bắt đầu bằng ký tự viết hoa A-Z.

2 references

`internal class Students``{` `/// <summary>` `/// This is Fields` `/// </summary>` `string _studName = "Jame Anderson";` `int _studAge = 27;` `/// <summary>` `/// This is Method` `/// </summary>`

1 reference

 `void Display()` `{` `Console.WriteLine("Student Name: "+ _studName);` `Console.WriteLine("Student Age: "+ _studAge);` `}` `/// <summary>` `/// This is Method` `/// </summary>` `/// <param name="args"></param>`

0 references

 `static void Main(string[] args)` `{` `Students objstudents = new Students();` `objstudents.Display();` `}``}`

1.8. HƯỚNG DẪN ĐẶT TÊN CLASS

- Thường dùng danh từ hoặc một cụm danh từ để đặt tên cho 1 class.
- Kí tự đầu tiên của mỗi từ là chữ in hoa.(Dùng quy tắc Pascal Case).
- Không sử dụng dạng tiền tố:

Ví dụ: **FileStream** là Tên chuẩn.

Không đặt tên theo dạng **CFileStream** hay **ClassFileStream**

- Trong tên class không sử dụng dấu gạch chân (_).
- Trong trường hợp cần thiết, nếu kí tự đầu tiên của tên 1 class bắt đầu bằng kí tự 'I' và kí tự này là kí tự bắt đầu của một từ và từ đó là 1 phần của tên class thì tên đó vẫn được chấp nhận.

Ví dụ: **IdentityStore**

- Sử dụng một từ ghép để đặt tên cho 1 class dẫn xuất, từ thứ 2 trong tên của class dẫn xuất nên lấy tên của class cơ sở.

Ví dụ:

ApplicationException là tên class dẫn xuất từ class cơ sở **Exception**, và với cách đặt tên này ta có thể hiểu rằng **ApplicationException** là một trong các loại **Exception**.

Với quy tắc này thì tùy trường hợp mà chúng ta có thể áp dụng để tránh đặt tên class dài lê thê với những từ không cần thiết.

1.9. HÀM MAIN

- Hàm Main() cho CLR biết rằng đây là phương thức đầu tiên của chương trình được khai báo trong một lớp và chỉ định nơi bắt đầu thực thi chương trình.
- Mọi chương trình C # được thực thi phải có phương thức Main () vì nó là điểm vào của chương trình.
- Kiểu trả về của Main () trong C # có thể là int hoặc void

1.10. TẠO OBJECT

Để tạo **Object** từ **Class**, dùng từ khoá **new**, lúc đó chúng ta có thể truy cập vào các phương thức, thuộc tính của Class.

Khi gặp từ khóa new, trình biên dịch Just-In-Time (JIT) sẽ cấp phát bộ nhớ cho đối tượng và trả về một tham chiếu của đối tượng đó đã được cấp phát bộ nhớ.

Cú pháp:

```
<ClassName> <objectName> = new <ClassName> ();
```

2 references

`class StudentDetails``{` `string _studName = "James";` `int rollNumber = 20;`

0 references

`static void Main(string[] args)``{` `StudentDetails objstudents = new StudentDetails();` `Console.WriteLine("Student Name: " + objstudents._studName);` `Console.WriteLine("Roll Number: " + objstudents.rollNumber);``}``}`

2.1. PHƯƠNG THỨC - METHOD

- Là một khối các câu lệnh để thực thi một công việc nào đó, được khai báo ở trong **class** hoặc **struct**.
- Phương thức là các hành vi hay phương thức hoạt động của đối tượng.
- Phương thức trong C# là một tập các câu lệnh cùng nhau thực hiện một tác vụ nào đó. Mỗi chương trình C# có ít nhất một lớp với một phương thức là Main.

Ví dụ:

- Lớp Car có phương thức Brake() đại diện cho hành động phanh
- Để thực hiện hành động này, phương thức Brake() sẽ phải được gọi bởi một đối tượng của lớp

Car

```
<<kiểu trả về>> <<tên phương thức>> ([danh sách tham số])  
{  
    // thân phương thức  
}
```


2.2. TẠO METHOD

Đặt tên Method

- Dùng các động từ hay các cụm động từ để đặt tên cho methods.
- Dùng Pascal Case.

Ví dụ tên các methods chuẩn:

RemoveAll()

GetCharArray()

Invoke()

- **Cú pháp**

```
<access_modifier><return_type><MethodName>([list of parameters]){  
// body of the method  
}
```

- Cấu trúc của Method:

- **Access-Modifier:** xác định giới hạn truy cập của method.
- **Return-Type:** kiểu giá trị trả về như *string*, *int*, v.v hoặc kiểu “*void*” nếu không trả về giá trị
- **Method-Name:** tên của method
- **Parameter-List:** danh sách các tham số hoặc đối số truyền vào method

```
public class Employee{  
    public String fullname;  
    public double salary;
```

```
    public void input(){...}  
    public void output(){...}
```

Kiểu trả về là **void** nên thân phương thức
không chứa lệnh return giá trị

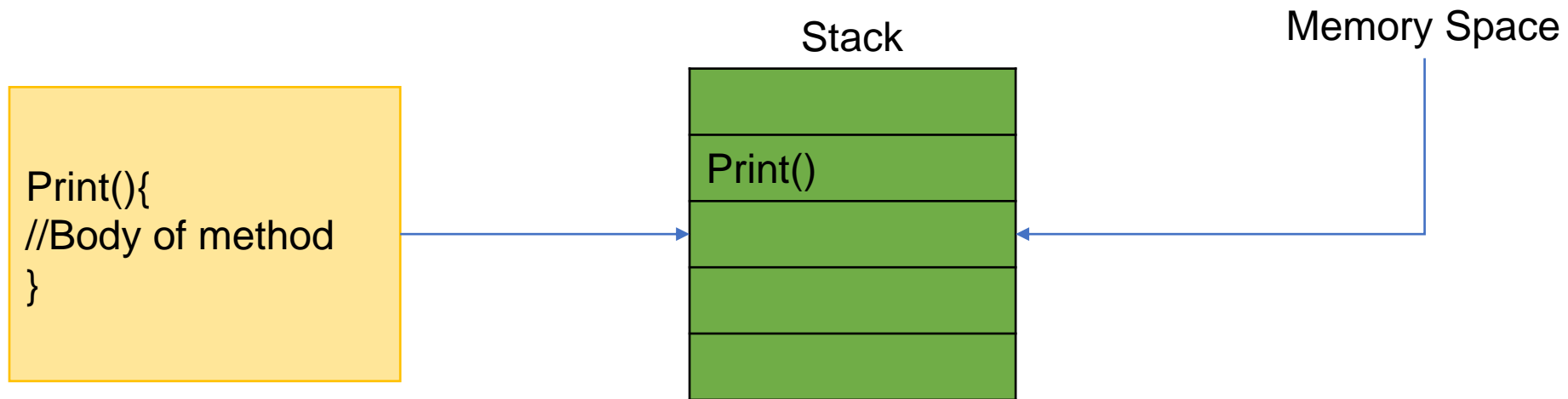
```
    public void setInfo(String fullname, double salary) {  
        this.fullname = fullname;  
        this.salary = salary;  
    }
```

