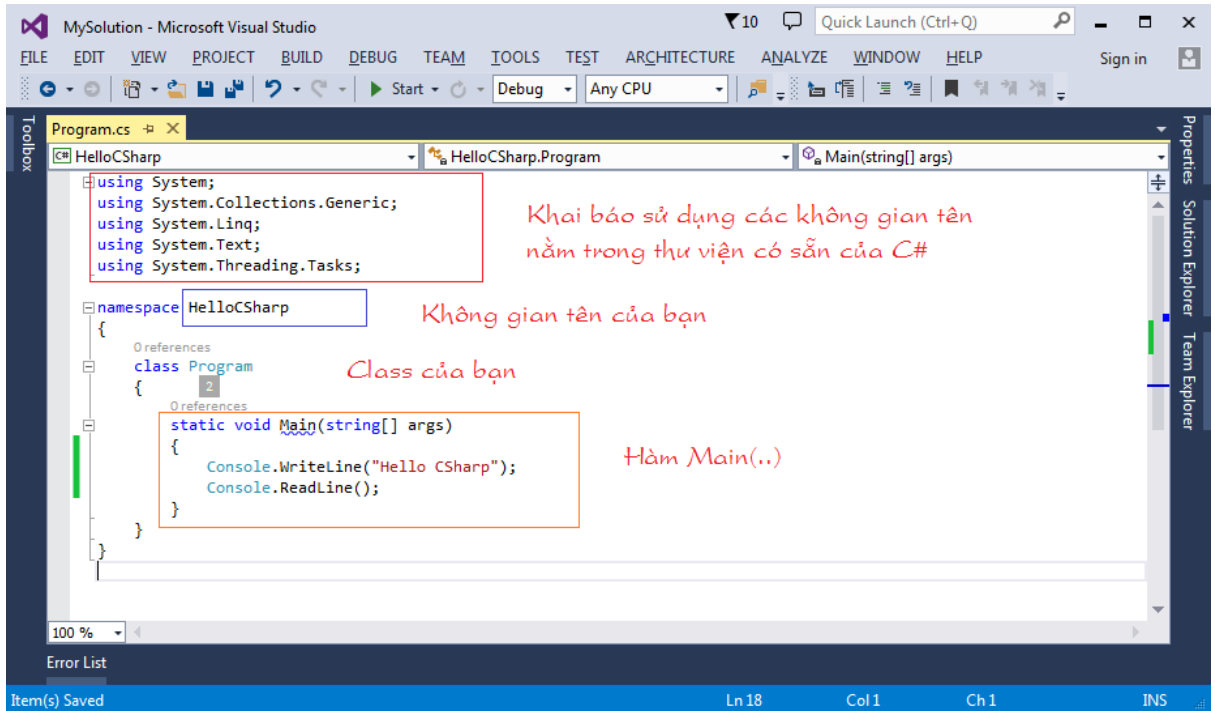


C# Note

- Một không gian tên (namespace) có thể chứa 1 hoặc nhiều class.



- Item(s) Saved
- Ln 18 Col 1 Ch 1 INS
- Nếu muốn sử dụng một lớp nào đó, thì phải khai báo sử dụng lớp đó, hoặc khai báo sử dụng không gian tên chứa lớp đó.
- // Khai báo sử dụng namespace System.
- // (Nghĩa là có thể sử dụng tất cả các class có trong namespace này).
- using System;
- **static** là từ khóa thông báo rằng đây là phương thức tĩnh.
- **void** là từ khóa thông báo rằng phương thức này không trả về gì cả.
- **args** là tham số của phương thức nó là một mảng các **string** (chuỗi) - **string[]**.
- static void Main(string[] args)
- {
- // Ghi ra màn hình Console một dòng chữ.
- Console.WriteLine("Hello CSharp");
- // Đợi người dùng gõ vào một dòng chữ trước khi tắt màn hình Console. (phải nhấn enter để thoát).
- Console.ReadLine();
- }
- // Khai báo sử dụng không gian tên System
- // (Nó có chứa class Console).
- using System;
-
- // Và có thể sử dụng class Console:
- Console.WriteLine("Hello CSharp");

-
- // Nếu bạn không muốn khai báo sử dụng không gian tên
- // Nhưng muốn in ra một dòng text, bạn phải viết đầy đủ:
- `System.Console.WriteLine("Hello CSharp");`
- Trong một ứng dụng **C#** cần khai báo rõ ràng một lớp có phương thức **Main(string[])** dùng làm điểm bắt đầu để chạy một ứng dụng.

- Các kiểu dữ liệu trong C#

Kiểu	Mô tả	Phạm vi	Giá trị mặc định
bool	Giá trị Boolean (Đúng hoặc sai).	True hoặc False	False
byte	Số tự nhiên không dấu 8-bit	0 tới 255	0
char	Ký tự unicode 16-bit	U +0000 tới U +ffff	'\0'
decimal	Có độ chính xác đến 28 con số và giá trị thập phân (Sử dụng 128-bit)	$(-7.9 \times 10^{28}$ tới $7.9 \times 10^{28}) / 100$ tới 28	0.0M
double	Kiểu dấu chấm động có độ chính xác gấp đôi (Sử dụng 64-bit)	$(+/-)5.0 \times 10^{-324}$ tới $(+/-)1.7 \times 10^{308}$	0.0D
float	Kiểu dấu chấm động (Sử dụng 32-bit)	-3.4×10^{38} to $+ 3.4 \times 10^{38}$	0.0F
int	Số nguyên có dấu 32-bit	-2,147,483,648 tới 2,147,483,647	0
long	64-bit signed integer type	-923,372,036,854,775,808 tới 9,223,372,036,854,775,807	0L
sbyte	Số nguyên có dấu 8-bit	-128 tới 127	0
short	Số nguyên có dấu 16-bit	-32,768 tới 32,767	0
uint	Số nguyên không dấu 32-bit	0 tới 4,294,967,295	0
ulong	Số nguyên không dấu 64-bit	0 tới 18,446,744,073,709,551,615	0
ushort	Số nguyên không dấu 16-bit	0 tới 65,535	0

Biến và khai báo

- Mỗi biến trong C# có một kiểu dữ liệu cụ thể. Trong đó xác định kích thước và phạm vi giá trị có thể được lưu trữ trong bộ nhớ, và tập hợp các toán tử có thể áp dụng cho biến
- Biến có thể thay đổi giá trị trong quá trình tồn tại của nó trong chương trình.
- Các biến có giá trị cố định được gọi là các hằng số, sử dụng từ khóa const để khai báo một biến là hằng số.
- `// Khai báo một biến.`
- `<kiểu dữ liệu> <tên biến>;`
-
- `// Khai báo một biến đồng thời gán luôn giá trị.`
- `<kiểu dữ liệu> <tên biến> = <giá trị>;`
-
- `// Khai báo một hằng số:`
- `const <kiểu dữ liệu> <tên hằng số> = <giá trị>;`

Câu lệnh rẽ nhánh

Câu lệnh If-else

if là một câu lệnh kiểm tra một điều kiện gì đó trong **C#**. Chẳng hạn: Nếu **a > b** thì làm gì đó
....

Các toán tử so sánh thông dụng:

Toán tử	Ý nghĩa	Ví dụ
>	Lớn hơn	5 > 4 là đúng (true)
<	Nhỏ hơn	4 < 5 là đúng (true)
>=	Lớn hơn hoặc bằng	4 >= 4 là đúng (true)
<=	Nhỏ hơn hoặc bằng	3 <= 4 là đúng (true)
==	Bằng nhau	1 == 1 là đúng (true)
!=	Không bằng nhau	1 != 2 là đúng (true)
&&	Và	a > 4 && a < 10

	Hoặc	a == 1 a == 4
--	------	------------------

// Cú pháp

```
if ( điều kiện)
{
    // Làm gì đó tại đây.
}
// Ví dụ 1:
if ( 5 < 10 )
{
    Console.WriteLine( "Five is now less than ten");
}
```

```
// Ví dụ 2:
if ( true )
{
    Console.WriteLine( "Do something here");
}
```

Cấu trúc đầy đủ của if - else if - else:

// Chú ý rằng sẽ chỉ có nhiều nhất một khối được chạy
// Chương trình kiểm tra điều kiện từ trên xuống dưới khi bắt gặp một điều
// kiện đúng khối lệnh tại đó sẽ được chạy, và chương trình không kiểm tra
tiếp các điều kiện
// còn lại trong cấu trúc rẽ nhánh.

```
// Nếu điều kiện 1 đúng thì ...
if (điều kiện 1)
{
    // ... làm gì đó khi điều kiện một đúng.
}
// Ngược lại nếu điều kiện 2 đúng thì ....
else if(điều kiện 2 )
{
    // ... làm gì đó khi điều kiện 2 đúng (điều kiện 1 sai).
}
// Ngược lại nếu điều kiện N đúng thì ...
else if(điều kiện N)
{
    // .. làm gì đó khi điều kiện N đúng (các điều kiện ở trên sai)
}
// Ngược lại nếu các điều kiện ở trên đều sai thì
else
{
    // ... làm gì đó tại đây khi mọi điều kiện trên đều sai.
}
```

- Kiểm tra chuỗi nhập vào có phải là số hay không?

```
int age;  
    string inputAge = Console.ReadLine();  
    if (Int32.TryParse(inputAge, out age)){  
  
}else{ // chuỗi nhập vào phải là số}
```

Câu lệnh Switch-Case

Cú pháp câu lệnh rẽ nhánh **switch**:

```
// Sử dụng switch để kiểm tra một giá trị của một biến  
switch ( <variable> )  
{  
    case value1:  
        // Làm gì đó nếu giá trị của biến == value1  
        break;  
    case value2:  
        // Làm gì đó nếu giá trị của biến == value2  
        break;  
    ...  
    default:  
        // Làm điều gì đó tại đây nếu giá trị của biến không thuộc các giá trị liệt  
        kê ở trên.  
        break;  
}
```

Lệnh break trong trường hợp này nói với chương trình rằng thoát ra khỏi switch.

Ví dụ:

```
using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;  
using System.Threading.Tasks;  
  
namespace HelloCSharp  
{  
    class BreakExample  
    {  
        static void Main(string[] args)  
        {  
            // Đề nghị người dùng chọn 1 lựa chọn.  
            Console.WriteLine("Please select one option:\n");  
  
            Console.WriteLine("1 - Play a game \n");  
            Console.WriteLine("2 - Play music \n");  
            Console.WriteLine("3 - Shutdown computer \n");  
        }  
    }  
}
```

```

// Khai báo một biến option
int option;

// Chuỗi người dùng nhập vào từ bàn phím
string inputStr = Console.ReadLine();

// Chuyển chuỗi thành số nguyên.
option = Int32.Parse(inputStr);

// Kiểm tra giá trị của option
switch (option)
{
    case 1:
        Console.WriteLine("You choose to play the game");
        break;
    case 2:
        Console.WriteLine("You choose to play the music");
        break;
    case 3:
        Console.WriteLine("You choose to shutdown the computer");
        break;
    default:
        Console.WriteLine("Nothing to do...");
        break;
}

Console.ReadLine();
}
}
}

```

Bạn có thể gộp nhiều trường hợp (case) để thực thi cùng một khối lệnh.

```

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

namespace HelloCSharp
{
    class BreakExample2
    {
        static void Main(string[] args)
        {
            // Khai báo biến option và gán giá trị 3.
            int option = 3;

```

```

        Console.WriteLine("Option = {0}", option);

        // Kiểm tra giá trị của option
        switch (option)
        {
            case 1:
                Console.WriteLine("Case 1");
                break;
            // Trường hợp chọn 2,3,4,5 sử lý giống nhau.
            case 2:
            case 3:
            case 4:
            case 5:
                Console.WriteLine("Case 2,3,4,5!!!");
                break;
            default:
                Console.WriteLine("Nothing to do...");
                break;
        }

        Console.ReadLine();
    }
}

```

Vòng lặp trong C#

Vòng lặp (loop) được sử dụng để chạy lặp lại một khối lệnh. Nó làm chương trình của bạn thực thi lặp đi lặp lại một khối lệnh nhiều lần, đây là một trong các nhiệm vụ cơ bản trong lập trình.

C# hỗ trợ 3 loại vòng lặp khác nhau:

- **FOR**
- **WHILE**
- **DO WHILE**

Vòng lặp for

Cấu trúc của vòng lặp **for**:

```

for ( khởi tạo biến; điều kiện; cập nhập giá trị mới cho biến )
{
    // Thực thi khối lệnh khi điều kiện còn đúng
}
// Ví dụ 1:
// Tạo một biến x và gán giá trị ban đầu của nó là 0

```

```
// Điều kiện kiểm tra là x < 5
// Nếu x < 5 đúng thì khối lệnh được chạy
// Mỗi lần chạy xong khối lệnh giá trị x lại được cập nhập mới tại đây là
tăng x lên 1.
for (int x = 0; x < 5 ; x = x + 1)
{
    // Làm gì đó tại đây khi x < 5 đúng.
}
```

```
// Ví dụ 2:
// Tạo một biến x và gán giá trị ban đầu của nó là 2
// Điều kiện kiểm tra là x < 15
// Nếu x < 15 đúng thì khối lệnh được chạy
// Mỗi lần chạy xong khối lệnh giá trị x lại được cập nhập mới tại đây là
tăng x lên 3.
for (int x = 2; x < 15 ; x = x + 3)
{
    // Làm gì đó tại đây khi x < 15 đúng.
}
```

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

```
namespace HelloCSharp
{
    class ForLoopExample
    {
        static void Main(string[] args)
        {
            Console.WriteLine("For loop example");

            // Tạo một biến x và gán giá trị ban đầu của nó là 2
            // Điều kiện kiểm tra là x < 15
            // Nếu x < 15 đúng thì khối lệnh được chạy
            // Mỗi lần chạy xong khối lệnh giá trị x lại được cập nhập mới
            // tại đây là tăng x lên 3.
            for (int x = 2; x < 15; x = x + 3)
            {
                Console.WriteLine( );
                Console.WriteLine("Value of x = {0}", x);
            }

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



```
}
```

Vòng lặp while

Cú pháp của vòng lặp **while**:

```
while (điều kiện)
{
    // Trong khi điều kiện đúng thì thực thi khối lệnh.
}
```

Ví dụ:

```
// Khai báo một biến x
int x = 2;

while ( x < 10)
{
    // Làm gì đó tại đây khi x < 10 còn đúng.
    // ....
    // Cập nhập giá trị mới cho biến x.
    x = x + 3;
}
```

Ví dụ:

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

namespace HelloCSharp
{
    class WhileLoopExample
    {
        static void Main(string[] args)
        {
            Console.WriteLine("While loop example");

            // Tạo một biến x và gán giá trị ban đầu của nó là 2
            int x = 2;

            // Điều kiện kiểm tra là x < 10
            // Nếu x < 10 đúng thì khối lệnh được chạy.
            while (x < 10)
            {
                Console.WriteLine("Value of x = {0}", x);

                x = x + 3;
            }
        }
    }
}
```

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

Vòng lặp do-while

Cú pháp của vòng lặp **do-while**:

// Đặc điểm của vòng lặp DO-WHILE là nó sẽ thực khi khối lệnh ít nhất 1 lần.

// Mỗi lần chạy xong khối lệnh nó lại kiểm tra điều kiện xem có thực thi tiếp không.

do

{

// Làm gì đó tại đây

// Sau đó mới kiểm tra tiếp điều kiện xem có tiếp tục chạy khối lệnh này nữa hay không.

} while (điều kiện); // Chú ý: cần có dấu chấm phẩy tại đây.

Ví dụ:

```
using System;
```

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

```
using System.Linq;
```

```
using System.Text;
```

```
using System.Threading.Tasks;
```

```
namespace HelloCSharp
```

```
{
```

```
    class DoWhileLoopExample
```

```
    {
```

```
        static void Main(string[] args)
```

```
        {
```

```
            Console.WriteLine("Do-While loop example");
```

```
            // Tạo một biến x và gán giá trị ban đầu của nó là 2
```

```
            int x = 2;
```

```
            // Thực hiện khối lệnh.
```

```
            // Sau đó kiểm tra điều kiện xem có thực hiện khối lệnh nữa
```

không.

```
            do
```

```
            {
```

```
                Console.WriteLine("Value of x = {0}", x);
```

```
                x = x + 3;
```

```
            } while (x < 10); // Chú ý: Cần có dấu chấm phẩy tại đây.
```

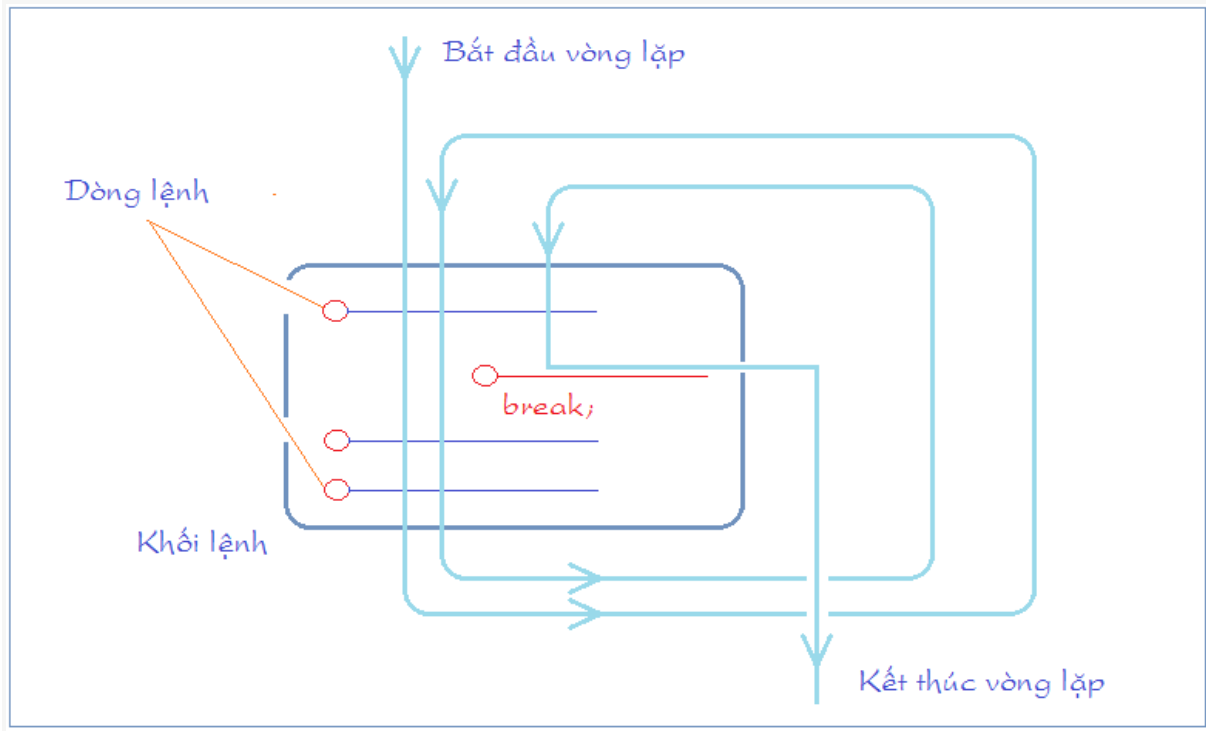
```

        Console.ReadLine();
    }
}

```

Lệnh break trong vòng lặp

break là một lệnh nó có thể nằm trong một khối lệnh của một vòng lặp. Đây là lệnh kết thúc vòng lặp vô điều kiện.



```

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

namespace HelloCSharp
{
    class LoopBreakExample
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Break example");

            // Tạo một biến x và gán giá trị ban đầu của nó là 2
            int x = 2;

```

```

while (x < 15)
{
    Console.WriteLine("-----\n");
    Console.WriteLine("x = {0}", x);

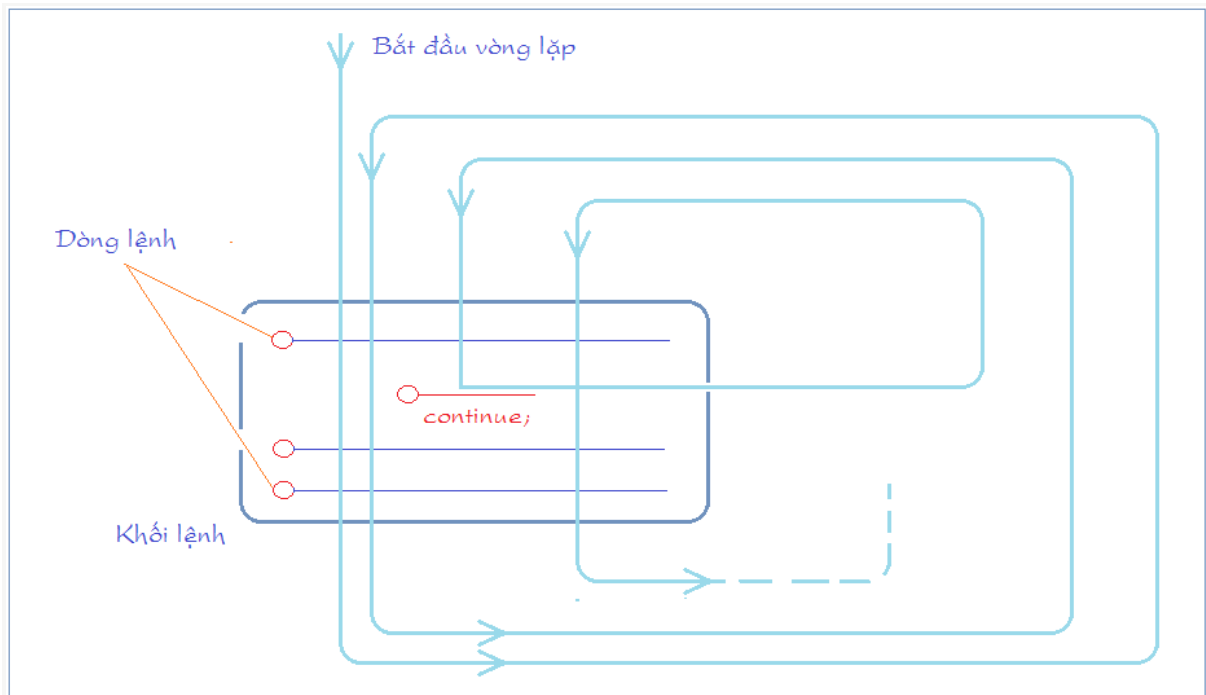
    // Kiểm tra nếu x = 5 thì thoát ra khỏi vòng lặp.
    if (x == 5)
    {
        break;
    }
    // Tăng x lên 1 (Viết ngắn gọn cho x = x + 1;).
    x++;
    Console.WriteLine("x after ++ = {0}", x);
}

Console.ReadLine();
}
}
}

```

Lệnh continue trong vòng lặp

continue là một lệnh, nó có thể nằm trong một vòng lặp, khi bắt gặp lệnh *continue* chương trình sẽ bỏ qua các dòng lệnh trong khối phía dưới của *continue* và bắt đầu một vòng lặp mới.



```

using System;
using System.Collections.Generic;
using System.Linq;

```

```

using System.Text;
using System.Threading.Tasks;

namespace HelloCSharp
{
    class LoopContinueExample
    {
        static void Main(string[] args)
        {

            Console.WriteLine("Continue example");

            // Tạo một biến x và gán giá trị ban đầu của nó là 2
            int x = 2;

            while (x < 7)
            {

                Console.WriteLine("-----\n");
                Console.WriteLine("x = {0}", x);

                // % là phép chia lấy số dư
                // Nếu x chẵn, thì bỏ qua các dòng lệnh phía dưới
                // của continue, tiếp tục vòng lặp mới (nếu có).
                if (x % 2 == 0)
                {
                    // Tăng x lên 1 (Viết ngắn gọn cho x = x + 1);
                    x++;
                    continue;
                }
                else
                {
                    // Tăng x lên 1 (Viết ngắn gọn cho x = x + 1);
                    x++;
                }
                Console.WriteLine("x after ++ = {0}", x);

            }

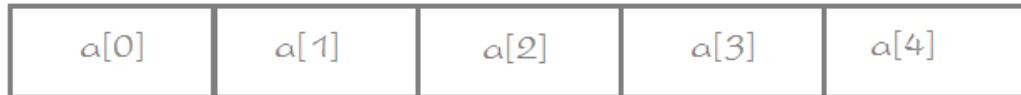
            Console.ReadLine();
        }
    }
}

```

Mảng trong C#

Mảng một chiều

```
int[] a;
```



Cú pháp khai báo mảng một chiều:

// Cách 1:

// Khai báo một mảng các số int, chỉ rõ các phần tử.

```
int[] years = { 2001, 2003, 2005, 1980, 2003 };
```

// Cách 2:

// Khai báo một mảng các số float, chỉ rõ số phần tử.

// (3 phần tử).

```
float[] salaries = new float[3];
```

Ví dụ:

```
using System;
```

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

```
using System.Linq;
```

```
using System.Text;
```

```
using System.Threading.Tasks;
```

```
namespace HelloCSharp
```

```
{
```

```
    class ArrayExample1
```

```
    {
```

```
        static void Main(string[] args)
```

```
        {
```

```
            // Cách 1:
```

```
            // Khai báo một mảng, gán luôn các giá trị.
```

```
            int[] years = { 2001, 2003, 2005, 1980, 2003 };
```

// Length là một thuộc tính của mảng, nó trả về số phần tử của mảng.

```
Console.WriteLine("Element count of array years = {0} \n",  
years.Length);
```

```
// Sử dụng vòng lặp for để in ra các phần tử của mảng.
```

```
for (int i = 0; i < years.Length; i++) {
```

```
    Console.WriteLine("Element at {0} = {1}", i, years[i]);
```

```
}
```

```

// Cách 2:
// Khai báo một mảng có 3 phần tử.
float[] salaries = new float[3];

// Gán các giá trị cho các phần tử.
salaries[0] = 1000;
salaries[1] = 1200;
salaries[2] = 1100;

Console.ReadLine();
}
}
}

```

Mảng hai chiều

		Column				
		0	1	2	3	4
Row	0	a[0,0]	a[0,1]	a[0,2]	a[0,3]	a[0,4]
	1	a[1,0]	a[1,1]	a[1,2]	a[1,3]	a[1,4]
	2	a[2,0]	a[2,1]	a[2,2]	a[2,3]	a[2,4]

Cú pháp khai báo một mảng 2 chiều:

```

// Khai báo mảng 2 chiều chỉ định các phần tử.
// 3 hàng & 5 cột

```

```

int[,] a = new int[,] {
    {1, 2, 3, 4, 5},
    {0, 3, 4, 5, 7},
    {0, 3, 4, 0, 0}
};

```

```

// Khai báo một mảng 2 chiều, số dòng 3, số cột 5.

```

```

// Các phần tử chưa được gán giá trị.

```

```

int[,] a = new int[3,5];

```

Ví dụ:

```

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

```

```

namespace HelloCSharp
{
    class ArrayExample2
    {
        static void Main(string[] args)
        {

            // Khai báo một mảng 2 chiều
            // Khởi tạo sẵn các giá trị.
            int[,] a = {
                { 1, 2, 3, 4, 5 },
                { 0, 3, 4, 5, 7 },
                { 0, 3, 4, 0, 0 }
            };

            // Sử dụng vòng lặp for để in ra các phần tử của mảng.
            for (int row = 0; row < 3; row++) {
                for (int col = 0; col < 5; col++) {
                    Console.WriteLine("Element at [{0},{1}] = {2}", row, col,
a[row,col]);
                }
                Console.WriteLine("-----");
            }

            // Khai báo một mảng 2 chiều có số dòng 3, số cột 5
            // Các phần tử chưa được gán giá trị.
            int[,] b = new int[3, 5];

            Console.ReadLine();
        }
    }
}

```

Mảng của mảng

```

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

namespace HelloCSharp
{
    class ArrayOfArrayExample
    {
        static void Main(string[] args)
        {

```



```

// Khai báo một mảng 3 phần tử.
// Mỗi phần tử là một mảng khác.
string[][] teams = new string[3][];

string[] mu = { "Beckham", "Giggs" };
string[] asenal = { "Oezil", "Szczęsny", "Walcott" };
string[] chelsea = { "Oscar", "Hazard", "Drogba" };

teams[0] = mu;
teams[1] = asenal;
teams[2] = chelsea;

// Sử dụng vòng lặp for để in ra các phần tử của mảng.
for (int row = 0; row < teams.Length; row++)
{
    for (int col = 0; col < teams[row].Length; col++)
    {
        Console.WriteLine("Element at [{0}], [{1}] = {2}", row,
col, teams[row][col]);
    }
    Console.WriteLine("-----");
}

Console.ReadLine();
}
}
}

```

Class, đối tượng và cấu tử

- Class
- Cấu tử (Constructor)

Constructors (được gọi là phương thức khởi tạo) là những phương thức đặc biệt cho phép thực thi, điều khiển chương trình ngay khi khởi tạo đối tượng. Trong C#, Constructors có tên giống như tên của Class và không có giá trị trả về.

- Đối tượng (Instance)

```

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

namespace HelloCSharp
{
    class Person
    {
        // Đây là một trường (Field)

```

```

// Lưu trữ tên người.
public string Name;

// Đây là một cấu tử, còn gọi là phương thức khởi tạo (Constructor)
// Dùng nó để khởi tạo đối tượng.
// Cấu tử này có một tham số.
// Cấu tử luôn có tên giống tên class.
public Person(string persionName)
{
    // Gán giá trị từ tham số vào cho trường name.
    this.Name = persionName;
}

// Đây là một phương thức trả về kiểu string.
public string GetName()
{
    return this.Name;
}
}

```

Như trên class Person không có phương thức Main. Tiếp theo class PersonTest là ví dụ khởi tạo các đối tượng của Person thông qua các cấu tử.

```

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

namespace HelloCSharp
{
    class PersonTest
    {
        static void Main(string[] args)
        {
            // Tạo một đối tượng từ class Person
            // Khởi tạo đối tượng này từ cấu tử của class Person
            // Cụ thể là Edison
            Person edison = new Person("Edison");

            // Class Person có hàm getName()
            // Sử dụng đối tượng để gọi hàm getName():
            String name = edison.GetName();
            Console.WriteLine("Person 1: " + name);

            // Tạo một đối tượng từ class Person.
            // Khởi tạo đối tượng này từ cấu tử của class Person

```

```

// Cụ thể là Bill Gates
Person billGate = new Person("Bill Gates");

// Class Person có trường name (public)
// Sử dụng đối tượng để tham chiếu tới nó.
String name2 = billGate.Name;
Console.WriteLine("Person 2: " + name2);

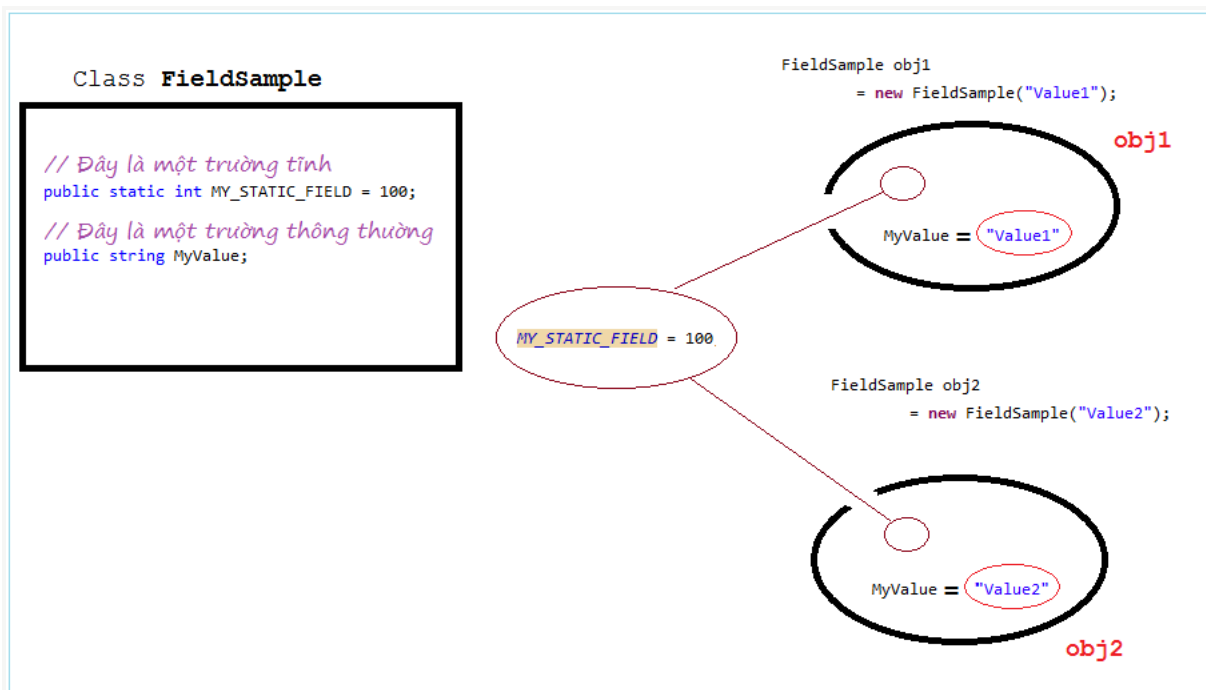
Console.ReadLine();
}
}
}

```

Trường (Field)

Trong phần tiếp theo này chúng ta sẽ thảo luận về một số khái niệm:

- Trường (Field)
 - Trường thông thường
 - Trường tĩnh (static Field)
 - Trường const (const Field)
 - Trường tĩnh và readonly (static readonly Field)



```

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

```

```

namespace HelloCSharp

```

```

{
    class FieldSample
    {
        // Đây là một trường tĩnh.
        public static int MY_STATIC_FIELD = 100;

        // Đây là một trường thông thường.
        public string MyValue;

        // Cấu tử khởi tạo đối tượng FieldSample.
        public FieldSample(string value)
        {
            this.MyValue = value;
        }
    }
}

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

namespace HelloCSharp
{
    class FieldSampleTest
    {
        static void Main(string[] args)
        {
            // In ra giá trị của trường static.
            // Với các trường tĩnh, bạn phải truy cập tới nó thông qua
class.
            Console.WriteLine("FieldSample.MY_STATIC_FIELD= {0}",
FieldSample.MY_STATIC_FIELD);

            // Bạn có thể thay đổi giá trị của trường tĩnh.
            FieldSample.MY_STATIC_FIELD = 200;

            Console.WriteLine(" ----- ");

            // Tạo đối tượng thứ nhất.
            FieldSample obj1 = new FieldSample("Value1");

            // Các trường không tĩnh bạn phải truy cập thông qua đối tượng.
            Console.WriteLine("obj1.MyValue= {0}", obj1.MyValue);

            // Tạo đối tượng thứ 2:

```

```

        FieldSample obj2 = new FieldSample("Value2");

        //
        Console.WriteLine("obj2.MyValue= {0}" , obj2.MyValue);

        // Bạn có thể thay đổi giá trị của trường.
        obj2.MyValue = "Value2-2";

        Console.ReadLine();
    }

}
}

```

Ví dụ readonly & static readonly.

```

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

namespace HelloCSharp
{
    class ConstFieldExample
    {
        // Một trường hằng số, giá trị của nó được xác định sẵn tại thời điểm
        biên dịch.
        // Trường const không cho phép gán giá trị mới.
        // Chú ý: Trường const (Đã là static).
        public const int MY_VALUE = 100;

        // Một trường tĩnh và readonly.
        // Giá trị của nó có thể gán sẵn, hoặc chỉ được gán 1 lần trong cấu
        tử tĩnh.
        public static readonly DateTime INIT_DATE_TIME1 = DateTime.Now;

        // Một trường readonly.
        // Giá trị của nó có thể gán sẵn, hoặc chỉ được gán một lần tại cấu
        tử (không tĩnh).
        public readonly DateTime INIT_DATE_TIME2 ;

        public ConstFieldExample()
        {
            // Gán giá trị cho trường readonly (Chỉ được phép gán 1 lần) .
            INIT_DATE_TIME2 = DateTime.Now;
        }
    }
}

```

Phương thức (Method)

Phương thức (Method)

- Phương thức thông thường.
- Phương thức tĩnh
- Phương thức sealed. (Sẽ được đề cập trong phần thừa kế của class).

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

namespace HelloCSharp
{
    class MethodSample
    {
        public string text = "Some text";

        // Cấu tử mặc định.
        // Nghĩa là cấu tử không có tham số.
        public MethodSample()
        {

        }

        // Đây là một phương thức trả về kiểu String.
        // Phương thức này không có tham số
        public string GetText()
        {
            return this.text;
        }

        // Đây là một phương thức có 1 tham số String.
        // Phương thức này trả về void (Hay gọi là ko trả về gì)
        public void SetText(string text)
        {
            // this.text tham chiếu tới trường text.
            // phân biệt với tham số text.
            this.text = text;
        }

        // Đây là một phương thức tĩnh.
        // Trả về kiểu int, có 3 tham số.
        public static int Sum(int a, int b, int c)
        {
            int d = a + b + c;
        }
    }
}
```

```

        return d;
    }
}
}
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace HelloCSharp
{
    class MethodSampleTest
    {
        static void Main(string[] args)
        {
            // Tạo đối tượng MethodSample
            MethodSample obj = new MethodSample();

            // Các phương thức không tĩnh cần phải được gọi thông qua đối
            tượng.
            // Gọi phương thức GetText()
            String text = obj.GetText();

            Console.WriteLine("Text = " + text);

            // Các phương thức không tĩnh cần phải được gọi thông qua đối
            tượng.
            // Gọi method SetText(String)
            obj.SetText("New Text");

            Console.WriteLine("Text = " + obj.GetText());

            // Các phương thức tĩnh cần phải được gọi thông qua Class.
            int sum = MethodSample.Sum(10, 20, 30);

            Console.WriteLine("Sum 10,20,30= " + sum);

            Console.ReadLine();
        }
    }
}

```

Note: - Các phương thức không tĩnh cần phải được gọi thông qua đối tượng.
 - Các phương thức tĩnh cần phải được gọi thông qua Class.

Thừa kế trong C#

CSharp cho phép viết class mở rộng từ một class khác. Class mở rộng từ một class khác được gọi là class con. Class con có được thừa kế các trường, thuộc tính và các method từ class cha.

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

namespace HelloCSharp
{
    // Mô phỏng một lớp động vật.
    class Animal
    {
        public Animal()
        {

        }

        public void Move()
        {
            Console.WriteLine("Move ...!");
        }

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

namespace HelloCSharp
{
    class Cat : Animal
    {

        public void Say()
        {
            Console.WriteLine("Meo");
        }

        // Một method của class Cat.
        public void Catch()
        {
            Console.WriteLine("Catch Mouse");
        }

    }
}
```



```

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

namespace HelloCSharp
{
    // Con kiến
    class Ant : Animal
    {
    }
}

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

namespace HelloCSharp
{
    class AnimalTest
    {

        static void Main(string[] args)
        {

            // Khai báo một đối tượng Cat.
            Cat tom = new Cat();

            // Kiểm tra xem 'tom' có phải là đối tượng Animal ko.
            // Kết quả rõ ràng là true.
            bool isAnimal = tom is Animal;

            // ==> true
            Console.WriteLine("tom is Animal? " + isAnimal);

            // Gọi method Catch
            tom.Catch();

            // ==> Meo
            // Gọi vào method Say() của Cat.
            tom.Say();

            Console.WriteLine("-----");

            // Khai báo một đối tượng Animal

```

```

        // Khởi tạo đối tượng thông qua cấu tử của Cat.
        Animal tom2 = new Cat();

        // Gọi method Move()
        tom2.Move();

        Console.WriteLine("-----");

        // Thông qua cấu tử của class con, Ant.
        Ant ant = new Ant();

        // Gọi method Move() thừa kế được từ Animal.
        ant.Move();

        Console.ReadLine();
    }
}

```

Thừa kế và đa hình trong C#

Thừa kế và đa hình - đây là một khái niệm vô cùng quan trọng trong CSharp. Mà bạn bắt buộc phải hiểu nó.

Class, đối tượng và cấu tử(hàm tạo - constructor)

```

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

namespace InheritancePolymorphism
{
    class Person
    {
        // Trường name - thông tin tên người
        public String Name;

        // Trường bornYear - thông tin năm sinh
        public int BornYear;

        // Nơi sinh
        public String PlaceOfBirth;

        // Cấu tử 3 tham số. Mục đích nhằm để khởi tạo các giá trị cho các
        // trường của Person.
        // Chỉ định rõ tên, năm sinh, nơi sinh.
    }
}

```

```

        public Person(String Name, int BornYear, String PlaceOfBirth)
        {
            this.Name = Name;
            this.BornYear = BornYear;
            this.PlaceOfBirth = PlaceOfBirth;
        }

        // Cấu tử 2 tham số. Mục đích khởi tạo giá trị cho 2 trường tên và
        năm sinh cho Person.
        // Nơi sinh không được khởi tạo.
        public Person(String Name, int BornYear)
        {
            this.Name = Name;
            this.BornYear = BornYear;
        }
    }
}

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

namespace InheritancePolymorphism
{
    class PersonDemo
    {
        static void Main(string[] args)
        {
            // Đối tượng: Thomas Edison.
            // Khởi tạo theo cấu tử 2 tham số.
            Person edison = new Person("Thomas Edison", 1847);

            Console.WriteLine("Info:");
            Console.WriteLine("Name: " + edison.Name);
            Console.WriteLine("Born Year: " + edison.BornYear);
            Console.WriteLine("Place Of Birth: " + edison.PlaceOfBirth);

            // Đối tượng: Bill Gates
            // Khởi tạo theo cấu tử 3 tham số.
            Person billGates = new Person("Bill Gate", 1955, "Seattle,
Washington");

            Console.WriteLine("-----");

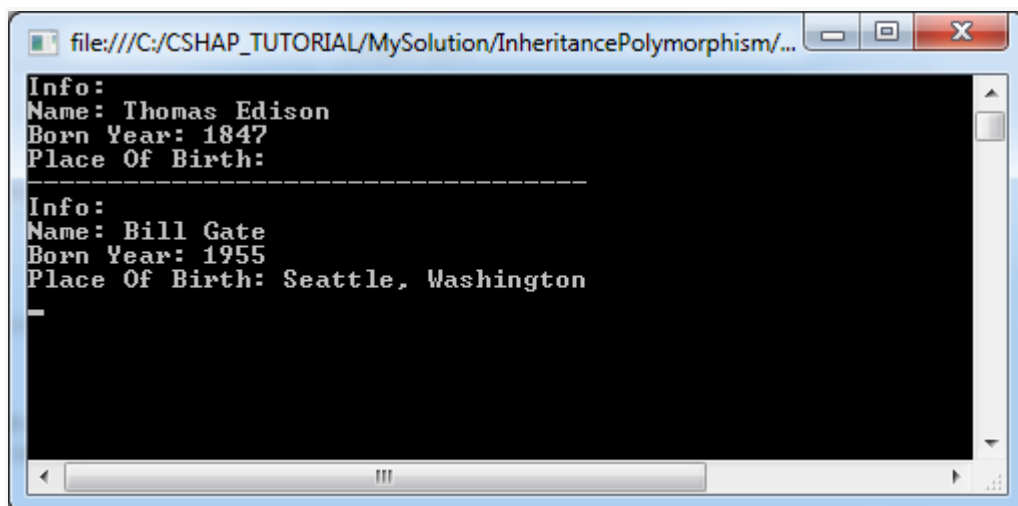
```

```

        Console.WriteLine("Info:");
        Console.WriteLine("Name: " + billGates.Name);
        Console.WriteLine("Born Year: " + billGates.BornYear);
        Console.WriteLine("Place Of Birth: " + billGates.PlaceOfBirth);

        Console.ReadLine();
    }
}
}

```



Phân biệt Class, cấu tử (constructor) và đối tượng:

Class Person mô phỏng một lớp người, nó là một thứ gì đó trừu tượng, nhưng nó có các trường để mang thông tin, trong ví dụ trên là tên, năm sinh, nơi sinh.

Constructor - Hoặc còn gọi là "Phương thức khởi tạo"

- Constructor luôn có tên giống tên class
- Một class có một hoặc nhiều constructor.
- Constructor có hoặc không có tham số, constructor không có tham số còn gọi là constructor mặc định.
- Constructor là cách để tạo ra một đối tượng của class.

Như vậy class Person (Mô tả lớp người) là thứ trừu tượng, nhưng khi chỉ rõ vào bạn hoặc tôi thì đó là 2 đối tượng thuộc class Person. Và constructor là phương thức đặc biệt để tạo ra đối tượng, constructor sẽ gán các giá trị vào các trường (field) của class cho đối tượng..

Đây là hình ảnh minh họa các trường của class được gán giá trị thế nào, khi bạn tạo đối tượng từ constructor.

```
Person edison = new Person("Thomas Edison", 1847);
```

```
class Person
{
    public String Name;
    public int BornYear;
    public String PlaceOfBirth;

    public Person(String Name, int BornYear, String PlaceOfBirth)
    {
        this.Name = Name;
        this.BornYear = BornYear;
        this.PlaceOfBirth = PlaceOfBirth;
    }

    public Person(String Name, int BornYear)
    {
        this.Name = Name;
        this.BornYear = BornYear;
    }
}
```

virtual: Nói rằng phương thức này có thể ghi đè tại các class con.

```
public virtual void Move()
{
    Console.WriteLine("Animal Move");
}
```

Thừa kế trong CSharp

Như đã biết ở phần trước, cấu tử của class (còn gọi là phương thức khởi tạo) sử dụng để tạo một đối tượng, và khởi tạo giá trị cho các trường (field).

Cấu tử của class con bao giờ cũng gọi tới một cấu tử ở class cha để khởi tạo giá trị cho các trường ở class cha, sau đó nó mới khởi tạo giá trị cho các trường của nó.

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

namespace InheritancePolymorphism
{
    public abstract class Animal
    {
        // Trường Name.
        // Tên, ví dụ Mèo Tom, Chuột Jerry.
        public string Name;

        // Cấu tử mặc định
```

```

public Animal()
{
    Console.WriteLine("- Animal()");
}

public Animal(string Name)
{
    // Gán giá trị cho trường Name.
    this.Name = Name;
    Console.WriteLine("- Animal(string)");
}

// Phương thức mô tả hành vi di chuyển của con vật.
// virtual: Nói rằng phương thức này có thể ghi đè tại các class con.
public virtual void Move()
{
    Console.WriteLine("Animal Move");
}

public void Sleep()
{
    Console.WriteLine("Sleep");
}

}
}

```

Cat là class con thừa kế từ class Animal, nó cũng có các trường của mình.

```

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

namespace InheritancePolymorphism
{
    public class Cat : Animal
    {
        public int Age;
        public int Height;

        // Đây là cấu tử 3 tham số của Cat.
        // Sử dụng :base(name) để gọi đến cấu tử của class cha:
        Animal(string) .
        // Các trường của class cha sẽ được gán giá trị.
        // Sau đó các trường của class này mới được gán giá trị.
        public Cat(string name, int Age, int Height)
            : base(name)
    }
}

```

```

    {
        this.Age = Age;
        this.Height = Height;
        Console.WriteLine("- Cat(string,int,int)");
    }

    // Cấu tử này gọi tới cấu tử mặc định (Không tham số) của class cha.
    public Cat(int Age, int Height)
        : base()
    {
        this.Age = Age;
        this.Height = Height;
        Console.WriteLine("- Cat(int,int)");
    }

    public void Say()
    {
        Console.WriteLine("Meo");
    }

    // Viết lại hành vi di chuyển của loài Mèo.
    // Ghi đè phương thức Move() của class cha (Animal).
    public override void Move()
    {
        Console.WriteLine("Cat Move ...");
    }
}

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

namespace InheritancePolymorphism
{
    class CatTest
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Create Cat object from Cat(string,int,int)");

            // Khởi tạo một đối tượng Cat từ cấu tử 3 tham số.
            // Trường Name của Animal sẽ được gán giá trị "Tom".
            // Trường Age của Cat sẽ được gán giá trị 3

```

```

// Trường Height của Cat sẽ được gán giá trị 20.
Cat tom = new Cat("Tom",3, 20);

Console.WriteLine("-----");

Console.WriteLine("Name = {0}", tom.Name);
Console.WriteLine("Age = {0}", tom.Age);
Console.WriteLine("Height = {0}", tom.Height);

Console.WriteLine("-----");

// Gọi method thừa kế từ Animal
tom.Move();

// Gọi method Say() (của class Cat)
tom.Say();

Console.ReadLine();
}
}
}

```

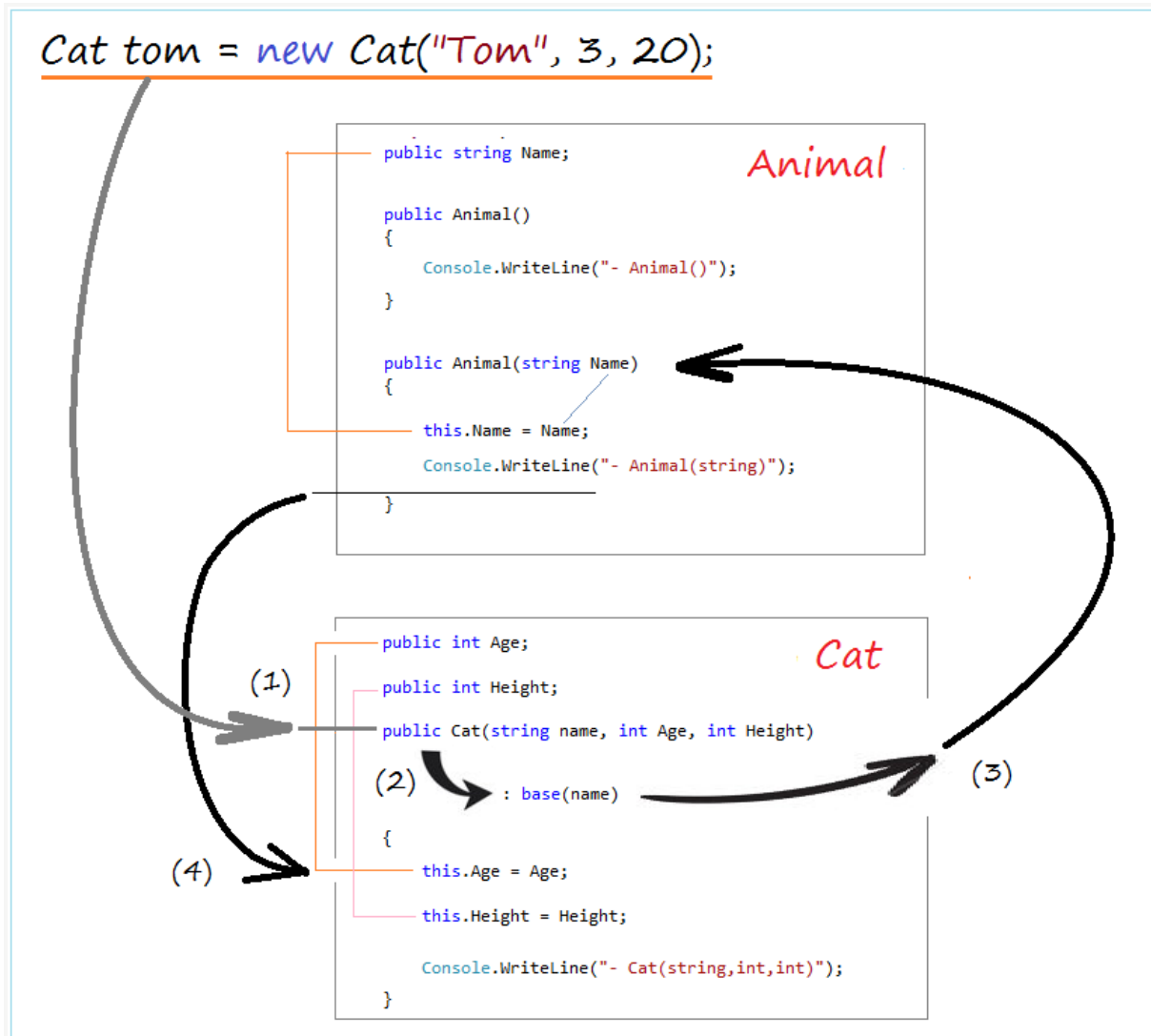
```

file:///C:/CSHAP_TUTORIAL/MySolution/InheritancePolymorphism/bin/Debug/InheritancePolymor...
Create Cat object from Cat(string,int,int)
- Animal(string)
- Cat(string,int,int)
-----
Name = Tom
Age = 3
Height = 20
-----
Sleep
Meo

```

Điều gì đã xảy ra khi bạn khởi tạo một đối tượng từ một cấu tử. Nó sẽ gọi lên một cấu tử của class cha như thế nào? Bạn hãy xem hình minh họa dưới đây:

Cat tom = new Cat("Tom", 3, 20);



Với hình minh họa trên bạn thấy rằng, cấu tử của class cha bao giờ cũng được gọi trước cấu tử của class con, nó sẽ gán giá trị cho các trường của class cha trước, sau đó các trường của class con mới được gán giá trị.

Khi bạn viết một cấu tử không khai báo rõ ràng nó base từ cấu tử nào của class cha, CSharp tự hiểu là cấu tử đó base từ cấu tử mặc định của class cha.

// Cấu tử này không ghi rõ base từ cấu tử nào của class cha.

```
public Cat(int Age, int Height)
{
}
}
```

// Nó sẽ tương đương với:

```
public Cat(int Age, int Height) : base()
{
}
}
```

Một cấu tử có thể gọi tới một cấu tử khác sử dụng `:this`.

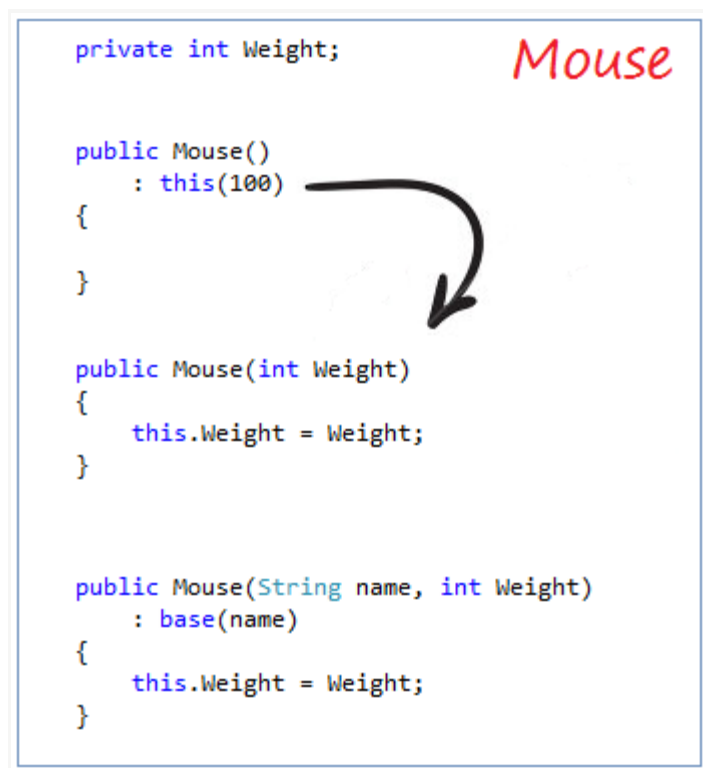
```

private int Weight;

// Cấu tử mặc định (Không có tham số).
// Gọi tới cấu tử Mouse(int)
public Mouse()    : this(100)
{
}

// Cấu tử 1 tham số.
// Không ghi rõ :base
// Nghĩa là base từ cấu tử mặc định của class cha
public Mouse(int Weight)
{
    this.Weight = Weight;
}

```



```

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

namespace InheritancePolymorphism
{
    public class Mouse : Animal
    {
        private int Weight;
    }
}

```

```

// Cấu tử mặc định (Không có tham số) .
// Gọi tới cấu tử Mouse(int)
public Mouse()
    : this(100)
{
}

// Cấu tử 1 tham số.
// Không ghi rõ :base
// Nghĩa là base từ cấu tử mặc định của class cha
public Mouse(int Weight)
{
    this.Weight = Weight;
}

// Cấu tử 2 tham số
public Mouse(String name, int Weight)
    : base(name)
{
    this.Weight = Weight;
}

public int GetWeight()
{
    return Weight;
}

public void SetWeight(int Weight)
{
    this.Weight = Weight;
}
}
}

```

Sử dụng toán tử 'is' bạn có thể kiểm tra một đối tượng có phải là kiểu của một class nào đó hay không. Hãy xem ví dụ dưới đây:

```

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

namespace InheritancePolymorphism
{
    class IsOperatorDemo
    {
        static void Main(string[] args)

```

```

{
    // Khởi tạo một đối tượng động vật.
    // Animal là class trừu tượng
    // nó ko thể tạo đối tượng từ cấu tử của nó.
    Animal tom = new Cat("Tom", 3, 20);

    Console.WriteLine("Animal Sleep:");
    // Gọi phương thức Sleep() của Animal
    tom.Sleep();

    // Sử dụng toán tử 'is' để kiểm tra xem
    // một đối tượng có phải kiểu nào đó không.
    bool isMouse = tom is Mouse; // false

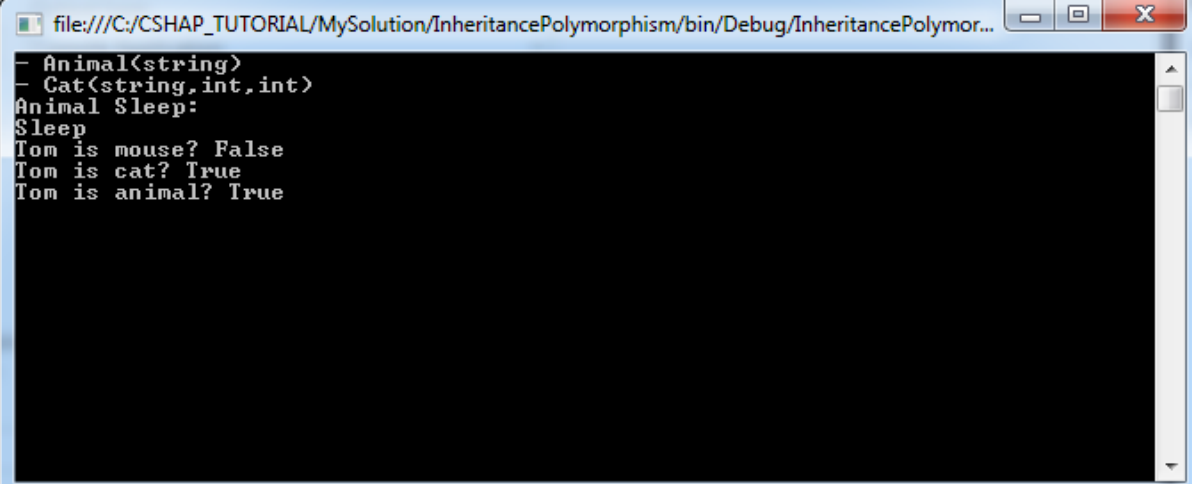
    Console.WriteLine("Tom is mouse? " + isMouse);

    bool isCat = tom is Cat; // true
    Console.WriteLine("Tom is cat? " + isCat);

    bool isAnimal = tom is Animal; // true
    Console.WriteLine("Tom is animal? " + isAnimal);

    Console.ReadLine();
}
}
}

```



```

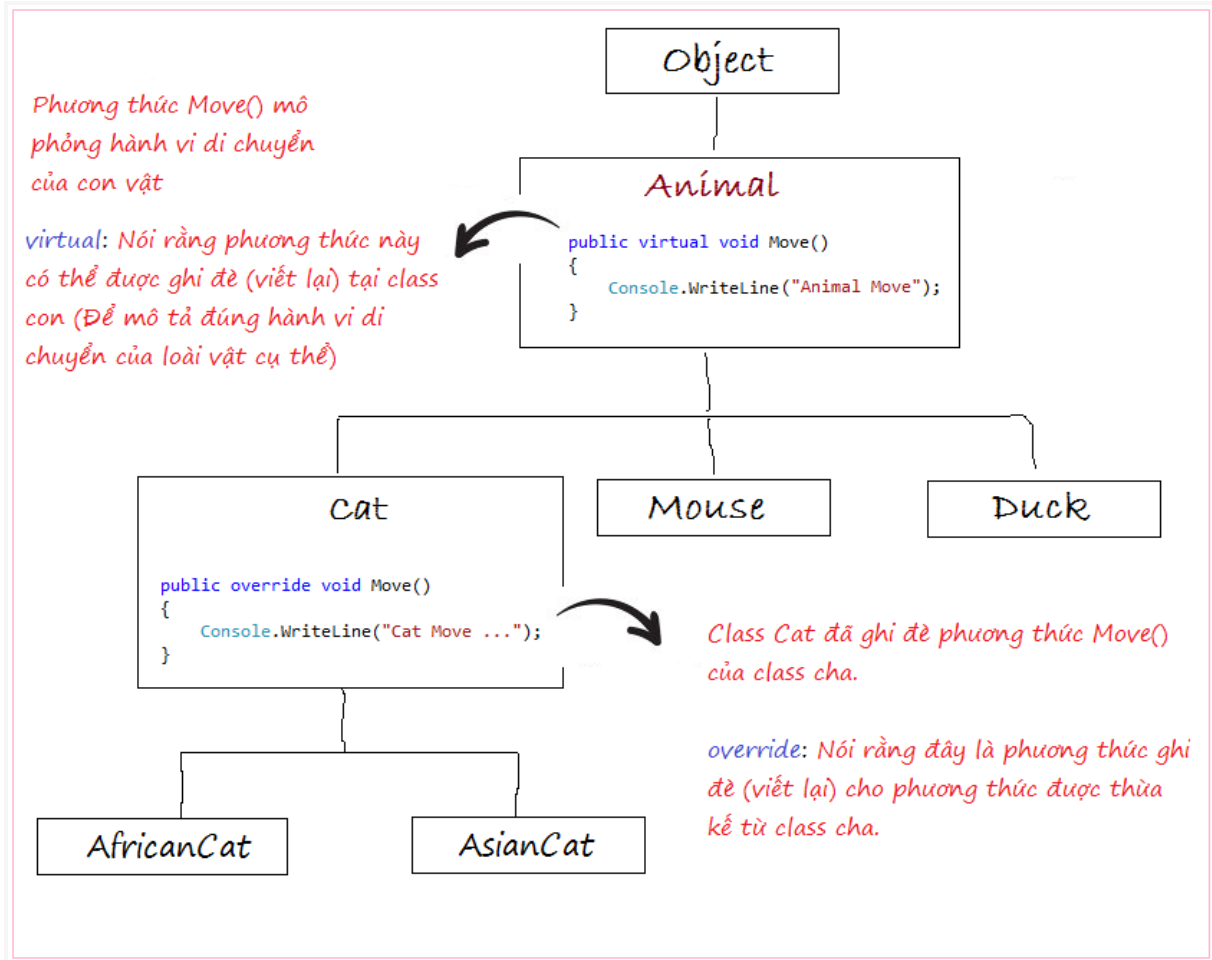
file:///C:/CSHAP_TUTORIAL/MySolution/InheritancePolymorphism/bin/Debug/InheritancePolymor...
- Animal<string>
- Cat<string,int,int>
Animal Sleep:
Sleep
Tom is mouse? False
Tom is cat? True
Tom is animal? True

```

Đa hình trong CSharp

Đa hình (Polymorphism) từ này có nghĩa là có nhiều hình thức. Trong mô hình lập trình hướng đối tượng, đa hình thường được diễn tả như là "một giao diện, nhiều chức năng".

Đa hình có thể là tĩnh hoặc động. Trong đa hình tĩnh, phản ứng với một chức năng được xác định tại thời gian biên dịch. Trong đa hình động, nó được quyết định tại thời gian chạy (run-time).



```

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

namespace InheritancePolymorphism
{
    class PolymorphismCatDemo
    {
        static void Main(string[] args)
        {
            // Bạn khai báo một đối tượng là một động vật (Animal).
            // Bằng cách tạo nó bởi cấu từ của class Cat.

            // Đối tượng 'tom' khai báo là Animal
            // vì vậy nó chỉ có thể gọi các phương thức của Animal.

            Animal tom = new Cat("Tom", 3, 20);
        }
    }
}
  
```

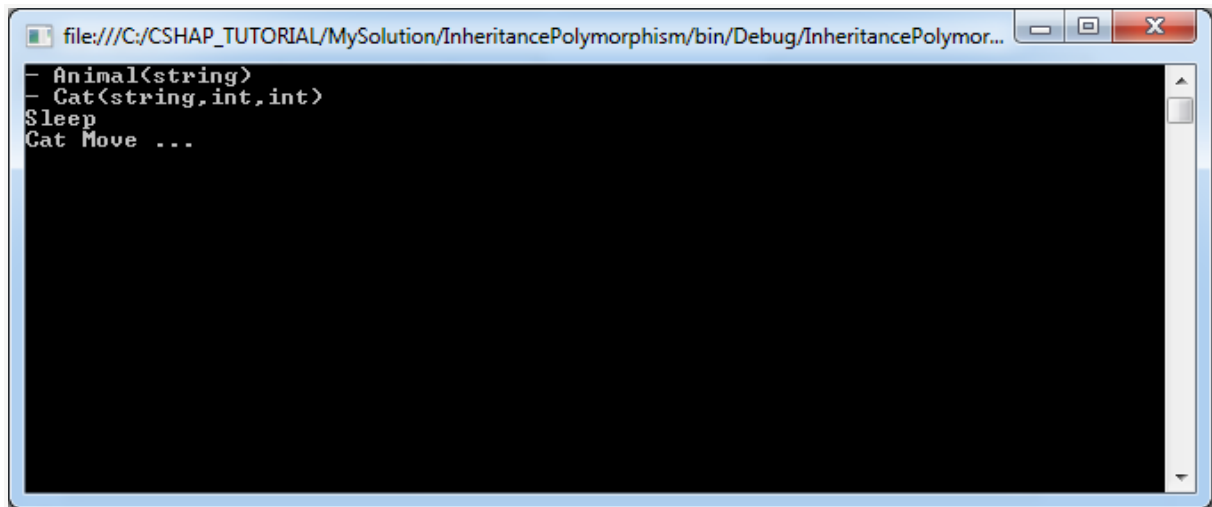
```

        // Gọi method Sleep Animal
        tom.Sleep();

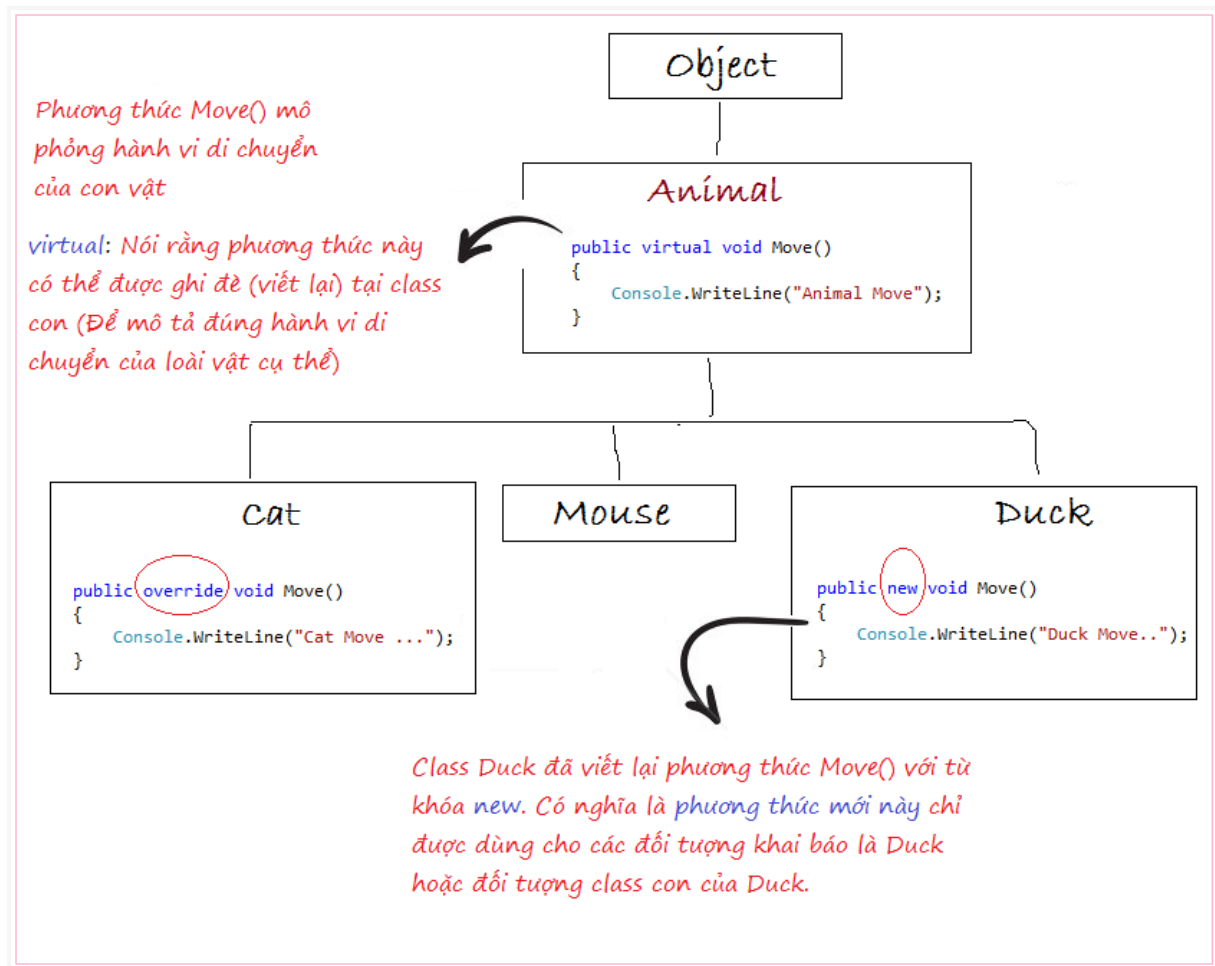
        // Gọi method Move()
        // Move() là phương thức có trong Animal.
        // Move() được ghi đè trong class Cat.
        // 'tom' thực tế là Cat, nó sẽ gọi hàm viết đè trong Cat.
        tom.Move(); // ==> Cat Move.

        Console.ReadLine();
    }
}

```



Bạn có 2 cách để ghi đè một phương thức từ class cha là **override** và **new**. Hãy xem hình minh họa dưới đây:



```

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

```

```

namespace InheritancePolymorphism
{
    class Duck : Animal
    {
        public Duck(string name)
            : base(name)
        {
            Console.WriteLine("- Duck(string)");
        }

        public new void Move()
        {
            Console.WriteLine("Duck Move..");
        }
    }
}

```

```

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

namespace InheritancePolymorphism
{
    class InheritanceDuckDemo
    {
        static void Main(string[] args)
        {
            // Bạn khai báo một đối tượng là một động vật (Animal).
            // Bằng cách tạo nó bởi cấu tử của class Duck (Một con vịt).

            // Đối tượng 'donald' khai báo là Animal
            // vì vậy nó chỉ có thể gọi các phương thức của Animal.

            Animal donald = new Duck("Donald");

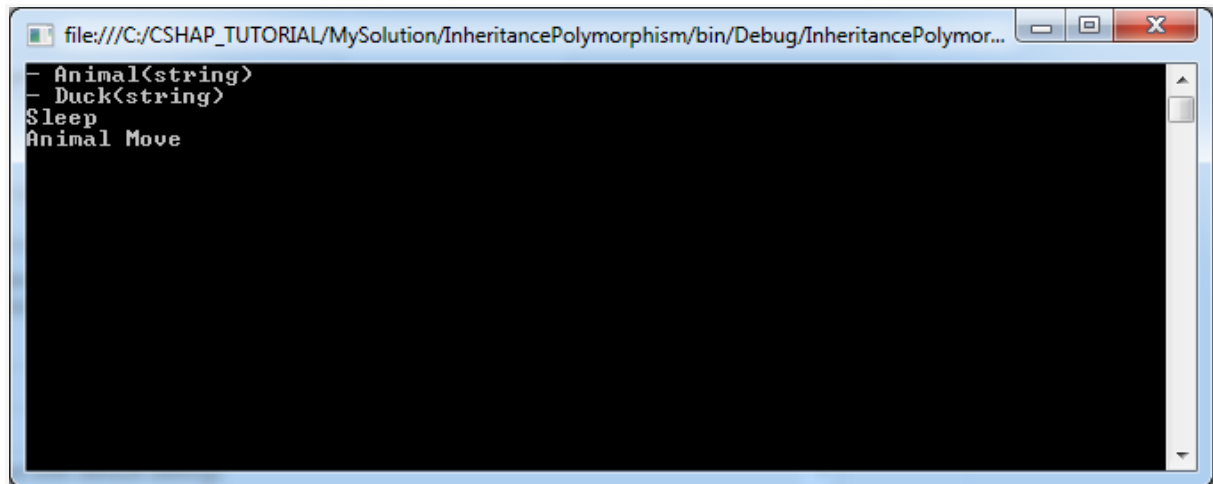
            // Gọi method Sleep của Animal
            donald.Sleep();

            // Gọi method Move()
            // Move() là phương thức có trong Animal.
            // Move() được ghi đè trong class Duck (Theo từ khóa new,
            // vì vậy chỉ các đối tượng khai báo là Duck hoặc con của Duck
            mới được dùng).

            // 'donald' thực tế là Duck, nhưng nó đang được khai báo là
            Animal.
            // (Phương thức Move() thừa kế từ Animal sẽ được gọi).
            donald.Move(); // ==> Animal Move.

            Console.ReadLine();
        }
    }
}

```

Abstract class và Interface trong C#

Giống nhau:

- Abstract class và interface đều không thể khởi tạo đối tượng bên trong được.
- Abstract class và interface đều có thể khai báo các phương thức nhưng không thực hiện chúng.
- Abstract class và interface đều bao gồm các phương thức abstract.
- Abstract class và interface đều được thực thi từ các class con hay còn gọi kế thừa, dẫn xuất.
- Abstract class và interface đều có thể kế thừa từ nhiều interface.

Khác nhau:

Abstract Class	Interface
Cho phép khai báo field	Không cho phép
Các phương thức có thể có thân hàm hoặc không có thân hàm.	Chỉ khai báo không có thân hàm
Class dẫn xuất chỉ kế thừa được từ 1 abstract class và nhiều interface.	Class triển khai có thể triển khai nhiều interface.
Có chứa constructor	Không có
Các phương thức có từ khóa access modifier	Không có

Abstract class

Abstract class giống như 1 type, tức là 1 kiểu chung cho các class dẫn xuất, các class dẫn xuất thuộc về 1 kiểu đó, nó sẽ cung cấp các thuộc tính và hành vi chung nhất cho kiểu đó.

```
abstract class Person
{
    protected string _name;

    protected DateTime _birthDate;

    protected void Say()
    {
        Console.WriteLine("Person is saying....");
    }

    protected abstract void Work();
}

class Employee : Person
{
    private float _salary;

    protected override void Work()
    {
        throw new NotImplementedException();
    }
}

class Student : Person
{
    private float _mark;

    protected override void Work()
    {
        throw new NotImplementedException();
    }
}
```

```
}
```

```
}
```

Chúng ta có abstract class `Person` chứa các thông tin chung cho người như tên hay ngày sinh nhật. Còn lại các class dẫn xuất như `Employee` hay `Student` sẽ có các tính chất riêng của nó thì khai báo thêm. Còn những gì có thể dùng chung sẽ được khai báo trong `Person`, như vậy sẽ giảm thiểu trùng lặp code, và dễ bảo trì hơn.

Vậy chúng ta hiểu rằng mối quan hệ giữa Abstract class và derived class là mối quan hệ is-a tức là mối quan hệ thuộc về tập cha con. Nên một con chỉ thuộc về 1 cha thôi, trong đó cha quy định các gen di truyền chung nhất cho các con, các con có quyền bổ sung thêm, có quyền thể hiện hoặc không thể hiện một đặc tính hay 1 hành động của cha.

Đối với Interface

Khác với Abstract class giống một bản thiết kế của toàn class thì interface chỉ là bản thiết kế của các chi tiết method. Mối quan hệ giữa lớp thực thi và interface là mối quan hệ can-do vì đơn thuần, class triển khai interface nó phải triển khai toàn bộ method của interface đã định nghĩa. Vậy là class đó có thể làm những gì interface quy định.

Như vậy hoàn toàn 1 class có thể can-do nhiều method ở nhiều interface khác nhau, ví dụ:

```
interface IRunnable
```

```
{
```

```
void Run();
```

```
}
```

```
interface IWorkable
```

```
{
```

```
void Work();
```

```
}
```

```
interface IEatable
```

```
{
```

```
void Eat();
```

```
}
```

```
class Person : IRunnable, IWorkable, IEatable
```

```
{
```

```
protected string _name;
```

```
protected DateTime _birthDate;
```

```

public void Run()
{
    Console.Write("Person runing...");
}

public void Work()
{
    Console.Write("Person working...");
}

public void Eat()
{
    Console.Write("Person eating...");
}
}

```

Chúng ta thấy cùng là con người nhưng có thể can-do tức là có thể thực hiện nhiều hành động, các interface chỉ định nghĩa tên các hành động, để các class implement triển khai chi tiết, tức triển khai interface nào có thể can-do được các hành động của interface đó.

Vậy sự khác nhau chính giữa abstract class và interface ở đây chính là mối quan hệ giữa class dẫn xuất và chúng. Đồng thời khác nhau ở mục đích sử dụng của chúng.

```

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

namespace AbstractClassInterface
{
    // Một class có ít nhất một phương thức trừu tượng,
    // phải khai báo là trừu tượng (abstract).
    public abstract class ClassA
    {
        // Đây là một method trừu tượng.
    }
}

```

```

// Nó không có thân hàm.
// Method này có access modifier là: public
// (access modifier: Độ truy cập).
public abstract void DoSomething();

// Method này có access modifier là protected
protected abstract String DoNothing();

protected abstract void Todo();
}

// Đây là một class trừu tượng.
// Chủ động khai báo abstract, mặc dù nó không có method trừu tượng nào.
public abstract class ClassB
{

}
}

```

Đặc điểm của một class trừu tượng là:

1. Nó được khai báo với từ khóa **abstract**.
2. Nó có thể khai báo 0, 1 hoặc nhiều method trừu tượng bên trong.
3. Bạn không thể khởi tạo 1 đối tượng trực tiếp từ một class trừu tượng.

Interface

Chúng ta biết rằng một class chỉ có thể mở rộng từ một class khác.

```

// Class B là con của class A, hay nói là B mở rộng từ A
// CSharp chỉ cho phép một class mở rộng từ duy nhất một class khác.
public class B : A
{
    // ....
}

// Trong trường hợp bạn không viết mở rộng từ một class nào.
// CSharp tự hiểu là nó thừa kế từ class Object.
public class B
{

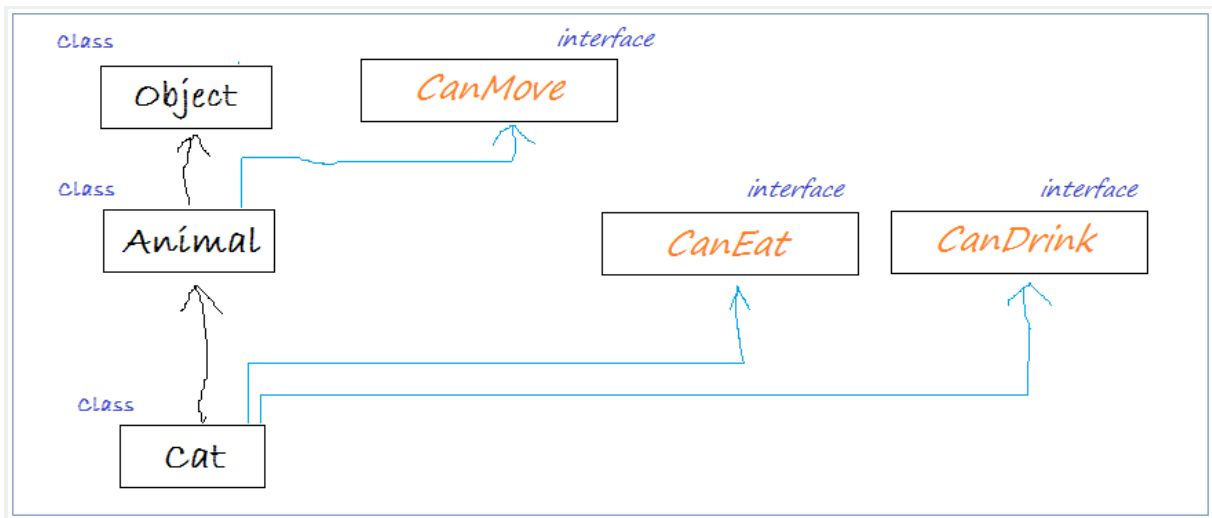
}

// Cách khai báo class này, và cách trên là tương đương nhau.
public class B : Object
{

}

```

Nhưng một class có thể mở rộng từ nhiều Interface



```
// Một class có thể mở rộng từ duy nhất một class
// Nhưng có thể mở rộng từ nhiều Interface.
public class Cat : Animal, CanEat, CanDrink
{
    // ....
}
```

Các đặc điểm của interface trong CSharp.

1. Interface có modifier là public hoặc internal, nếu không ghi rõ mặc định là internal.
2. Interface không thể định nghĩa các trường (Field).
3. Các method của nó đều là method trừu tượng (abstract) và công khai (public), và không có thân hàm. Nhưng khi khai báo phương thức bạn lại không được phép ghi **public** hoặc **abstract**.
4. Interface không có cấu tử (Constructor).

Cấu trúc của một Interface

Một interface trong **CSharp** có thể khai báo modifier là **public** hoặc **internal**, nếu không khai báo gì mặc định được hiểu là **internal**. Interface có modifier là **public** có thể được sử dụng ở mọi nơi, đối với interface có modifier là **internal** chỉ được sử dụng trong nội bộ Assembly.

Một Assembly là chính là sản phẩm đã biên dịch (compile) của mã của bạn, thường là một DLL, nhưng EXE cũng có thể coi là một assembly. Nó là đơn vị nhỏ nhất của việc triển khai cho bất kỳ dự án .NET nào.

Assembly một cách cụ thể chứa mã **.NET** theo **MSIL (Microsoft Intermediate language - Một ngôn ngữ trung gian)** sẽ được biên dịch thành mã máy (Native code) ("JITted" - biên dịch bởi các trình biên dịch Just-In-Time) trong lần đầu tiên nó được thực thi trên máy tính,. Đó là mã đã được biên dịch cũng sẽ được lưu trữ trong **Assembly** và tái sử dụng cho các lần gọi tiếp theo.

```
using System;
```

```

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

namespace AbstractClassInterface
{
    // Đây là một Interface không khai báo access modifier.
    // Mặc định modifier của nó là 'internal'.
    // Nó chỉ được dùng trong nội bộ một Assembly.
    interface NoAccessModifierInterface
    {

    }

}

```

Các phương thức trong Interface đều là công khai (public) và trừu tượng (abstract). Nó không có thân, nhưng bạn không được phép viết **public** hoặc **abstract** khi định nghĩa phương thức

```

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

namespace AbstractClassInterface
{
    // Interface này định nghĩa những thứ có khả năng di chuyển.
    public interface CanMove
    {

        // (Chạy)
        // Các method trong Interface đều là công khai và trừu tượng (public
        abstract)
        // (Nhưng bạn không được phép viết public hoặc abstract ở đây)
        void Run();

        // (Quay trở lại)
        // Cho dù không viết rõ public abstract thì CSharp luôn hiểu là vậy.
        void Back();

        // (Lấy ra vận tốc chạy).
        int GetVelocity();

    }

}

```

```

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

namespace AbstractClassInterface
{
    // Interface này định nghĩa những thứ có khả năng biết uống.
    public interface CanDrink
    {
        void Drink();
    }
}

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

namespace AbstractClassInterface
{
    // Interface này định nghĩa thứ có khả năng biết ăn.
    public interface CanEat
    {
        void Eat();
    }
}

```

Class thi hành Interface

Khi một class thi hành một Interface, **bạn phải thực hiện hoặc khai báo lại toàn bộ các phương thức có trong Interface đó.**

1. Nếu bạn triển khai một phương thức nào đó của interface, bạn phải viết nội dung cho phương thức, khai báo phương thức là public.
2. Nếu bạn không triển khai một phương thức nào đó của interface, bạn phải khai báo lại nó trong class với từ khóa '**public abstract**' và không được viết nội dung của phương thức.

```

using System;
using System.Collections.Generic;
using System.Linq;

```



```

using System.Text;
using System.Threading.Tasks;

namespace AbstractClassInterface
{
    // Animal (Class mô phỏng lớp động vật)
    // Nó mở rộng từ class Object (Mặc dù không ghi rõ).
    // Và khai báo thi hành (hoặc gọi là thừa kế) interface CanMove.
    // Interface CanMove có 3 method trừu tượng.
    // Class này mới triển khai 1 method
    // Vì vậy nó bắt buộc phải khai báo abstract
    // Các method còn lại phải được khai báo lại với 'public abstract'.
    public abstract class Animal : CanMove
    {
        // Triển khai method Run() từ interface CanMove.
        // Bạn phải viết nội dung của phương thức.
        // Modifier phải là public.
        public void Run()
        {
            Console.WriteLine("Animal run...");
        }

        // Nếu bạn không triển khai một phương thức nào đó của Interface
        // bạn phải viết lại nó dưới dạng một phương thức trừu tượng.
        // (Luôn luôn là public abstract)
        public abstract void Back();

        // Nếu bạn không triển khai một phương thức nào đó của Interface
        // bạn phải viết lại nó dưới dạng một phương thức trừu tượng.
        // (Luôn luôn là public abstract)
        public abstract int GetVelocity();
    }
}

```

```

}

```

Class Cat thừa kế từ class Animal đồng thời thực hiện 2 interface *CanDrink*, *CanEat*.

```

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

namespace AbstractClassInterface
{

```

```

// Class Cat mở rộng từ class Animal và thi hành 2 interface CanEat,
CanDrink.
public class Cat : Animal, CanEat, CanDrink
{

    private String name;

    public Cat(String name)
    {
        this.name = name;
    }

    public String getName()
    {
        return this.name;
    }

    // Triển khai phương thức trù tượng của Animal.
    // (Phải ghi rõ 'override' ).
    public override void Back()
    {
        Console.WriteLine(name + " cat back ...");
    }

    // Triển khai phương thức trù tượng của Animal
    // (Phải ghi rõ 'override' )
    public override int GetVelocity()
    {
        return 110;
    }

    // Triển khai method của interface CanEat
    public void Eat()
    {
        Console.WriteLine(name + " cat eat ...");
    }

    // Triển khai method của interface CanDrink
    public void Drink()
    {
        Console.WriteLine(name + " cat drink ...");
    }

}

}
using System;

```

```

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

namespace AbstractClassInterface
{
    public class Mouse : Animal, CanEat, CanDrink
    {

        // Triển khai phương thức trau tượng của Animal.
        // (Phải có từ khóa 'override').
        public override void Back()
        {
            Console.WriteLine("Mouse back ...");
        }

        // Triển khai phương thức trau tượng của Animal.
        // (Phải có từ khóa 'override').
        public override int GetVelocity()
        {
            return 85;
        }

        // Triển khai phương thức của interface CanDrink.
        public void Drink()
        {
            Console.WriteLine("Mouse drink ...");
        }

        // Triển khai phương thức của interface CanEat.
        public void Eat()
        {
            Console.WriteLine("Mouse eat ...");
        }

    }

}

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

namespace AbstractClassInterface
{

```

```

public class AnimalDemo
{

    public static void Main(string[] args)
    {

        // Khởi tạo một đối tượng CanEat
        // Một đối tượng khai báo là CanEat
        // Nhưng thực tế là Cat.
        CanEat canEat1 = new Cat("Tom");

        // Một đối tượng khai báo là CanEat
        // Nhưng thực tế là Mouse.
        CanEat canEat2 = new Mouse();

        // Tính đa hình thể hiện rõ tại đây.
        // CSharp luôn biết một đối tượng là kiểu gì
        // ==> Tom cat eat ...
        canEat1.Eat();

        // ==> Mouse eat ...
        canEat2.Eat();

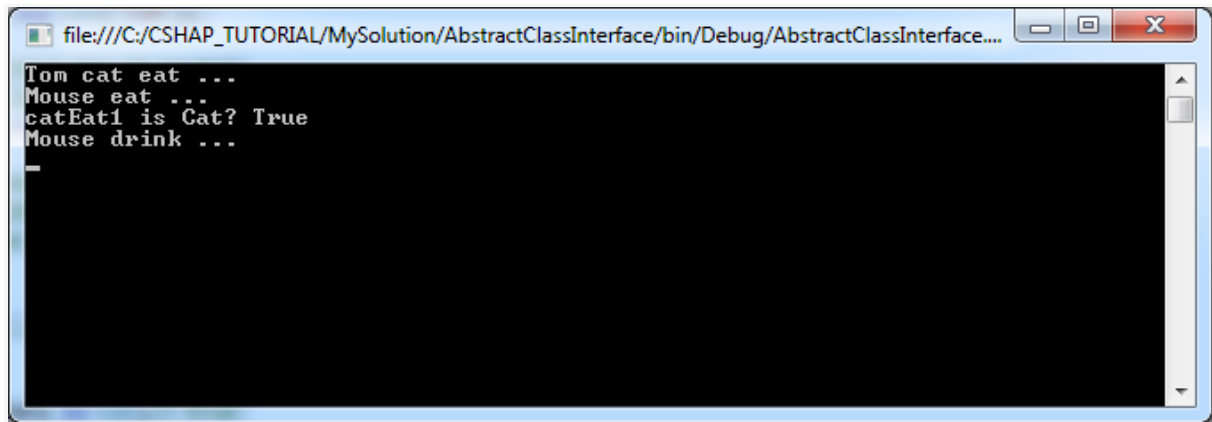
        bool isCat = canEat1 is Cat;// true

        Console.WriteLine("catEat1 is Cat? " + isCat);

        // Kiểm tra 'canEat2' có phải là chuột hay không?.
        if (canEat2 is Mouse)
        {
            // Ép kiểu
            Mouse mouse = (Mouse)canEat2;
            // Gọi method Drink (Thừa kế từ CanDrink).
            mouse.Drink();
        }

        Console.ReadLine();
    }
}

```



```
file:///C:/CSHAP_TUTORIAL/MySolution/AbstractClassInterface/bin/Debug/AbstractClassInterface...
Tom cat eat ...
Mouse eat ...
catEat1 is Cat? True
Mouse drink ...
-
```

Access Modifier trong C#

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

Có 5 kiểu của CSharp access modifiers:

1. private
2. protected
3. internal
4. protected internal
5. public

Tổng quan về access modifier

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)

Bảng minh họa dưới đây cho bạn cái nhìn tổng quan về cách sử dụng các **access modifier**.

	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

private access modifie

private access modifier chỉ cho phép truy cập trong nội bộ một class.

```
namespace AccessModifierTutorial
{
```

```
    class Person
```

```
    {
        private string Name; private field
```

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

```
        private method
```

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

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

```
    class Diary
```

```
    {
        public void Logging()
        {
            (private static)
```

```
            DoSomething("Code CSharp");
```

```
            ShowSecret();
```

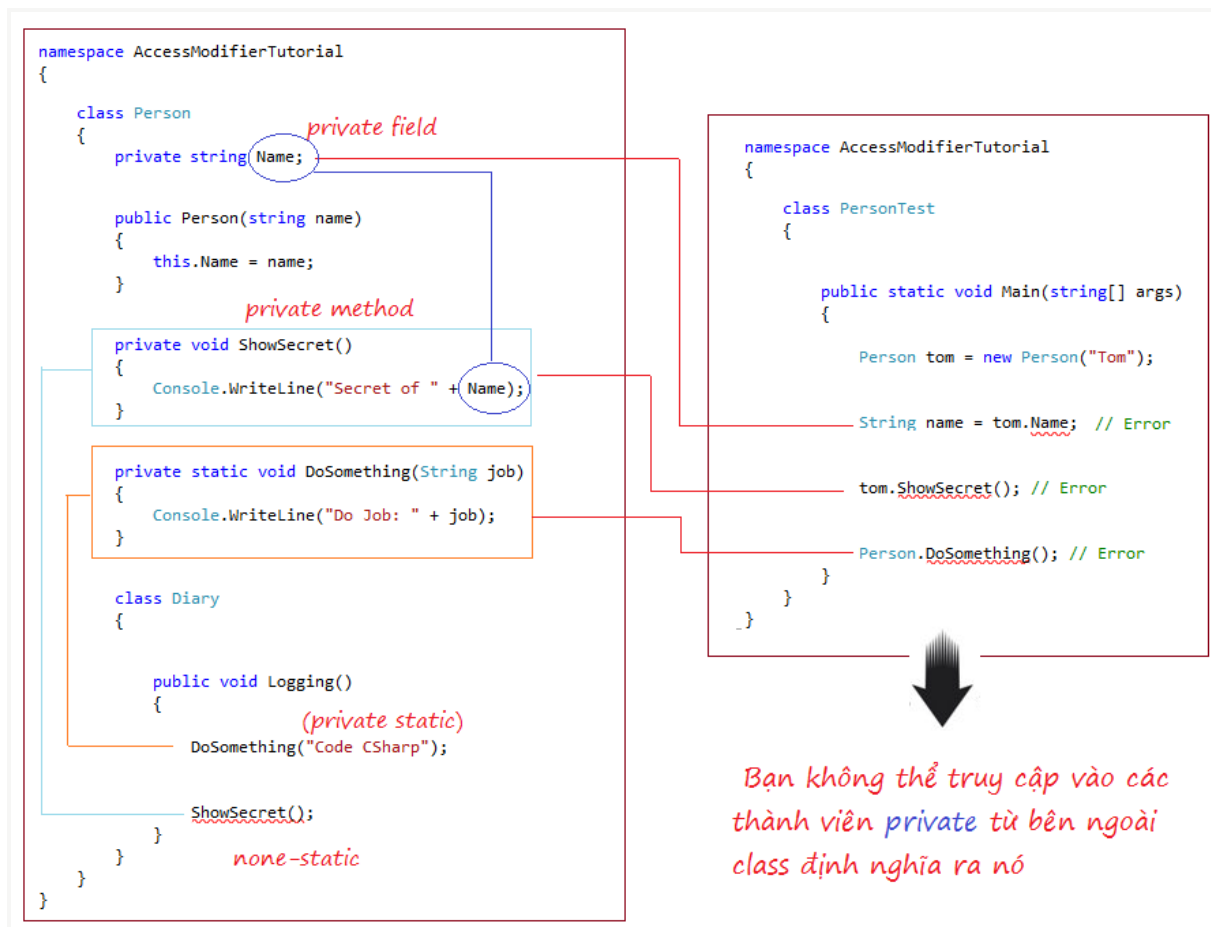
```
        }
    }
}
```

Các thành viên *private* có thể được truy cập mọi nơi trong class đã định nghĩa ra nó

Các thành viên *private static* cũng có thể được truy cập trong inner class.

Lỗi do phương thức này không phải static

Bạn không thể truy cập vào các thành viên **private** ở bên ngoài class định nghĩa thành viên đó. **CSharp** sẽ thông báo lỗi tại thời điểm biên dịch class.



private constructor

Phương thức khởi tạo (constructor), phương thức (method), trường (field) đều được gọi là các thành viên trong class.

Nếu bạn tạo một class, và có một phương thức khởi tạo (constructor) có access modifier là private, bạn không thể tạo một đối tượng của class này từ private constructor đó từ bên ngoài class này.


```
namespace AccessModifierTutorial
{
    class Animal
    {
        private String Name;

        private Animal(String name)
        {
            this.Name = name;
        }

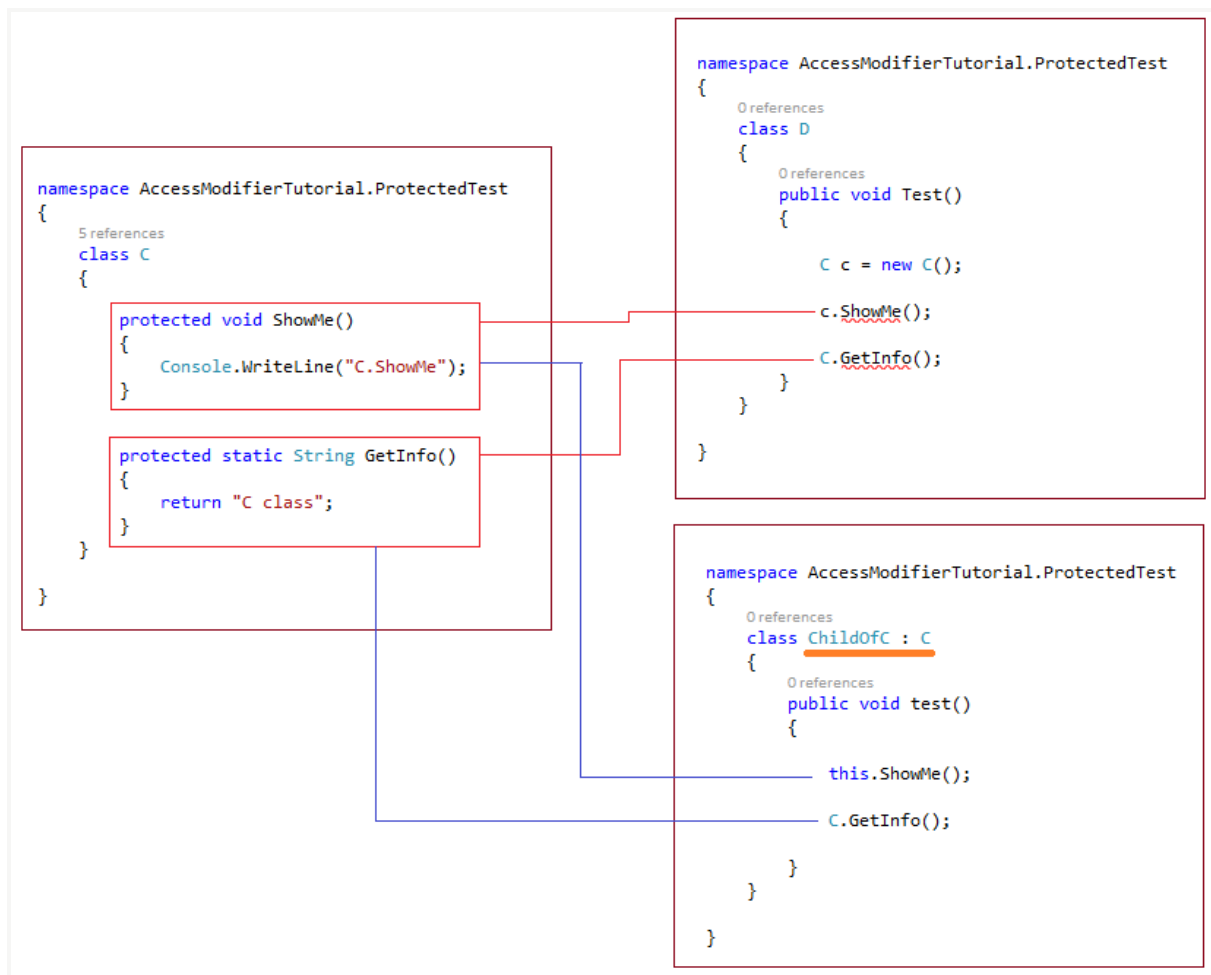
        public String GetName()
        {
            return this.Name;
        }
    }
}
```

```
namespace AccessModifierTutorial
{
    class AnimalTest
    {
        public static void Main(string[] args)
        {
            Animal a = new Animal("Tiger");
        }
    }
}
```

protected access modifier

protected access modifier có thể truy cập bên trong package, hoặc bên ngoài package nhưng phải thông qua tính kế thừa.

protected access modifier chỉ áp dụng cho field, method và constructor. Nó không thể áp dụng cho class (class, interface, ..).



internal access modifier

internal là độ truy cập nội bộ, nó bị giới hạn trong một Assembly.

Một **Assembly** là chính là sản phẩm đã biên dịch của mã của bạn, thường là một **DLL**, nhưng **EXE** cũng có thể coi là một **assembly**. Nó là đơn vị nhỏ nhất của việc triển khai cho bất kỳ dự án **.NET** nào.

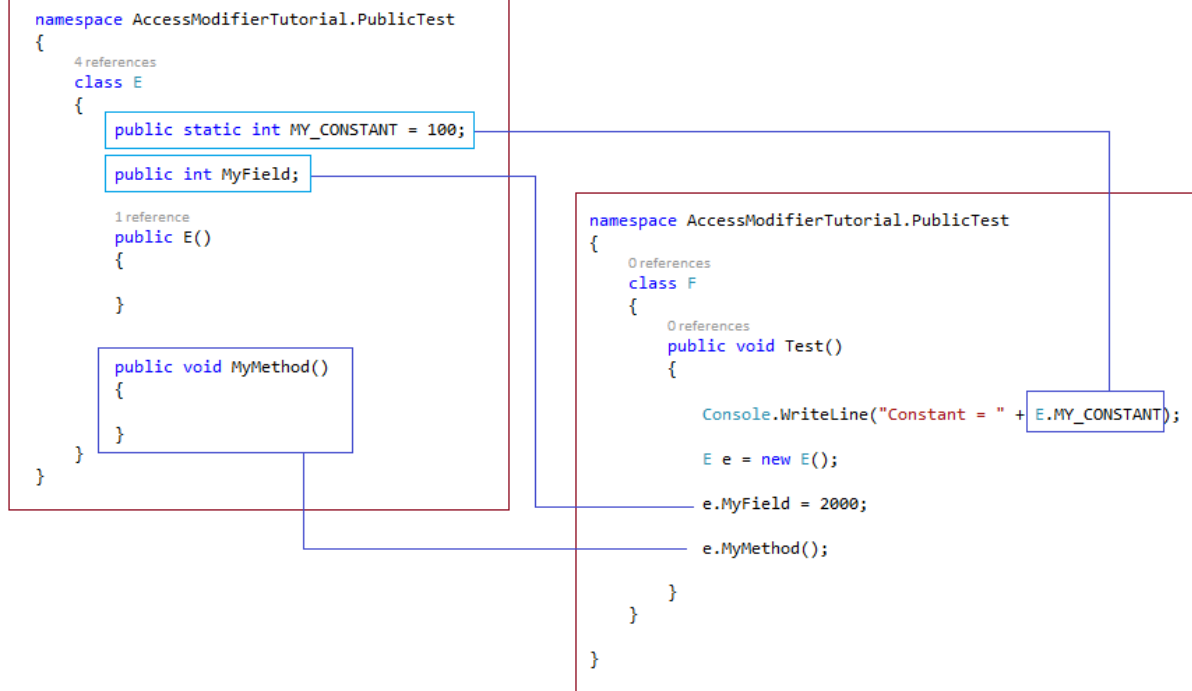
Assembly một cách cụ thể chứa mã **.NET** theo **MSIL (Microsoft Intermediate language - Một ngôn ngữ trung gian)** sẽ được biên dịch thành mã máy tính (Native code) ("**JITted**") - Được biên dịch bởi các trình biên dịch **Just-In-Time** trong lần đầu tiên nó được thực thi trên máy tính,. Đó là mã đã được biên dịch cũng sẽ được lưu trữ trong **Assembly** và tái sử dụng cho các lần gọi tiếp theo.

protected internal access modifier

Độ truy cập **protected internal** là kết hợp giữa hai độ truy cập **protected** và **internal**, khi một thành viên của class có độ truy cập này, bạn chỉ có thể truy cập vào thành viên đó trong cùng class định nghĩa ra nó hoặc các class con và nằm trong cùng một **Assembly**

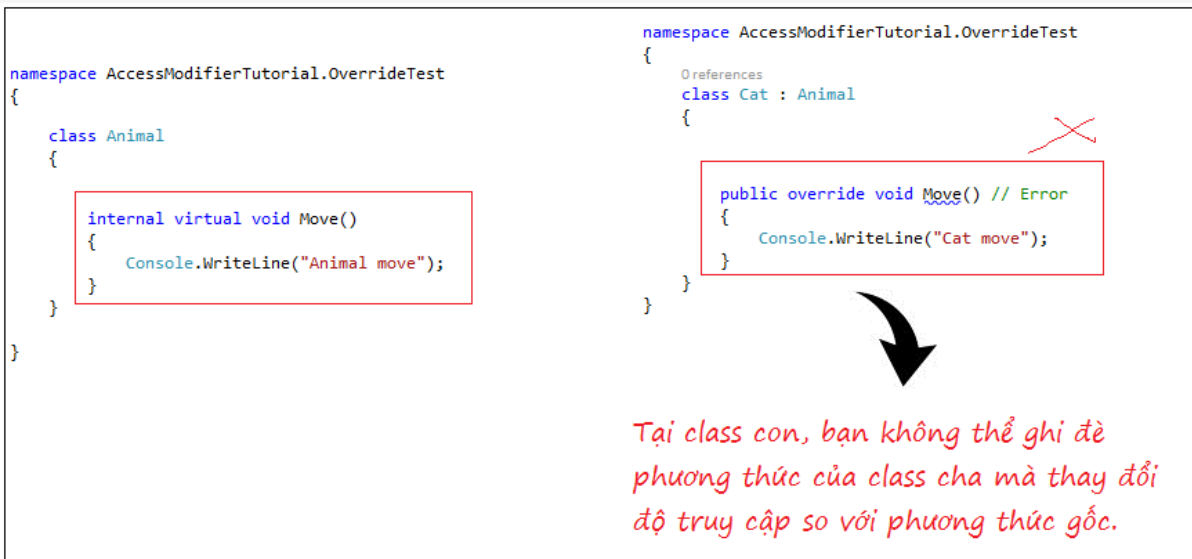
public access modifier

public access modifier là mạnh mẽ nhất và có thể truy cập ở mọi nơi. Nó có phạm vi truy cập rộng nhất so với các modifier khác.



Độ truy cập và thừa kế

Trong **CSharp** bạn có thể ghi đè (override) một phương thức (method) của class cha bởi một phương thức cùng tên cùng tham số, cùng kiểu trả về tại class con, tuy nhiên bạn không được phép thay đổi độ truy cập (access modifier) của nó.



Tuy nhiên, bạn có thể tạo một phương thức cùng tên cùng tham số, cùng kiểu trả về nhưng khác độ truy cập nếu sử dụng từ khóa **new**, thực tế đây là một phương thức khác chẳng liên quan gì tới phương thức của class cha.

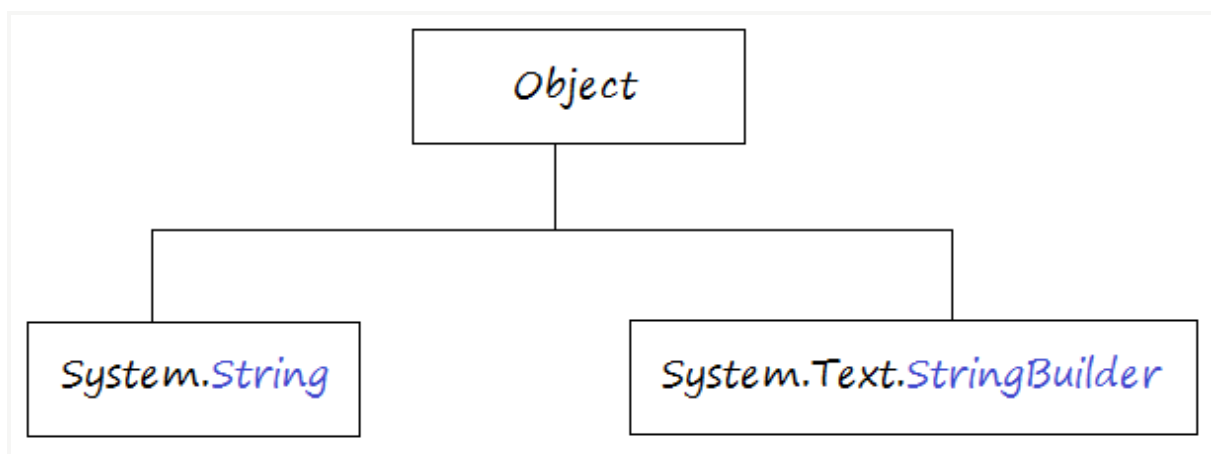
```
namespace AccessModifierTutorial.OverrideTest
{
    class Animal
    {
        internal virtual void Move()
        {
            Console.WriteLine("Animal move");
        }
    }
}

namespace AccessModifierTutorial.OverrideTest
{
    References
    class Mouse : Animal
    {
        protected new void Move()
        {
            Console.WriteLine("Mouse move");
        }
    }
}
```

Tại class con, bạn có thể tạo một phương thức cùng tên, cùng tham số, cùng kiểu trả về và khác độ truy cập với phương thức ở class cha, nhưng phải sử dụng từ khóa 'new'.

Hướng dẫn sử dụng C# String và StringBuilder

Sơ đồ thừa kế:



Khi làm việc với các dữ liệu văn bản, CSharp cung cấp cho bạn 2 class String và StringBuilder. Nếu làm việc với các dữ liệu lớn bạn nên sử dụng StringBuilder để đạt hiệu năng nhanh nhất. Về cơ bản 2 class này có nhiều điểm giống nhau.

- String là bất biến (*immutable*), khái niệm này sẽ được nói chi tiết ở trong tài liệu, và không cho phép có class con.
- StringBuilder có thể thay đổi (*mutable*)

Khái niệm mutable & immutable

// Đây là một class có 1 trường value.

```

// Sau khi khởi tạo đối tượng bạn có thể sét đặt lại giá trị của trường
Value
// thông qua việc gọi method SetNewValue(int).
// Như vậy đây là class có thể thay đổi (mutable).
class MutableClassExample
{
    private int Value;

    public MutableClassExample(int value)
    {
        this.Value = value;
    }

    public void SetNewValue(int newValue)
    {
        this.Value = newValue;
    }
}

// Đây là một class với trường Value, Name.
// Khi bạn khởi tạo đối tượng class này
// bạn không thể sét đặt lại Value từ bên ngoài, và tất cả các trường khác
của nó cũng thế.
// Class này không hề có các hàm để sét đặt lại các trường (field) từ bên
ngoài.
// Nếu muốn bạn chỉ có thể tạo mới một đối tượng khác.
// Điều đó có nghĩa là class này là không thể thay đổi (immutable)
class ImmutableClassExample
{
    private int Value;
    private String Name;

    public ImmutableClassExample(String name, int value)
    {
        this.Value = value;
        this.Name = name;
    }

    public String GetName()
    {
        return Name;
    }

    public int GetValue()
    {
        return Value;
    }
}

```

String là một class bất biến (immutable), **String** có nhiều thuộc tính (trường), ví dụ Length,... nhưng các giá trị đó là không thể thay đổi.

String và string

Trong C# đôi khi bạn thấy **String** và **string** được sử dụng song song. Thực tế chúng không có khác biệt gì, **string** có thể coi là một bí danh (alias) cho **System.String** (Tên đầy đủ bao gồm cả namespace của class String).

Bảng dưới đây mô tả danh sách đầy đủ các bí danh cho các class thông dụng.

Bí danh	Class
object	System.Object
string	System.String
bool	System.Boolean
byte	System.Byte
sbyte	System.SByte
short	System.Int16
ushort	System.UInt16
int	System.Int32
uint	System.UInt32
long	System.Int64
ulong	System.UInt64
float	System.Single
double	System.Double
decimal	System.Decimal
char	System.Char

String

String là một class rất quan trọng trong **CSharp**, và bất kỳ ai bắt đầu với **CSharp** đều đã sử dụng câu lệnh **Console.WriteLine()** để in ra một String lên màn hình **Console**. Nhiều người thường không hề có ý niệm rằng **String** là không thể thay đổi (*immutable*) và là sealed (Bị niêm phong, không cho phép có class con), tất cả các thay đổi trên **String** đều tạo ra một đối tượng **String** khác.

```
[SerializableAttribute]
[ComVisibleAttribute(true)]
public sealed class String : IComparable, ICloneable, IConvertible,
    IEnumerable, IComparable<string>, IEnumerable<char>, IEquatable<string>
```

Các phương thức của String

Dưới đây là danh sách một vài phương thức thông dụng của **String**.

```
public bool EndsWith(string value)
```

```
public bool EndsWith(string value, StringComparison comparisonType)
```

```
public bool Equals(string value)
```

```
public int IndexOf(char value)
```

```
public int IndexOf(char value, int startIndex)
```

```
public int IndexOf(string value, int startIndex, int count)
```

```
public int IndexOf(string value, int startIndex, StringComparison
comparisonType)
```

```
public int IndexOf(string value, StringComparison comparisonType)
```

```
public string Insert(int startIndex, string value)
```

```
public int LastIndexOf(char value)
```

```
public int LastIndexOf(char value, int startIndex)
```

```
public int LastIndexOf(char value, int startIndex, int count)
```

```
public int LastIndexOf(string value)
```

```
public int LastIndexOf(string value, int startIndex)
```

```
public int LastIndexOf(string value, int startIndex, int count)
```

```
public int LastIndexOf(string value, int startIndex, int count,
StringComparison comparisonType)

public int LastIndexOf(string value, int startIndex, StringComparison
comparisonType)

public int LastIndexOf(string value, StringComparison comparisonType)

public int LastIndexOfAny(char[] anyOf)

public int LastIndexOfAny(char[] anyOf, int startIndex)

public int LastIndexOfAny(char[] anyOf, int startIndex, int count)

public int IndexOf(string value, int startIndex, int count,
StringComparison comparisonType)

public string Replace(char oldChar, char newChar)

public string Replace(string oldValue, string newValue)

public string[] Split(params char[] separator)

public string[] Split(char[] separator, int count)

public string[] Split(char[] separator, int count, StringSplitOptions
options)

public string[] Split(char[] separator, StringSplitOptions options)

public string[] Split(string[] separator, StringSplitOptions options)

public bool StartsWith(string value)

public bool StartsWith(string value, bool ignoreCase, CultureInfo culture)

public bool StartsWith(string value, StringComparison comparisonType)

public string Substring(int startIndex)

public string Substring(int startIndex, int length)

public char[] ToCharArray()

public char[] ToCharArray(int startIndex, int length)

public string ToLower()
```



```

public string ToLower(CultureInfo culture)

public string ToLowerInvariant()

public override string ToString()

public string ToUpper()

public string ToUpper(CultureInfo culture)

public string ToUpperInvariant()

public string Trim()

public string Trim(params char[] trimChars)

public string TrimEnd(params char[] trimChars)

public string TrimStart(params char[] trimChars)

```

Length

Length là một thuộc tính của **string**, nó trả về số ký tự Unicode trong string này.