Kiểu trả về là **double** nên thân phương
thức phải chứa lệnh **return số thực**

```
    public double incomeTax(){  
        if(this.salary < 5000000){  
            return 0;  
        }  
        double tax = (this.salary - 5000000) * 10/100;  
        return tax;  
    }  
}
```

2.3. INVOKING METHOD – LỜI GỌI HÀM

- Một phương thức có thể được gọi trong một lớp bằng cách tạo một đối tượng của lớp trong đó tên đối tượng được theo sau bởi dấu chấm (.) và tên của phương thức theo sau là tham số.
- Hình bên dưới hiển thị cách thức gọi phương thức hoặc call được lưu trữ trong ngăn xếp bộ nhớ và xác định bên trong phương thức.



```
class Book
{
    string _bookName;
    1 reference
    public string Print()
    {
        return _bookName;
    }
    1 reference
    public void Input(string bkName)
    {
        _bookName = bkName;
    }
    0 references
    static void Main(string[] args)
    {
        Book objBook = new Book();
        objBook.Input("C#-The Complete Reference");
        Console.WriteLine(objBook.Print());
    }
}
```

2.4. PHƯƠNG THỨC PARAMETER(THAM SỐ) VÀ ARGUMENT(ĐỐI SỐ)

Parameters

Là các biến được **sử dụng trong một hàm** mà giá trị của biến đó được **cung cấp bởi lời gọi hàm**. Các tham số được đặt bên trong dấu ngoặc đơn, cú pháp giống khai báo biến, **cách nhau bằng dấu phẩy “,”**

Arguments

Là các **giá trị truyền vào** hàm qua lời gọi hàm, cách nhau bởi **dấu phẩy “,”**. Số lượng đối số tương ứng với số lượng tham số của hàm.

```
namespace ConsoleApplicationCS
{
    class Program
    {
        static void Main(string[] args /*đây là Parameter*/)
        {
            System.Console.WriteLine("Sum Program:" /*còn đây là Argument*/);
            System.Console.WriteLine(Sum(5, 10 /*đây cũng là Argument*/));
            System.Console.ReadLine();
        }

        static int Sum (int a, int b /*cả 2 đều là Parameter*/)
        {
            return a + b;
        }
    }
}
```

2.5. ĐẶT TÊN VÀ KHÔNG BẮT BUỘC ARGUMENT(ĐỐI SỐ)

- Một phương thức trong chương trình C# có thể chấp nhận nhiều đối số được truyền dựa trên vị trí của các tham số trong nhận dạng phương thức
- Phương thức gọi có thể đặt tên rõ ràng cho một hoặc nhiều đối số được truyền vào phương thức thay vì truyền các đối số dựa trên vị trí của chúng.
- Một đối số được truyền bởi tên của nó thay vì vị trí của nó được gọi là một đối số được đặt tên.
- Trong khi truyền các đối số được đặt tên, thứ tự của các đối số được khai báo trong phương thức không quan trọng.
- Các đối số được đặt tên có lợi vì bạn không phải nhớ thứ tự chính xác của các tham số trong danh sách tham số của các phương thức.

2 references

`internal class Students``{`

4 references

`void printName(string firstName, string lastName)``{` `Console.WriteLine("First Name = {0}, Last Name = {1}", firstName, lastName);``}`

0 references

`static void Main(string[] args)``{` `Students objstudents = new Students();` `/*Passing argument by position*/` `objstudents.printName("Henry", "Parker");` `/*Passing named argument*/` `objstudents.printName(firstName: "Henry", lastName: "Parker");` `objstudents.printName(lastName: "Parker", firstName: "Henry");` `/*Passing named argument after positional argument*/` `objstudents.printName("Henry", lastName: "Parker");``}``}`

2 references

```
class TestProgram
```

```
{
```

1 reference

```
void Count(int boys, int girls)
{
    Console.WriteLine(boys+girls);
}
```

0 references

```
static void Main(string[] args)
{
    TestProgram objTest = new TestProgram();
    objTest.Count(boys:16, girls: 24);
}
```

```
}
```

Named Arguments

2 references

```
class OptionalParameterExample
```

```
{
```

2 references

```
void printMessage(string message = "Hello user!")
{
    Console.WriteLine("{0}",message);
}
```

0 references

```
static void Main(string[] args)
{
    OptionalParameterExample objExample = new OptionalParameterExample();
    objExample.printMessage("Welcome user!");
    objExample.printMessage();
}
```

```
}
```

Optional Arguments

2.6. LỚP STATIC – LỚP TĨNH

- Là các lớp không thể khởi tạo hoặc kế thừa và từ khóa static được sử dụng trước tên lớp bao gồm các thành phần dữ liệu tĩnh và các phương thức tĩnh.
- Không thể tạo một phiên bản của lớp tĩnh bằng cách sử dụng từ khóa new. Các tính năng chính của các lớp tĩnh như sau:
 - Chúng chỉ có thể chứa các thành phần tĩnh.
 - Chúng không thể được khởi tạo hoặc kế thừa và không thể chứa các hàm dựng.Tuy nhiên, lập trình viên có thể tạo các hàm tạo tĩnh để khởi tạo các thành phần tĩnh

- Code tạo lớp static Product có biến static _productID và price, phương thức static gọi Display()
- Định nghĩa 1 hàm dựng Product() khởi tạo các biến lớp có giá trị tương ứng từ 10 và 156.32
- Vì không cần tạo các đối tượng của lớp static để gọi phương thức cần thiết, nên việc thực hiện chương trình đơn giản và nhanh hơn so với các chương trình chứa lớp thể hiện

```
2 references
static class Product
{
    static int _productID;
    static double _price;

    0 references
    static Product()
    {
        _productID = 10;
        _price = 156.32;
    }

    1 reference
    public static void Display()
    {
        Console.WriteLine("Product ID: " + _productID);
        Console.WriteLine("Product price: " + _price);
    }
}

0 references
class Medicine
{
    0 references
    static void Main(string[] args)
    {
        Product.Display();
    }
}
```

2.7. PHƯƠNG THỨC STATIC

- Một phương thức được gọi bằng cách sử dụng một đối tượng của lớp nhưng có thể phương thức được gọi mà không cần tạo bất kỳ đối tượng nào của lớp bằng cách khai báo gọi là phương thức tĩnh.
- Phương thức static được khai báo bằng cách sử dụng từ khóa static. Ví dụ, phương thức Main() là phương thức tĩnh và nó không yêu cầu bất kỳ thể hiện nào của lớp để nó được gọi.
- Một phương thức tĩnh chỉ có thể tham chiếu trực tiếp đến các biến tĩnh và các phương thức tĩnh khác của lớp nhưng có thể tham chiếu đến các phương thức và biến không tĩnh bằng cách sử dụng lớp instance.
- **Cú pháp:**

```
static <return_type><MethodName>(){  
//body of the method  
}
```

3 references

class Calculate

{

1 reference

public static void Addition(int val1, int val2)

{

Console.WriteLine(val1 + val2);

}

1 reference

public void Multiply(int val1, int val2)

{

Console.WriteLine(val1 * val2);

}

}

0 references

class StaticMethods

{

0 references

static void Main(string[] args)

{

Calculate.Addition(10, 50);

Calculate objCal = new Calculate();

objCal.Multiply(10, 20);

}

}

2.8. BIẾN STATIC

- Ngoài phương thức static, còn có biến static trong C#.
- Biến static là một biến đặc biệt, truy cập mà không cần sử dụng đối tượng của lớp.
- Một biến được khai báo là static bằng cách sử dụng từ khóa static. Khi biến static được tạo, nó sẽ tự động được khởi tạo trước khi được truy cập.
- Chỉ một bản sao của một biến static được chia sẻ bởi tất cả các đối tượng của lớp.
- Do đó, sự thay đổi giá trị của một biến như vậy được phản ánh bởi tất cả các đối tượng của lớp.
- Một thể hiện của một lớp không thể truy cập các biến static

```
class Employee
{
    public static int EmpId = 20;
    public static string EmpName = "James";
    0 references
    public static void Main(string[] args)
    {
        Console.WriteLine("Employee ID: "+ EmpId);
        Console.WriteLine("Employee Name: "+EmpName);
    }
}
```

3.1. ĐẶC TẢ TRUY XUẤT – ACCESS MODIFIER

Các access modifiers trong CSharp xác định độ truy cập (Phạm vi) vào dữ liệu của của các trường, phương thức, cấu tử (constructor) hoặc class.

Có 5 kiểu access modifiers trong CSharp:

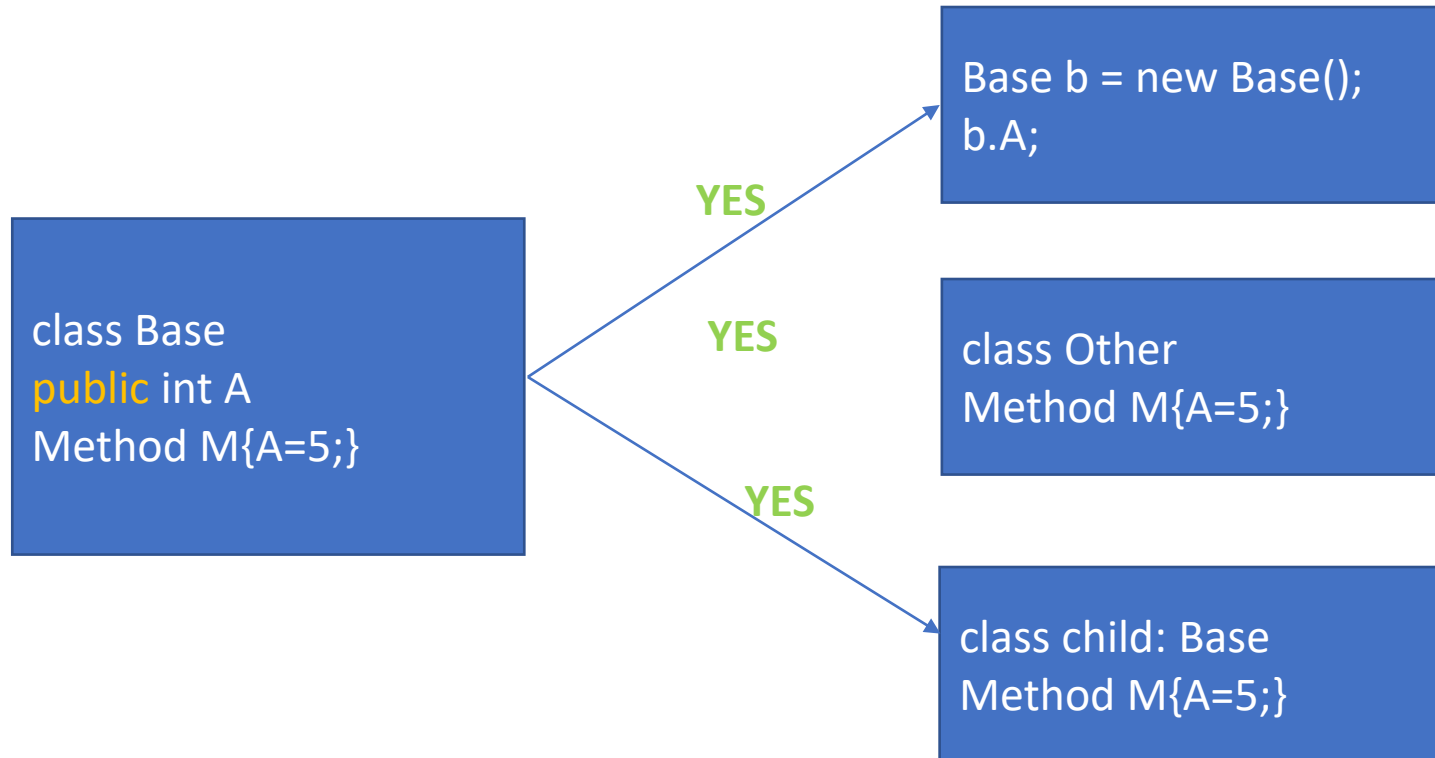
- private
- protected
- internal
- protected internal
- public