```

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

namespace StringTutorial
{
    class LengthDemo
    {
        public static void Main(string[] args)
        {
            String str = "This is text";

            // Length là một thuộc tính của string.
            // Nó chính là độ dài của chuỗi (số ký tự của chuỗi).
            int len = str.Length;

            Console.WriteLine("String Length is : " + len);

            Console.Read();
        }
    }
}

```

Concat(...)

Concat là một phương thức tĩnh dùng để nối (concatenate) nhiều chuỗi với nhau và trả về một **String** mới.

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

namespace StringTutorial
{
    class ConcatDemo
    {
        public static void Main(string[] args)
        {
            string s1 = "One";
            string s2 = "Two";
            string s3 = "Three";

            // Giống với s1 + s2;
            string s = String.Concat(s1, s2);

            Console.WriteLine("Concat s1, s2 : " + s);

            // Giống với s1 + s2 + s3;
            s = String.Concat(s1, s2, s3);

            Console.WriteLine("Concat s1, s2, s3 : " + s);

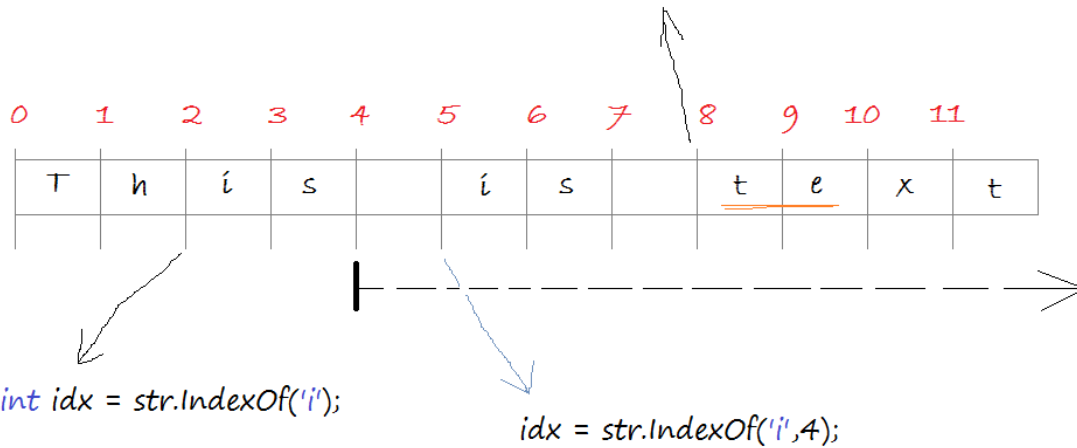
            Console.Read();
        }
    }
}
```

IndexOf(..)

IndexOf(..) là một phương thức trả về chỉ số vị trí xuất hiện lần đầu tiên một ký tự chỉ định hoặc một chuỗi con chỉ định trong chuỗi hiện tại. Có 8 phương thức IndexOf nhưng khác nhau tham số. Chỉ số được bắt đầu từ số 0 (Không phải 1).

```
String str = "This is text";
```

```
int idx = str.IndexOf("te");
```



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

```
namespace StringTutorial
```

```
{
```

```
    class IndexOfDemo
```

```
    {
```

```
        public static void Main(string[] args)
```

```
        {
```

```
            String str = "This is text";
```

```
            // Tìm vị trí xuất hiện ký tự 'i' đầu tiên.
```

```
            int idx = str.IndexOf('i'); // ==> 2
```

```
            Console.WriteLine("- IndexOf('i') = " + idx);
```

```
            // Tìm vị trí xuất hiện ký tự 'i' đầu tiên
```

```
            // tính từ chỉ số thứ 4 trở về cuối chuỗi.
```

```
            idx = str.IndexOf('i', 4); // ==> 5
```

```
            Console.WriteLine("- indexOf('i',4) = " + idx);
```

```
            // Tìm vị trí xuất hiện chuỗi con "te" đầu tiên.
```

```
            idx = str.IndexOf("te"); // ==> 8
```

```
            Console.WriteLine("- IndexOf('te') = " + idx);
```

```
            Console.Read();
```

```
        }
```

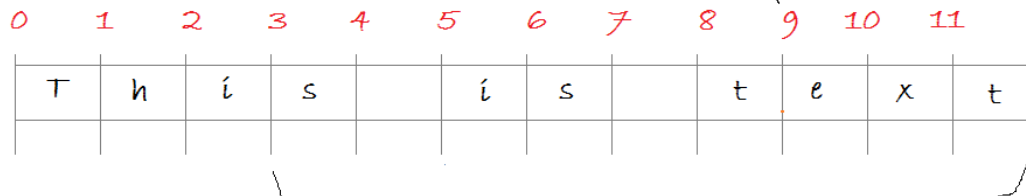
```
    }
```

```
}
```

Substring(..)

```
String str = "This is text";
```

```
String substr = str.substring(2,7);
```



```
String substr = str.substring(3)
```

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

```
namespace StringTutorial
```

```
{
```

```
    class SubstringDemo
```

```
    {
```

```
        public static void Main(string[] args)
```

```
        {
```

```
            string str = "This is text";
```

```
            // Trả về chuỗi con từ chỉ số thứ 3 tới cuối chuỗi.
```

```
            string substr = str.Substring(3);
```

```
            Console.WriteLine("- Substring(3)=\"" + substr);
```

```
            // Trả về chuỗi con từ chỉ số thứ 2, và độ dài 7 ký tự
```

```
            substr = str.Substring(2, 7);
```

```
            Console.WriteLine("- Substring(2, 7) =" + substr);
```

```
            Console.Read();
```

```
        }
```

```
    }  
}
```

Replace(...)

Replace(..): Thay thế một chuỗi con bởi một chuỗi con khác trong string hiện tại, và trả về một chuỗi mới.

```
using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;  
using System.Threading.Tasks;  
  
namespace StringTutorial  
{  
    class ReplaceDemo  
    {  
        public static void Main(string[] args)  
        {  
            String str = "This is text";  
  
            // Thay thế hết các ký tự 'i' bởi ký tự 'x'.  
            String s2 = str.Replace('i', 'x');  
  
            Console.WriteLine("- s2=" + s2); // ==> "Thxs xs text".  
  
            // Thay thế tất cả các chuỗi con "is" bởi "abc".  
            String s3 = str.Replace("is", "abc");  
  
            Console.WriteLine("- s3=" + s3); // ==> "Thabc abc text".  
  
            // Thay thế chuỗi con "is" xuất hiện lần đầu bởi "abc".  
            String s4 = ReplaceFirst(str, "is", "abc");  
  
            Console.WriteLine("- s4=" + s4); // ==> "Thabc is text".  
  
            Console.Read();  
        }  
  
        // Thay thế chuỗi con xuất hiện lần đầu tiên.  
        static string ReplaceFirst(string text, string search, string  
replace)  
        {  
            int pos = text.IndexOf(search);  
            if (pos < 0)  
            {  
                return text;  
            }  
        }  
    }  
}
```

```

    }
    return text.Substring(0, pos) + replace + text.Substring(pos +
search.Length);
    }
}
}

```

Các ví dụ khác

```

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

namespace StringTutorial
{
    class StringOtherDemo
    {
        public static void Main(string[] args)
        {
            String str = "This is text";

            Console.WriteLine("- str=" + str);

            // Trả về chuỗi chữ thường.
            String s2 = str.ToLower();

            Console.WriteLine("- s2=" + s2);

            // Trả về chuỗi chữ hoa.
            String s3 = str.ToUpper();

            Console.WriteLine("- s3=" + s3);

            // Kiểm tra xem chuỗi có bắt đầu bởi chuỗi con "This" hay
không.
            bool swith = str.StartsWith("This");

            Console.WriteLine("- 'str' startsWith This ? " + swith);

            // Một String với khoảng trắng phía trước và sau.
            // \t là ký tự TAB.
            // \n là ký tự xuống dòng.
            str = " \t CSharp is hot! \t \n ";

            Console.WriteLine("- str=" + str);

```

```

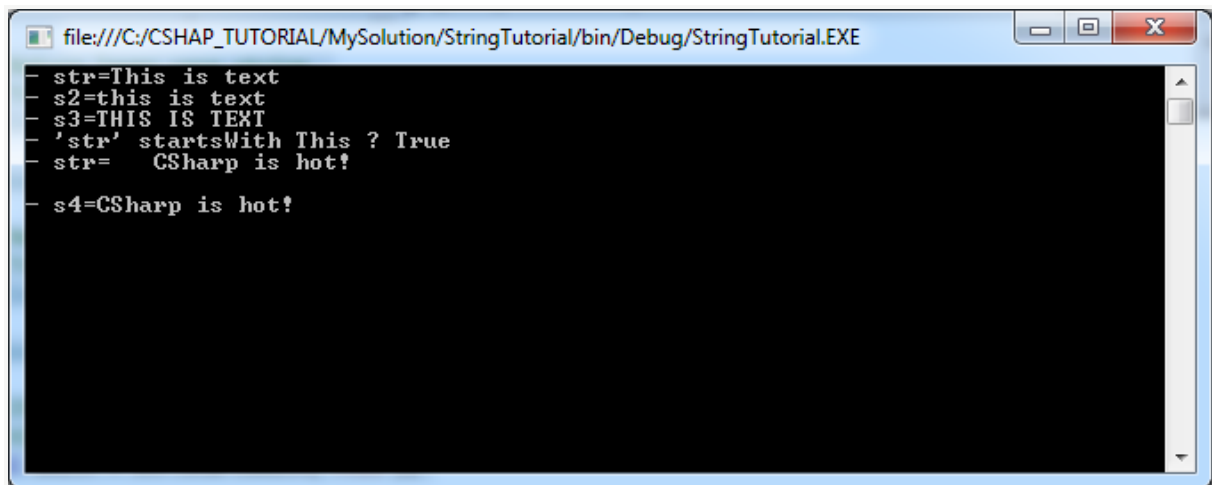
        // Trả về một String mới loại bỏ khoảng trắng ở đầu và cuối
        chuỗi.

        String s4 = str.Trim();

        Console.WriteLine("- s4=" + s4);

        Console.Read();
    }
}
}

```



```

file:///C:/CSHAP_TUTORIAL/MySolution/StringTutorial/bin/Debug/StringTutorial.EXE
- str=This is text
- s2=this is text
- s3=THIS IS TEXT
- 'str' startsWith This ? True
- str= CSharp is hot!
- s4=CSharp is hot!

```

StringBuilder

Trong C# mỗi khi bạn sửa đổi một **String** kết quả đều tạo ra một đối tượng **String** mới. Trong khi đó **StringBuilder** chứa trong nó một mảng các ký tự, mảng này sẽ tự động thay thế bởi một mảng lớn hơn nếu thấy cần thiết, và copy các ký tự ở mảng cũ sang. Nếu bạn phải thao tác ghép chuỗi nhiều lần thì bạn nên sử dụng **StringBuilder**, nó giúp làm tăng hiệu năng của chương trình. Tuy nhiên nếu chỉ ghép nối một vài chuỗi thì điều đó không cần thiết, bạn không nên lạm dụng **StringBuilder** trong trường hợp đó.

```

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

namespace StringTutorial
{
    class StringBuilderDemo
    {
        public static void Main(string[] args)
        {

```

```

// Tạo đối tượng StringBuilder
// Hiện tại chưa có dữ liệu trên StringBuilder.
StringBuilder sb = new StringBuilder(10);

// Nối thêm một chuỗi con
sb.Append("Hello...");
Console.WriteLine("- sb after appends a string: " + sb);

// Nối thêm một ký tự.
char c = '!';
sb.Append(c);
Console.WriteLine("- sb after appending a char: " + sb);

// Tròn một String tại chỉ số 5.
sb.Insert(8, " CSharp");
Console.WriteLine("- sb after insert string: " + sb);

// Xóa một chuỗi con bắt đầu tại chỉ số 5, với 3 ký tự.
sb.Remove(5, 3);

Console.WriteLine("- sb after remove: " + sb);

// Lấy ra string trong StringBuilder.
String s = sb.ToString();

Console.WriteLine("- String of sb: " + s);

Console.Read();
}
}
}

```

```

file:///C:/CSHAP_TUTORIAL/MySolution/StringTutorial/bin/Debug/StringTutorial.EXE
- sb after appends a string: Hello...
- sb after appending a char: Hello...!
- sb after insert string: Hello... CSharp!
- sb after remove: Hello CSharp!
- String of sb: Hello CSharp!

```

C# Property

Property là một thành viên (member) của một class, interface. Nó là mở rộng của một trường (field). Property cho phép bạn truy cập vào một trường hoặc thay đổi giá trị của trường đó, mà không cần thiết phải truy cập trực tiếp vào trường.

Bạn có thể tạo một Property chỉ cho phép truy cập vào một trường, không cho phép thay đổi giá trị của trường, và ngược lại. Đây chính là điểm lợi hại nhất của một Property.

Với trường (field), nếu bạn có thể truy cập vào nó từ bên ngoài, bạn cũng có thể thay đổi giá trị của nó, điều này rõ ràng là nguy hiểm

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

namespace CSharpPropertyTutorial
{
    class Employee
    {
        // Để có thể truy cập ở bên ngoài, trường này phải public hoặc
        protected.
        public string code;

        // Để có thể truy cập ở bên ngoài, trường này phải public hoặc
        protected.
        public string name;

        public Employee(string code, string name)
        {
            this.code = code;
            this.name = name;
        }
    }

    class EmployeeTest
    {
        public static void Main(string[] args)
        {
            // Tạo một đối tượng Employee.
            Employee john = new Employee("E01", "John");

            // Bạn có thể truy cập vào tên của nhân viên
            // (name là trường public vì vậy bạn có thể truy cập ở bên
            ngoài).
            Console.WriteLine("Employee Name = " + john.name);
        }
    }
}
```

```

        // Tuy nhiên bạn cũng có thể sét giá trị mới cho trường name.
        // (Điều này rõ ràng là nguy hiểm).
        john.name = "Marry";

        Console.WriteLine("Employee Name = " + john.name);

        Console.Read();
    }
}

```

Property là một giải pháp để giải quyết vấn đề nêu trên. Dưới đây là một ví dụ sử dụng **Property Code, Name** để truy cập vào trường **code, name** của class **Employee2**.

```

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

namespace CSharpPropertyTutorial
{
    class Employee2
    {
        // Trường này là private, không cho phép truy cập từ bên ngoài.
        private string code;

        // Trường này là private, không cho phép truy cập từ bên ngoài.
        private string name;

        // Khai báo một property, là public, có thể truy cập từ bên ngoài.
        public string Code
        {
            get
            {
                return this.code;
            }
            set
            {
                // value là một từ khóa đặc biệt,
                // nó ám chỉ giá trị mới được gán cho thuộc tính.
                this.code = value;
            }
        }

        // Khai báo một property, là public, chỉ cho phép truy cập
        // không cho phép gán giá trị mới.
        public string Name

```

```

        {
            get
            {
                return this.name;
            }
        }

        public Employee2(string code, string name)
        {
            this.code = code;
            this.name = name;
        }
    }

}

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

namespace CSharpPropertyTutorial
{
    class Employee2Test
    {
        public static void Main(string[] args)
        {
            // Tạo một đối tượng Employee.
            Employee2 john = new Employee2("E01", "John");

            Console.WriteLine("Employee Code = " + john.Code);
            Console.WriteLine("Employee Name = " + john.Name);
            Console.WriteLine("-----");

            // Gán giá trị mới cho thuộc tính Code.
            john.Code = "E02";

            // Không thể gán giá trị mới cho Name.
            // john.Name = "Marry";

            Console.WriteLine("Employee Code = " + john.Code);
            Console.WriteLine("Employee Name = " + john.Name);

            Console.Read();
        }
    }
}

```

```
    }  
}  
  
}
```

Property trừu tượng (Abstract Property)

Property sử dụng dụng để **set** và **get** giá trị của một trường, về bản chất nó được coi là một phương thức đặc biệt, vì vậy nó cũng có thể khai báo trừu tượng (abstract), và nó sẽ được thực hiện (implements) tại một class con. Class có thuộc tính (property) khai báo là abstract thì nó phải khai báo là abstract. Các thuộc tính trừu tượng cũng có thể được khai báo trong Interface.

```
using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;  
using System.Threading.Tasks;  
  
namespace CSharpPropertyTutorial  
{  
    abstract class Animal  
    {  
  
        // Một thuộc tính trừu tượng  
        public abstract string Name  
        {  
            get;  
        }  
  
        // Một thuộc tính trừu tượng có cả set & get  
        public abstract int Age  
        {  
            get;  
            set;  
        }  
    }  
}  
  
using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;  
using System.Threading.Tasks;  
  
namespace CSharpPropertyTutorial  
{
```

```

class Cat : Animal
{
    private string name;
    private int age;

    // Triển khai property trừu tượng khai báo trong class Animal.
    public override string Name
    {
        get
        {
            return this.name;
        }
    }

    // Triển khai property trừu tượng khai báo trong class Animal.
    public override int Age
    {
        get
        {
            return this.age;
        }
        set
        {
            this.age = value;
        }
    }

    // Constructor.
    public Cat(string name, int age)
    {
        this.name = name;
        this.age = age;
    }
}
}

```

Property trong Interface

Bạn cũng có thể khai báo **Property** trong một Interface, các thuộc tính này sẽ được thực hiện (implements) tại các class con.

```

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

namespace CSharpPropertyTutorial
{

```

```

interface IColor
{
    // Một thuộc tính trừu tượng của Interface.
    String Color
    {
        get;
    }
}

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

namespace CSharpPropertyTutorial
{
    class Ball : IColor
    {
        private string color;

        // Triển khai property được khai báo trong Interace IColor
        // (Không được viết từ khóa override ở đây,
        // vì nó triển khai property của Interface).
        public string Color
        {
            get
            {
                return this.color;
            }
        }

        // Constructor.
        public Ball(String color)
        {
            this.color = color;
        }
    }
}

```

C# Enum

enum trong C# là một từ khóa, nó sử dụng để khai báo một tập hợp kiểu liệt kê (enumeration).

```

using System;
using System.Collections.Generic;
using System.Linq;

```

```

using System.Text;
using System.Threading.Tasks;

namespace CSharpEnumTutorial
{
    class WeekDayConstants
    {
        public const int MONDAY = 2;
        public const int TUESDAY = 3;
        public const int WEDNESDAY = 4;
        public const int THURSDAY = 5;

        public const int FRIDAY = 6;
        public const int SATURDAY = 7;
        public const int SUNDAY = 1;
    }
}

```

Một class với một method mô phỏng lấy ra tên công việc sẽ làm ứng với ngày cụ thể trong tuần. (Giống với thời khóa biểu)

```

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

namespace CSharpEnumTutorial
{
    // Thời khóa biểu.
    class Timetable
    {
        // Tham số truyền vào là ngày trong tuần.
        // Trả về tên công việc sẽ làm.
        public static String getJob(int dayInWeek)
        {
            if (dayInWeek == WeekDayConstants.SATURDAY
                || dayInWeek == WeekDayConstants.SUNDAY)
            {
                return "Nothing";
            }
            return "Coding";
        }
    }
}

```

Rõ ràng các đoạn mã như vậy là không an toàn. Chẳng hạn như khi bạn gõ các giá trị cho ngày trong tuần chẳng may trùng nhau. Hoặc khi gọi phương thức **Timetable.getJob(int)** mà truyền vào giá trị nằm ngoài các giá trị định nghĩa trước.

1. **Không phải là kiểu an toàn:** Đầu tiên thấy rằng mã (code) của bạn không an toàn, bạn có thể gọi phương thức **GetJob(int)** và truyền vào bất kỳ giá trị nào.
2. **Không có ý nghĩa trong in ấn:** Nếu bạn muốn in ra các ngày trong tuần nó sẽ là các con số, thay vì một chữ có ý nghĩa như *"MONDAY"*.

Có thể sử dụng toán tử == để so sánh các phần tử của enum

Enum là một đối tượng tham chiếu (reference object) giống như class, interface nhưng nó cũng có thể sử dụng cách so sánh ==.

```
// Để so sánh các đối tượng tham chiếu thông thường phải sử dụng method equals(..)
```

```
object obj1 = ...;
```

```
// So sánh đối tượng với null, có thể sử dụng toán tử ==
```

```
if (obj1 == null)
```

```
{
```

```
}
```

```
object obj2 = ...;
```

```
// So sánh khác null.
```

```
if (obj1 != null)
```

```
{
```

```
    // So sánh 2 đối tượng với nhau.
```

```
    if (obj1.Equals(obj2))
```

```
    {
```

```
    }
```

```
}
```

Duyệt trên các phần tử của Enum

Chúng ta có thể duyệt trên tất cả các phần tử của Enum.

```
using System;
```

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

```
using System.Linq;
```

```
using System.Text;
```

```
using System.Threading.Tasks;
```



```

namespace CSharpEnumTutorial
{
    class ValuesDemo
    {
        public static void Main(string[] args)
        {

            // Lấy ra tất cả các phần tử của Enum.
            Array allDays = Enum.GetValues(typeof(WeekDay));

            foreach (WeekDay day in allDays)
            {
                Console.WriteLine("Day: " + day);
            }

            Console.Read();
        }
    }
}

```

Enum và Attribute

Bạn có thể gắn các Attribute lên các phần tử của Enum, điều này giúp cho Enum mang nhiều thông tin hơn, và bạn có thể lấy ra các thông tin đó ứng với từng phần tử của Enum.

```

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

namespace CSharpEnumTutorial
{
    class GenderAttr : Attribute
    {
        // code: M, text = Male
        // code: F, text = Female
        internal GenderAttr(string code, string text)
        {
            this.Code = code;
            this.Text = text;
        }

        public string Code { get; private set; }
    }
}

```

```

        public string Text { get; private set; }

    }

}

enum Gender (Giới tính) có 2 phần tử MALE (Nam) và FEMALE (Nữ).
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace CSharpEnumTutorial
{
    public enum Gender
    {

        // Một phần tử của Enum, có thuộc tính
        [GenderAttr("M", "Male")]
        MALE,

        // Một phần tử của Enum, có thuộc tính
        [GenderAttr("F", "Female")]
        FEMALE

    }

}

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

namespace CSharpEnumTutorial
{
    class Genders
    {

        // Trả về Gender ứng với code.
        // (Phương thức này có thể trả về null).
        public static Gender? GetGenderByCode(string code)
        {

            // Lấy hết tất cả các phần tử của Enum.

```

```

        Array allGenders = Enum.GetValues(typeof(Gender));

        foreach (Gender gender in allGenders)
        {
            string c = GetCode(gender);
            if (c == code)
            {
                return gender;
            }
        }
        return null;
    }

    public static string GetText(Gender gender)
    {
        GenderAttr genderAttr = GetAttr(gender);
        return genderAttr.Text;
    }

    public static string GetCode(Gender gender)
    {
        GenderAttr genderAttr = GetAttr(gender);
        return genderAttr.Code;
    }

    private static GenderAttr GetAttr(Gender gender)
    {
        MemberInfo memberInfo = GetMemberInfo(gender);
        return (GenderAttr)Attribute.GetCustomAttribute(memberInfo,
typeof(GenderAttr));
    }

    private static MemberInfo GetMemberInfo(Gender gender)
    {
        MemberInfo memberInfo
            = typeof(Gender).GetField(Enum.GetName(typeof(Gender),
gender));

        return memberInfo;
    }
}

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

```

```

namespace CSharpEnumTutorial
{
    class GenderTest
    {
        public static void Main(string[] args)
        {

            Gender marryGender = Gender.FEMALE;

            Console.WriteLine("marryGender: " + marryGender);

            Console.WriteLine("Code: " + Genders.GetCode(marryGender)); //
F
            Console.WriteLine("Text: " + Genders.GetText(marryGender)); //
Femate

            String code = "M";
            Console.WriteLine("Code: " + code);

            // Phương thức có thể trả về null.
            Gender? gender = Genders.GetGenderByCode(code);

            Console.WriteLine("Gender by code: " + gender);

            Console.Read();
        }
    }
}

```

Enum có thể có phương thức hay không

Trong C# enum không thể có phương thức, tuy nhiên trong trường hợp bạn muốn có một cái gì đó giống như Enum và có phương thức bạn có thể định nghĩa một class, class này không cho phép tạo thêm các đối tượng ngoài các đối tượng đã được tạo sẵn của nó.

```

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

namespace CSharpEnumTutorial
{
    class GenderX

```

```

{

    public static readonly GenderX MALE = new GenderX("M", "Male");

    public static readonly GenderX FEMALE = new GenderX("F", "Female");

    private string code;
    private string text;

    // Cấu tử (Constructor) private: Không cho phép tạo đối tượng từ bên
    ngoài class.
    private GenderX(string code, string text)
    {
        this.code = code;
        this.text = text;
    }

    public string GetCode()
    {
        return this.code;
    }

    public string GetText()
    {
        return this.text;
    }

    public static GenderX GetGenderByCode(string code)
    {
        if (MALE.code.Equals(code))
        {
            return MALE;
        }
        else if (FEMALE.code.Equals(code))
        {
            return FEMALE;
        }
        return null;
    }

}
}

```

C# Structure

Trong C#, Struct (cấu trúc) là một kiểu giá trị đặc biệt, nó tạo ra một biến để lưu trữ nhiều giá trị riêng lẻ, mà các giá trị này có liên quan tới nhau.

Ví dụ thông tin về một nhân viên bao gồm:

- Mã nhân viên
- Tên nhân viên
- Chức vụ

Bạn có thể tạo ra 3 biến để lưu trữ các thông tin trên của nhân viên. Tuy nhiên bạn có thể tạo ra một Struct để lưu trữ cả 3 thông tin trên trong một biến duy nhất.

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

namespace CSharpStructureTutorial
{
    struct Employee
    {

        public string empNumber;
        public string empName;
        public string position;

        public Employee(string empNumber, string empName, string position)
        {
            this.empNumber = empNumber;
            this.empName = empName;
            this.position = position;
        }
    }
}

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

namespace CSharpStructureTutorial
{
    class EmployeeTest
    {

        public static void Main(string[] args)
        {
            Employee john = new Employee("E01", "John", "CLERK");

            Console.WriteLine("Emp Number: " + john.empNumber);
            Console.WriteLine("Emp Name: " + john.empName);
        }
    }
}
```

```

        Console.WriteLine("Emp Position: " + john.position);

        Console.Read();
    }
}

```

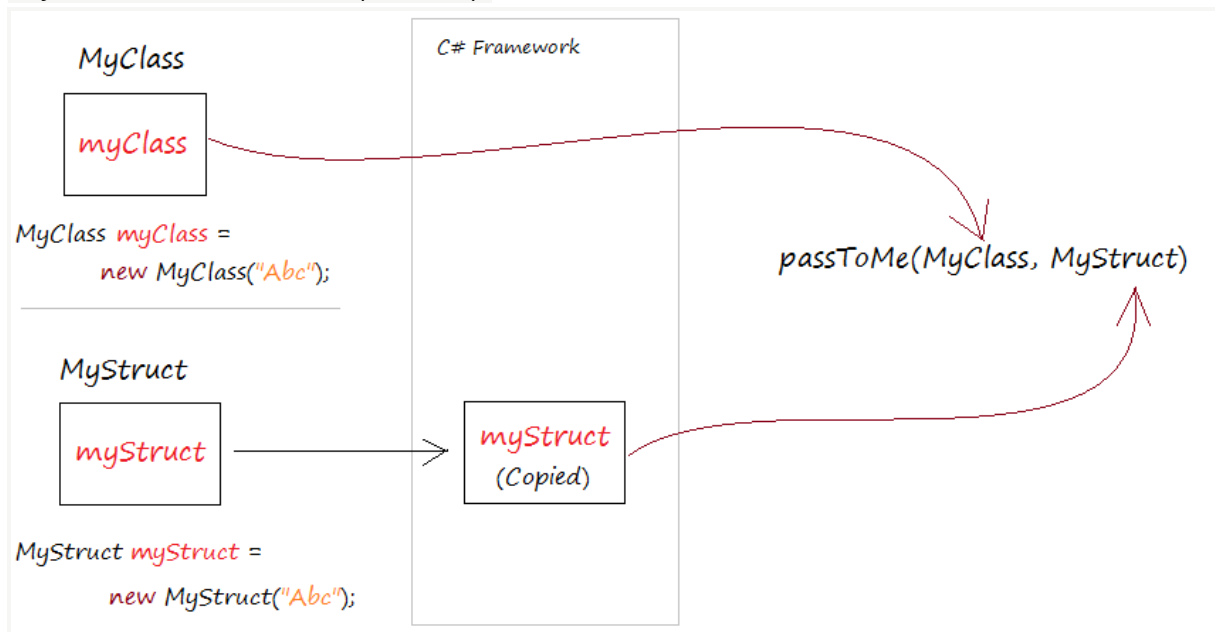
Struct so với Class

Struct thường được sử dụng để tạo ra một đối tượng để lưu trữ các giá trị, trong khi đó **class** được sử dụng đa dạng hơn.

1. **Struct** không cho phép thừa kế, nó không thể mở rộng từ một class hoặc một **struct** nào.
2. **Struct** không cho phép thực hiện (implements) Interface.

Một trong các **struct** hay được sử dụng trong **C#** đó là **DateTime**, nó mô tả ngày tháng và thời gian.

Nếu **struct** xuất hiện như một tham số trong một phương thức, nó được truyền dưới dạng giá trị. Trong khi đó nếu **class** xuất hiện như là một tham số trong một phương thức nó được truyền như một tham chiếu (reference).



```

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

namespace CSharpStructureTutorial
{

```

```

class MyClass
{
    public string name = "Abc";

    public MyClass(string name)
    {
        this.name = name;
    }
}

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

namespace CSharpStructureTutorial
{
    struct MyStruct
    {
        public string name;

        public MyStruct(string name)
        {
            this.name = name;
        }
    }
}

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

namespace CSharpStructureTutorial
{
    class MyTest
    {
        private static void passToMe(MyClass myClass, MyStruct myStruct)
        {
            // Thay đổi giá trị trường name.
            myClass.name = "New Name";
        }
    }
}

```



```

        // Thay đổi giá trị trường name.
        myStruct.name = "New Name";
    }

    public static void Main(string[] args)
    {
        MyClass myClass = new MyClass("Abc");
        MyStruct myStruct = new MyStruct("Abc");

        Console.WriteLine("Before pass to method");
        Console.WriteLine("myClass.name = " + myClass.name); // Abc
        Console.WriteLine("myStruct.name = " + myStruct.name); // Abc

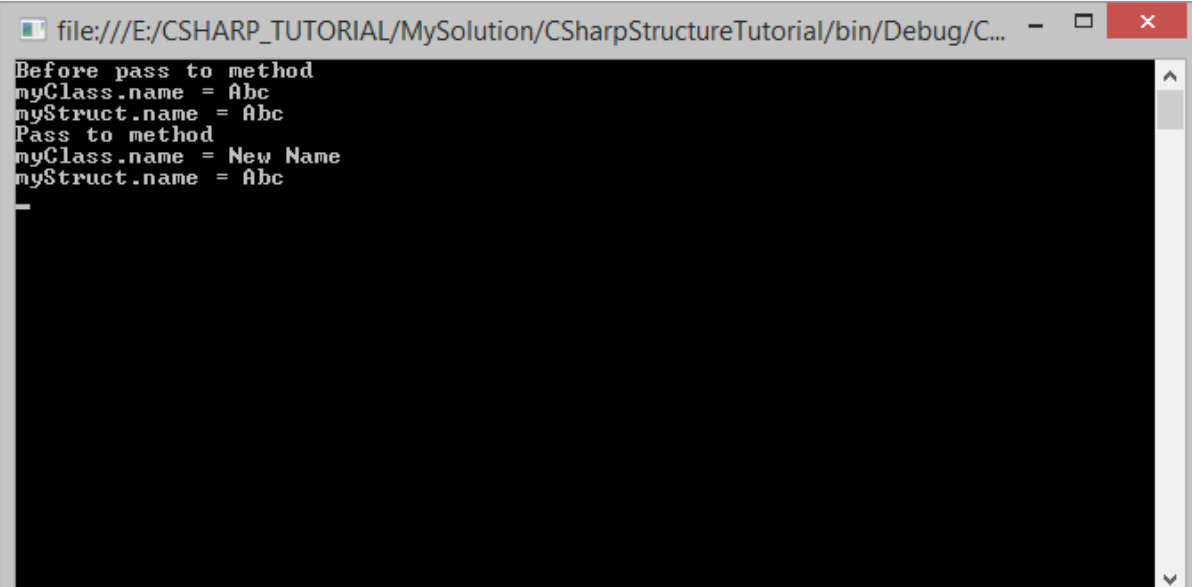
        Console.WriteLine("Pass to method");

        // myStruct truyền vào phương thức là một bản copy.
        // (Không phải là đối tượng gốc).
        passToMe(myClass, myStruct);

        Console.WriteLine("myClass.name = " + myClass.name); // New Name
        Console.WriteLine("myStruct.name = " + myStruct.name); // Abc

        Console.Read();
    }
}

```



```

file:///E:/CSHARP_TUTORIAL/MySolution/CSharpStructureTutorial/bin/Debug/C...
Before pass to method
myClass.name = Abc
myStruct.name = Abc
Pass to method
myClass.name = New Name
myStruct.name = Abc

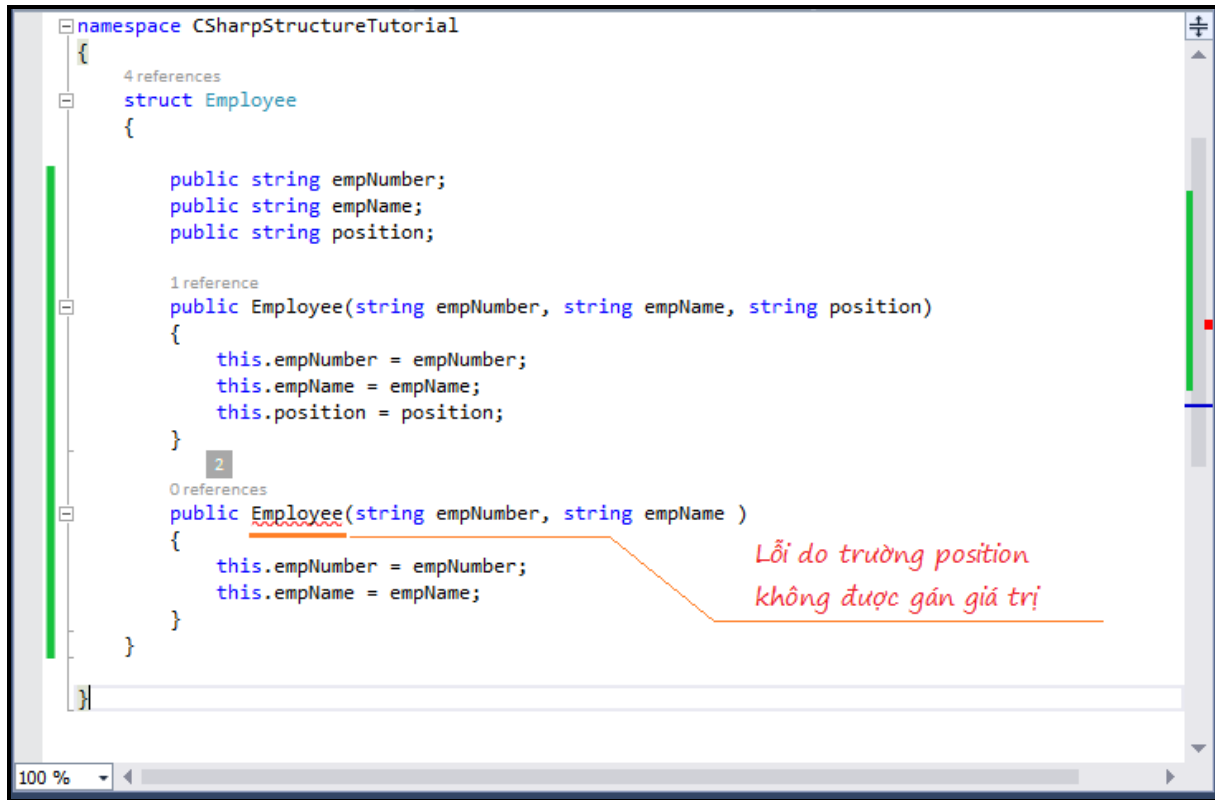
```

Constructor của Struct

Struct có thể có các Constructor (Phương thức khởi tạo), nhưng không có Destructor (Phương thức hủy đối tượng).

Dưới đây là một số chú ý đối với Constructor:

1. Bạn không thể viết một Constructor mặc định (Không có tham số) cho struct.
2. Trong Constructor bạn phải gán các giá trị cho tất cả các trường của struct.



Phương thức và thuộc tính của Struct

Struct có thể có các phương thức và thuộc tính (Property), và vì Struct không có thừa kế nên tất cả các phương thức, thuộc tính của nó không được phép trừu tượng (abstract).

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

```
namespace CSharpStructureTutorial
{
    struct Book
    {
```

```

private string title;
private string author;

// Property
public string Title
{
    get
    {
        return this.title;
    }
    set
    {
        this.title = value;
    }
}

// Property
public string Author
{
    get
    {
        return this.author;
    }
}

// Constructor.
public Book(string title, string author)
{
    this.title = title;
    this.author = author;
}

// Method.
public string GetInfo()
{
    return "Book Title: " + this.title + ", Author: " +
this.author;
}

}
}

```

Indexer và Event của Struct

Struct có thể có **Indexer** và **Event**, xem thêm về Indexer và Event tại:

- [Hướng dẫn sử dụng C# Delegate và Event](#)

C# Generics

Kiểu Generic Class, Interface

Lớp Generics

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

namespace GenericsTutorial
{
    public class KeyValue<K, V>
    {
        private K key;
        private V value;

        public KeyValue(K key, V value)
        {
            this.key = key;
            this.value = value;
        }

        public K GetKey()
        {
            return key;
        }

        public void SetKey(K key)
        {
            this.key = key;
        }

        public V GetValue()
        {
            return value;
        }

        public void SetValue(V value)
        {
            this.value = value;
        }
    }
}
```

```

    }

}

```

K, V trong lớp **KeyValue<K,V>** được gọi là tham số generics nó là một kiểu dữ liệu nào đó. Khi sử dụng lớp này bạn phải xác định kiểu tham số cụ thể.

```

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

namespace GenericsTutorial
{
    public class KeyValueDemo
    {

        public static void Main(string[] args)
        {

            // Tạo một đối tượng KeyValue
            // int: Số điện thoại (K = int)
            // string: Tên người dùng. (V = string).
            KeyValue<int, string> entry = new KeyValue<int,
string>(12000111, "Tom");

            // C# hiểu kiểu trả về là int (K = int).
            int phone = entry.GetKey();

            // C# hiểu kiểu trả về là string (V = string).
            string name = entry.GetValue();

            Console.WriteLine("Phone = " + phone + " / name = " + name);

            Console.Read();
        }

    }

}

```

Thừa kế lớp Generics

Một lớp mở rộng từ một lớp generics, nó có thể chỉ định rõ kiểu cho tham số generics, giữ nguyên các tham số generics hoặc thêm các tham số generics.

```

using System;
using System.Collections.Generic;

```

```

using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace GenericsTutorial
{
    // Class này mở rộng từ class KeyValue<K,V>
    // Và chỉ định rõ K,V
    // K = int   (Số điện thoại).
    // V = string (Tên người dùng).
    public class PhoneNameEntry : KeyValue<int, string>
    {
        public PhoneNameEntry(int key, string value)
            : base(key, value)
        {
        }
    }
}

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

namespace GenericsTutorial
{
    public class PhoneNameEntryDemo
    {
        public static void Main(string[] args)
        {
            PhoneNameEntry entry = new PhoneNameEntry(120001111, "Tom");

            // C# hiểu kiểu trả về là int.
            int phone = entry.GetKey();

            // C# hiểu kiểu trả về là string.
            string name = entry.GetValue();

            Console.WriteLine("Phone = " + phone + " / name = " + name);

            Console.Read();
        }
    }
}

```

```

    }

}

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

namespace GenericsTutorial
{
    // Class này mở rộng class KeyValue<K,V>
    // Xác định rõ kiểu tham số K là String.
    // Vẫn giữ kiểu tham số generic V.
    public class StringAndValueEntry<V> : KeyValue<string, V>
    {
        public StringAndValueEntry(string key, V value)
            : base(key, value)
        {
        }
    }
}

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

namespace GenericsTutorial
{
    public class StringAndValueEntryDemo
    {
        public static void main(String[] args)
        {
            // (Mã nhân viên, Tên nhân viên).
            // V = string (Tên nhân viên)
            StringAndValueEntry<String> entry = new
StringAndValueEntry<String>("E001", "Tom");

```

```

        String empNumber = entry.GetKey();

        String empName = entry.GetValue();

        Console.WriteLine("Emp Number = " + empNumber);
        Console.WriteLine("Emp Name = " + empName);

        Console.Read();

    }
}

}

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

namespace GenericsTutorial
{
    // Class này mở rộng class KeyValue<K,V>
    // Nó có thêm một tham số generics I.
    public class KeyValueInfo<K, V, I> : KeyValue<K, V>
    {
        private I info;

        public KeyValueInfo(K key, V value)
            : base(key, value)
        {
        }

        public KeyValueInfo(K key, V value, I info)
            : base(key, value)
        {
            this.info = info;
        }

        public I GetInfo()
        {
            return info;
        }

        public void SetInfo(I info)
        {
            this.info = info;
        }
    }
}

```



```
    }  
  
    }  
  
}
```

Interface Generics

Một Interface có tham số Generics:

```
using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;  
using System.Threading.Tasks;  
  
namespace GenericsTutorial  
{  
    public interface GenericInterface<G>  
    {  
  
        G DoSomething();  
  
    }  
  
}  
  
using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;  
using System.Threading.Tasks;  
  
namespace GenericsTutorial  
{  
  
    public class GenericInterfaceImpl<G> : GenericInterface<G>  
    {  
  
        private G something;  
  
        public G DoSomething()  
        {  
            return something;  
        }  
  
    }  
  
}
```

Sử dụng Generic với Exception

Bạn có thể định nghĩa một **Exception** có các tham số Generics.

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

namespace GenericsTutorial
{
    class MyException<E> : ApplicationException
    {

    }

}

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

namespace GenericsTutorial
{
    class UsingGenericExceptionValid01
    {

        public void SomeMethod()
        {
            try
            {
                // ...

            }
            // Hợp lệ
            catch (MyException<string> e)
            {
                // Làm gì đó ở đây.
            }
            // Hợp lệ
            catch (MyException<int> e)
            {
                // Làm gì đó ở đây.
            }
            catch (Exception e)
            {
            }
        }
    }
}
```

Sử dụng **Generics Exception** hợp lệ:

```

namespace GenericsTutorial
{
    class UsingGenericExceptionInvalid
    {
        public void SomeMethod()
        {
            try
            {
                // ...

            }
            // Hợp lệ
            catch (MyException<string> e)
            {
                // Làm gì đó ở đây.
            }
            // Không hợp lệ (Không hiểu tham số K). *****
            catch (MyException<K> e)
            {
                // Làm gì đó ở đây.
            }
            catch (Exception e)
            {
            }
        }
    }
}

```

Phương thức generics

Một phương thức trong lớp hoặc Interface có thể được Generic hóa (generify).

```

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

namespace GenericsTutorial
{
    public class MyUtils
    {
        // <K,V> : Nói rằng phương thức này có 2 kiểu tham số K,V
        // Phương thức trả về kiểu K.
        public static K GetKey<K, V>(KeyValue<K, V> entry)
        {
            K key = entry.GetKey();

```

```

        return key;
    }

    // <K,V> : Nói rằng phương thức này có 2 kiểu tham số K,V
    // Phương thức trả về kiểu V.
    public static V GetValue<K, V>(KeyValue<K, V> entry)
    {
        V value = entry.GetValue();
        return value;
    }

    // List<E>: Danh sách chứa các phần tử kiểu E
    // Phương thức trả về kiểu E.
    public static E GetFirstElement<E>(List<E> list, E defaultValue)
    {
        if (list == null || list.Count == 0)
        {
            return defaultValue;
        }
        E first = list.ElementAt(0);
        return first;
    }

}

}

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

namespace GenericsTutorial
{
    public class MyUtilsDemo
    {

        public static void Main(string[] args)
        {

            // K = int: Phone
            // V = string: Name
            KeyValue<int, string> entry1 = new KeyValue<int,
String>(12000111, "Tom");
            KeyValue<int, string> entry2 = new KeyValue<int,
String>(12000112, "Jerry");

            // (K = int).

```

```

        int phone = MyUtils.GetKey(entry1);
        Console.WriteLine("Phone = " + phone);

        // Một danh sách chứa các phần tử kiểu KeyValue<int,string>.
        List<KeyValue<int, string>> list = new List<KeyValue<int,
string>>();

        // Thêm phần tử vào danh sách.
        list.Add(entry1);
        list.Add(entry2);

        KeyValue<int, string> firstEntry = MyUtils.GetFirstElement(list,
null);

        if (firstEntry != null)
        {
            Console.WriteLine("Value = " + firstEntry.GetValue());
        }

        Console.Read();
    }
}

```

Khởi tạo đối tượng Generic

Đôi khi bạn muốn khởi tạo một đối tượng **Generic**:

```

public void DoSomething<T>()
{
    // Khởi tạo đối tượng Generic
    T t = new T(); // Error
}

```

Nguyên nhân lỗi ở trên là do kiểu tham số **T** không chắc chắn nó có phương thức khởi tạo (constructor) **T()**, vì vậy bạn cần phải thêm ràng buộc **when T : new()**. Hãy xem ví dụ dưới đây:

```

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

namespace GenericsTutorial
{
    class GenericInitializationExample
    {

```

```

// Tham số T phải là kiểu có cấu tử mặc định.
public void DoSomething<T>()
    where T : new()
{
    T t = new T();
}

// Tham số K phải là kiểu có cấu tử mặc định
// và mở rộng từ class KeyValue.
public void ToDoSomething<K>()
    where K: KeyValue<K,string>, new( )
{
    K key = new K();
}

public T DoDefault<T>()
{
    // Trả về null nếu T là kiểu tham số.
    // Hoặc 0 nếu T là kiểu số.
    return default(T);
}
}
}

```

Mảng Generic

Trong **C#** bạn có thể khai báo một mảng **Generics**:

// Khởi tạo một mảng.

```

T[] myArray = new T[10];
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace GenericsTutorial
{
    class GenericArrayExample
    {
        public static T[] FilledArray<T>(T value, int count)

```

```

    {
        T[] ret = new T[count];
        for (int i = 0; i < count; i++)
        {
            ret[i] = value;
        }
        return ret;
    }

    public static void Main(string[] args)
    {

        string value = "Hello";

        string[] filledArray = FilledArray<string>(value, 10);

        foreach (string s in filledArray)
        {
            Console.WriteLine(s);
        }
    }
}

```

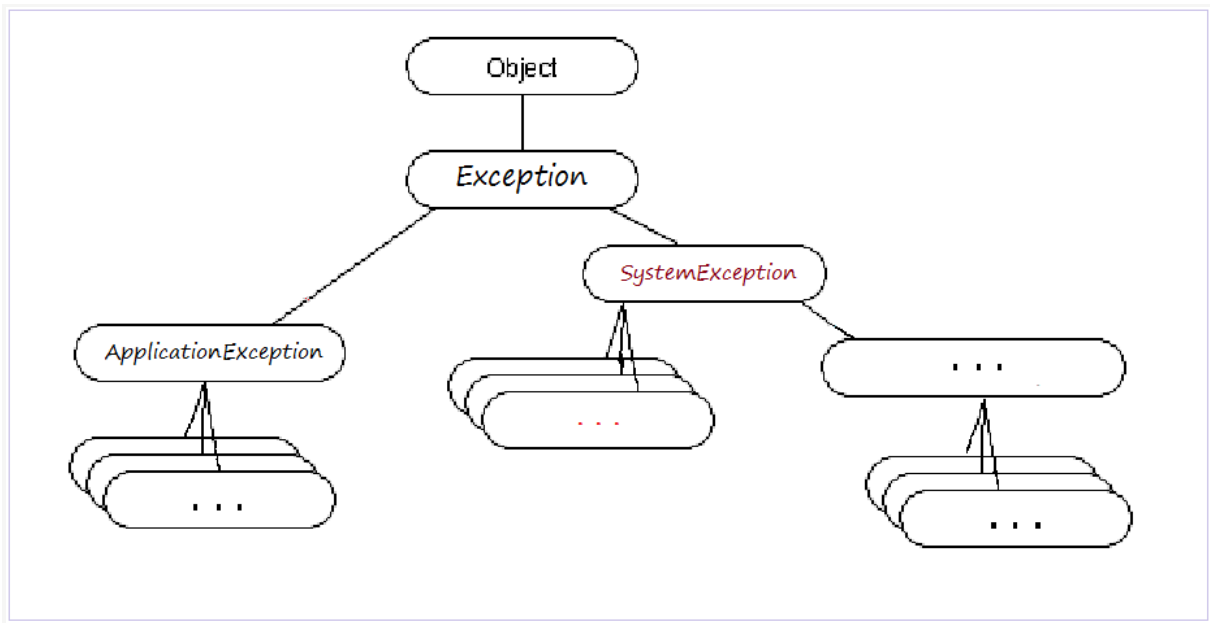
Xử lý ngoại lệ trong C#

Phân cấp các ngoại lệ

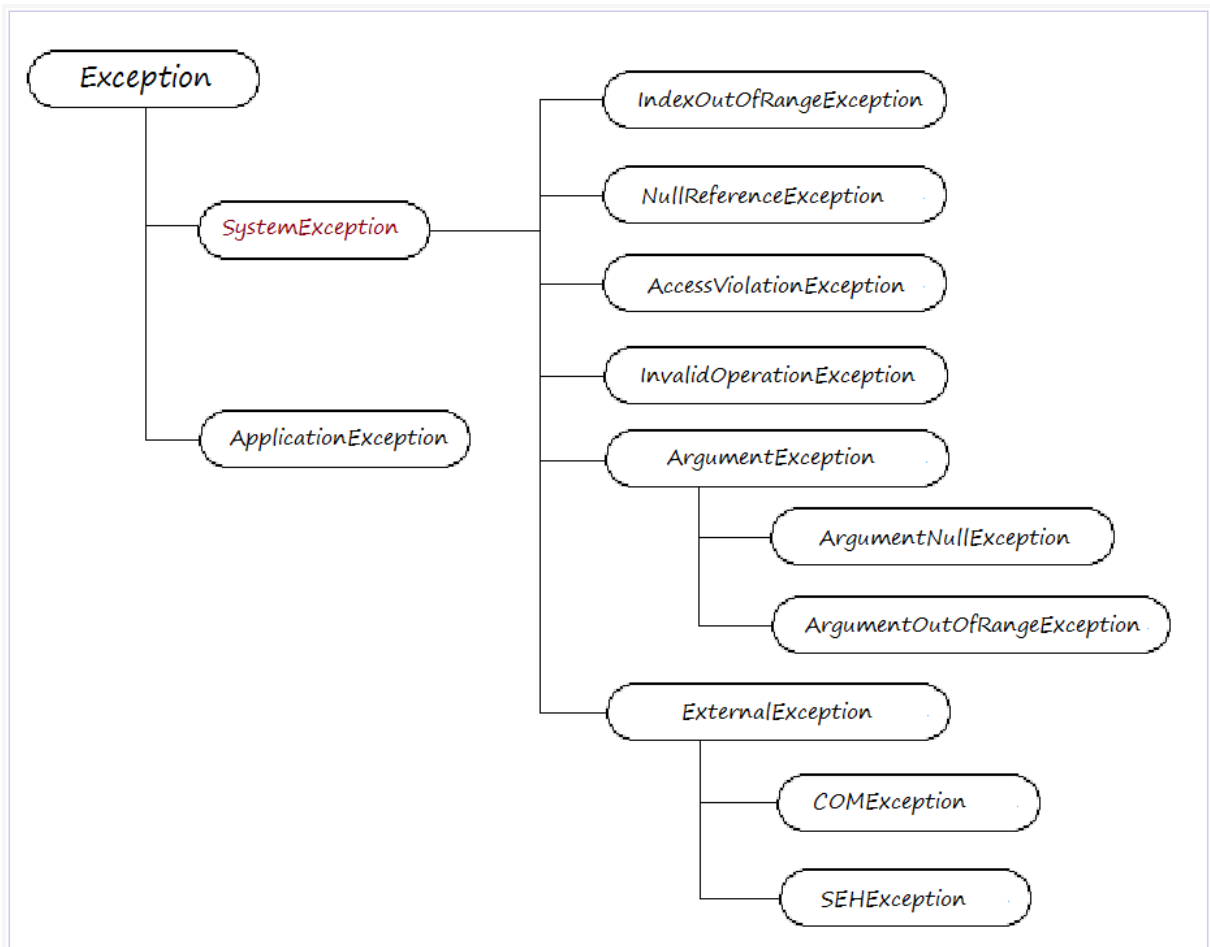
Đây là mô hình sơ đồ phân cấp của Exception trong CSharp.

- Class ở mức cao nhất là **Exception**
- Hai class con trực tiếp là **SystemException** và **ApplicationException**.

Các Exception sẵn có của **CSharp** thông thường được bắt nguồn (derived) từ **SystemException**. Trong khi đó các Exception của người dùng (lập trình viên) nên thừa kế từ **ApplicationException** hoặc từ các class con của nó.



Một số Exception thông dụng sẵn có trong **CSharp**.

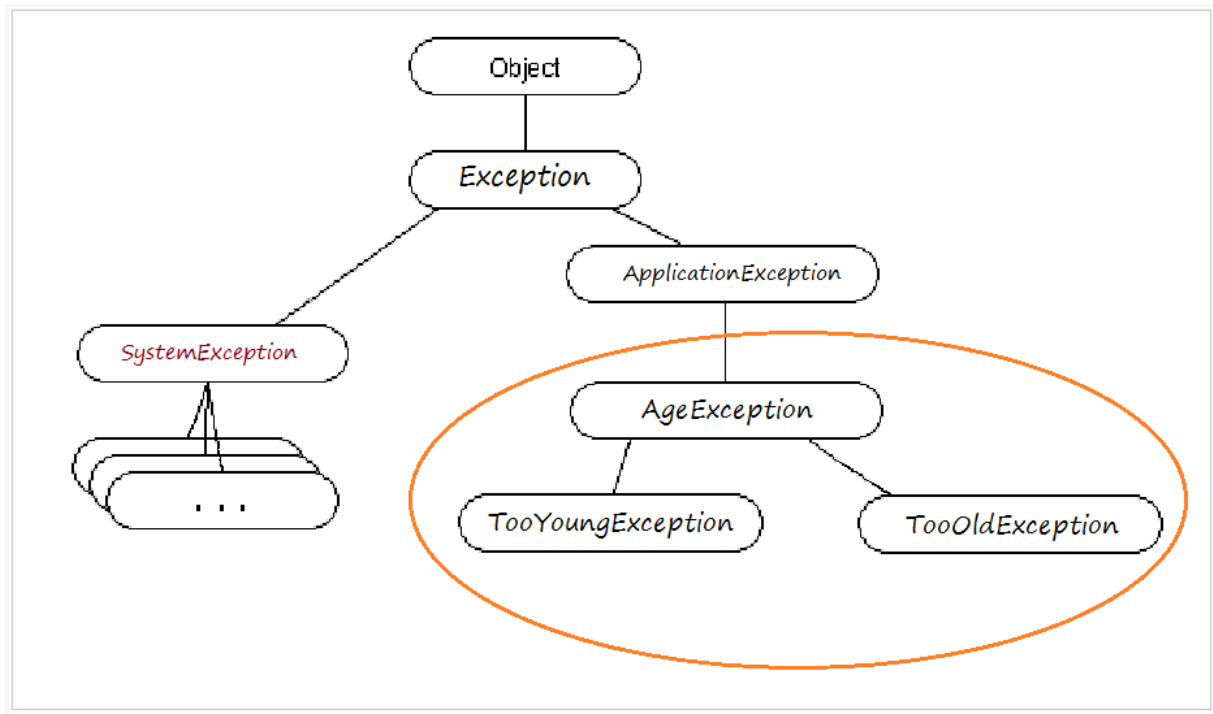


Kiểu ngoại lệ	Mô tả
---------------	-------

Exception	Class cơ bản của mọi ngoại lệ.
SystemException	Class cơ bản của mọi ngoại lệ phát ra tại thời điểm chạy của chương trình.
IndexOutOfRangeException	Được ném ra tại thời điểm chạy khi tham chiếu vào một phần tử của mảng với chỉ số không đúng.
NullReferenceException	Ném ra tại thời điểm chạy khi một đối tượng null được tham chiếu.
AccessViolationException	Ném ra tại thời điểm chạy khi tham chiếu vào vùng bộ nhớ không hợp lệ.
InvalidOperationException	Ném ra bởi phương thức khi ở trạng thái không hợp lệ.
ArgumentException	Class cơ bản cho các ngoại lệ liên quan tới đối số.
ArgumentNullException	Class này là con của ArgumentException , nó được ném ra bởi phương thức mà không cho phép thông số null truyền vào.
ArgumentOutOfRangeException	Class này là con của ArgumentException , nó được ném ra bởi phương thức khi một đối số không thuộc phạm vi cho phép truyền vào nó.
ExternalException	Class cơ bản cho các ngoại lệ xảy ra hoặc nhắm tới môi trường bên ngoài thời gian chạy.
COMException	Class này mở rộng từ ExternalException , ngoại lệ đóng gói thông tin COM.
SEHException	Class này mở rộng từ ExternalException , nó tóm lược các ngoại lệ từ Win32.

Bắt ngoại lệ thông qua try-catch

Chúng ta viết một exception thừa kế từ class **ApplicationException**.



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

namespace ExceptionTutorial
{
    class AgeException : ApplicationException
    {
        public AgeException(String message)
            : base(message)
        {
        }
    }

    class TooYoungException : AgeException
    {
        public TooYoungException(String message)
            : base(message)
        {
        }
    }
}
```

```

class TooOldException : AgeException
{

    public TooOldException(String message)
        : base(message)
    {

    }

}
}

```

Và class **AgeUtils** có method tĩnh dùng cho việc kiểm tra tuổi

```

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

namespace ExceptionTutorial
{
    class AgeUtils
    {
        // Method này làm nhiệm vụ kiểm tra tuổi.
        // Nếu tuổi nhỏ hơn 18 method sẽ ném ra ngoại lệ TooYoungException
        // Nếu tuổi lớn hơn 40 method sẽ ném ra ngoại lệ TooOldException
        public static void checkAge(int age)
        {
            if (age < 18)
            {
                // Nếu tuổi nhỏ hơn 18, ngoại lệ sẽ được ném ra
                // Method này kết thúc tại đây.
                throw new TooYoungException("Age " + age + " too young");
            }
            else if (age > 40)
            {
                // Nếu tuổi lớn hơn 40, ngoại lệ sẽ được ném ra.
                // Method này kết thúc tại đây.
                throw new TooOldException("Age " + age + " too old");
            }
            // Nếu tuổi nằm trong khoảng 18-40.
            // Đoạn code này sẽ được chạy.
            Console.WriteLine("Age " + age + " OK!");
        }
    }
}

```

```

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

namespace ExceptionTutorial
{
    class TryCatchDemo1
    {
        public static void Main(string[] args)
        {
            // Bắt đầu tuyển dụng
            Console.WriteLine("Start Recruiting ...");
            // Kiểm tra tuổi của bạn.
            Console.WriteLine("Check your Age");
            int age = 50;

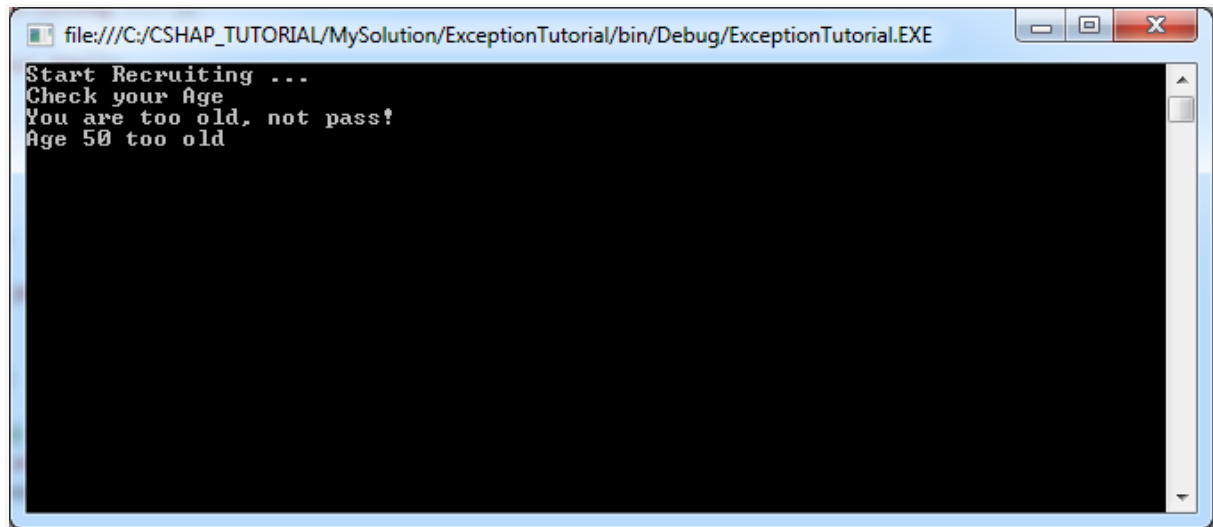
            try
            {
                AgeUtils.checkAge(age);

                Console.WriteLine("You pass!");
            }
            catch (TooYoungException e)
            {
                // Thông báo về ngoại lệ "quá trẻ" ..
                Console.WriteLine("You are too young, not pass!");
                Console.WriteLine(e.Message);
            }
            catch (TooOldException e)
            {
                // Thông báo về ngoại lệ "quá nhiều tuổi" ..
                Console.WriteLine("You are too old, not pass!");
                Console.WriteLine(e.Message);
            }

            Console.Read();
        }
    }
}

```

```
}
```



Ví dụ dưới đây, chúng ta sẽ bắt các ngoại lệ thông qua ngoại lệ ở cấp cao hơn. Ở cấp cao hơn nó sẽ tóm được ngoại lệ đó và tất cả các ngoại lệ con.

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

namespace ExceptionTutorial
{
    class TryCatchDemo2
    {
        public static void Main(string[] args)
        {

            // Bắt đầu tuyển dụng
            Console.WriteLine("Start Recruiting ...");
            // Kiểm tra tuổi của bạn.
            Console.WriteLine("Check your Age");
            int age = 15;

            try
            {

                // Chỗ này có thể bị ngoại lệ TooOldException,
                // hoặc TooYoungException
                AgeUtils.checkAge(age);

                Console.WriteLine("You pass!");

            }
            catch (AgeException e)
            {
            }
```

```

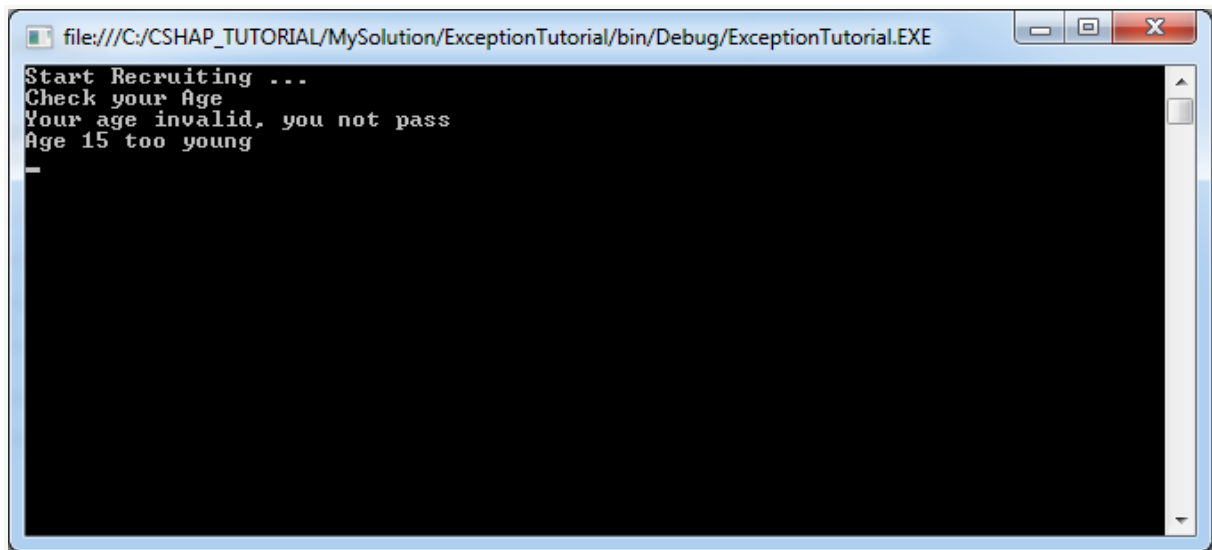
        // Nếu có ngoại lệ xảy ra, kiểu AgeException
        // Khối catch này sẽ được chạy

        Console.WriteLine("Your age invalid, you not pass");
        Console.WriteLine(e.Message);

    }

    Console.Read();
}
}
}

```



Khối try-catch-finally

```

try {

    // Làm gì đó tại đây

} catch (Exception1 e) {

    // Làm gì đó tại đây

} catch (Exception2 e) {

    // Làm gì đó tại đây

} finally {

    // Khối finally luôn luôn được thực thi
    // Làm gì đó tại đây.
}

```

```

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

namespace ExceptionTutorial
{
    class TryCatchFinallyDemo
    {
        public static void Main(string[] args)
        {
            String text = "001234A2";

            int value = toInteger(text);

            Console.WriteLine("Value= " + value);

            Console.Read();
        }

        public static int toInteger(String text)
        {
            try
            {
                Console.WriteLine("Begin parse text: " + text);

                // Tại đây có thể phát sinh ngoại lệ FormatException
                int value = int.Parse(text);

                return value;
            }
            catch (FormatException e)
            {
                // Trong trường hợp 'text' không phải là số.
                // Khối catch này sẽ được thực thi.
                Console.WriteLine("Number format exception: " + e.Message);

                // FormatException xảy ra, trả về 0.
                return 0;
            }
        }
    }
}

```



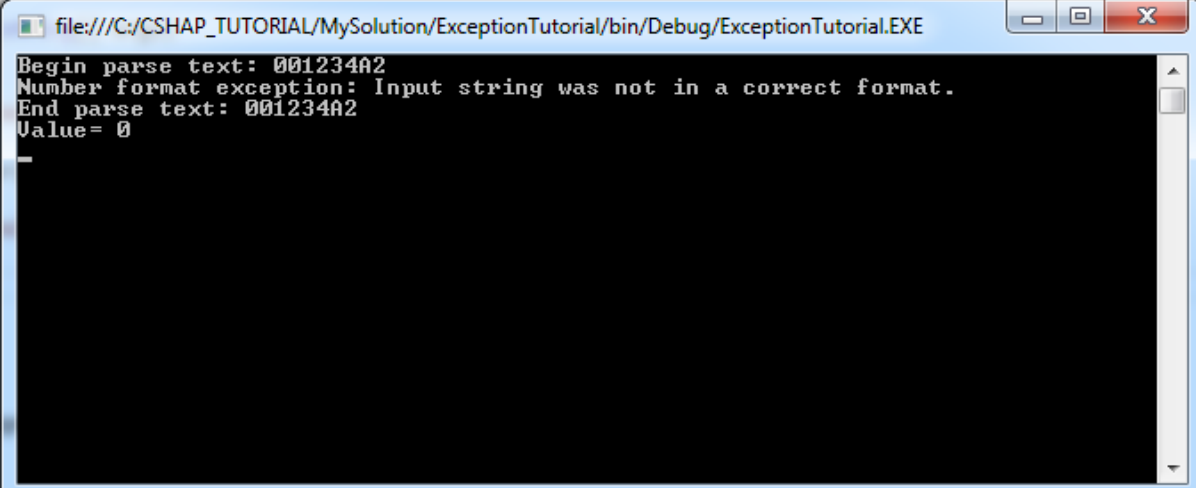
```
        finally
        {

            Console.WriteLine("End parse text: " + text);

        }

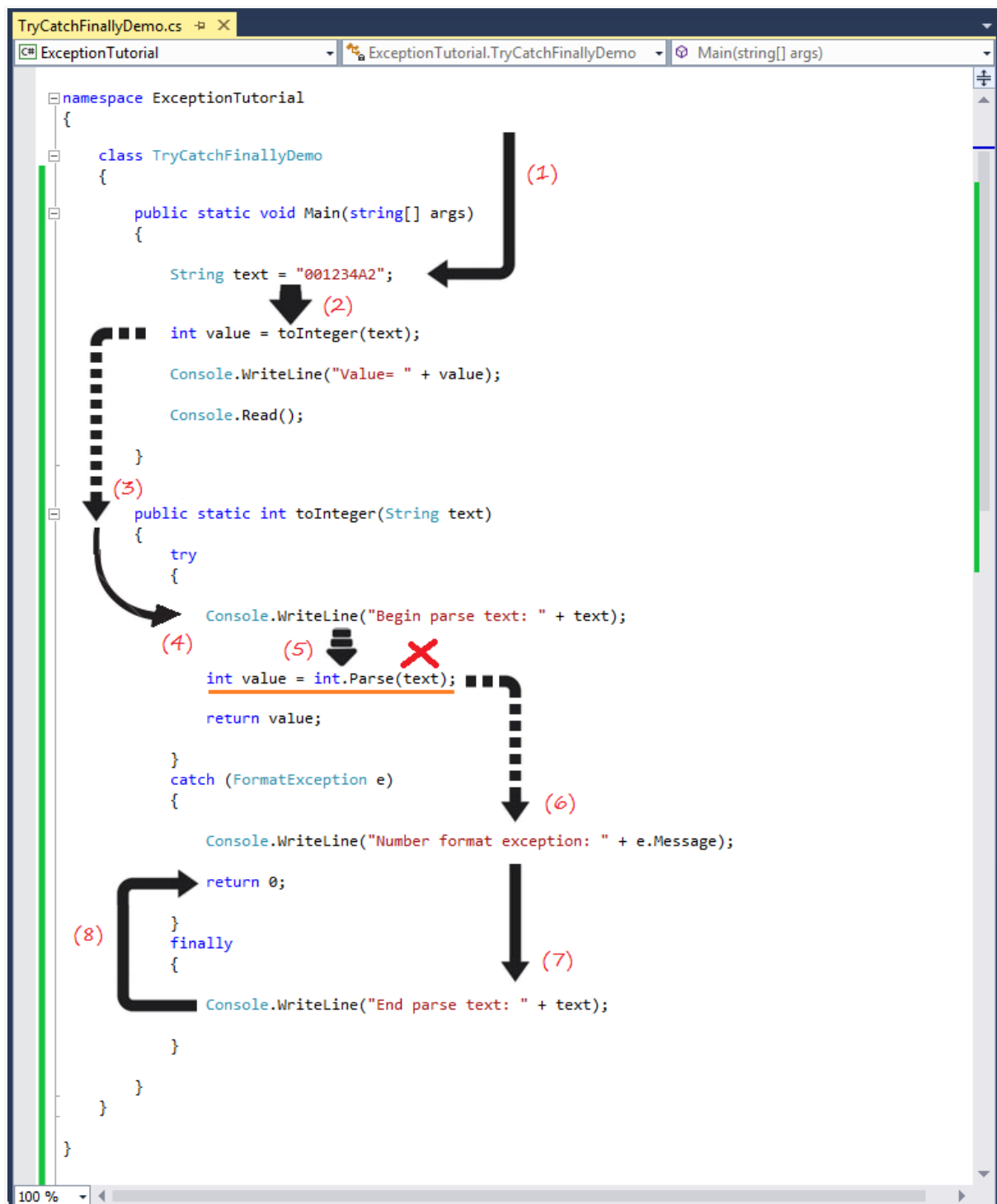
    }

}
```



```
file:///C:/CSHAP_TUTORIAL/MySolution/ExceptionTutorial/bin/Debug/ExceptionTutorial.EXE
Begin parse text: 001234A2
Number format exception: Input string was not in a correct format.
End parse text: 001234A2
Value= 0
```

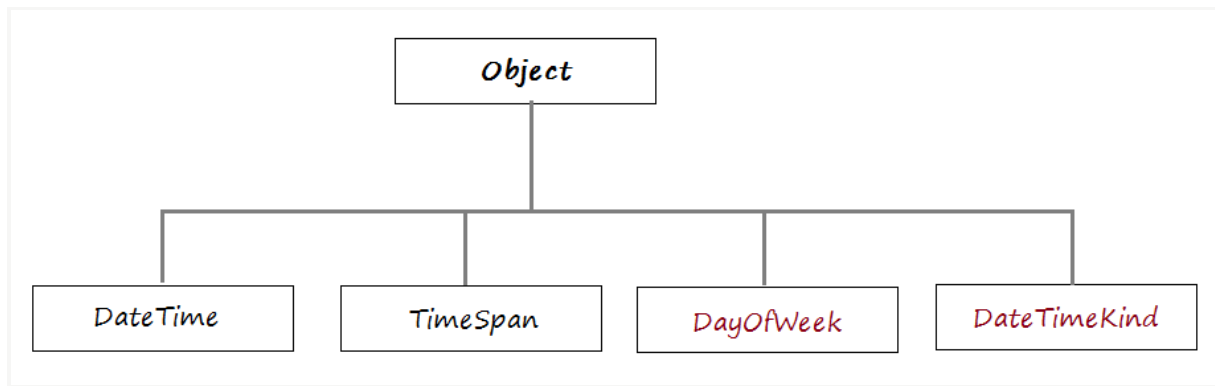
Đây là sơ luồng đi của chương trình. Khối **finally** luôn được thực thi.



Date Time trong C#

Các class liên quan Date, Time trong C#

Trong .NET Framework, `System.DateTime` là một class đại diện cho ngày tháng và thời gian, giá trị của nó nằm trong khoảng 12:00:00 đêm ngày 01-01-0001 tới 11:59:59 tối ngày 31-12-9999.



Có nhiều phương thức khởi tạo (constructor) để bạn khởi tạo một đối tượng **DateTime**.

```
public DateTime(int year, int month, int day)
```

```
public DateTime(int year, int month, int day, Calendar calendar)
```

```
public DateTime(int year, int month, int day, int hour, int minute, int second)
```

```
public DateTime(int year, int month, int day, int hour, int minute, int second, Calendar calendar)
```

```
public DateTime(int year, int month, int day, int hour, int minute, int second, DateTimeKind kind)
```

```
public DateTime(int year, int month, int day, int hour, int minute, int second, int millisecond)
```

```
public DateTime(int year, int month, int day, int hour,
                int minute, int second, int millisecond, Calendar
                calendar)
```

```
public DateTime(int year, int month, int day, int hour,
                int minute, int second, int millisecond, Calendar
                calendar, DateTimeKind kind)
```

```
public DateTime(int year, int month, int day, int hour,
                int minute, int second, int millisecond, DateTimeKind
                kind)
```

```
public DateTime(long ticks)
```

```
public DateTime(long ticks, DateTimeKind kind)
```

Now là một thuộc tính tĩnh của **DateTime** nó trả về đối tượng **DateTime** mô tả thời điểm hiện tại.

Các thuộc tính DateTime

Thuộc tính	Kiểu dữ liệu	Mô tả
Date	DateTime	Lấy ra thành phần date (Chỉ chứa thông tin ngày tháng năm) của đối tượng này.
Day	int	Lấy ra ngày của tháng được mô tả bởi đối tượng này.
DayOfWeek	DayOfWeek	Lấy ra ngày trong tuần được đại diện bởi đối tượng này.
DayOfYear	int	Lấy ra ngày của năm được đại diện bởi đối tượng này.
Hour	int	Lấy ra thành phần giờ được đại diện bởi đối tượng này.
Kind	DateTimeKind	Lấy giá trị cho biết liệu thời gian được đại diện bởi đối tượng này dựa trên thời gian địa phương, Coordinated Universal Time (UTC), hoặc không.
Millisecond	int	Lấy ra thành phần mili giây được đại diện bởi đối tượng này.
Minute	int	Lấy ra thành phần phút được đại diện bởi đối tượng này.
Month	int	Lấy ra thành phần tháng được đại diện bởi đối tượng này.
Now	DateTime	Lấy ra đối tượng DateTime được sét thông tin ngày tháng thời gian hiện tại theo máy tính địa phương.
Second	int	Lấy ra thành phần giây được đại diện bởi đối tượng này.
Ticks	long	Lấy ra số lượng "tick" được đại diện bởi đối tượng này. (1 phút = 600 triệu tick)
TimeOfDay	TimeSpan	Trả về thời gian của ngày được đại diện bởi đối tượng này.
Today	DateTime	Trả về ngày hiện tại.

UtcNow	DateTime	Trả về đối tượng DateTime được sét thời gian hiện tại của máy tính, được thể hiện dưới dạng Coordinated Universal Time (UTC).
Year	int	Lấy ra thành phần năm được đại diện bởi đối tượng này.

```

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

namespace DateTimeTutorial
{
    class DateTimePropertiesExample
    {

        public static void Main(string[] args)
        {

            // Tạo một đối tượng DateTime (năm, tháng, ngày, giờ, phút,
giây).
            DateTime aDateTime = new DateTime(2005, 11, 20, 12, 1, 10);

            // In ra các thông tin:

            Console.WriteLine("Day:{0}", aDateTime.Day);
            Console.WriteLine("Month:{0}", aDateTime.Month);
            Console.WriteLine("Year:{0}", aDateTime.Year);
            Console.WriteLine("Hour:{0}", aDateTime.Hour);
            Console.WriteLine("Minute:{0}", aDateTime.Minute);
            Console.WriteLine("Second:{0}", aDateTime.Second);
            Console.WriteLine("Millisecond:{0}", aDateTime.Millisecond);

            // Enum {Monday, Tuesday,... Sunday}
            DayOfWeek dayOfWeek = aDateTime.DayOfWeek;

            Console.WriteLine("Day of Week:{0}", dayOfWeek );

            Console.WriteLine("Day of Year: {0}", aDateTime.DayOfYear);

            // Một đối tượng chỉ mô tả thời gian (giờ phút giây,...)
            TimeSpan timeOfDay = aDateTime.TimeOfDay;

            Console.WriteLine("Time of Day:{0}", timeOfDay);

```

```

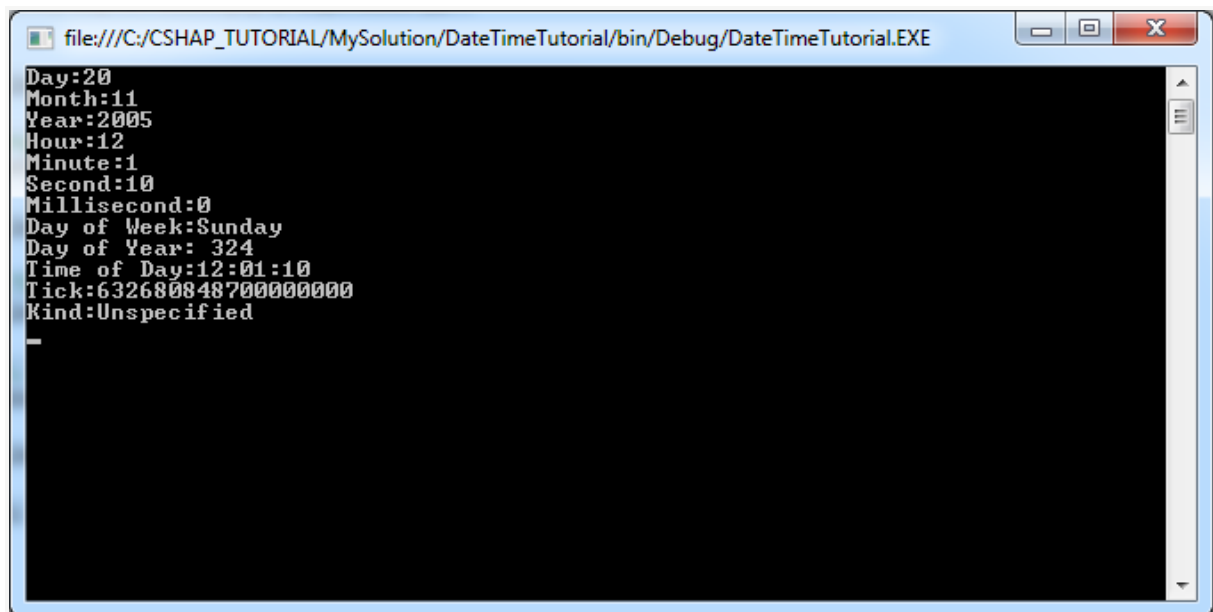
        // Quy đổi ra Ticks (1 giây = 10.000.000 Ticks)
        Console.WriteLine("Tick:{0}", aDateTime.Ticks);

        // {Local, Itc, Unspecified}
        DateTimeKind kind = aDateTime.Kind;

        Console.WriteLine("Kind:{0}", kind);

        Console.Read();
    }
}
}

```



Thêm và bớt thời gian

DateTime cung cấp các phương thức cho phép bạn thêm, hoặc trừ một khoảng thời gian. **TimeSpan** là một class chứa thông tin một khoảng thời gian, nó có thể tham gia như một tham số trong các phương thức thêm bớt thời gian của **DateTime**.

```

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

namespace DateTimeTutorial
{
    class AddSubtractExample
    {

```

```

public static void Main(string[] args)
{
    // Thời điểm hiện tại.
    DateTime aDateTime = DateTime.Now;

    Console.WriteLine("Now is " + aDateTime);

    // Một khoảng thời gian.
    // 1 giờ + 1 phút
    TimeSpan aInterval = new System.TimeSpan(0, 1, 1, 0);

    // Thêm khoảng thời gian.
    DateTime newTime = aDateTime.Add(aInterval);

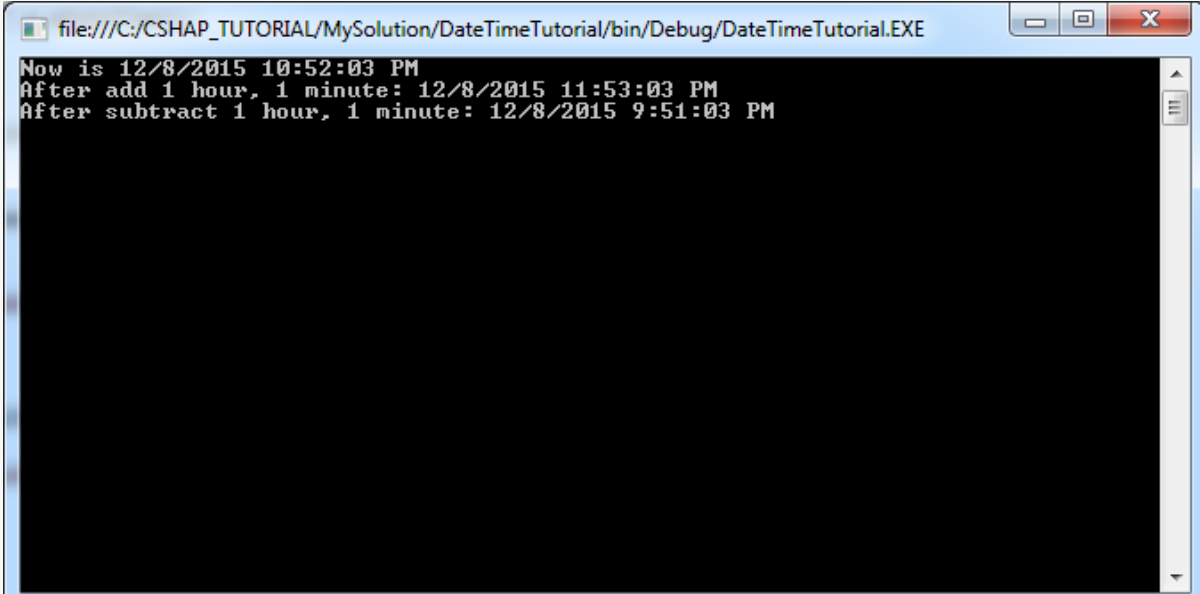
    Console.WriteLine("After add 1 hour, 1 minute: " + newTime);

    // Trừ khoảng thời gian.
    newTime = aDateTime.Subtract(aInterval);

    Console.WriteLine("After subtract 1 hour, 1 minute: " +
newTime);

    Console.Read();
}
}
}

```



```

file:///C:/CSHAP_TUTORIAL/MySolution/DateTimeTutorial/bin/Debug/DateTimeTutorial.EXE
Now is 12/8/2015 10:52:03 PM
After add 1 hour, 1 minute: 12/8/2015 11:53:03 PM
After subtract 1 hour, 1 minute: 12/8/2015 9:51:03 PM

```

Class **DateTime** cũng có các phương thức cho phép thêm bớt một loại đơn vị thời gian chẳng hạn:

- AddYears
- AddDays
- AddMinutes
- ...

Đôi khi bạn cần tìm ngày đầu tiên hoặc ngày cuối cùng của một tháng hoặc một năm cụ thể. Chẳng hạn bạn đặt ra câu hỏi tháng 2 năm 2015 là ngày bao nhiêu? ngày 28 hay 29. Ví dụ dưới đây có một vài phương thức tiện ích để làm điều này:

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

namespace DateTimeTutorial
{
    class FirstLastDayDemo
    {
        public static void Main(string[] args)
        {

            Console.WriteLine("Today is " + DateTime.Today);

            DateTime yesterday = GetYesterday();

            Console.WriteLine("Yesterday is " + yesterday);

            // Ngày đầu tiên của tháng 2 năm 2015
            DateTime aDateTime = GetFirstDayInMonth(2015, 2);

            Console.WriteLine("First day of 2-2015: " + aDateTime);

            // Ngày cuối cùng của tháng 2 năm 2015
            aDateTime = GetLastDayInMonth(2015, 2);

            Console.WriteLine("Last day of 2-2015: " + aDateTime);

            // Ngày đầu tiên của năm 2015
            aDateTime = GetFirstDayInYear(2015);

            Console.WriteLine("First day year 2015: " + aDateTime);

            // Ngày cuối cùng của năm 2015
            aDateTime = GetLastDayInYear(2015);

            Console.WriteLine("First day year 2015: " + aDateTime);
```



```

        Console.Read();
    }

    // Trả về ngày hôm qua.
    public static DateTime GetYesterday()
    {
        // Ngày hôm nay.
        DateTime today = DateTime.Today;
        // Trả về ngày trước 1 ngày.
        return today.AddDays(-1);
    }

    // Trả về ngày đầu tiên của năm
    public static DateTime GetFirstDayInYear(int year)
    {
        DateTime aDateTime = new DateTime(year, 1, 1);
        return aDateTime;
    }

    // Trả về ngày cuối cùng của năm.
    public static DateTime GetLastDayInYear(int year)
    {
        DateTime aDateTime = new DateTime(year + 1, 1, 1);

        // Trừ đi một ngày.
        DateTime retDateTime = aDateTime.AddDays(-1);

        return retDateTime;
    }

    // Trả về ngày đầu tiên của tháng
    public static DateTime GetFistDayInMonth(int year, int month)
    {
        DateTime aDateTime = new DateTime(year, month, 1);

        return aDateTime;
    }

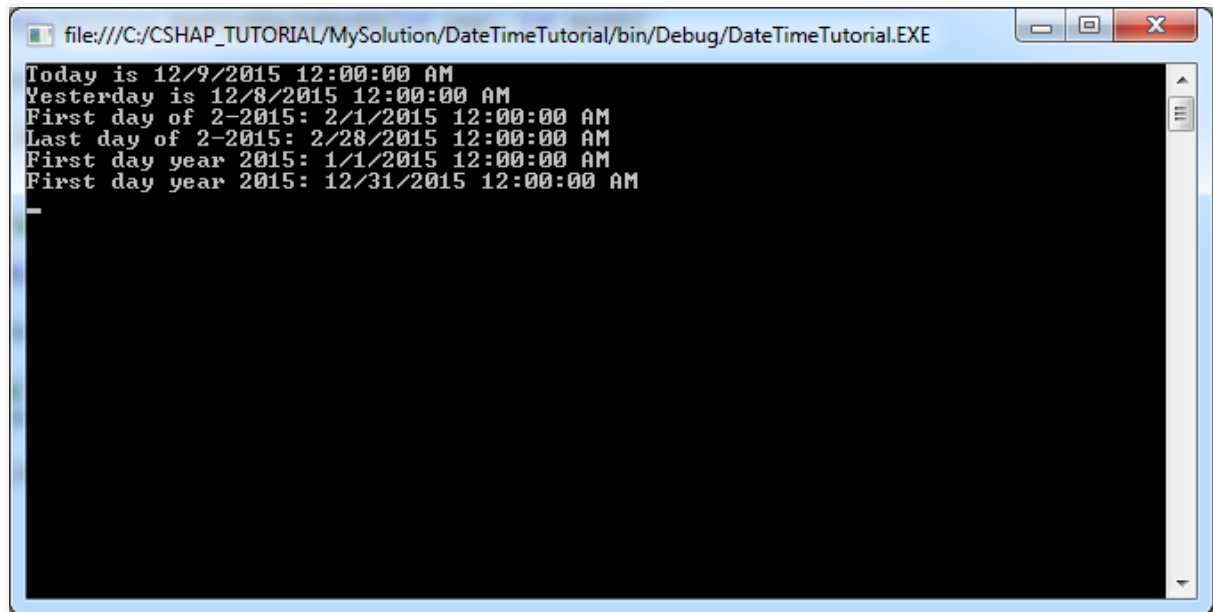
    // Trả về ngày cuối cùng của tháng.
    public static DateTime GetLastDayInMonth(int year, int month)
    {
        DateTime aDateTime = new DateTime(year, month, 1);

        // Cộng thêm 1 tháng và trừ đi một ngày.
        DateTime retDateTime = aDateTime.AddMonths(1).AddDays(-1);

        return retDateTime;
    }
}

```

```
}  
  
}
```



Đo khoảng thời gian

Bạn có hai đối tượng **DateTime**, bạn có thể tính được khoảng thời gian giữa 2 đối tượng này, kết quả nhận được là một đối tượng **TimeSpan**

```
using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;  
using System.Threading.Tasks;  
  
namespace DateTimeTutorial  
{  
    class IntervalDemo  
    {  
        public static void Main(string[] args)  
        {  
            // Thời điểm hiện tại.  
            DateTime aDateTime = DateTime.Now;  
  
            // Thời điểm năm 2000  
            DateTime y2K = new DateTime(2000,1,1);  
  
            // Khoảng thời gian từ năm 2000 tới nay.  
            TimeSpan interval = aDateTime.Subtract(y2K);  
        }  
    }  
}
```

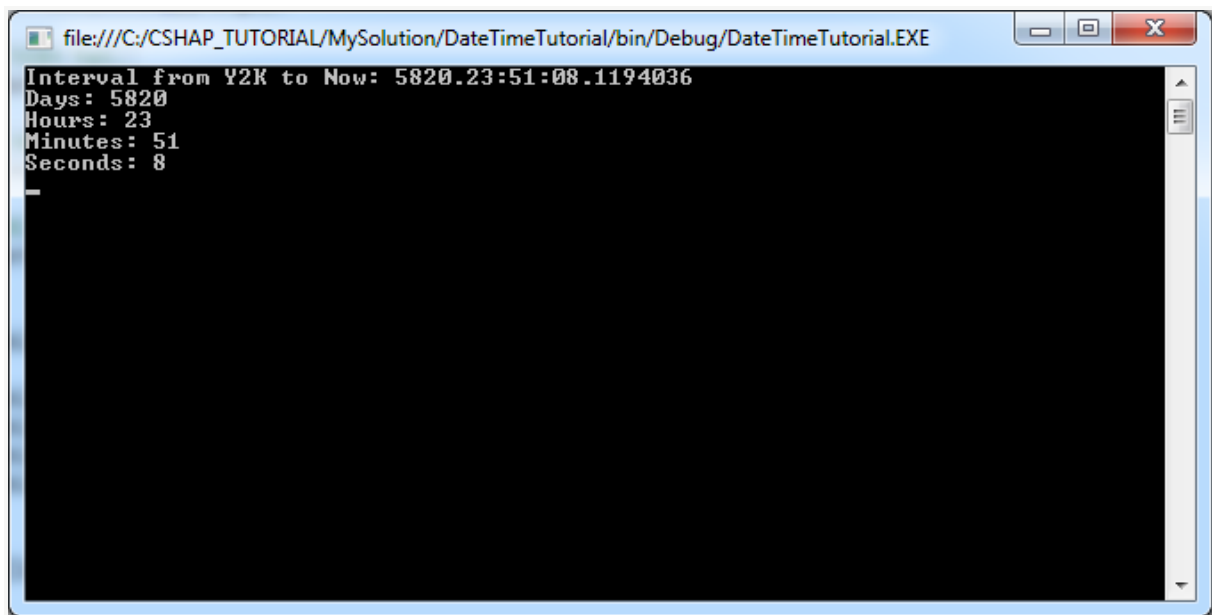
```

        Console.WriteLine("Interval from Y2K to Now: " + interval);

        Console.WriteLine("Days: " + interval.Days);
        Console.WriteLine("Hours: " + interval.Hours);
        Console.WriteLine("Minutes: " + interval.Minutes);
        Console.WriteLine("Seconds: " + interval.Seconds);

        Console.Read();
    }
}

```



So sánh hai đối tượng DateTime

DateTime có một phương thức tĩnh là *Compare*. Phương thức dùng để so sánh 2 đối tượng DateTime xem đối tượng nào sớm hơn đối tượng còn lại:

```

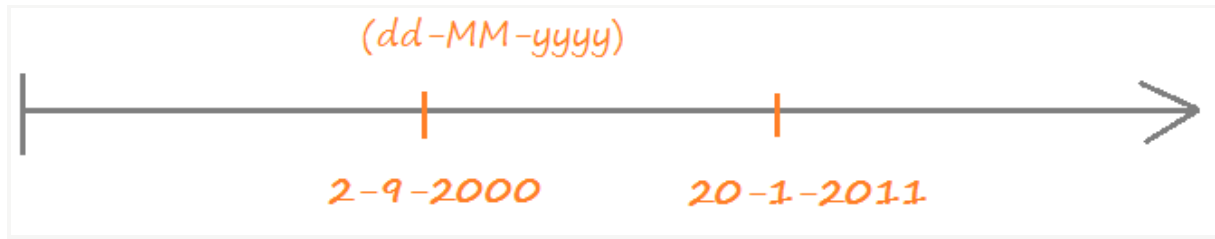
// Nếu giá trị < 0 nghĩa là firstDateTime sớm hơn (đứng trước)
// Nếu giá trị > 0 nghĩa là secondDateTime sớm hơn (đứng trước).
// Nếu giá trị = 0 nghĩa là 2 đối tượng này giống nhau về mặt thời gian.

```

```

public static int Compare(DateTime firstDateTime, DateTime secondDateTime);

```



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

namespace DateTimeTutorial
{
    class CompareDateTimeExample
    {
        public static void Main(string[] args)
        {
            // Thời điểm hiện tại.
            DateTime firstDateTime = new DateTime(2000, 9, 2);

            DateTime secondDateTime = new DateTime(2011, 1, 20);

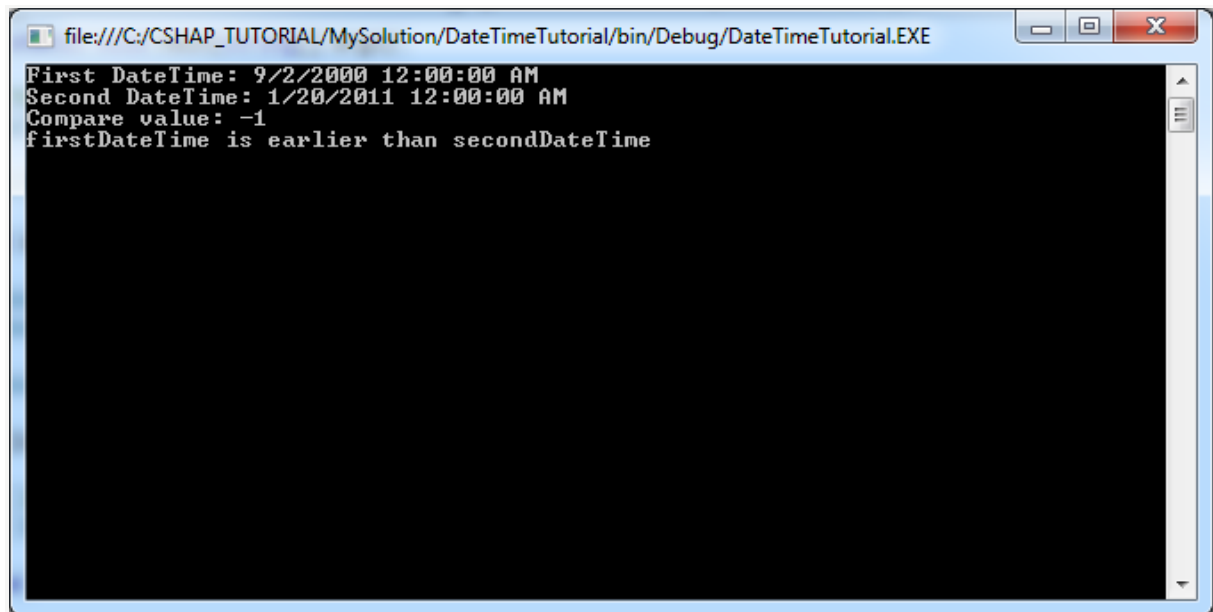
            int compare = DateTime.Compare(firstDateTime, secondDateTime);

            Console.WriteLine("First DateTime: " + firstDateTime);
            Console.WriteLine("Second DateTime: " + secondDateTime);

            Console.WriteLine("Compare value: " + compare); // -1

            if (compare < 0)
            {
                // firstDateTime sớm hơn secondDateTime
                Console.WriteLine("firstDateTime is earlier than
secondDateTime");
            }
            else
            {
                // firstDateTime muộn hơn secondDateTime
                Console.WriteLine("firstDateTime is later than
secondDateTime");
            }

            Console.Read();
        }
    }
}
```



```
file:///C:/CSHAP_TUTORIAL/MySolution/DateTimeTutorial/bin/Debug/DateTimeTutorial.EXE
First DateTime: 9/2/2000 12:00:00 AM
Second DateTime: 1/20/2011 12:00:00 AM
Compare value: -1
firstDateTime is earlier than secondDateTime
```

Định dạng tiêu chuẩn DateTime

Định dạng DateTime nghĩa là chuyển đổi đối tượng DateTime thành một string theo một khuôn mẫu nào đó, chẳng hạn theo định dạng ngày/tháng/năm, ... hoặc định dạng dựa vào địa phương (locale) cụ thể.

Phương thức `GetDateTimeFormats` của DateTime:

- Chuyển đổi giá trị của đối tượng này (DateTime) thành một mảng các string đã định dạng theo các chuẩn được hỗ trợ.

```
DateTime aDateTime = new DateTime(2015, 12, 20, 11, 30, 50);
```

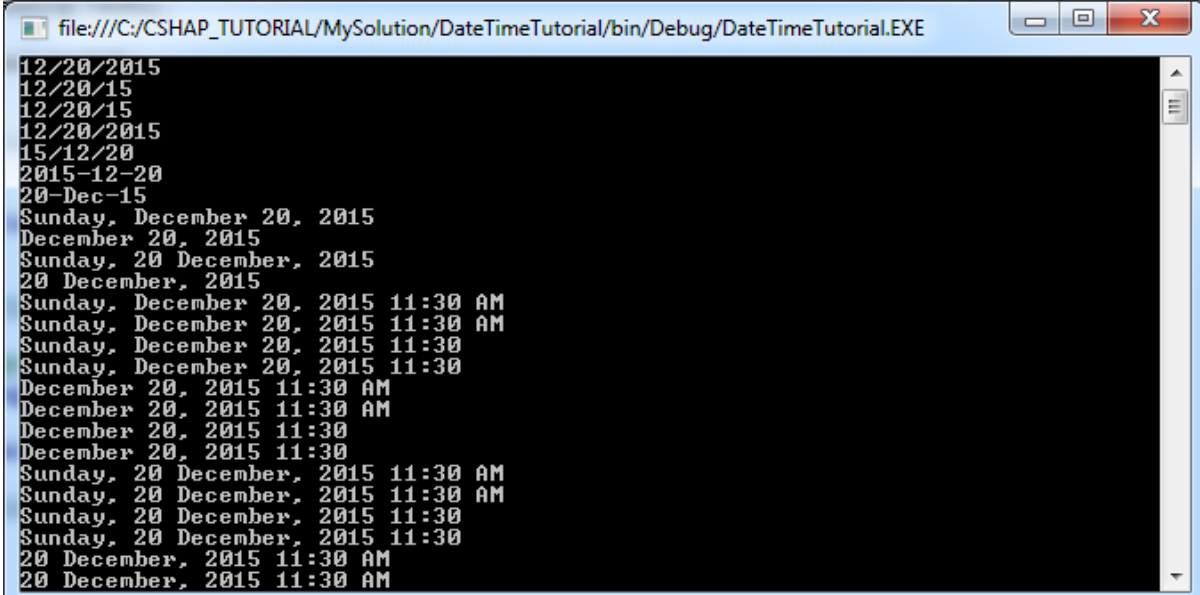
```
string[] formattedStrings =  
    aDateTime.GetDateTimeFormats();
```



```
12/20/2015  
12/20/15  
...  
Sunday, December 20, 2015  
...  
11:30:50 AM  
...  
12/20/2015 11:30 AM  
12/20/2015 11:30 AM  
...
```

```
using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;  
using System.Threading.Tasks;  
  
namespace DateTimeTutorial  
{  
    class AllStandardFormatsDemo  
    {  
        public static void Main(string[] args)  
        {  
  
            DateTime aDateTime = new DateTime(2015, 12, 20, 11, 30, 50);  
  
            // Một mảng các string kết quả định dạng được hỗ trợ.  
            string[] formattedStrings = aDateTime.GetDateTimeFormats();  
  
            foreach (string format in formattedStrings)  
            {  
                Console.WriteLine(format);  
            }  
  
            Console.Read();  
        }  
    }  
}
```

}



```
file:///C:/CSHAP_TUTORIAL/MySolution/DateTimeTutorial/bin/Debug/DateTimeTutorial.EXE
12/20/2015
12/20/15
12/20/15
12/20/2015
15/12/20
2015-12-20
20-Dec-15
Sunday, December 20, 2015
December 20, 2015
Sunday, 20 December, 2015
20 December, 2015
Sunday, December 20, 2015 11:30 AM
Sunday, December 20, 2015 11:30 AM
Sunday, December 20, 2015 11:30
Sunday, December 20, 2015 11:30
December 20, 2015 11:30 AM
December 20, 2015 11:30 AM
December 20, 2015 11:30
December 20, 2015 11:30
Sunday, 20 December, 2015 11:30 AM
Sunday, 20 December, 2015 11:30 AM
Sunday, 20 December, 2015 11:30
Sunday, 20 December, 2015 11:30
20 December, 2015 11:30 AM
20 December, 2015 11:30 AM
```

Ví dụ trên liệt kê ra các string sau khi định dạng một đối tượng `DateTime` theo các tiêu chuẩn có sẵn được hỗ trợ bởi `.NET`. Để lấy định dạng theo một mẫu cụ thể bạn sử dụng một trong các phương thức sau:

Methods	Description
<code>ToString(String, IFormatProvider)</code>	Chuyển đổi các giá trị của đối tượng <code>DateTime</code> hiện thành chuỗi đại diện tương đương của nó bằng cách sử dụng định dạng quy định (Tham số <code>String</code>) và các thông tin định dạng văn hóa cụ thể (Tham số <code>IFormatProvider</code>).
<code>ToString(IFormat Provider)</code>	Chuyển đổi giá trị của đối tượng <code>DateTime</code> hiện tại thành một string tương ứng với thông tin định dạng văn hóa (culture) cho bởi tham số.
<code>ToString(String)</code>	Chuyển đổi các giá trị của đối tượng <code>DateTime</code> hiện thành một chuỗi tương đương của nó bằng cách sử dụng định dạng quy định và các quy ước định dạng của các nền văn hóa hiện nay.

*Định dạng tiêu chuẩn 'd'.
Tùy theo văn hóa mỗi nước.*

*M/d/yyyy
M/d/yy
MM/dd/yy
MM/dd/yyyy
yy/MM/dd
yyyy-MM-dd
dd-MMM-yy*

Ví dụ dưới đây định dạng DateTime theo định dạng 'd', và chỉ định rõ văn hóa (culture) trong tham số.

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

namespace DateTimeTutorial
{
    class SimpleDateTimeFormat
    {
        public static void Main(string[] args)
        {

            DateTime aDateTime = new DateTime(2015, 12, 20, 11, 30, 50);

            Console.WriteLine("DateTime: " + aDateTime);

            String d_formatString = aDateTime.ToString("d");

            Console.WriteLine("Format 'd' : " + d_formatString);

            // Theo văn hóa Mỹ.
            CultureInfo enUs = new CultureInfo("en-US");
```



```

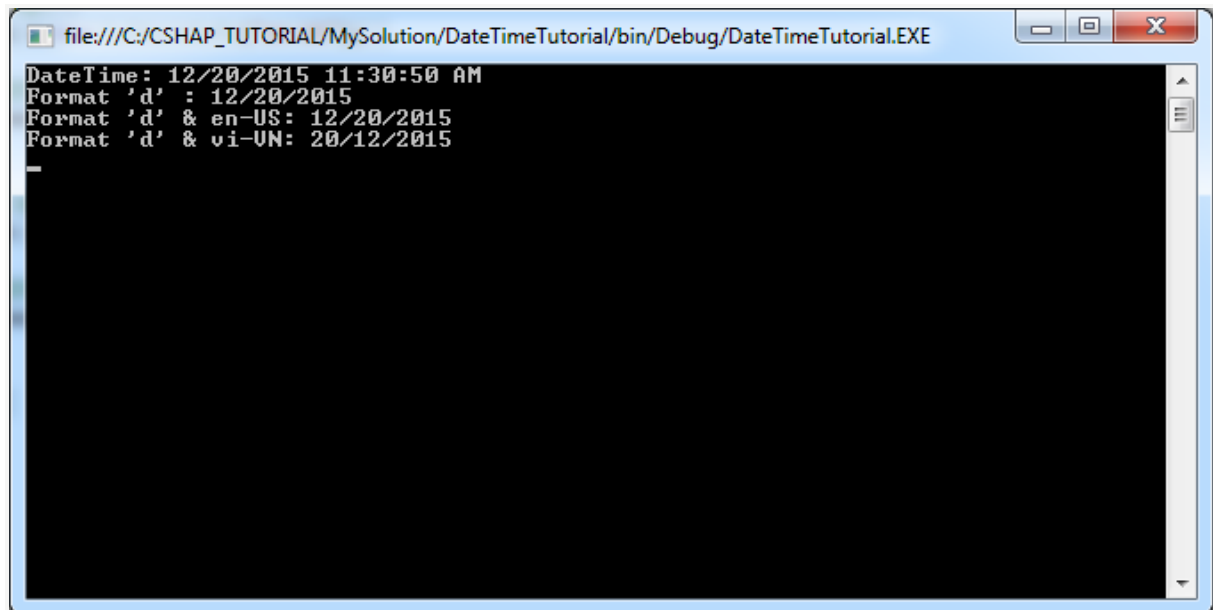
// ==> 12/20/2015 (MM/dd/yyyy)
Console.WriteLine("Format 'd' & en-US: " +
aDateTime.ToString("d", enUs));

// Theo văn hóa Việt Nam.
CultureInfo viVn = new CultureInfo("vi-VN");

// ==> 12/20/2015 (dd/MM/yyyy)
Console.WriteLine("Format 'd' & vi-VN: " +
aDateTime.ToString("d", viVn));

Console.Read();
}
}
}

```



Các ký tự định dạng tiêu chuẩn.

Code	Pattern
"d"	Ngày tháng năm ngắn
"D"	Ngày tháng năm dài
"f"	Ngày tháng năm dài, thời gian ngắn
"F"	Ngày tháng năm dài, thời gian dài.

"g"	Ngày tháng thời gian nói chung. Thời gian ngắn.
"G"	Ngày tháng thời gian nói chung. Thời gian dài.
"M", "m"	Tháng/ngày.
"O", "o"	Round-trip date/time.
"R", "r"	RFC1123
"s"	Ngày tháng thời gian có thể sắp xếp
"t"	Thời gian ngắn
"T"	Thời gian dài
"u"	Ngày tháng năm có thể sắp xếp phổ biến (Universal sortable date time).
"U"	Ngày tháng năm thời gian dài, phổ biến (Universal full date time).
"Y", "y"	Năm tháng

DateTime.Parse(string)

Nếu bạn có một chuỗi Date định dạng tiêu chuẩn bạn dễ dàng chuyển nó thành đối tượng DateTime thông qua phương thức `DateTime.Parse(string)`. Thông thường các chuỗi ngày tháng thời gian bạn thấy trên Internet là các chuỗi định dạng chuẩn, các cơ sở dữ liệu MySQL hoặc SQL Server cũng sử dụng các định dạng chuẩn để hiển thị ngày tháng thời gian.

```
// Chuỗi thời gian bạn thấy trên Http Header
```

```
string httpTime = "Fri, 21 Feb 2011 03:11:31 GMT";
```

```
// Chuỗi thời gian thấy trên w3.org
```

```
string w3Time = "2016/05/26 14:37:11";
```

```
// Chuỗi thời gian thấy trên nytimes.com
```

```
string nyTime = "Thursday, February 26, 2012";
```

```
// Chuỗi thời gian tiêu chuẩn ISO 8601.
```

```
string isoTime = "2016-02-10";
```

```
// Chuỗi thời gian khi xem ngày giờ tạo/ sửa của file trên windows.
```

```
string windowsTime = "11/21/2015 11:35 PM";
```

```
// Chuỗi thời gian trên Windows Date and Time panel
```

```
string windowsPanelTime = "11:07:03 PM";
```

Tùy biến định dạng DateTime

Tùy biến DateTime --> string:

Sử dụng DateTime.ToString(string):

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

namespace DateTimeTutorial
{
    class CustomDateTimeFormatExample
    {
        public static void Main(string[] args)
        {
            // Kiểu định dạng.
            string format = "dd/MM/yyyy HH:mm:ss";

            DateTime now = DateTime.Now;

            string s = now.ToString(format);

            Console.WriteLine("Now: " + s);

            Console.Read();
        }
    }
}
```

Tùy biến string --> DateTime

```
public static DateTime Parse(string s)

public static DateTime Parse(string s, IFormatProvider provider)

public static DateTime Parse(string s, IFormatProvider provider,
DateTimeStyles styles)

public static bool TryParseExact(string s, string format,
```

```

        IFormatProvider provider, DateTimeStyles style,
        out DateTime result)

```

```

public static bool TryParseExact(string s, string[] formats,
        IFormatProvider provider, DateTimeStyles style,
        out DateTime result)

```

Phương thức	Ví dụ
static DateTime Parse(string)	// Xem thêm phần định dạng chuẩn DateTime. string httpHeaderTime = "Fri, 21 Feb 2011 03:11:31 GMT"; DateTime.Parse(httpHeaderTime);
static DateTime ParseExact(string s, string format, IFormatProvider provider)	string dateString = "20160319 09:57"; DateTime.ParseExact(dateString, "yyyyMMdd HH:mm", null);
static bool TryParseExact(string s, string format, IFormatProvider provider, DateTimeStyles style, out DateTime result)	Phương thức này rất giống với ParseExact, tuy nhiên nó trả về boolean, true nếu chuỗi thời gian là phân tích được, ngược lại trả về false. (Xem thêm ví dụ dưới đây).

```

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

namespace DateTimeTutorial
{
    class ParseDateStringExample
    {

        public static void Main(string[] args)
        {

            string dateString = "20160319 09:57";

            // Sử dụng ParseExact

```

```

        DateTime dateTime = DateTime.ParseExact(dateString, "yyyyMMdd
HH:mm", null);

        Console.WriteLine("Input dateString: " + dateString);

        Console.WriteLine("Parse Results: " +
dateTime.ToString("dd-MM-yyyy HH:mm:ss"));

        // Chuỗi datetime có khoảng trắng phía trước.
        dateString = " 20110319 11:57";

        // Sử dụng TryParseExact.
        // Phương thức trả về true nếu chuỗi dateString có thể phân tích
được.
        // Và trả về giá trị tại tham số 'out dateTime'.
        bool successful = DateTime.TryParseExact(dateString, "yyyyMMdd
HH:mm", null,
            System.Globalization.DateTimeStyles.AllowLeadingWhite,
            out dateTime);

        Console.WriteLine("\n-----\n");

        Console.WriteLine("Input dateString: " + dateString);

        Console.WriteLine("Can Parse? :" + successful);

        if (successful)
        {
            Console.WriteLine("Parse Results: " +
dateTime.ToString("dd-MM-yyyy HH:mm:ss"));
        }

        Console.Read();
    }
}

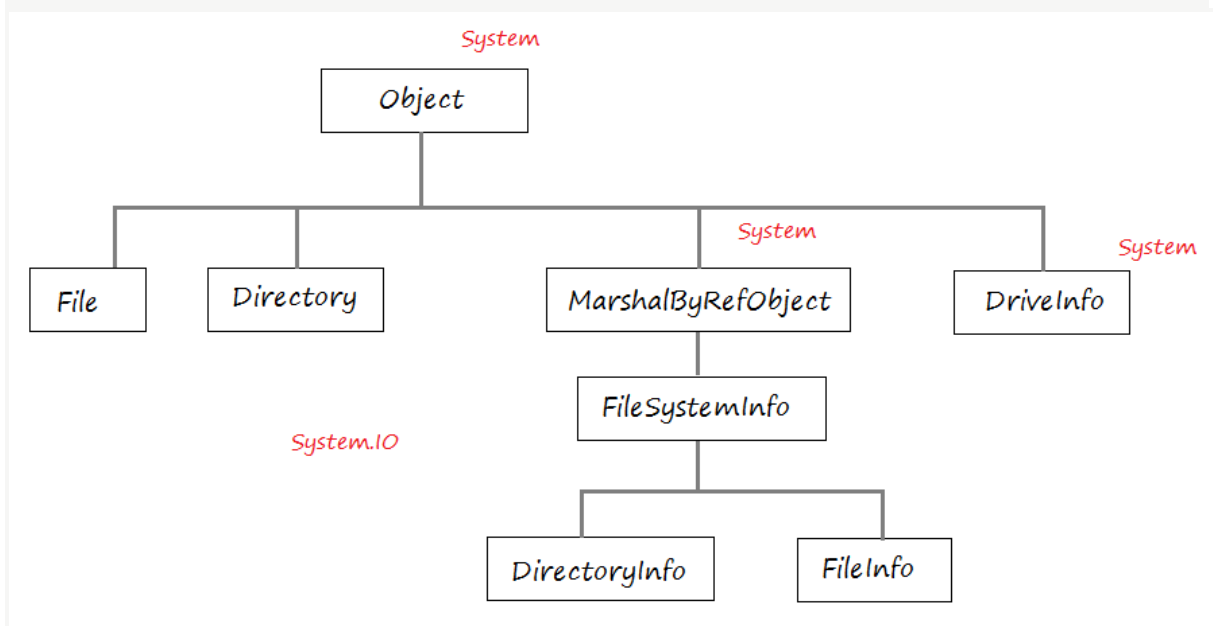
```

```
file:///E:/CSHARP_TUTORIAL/MySolution/DateTimeTutorial/bin/Debug/DateTi...
Input dateString: 20160319 09:57
Parse Results: 19-03-2016 09:57:00

-----
Input dateString: 20110319 11:57
Can Parse? :True
Parse Results: 19-03-2011 11:57:00
```

Thao tác với tập tin và thư mục trong C#

Hệ thống phân cấp các lớp



Class	Mô tả
File	File là một class tiện ích. Nó cung cấp các phương thức tĩnh cho việc tạo, copy, xóa, di chuyển và mở một file, và hỗ trợ tạo đối tượng FileStream.
Directory	Directory là một class tiện ích. Nó cung cấp các phương thức tĩnh để tạo, di chuyển, và liệt kê các thư mục và các thư mục con. Class này không cho

	phép có class con.
FileInfo	FileInfo là một class đại diện cho một file, nó cung cấp các thuộc tính, phương thức cho việc tạo, copy, xóa, di chuyển và mở file. Nó hỗ trợ tạo đối tượng FileStream. Class này không cho phép có class con.
DirectoryInfo	DirectoryInfo là một class đại diện cho một thư mục, nó cung cấp phương thức cho việc tạo, di chuyển, liệt kê các thư mục và các thư mục con. Class này không cho phép có class con.
DriveInfo	DriveInfo là một class, nó cung cấp các phương thức truy cập thông tin ổ cứng.