3.1. ĐẶC TẢ TRUY XUẤT – ACCESS MODIFIER (2)

Độ truy cập (Modifier)	Mô tả
private	Truy cập bị hạn chế trong phạm vi của định nghĩa Class. Đây là loại phạm vi truy cập mặc định nếu không được chính thức chỉ định
protected	Truy cập bị giới hạn trong phạm vi định nghĩa của Class và bất kỳ các class con thừa kế từ class này.
internal	Truy cập bị giới hạn trong phạm vi Assembly của dự án hiện tại.
protected internal	Truy cập bị giới hạn trong phạm vi Assembly hiện tại và trong class định nghĩa hoặc các class con.
public	Không có bất kỳ giới hạn nào khi truy cập vào các thành viên công khai (public)

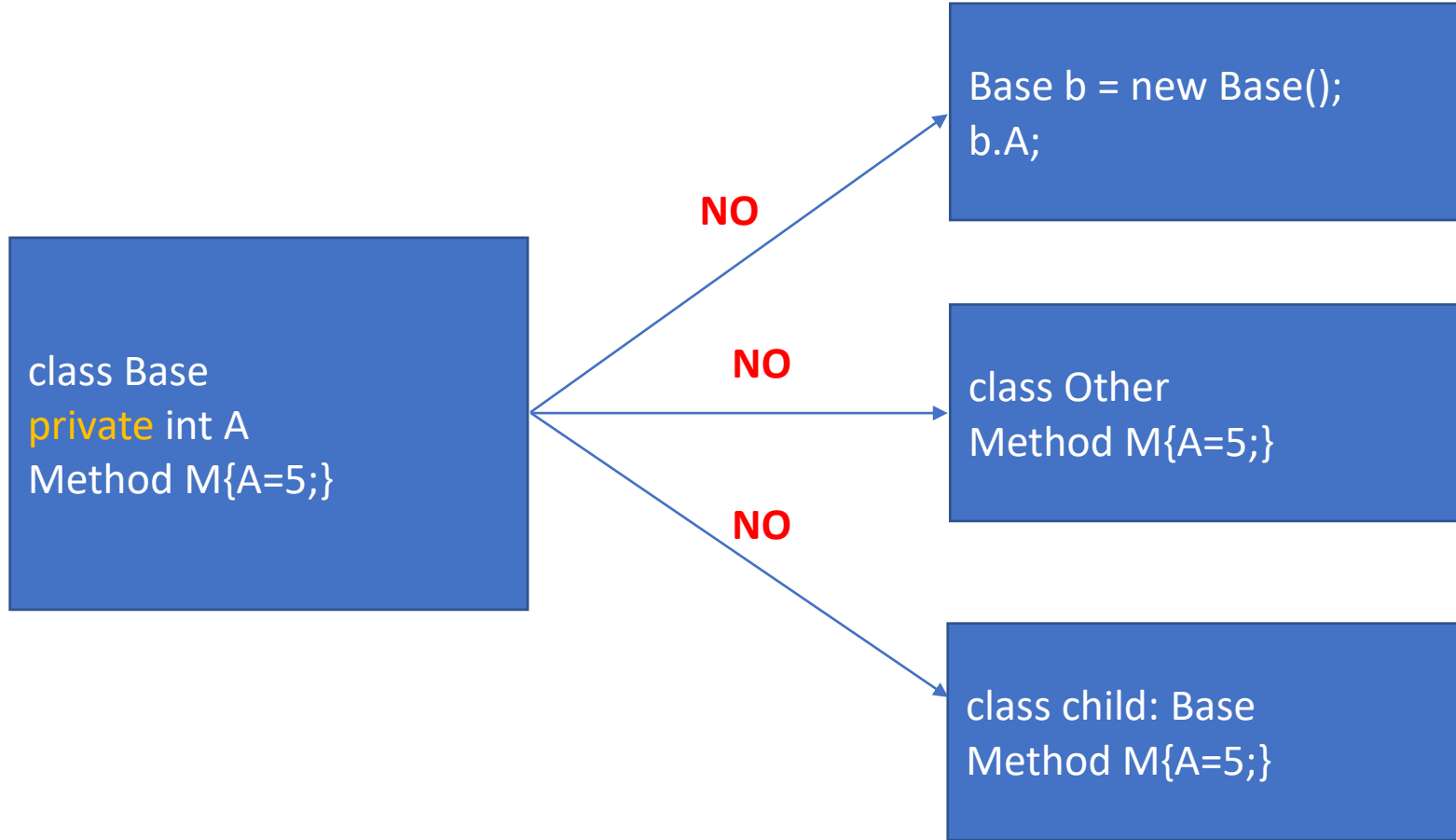
3.2. PUBLIC

```
class Employee
{
    //No access restrictions
    public string Name = "Willson";
}
```