File

File là một class tiện ích. Nó cung cấp các phương thức tĩnh cho việc tạo, copy, xóa, di chuyển và mở một file, và hỗ trợ tạo đối tượng FileStream.

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

namespace FileDirectoryTutorial
{
    class DeleteFileDemo
    {
        public static void Main(string[] args)
        {
            string filePath = "C:/test/test.txt";

            // Kiểm tra file có tồn tại không.
            if (File.Exists(filePath))
            {
                // Xóa file
                File.Delete(filePath);

                // Kiểm tra lại xem file còn tồn tại không.
                if (!File.Exists(filePath))
                {
                    Console.WriteLine("File deleted...");
                }
            }
            else
            {

```

```

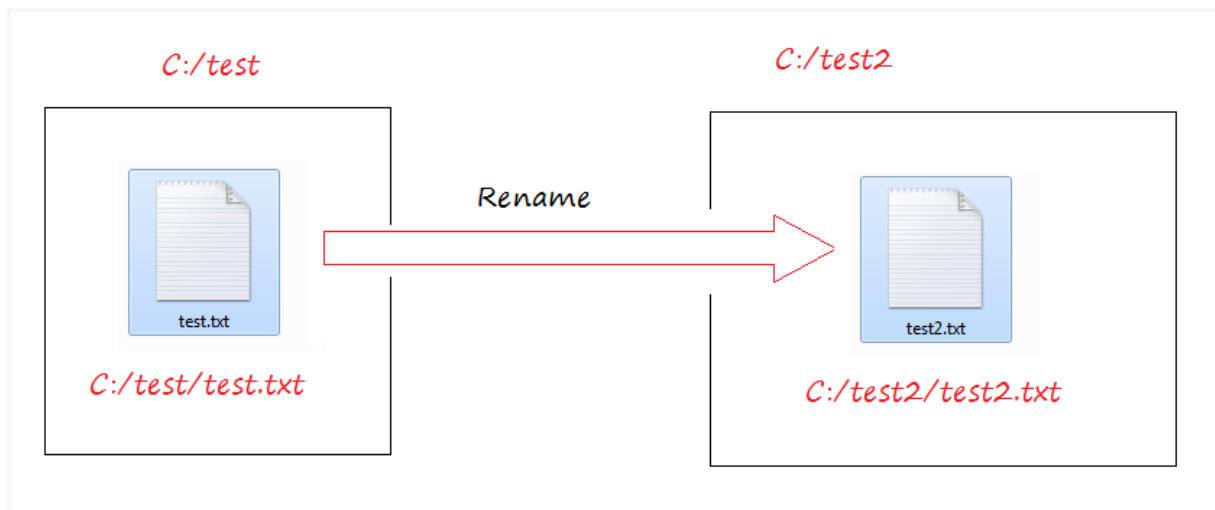
        Console.WriteLine("File test.txt does not yet exist!");
    }

    Console.ReadKey();
}
}
}

```

Note: Đường dẫn trong C# và trong window ngược nhau về dấu xược: window là \ ; c# là /

Đổi tên file là một hành động có thể bao gồm di chuyển file tới một thư mục khác và đổi tên file. Trong trường hợp file bị di chuyển tới một thư mục khác phải đảm bảo rằng thư mục mới đã tồn tại.



```

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

namespace FileDirectoryTutorial
{
    class RenameFileDemo
    {
        public static void Main(string[] args)
        {
            String filePath = "C:/test/test.txt";

            if (File.Exists(filePath))
            {
                Console.WriteLine(filePath + " exist");

                Console.WriteLine("Please enter a new name for this file:");
            }
        }
    }
}

```



```

        // String người dùng nhập vào.
        // Ví dụ: C:/test/test2.txt
        string newFilename = Console.ReadLine();

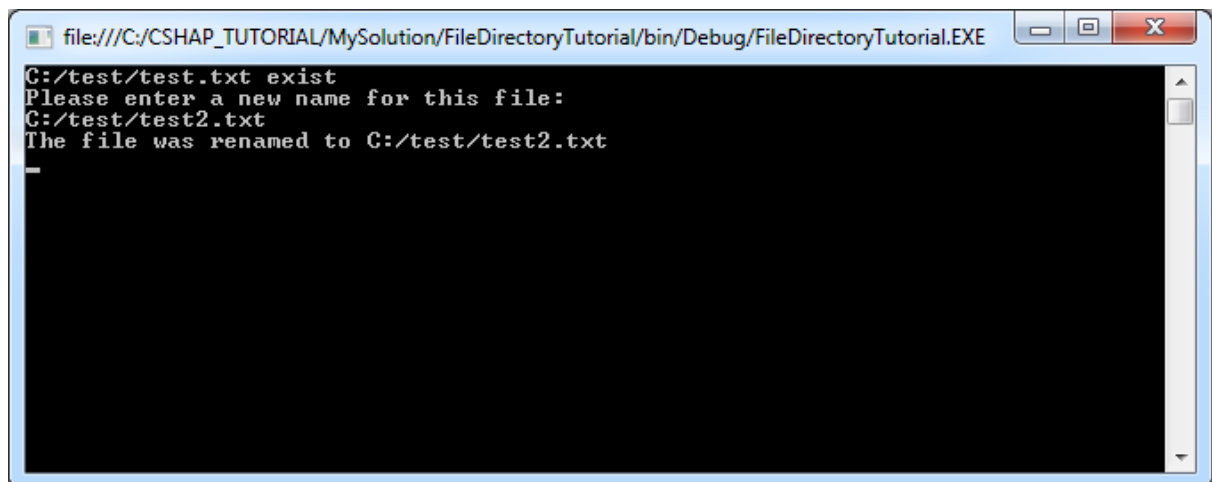
        if (newFilename != String.Empty)
        {
            // Đổi tên file.
            // Bạn có thể chuyển file tới một thư mục khác
            // nhưng phải đảm bảo thư mục đó tồn tại
            // (nếu không ngoại lệ DirectoryNotFoundException sẽ được
ném ra).

            File.Move(filePath, newFilename);

            if (File.Exists(newFilename))
            {
                Console.WriteLine("The file was renamed to " +
newFilename);
            }
        }
        else
        {
            Console.WriteLine("Path " + filePath + " does not exist.");
        }

        Console.ReadLine();
    }
}
}

```



Directory

Directory là một class tiện ích. Nó cung cấp các phương thức tĩnh để tạo, di chuyển, và liệt kê các thư mục và các thư mục con. Class này không cho phép có class con.

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

namespace FileDirectoryTutorial
{
    class DirectoryInformationDemo
    {
        public static void Main(string[] args)
        {
            String dirPath = "C:/test/CSharp";

            // Kiểm tra xem đường dẫn thư mục tồn tại không.
            bool exist = Directory.Exists(dirPath);

            // Nếu không tồn tại, tạo thư mục này.
            if (!exist)
            {
                Console.WriteLine(dirPath + " does not exist.");
                Console.WriteLine("Create directory: " + dirPath);

                // Tạo thư mục.
                Directory.CreateDirectory(dirPath);
            }

            Console.WriteLine("Directory Information " + dirPath);

            // In ra các thông tin thư mục trên.

            // Thời gian tạo.
            Console.WriteLine("Creation time: "+
Directory.GetCreationTime(dirPath));

            // Lăn ghi dữ liệu cuối cùng vào thư mục.
            Console.WriteLine("Last Write Time: " +
Directory.GetLastWriteTime(dirPath));

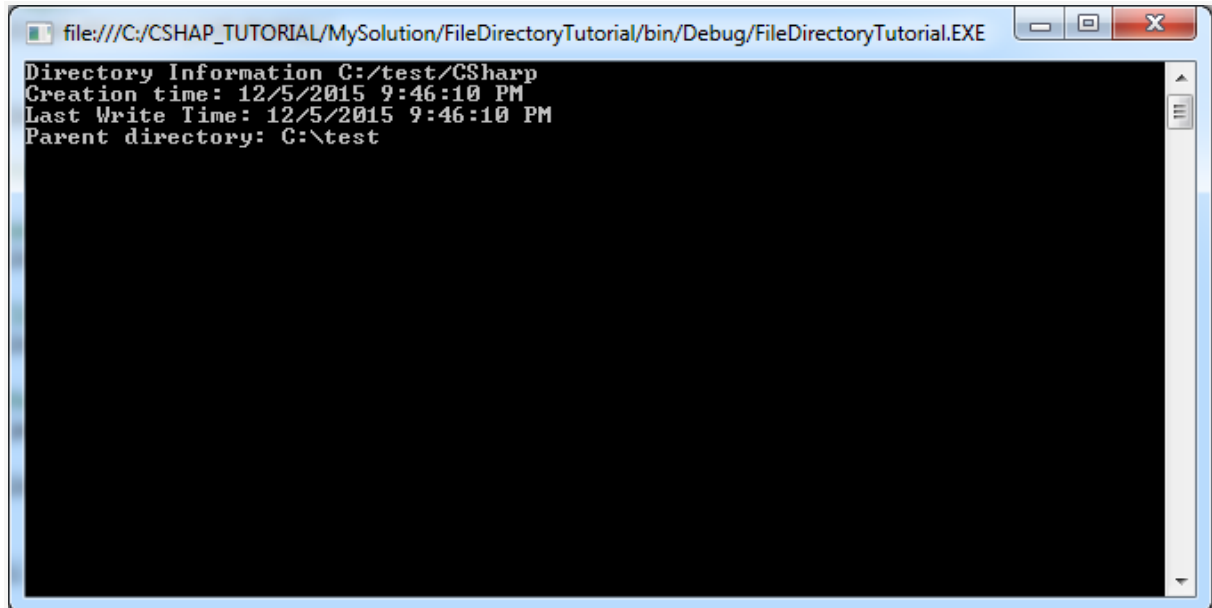
            // Thông tin thư mục cha.
            DirectoryInfo parentInfo = Directory.GetParent(dirPath);

            Console.WriteLine("Parent directory: " + parentInfo.FullName);
        }
    }
}
```

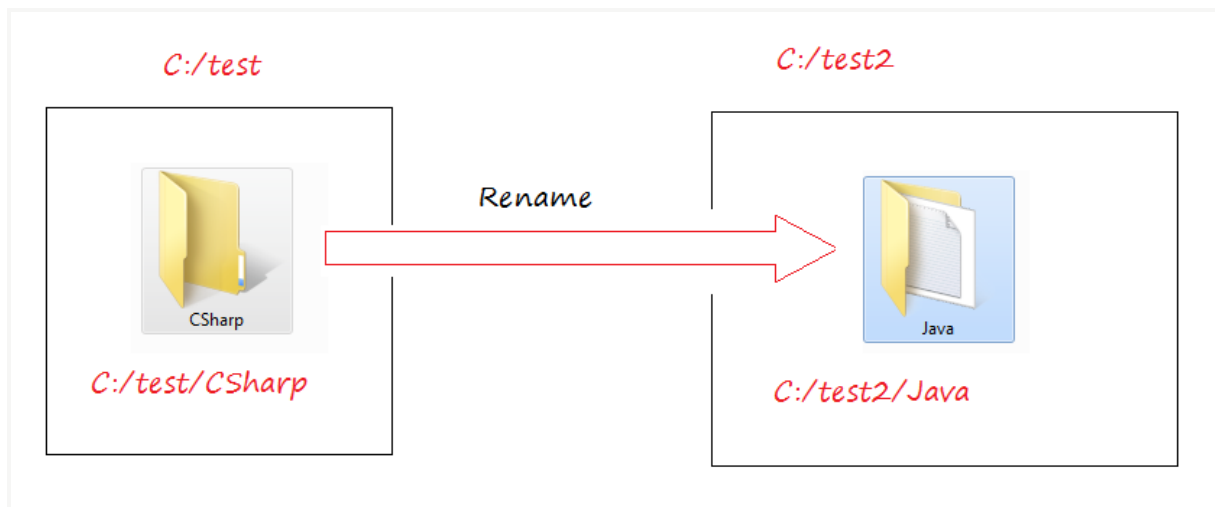
```

        Console.Read();
    }
}

```



Đổi tên thư mục:



Bạn có thể thay đổi tên một thư mục. Nó có thể làm thư mục đó chuyển ra khỏi thư mục cha hiện tại. Nhưng bạn phải đảm bảo rằng thư mục cha mới đã tồn tại.

```

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

namespace FileDirectoryTutorial
{

```

```

class RenameDirectoryDemo
{
    public static void Main(string[] args)
    {
        // Một đường dẫn thư mục.
        String dirPath = "C:/test/CSharp";

        // Nếu đường dẫn này tồn tại.
        if (!Directory.Exists(dirPath))
        {
            Console.WriteLine(dirPath + " does not exist.");
            Console.Read();
            // Kết thúc chương trình.
            return;
        }

        Console.WriteLine(dirPath + " exist");

        Console.WriteLine("Please enter a new name for this directory:");

        // String người dùng nhập vào.
        // Ví dụ: C:/test2/Java
        string newDirname = Console.ReadLine();

        if (newDirname == String.Empty)
        {
            Console.WriteLine("You not enter new directory name. Cancel
rename.");
            Console.Read();
            // Kết thúc chương trình.
            return;
        }

        // Nếu người dùng nhập vào đường dẫn thư mục mới đã tồn tại.
        if (Directory.Exists(newDirname))
        {
            Console.WriteLine("Cannot rename directory. New directory
already exist.");
            Console.Read();
            // Kết thúc chương trình.
            return;
        }

        DirectoryInfo parentInfo = Directory.GetParent(newDirname);

        // Tạo thư mục cha của thư mục mới mà người dùng nhập vào.
        Directory.CreateDirectory(parentInfo.FullName);
    }
}

```

```

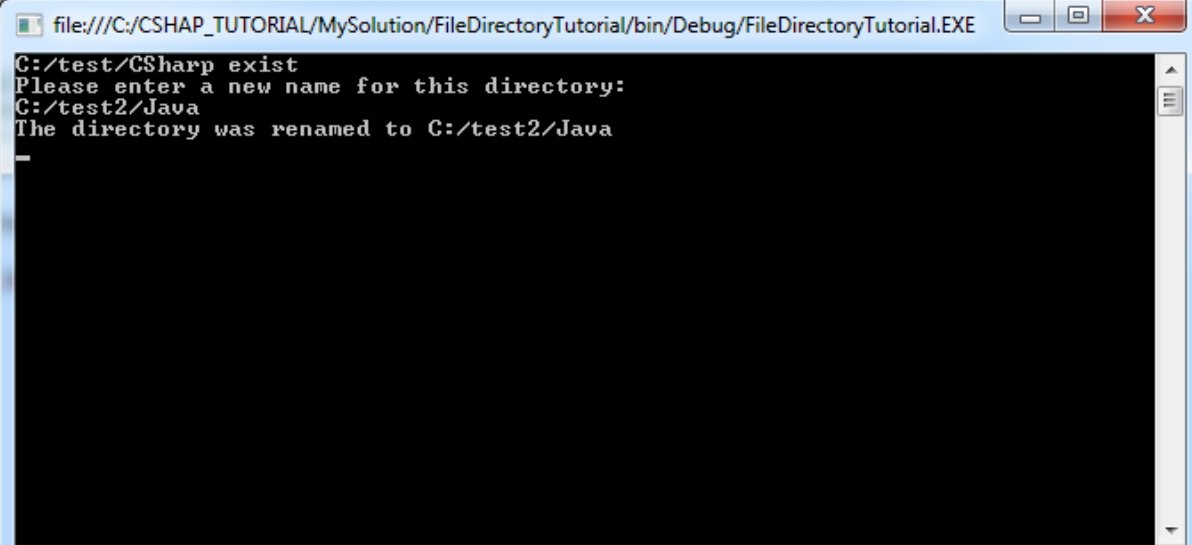
        // Đổi tên thư mục.
        // Bạn có thể chuyển thư mục tới một thư mục khác
        // nhưng phải đảm bảo thư mục đó tồn tại
        // (nếu không ngoại lệ DirectoryNotFoundException sẽ được ném
ra).

        Directory.Move(dirPath, newDirname);

        if (Directory.Exists(newDirname))
        {
            Console.WriteLine("The directory was renamed to " +
newDirname);
        }

        Console.ReadLine();
    }
}
}

```



```

file:///C:/CSHAP_TUTORIAL/MySolution/FileDirectoryTutorial/bin/Debug/FileDirectoryTutorial.EXE
C:/test/CSharp exist
Please enter a new name for this directory:
C:/test2/Java
The directory was renamed to C:/test2/Java

```

FileInfo

FileInfo là một class đại diện cho một file, nó cung cấp các thuộc tính, phương thức cho việc tạo, copy, xóa, di chuyển và mở file. Nó hỗ trợ tạo đối tượng **FileStream**. Class này không cho phép có class con.

Sự khác biệt giữa 2 class **File** và **FileInfo** đó là **File** là một class tiện ích tất các phương thức của nó là tĩnh, còn **FileInfo** đại diện cho một file cụ thể.

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

namespace FileDirectoryTutorial
{
    class FileInfoDemo
    {
        static void Main(string[] args)
        {
            // Một đối tượng đại diện cho một file.
            FileInfo testFile = new FileInfo("C:/test/test.txt");

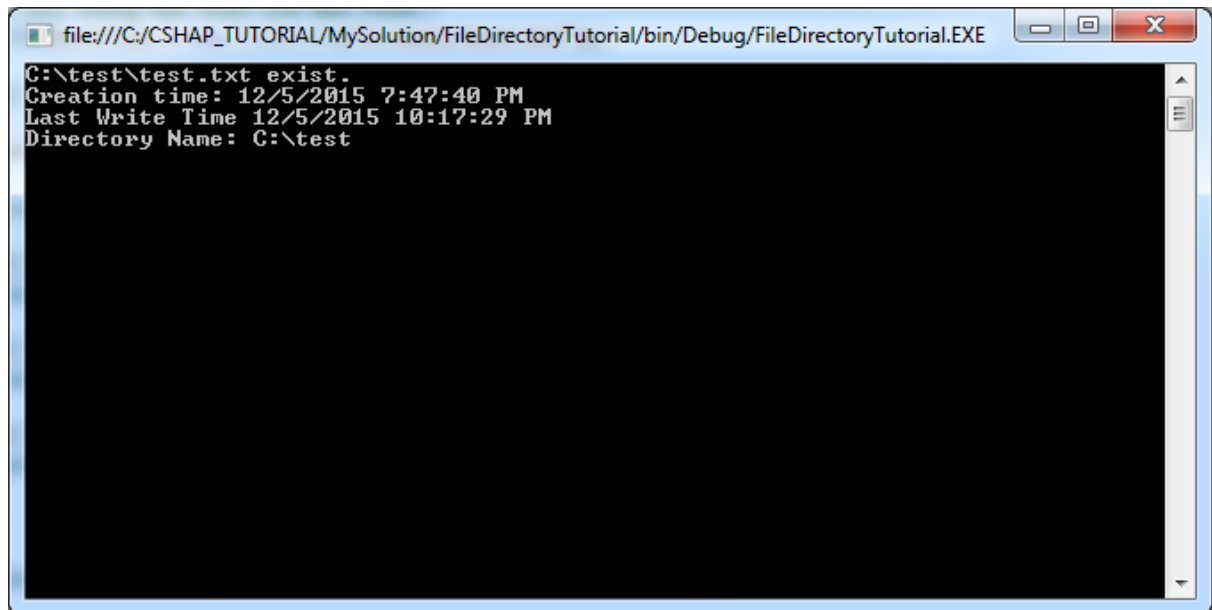
            // Ghi ra các thông tin.

            if (testFile.Exists)
            {
                Console.WriteLine(testFile.FullName + " exist.");

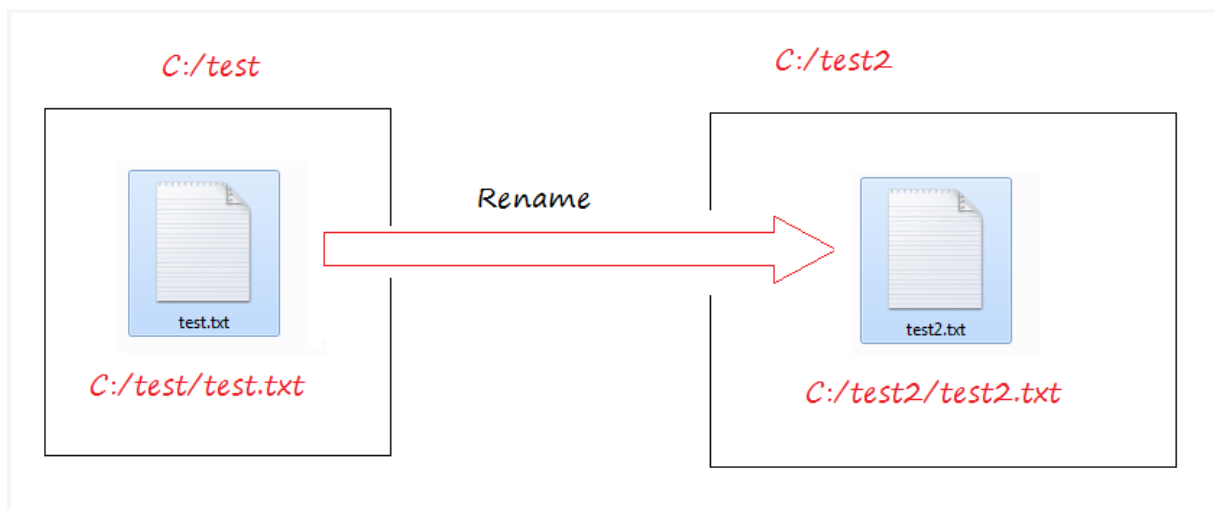
                // Thông tin ngày tạo.
                Console.WriteLine("Creation time: " + testFile.CreationTime);

                // Thông tin ngày sửa cuối.
                Console.WriteLine("Last Write Time " +
testFile.LastWriteTime);

                // Tên thư mục chứa.
                Console.WriteLine("Directory Name: " +
testFile.DirectoryName);
            }
            else
            {
                Console.WriteLine(testFile.FullName + " does not exist.");
            }
            Console.Read();
        }
    }
}
```



Đổi tên file là một hành động có thể bao gồm di chuyển file tới một thư mục khác và đổi tên file. Trong trường hợp file bị di chuyển tới một thư mục khác phải đảm bảo rằng thư mục mới tồn tại.



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

namespace FileDirectoryTutorial
{
    class RenameFileInfoDemo
    {
        public static void Main(string[] args)
        {
            FileInfo fileInfo = new FileInfo("C:/test/test.txt");
```

```

if (!fileInfo.Exists)
{
    Console.WriteLine(fileInfo.FullName + " does not exist.");
    Console.Read();
    // Kết thúc chương trình.
    return;
}

Console.WriteLine(fileInfo.FullName + " exist");

Console.WriteLine("Please enter a new name for this file:");

// String người dùng nhập vào.
// Ví dụ: C:/test/test2.txt
string newFilename = Console.ReadLine();

if (newFilename == String.Empty)
{
    Console.WriteLine("You not enter new file name. Cancel
rename");

    Console.Read();
    // Kết thúc chương trình.
    return;
}

FileInfo newFileInfo = new FileInfo(newFilename);

// Nếu newFileInfo tồn tại (Không thể đổi tên).
if (newFileInfo.Exists)
{
    Console.WriteLine("Can not rename file to " +
newFileInfo.FullName + ". File already exist.");

    Console.Read();
    // Kết thúc chương trình.
    return;
}

// Đảm bảo rằng thư mục chuyển tới tồn tại.
newFileInfo.Directory.Create();

// Đổi tên file.
fileInfo.MoveTo(newFileInfo.FullName);

```



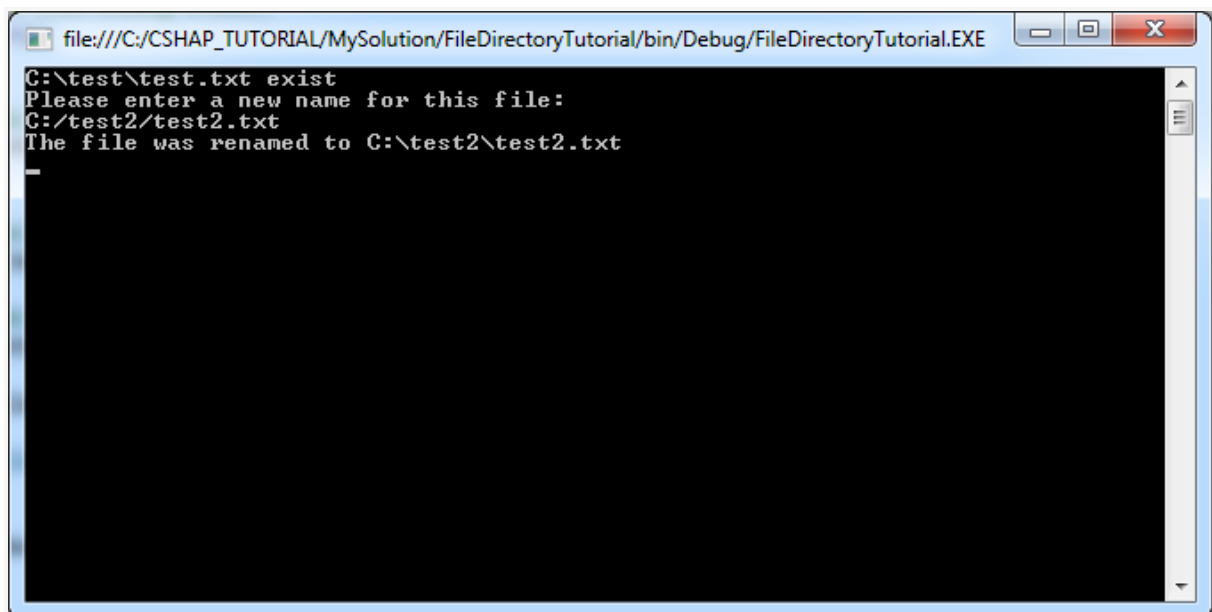
```

        // Refresh trạng thái.
        newFileInfo.Refresh();

        if (newFileInfo.Exists)
        {
            Console.WriteLine("The file was renamed to " +
newFileInfo.FullName);
        }

        Console.ReadLine();
    }
}
}

```



DirectoryInfo

DirectoryInfo là một class đại diện cho một thư mục, nó cung cấp phương thức cho việc tạo, di chuyển, liệt kê các thư mục và các thư mục con. Class này không cho phép có class con.

Sự khác biệt giữa 2 class Directory và DirectoryInfo đó là Directory là một class tiện ích, tất cả các phương thức của nó là tĩnh, còn DirectoryInfo đại diện cho một thư mục cụ thể.

```

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

```

```

namespace FileDirectoryTutorial

```

```

{
    class DirectoryInfoDemo
    {
        static void Main(string[] args)
        {
            // Một đối tượng đại diện cho một thư mục.
            DirectoryInfo dirInfo = new
DirectoryInfo("C:/Windows/System32/drivers");

            // Ghi ra các thông tin.
            // Thông tin ngày tạo.
            Console.WriteLine("Creation time: " + dirInfo.CreationTime);

            // Thông tin ngày sửa cuối.
            Console.WriteLine("Last Write Time " + dirInfo.LastWriteTime);

            // Tên thư mục chứa.
            Console.WriteLine("Directory Name: " + dirInfo.FullName);

            // Mảng các thư mục con.
            DirectoryInfo[] childDirs = dirInfo.GetDirectories();
            // Mảng các file con.
            FileInfo[] childFiles = dirInfo.GetFiles();

            foreach(DirectoryInfo childDir in childDirs ){
                Console.WriteLine(" - Directory: " + childDir.FullName);
            }

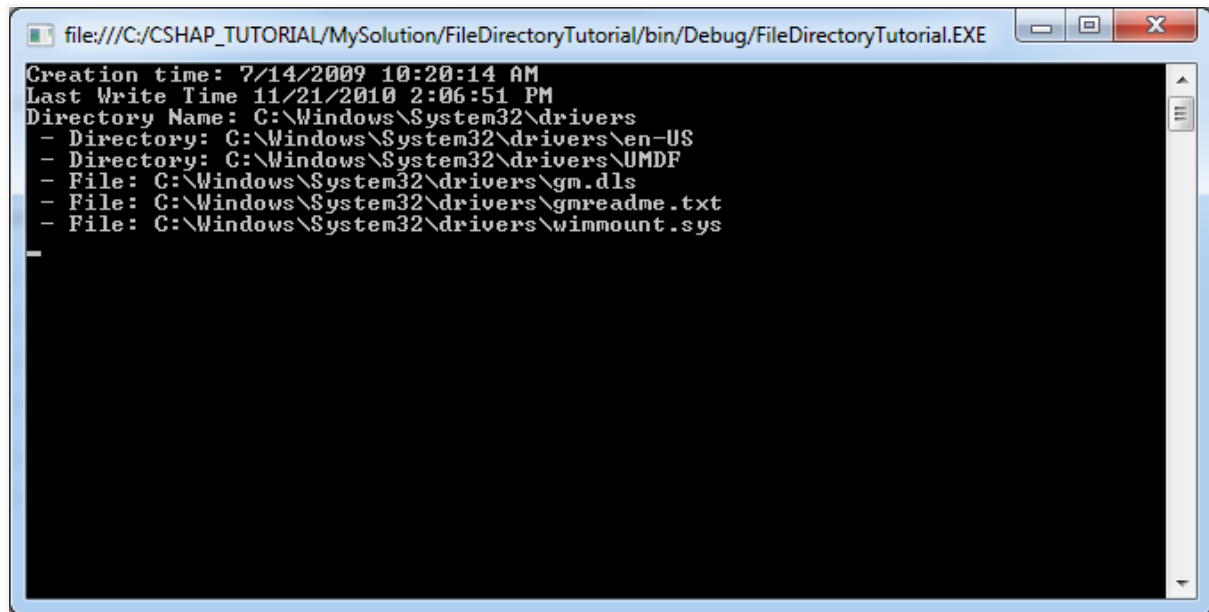
            foreach (FileInfo childFile in childFiles)
            {
                Console.WriteLine(" - File: " + childFile.FullName);
            }

            Console.Read();
        }
    }
}

```

DriveInfo

DirveInfo là một class, nó cung cấp các phương thức truy cập thông tin ổ cứng.



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

namespace FileDirectoryTutorial
{
    class DriveInfoDemo
    {
        static void Main(string[] args)
        {
            DriveInfo[] drives = DriveInfo.GetDrives();

            foreach (DriveInfo drive in drives)
            {
                Console.WriteLine(" ===== ");
                // Tên ổ đĩa
                Console.WriteLine("Drive {0}", drive.Name);
                // Loại ổ đĩa
                Console.WriteLine(" Drive type: {0}", drive.DriveType);

                // Nếu ổ đĩa sẵn sàng.
                if (drive.IsReady)
                {
                    Console.WriteLine(" Volume label: {0}",
drive.VolumeLabel);
                    Console.WriteLine(" File system: {0}",
drive.DriveFormat);
                    Console.WriteLine(
                        " Available space to current user:{0, 15} bytes",
```

```

        drive.AvailableFreeSpace);

        Console.WriteLine(
            "    Total available space:          {0, 15} bytes",
            drive.TotalFreeSpace);

        Console.WriteLine(
            "    Total size of drive:                {0, 15} bytes ",
            drive.TotalSize);
    }
}

Console.Read();
}
}
}

```

The screenshot shows a Windows command prompt window titled "file:///C:/CSHAP_TUTORIAL/MySolution/FileDirectoryTutorial/bin/Debug/FileDirectoryTutorial.EXE". The window displays the output of a program that lists disk information for drives A, C, and D. The output is as follows:

```

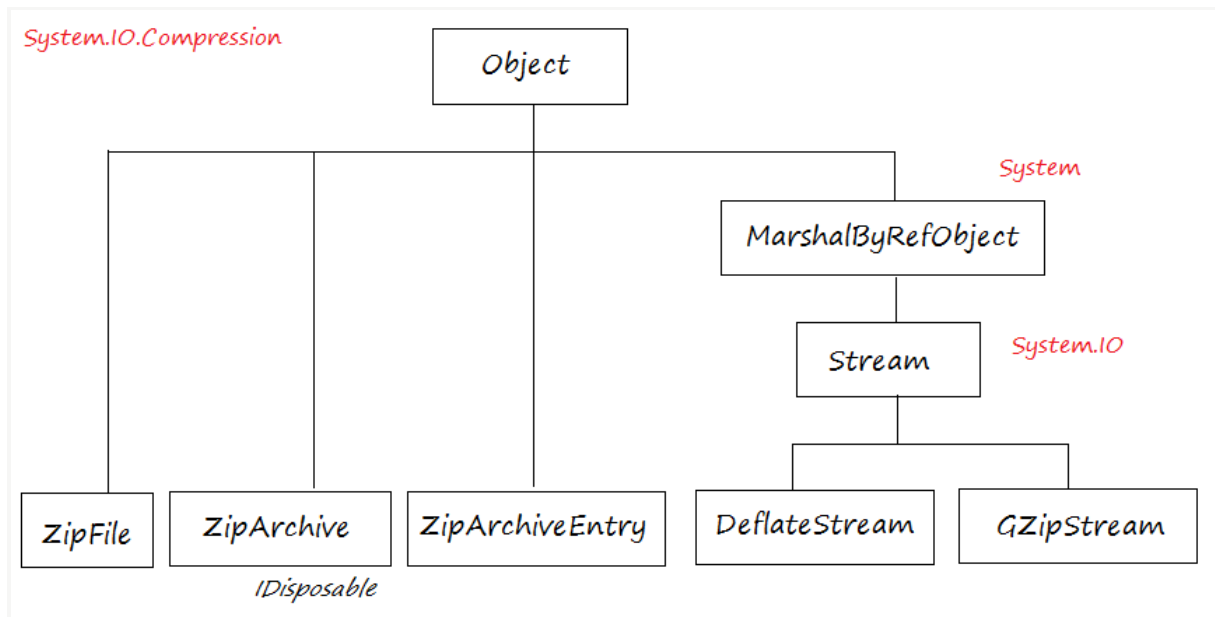
=====
Drive A:\
  Drive type: Removable
=====
Drive C:\
  Drive type: Fixed
  Volume label:
  File system: NTFS
  Available space to current user:    5082611712 bytes
  Total available space:              5082611712 bytes
  Total size of drive:                32209104896 bytes
=====
Drive D:\
  Drive type: CDROM
  Volume label: Win8_1_x64
  File system: UDF
  Available space to current user:    0 bytes
  Total available space:              0 bytes
  Total size of drive:                5321455616 bytes

```

Nén và giải nén trong C#

Sơ đồ phân cấp các lớp

Dưới đây là danh mục các class sử dụng cho mục đích nén và giải nén file. Chúng nằm trong namespace `System.IO.Compression`.



Class	Mô tả
ZipFile	Cung cấp các phương thức tĩnh cho việc tạo, trích dữ liệu và mở file dữ liệu zip.
ZipArchive	Đại diện cho tập các file được nén trong một file có định dạng ZIP.
ZipArchive Entry	Đại diện cho một tập tin nằm trong file định dạng ZIP.
DeflateStream	Cung cấp các phương thức và thuộc tính cho các luồng (stream) nén và giải nén bằng cách sử dụng thuật toán Deflate.
GZipStream	Cung cấp các phương thức và thuộc tính được sử dụng để nén và giải nén các luồng (stream).

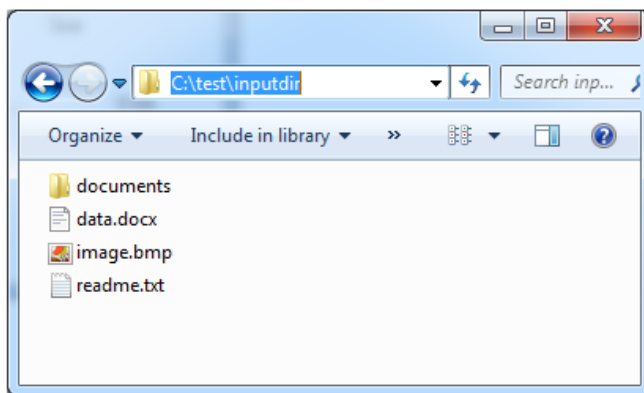
Chú ý rằng các class này được đưa vào C# từ phiên bản 4.5, vì vậy project của bạn phải sử dụng .NET phiên bản 4.5 hoặc mới hơn.

ZipFile

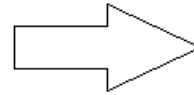
Class **ZipFile** là một class tiện ích, nó có nhiều phương thức tĩnh giúp bạn mở file zip, trích lấy dữ liệu, hoặc các tình huống hay được sử dụng như nén một thư mục thành một file zip, giải nén file zip ra một thư mục, ..

Ví dụ đơn giản dưới đây sử dụng các phương thức tiện ích của class **ZipFile** nén một thư mục thành một file zip và sau đó giải nén file này sang một thư mục khác.

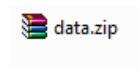
C:/test/inputdir



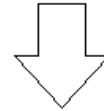
Nén



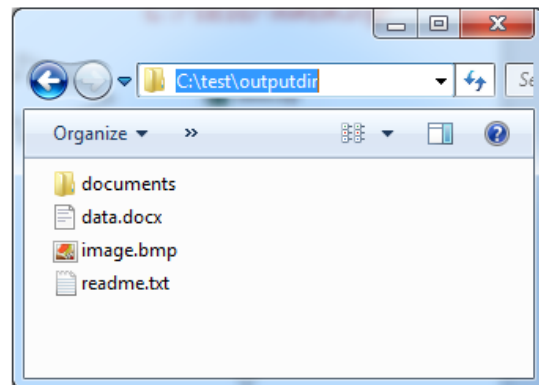
C:/test/data.zip



Giải nén



C:/test/outputdir



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

namespace CompressionTutorial
{
    class ZipDirectoryDemo
    {
        public static void Main(string[] args)
        {

            // Thư mục sẽ nén
            string inputDir = "C:/test/inputdir";

            // File đầu ra khi nén thư mục trên.
            string zipPath = "C:/test/data.zip";

            // Giải nén file zip ra thư mục.
            string extractPath = "C:/test/outputdir";

            // Tạo ra file zip bằng cách nén cả thư mục.
            ZipFile.CreateFromDirectory(inputDir, zipPath);
        }
    }
}
```

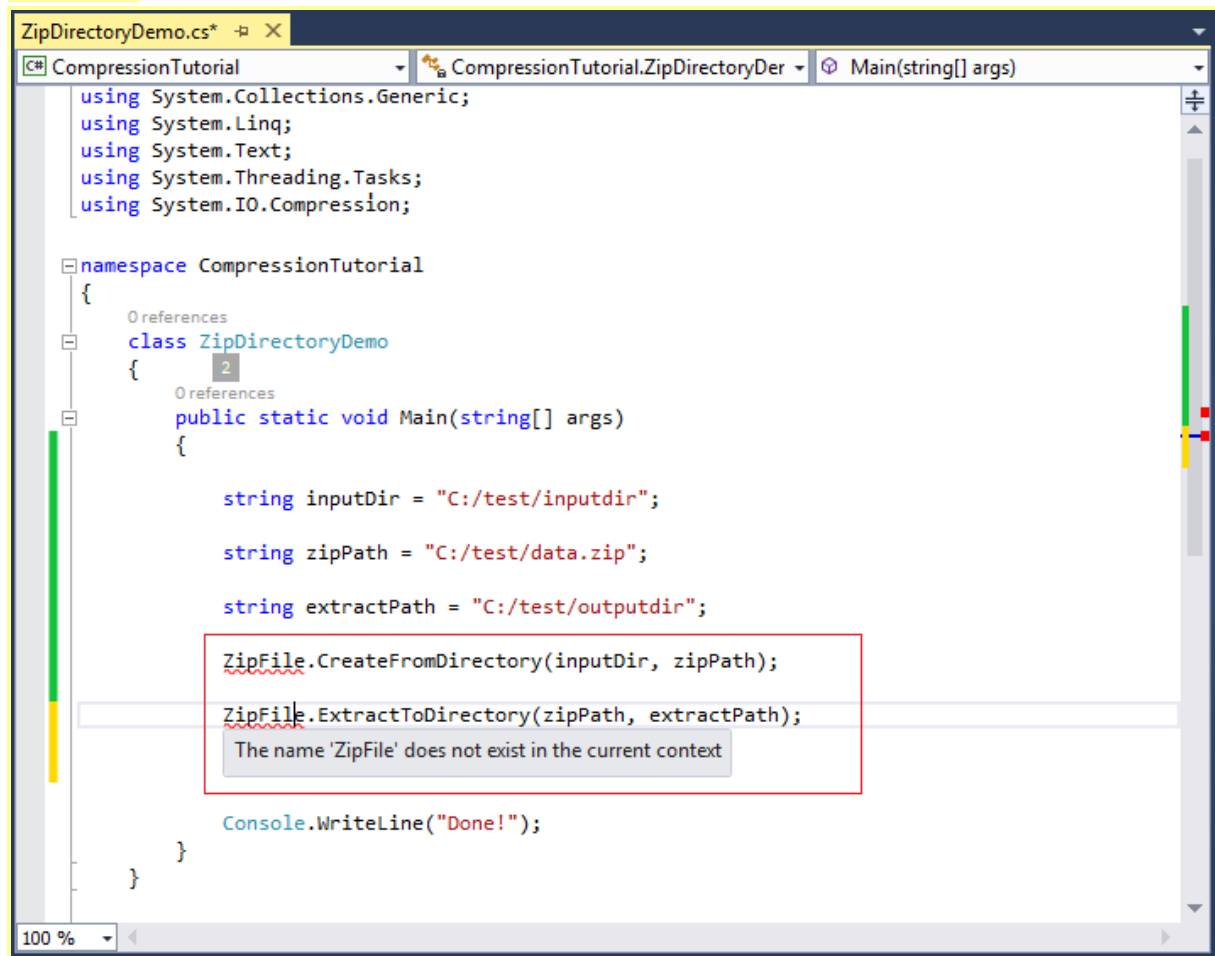
```

        // Giải nén file zip ra một thư mục.
        ZipFile.ExtractToDirectory(zipPath, extractPath);

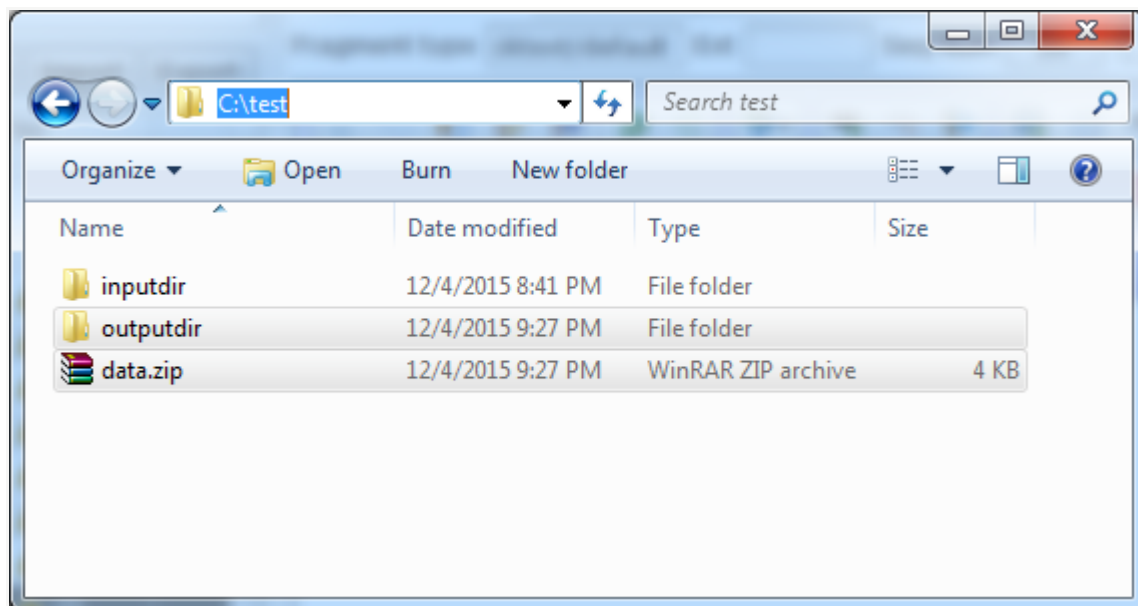
        Console.WriteLine("Done!");
    }
}

```

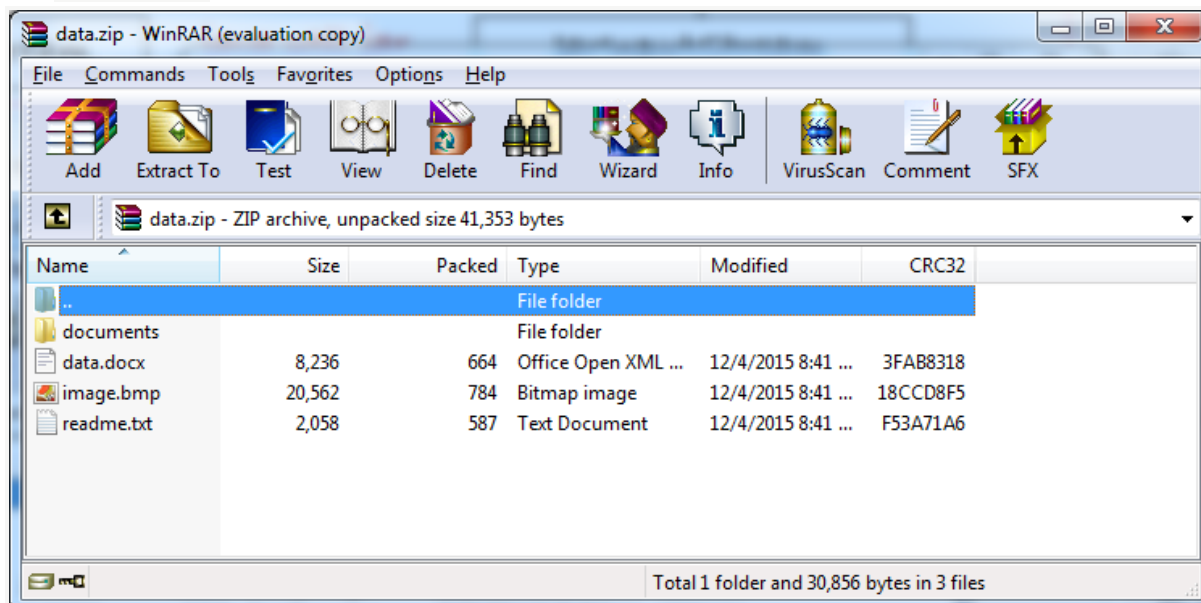
Nếu bạn nhận được một thông báo lỗi: **"The name 'ZipFile' does not exist in the current context"** (Mặc dù đã khai báo `using System.IO.Compression`) điều đó có nghĩa là project của bạn sử dụng .NET cũ hơn 4.5 hoặc chương trình không tìm thấy thư viện DLL.



Chạy ví dụ và nhận được kết quả

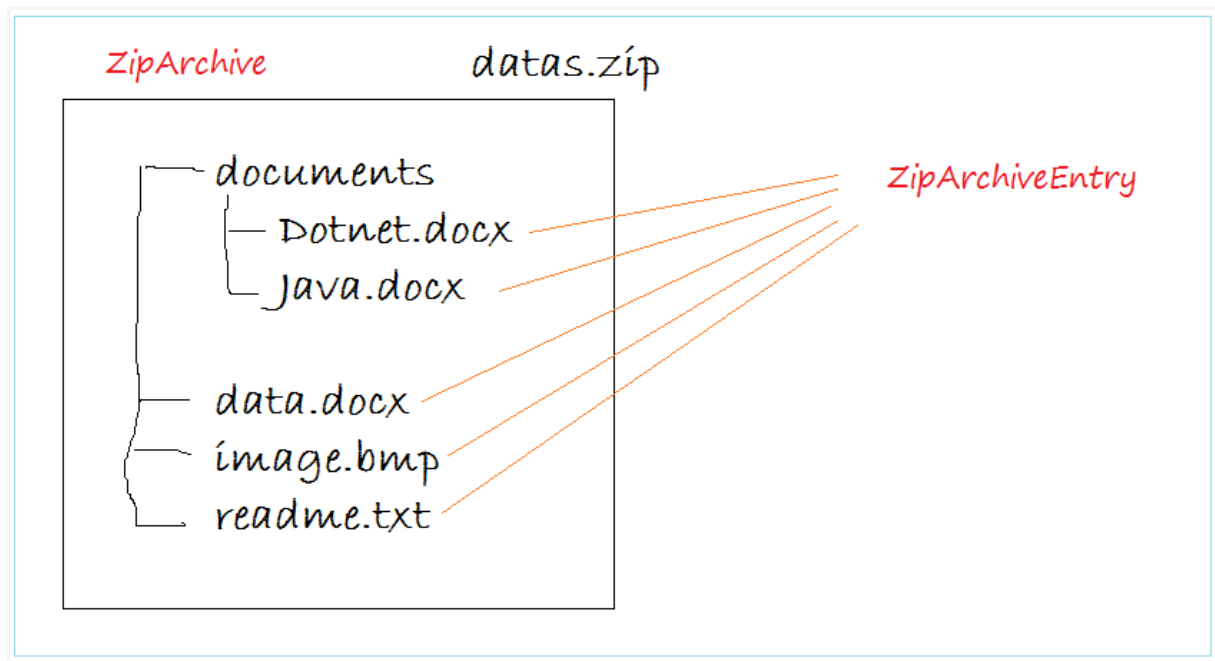


- **data.zip**



ZipArchive

ZipArchive đại diện cho một bó các file đã nén trong một file định dạng ZIP. Bạn có thể tạo được đối tượng **ZipArchive** thông qua phương thức **OpenRead** của class **ZipFile**. Thông qua **ZipArchive** bạn có thể đọc các file con đã được nén trong file zip.



Ví dụ dưới đây liệt kê ra các **ZipArchiveEntry** có trong file zip.

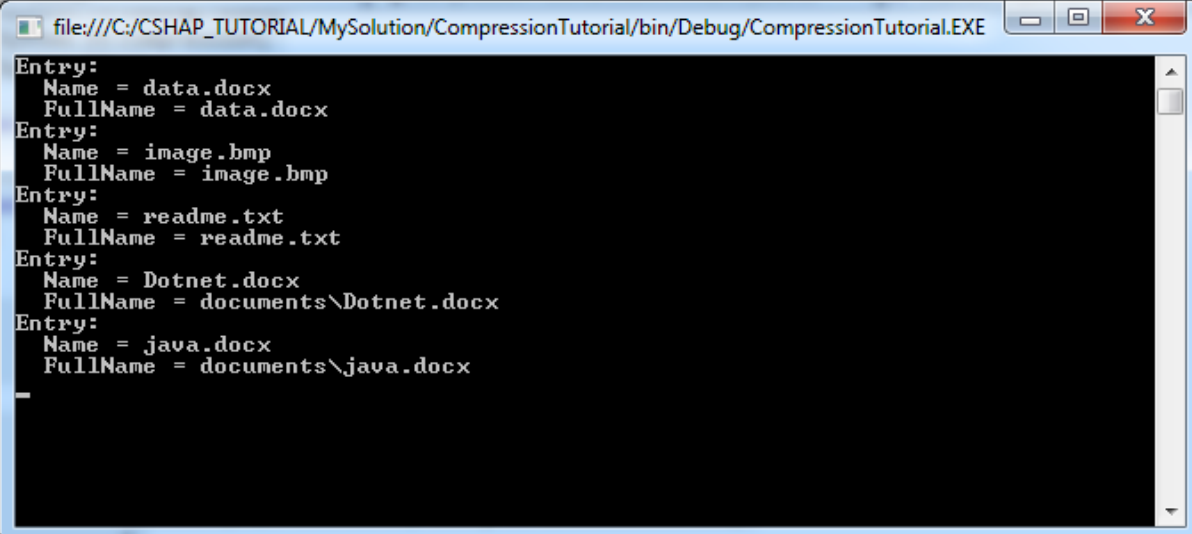
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.IO.Compression;
using System.IO;

namespace CompressionTutorial
{
    class ListArchiveEntryDemo
    {
        public static void Main(string[] args)
        {
            string zipPath = "c:/test/data.zip";

            using (ZipArchive archive = ZipFile.OpenRead(zipPath))
            {
                // Duyệt danh sách các ZipArchiveEntry.
                foreach (ZipArchiveEntry entry in archive.Entries)
                {
                    Console.WriteLine("Entry:");
                    Console.WriteLine("  Name = " + entry.Name);
                    Console.WriteLine("  FullName = " + entry.FullName);
                }
            }

            Console.Read();
        }
    }
}
```

```
}  
  
}
```



```
file:///C:/CSHAP_TUTORIAL/MySolution/CompressionTutorial/bin/Debug/CompressionTutorial.EXE  
Entry:  
  Name = data.docx  
  FullName = data.docx  
Entry:  
  Name = image.bmp  
  FullName = image.bmp  
Entry:  
  Name = readme.txt  
  FullName = readme.txt  
Entry:  
  Name = Dotnet.docx  
  FullName = documents\Dotnet.docx  
Entry:  
  Name = java.docx  
  FullName = documents\java.docx  
-
```

Trích các file dữ liệu trong file zip:

```
using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;  
using System.Threading.Tasks;  
using System.IO.Compression;  
using System.IO;  
  
namespace CompressionTutorial  
{  
    class ExtractDemo {  
        static void Main(string[] args)  
        {  
            string zipPath = "c:/test/data.zip";  
            // Thư mục giải nén ra.  
            string extractPath = "c:/test/extract";  
  
            // if it doesn't exist, create  
            // Nếu thư mục không tồn tại, tạo nó.  
            if (!Directory.Exists(extractPath))  
            {  
                System.IO.Directory.CreateDirectory(extractPath);  
            }  
  
            using (ZipArchive archive = ZipFile.OpenRead(zipPath))  
            {  
                foreach (ZipArchiveEntry entry in archive.Entries)  
                {  
                    Console.WriteLine("Found: " + entry.FullName);  
                }  
            }  
        }  
    }  
}
```

```

        // Tìm kiếm các Entry có đuôi .docx
        if (entry.FullName.EndsWith(".docx",
StringComparison.OrdinalIgnoreCase))
        {
            // Ví dụ: documents/Dotnet.docx
            Console.WriteLine(" - Extract entry: " +
entry.FullName);

            // Ví dụ: C:/test/extract/documents/Dotnet.docx
            string entryOutputPath = Path.Combine(extractPath,
entry.FullName);

            Console.WriteLine(" - Entry Ouput Path: " +
entryOutputPath);

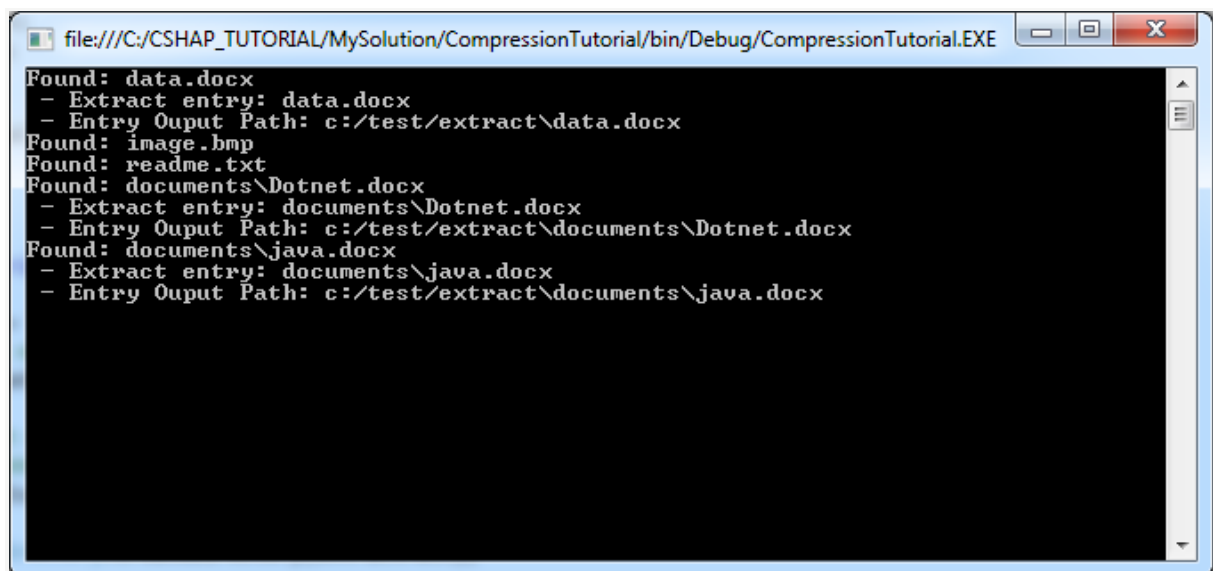
            FileInfo fileInfo = new FileInfo(entryOutputPath);

            // Đảm bảo rằng thư mục chứa file tồn tại.
            // Ví dụ: C:/test/extract/documents
            fileInfo.Directory.Create();

            // Ghi đè file cũ nếu nó đã tồn tại.
            entry.ExtractToFile(entryOutputPath, true);
        }
    }
}

Console.ReadLine();
}
}
}

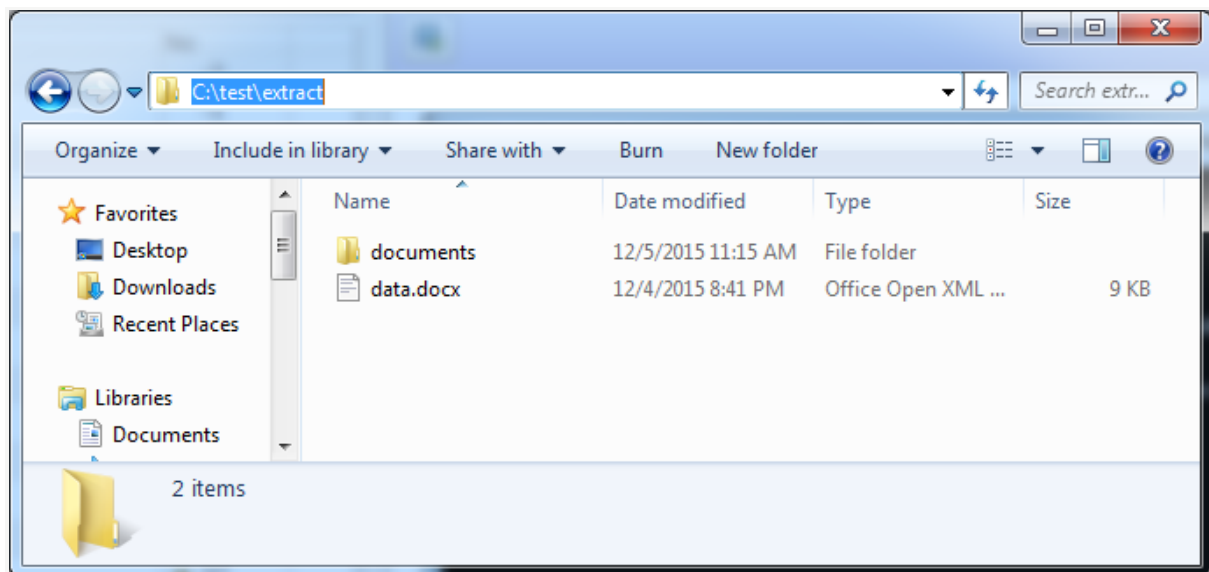
```



```

file:///C:/CSHAP_TUTORIAL/MySolution/CompressionTutorial/bin/Debug/CompressionTutorial.EXE
Found: data.docx
- Extract entry: data.docx
- Entry Ouput Path: c:/test/extract\data.docx
Found: image.bmp
Found: readme.txt
Found: documents\Dotnet.docx
- Extract entry: documents\Dotnet.docx
- Entry Ouput Path: c:/test/extract\documents\Dotnet.docx
Found: documents\java.docx
- Extract entry: documents\java.docx
- Entry Ouput Path: c:/test/extract\documents\java.docx

```



Bạn cũng có thể trên thêm các file vào trong một file zip có sẵn.

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

namespace CompressionTutorial
{
    class AddEntryDemo
    {
        static void Main(string[] args)
        {
            string zipPath = "C:/test/data.zip";

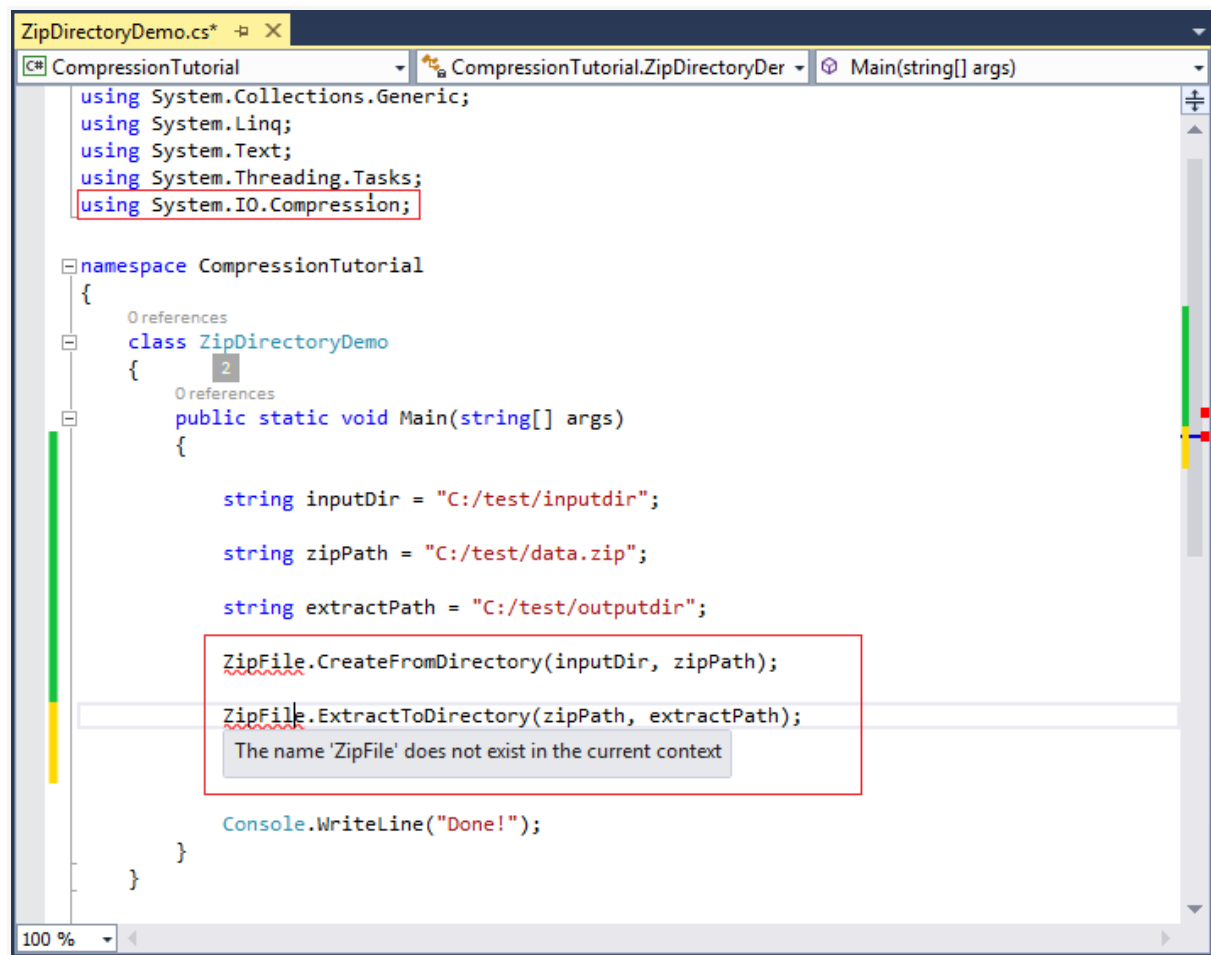
            // Mở một luồng đọc file zip.
            using (FileStream zipStream = new FileStream(zipPath,
                FileMode.Open))
            {
                // Tạo đối tượng ZipArchive.
                using (ZipArchive archive = new ZipArchive(zipStream,
                    ZipArchiveMode.Update))
                {
                    // Thêm một entry vào ZipArchive.
                    ZipArchiveEntry readmeEntry =
                        archive.CreateEntry("note/Note.txt");

                    // Tạo một luồng ghi nội dung vào entry.
                    using (StreamWriter writer = new
                        StreamWriter(readmeEntry.Open()))
                    {
```

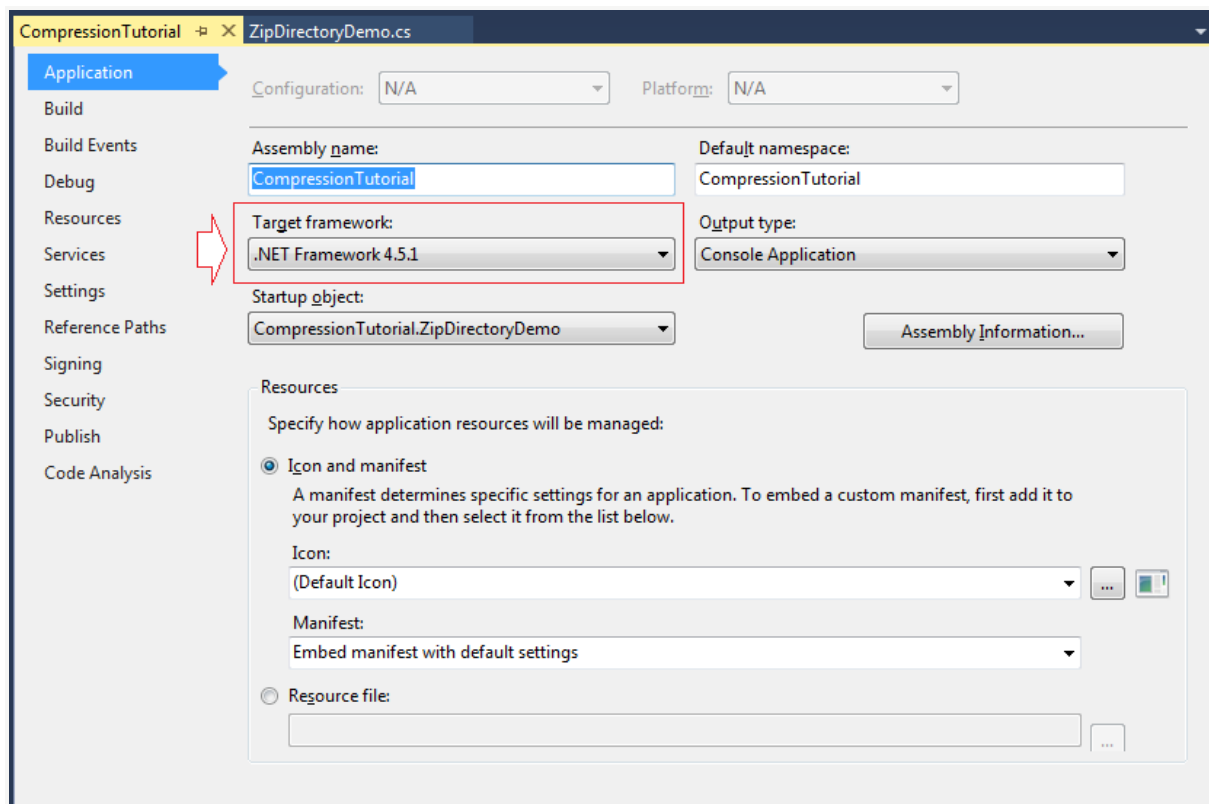
}



Khi bạn nhận được lỗi: **"The name 'ZipFile' does not exist in the current context"** (Mặc dù đã khai báo **using System.IO.Compression**) điều đó có nghĩa là bạn đã sử dụng .NET cũ hơn phiên bản 4.5 hoặc chương trình không tìm được thư viện DLL cần thiết.



Nhấn phải chuột vào Project chọn Properties, đảm bảo rằng project của bạn đã sử dụng .NET Framework 4.5 trở lên.

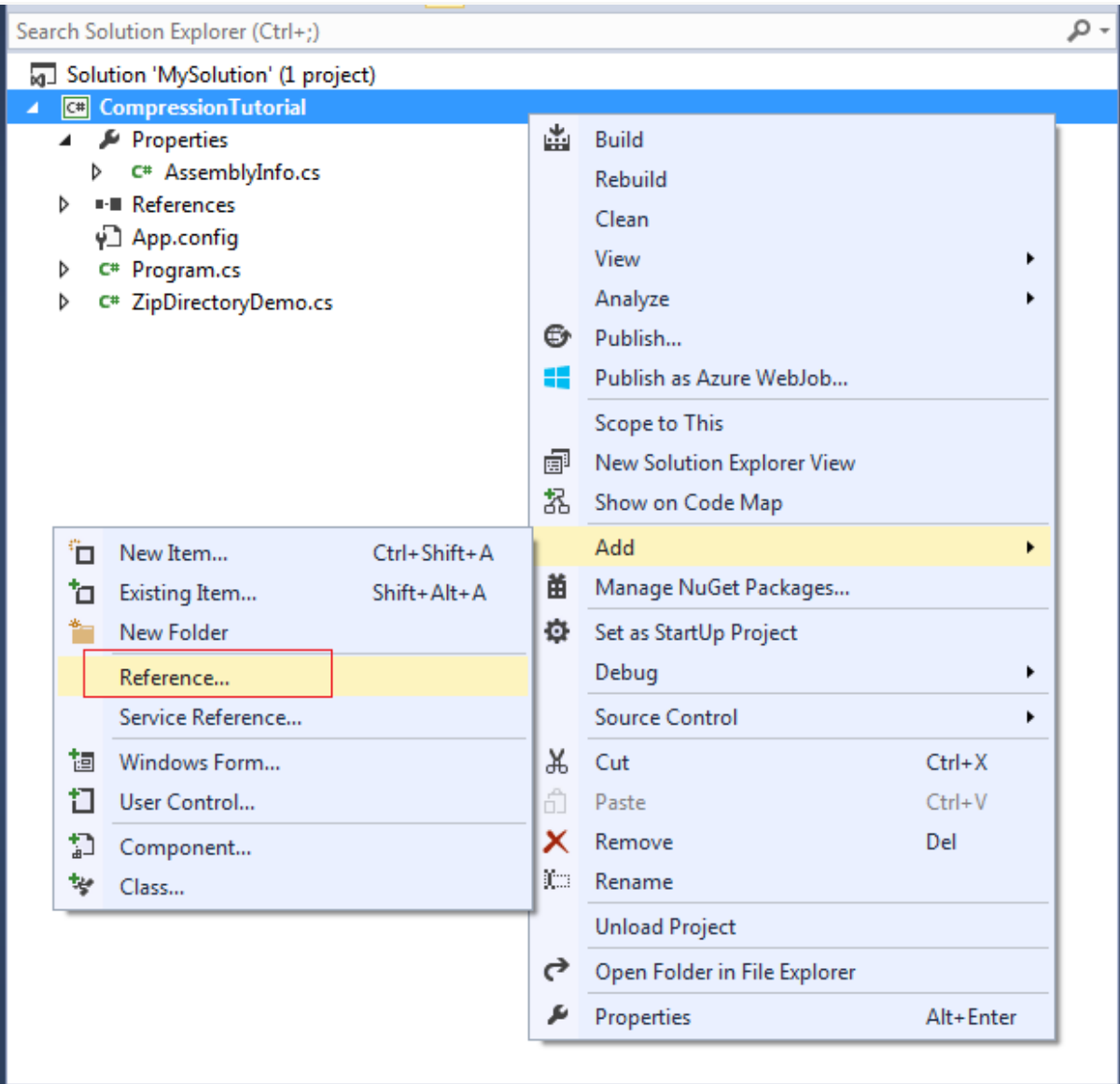


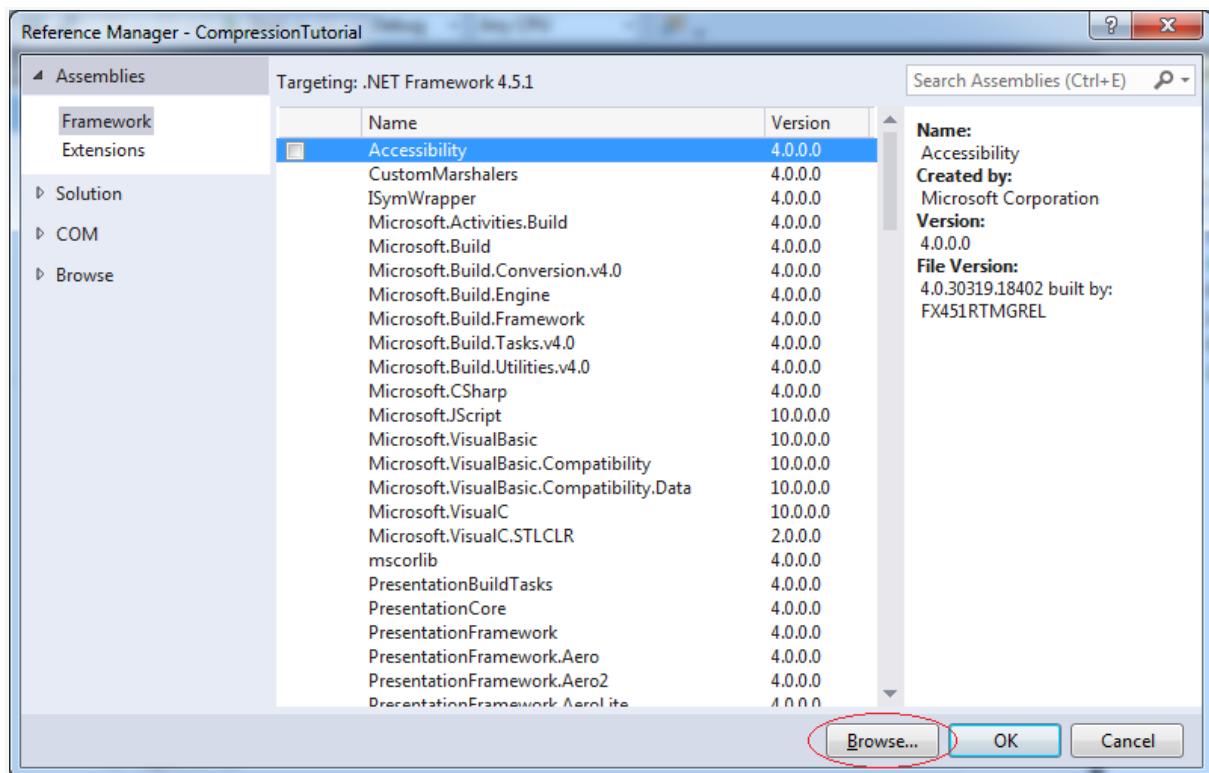
Chạy lại class của bạn xem trình biên dịch còn thông báo lỗi đó nữa hay không. Trong trường hợp vẫn thông báo lỗi bạn cần chỉ định rõ vị trí thư viện DLL.

- C:\Program Files (x86)\Reference Assemblies\Microsoft\Framework\.NETFramework\v4.5\System.IO.Compression.FileSystem.dll
- C:\Program Files (x86)\Reference Assemblies\Microsoft\Framework\.NETFramework\v4.5.1\System.IO.Compression.FileSystem.dll

Nhấn phải chuột vào Project chọn:

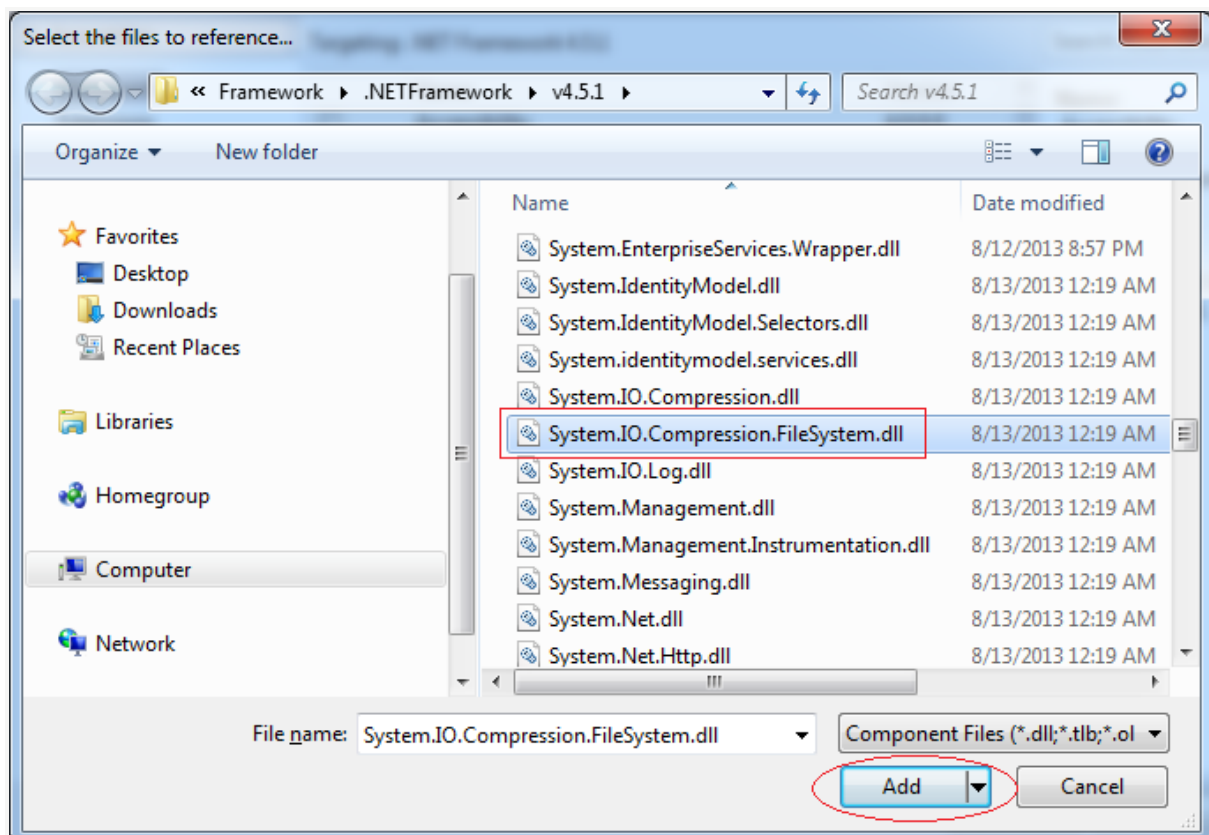
- **Add/Reference..**

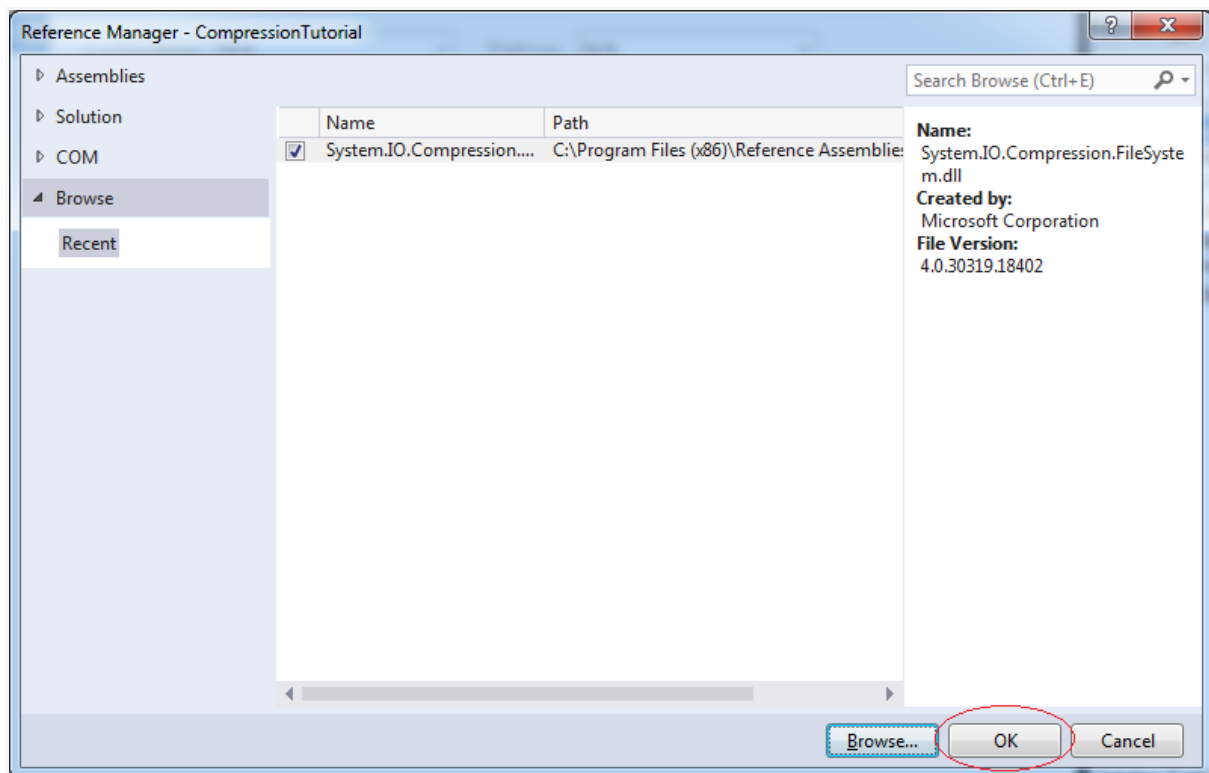




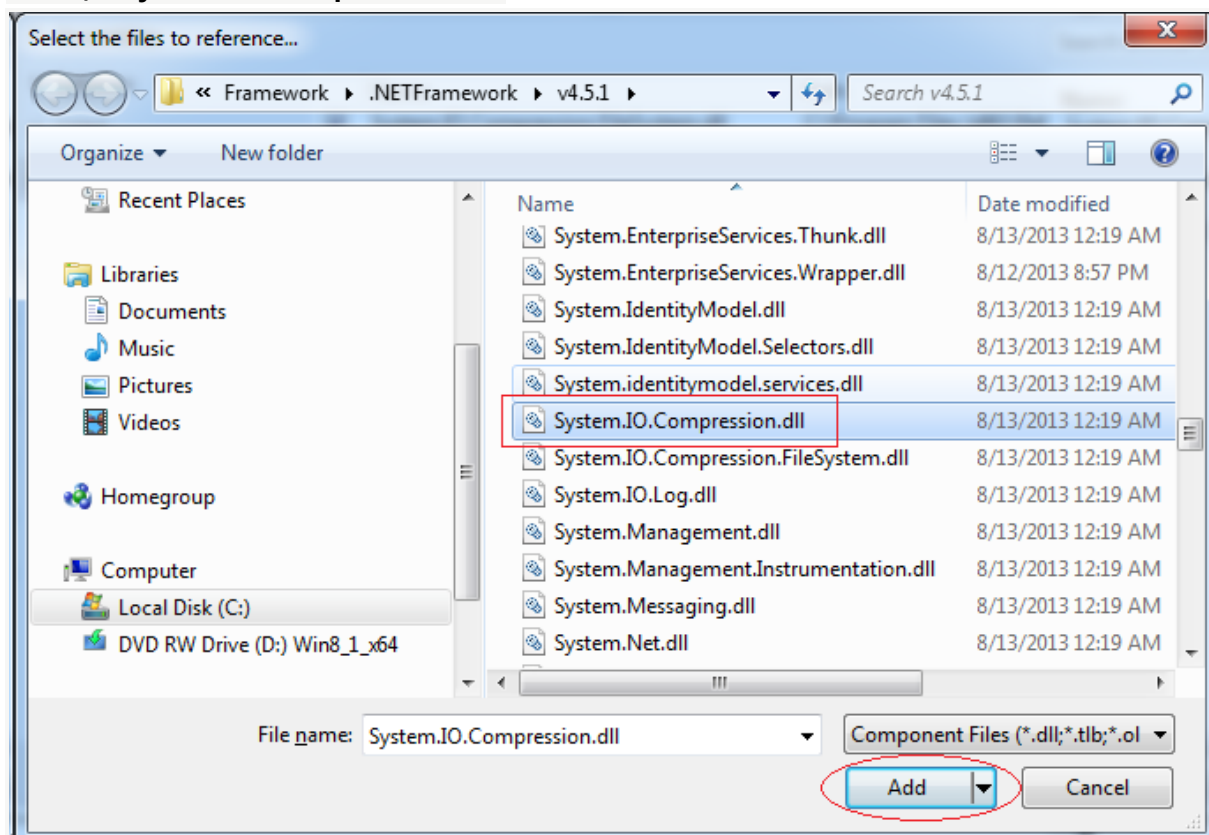
Chọn file:

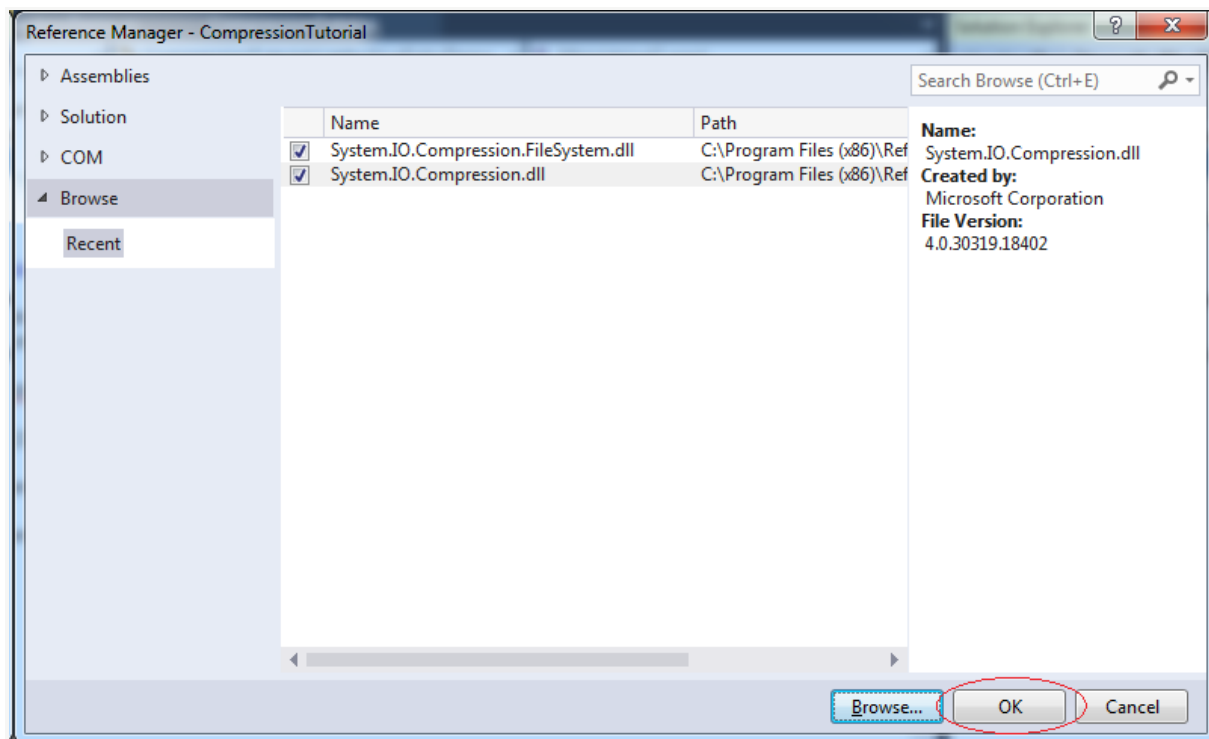
- **System.IO.Compression.FileSystem.dll**



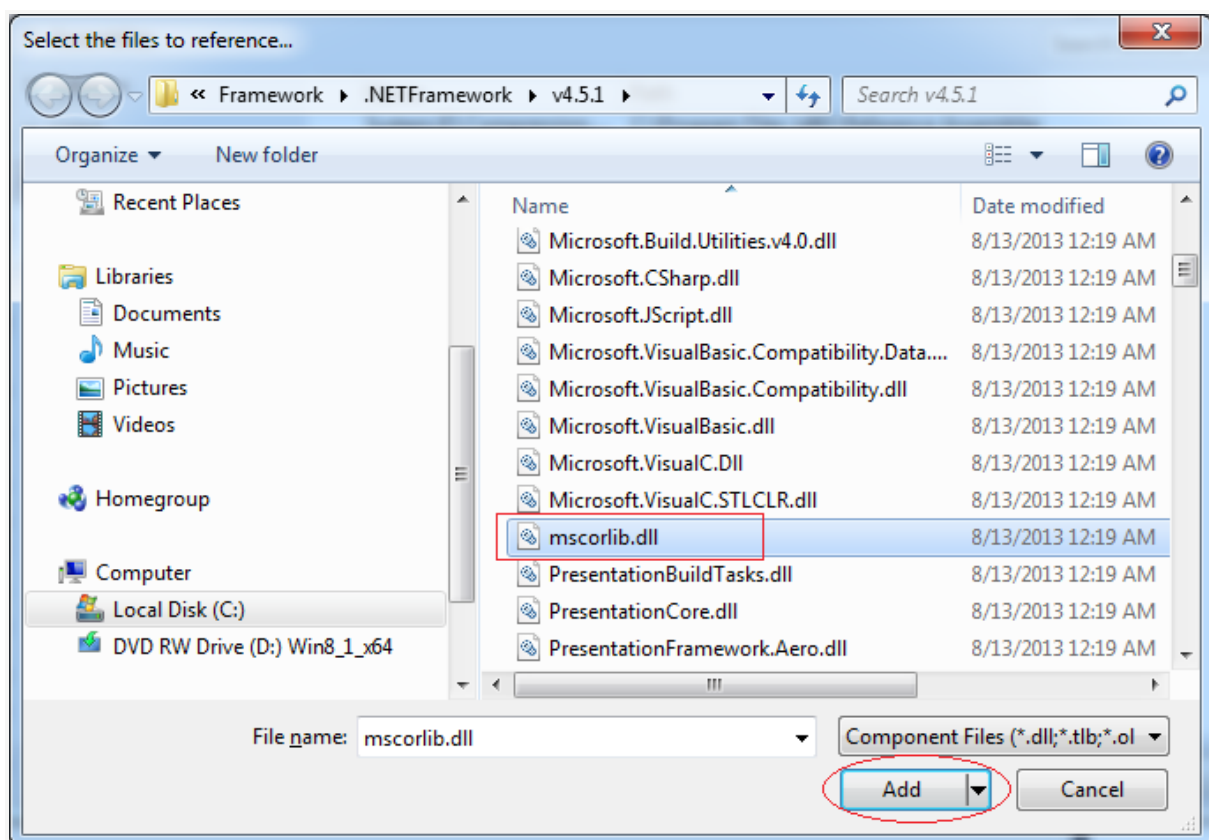


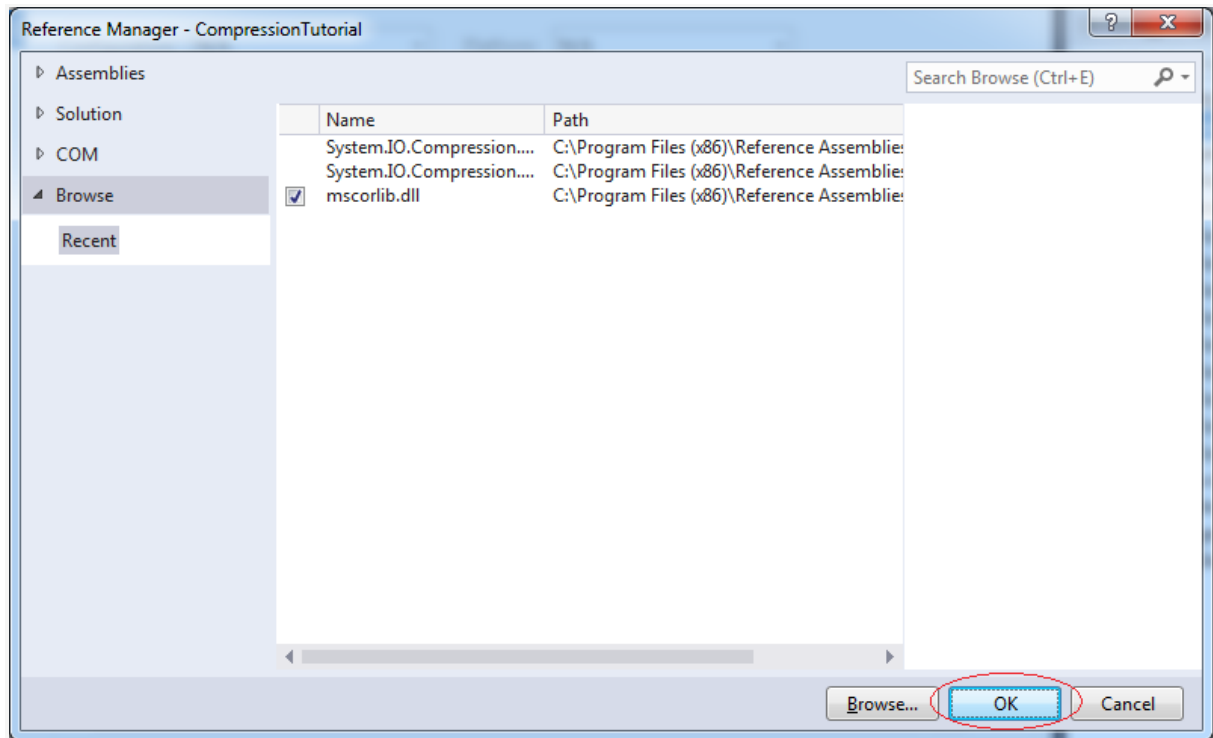
Tương tự nếu bạn nhận được thông báo "The name 'ZipArchive' does not exist in the current context" (Mặc dù đã khai báo **using System.IO.Compression**) bạn cần khai báo sử dụng thư viện **System.IO.Compression.dll**:





Tương tự nếu bạn nhận được thông báo "The name 'Path' does not exist in the current context" (Mặc dù đã khai báo **using System.IO**) bạn cần khai báo sử dụng thư viện **mscorlib.dll**:





Đa luồng trong C#

Khái niệm về đa luồng (Multithreading)

Đa luồng là một khái niệm quan trọng trong các ngôn ngữ lập trình, và C# cũng vậy, đó là cách tạo ra các luồng chương trình chạy song song với nhau.

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

```
namespace MultithreadingTutorial
{
```

```
    class HelloThread
    {
```

```
        public static void Main(string[] args)
        {
```

```
            Console.WriteLine("Create new Thread...\n");
```

```
            // Tạo ra một luồng con, để chạy song song với luồng chính.
            Thread newThread = new Thread(WriteB);
```

```
            Console.WriteLine("Start newThread...\n");
```

```
            // Kích hoạt chạy luồng newThread.
            newThread.Start();
```

```
        Console.WriteLine("Call Write('-') in main Thread...\n");

        // Trong luồng chính ghi ra các ký tự '-'
        for (int i = 0; i < 50; i++)
        {
            Console.Write('-');

            // Dừng lại 70 mili giây
            Thread.Sleep(70);
        }

        Console.WriteLine("Main Thread finished!\n");
        Console.Read();
    }

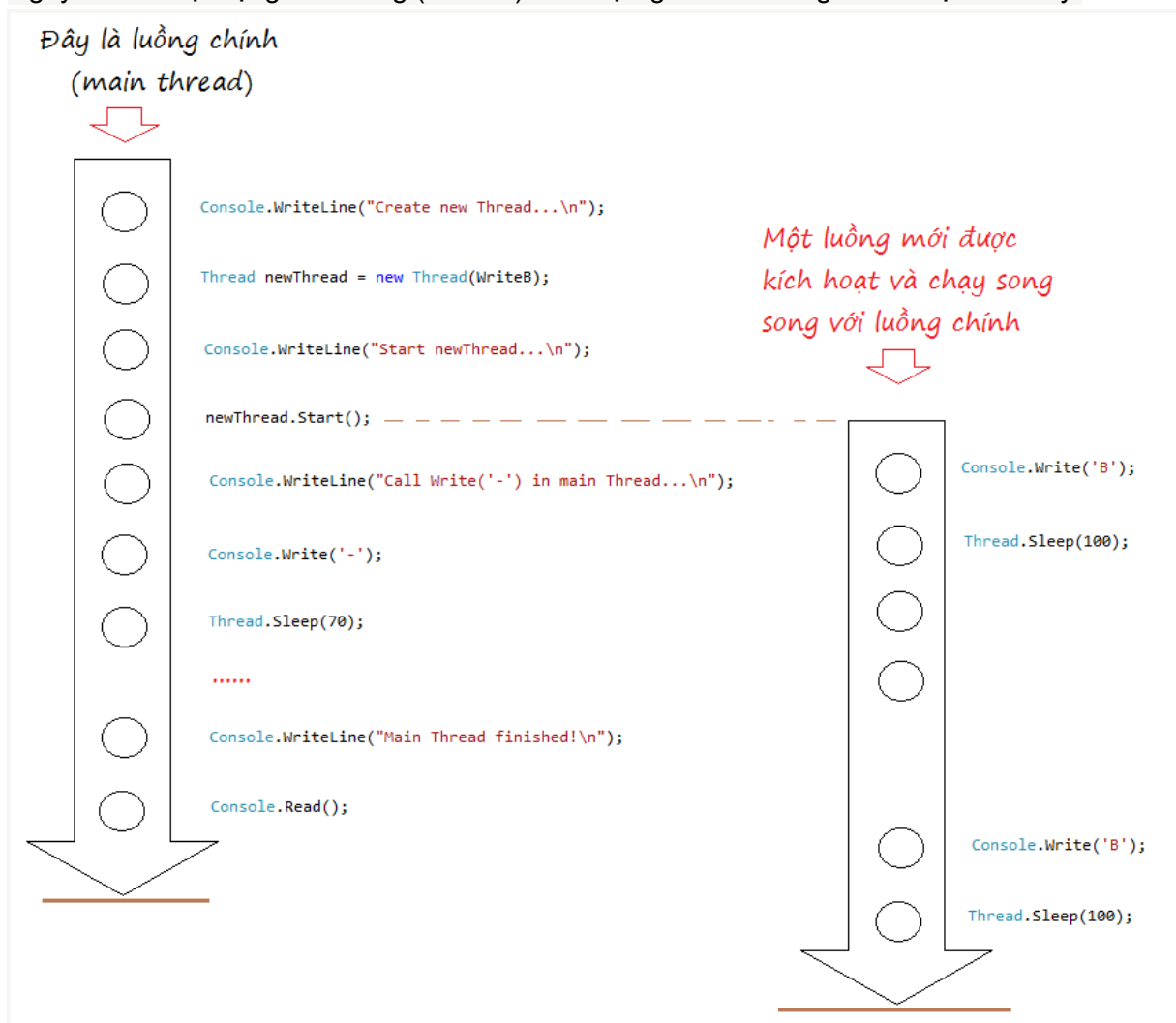
    public static void WriteB()
    {
        // Vòng lặp 100 lần ghi ra ký tự 'B'
        for (int i = 0; i < 100; i++)
        {
            Console.Write('B');

            // Dừng lại 100 mili giây
            Thread.Sleep(100);
        }
    }
}

}
```



Nguyên tắc hoạt động của luồng (Thread) chỉ được giải thích trong minh họa dưới đây:



Truyền tham số vào Thread

Ở phần trên bạn đã làm quen với ví dụ **HelloThread**, bạn đã tạo ra một đối tượng bao lấy (wrap) một phương thức tĩnh để thực thi phương thức này song song với luồng (thread) cha.

Phương thức tĩnh có thể trở thành một tham số truyền vào Constructor của lớp **Thread** nếu phương thức đó không có tham số, hoặc có một tham số duy nhất kiểu **object**.

```
/ Một phương thức tĩnh, không có tham số.
public static void LetGo()
{
    // Làm gì đó ở đây.
}

// Phương thức tĩnh có 1 tham số duy nhất, và kiểu là object.
public static void GetGo(object value)
{
    // Làm gì đó ở đây.
}

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

namespace MultithreadingTutorial
{
    class MyWork
    {
        public static void DoWork(object ch)
        {
            for (int i = 0; i < 100; i++)
            {
                // Ghi ra màn hình
                Console.Write(ch);

                // Nghỉ 50 mili giây.
                Thread.Sleep(50);
            }
        }
    }
}

using System;
```

```

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

namespace MultithreadingTutorial
{
    class ThreadParamDemo
    {
        public static void Main(string[] args)
        {
            Console.WriteLine("Create new thread.. \n");

            // Tạo một đối tượng Thread bao lấy phương thức tính
            MyWork.DoWork
            Thread workThread = new Thread(MyWork.DoWork);

            Console.WriteLine("Start workThread...\n");

            // Chạy workThread,
            // và truyền vào tham số cho phương thức MyWork.DoWork.
            workThread.Start("");

            for (int i = 0; i < 20; i++)
            {
                Console.Write(".");

                // Ngừng 30 giây.
                Thread.Sleep(30);
            }

            Console.WriteLine("MainThread ends");
            Console.Read();
        }
    }
}

```

Thread sử dụng phương thức không tĩnh

Bạn cũng có thể tạo một luồng (thread) sử dụng các phương thức thông thường. Xem ví dụ:

```

using System;
using System.Collections.Generic;
using System.Linq;

```



```

using System.Text;
using System.Threading.Tasks;
using System.Threading;

namespace MultithreadingTutorial
{
    class Worker
    {
        private string name;
        private int loop;

        public Worker(string name, int loop)
        {
            this.name = name;
            this.loop = loop;
        }

        public void DoWork(object value)
        {
            for (int i = 0; i < loop; i++)
            {
                Console.WriteLine(name + " working " + value);
                Thread.Sleep(50);
            }
        }
    }
}

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

namespace MultithreadingTutorial
{
    class WorkerTest
    {
        public static void Main(string[] args)
        {
            Worker w1 = new Worker("Tran", 10);

            // Tạo một đối tượng Thread.
            Thread workerThread1 = new Thread(w1.DoWork);

```

```

        // Truyền tham số vào phương thức DoWork.
        workerThread1.Start("A");

        Worker w2 = new Worker("Marry",15);

        // Tạo một đối tượng Thread.
        Thread workerThread2 = new Thread(w2.DoWork);

        // Truyền tham số vào phương thức DoWork.
        workerThread2.Start("B");

        Console.Read();
    }
}
}

```

```

file:///E:/CSHARP_TUTORIAL/MySolution/MultithreadingTutorial/bin/Debug/M...
Tran working A
Marry working B
Marry working B
Tran working A
Tran working A
Marry working B
Tran working A
Marry working B
Tran working A
Marry working B
Tran working A
Marry working B
Marry working B
Tran working A
Tran working A
Marry working B
Marry working B
Tran working A
Marry working B
Marry working B
Marry working B
Marry working B

```

ThreadStart Delegate

ThreadStart là một class ủy quyền (Delegate), nó được khởi tạo bằng cách bao lấy một phương thức. Và nó được truyền vào như một tham số để khởi tạo đối tượng **Thread**.

```

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

```

```

namespace MultithreadingTutorial
{
    class Programmer
    {
        private string name;
        public Programmer(string name)
        {
            this.name= name;
        }

        // Đây là một phương thức không tĩnh, không tham số.
        public void DoCode()
        {
            for (int i = 0; i < 5; i++)
            {
                Console.WriteLine(name + " coding ... ");
                Thread.Sleep(50);
            }
        }
    }
}

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

namespace MultithreadingTutorial
{
    class ThreadStartDemo
    {
        public static void Main(string[] args)
        {
            // Tạo một đối tượng ThreadStart bao lấy 1 phương thức tĩnh.
            // (Nó chỉ có thể bao lấy các phương thức không tham số)
            // (Nó là một đối tượng ủy quyền để thực thi phương thức).
            ThreadStart threadStart1 = new ThreadStart(DoWork);

            // Tạo một luồng bao lấy threadStart1.
            Thread workThread = new Thread(threadStart1);

```

```

        // Gọi start thread
        workThread.Start();

        // Khởi tạo một đối tượng Programmer.
        Programmer tran = new Programmer("Tran");

        // Bạn cũng có thể tạo ra đối tượng ThreadStart bao lấy phương
thức không tĩnh.
        // (ThreadStart chỉ có thể bao lấy các phương thức không tham
số)

        ThreadStart threadStart2 = new ThreadStart(tran.DoCode);

        // Tạo một luồng bao lấy threadStart2.
        Thread progThread = new Thread(threadStart2);

        progThread.Start();

        Console.WriteLine("Main thread ends");
        Console.Read();
    }

    public static void DoWork()
    {
        for (int i = 0; i < 10; i++)
        {
            Console.Write("*");
            Thread.Sleep(100);
        }
    }
}
}

```



```
file:///E:/CSHARP_TUTORIAL/MySolution/MultithreadingTutorial/bin/Debug/M...
Main thread ends
*Tran coddng ...
Tran coddng ...
Tran coddng ...
*Tran coddng ...
*Tran coddng ...
*****
```

Thread với các code nặc danh

Ở các phần trên bạn đã tạo ra các **Thread** sử dụng một phương thức cụ thể. Bạn có thể tạo ra một thread để thực thi một đoạn code bất kỳ.

// Sử dụng delegate() ám chỉ rằng bạn đang tạo ra một phương thức nặc danh.
delegate()

```
{
    // Làm gì đó ở đây.
}
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Threading;

namespace MultithreadingTutorial
{
    class ThreadUsingSnippetCode
    {

        public static void Main(string[] args)
        {

            Console.WriteLine("Create thread 1");

            // Tạo ra một thread để thực thi một đoạn code.
            Thread newThread1 = new Thread(
                delegate()
                {
                    for (int i = 0; i < 10; i++)
                    {
```

```

        Console.WriteLine("Code in delegate() " + i);
        Thread.Sleep(50);
    }

    }

);

Console.WriteLine("Start newThread1");

// Bắt đầu thread.
newThread1.Start();

Console.WriteLine("Create thread 2");

// Tạo ra một thread để thực thi một đoạn code.
Thread newThread2 = new Thread(
    delegate(object value)
    {
        for (int i = 0; i < 10; i++)
        {
            Console.WriteLine("Code in delegate(object) " + i +
" - " + value);
            Thread.Sleep(100);
        }
    }
);

Console.WriteLine("Start newThread2");

// Bắt đầu thread 2.
// Truyền giá trị vào cho delegate().
newThread2.Start("!!!");

Console.WriteLine("Main thread ends");
Console.Read();

}

}

}

```

```
file:///E:/CSHARP_TUTORIAL/MySolution/MultithreadingTutorial/bin/Debug/M...
Create thread 1
Start newThread1
Create thread 2
Start newThread2
Main thread ends
Code in delegate<> 0
Code in delegate<object> 0 - !!!
Code in delegate<> 1
Code in delegate<> 2
Code in delegate<object> 1 - !!!
Code in delegate<> 3
Code in delegate<object> 2 - !!!
Code in delegate<> 4
Code in delegate<> 5
Code in delegate<object> 3 - !!!
Code in delegate<> 6
Code in delegate<> 7
Code in delegate<object> 4 - !!!
Code in delegate<> 8
Code in delegate<> 9
Code in delegate<object> 5 - !!!
Code in delegate<object> 6 - !!!
Code in delegate<object> 7 - !!!
Code in delegate<object> 8 - !!!
Code in delegate<object> 9 - !!!
```

Đặt tên cho Thread

Trong lập trình đa luồng bạn có thể chủ động đặt tên cho luồng (thread), nó thực sự có ích trong trường hợp gỡ lỗi (Debugging), để biết đoạn code đó đang được thực thi trong thread nào.

Trong một thread bạn có thể gọi ***Thread.CurrentThread.Name*** để lấy ra tên của luồng đang thực thi tại thời điểm đó.

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

namespace MultithreadingTutorial
{
    class NamingThreadDemo
    {
        public static void Main(string[] args)
        {
            // Sét đặt tên cho luồng hiện thời
            // (Đang là luồng chính).
            Thread.CurrentThread.Name = "Main";

            Console.WriteLine("Code of " + Thread.CurrentThread.Name);

            Console.WriteLine("Create new thread");
        }
    }
}
```


Độ ưu tiên giữa các Thread

Trong C# có 5 mức độ ưu tiên của một luồng, chúng được định nghĩa trong enum **ThreadPriority**.

```
enum ThreadPriority {  
    Lowest,  
    BelowNormal,  
    Normal,  
    AboveNormal,  
    Highest  
}
```

Thông thường với các máy tính tốc độ cao, nếu các luồng chỉ làm số lượng công việc ít, bạn rất khó phát hiện ra sự khác biệt giữa các luồng có ưu tiên cao và luồng có ưu tiên thấp.

```
using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;  
using System.Threading.Tasks;  
using System.Threading;  
  
namespace MultithreadingTutorial  
{  
    class ThreadPriorityDemo  
    {  
        private static DateTime endsDateTime1;  
        private static DateTime endsDateTime2;  
  
        public static void Main(string[] args)  
        {  
            endsDateTime1 = DateTime.Now;  
            endsDateTime2 = DateTime.Now;  
  
            Thread thread1 = new Thread>Hello1);  
  
            // Sét độ ưu tiên cao nhất cho thread1  
            thread1.Priority = ThreadPriority.Highest;  
  
            Thread thread2 = new Thread>Hello2);  
  
            // Sét độ ưu tiên thấp nhất cho thread2.  
            thread2.Priority = ThreadPriority.Lowest;  
  
            thread2.Start(); thread1.Start();  
        }  
    }  
}
```

```

        Console.Read();
    }

    public static void Hello1()
    {
        for (int i = 0; i < 100000; i++)
        {
            Console.WriteLine("Hello from thread 1: "+ i);
        }
        // Thời điểm thread1 kết thúc.
        endsDateTime1 = DateTime.Now;

        PrintInterval();
    }

    public static void Hello2()
    {
        for (int i = 0; i < 100000; i++)
        {
            Console.WriteLine("Hello from thread 2: "+ i);
        }
        // Thời điểm thread2 kết thúc.
        endsDateTime2 = DateTime.Now;

        PrintInterval();
    }

    private static void PrintInterval()
    {
        // Khoảng thời gian
        TimeSpan interval = endsDateTime2 - endsDateTime1;

        // Tổng số mili giây hơn kém nhau.
        Console.WriteLine("Thread2 - Thread1 = " +
interval.TotalMilliseconds + " milliseconds");
    }
}

```

```
file:///E:/CSHARP_TUTORIAL/MySolution/MultithreadingTutorial/bin/Debug/M...
Hello from thread 2: 99977
Hello from thread 2: 99978
Hello from thread 2: 99979
Hello from thread 2: 99980
Hello from thread 2: 99981
Hello from thread 2: 99982
Hello from thread 2: 99983
Hello from thread 2: 99984
Hello from thread 2: 99985
Hello from thread 2: 99986
Hello from thread 2: 99987
Hello from thread 2: 99988
Hello from thread 2: 99989
Hello from thread 2: 99990
Hello from thread 2: 99991
Hello from thread 2: 99992
Hello from thread 2: 99993
Hello from thread 2: 99994
Hello from thread 2: 99995
Hello from thread 2: 99996
Hello from thread 2: 99997
Hello from thread 2: 99998
Hello from thread 2: 99999
Thread2 - Thread1 = 31.2875 milliseconds
```

*Luồng 1 có độ ưu tiên cao hơn
đã kết thúc sớm hơn luồng 2 là
31 milli giây*

Sử dụng Join()

Thread.Join() là một method thông báo rằng hãy chờ thread này hoàn thành rồi thread cha mới được tiếp tục chạy.

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

```
namespace MultithreadingTutorial
{
    class ThreadJoinDemo
    {
        public static void Main(string[] args)
        {
            Console.WriteLine("Create new thread");

            Thread letgoThread = new Thread(LetGo);

            // Bắt đầu Thread.
            letgoThread.Start();

            // Nói với thread cha (ở đây sẽ là Main thread)
            // hãy chờ cho letgoThread xong rồi mới tiếp tục chạy.
            letgoThread.Join();
        }
    }
}
```

```

        // Dòng này phải chờ cho letgoThread hoàn thành mới được chạy.
        Console.WriteLine("Main thread ends");
        Console.Read();
    }

    public static void LetGo()
    {
        for (int i = 0; i < 15; i++)
        {
            Console.WriteLine("Let's Go " + i);
        }
    }
}

```

```

file:///E:/CSHARP_TUTORIAL/MySolution/MultithreadingTutorial/bin/Debug/M...
Create new thread
Let's Go 0
Let's Go 1
Let's Go 2
Let's Go 3
Let's Go 4
Let's Go 5
Let's Go 6
Let's Go 7
Let's Go 8
Let's Go 9
Let's Go 10
Let's Go 11
Let's Go 12
Let's Go 13
Let's Go 14
Main thread ends

```

Sử dụng Yield()

Về mặt lý thuyết, **"Yield"** có nghĩa là để cho đi, từ bỏ, đầu hàng. Một luồng **Yield** nói với hệ điều hành rằng nó sẵn sàng để cho các thread khác được sắp xếp ở vị trí của nó. Điều này cho thấy rằng nó không phải làm một cái gì đó quá quan trọng. Lưu ý rằng nó chỉ là một gợi ý, mặc dù, và không đảm bảo có hiệu lực ở tất cả.

Như vậy phương thức **Yield()** được sử dụng khi bạn thấy rằng thread đó đang rảnh rỗi, nó không phải làm việc gì quan trọng, nên nó gợi ý hệ điều hành tạm thời nhường quyền ưu tiên cho các luồng khác.

Ví dụ dưới đây, có 2 luồng, mỗi luồng in ra một dòng văn bản 100K lần (con số đủ lớn để thấy sự khác biệt). Một luồng được sét độ ưu tiên cao nhất và một luồng được sét độ ưu tiên ít nhất. Đo khoảng thời gian kết thúc của 2 luồng.