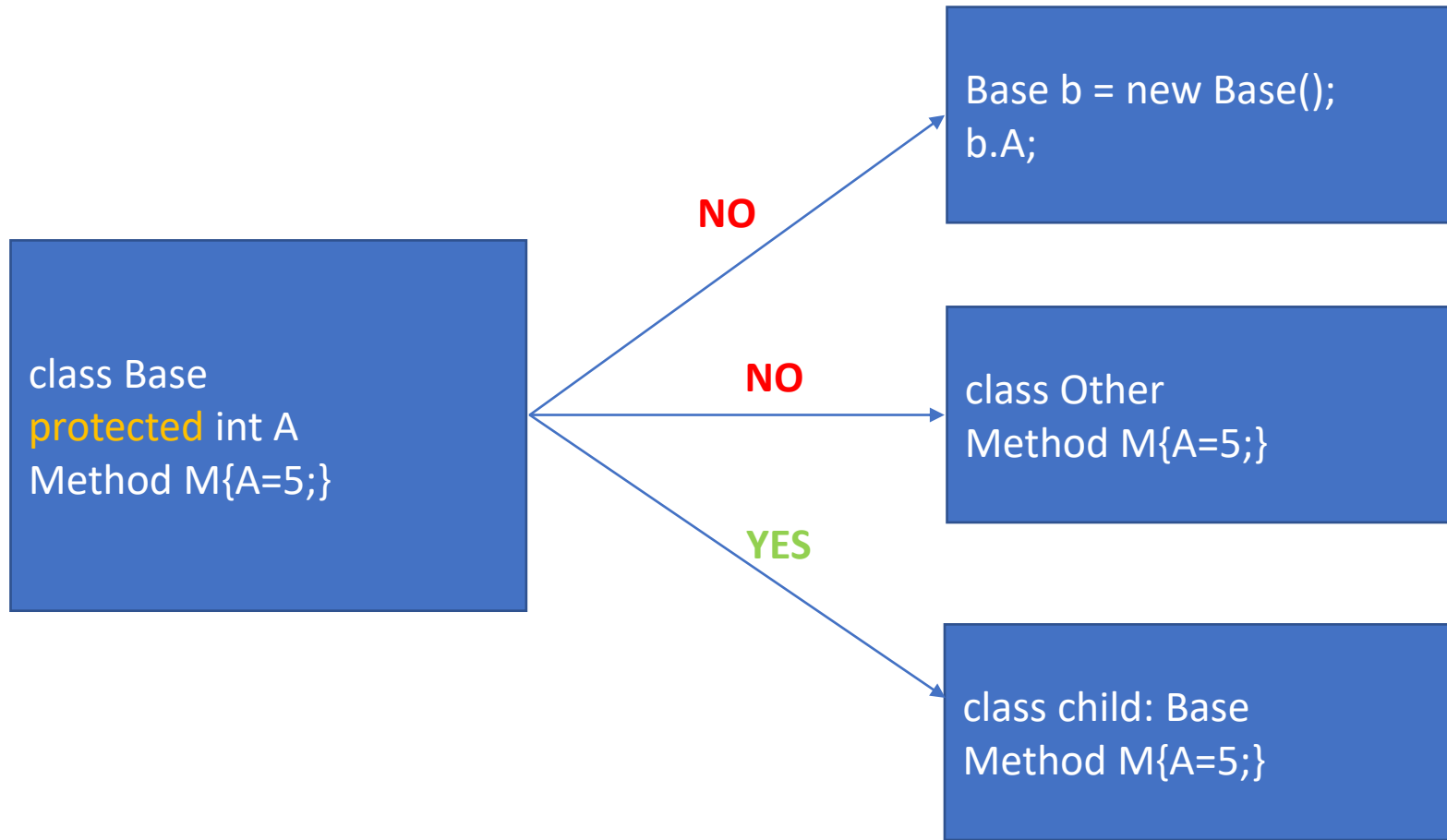
3.3. PRIVATE

```
class Employee
{
    //Accessible only within the class
    private float salary;
}
```



3.4. PROTECTED

```
class Employee
{
    //Protected access
    protected float _salary;
}
```



3.5. INTERNAL

```
class Employee
{
    //Only accessible within the same assembly
    internal static int NumOne = 3;
}
```

APPLICATION 1.exe

class Base
Internal int A
Method M{A=5;}

YES

Base b = new Base();
b.A;

YES

class Other
Method M{A=5;}

YES

class child: Base
Method M{A=5;}

NO

class child2: Base
Method M{A=5;}

NO

Base b = new Base();
b.A;

APPLICATION 2.exe

3.6. ASSEMBLY

- ❑ Assembly: là file đã được biên dịch(precompiled), có 2 dạng (.exe và .dll)
- ❑ Assembly có thể chứa một hoặc nhiều namespace

	Cùng Assembly			Khác Assembly	
	Trong class định nghĩa?	Trong class con	Ngoài class định nghĩa, ngoài class con	Trong class con	Ngoài class con
private	Y				
protected	Y	Y		Y	
internal	Y	Y	Y		
protected internal	Y	Y	Y		
public	Y	Y	Y	Y	Y

```
namespace AccessModifierTutorial
{
    class Person
    {
        private string Name;

        public Person(string name)
        {
            this.Name = name;
        }

        private void ShowSecret()
        {
            Console.WriteLine("Secret of " + Name);
        }

        private static void DoSomething(String job)
        {
            Console.WriteLine("Do Job: " + job);
        }

        class Diary
        {
            public void Logging()
            {
                DoSomething("Code CSharp");
            }

            ShowSecret();
        }
    }
}
```

private field

private method

(private static)

none-static

```
namespace AccessModifierTutorial
{
    class PersonTest
    {
        public static void Main(string[] args)
        {
            Person tom = new Person("Tom");

            String name = tom.Name; // Error

            tom.ShowSecret(); // Error

            Person.DoSomething(); // Error
        }
    }
}
```



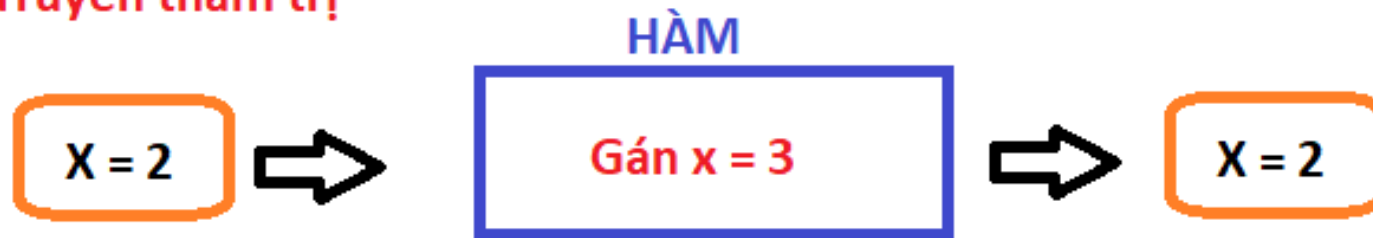
Bạn không thể truy cập vào các thành viên **private** từ bên ngoài class định nghĩa ra nó