```
using System;
```

```

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

namespace MultithreadingTutorial
{
    class ThreadYieldDemo
    {

        private static DateTime importantEndTime;
        private static DateTime unImportantEndTime;

        public static void Main(string[] args)
        {
            importantEndTime = DateTime.Now;
            unImportantEndTime = DateTime.Now;

            Console.WriteLine("Create thread 1");

            Thread importantThread = new Thread(ImportantWork);

            // Sét đặt quyền ưu tiên cao nhất cho luồng này.
            importantThread.Priority = ThreadPriority.Highest;

            Console.WriteLine("Create thread 2");

            Thread unImportantThread = new Thread(UnImportantWork);

            // Sét đặt quyền ưu tiên thấp nhất cho luồng này.
            unImportantThread.Priority = ThreadPriority.Lowest;

            // Bắt đầu các luồng.
            unImportantThread.Start();
            importantThread.Start();

            Console.Read();

        }

        // Một việc quan trọng, yêu cầu ưu tiên cao.
        public static void ImportantWork()
        {
            for (int i = 0; i < 100000; i++)
            {
                Console.WriteLine("\n Important work " + i);
            }
        }
    }
}

```

```

        // Thông báo với hệ điều hành, luồng này
        // nhường độ ưu tiên cho các luồng khác.
        Thread.Yield();
    }
    // Thời điểm thread này kết thúc.
    importantEndTime = DateTime.Now;
    PrintTime();
}

public static void UnImportantWork()
{
    for (int i = 0; i < 100000; i++)
    {
        Console.WriteLine("\n -- UnImportant work " + i);
    }
    // Thời điểm thread này kết thúc.
    unImportantEndTime = DateTime.Now;
    PrintTime();
}

private static void PrintTime()
{
    // Khoảng thời gian
    TimeSpan interval = unImportantEndTime - importantEndTime;

    // Tổng số mili giây hơn kém nhau.
    Console.WriteLine("UnImportant Thread - Important Thread = " +
interval.TotalMilliseconds + " mliseconds");
}

}

}

```

Chạy class trên trong trường hợp không có **Thread.Yield()**:

```
file:///E:/CSHARP_TUTORIAL/MySolution/MultithreadingTutorial/bin/Debug/M...
-- UnImportant work 99988
-- UnImportant work 99989
-- UnImportant work 99990
-- UnImportant work 99991
-- UnImportant work 99992
-- UnImportant work 99993
-- UnImportant work 99994
-- UnImportant work 99995
-- UnImportant work 99996
-- UnImportant work 99997
-- UnImportant work 99998
-- UnImportant work 99999
UnImportant Thread - Important Thread = 190.8192 milliseconds
```

Trường hợp không sử dụng Thread.Yield() trong luồng ưu tiên cao hơn.

Luồng ưu tiên cao hơn đã kết thúc sớm hơn luồng kia 190 milli giây

Chạy class trên trong trường hợp luồng ưu tiên cao hơn liên tục gọi **Thread.Yield()** để yêu cầu hệ thống tạm thời nhường ưu tiên sang các luồng khác.

```
file:///E:/CSHARP_TUTORIAL/MySolution/MultithreadingTutorial/bin/Debug/M...
Important work 99988
Important work 99989
Important work 99990
Important work 99991
Important work 99992
Important work 99993
Important work 99994
Important work 99995
Important work 99996
Important work 99997
Important work 99998
Important work 99999
UnImportant Thread - Important Thread = -928.2448 milliseconds
```

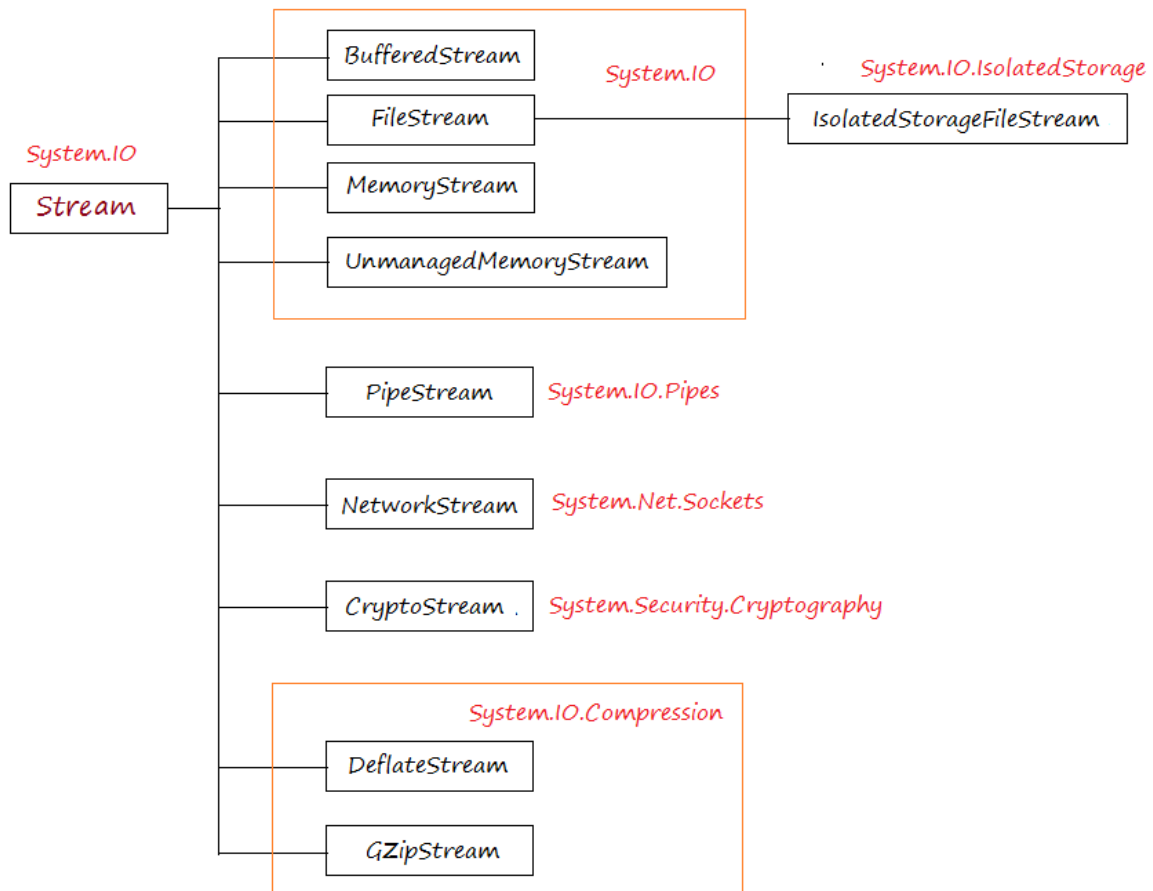
Trường hợp luồng ưu tiên cao hơn liên tục gọi Thread.Yield() để gợi ý hệ điều hành nhường ưu tiên cho các luồng khác.

Luồng ưu tiên cao hơn đã kết thúc chậm hơn luồng không ưu tiên 928 milli giây

Stream - luồng vào ra nhị phân trong C#

Tổng quan về Stream

Stream là một class nó mô phỏng một dòng các byte được sắp hàng một cách liên tiếp nhau. Chẳng hạn như việc truyền tải dữ liệu trên mạng các dữ liệu truyền đi là dòng các byte liên tiếp nhau từ byte đầu tiên cho tới các byte cuối cùng.



Stream là một class cơ sở, các luồng (stream) khác mở rộng từ class này. Có một vài class đã được xây dựng sẵn trong **C#**, chúng mở rộng từ lớp **Stream** cho các mục đích khác nhau, chẳng hạn:

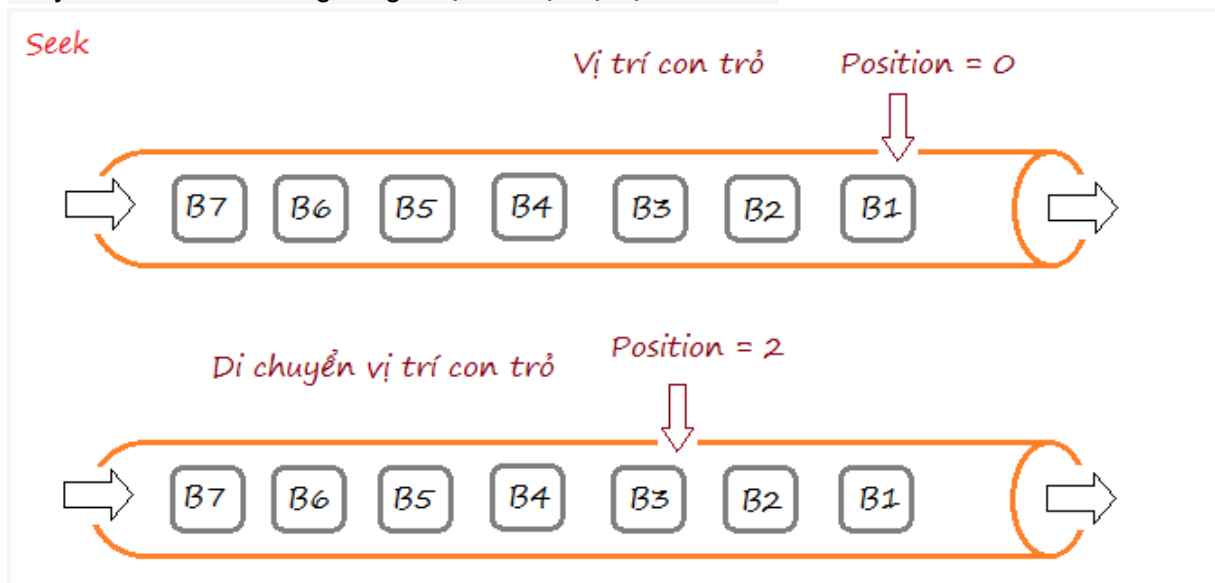
Class	Mô tả
BufferedStream	Một luồng tiện ích, nó bao bọc (wrap) một luồng khác giúp nâng cao hiệu năng luồng.
FileStream	Luồng sử dụng để đọc ghi dữ liệu vào file.
MemoryStream	Luồng làm việc với các dữ liệu trên bộ nhớ.
UnmanagedMemoryStream	
IsolatedStorageFileStream	
PipeStream	
NetworkStream	

CryptoStream	Luồng đọc ghi dữ liệu được mật mã hóa.
DeflateStream	
GZipStream	

Stream là một lớp trừu tượng, tự nó không thể khởi tạo một đối tượng, bạn có thể khởi tạo một đối tượng Stream từ các phương thức khởi tạo (Constructor) của class con. Lớp **Stream** cung cấp các phương thức cơ bản làm việc với luồng dữ liệu, cụ thể là các phương thức đọc ghi một **byte** hoặc một mảng các **byte**..



Tùy thuộc vào luồng, có những luồng hỗ trợ cả đọc và ghi, và cả tìm kiếm (seek) bằng cách di chuyển con trỏ trên luồng, và ghi đọc dữ liệu tại vị trí con trỏ.



Các thuộc tính (property) của **Stream**:

Thuộc tính	Mô tả
CanRead	Thuộc tính cho biết luồng này có hỗ trợ đọc không.

CanSeek	Thuộc tính cho biết luồng này có hỗ trợ tìm kiếm (seek) hay không
CanWrite	Thuộc tính cho biết luồng này có hỗ trợ ghi hay không
Length	Trả về độ dài của luồng (Số bytes)
Position	Vị trí hiện tại của con trỏ trên luồng.

Ví dụ cơ bản Stream

Với **Stream** bạn có thể ghi từng **byte** hoặc ghi một mảng các **byte** vào luồng (stream). Và khi đọc bạn có thể đọc từng **byte** hoặc đọc nhiều **byte** và gán vào một mảng tạm.

*Một **byte** là 8 **bit**, trong đó một **bit** là 0 hoặc 1. Như vậy 1 **byte** tương ứng với một số từ 0 tới 255 ($2^8 - 1$).*

Ví dụ luồng ghi dữ liệu

Và bây giờ hãy bắt đầu với một ví dụ đơn giản, tạo một **Stream** ghi dữ liệu vào File. Bạn có thể ghi từng byte vào stream hoặc ghi một mảng các byte vào Stream.

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

namespace CSharpStreamsTutorial
{
    class StreamWriterDemo
    {
        public static void Main(string[] args)
        {
            string path = @"C:\temp\MyTest.txt";

            // Tạo thư mục cha.
            Directory.CreateDirectory(@"C:\temp");

            // Tạo một đối tượng Stream từ class con FileStream.
            // FileMode.Create: Tạo file mới để ghi, nếu file đã tồn tại ghi
            // đè file này.
            Stream writingStream = new FileStream(path, FileMode.Create);

            try
            {
                // Một mảng byte (1byte < 2^8).
```

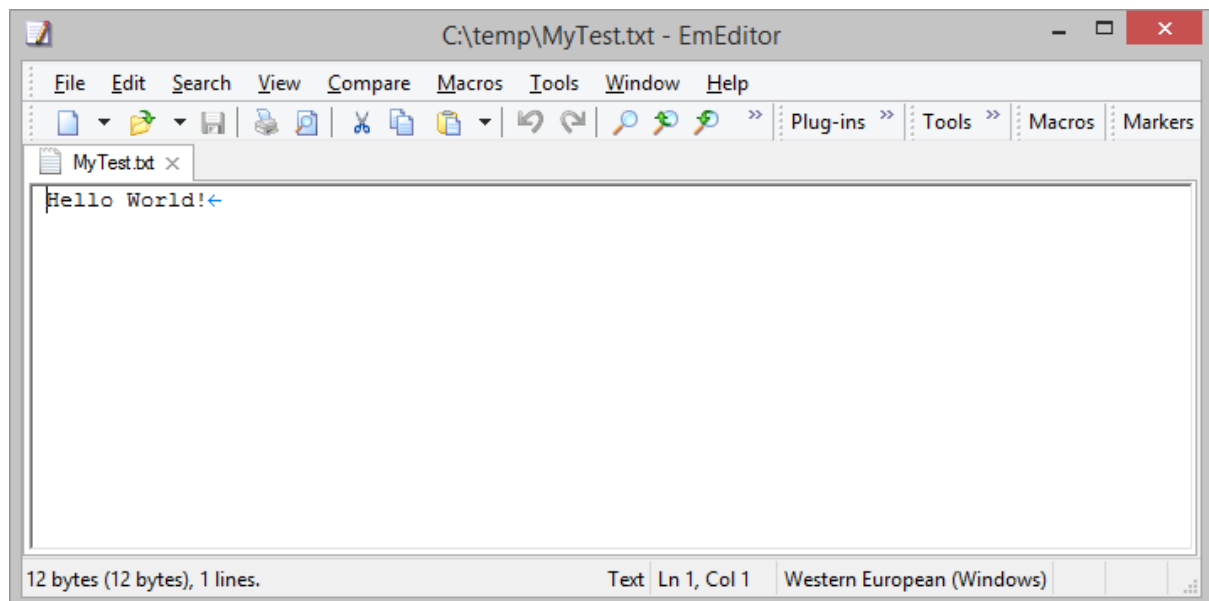
```

        // Tương ứng với {'H','e','l','l','o',' ','
        ', 'W','o','r','l','d'}
        byte[] bytes = new byte[] { 72, 101, 108, 108, 111, 32, 87,
        111, 114, 108, 100 };

        if (writingStream.CanWrite)
        {
            writingStream.Write(bytes, 0, bytes.Length);

            // Ghi thêm một byte (33 = '!')
            writingStream.WriteByte(33);
        }
    }
    catch (Exception e)
    {
        Console.WriteLine("Error:" + e);
    }
    finally
    {
        // Đóng Stream, giải phóng tài nguyên.
        writingStream.Close();
    }
    Console.ReadLine();
}
}
}

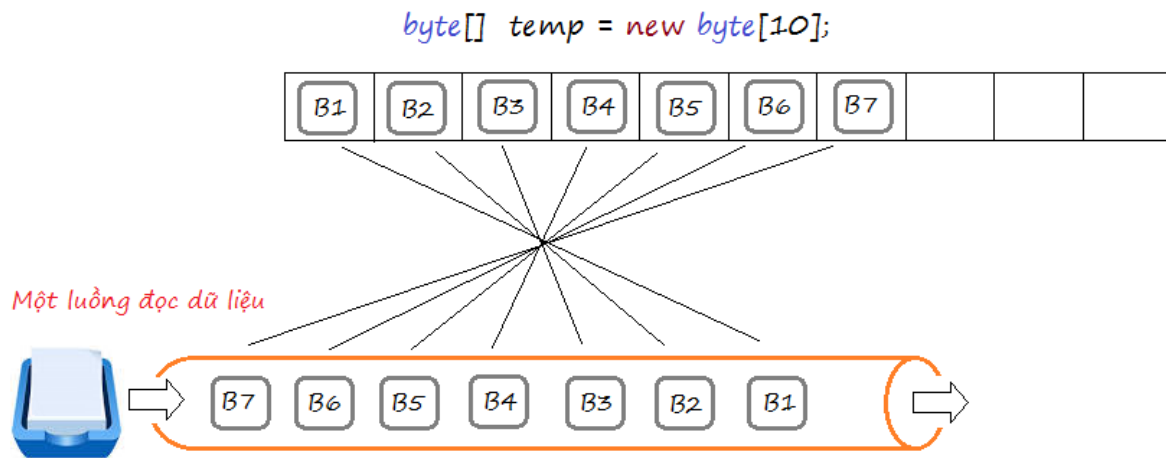
```



Ví dụ luồng đọc dữ liệu

Ví dụ ở trên bạn đã ghi dữ liệu vào file **C:\temp\MyTest.txt**, bây giờ bạn có thể viết một stream đọc dữ liệu từ file đó.

Một mảng tạm để chứa các byte đọc được:



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

namespace CSharpStreamsTutorial
{
    class StreamReadDemo
    {
        public static void Main(string[] args)
        {
            String path = @"C:\temp\MyTest.txt";

            if (!File.Exists(path))
            {
                Console.WriteLine("File " + path + " does not exists!");
                return;
            }

            // File đã được tạo ra, giờ tạo một Stream để đọc.
            // Tạo một đối tượng Stream từ class con FileStream.
            // FileMode.Open: Mở file để đọc.
            using (Stream readingStream = new FileStream(path,
                FileMode.Open))
            {
                byte[] temp = new byte[10];
                UTF8Encoding encoding = new UTF8Encoding(true);
```

```

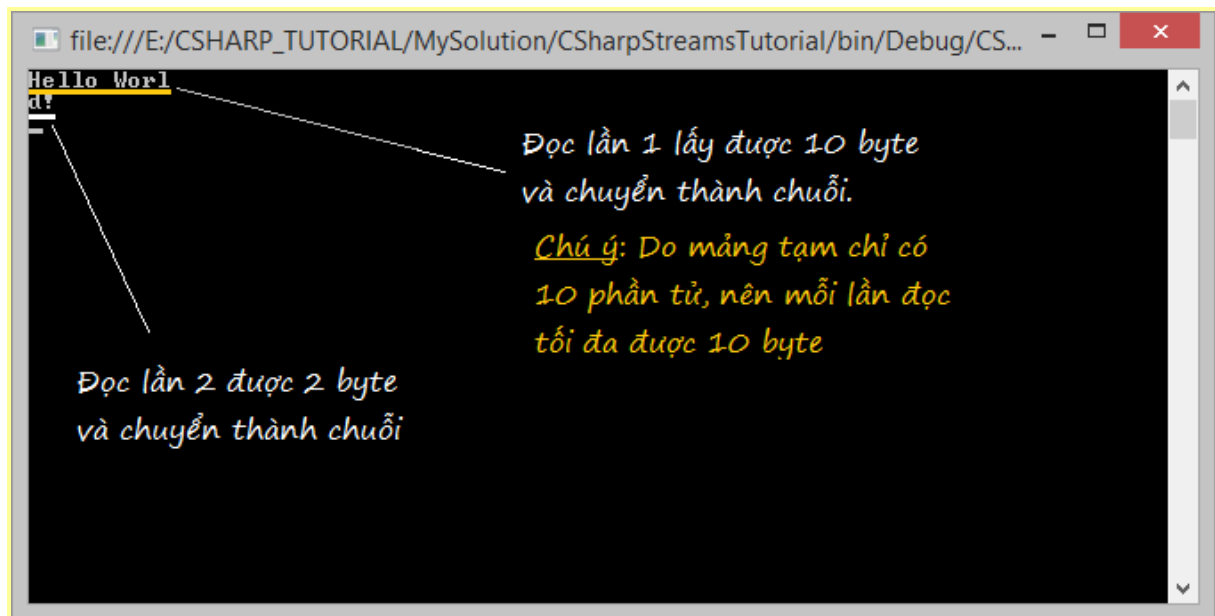
        int len = 0;

        // Đọc các phần tử trên luồng gán vào các phần tử của mảng
temp.
        // (Gán vào các vị trí bắt đầu từ 0, mỗi lần đọc tối đa
temp.Length phần tử)
        // Đồng thời trả về số byte đọc được.
        while ((len = readingStream.Read(temp, 0, temp.Length)) > 0)
        {
            // Chuyển mảng temp chứa các byte vừa đọc được thành
chuỗi.

            // (Lấy 'len' phần tử bắt đầu từ vị trí 0).
            String s = encoding.GetString(temp, 0, len);
            Console.WriteLine(s);
        }

        Console.ReadLine();
    }
}
}
}
}

```



FileStream

FileStream là một lớp mở rộng từ lớp **Stream**, **FileStream** được sử dụng để đọc và ghi dữ liệu vào file, nó được thừa kế các thuộc tính (property), phương thức từ **Stream**, đồng thời có thêm các chức năng dành riêng cho đọc ghi dữ liệu vào file.

Có một vài chế độ đọc ghi dữ liệu vào file:

FileMode	Mô tả
Append	Mở file nếu nó đã tồn tại, di chuyển con trỏ về cuối tập tin để nó thể ghi nối tiếp vào file, nếu file không tồn tại nó sẽ được tạo ra.
Create	Nói với hệ điều hành tạo một tập tin mới. Nếu tập tin đã tồn tại, nó sẽ được ghi đè.
CreateNew	Nói với hệ điều hành tạo ra một file mới. Nếu file đã tồn tại ngoại lệ IOException sẽ được ném ra. Chế độ này yêu cầu phải có quyền FileIOPermissionAccess.Write
Open	Nói với hệ điều hành để mở một file đã tồn tại. Một ngoại lệ System.IO.FileNotFoundException sẽ được ném ra nếu file không tồn tại.
OpenOrCreate	Nói với hệ điều hành nên mở một tập tin nếu nó tồn tại; nếu không, một tập tin mới sẽ được tạo ra.
Truncate	Nói với hệ điều hành nên mở tập tin khi nó tồn tại. Và khi file được mở, nó sẽ bị cắt hết nội dung trở về 0 byte.

```

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

namespace CSharpStreamsTutorial
{
    class FileStreamFileModeDemo
    {
        public static void Main(string[] args)
        {
            String path = @"C:\temp\MyTest.txt";

            if (!File.Exists(path))
            {
                Console.WriteLine("File " + path + " does not exists!");

                // Đảm bảo rằng thư mục chứa tồn tại.
                Directory.CreateDirectory(@"C:\temp");
            }

            // Tạo ra một FileStream để ghi dữ liệu.
            // (FileMode.Append: Mở file ra để ghi tiếp vào phía cuối của
file,

```

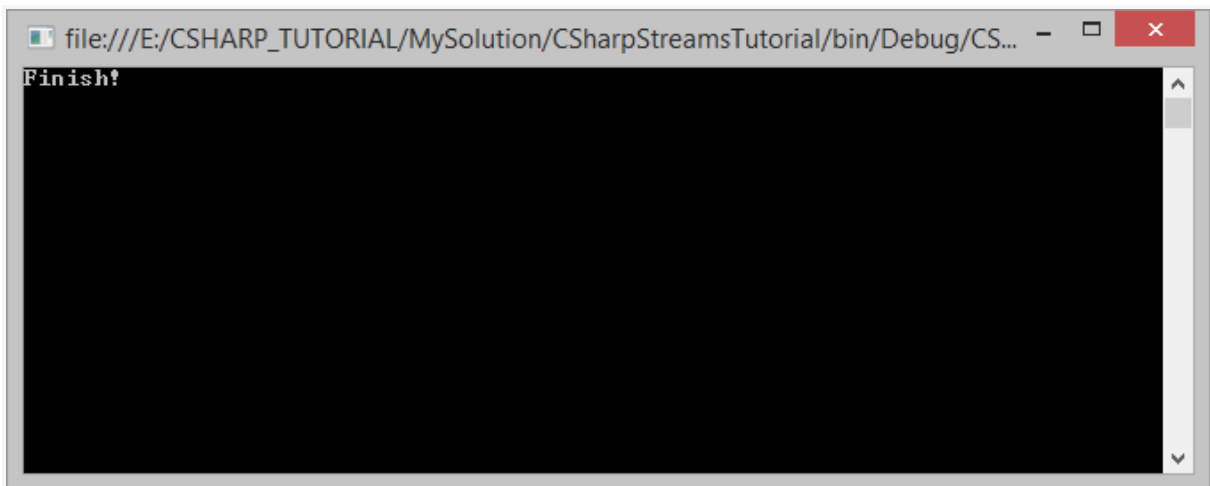
```
// nếu file không tồn tại sẽ tạo mới).

using (FileStream writeFileStream = new FileStream(path,
FileMode.Append) )
{
    string s = "\nHello every body!";

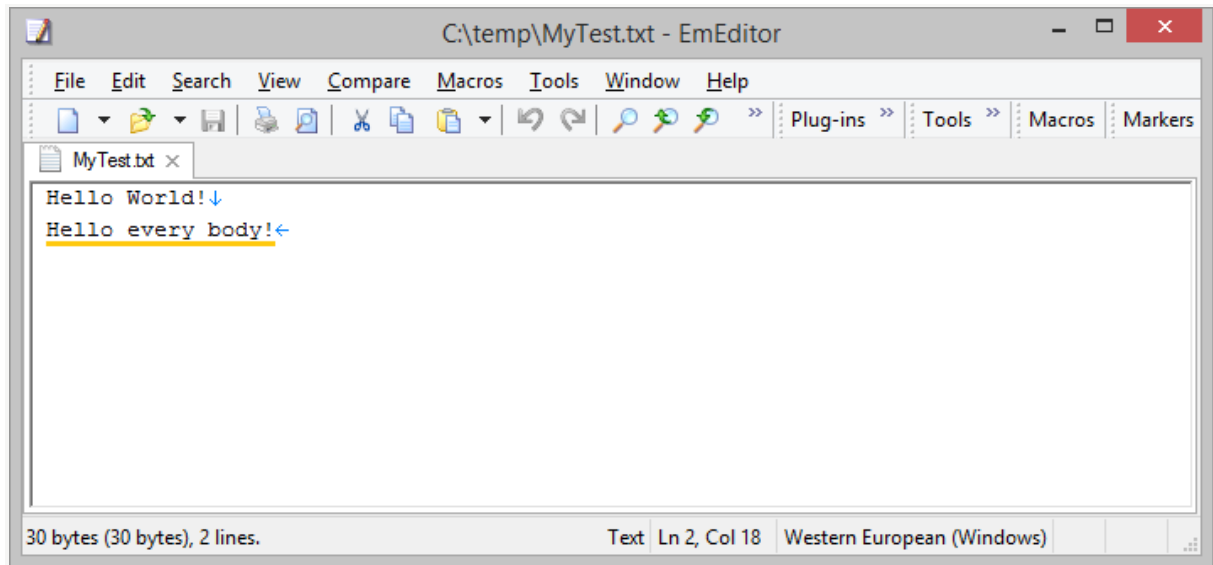
    // Chuyển một chuỗi thành mảng các byte theo mã hóa UTF8.
    byte[] bytes = Encoding.UTF8.GetBytes(s);

    // Ghi các byte xuống file.
    writeFileStream.Write(bytes, 0, bytes.Length);
}
Console.WriteLine("Finish!");

Console.ReadLine();
}
}
```



Với **FileMode.Append** dữ liệu sẽ được nối thêm vào file, nếu file đó đã tồn tại:



Phương thức khởi tạo (Constructor):

Class **FileStream** có 11 constructor (Không tính các constructor bị lỗi thời) dùng để khởi tạo một đối tượng **FileStream**:

```
FileStream(SafeFileHandle, FileAccess)
```

```
FileStream(SafeFileHandle, FileAccess, Int32)
```

```
FileStream(SafeFileHandle, FileAccess, Int32, Boolean)
```

```
FileStream(String, FileMode)
```

```
FileStream(String, FileMode, FileAccess)
```

```
FileStream(String, FileMode, FileAccess, FileShare)
```

```
FileStream(String, FileMode, FileAccess, FileShare, Int32)
```

```
FileStream(String, FileMode, FileAccess, FileShare, Int32, Boolean)
```

```
FileStream(String, FileMode, FileAccess, FileShare, Int32, FileOptions)
```

```
FileStream(String, FileMode, FileSystemRights, FileShare, Int32,  
FileOptions)
```

```
FileStream(String, FileMode, FileSystemRights, FileShare, Int32,  
FileOptions, FileSecurity)
```

Tuy nhiên bạn cũng có các cách khác để tạo đối tượng **FileStream**, chẳng hạn thông qua **FileInfo**, đây là class đại diện cho một file trong hệ thống.

Phương thức của FileInfo trả về FileStream.	Mô tả
Create()	Bởi mặc định, tất cả các quyền đọc ghi file mới này sẽ gán cho tất cả các users.
Open(FileMode)	Mở file với chế độ được chỉ định.
Open(FileMode, FileAccess)	Mở file với chỉ định chế độ đọc, ghi, hoặc quyền đọc ghi.
Open(FileMode, FileAccess, FileShare)	Mở file với chỉ định chế độ đọc, ghi, hoặc quyền đọc ghi, và các lựa chọn chia sẻ.
OpenWrite()	Tạo ra một FileStream chỉ để ghi dữ liệu.
OpenRead()	Tạo ra FileStream chỉ để đọc dữ liệu.

Ví dụ tạo **FileStream** từ **FileInfo**:

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

namespace CSharpStreamsTutorial
{
    class FileStreamFileInfoDemo
    {

        public static void Main(string[] args)
        {
            FileInfo afile = new FileInfo(@"C:\temp\MyTest.txt");

            if (afile.Exists)
            {
                Console.WriteLine("File does not exist!");
                Console.Read();
                return;
            }
            // Mở file và cắt hết dữ liệu file hiện tại.
            using (FileStream stream = afile.Open(FileMode.Truncate))
            {
                String s = "New text";

                byte[] bytes = Encoding.UTF8.GetBytes(s);

                stream.Write(bytes, 0, bytes.Length);
            }
        }
    }
}
```

```

    }

    Console.WriteLine("Finished!");
    Console.Read();
}
}
}

```

BufferedStream

BufferedStream là một lớp mở rộng từ lớp **Stream**, nó là một luồng (stream) bộ đệm bao lấy (wrap) một stream khác, giúp nâng cao hiệu quả đọc ghi dữ liệu.

BufferedStream chỉ có 2 phương thức khởi tạo (Constructor), nó bao lấy một stream khác.

Tôi đưa ra một tình huống, bạn tạo ra một luồng bộ đệm (**BufferedStream**) bao lấy

FileStream, với mục đích ghi dữ liệu xuống file. Các dữ liệu ghi vào luồng bộ đệm tạm thời sẽ nằm trên bộ nhớ, và khi bộ đệm đầy, dữ liệu tự động được đẩy (Flush) xuống file, bạn có thể chủ động đẩy dữ liệu xuống file bằng cách sử dụng phương thức **Flush()**. Sử dụng **BufferedStream** trong trường hợp này giúp giảm số lần phải ghi xuống ổ đĩa, và vì vậy nó làm tăng hiệu suất của chương trình.

```

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

namespace CSharpStreamsTutorial
{
    class BufferedStreamWriteFileDemo
    {
        public static void Main(string[] args)
        {
            String fileName = @"C:\temp\MyFile.txt";

            FileInfo file = new FileInfo(fileName);

            // Đảm bảo thư mục tồn tại.
            file.Directory.Create();

            // Tạo file mới, nếu nó đã tồn tại nó sẽ bị ghi đè.
            // Trả về một đối tượng FileStream.
            using (FileStream fileStream = file.Create())
            {
                // Tạo một đối tượng BufferedStream bao lấy FileStream.
                // (Chỉ định bộ đệm là 10000 bytes).
            }
        }
    }
}

```

```

        using (BufferedStream bs = new BufferedStream(fileStream,
10000))
        {
            int index = 0;
            for (index = 1; index < 2000; index++)
            {
                String s = "This is line " + index + "\n";

                byte[] bytes = Encoding.UTF8.GetBytes(s);

                // Ghi vào bộ đệm, khi bộ đệm đầy nó sẽ tự đẩy xuống
file.

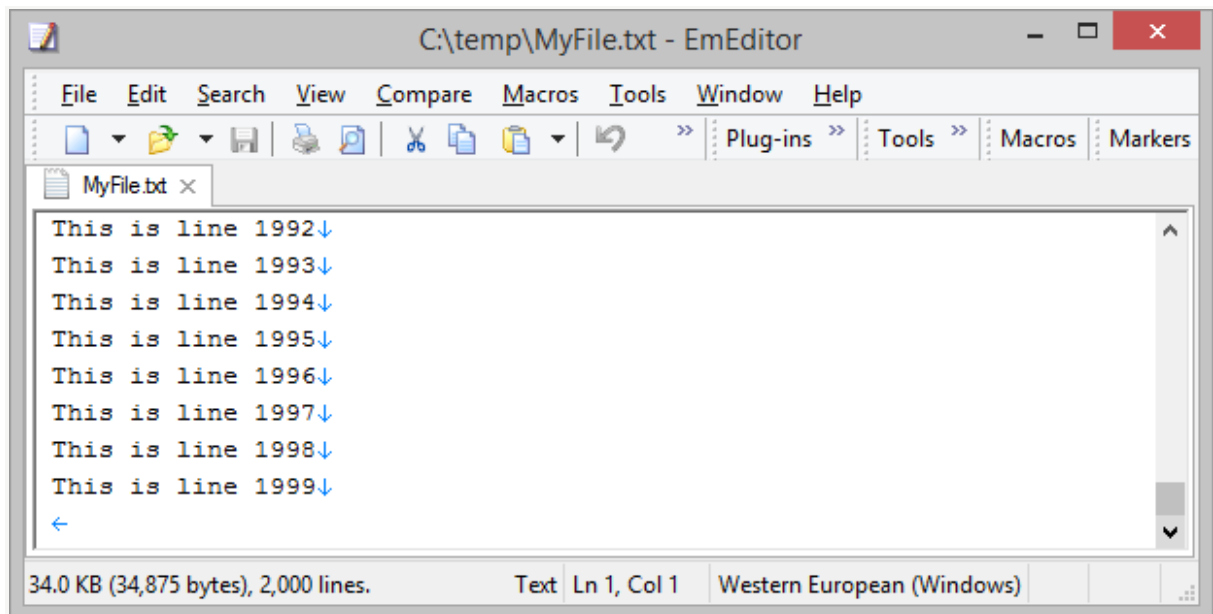
                bs.Write(bytes, 0, bytes.Length);
            }

            // Đẩy các dữ liệu còn lại trên bộ đệm xuống file.
            bs.Flush();
        }

    }

    Console.WriteLine("Finished!");
    Console.Read();
}
}
}

```

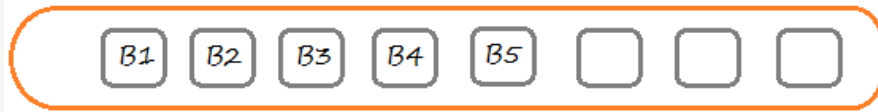


MemoryStream

MemoryStream là một lớp mở rộng trực tiếp từ lớp **Stream**, nó là luồng (stream) mà dữ liệu được lưu trữ (store) trên bộ nhớ.

Về bản chất **MemoryStream** là một đối tượng nó quản lý một bộ đệm (buffer) là một mảng các **byte**, khi các **byte** được ghi vào luồng này nó sẽ tự động được gán vào các vị trí tiếp theo tính từ vị trí hiện tại của con trỏ trên mảng. Khi bộ đệm đầy một mảng mới có kích thước lớn hơn được tạo ra, và copy các dữ liệu từ mảng cũ sang.

MemoryStream



Constructor:

MemoryStream()

MemoryStream(Byte[] buffer)

MemoryStream(Byte[] buffer, Boolean writable)

MemoryStream(Byte[] buffer, Int32 index, Int32 count, Boolean writable)

MemoryStream(Byte[] buffer, Int32 index, Int32 count, Boolean, Boolean publiclyVisible)

MemoryStream(Byte[], Int32, Int32, Boolean, Boolean)

MemoryStream(Int32 capacity)

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

```
namespace CSharpStreamsTutorial
{
    class MemoryStreamDemo
    {
        static void Main()
        {
```

```

// Tạo một đối tượng MemoryStream có dung lượng 100 bytes.
MemoryStream memoryStream = new MemoryStream(100);

byte[] javaBytes = Encoding.UTF8.GetBytes("Java");
byte[] csharpBytes = Encoding.UTF8.GetBytes("CSharp");

// Ghi các byte vào luồng bộ nhớ.
memoryStream.Write(javaBytes, 0, javaBytes.Length);
memoryStream.Write(csharpBytes, 0, csharpBytes.Length);

// Ghi ra sức chứa và độ dài của luồng.
// ==> Capacity: 100, Length: 10.
Console.WriteLine("Capacity: {0} , Length: {1}",
                  memoryStream.Capacity.ToString(),
                  memoryStream.Length.ToString());

// Lúc này vị trí con trỏ đang đứng ở sau ký tự 'p'.
// ==> 10.
Console.WriteLine("Position: " + memoryStream.Position);

// Di chuyển lùi con trỏ đi 6 byte, so với vị trí hiện tại.
memoryStream.Seek(-6, SeekOrigin.Current);

// Lúc này vị trí con trỏ đang đứng ở sau ký tự 'a' và trước
'C'.

// ==> 4.
Console.WriteLine("Position: " + memoryStream.Position);

byte[] vsBytes = Encoding.UTF8.GetBytes(" vs ");

// Ghi vào luồng bộ nhớ.
memoryStream.Write(vsBytes, 0, vsBytes.Length);

byte[] allBytes = memoryStream.GetBuffer();

string data = Encoding.UTF8.GetString(allBytes);

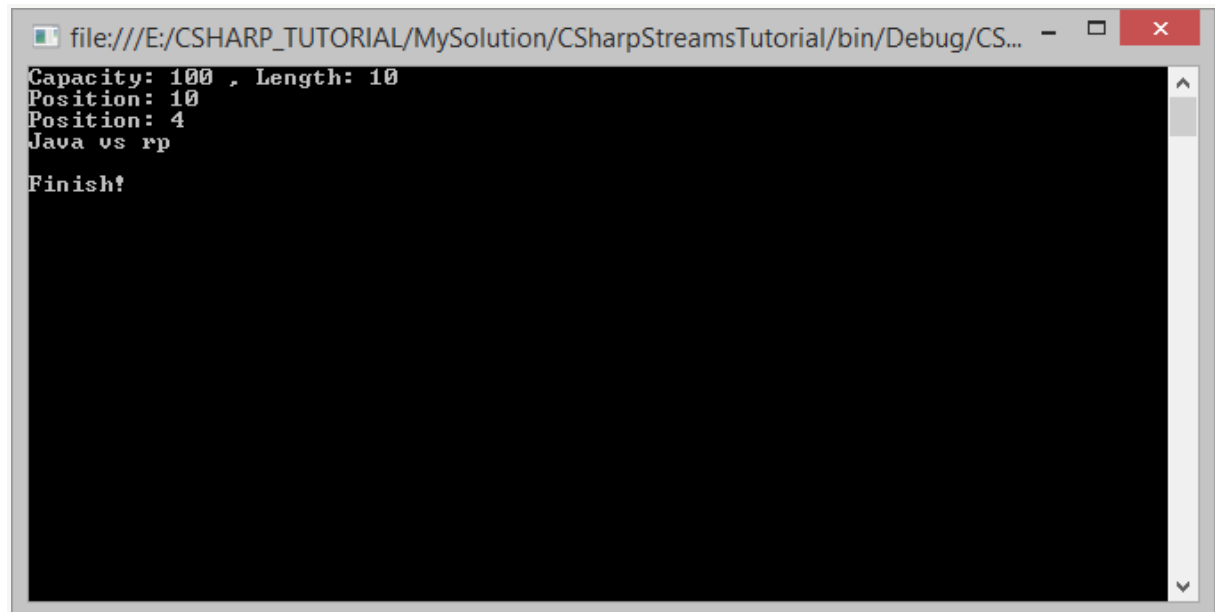
// ==> Java vs rp
Console.WriteLine(data);

Console.WriteLine("Finish!");
Console.Read();

}

```

```
}  
  
}
```



```
file:///E:/CSHARP_TUTORIAL/MySolution/CSharpStreamsTutorial/bin/Debug/CS...  
Capacity: 100 , Length: 10  
Position: 10  
Position: 4  
Java vs rp  
Finish!
```

UnmanagedMemoryStream

Sử dụng **UnmanagedMemoryStream** cho phép bạn đọc các luồng dữ liệu không được quản lý mà không cần sao chép tất cả chúng lên quản lý ở bộ nhớ **Heap** trước khi sử dụng. Nó giúp bạn tiết kiệm bộ nhớ nếu bạn đang phải đối phó với rất nhiều dữ liệu.

Lưu ý rằng có một giới hạn 2GB đối với **MemoryStream** vì vậy bạn phải sử dụng các **UnmanagedMemoryStream** nếu bạn vượt quá giới hạn này.

Tôi đưa ra một tình huống: Có các dữ liệu rời rạc nằm sẵn trên bộ nhớ. Và bạn có thể tập hợp chúng lại để quản lý bởi **UnmanagedMemoryStream** bằng cách quản lý các con trỏ (pointer) của các dữ liệu rời rạc nói trên, thay vì bạn copy chúng lên luồng (stream) để quản lý.

Inheritance Hierarchy

```
System.Object  
  System.MarshalByRefObject  
    System.IO.Stream  
      System.IO.UnmanagedMemoryStream  
        System.IO.MemoryMappedFiles.MemoryMappedViewStream
```

Constructor:

```
UnmanagedMemoryStream()
```

UnmanagedMemoryStream(Byte* pointer, Int64 length)

UnmanagedMemoryStream(Byte* pointer, Int64 length, Int64 capacity, FileAccess access)

UnmanagedMemoryStream(SafeBuffer buffer, Int64 offset, Int64 length)

UnmanagedMemoryStream(SafeBuffer buffer, Int64 offset, Int64 length, FileAccess access)

CryptoStream

CryptoStream là một lớp, sử dụng cho việc mật mã hóa luồng dữ liệu .

Hình ảnh minh họa dưới đây luồng **CryptoStream** bao lấy một luồng khác (chẳng hạn là luồng ghi file), khi bạn ghi dữ các **byte** lên **CryptoStream** các byte này sẽ bị mật mã hóa thành các byte khác trước khi đẩy sang luồng ghi vào file. Lúc này nội dung của file đã được mật mã hóa.

Chú ý rằng bạn có thể lựa chọn một thuật toán mật mã hóa khi tạo đối tượng **CryptoStream**.



Trong một tình huống ngược lại, một luồng **CryptoStream** bao lấy một luồng đọc file (File mà nội dung đã mã hóa ở trên), các byte trên luồng **FileStream** là các byte đã được mật mã hóa (encrypt), nó sẽ được giải mật (decrypt) bởi **CryptoStream**.

Một điều quan trọng bạn cần nhớ rằng, không phải thuật toán mật mã hóa nào cũng có 2 chiều mật mã hóa và giải mật mã hóa.



Ở đây tôi sử dụng thuật toán **DES** để mã hóa và giải mã, bạn cần cung cấp mảng 128 bit nó là chìa khóa bảo mật của bạn.

```
// Đối tượng cung cấp thuật toán mật mã hóa DES.
```

```
DESCryptoServiceProvider provider = new DESCryptoServiceProvider();
```

```
// Sử dụng khóa riêng của bạn (Phải là chuỗi 128bit = 8byte).
```

```
// (Tương đương với 8 ký tự ASCII)
```

```
provider.Key = ASCIIEncoding.ASCII.GetBytes("1234abcd");
```

```
provider.IV = ASCIIEncoding.ASCII.GetBytes("12345678");
```

```

// Đối tượng mật mã hóa.
ICryptoTransform encryptor = provider.CreateEncryptor();

// Đối tượng giải mật mã hóa
ICryptoTransform decryptor = provider.CreateDecryptor();
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.IO;
using System.Security.Cryptography;

namespace CSharpStreamsTutorial
{
    class CryptoStreamExample
    {
        public static void Main(string[] args)
        {
            // Cung cấp thuật toán mật mã hóa DES.
            DESCryptoServiceProvider provider = new
DESCryptoServiceProvider();

            // Sử dụng khóa riêng của bạn (Phải là chuỗi 128bit = 8byte).
            // (Tương đương với 8 ký tự ASCII)
            provider.Key = ASCIIEncoding.ASCII.GetBytes("1234abcd");
            provider.IV = ASCIIEncoding.ASCII.GetBytes("12345678");

            String encryedFile = @"C:\temp\EncryptedFile.txt";

            // Một luồng để ghi file.
            using (FileStream stream = new FileStream(encryedFile,
FileMode.OpenOrCreate, FileAccess.Write))
            {

                // Đối tượng mật mã hóa.
                ICryptoTransform encryptor = provider.CreateEncryptor();

                // Tạo một luồng mật mã hóa bao lấy luồng ghi file.
                using (CryptoStream cryptoStream = new CryptoStream(stream,
                    encryptor, CryptoStreamMode.Write))
                {
                    // Một mảng byte chưa được mật mã hóa.

```



```

        byte[] data = ASCIIEncoding.ASCII.GetBytes("Bear, I love
you. OK?");

        // Ghi vào luồng mật mã hóa.
        cryptoStream.Write(data, 0, data.Length);
    }

}

Console.WriteLine("Write to file: " + encryedFile);

// Tiếp theo đọc file mã hóa vừa được tạo ra ở trên.

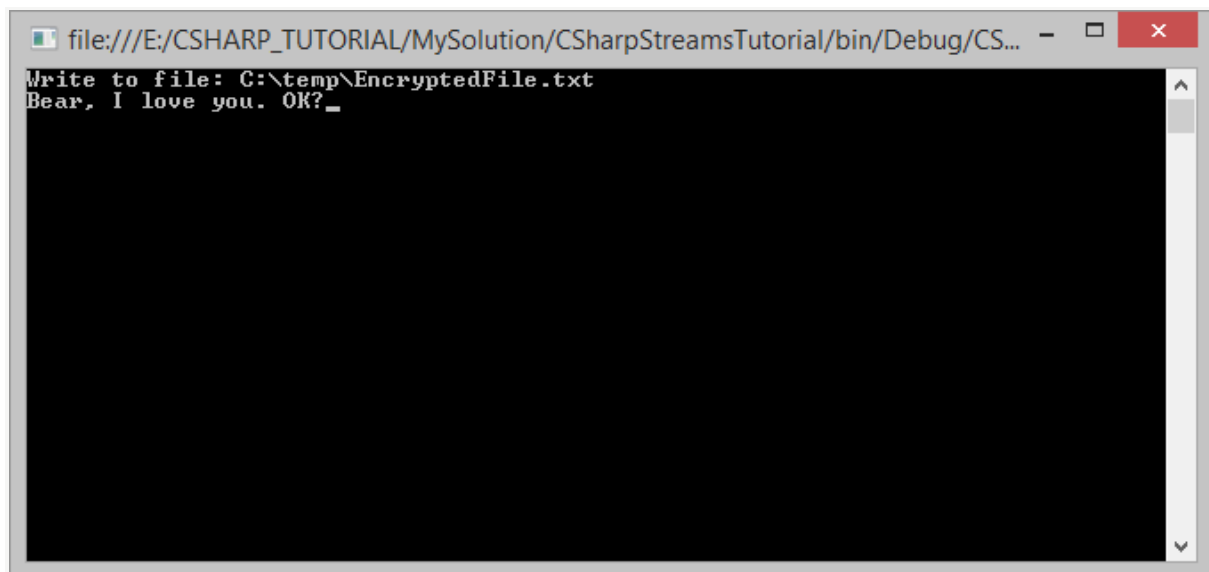
// Một luồng để đọc file
using (FileStream stream = new FileStream(encryedFile,
FileMode.Open, FileAccess.Read))
{
    // Đối tượng giải mật mã hóa
    ICryptoTransform decryptor = provider.CreateDecryptor();

    // Tạo một luồng mật mã hóa bao lấy luồng đọc file.
    using (CryptoStream cryptoStream = new CryptoStream(stream,
        decryptor, CryptoStreamMode.Read))
    {
        byte[] temp = new byte[1024];
        int read=0;
        while((read =cryptoStream.Read(temp,0,temp.Length )) >0
)
        {
            String s= Encoding.UTF8.GetString(temp,0,read);

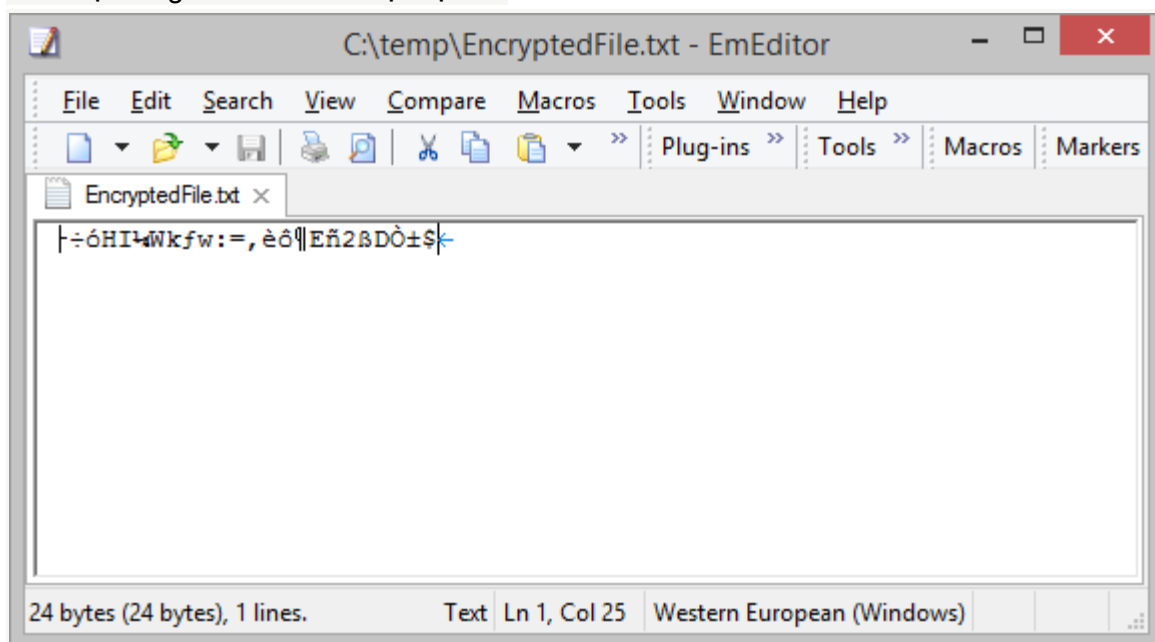
            Console.Write(s);
        }
    }

    // Finished
    Console.Read();
}
}
}

```



Xem nội dung của file vừa được tạo ra.



Biểu thức chính quy trong C#

Một biểu thức chính quy (Regular expressions) định nghĩa một khuôn mẫu (pattern) tìm kiếm chuỗi. Nó có thể được sử dụng tìm kiếm, sửa đổi, và thao tác trên văn bản. Khuôn mẫu được định nghĩa bởi biểu thức chính quy có thể khớp một hoặc một vài lần, hoặc không khớp với một văn bản cho trước.

Viết tắt của biểu thức chính quy là regex

Quy tắc viết biểu thức chính quy

N o	Regular Expressi	Mô tả
--------	---------------------	-------

	on	
1	.	Khớp (match) với một hoặc nhiều ký tự.
2	^regex	Biểu thức chính quy phải khớp tại điểm bắt đầu
3	regex\$	Biểu thức chính quy phải khớp ở cuối dòng.
4	[abc]	Thiết lập định nghĩa, có thể khớp với a hoặc b hoặc c.
5	[abc][vz]	Thiết lập định nghĩa, có thể khớp với a hoặc b hoặc c theo sau là v hoặc z.
6	[^abc]	Khi dấu ^ xuất hiện như là nhân vật đầu tiên trong dấu ngoặc vuông, nó phủ nhận mô hình. Điều này có thể khớp với bất kỳ ký tự nào ngoại trừ a hoặc b hoặc c.
7	[a-d1-7]	Phạm vi: phù hợp với một chuỗi giữa a và điểm d và con số từ 1 đến 7.
8	X Z	Tìm X hoặc Z.
9	XZ	Tìm X và theo sau là Z.
10	\$	Kiểm tra kết thúc dòng.
11	\d	Số bất kỳ, viết ngắn gọn cho [0-9]
12	\D	Ký tự không phải là số, viết ngắn gọn cho [^0-9]
13	\s	Ký tự khoảng trắng, viết ngắn gọn cho [\t\n\r\f]
14	\S	Ký tự không phải khoảng trắng, viết ngắn gọn cho [^\s]
15	\w	Ký tự chữ, viết ngắn gọn cho [a-zA-Z_0-9]
16	\W	Ký tự không phải chữ, viết ngắn gọn cho [^\w]
17	\S+	Một số ký tự không phải khoảng trắng (Một hoặc nhiều)

1 8	<code>\b</code>	Ký tự thuộc a-z hoặc A-Z hoặc 0-9 hoặc <code>_</code> , viết ngắn gọn cho <code>[a-zA-Z0-9_]</code> .
1 9	<code>*</code>	Xuất hiện 0 hoặc nhiều lần, viết ngắn gọn cho <code>{0,}</code>
2 0	<code>+</code>	Xuất hiện 1 hoặc nhiều lần, viết ngắn gọn cho <code>{1,}</code>
2 1	<code>?</code>	Xuất hiện 0 hoặc 1 lần, <code>?</code> viết ngắn gọn cho <code>{0,1}</code> .
2 2	<code>{X}</code>	Xuất hiện X lần, <code>{}</code>
2 3	<code>{X,Y}</code>	Xuất hiện trong khoảng X tới Y lần.
2 4	<code>*?</code>	<code>*</code> có nghĩa là xuất hiện 0 hoặc nhiều lần, thêm <code>?</code> phía sau nghĩa là tìm kiếm khớp nhỏ nhất.

Các ký tự đặc biệt trong C# Regex (Special characters)

Một số ký tự đặc biệt trong **C# Regex**:

```
\. [\{ (*+?^$|
```

Những ký tự liệt kê ở trên là các ký tự đặc biệt. Trong C# Regex bạn muốn nó hiểu các ký tự đó theo cách thông thường bạn cần thêm dấu `\` ở phía trước.