3.7. TỪ KHÓA REF VÀ OUT

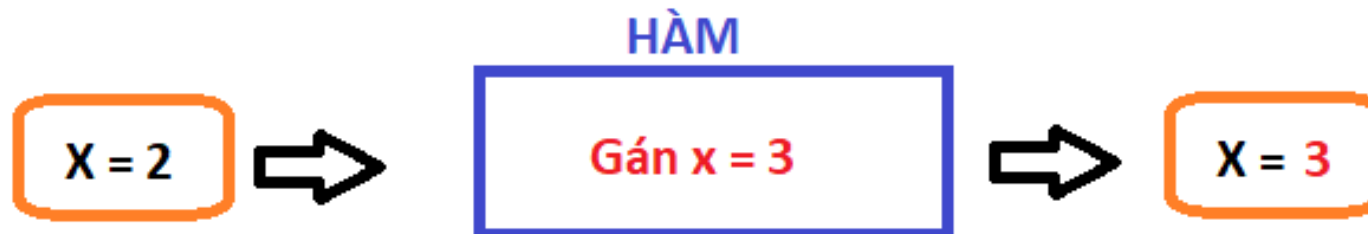
Trong lập trình C#, có hai loại truyền tham số vào hàm:

- **Truyền tham trị:** một bản sao của biến sẽ được tạo ra, sao chép giá trị của biến, truyền biến đã được sao chép này vào hàm, dù có thực hiện bao nhiêu phép tính toán cũng không ảnh hưởng đến biến gốc
- **Truyền tham chiếu:** truyền ngay địa chỉ của biến được lưu trên bộ nhớ vào hàm (hay hiểu cách khác là truyền chính biến đó vào hàm) khi thực hiện tính toán thì giá trị biến này thay đổi theo.

Truyền tham trị



Truyền tham chiếu



3.8. CÁCH SỬ DỤNG THAM CHIỀU

❑ Sử dụng từ khóa ref hoặc out

❖ Khi dùng ref: biến phải được khởi tạo trước khi truyền cho phương thức

❖ Khi dùng out: biến không cần khởi tạo trước, bên trong phương thức cần gán giá trị cho biến

❑ Khi khai báo và gọi phương thức thì bắt buộc dùng ref hoặc out trước tên biến

❑ Không thể truyền vào một hằng vì hằng là giá trị không thay đổi

webmaster@dotnet.vn

```
class RefParameters
{
    1 reference
    static void Calculate(ref int numValueOne, ref int numValueTwo)
    {
        numValueOne = numValueOne * 2;
        numValueTwo = numValueTwo / 2;
    }
    0 references
    static void Main(string[] args)
    {
        int numOne = 10;
        int numTwo = 20;
        Console.WriteLine("Value of Num1 and Num2 before calling method " + numOne + "," + numTwo);
        Calculate(ref numOne, ref numTwo);
        Console.WriteLine("Value of Num1 and Num2 after calling method " + numOne + "," + numTwo);
    }
}
```



C:\Users\lethiennhatquang\source\repos\Demo2\Demo2\bin\Debug

```
Value of Num1 and Num2 before calling method 10,20
Value of Num1 and Num2 after calling method 20,10
```


3.9. TỪ KHÓA OUT

Từ khóa out cũng tương tự từ khóa ref. Đó là:


- Vùng nhớ của các parameter sẽ được hàm sử dụng thao tác trực tiếp, dẫn đến khi kết thúc lời gọi hàm giá trị của các parameter có thể bị thay đổi.
- Phải có từ khóa out trước tên parameter của hàm và trước tên biến truyền vào khi gọi hàm sử dụng.

Nhưng có một sự khác biệt đó là:

- **Biến truyền vào có từ khóa out sẽ không cần khởi tạo giá trị ban đầu.**
- Parameter đó chỉ như một thùng chứa kết quả trả về khi kết thúc gọi hàm.
- Đồng thời parameter đó phải được khởi tạo ngay bên trong lời gọi hàm.
- **Cú pháp**

```
<access_modifier><return_type><MethodName>    (out    parameter1,    out    parameter2,  
parameter3, parameter4,... parameterN)  
{  
//actions to be performed  
}
```

```
class OutParameters
{
    1 reference
    static void Depreciation(out int val)
    {
        val = 200;
        int dep = val * 5 / 100;
        int amt = val - dep;
        Console.WriteLine("Depreciation Amount: " + dep);
        Console.WriteLine("Reduced value after depreciation: " + amt);
    }
    0 references
    static void Main(string[] args)
    {
        int value;
        Depreciation(out value);
        Console.ReadLine();
    }
}
```

 C:\Users\lethiennhatquang\source\repos\Demo2\Demo2\bin\Debug\Demo2.exe

```
Depreciation Amount: 10
Reduced value after depreciation: 190
```

```
static void Main(string[] args)
{
    // Khai báo và khởi tạo
    int a = 100;
    int b = 99;

    Console.WriteLine( "Before swap: a = {0}, b = {1}", a, b );

    // Truyền 2 biến a, b theo kiểu Tham chiếu, dùng từ khóa [ref]
    Swap( ref a, ref b );
    Console.WriteLine( "After swap: a = {0}, b = {1}", a, b );

    // Chỉ khai báo
    int x;

    // Truyền biến x theo kiểu Tham chiếu, dùng từ khóa [out]
    GetValue( out x );

    Console.WriteLine( "Now, value of x = {0}", x );

    Console.ReadKey();
}

// Method hoán vị 2 số Swap, dùng biến tham chiếu [ref]
public static void Swap(ref int a, ref int b)
{
    int tmp;
    tmp = a;
    a = b;
    b = tmp;
}

// Method GetValue, dùng biến tham chiếu [out]
public static void GetValue(out int x)
{
    x = 5;
}
```

CÂU HỎI

1. Tại sao không dùng return để trả kết quả về mà phải dùng hai từ khóa ref/out để lưu lại giá trị?

Có thể dùng return để trả kết quả về khi chúng ta chỉ có duy nhất một giá trị trả về, nếu có 2 hoặc nhiều hơn kết quả cần lưu lại, ta sẽ nhận ra sự hữu ích của hai từ khóa ref out

2. Có thể dùng out cho các trường hợp của ref được không?

Tùy trường hợp để các bạn quyết định dùng **ref** hay **out**:

- Nếu muốn thay đổi giá trị của biến đã có dữ liệu trước (hoặc trong hàm phải phụ thuộc giá trị biến để kiểm tra điều kiện) => dùng **ref**
- Nếu không cần quan tâm giá trị ban đầu của biến là gì, chỉ gán lại giá trị cho biến => dùng **out**.

Lưu ý: phải khởi tạo hoặc gán lại giá trị lại cho biến trước khi thay đổi giá trị của biến đó. Nếu không sẽ báo lỗi.

4.1. PHƯƠNG THỨC NẠP CHỒNG

- Là việc tạo ra nhiều phương thức(*method*) có **cùng tên**, trong **cùng một phạm vi**(*scope*), nhưng **khác nhau về đối số đầu vào**(*arguments*) hoặc **kiểu trả về**.
- Khi thực hiện **overloading** thì các methods sẽ có **cùng tên**, do đó mà trình biên dịch(*compiler*) sẽ dựa vào danh sách tham số của các methods để xác định method nào được gọi theo các ngữ cảnh khác nhau.
- Vậy để xác định danh sách tham số của các methods cùng tên là khác nhau khi sử dụng kỹ thuật **overloading** thì sẽ dựa vào điều gì!?

4.1. PHƯƠNG THỨC NẠP CHỒNG (2)

- Dựa vào **số lượng tham số**: chẳng hạn một method có 2 tham số, method khác có 3,... Bằng cách này, trình biên dịch sẽ gọi đúng method với số lượng chính xác của các tham số được truyền vào.

```
class Students
{
    0 references
    public string Hello(string name)
    {
        return "Hello: " + name;
    }

    0 references
    public string Hello(string firstName, string lastName)
    {
        return string.Format("Hello: {0} {1}", firstName, lastName);
    }
}
```

4.1. PHƯƠNG THỨC NẠP CHỒNG (3)

- Dựa vào **kiểu dữ liệu của tham số**: nếu hai methods có cùng số tham số, nhưng các kiểu khác nhau (int, string, long ...), thì trình biên dịch sẽ biết được method nào để gọi.

```
class Students
{
    0 references
    public string GetName(int masv) { return ""; }

    0 references
    public string GetName(string email) { return ""; }
}
```

4.1. PHƯƠNG THỨC NẠP CHỒNG (4)

- Dựa vào **thứ tự của các tham số**: khi chúng ta có một kết hợp số tham số và kiểu tham số, nhưng với một thứ tự khác, thì lại dễ dàng cho trình biên dịch biết method nào cần gọi.

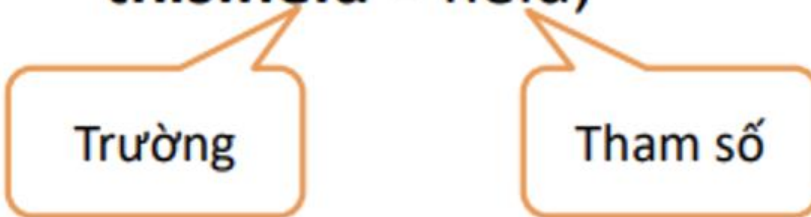
```
0 references
class Students
{
    0 references
    public string GetName(int masv, string email) { return ""; }

    0 references
    public string GetName(string address, int mobile) { return ""; }
}
```


4.2. TỪ KHÓA THIS

- ❑ `this` được sử dụng để đại diện cho đối tượng hiện tại.
- ❑ `this` được sử dụng trong lớp để tham chiếu tới các thành viên của lớp (field và method)
- ❑ Sử dụng **`this.field`** để phân biệt field với các biến cục bộ hoặc tham số của phương thức

```
public class MyClass{  
    int field;  
    void method(int field){  
        this.field = field;  
    }  
}
```



4.3. CONSTRUCTOR(HÀM DỰNG) VÀ DESTRUCTOR(HÀM HỦY)

- **Constructor** là một phương thức đặc biệt được gọi tự động tại thời điểm đối tượng được tạo ra.
 - Mục đích của hàm dựng dùng để khởi tạo dữ liệu cho dữ liệu thành viên.
 - Constructor phải trùng tên với tên lớp và không có kiểu trả về kể cả kiểu void.
- **Destructor** là hàm hủy hoạt động ngược lại với hàm tạo. Hàm hủy có tiền tố đằng trước
 - Nó phá hủy các đối tượng của các lớp. Nó chỉ có thể được định nghĩa một lần trong một lớp. Giống như hàm dựng, hàm hủy được gọi tự động.

4.4. HÀM DỰNG

Cú pháp:

```
class <ClassName>
{
    <ClassName> ()
    {
        //Initialization Code
    }
}
```

```
class Employee
{
    string _empName;
    int _empAge;
    string _deptName;
    1 reference
    Employee(string name, int num)
    {
        _empName = name;
        _empAge = num;
        _deptName = "Research & Development";
    }

    0 references
    static void Main(string[] args)
    {
        Employee objEmp = new Employee("John", 10);
        Console.WriteLine(objEmp._deptName);
    }
}
```

```
public class ChuNhat {  
    public double dai, rong;  
    public ChuNhat(double dai, double rong) {  
        this.dai = dai;  
        this.rong = rong;  
    }  
    public ChuNhat(double canh) {  
        this.dai = canh;  
        this.rong = canh;  
    }  
}
```

```
public static void main(String[] args) {  
    ChuNhat cn = new ChuNhat(20, 15);  
    ChuNhat vu = new ChuNhat(30);  
}
```

4.5. HÀM DỰNG MẶC ĐỊNH(DEFAULT) VÀ TĨNH(STATIC)

- Nếu không xây dựng phương thức khởi tạo nào cho lớp, C# sẽ tạo ra một phương thức khởi tạo ngầm định, nó thiết lập giá trị các thành viên dữ liệu nhận giá trị mặc định (tùy theo kiểu dữ liệu).
- Có thể xây dựng một phương thức khởi tạo không tham số có chỉ thị truy cập là static, phương thức khởi tạo này dùng để khởi tạo các thành viên dữ liệu tĩnh, nó tự động gọi khi truy cập một thành viên dữ liệu tĩnh lần đầu

4.6. HÀM DỰNG STATIC

Cú pháp:

```
class <ClassName>
{
    static <ClassName>()
    {
        //Initialization code
    }
}
```

```
class Multiplication
{
    static int _valueOne = 10;
    static int _product;
    0 references
    static Multiplication()
    {
        Console.WriteLine("Static Constructor initialized");
        _product = _valueOne * _valueOne;
    }

    1 reference
    public static void Method()
    {
        Console.WriteLine("Value of product: "+ _product);
    }

    0 references
    static void Main(string[] args)
    {
        Multiplication.Method();
    }
}
```