Chẳng hạn ký tự chấm `.` C# regex đang hiểu là một ký tự bất kỳ, nếu bạn muốn nó hiểu là một ký tự chấm thông thường, cần phải có dấu `\` phía trước.

```
// Mẫu regex mô tả một hoặc nhiều ký tự bất kỳ.
```

```
string regex = ".";
```

```
// Mẫu regex mô tả ký tự dấu chấm.
```

```
string regex = "\\.";
```

```
string regex = @"\.";
```

Sử dụng `Regex.IsMatch(string)`

```
// Kiểm tra đối tượng toàn bộ String có khớp với regex hay không.
```

```
public bool IsMatch(string regex)
```

Regex

Trong biểu thức chính quy của C#, ký tự dấu chấm (.) là một ký tự đặc biệt. Nó đại diện cho một hoặc nhiều ký tự bất kỳ. Khi bạn muốn C# hiểu nó là một dấu chấm theo nghĩa thông thường bạn cần viết là "\\." hoặc @"\.";

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

namespace RegularExpressionTutorial
{
    class DotExample
    {
        public static void Main(string[] args)
        {

            // Chuỗi có 0 ký tự (Rỗng)
            string s1 = "";
            Console.WriteLine("s1=" + s1);

            // Kiểm tra chuỗi s1
            // Khớp với 1 hoặc nhiều ký tự
            // Quy tắc .
            // ==> False
            bool match = Regex.IsMatch(s1, ".");
            Console.WriteLine(" -Match . " + match);

            // Chuỗi có 1 ký tự.
            string s2 = "a";
            Console.WriteLine("s2=" + s2);

            // Kiểm tra chuỗi s2
            // Khớp với 1 hoặc nhiều ký tự
            // Quy tắc .
            // ==> True
            match = Regex.IsMatch(s2, ".");
            Console.WriteLine(" -Match . " + match);

            // Chuỗi có 3 ký tự.
            string s3 = "abc";
            Console.WriteLine("s3=" + s3);
```

```

        // Kiểm tra s3
        // Khớp với một hoặc nhiều ký tự.
        // Quy tắc .
        // ==> true
        match = Regex.IsMatch(s3, ".");
        Console.WriteLine(" -Match . " + match);

        // Chuỗi có 3 ký tự.
        string s4 = "abc";
        Console.WriteLine("s4=" + s4);

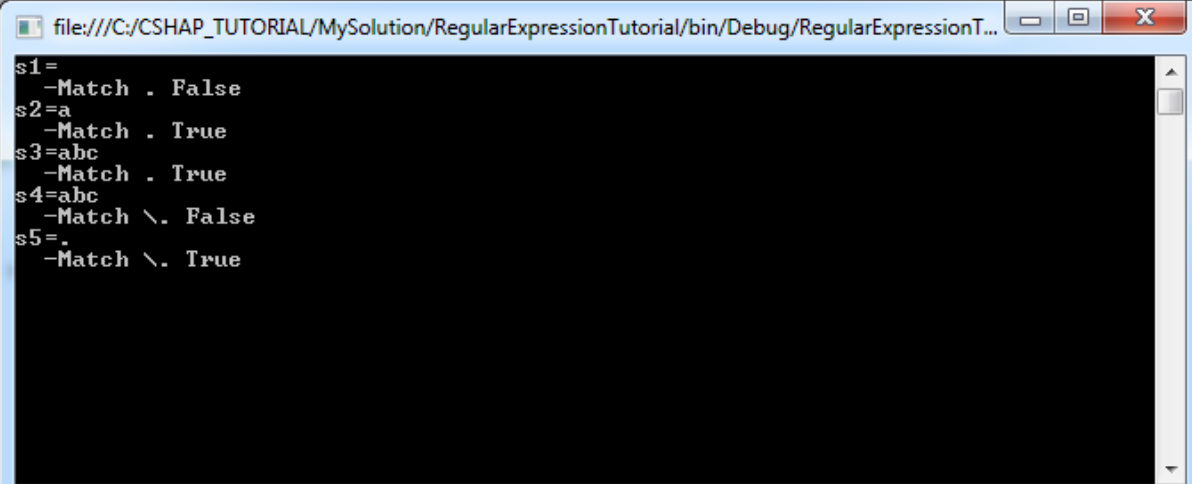
        // Kiểm tra chuỗi s4
        // Khớp với ký tự dấu chấm
        // ==> False
        match = Regex.IsMatch(s4, @"\.");
        Console.WriteLine(" -Match \\. " + match);

        // Chuỗi có 1 ký tự (Dấu chấm).
        string s5 = ".";
        Console.WriteLine("s5=" + s5);

        // Kiểm tra chuỗi s5
        // Khớp với ký tự dấu chấm
        // ==> True
        match = Regex.IsMatch(s5, @"\.");
        Console.WriteLine(" -Match \\. " + match);

        Console.Read();
    }
}

```



```

file:///C:/CSHAP_TUTORIAL/MySolution/RegularExpressionTutorial/bin/Debug/RegularExpressionT...
s1=
-Match . False
s2=a
-Match . True
s3=abc
-Match . True
s4=abc
-Match \. False
s5=.
-Match \. True

```

Một ví dụ khác sử dụng **Regex.IsMatch(string)**:

```
using System;
```

```

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

namespace RegularExpressionTutorial
{
    class RegexIsMatchExample
    {
        public static void Main(string[] args)
        {

            // Chuỗi với 1 ký tự
            string s2 = "m";
            Console.WriteLine("s2=" + s2);

            // Kiểm tra s2
            // Bắt đầu bởi ký tự m
            // Quy tắc ^
            // ==> true
            bool match = Regex.IsMatch(s2, "^m");
            Console.WriteLine("  -Match ^m " + match);

            // Chuỗi có 7 ký tự
            string s3 = "MMnnnnn";
            Console.WriteLine("s3=" + s3);

            // Kiểm tra s3
            // Bắt đầu bởi MM
            // Quy tắc ^
            // ==> true
            match = Regex.IsMatch(s3, "^MM");
            Console.WriteLine("  -Match ^MM " + match);

            // Kiểm tra s3
            // Bắt đầu bởi ký tự MM
            // Tiếp theo là ký tự n xuất hiện một hoặc nhiều lần
            // Quy tắc ^ và +
            match = Regex.IsMatch(s3, "^MMn+");
            Console.WriteLine("  -Match ^MMn+ " + match);

            // Chuỗi với 1 ký tự

```

```

String s4 = "p";
Console.WriteLine("s4=" + s4);

// Kiểm tra s4 kết thúc bởi p
// Quy tắc $
// ==> true
match = Regex.IsMatch(s4, "p$");
Console.WriteLine("  -Match p$ " + match);

// Chuỗi có 6 ký tự
string s5 = "122nnp";
Console.WriteLine("s5=" + s5);

// Kiểm tra s5 kết thúc bởi p
// ==> true
match = Regex.IsMatch(s5, "p$");
Console.WriteLine("  -Match p$ " + match);

// Kiểm tra s5
// Bắt đầu bởi một hoặc nhiều ký tự (Quy tắc .)
// Theo sau là ký tự n, xuất hiện 1 tới 3 lần (Quy tắc n{1,3} )
// Kết thúc bởi ký tự p (Quy tắc $)
// Kết hợp các quy tắc ., {x,y}, $
// ==> true
match = Regex.IsMatch(s5, ".n{1,3}p$");
Console.WriteLine("  -Match .n{1,3}p$ " + match);

String s6 = "2ybcd";
Console.WriteLine("s6=" + s6);

// Kiểm tra s6
// Bắt đầu bởi 2
// Tiếp theo là x hoặc y hoặc z (Quy tắc [xyz])
// Tiếp theo là bất kỳ xuất hiện 0 hoặc nhiều lần (Quy tắc *)
match = Regex.IsMatch(s6, "2[xyz].*");

Console.WriteLine("  -Match 2[xyz].* " + match);

string s7 = "2bkbv";
Console.WriteLine("s7=" + s7);

// Kiểm tra s7, bắt đầu bất kỳ (một hoặc nhiều lần)
// Tiếp theo là a hoặc b hoặc c (Quy tắc [abc] )
// Tiếp theo là z hoặc v (Quy tắc [zv] )
// Cuối cùng là bất kỳ, 0 hoặc nhiều lần (Quy tắc .*)

```



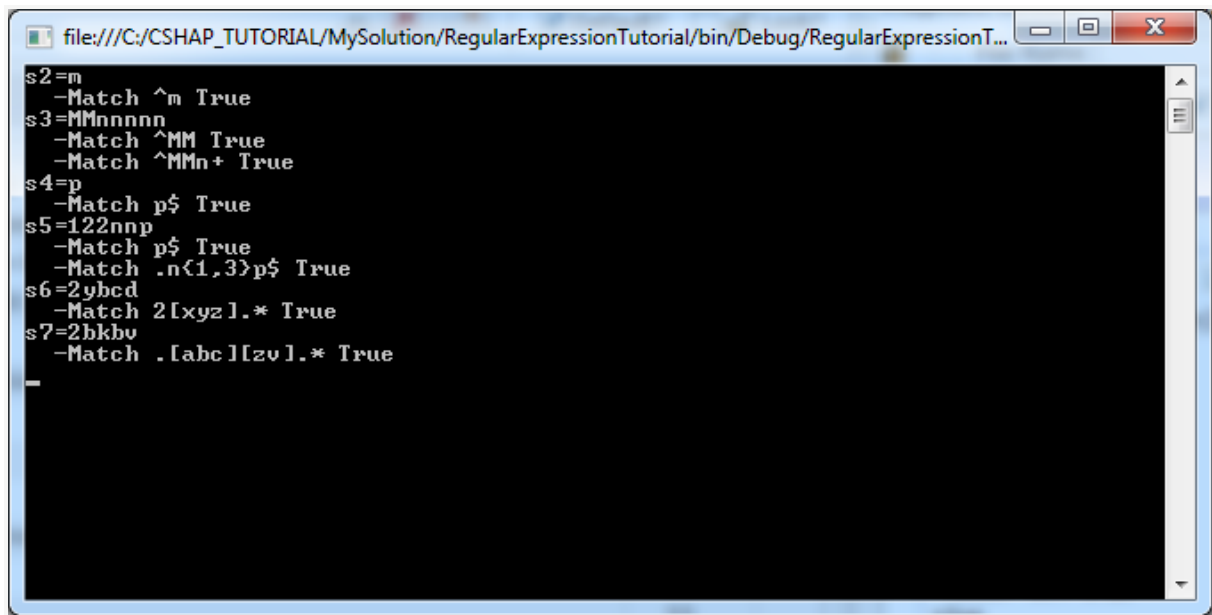
```

        // ==> true
        match = Regex.IsMatch(s7, "[abc][zv].*");

        Console.WriteLine("    -Match [abc][zv].* " + match);

        Console.Read();
    }
}
}

```



```

file:///C:/CSHAP_TUTORIAL/MySolution/RegularExpressionTutorial/bin/Debug/RegularExpressionT...
s2=m
-Match ^m True
s3=MMnnnnn
-Match ^MM True
-Match ^MMn+ True
s4=p
-Match p$ True
s5=122nnp
-Match p$ True
-Match .n{1,3}p$ True
s6=2ybcd
-Match 2[xyz].* True
s7=2hkbv
-Match [abc][zv].* True
-

```

```

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

namespace RegularExpressionTutorial
{
    class RegexIsMatchExample2
    {
        public static void Main(string[] args)
        {
            String s = "The film Tom and Jerry!";

            // Kiểm tra chuỗi s
            // Bắt đầu bởi ký tự bất kỳ, Xuất hiện 0 hoặc nhiều lần (Quy
tắc: .*)
            // Tiếp theo là Tom hoặc Jerry
            // Kết thúc với bất kỳ, xuất hiện 1 hoặc nhiều lần (Quy tắc .)

```

```
// Kết hợp các quy tắc: ., *, X|Z

bool match = Regex.IsMatch(s, ".*(Tom|Jerry).");
Console.WriteLine("s=" + s);
Console.WriteLine("-Match .*(Tom|Jerry). " + match);

s = "The cat";
// ==> false
match = Regex.IsMatch(s, ".*(Tom|Jerry).");
Console.WriteLine("s=" + s);
Console.WriteLine("-Match .*(Tom|Jerry). " + match);

s = "The Tom cat";
// ==> true
match = Regex.IsMatch(s, ".*(Tom|Jerry).");
Console.WriteLine("s=" + s);
Console.WriteLine("-Match .*(Tom|Jerry). " + match);

Console.Read();
}
}
```

```
s=The film Tom and Jerry!
-Match .*(Tom|Jerry). True
s=The cat
-Match .*(Tom|Jerry). False
s=The Tom cat
-Match .*(Tom|Jerry). True
```

Sử dụng Regex.Split & Regex.Replace

Một trong các phương thức hữu ích khác là **Regex.Split(string,string)**, phương thức này phân tách một chuỗi thành các chuỗi con. Chẳng hạn bạn có chuỗi **"One,Two,Three"** và bạn muốn phân tách thành 3 chuỗi con ngăn cách bởi dấu phẩy.

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

```

using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Text.RegularExpressions;

namespace RegularExpressionTutorial
{
    class SplitWithRegexExample
    {
        public static void Main(string[] args)
        {

            // \t: Ký tự TAB
            // \n: Ký tự xuống dòng
            string TEXT = "This \t\t is \n my \t text";

            Console.WriteLine("TEXT=" + TEXT);


            // Định nghĩa một Regex:
            // Khoảng trắng xuất hiện 1 hoặc nhiều lần.
            // Các ký tự khoảng trắng: \t\n\r\b\f
            // Kết hợp quy tắc: \s và +
            String regex = @"\s+";

            Console.WriteLine(" ----- ");

            String[] splitString = Regex.Split(TEXT, regex);

            // 4
            Console.WriteLine(splitString.Length);

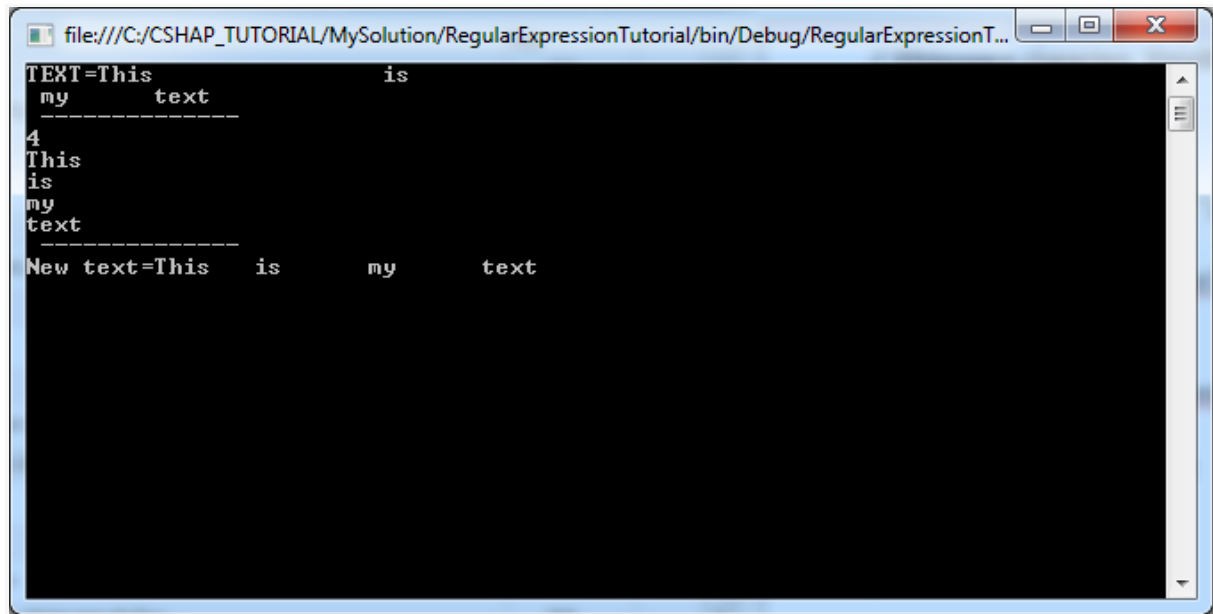
            foreach (string str in splitString)
            {
                Console.WriteLine(str);
            }

            Console.WriteLine(" ----- ");

            // Thay thế tất cả các khoảng trắng với ký tự tab.
            String newText = Regex.Replace(TEXT, @"\s+", "\t");
            Console.WriteLine("New text=" + newText);


            Console.Read();
        }
    }
}

```



Sử dụng MatchCollection & Match

Sử dụng **Regex.Matches(...)** để tìm kiếm tất cả các chuỗi con của một chuỗi, phù hợp với một biểu thức chính quy, phương thức trả về một đối tượng **MatchCollection**.

```
public MatchCollection Matches(  
    string input  
)
```

```
public MatchCollection Matches(  
    string input,  
    int startat  
)
```

```
public static MatchCollection Matches(  
    string input,  
    string pattern  
)
```

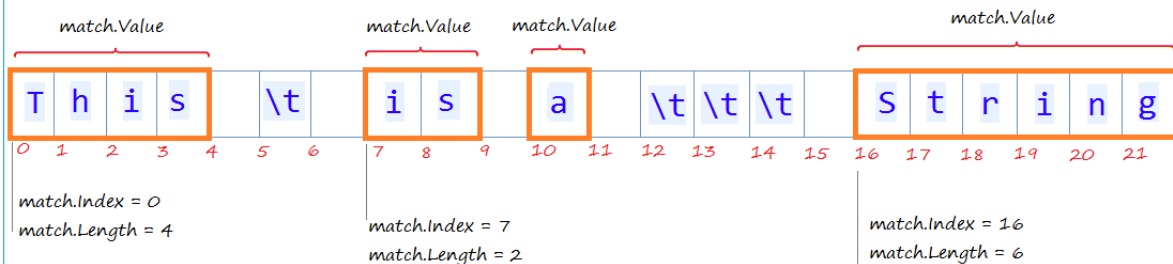
```
public static MatchCollection Matches(  
    string input,  
    string pattern,  
    RegexOptions options,  
    TimeSpan matchTimeout  
)
```

```
public static MatchCollection Matches(  
    string input,  
    string pattern,  
    RegexOptions options
```

)

Ví dụ dưới đây tách một chuỗi thành các chuỗi con, phân cách bởi các khoảng trắng (whitespace).

```
string TEXT = "This \t is a \t\t\t String";
string regex = @"\w+";
MatchCollection matchColl = Regex.Matches(TEXT, regex);
foreach (Match match in matchColl)
{
    Console.WriteLine(" ----- ");
    Console.WriteLine("Value: " + match.Value);
    Console.WriteLine("Index: " + match.Index);
    Console.WriteLine("Length: " + match.Length);
}
```



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

namespace RegularExpressionTutorial
{
    class MatchCollectionExample
    {
        public static void Main(string[] args)
        {
            string TEXT = "This \t is a \t\t\t String";

            // \w : Ký tự chữ, viết ngắn gọn cho [a-zA-Z_0-9]
            // \w+ : Ký tự chữ, xuất hiện một hoặc nhiều lần.
            string regex = @"\w+";

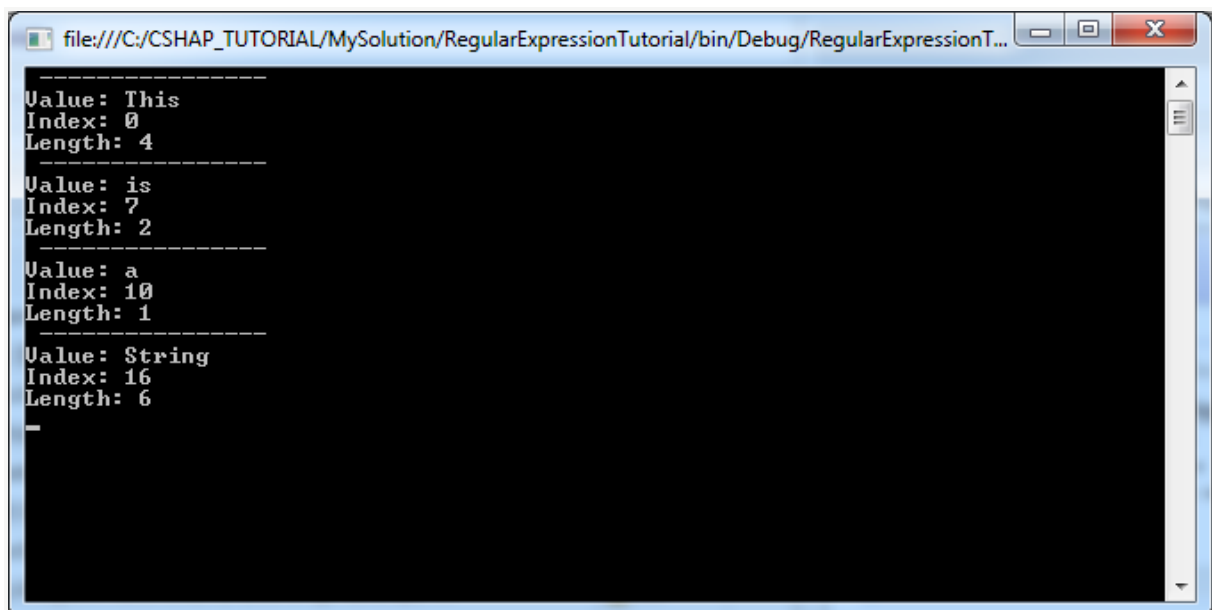
            MatchCollection matchColl = Regex.Matches(TEXT, regex);
```

```

foreach (Match match in matchColl)
{
    Console.WriteLine(" ----- ");
    Console.WriteLine("Value: " + match.Value);
    Console.WriteLine("Index: " + match.Index);
    Console.WriteLine("Length: " + match.Length);
}

Console.Read();
}
}
}

```



Nhóm (Group)

Một biểu thức chính quy bạn có thể tách ra thành các nhóm (group):

```
// Một biểu thức chính quy
string regex = @"\s+=\d+";
```

```
// Viết dưới dạng group, bởi dấu ()
string regex2 = @"(\s+) (=) (\d+)";
```

```
// Một cách khác.
string regex3 = @"(\s+) (= \d+)";
```

Các group có thể lồng nhau, và như vậy cần một quy tắc đánh chỉ số các group. Toàn bộ pattern được định nghĩa là group số 0. Còn lại được mô tả giống hình minh họa dưới đây:

The diagram shows a regular expression pattern: `(group)(?:non-capturing-group)(g(?:ro|u)p((nested)inside)(another)group)(?=assertion)`. Below the pattern, vertical lines indicate the positions of various components, numbered 1 through 5. Line 1 points to the opening parenthesis of the first group. Line 2 points to the closing parenthesis of the first group. Line 3 points to the opening parenthesis of the second group. Line 4 points to the closing parenthesis of the second group. Line 5 points to the opening parenthesis of the third group. Dashed lines connect these numbers to their corresponding components in the pattern.

Chú ý: Sử dụng `(?:pattern)` để thông báo với C# không xem đây là một group (None-capturing group)

Bạn có thể xác định một group có tên (`?<name>pattern`) hoặc (`? 'name' pattern`), Và bạn có thể truy cập các nội dung khớp với `match.Groups["groupName"]`. Điều này làm Regex dài hơn, nhưng mã này là có ý nghĩa hơn, dễ hơn.

Nhóm bắt theo tên cũng có thể được truy cập thông qua `match.Groups[groupIndex]` với các đề án đánh số tương tự.

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

namespace RegularExpressionTutorial
{
    class NamedGroupExample
    {
        public static void Main(string[] args)
        {
            string TEXT = " int a = 100; float b= 130; float c= 110 ; ";

            // Sử dụng (?<groupName>pattern) để định nghĩa một Group có tên:
            // groupName
            // Định nghĩa group có tên declare: sử dụng (?<declare> ...)
            // Và một group có tên value: sử dụng: (?<value> ..)
            string regex =
@"(?<declare>\s*(int|float)\s+[a-z]\s*)=(?<value>\s*\d+\s*)";

            MatchCollection matchCollection = Regex.Matches(TEXT, regex);

            foreach (Match match in matchCollection)
            {
                string group = match.Groups["declare"].Value;
                Console.WriteLine("Full Text: " + match.Value);
                Console.WriteLine("<declare>: " +
match.Groups["declare"].Value);
                Console.WriteLine("<value>: " + match.Groups["value"].Value);
                Console.WriteLine("-----");
            }
        }
    }
}
```

```

        Console.Read();
    }
}

```

```

Full Text:  int a = 100;
<declare>:  int a
<value>:    100
-----
Full Text:  float b= 130;
<declare>:  float b
<value>:    130
-----
Full Text:  float c= 110 ;
<declare>:  float c
<value>:    110
-----

```

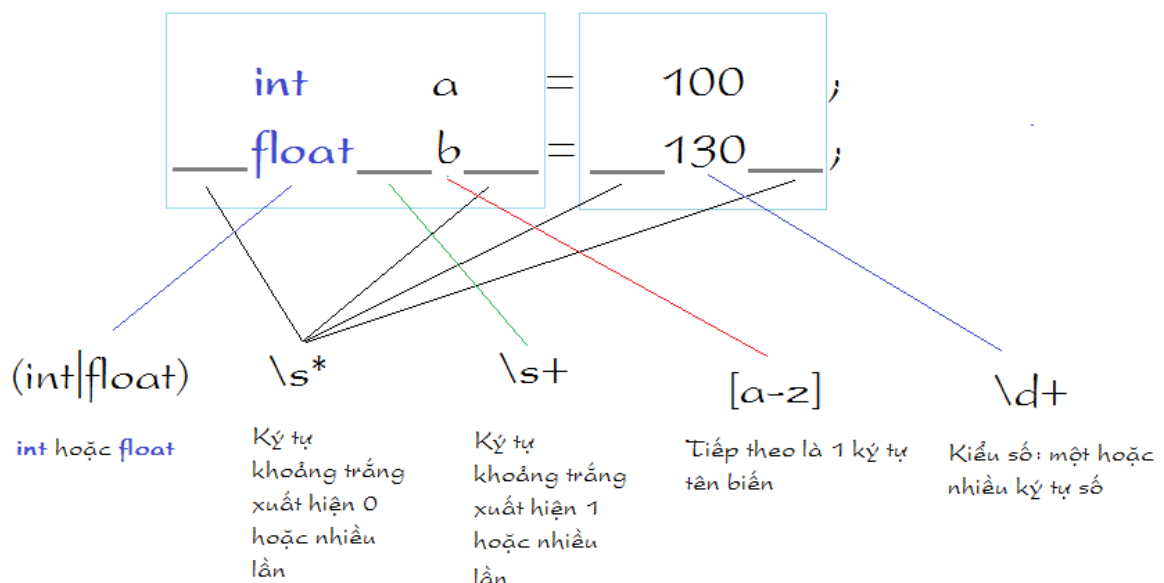
String regex = @"(?<declare>\s*(int|float)\s+[a-z]\s*)=(?<value>\s*\d+\s*)";

Group "declare"

Group "value"

(?<declare>\s*(int|float)\s+[a-z]\s*)

(?<value>\s*\d+\s*)



Sử dụng MatchCollection, Group và *?

Trong một số tình huống `*?` rất quan trọng, hãy xem một ví dụ sau:

```
// Đây là một regex
// Bắt gặp ký tự bất kỳ 0 hoặc nhiều lần,
// sau đó tới ký tự ' và tiếp theo là >
string regex = ".*'>";

// Đoạn TEXT1 sau đây có vẻ hợp với regex nói trên.
string TEXT1 = "FILE1'>";

// Đoạn TEXT2 sau cũng hợp với regex nói trên.
string TEXT2 = "FILE1'> <a href='http://HOST/file/FILE2'>";
```

String regex = ".*'>";

String TEXT1 = "FILE1'>";

String TEXT2 = "FILE1'> ";

`*?` sẽ tìm ra một phù hợp nhỏ nhất. Chúng ta xem ví dụ sau:

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

namespace RegularExpressionTutorial
{
    class NamedGroupExample2
    {
        public static void Main(string[] args)
        {
            string TEXT = "<a href='http://HOST/file/FILE1'>File 1</a>"
                + "<a href='http://HOST/file/FILE2'>File 2</a>";

            // Define group named fileName.
            // *? ==> ? after a quantifier makes it a reluctant quantifier.
            // It tries to find the smallest match.
            string regex = "/file/(?<fileName>.*?)'>";
```

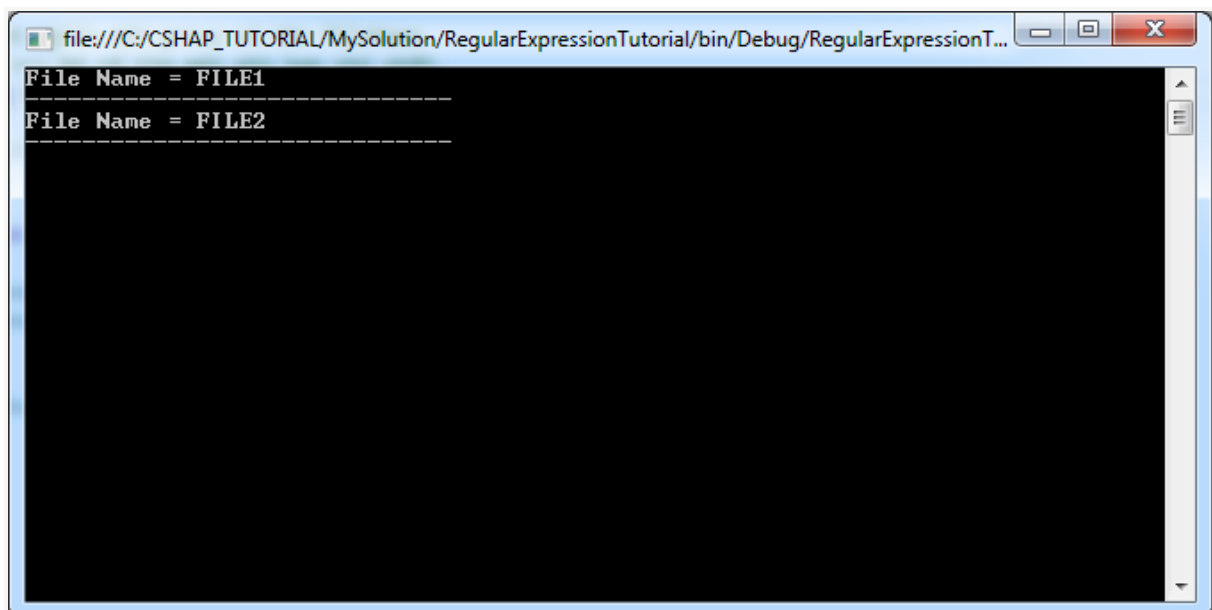
```

        MatchCollection matchCollection = Regex.Matches(TEXT, regex);

        foreach (Match match in matchCollection)
        {
            Console.WriteLine("File Name = " +
match.Groups["fileName"].Value);
            Console.WriteLine("-----");
        }

        Console.Read();
    }
}

```



C# Delegate và Event

Delegate là gì?

Trong C# mỗi một hàm (phương thức, hoặc constructor) đều có một kiểu hàm. Hãy xem phương thức *SayHello* dưới đây:

Đây là một phương thức

```
public string SayHello(string firstName, string lastName)
{
    return "Hello " + firstName + " " + lastName;
}
```



Kiểu hàm

$(string, string) \rightarrow (string)$

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

namespace CSharpDelegatesTutorial
{
    class HelloProgram
    {
        // Đây là phương thức có một tham số string và trả về string
        // Kiểu hàm: (string) -> (string)
        public string SayHello(string name)
        {
            return "Hello " + name;
        }

        // Đây là một phương thức có 2 tham số string, và trả về string
        // Kiểu hàm: (string, string) -> (string)
        public string SayHello(string firstName, string lastName)
        {
            return "Hello " + firstName + " " + lastName;
        }

        // Đây là một phương thức có 1 tham số, và không trả về gì.
    }
}
```

```

        // Kiểu hàm: (string) -> ()
        public void Silent(string name)
        {

        }

    }
}

```

Hai phương thức dưới đây có cùng một kiểu hàm:

(int, int) -> (int)

```

public static int sum(int a, int b)
{
    return a + b;
}

```

```

public int max(int a, int b)
{
    if (a > b)
    {
        return a;
    }
    return b;
}

```

C# sử dụng từ khóa ***delegate*** (Người đại diện) để định nghĩa ra một thực thể đại diện cho các hàm (phương thức, hoặc constructor) có cùng một kiểu hàm.

// Cú pháp định nghĩa một delegate:

```

delegate <return_type> <delegate_name> <parameter_list>
// Định nghĩa một kiểu đại diện cho các hàm có kiểu
// (string,string) -> (string).

```

```

private delegate string MyDelegate(string s1, string s2);
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

```

```

namespace CSharpDelegatesTutorial
{
    class MathUtils
    {

```

```

        // (int, int) -> (int)
        public static int sum(int a, int b)
        {
            return a + b;
        }

        // (int, int) -> (int)
        public static int minus(int a, int b)
        {
            return a - b;
        }

        // (int, int) -> (int)
        public static int multiple(int a, int b)
        {
            return a * b;
        }
    }
}

```

Ví dụ dưới đây, định nghĩa ra một delegate **IntIntToInt** đại diện cho các hàm có kiểu hàm là **(int, int) -> (int)**.

```

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

namespace CSharpDelegatesTutorial
{
    class MyFirstDelegate
    {
        // Định nghĩa ra một delegate
        // đại diện cho các hàm có kiểu (int, int) -> (int)
        delegate int IntIntToInt(int a, int b);

        public static void Main(string[] args)
        {
            // Tạo một đối tượng delegate.
            // Truyền vào tham số là một hàm có cùng kiểu hàm với delegate.
            IntIntToInt iiToInt = new IntIntToInt(MathUtils.sum);
        }
    }
}

```

```

// Khi bạn thực thi một delegate.
// Nó sẽ gọi hàm (hoặc phương thức) mà nó đại diện.
int value = iiToInt(10, 20); // 30

Console.WriteLine("Value = {0}", value);

// Gán delegate đại diện cho phương thức khác.
iiToInt = new IntIntToInt(MathUtils.multiply);

value = iiToInt(10, 20); // 200

Console.WriteLine("Value = {0}", value);

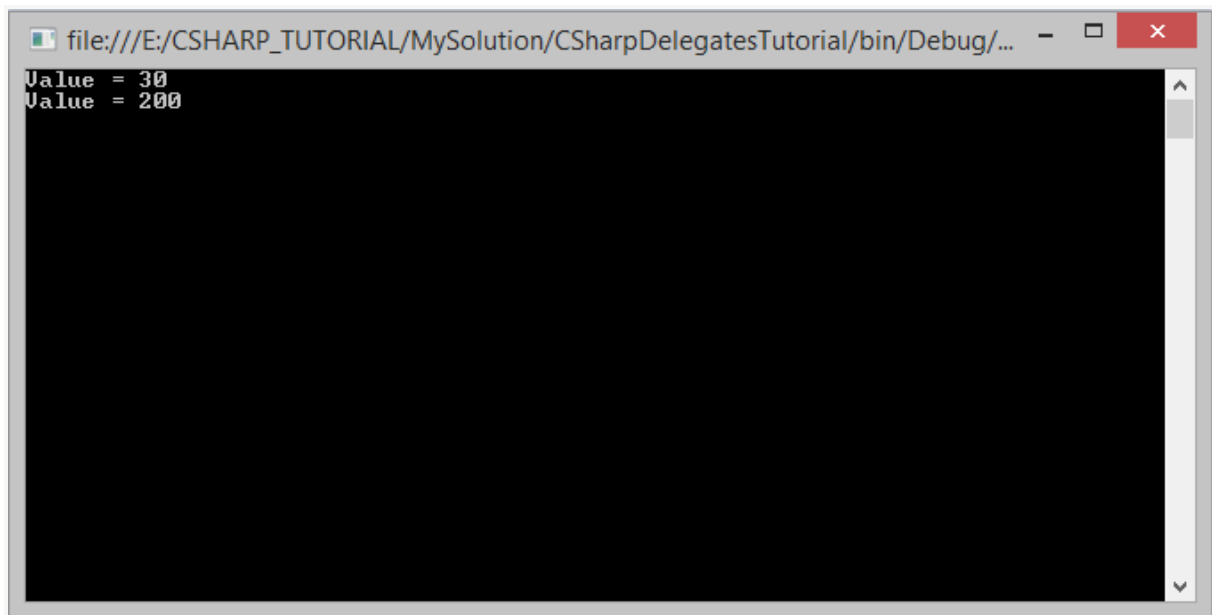
Console.Read();

}

}

}

```



Bạn cũng có thể tạo đối tượng **delegate** đại diện cho các hàm không tĩnh (none static). Xem ví dụ:

```

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

```

```

namespace Hello_CSharp_Demo01.DelegateAndEvent
{
    public class StringDelegate
    {
        public string SayHello(string name)
        {
            return "Hello" + name;
        }
        public string SayHello(string FirstName, string LastName)
        {
            return "Hello" + FirstName + LastName;
        }
        public void Slient(string name)
        {

        }
    }
}

```

```

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

```

```

namespace Hello_CSharp_Demo01.DelegateAndEvent
{
    class StringDelegateDemo
    {
        private delegate string DelegateString(string s);
        public static void Main(string[] args)
        {
            StringDelegate hello = new StringDelegate();
            DelegateString deStr = new DelegateString(hello.SayHello);
            string name = deStr("Phan Huu Hoang");
            Console.WriteLine(name);
            Console.Read();
        }
    }
}

```

Hàm trả về một hàm

Trong C#, với Delegate bạn có thể tạo ra một hàm trả về một hàm (Thực tế là một hàm trả về một Delegate).

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

namespace CSharpDelegatesTutorial
{
    class TaxFormulas
    {
        // Một Delegate đại diện cho các hàm kiểu (float) -> (float).
        public delegate float TaxFormula(float salary);

        // Công thức tính thuế của Mỹ (10% lương).
        public static float UsaFormula(float salary)
        {
            return 10 * salary / 100;
        }

        // Công thức tính thuế của Vietnam (5% lương).
        public static float VietnamFormula(float salary)
        {
            return 5 * salary / 100;
        }

        // Công thức tính thuế khác (7% lương).
        public static float DefaultFormula(float salary)
        {
            return 7 * salary / 100;
        }

        // Trả về một hàm tính lương dựa trên mã Quốc gia. (VN, USA, ..)
        public static TaxFormula GetSalaryFormula(string countryCode)
        {
            if (countryCode == "VN")
            {
                return TaxFormulas.VietnamFormula;
            }
            else if (countryCode == "USA")
            {
                return TaxFormulas.UsaFormula;
            }
            return TaxFormulas.DefaultFormula;
        }
    }
}
```



```

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

namespace CSharpDelegatesTutorial
{
    class TaxFormulaTest
    {

        public static void Main(string[] args)
        {
            float salary = 1000f;

            // Công thức tính thuế theo quốc gia Việt Nam.
            TaxFormulas.TaxFormula formula =
TaxFormulas.GetSalaryFormula("VN");

            float tax = formula(salary);

            Console.WriteLine("Tax in Vietnam = {0}", tax);

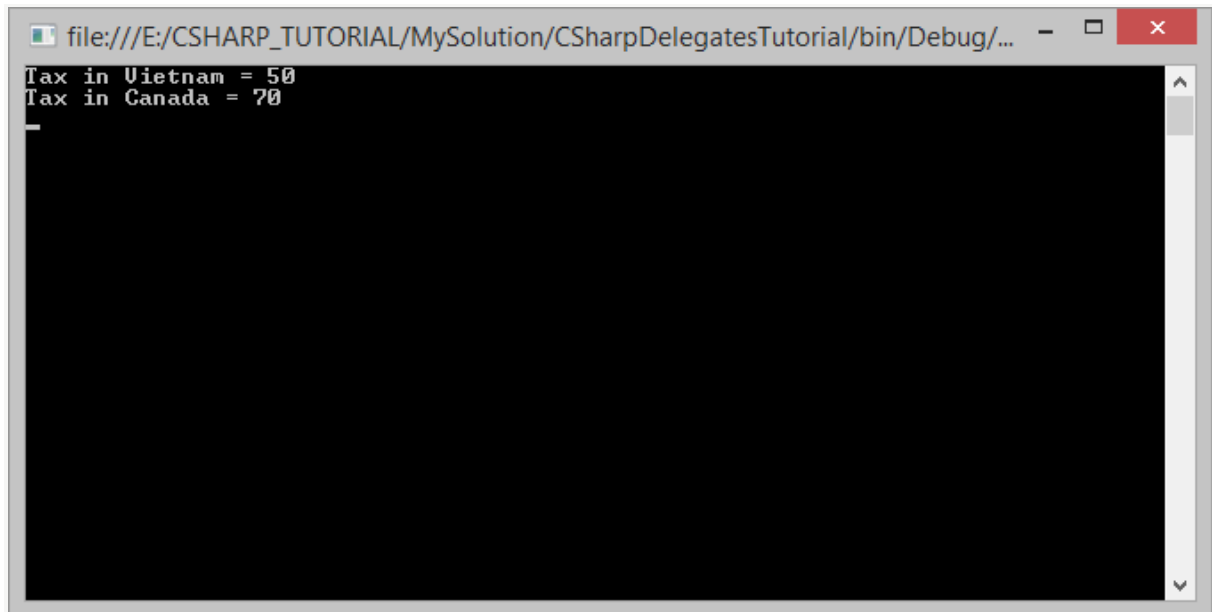
            // Công thức tính thuế tại Canada
            formula = TaxFormulas.GetSalaryFormula("CA");

            tax = formula(salary);

            Console.WriteLine("Tax in Canada = {0}", tax);

            Console.Read();
        }
    }
}

```



```
file:///E:/CSHARP_TUTORIAL/MySolution/CSharpDelegatesTutorial/bin/Debug/...
Tax in Vietnam = 50
Tax in Canada = 70
```

Phương thức nặc danh

Dựa vào Delegate bạn có thể tạo ra một phương thức nặc danh (Anonymous method), nó là một phương thức không có tên, chỉ có thân (body) của phương thức, nó là một khối lệnh (block) có 0 hoặc nhiều tham số, và có thể có kiểu trả về.

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

namespace CSharpDelegatesTutorial
{
    class AnonymousMethod
    {

        // Một Delegate đại diện cho các hàm kiểu: (float) -> (float).
        // Tính thuế dựa trên lương.
        public delegate float TaxFormula(float salary);

        public static TaxFormula GetTaxFormula(string countryCode)
        {
            if ("USA" == countryCode)
            {
                TaxFormula usaFormula = delegate(float salary)
                {
                    return 10 * salary / 100;
                };
                return usaFormula;
            }
        }
    }
}
```

```

    }
    else if ("VN" == countryCode)
    {
        TaxFormula vnFormula = delegate(float salary)
        {
            return 5 * salary / 100;
        };
        return vnFormula;
    }
    return delegate(float salary)
    {
        return 7 * salary / 100;
    };
}

public static void Main(string[] args)
{
    string countryCode = "VN";
    float salary = 1000;

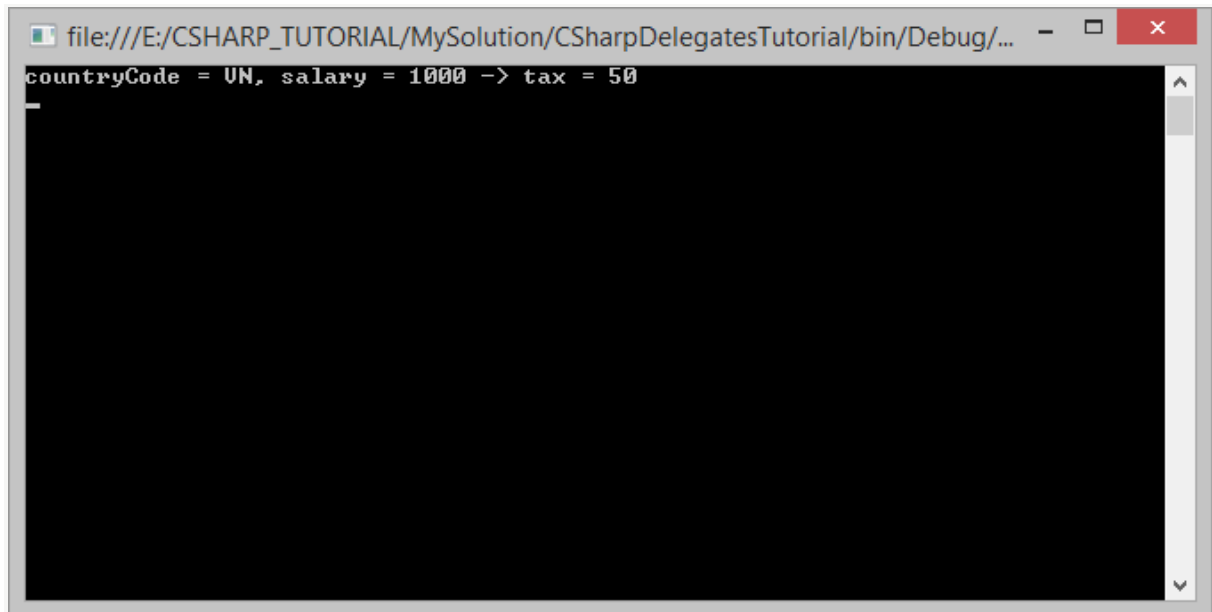
    TaxFormula formula = GetTaxFormula(countryCode);

    float tax = formula(salary);

    Console.WriteLine("countryCode = {0}, salary = {1} -> tax =
{2}"
                      , countryCode, salary, tax);

    Console.Read();
}
}
}

```



Multicasting của một Delegate

C# cho phép bạn cộng (+) hai đối tượng Delegate với nhau để tạo thành một đối tượng Delegate mới. Chú ý rằng các đối tượng Delegate có thể cộng được phải có cùng kiểu hàm, và là hàm không có kiểu trả về. Khi đối tượng delegate mới được gọi, tất cả các delegate con cũng sẽ được thực thi. Khái niệm này được gọi là **Multicasting** của delegate.

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

namespace CSharpDelegatesTutorial
{
    class Greetings
    {
        // Kiểu hàm: (String) -> ()
        public static void Hello(String name)
        {
            Console.WriteLine("Hello " + name);
        }

        // Kiểu hàm: (String) -> ()
        public static void Bye(string name)
        {
            Console.WriteLine("Bye " + name);
        }
    }
}
```

```

        // Kiểu hàm: (String) -> ()
        public static void Hi(string name)
        {
            Console.WriteLine("Hi " + name);
        }
    }

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

namespace CSharpDelegatesTutorial
{
    class MulticastingTest
    {
        // Khai báo một kiểu Delegate

        public delegate void Greeting(string name);

        public static void Main(string[] args)
        {
            // Tạo các đối tượng Delegate.
            Greeting hello = new Greeting(Greetings.Hello);
            Greeting bye = new Greeting(Greetings.Bye);
            Greeting hi = new Greeting(Greetings.Hi);

            // Tạo một Delegate là hợp của 3 đối tượng trên.

            Greeting greeting = hello + bye;

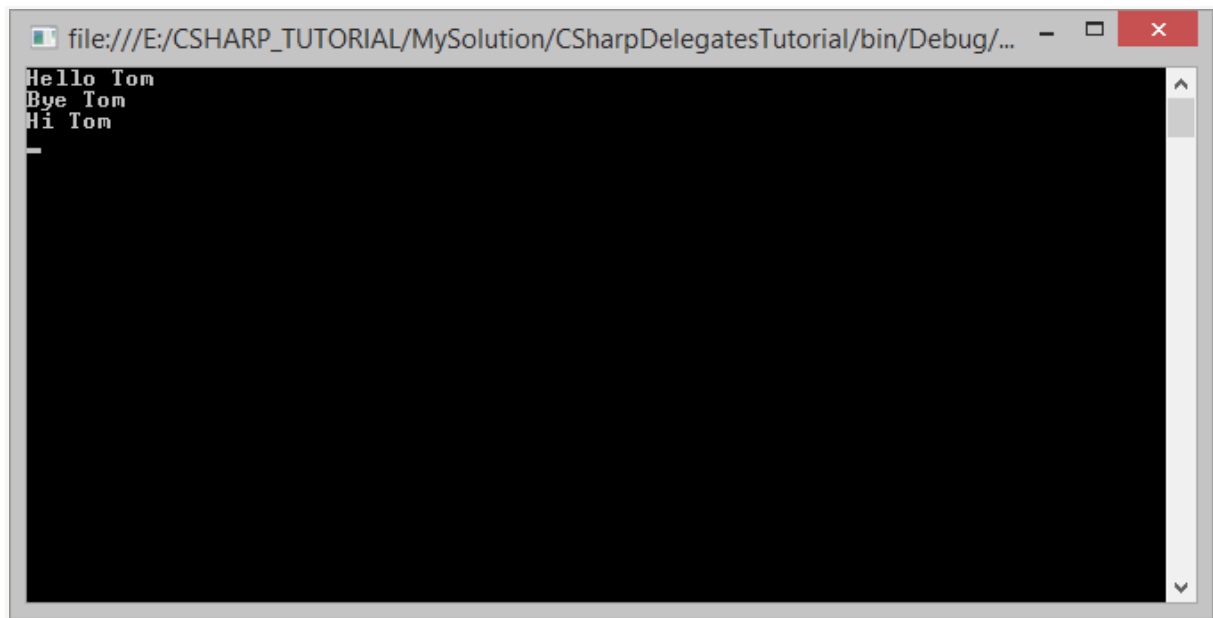
            // Bạn cũng có thể sử dụng toán tử +=
            greeting += hi;

            // Gọi thực thi greeting
            greeting("Tom");

            Console.Read();
        }
    }
}

```

```
}
```



Event

Trong C#, Event là một đối tượng đặc biệt của Delegate, nó là nơi chứa các phương thức, các phương thức này sẽ được thực thi đồng loạt khi sự kiện xảy ra. Bạn có thể thêm các phương thức sẽ được thực thi vào đối tượng Event của đối tượng phát ra sự kiện.

Lớp **Button** mô phỏng một nút, nó định nghĩa ra một Event (sự kiện) để thông báo rằng nó vừa bị nhấn (Click). Khi button bị nhấn, event sẽ được thực thi. Bạn cần thêm các phương thức cần thực thi vào cho đối tượng Event từ bên ngoài.

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

namespace CSharpEventsTutorial
{
    class Button
    {

        private string label;

        public delegate void ClickHandler(Button source, int x, int y);

        // Định nghĩa ra sự kiện, nó chưa được gán giá trị.
        // Giá trị của nó được gán ở bên ngoài.
        public event ClickHandler OnButtonClick;

        public Button(string label)
```

```

{
    this.label = label;
}

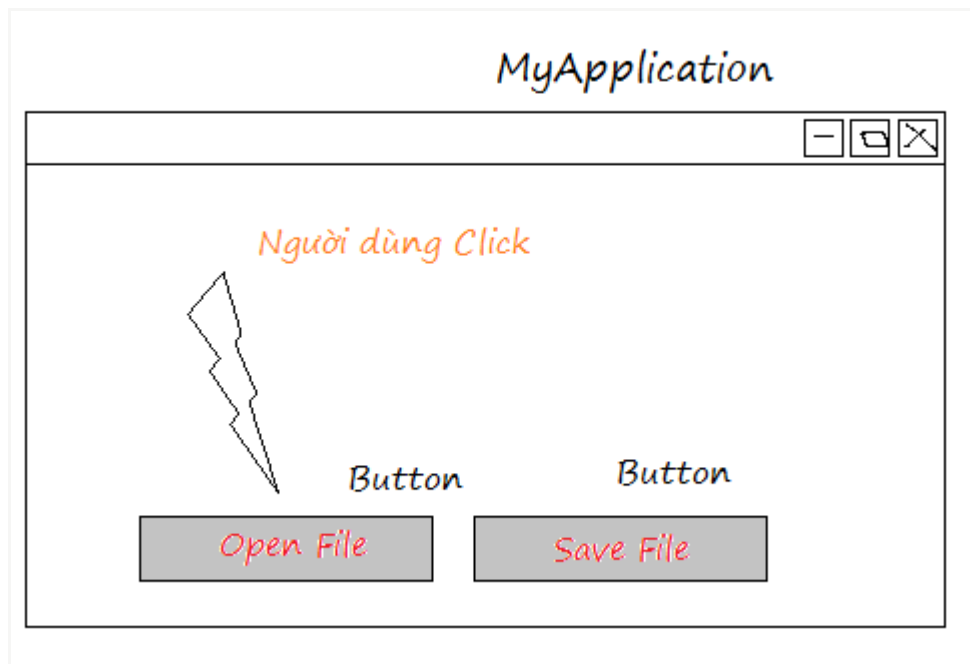
// Mô phỏng Button này vừa bị Click.
// Xác định vị trí x, y mà người dùng Click vào.
public void Clicked()
{
    Random random = new Random();

    // Một số ngẫu nhiên từ 1 -> 100
    int x = random.Next(1, 100);

    // Một số ngẫu nhiên từ 1 -> 20
    int y = random.Next(1, 20);

    // Thông báo sự kiện Button vừa bị Click.
    if (OnClick != null)
    {
        OnClick(this, x, y);
    }
}
}
}

```



Lớp **MyApplication** mô phỏng một ứng dụng có 2 nút, "Open File" và "Save File".

Bạn cần viết phương thức để làm gì đó khi người dùng click vào **"Open File"**, thêm phương thức này vào cho sự kiện **OnButtonClick** của nút **openButton**

Bạn cần viết phương thức để làm gì đó khi người dùng click vào **"Save File"**, và thêm phương thức này vào cho sự kiện **OnButtonClick** của nút **saveButton**.

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

namespace CSharpEventsTutorial
{
    class MyApplication
    {
        private Button openButton;
        private Button saveButton;
        private string fileName;

        // Mô phỏng một ứng dụng và có các Button.
        public MyApplication()
        {
            // Mô phỏng thêm 1 Button vào giao diện.
            this.openButton = new Button("Open File");

            // Mô phỏng thêm 1 Button vào giao diện.
            this.saveButton = new Button("Save File");

            // Khai báo xử lý khi có sự kiện 'Open Button' bị Click.
            // (Tính năng Multicasting của Delegate)
            this.openButton.OnButtonClick += this.OpenButtonClicked;

            // Khai báo xử lý khi sự kiện 'Save Button' bị Click.
            // (Tính năng Multicasting của Delegate)
            this.saveButton.OnButtonClick += this.SaveButtonClicked;
        }

        private void OpenButtonClicked(Button source, int x, int y)
        {
            // Mô phỏng mở ra một cửa sổ để chọn File để mở.
            Console.WriteLine("Open Dialog to Select a file");
            //
            this.fileName = "File" + x + "_" + y + ".txt";
            Console.WriteLine("Opening file: " + this.fileName);
        }

        private void SaveButtonClicked(Button source, int x, int y)
```



```

    {
        if(this.fileName== null)  {
            Console.WriteLine("No file to save!");
            return;
        }
        // Mô phỏng file đã được Save.
        Console.WriteLine("Saved file: " + this.fileName);
    }

public static void Main(string[] args)
{
    // Mô phỏng mở ứng dụng
    MyApplication myApp = new MyApplication();

    Console.WriteLine("User Click on Open Button ....");

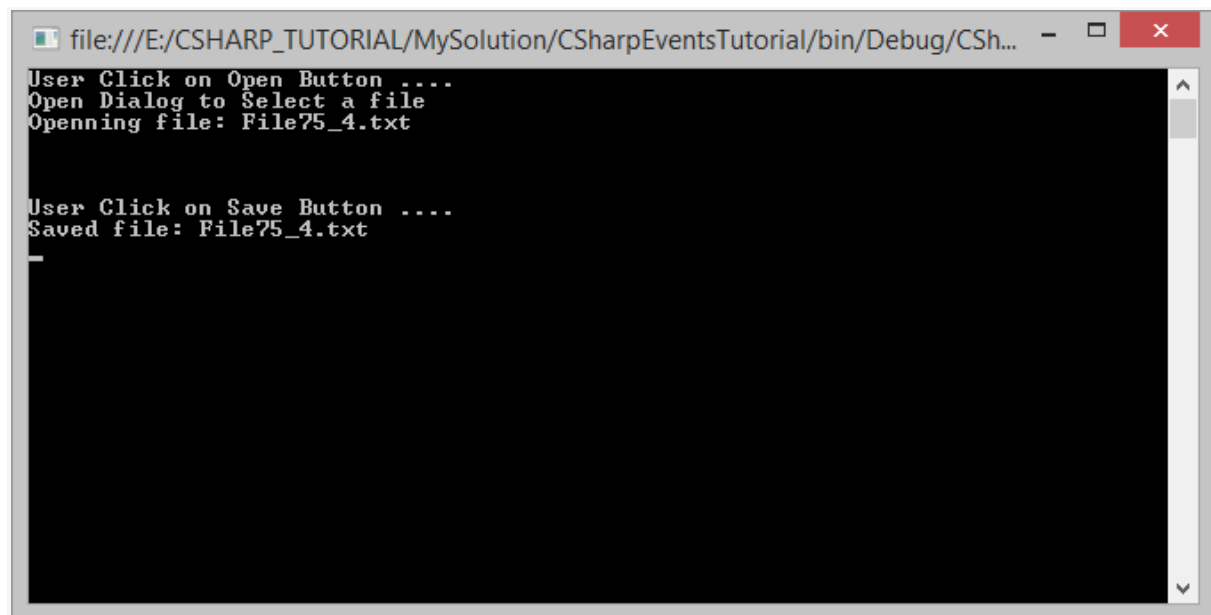
    // Mô phỏng openButton bị click
    myApp.openButton.Clicked();

    Console.WriteLine("\n\n");
    Console.WriteLine("User Click on Save Button ....");

    // Mô phỏng saveButton bị click
    myApp.saveButton.Clicked();

    Console.Read();
}
}
}

```



The image shows a screenshot of a Windows console window. The title bar at the top indicates the file path: `file:///E:/CSHARP_TUTORIAL/MySolution/CSharpEventsTutorial/bin/Debug/CSH...`. The console output, displayed in a monospaced font, shows the following sequence of events:

```
User Click on Open Button ....  
Open Dialog to Select a file  
Opening file: File75_4.txt  
  
User Click on Save Button ....  
Saved file: File75_4.txt
```

A vertical scrollbar is visible on the right side of the console area, and a small horizontal line is present below the 'Saved file' message.