4.7. HÀM DỰNG NẠP CHỒNG

- Nó khá giống với Phương thức nạp chồng.
- Nó có khả năng xác định lại một Constructor ở nhiều dạng. Người dùng có thể thực hiện nạp chồng hàm tạo bằng cách xác định hai hoặc nhiều hàm tạo trong một lớp có cùng tên.
- C # có thể phân biệt các hàm tạo với các Signature(Không bao gồm kiểu trả về của phương thức) khác nhau. Nghĩa là hàm tạo phải có cùng tên nhưng với danh sách tham số khác nhau.
- Chúng ta có thể nạp chồng các hàm tạo theo những cách khác nhau như sau:
 - Bằng cách sử dụng các kiểu đối số khác nhau
 - Bằng cách sử dụng số lượng đối số khác nhau
 - Bằng cách sử dụng các thứ tự đối số khác nhau

```
public class Rectangle
{
    double _length;
    double _breadth;

    1 reference
    public Rectangle()
    {
        _length = 13.5;
        _breadth = 20.5;
    }

    1 reference
    public Rectangle(double len, double wide)
    {
        _length = len;
        _breadth = wide;
    }

    2 references
    public double Area()
    {
        return _length * _breadth;
    }

    0 references
    static void Main(string[] args)
    {
        Rectangle objRectangle = new Rectangle();
        Console.WriteLine("Area of Rectangle: " + objRectangle.Area());
        Rectangle objRectangle2 = new Rectangle(2.5, 6.9);
        Console.WriteLine("Area of Rectangle: " + objRectangle2.Area());
    }
}
```


4.8. HÀM HỦY

- Hàm huỷ được gọi trước khi giải phóng (xoá bỏ) một đối tượng để thực hiện một số công việc có tính “dọn dẹp” trước khi đối tượng được huỷ bỏ, ví dụ như giải phóng một vùng nhớ mà đối tượng đang quản lý, xoá đối tượng khỏi màn hình nếu như nó đang hiển thị,...
- Nếu trong lớp không định nghĩa hàm huỷ, thì một hàm huỷ mặc định không làm gì cả được phát sinh. Đối với nhiều lớp thì hàm huỷ mặc định là đủ, và không cần đưa vào một hàm huỷ mới.
- Một **Destructor** trong C# là một hàm thành viên đặc biệt:
 - Có tên giống với tên class, và có thêm dấu ngã ~ ở trước. Chẳng hạn: ~Car() , ~Student(),...
 - Không có giá trị trả về, có thể có hoặc không có tham số.

BÀI TẬP

webmaster@dotnet.vn

SinhVien
+ hoTen: String + diemTB: double
+ xepLoai(): String + xuất(): void + nhập(): void
+ SinhVien() + SinhVien(hoTen, diemTB)



DEMO

Xây dựng lớp mô tả sinh viên như mô hình trên.
Trong đó nhập() cho phép nhập họ tên và điểm từ bàn phím; xuất() cho phép xuất họ tên, điểm và học lực ra màn hình; xepLoai() dựa vào điểm để xếp loại học lực
Sử dụng 2 hàm tạo để tạo 2 đối tượng sinh viên

Bài 2: Tạo lớp SanPham gồm 3 thuộc tính là tên, giá và giảm giá. Lớp cũng gồm 2 phương thức là tính thuế nhập khẩu (10% giá sản phẩm) và xuất thông tin ra màn hình. Thông tin xuất ra màn hình gồm:

- ✓ Tên sản phẩm:
- ✓ Đơn giá:
- ✓ Giảm giá:
- ✓ Thuế nhập khẩu

SanPham
+tenSp: String +donGia: double +giamGia: double
+getThueNhapKhau(): double +xuat(): void +nhap():void

Bài 3: Viết chương trình tạo ra 2 sản phẩm thông tin được nhập từ bàn phím sau đó gọi phương thức xuất để xuất thông tin 2 đối tượng sản phẩm đã tạo.

HƯỚNG DẪN:

- ✓ Tạo lớp chứa phương thức main()
- ✓ Trong phương thức main() tạo 2 đối tượng sp1 và sp2 từ lớp sản phẩm
- ✓ Gọi phương thức nhap() của 2 đối tượng sp1 và sp2 để nhập dữ liệu từ bàn phím
- ✓ Gọi phương thức xuất() của 2 đối tượng sp1 và sp2 để xuất thông tin của mỗi đối tượng ra màn hình

Bài 4: Nâng cấp lớp SanPham bằng cách bổ sung public cho các đặt tả truy xuất cho phương thức xuất() và private cho getThueThuNhap(). Đồng thời bổ sung 2 hàm tạo, hàm tạo thứ nhất gồm 3 tham số là tên, giá và giảm giá, hàm tạo thứ 2 gồm 2 tham số là tên và giá (ngầm hiểu không giảm giá).

Viết chương trình tạo 2 sản phẩm có giảm giá và không giảm giá sau đó xuất thông tin 2 sản phẩm ra màn hình.

HƯỚNG DẪN

- ✓ Bổ sung 2 hàm tạo

```
public SanPham(String tenSp, double donGia, double giamGia){  
    this.tenSp = tenSp;  
    ...  
}  
  
public SanPham(String tenSp, double donGia){  
  
}
```

- ✓ Sử dụng hàm tạo để tạo sản phẩm

```
SanPham sp1 = new SanPham(tenSp, donGia)
```


Bài 5: Nâng cấp lớp SanPham bằng cách khai báo các trường dữ liệu với đặc tả truy xuất là private để hạn chế truy xuất trực tiếp đến các trường này sau đó bổ sung các phương thức getter và setter để đọc ghi dữ liệu các trường

HƯỚNG DẪN:

- ✓ Cứ mỗi trường dữ liệu được khai private bạn cần định nghĩa một cặp phương thức getter/setter để cho phép đọc ghi dữ liệu thông qua các phương thức này. Với cách làm này bạn dễ dàng nâng cấp lớp để bảo vệ hoặc thực hiện các tính toán cần thiết trước khi thực hiện các thao tác đọc ghi dữ liệu

```
public String getTenSp(){  
    return this.tenSp;  
}  
public void setTenSp(String tenSp){  
    this.tenSp = tenSp;  
}
```