

TÀI LIỆU LẬP TRÌNH C# VÀ CẤU HÌNH VISUAL STUDIO 2013

Bắt đầu với C# cần những gì?

Hướng dẫn cài đặt và cấu hình Visual Studio 2013

Học nhanh C# cho người mới bắt đầu

- 1- Giới thiệu
- 2- Tạo Project C# đầu tiên của bạn
- 3- Giải thích cấu trúc của một class
- 4- Giải thích cấu trúc Project
- 5- Chú ý quan trọng với một chương trình C#
- 6- Thêm mới class
- 7- Các kiểu dữ liệu trong C#
- 8- Biến và khai báo
- 9- Câu lệnh rẽ nhánh
 - 9.1- Câu lệnh If-else
 - 9.2- Câu lệnh Switch-Case
- 10- Vòng lặp trong C#
 - 10.1- Vòng lặp for
 - 10.2- Vòng lặp while
 - 10.3- Vòng lặp do-while
 - 10.4- Lệnh break trong vòng lặp
 - 10.5- Lệnh continue trong vòng lặp
- 11- Mảng trong C#
 - 11.1- Mảng một chiều
 - 11.2- Mảng hai chiều
 - 11.3- Mảng của mảng
- 12- Class, đối tượng và cấu tử
- 13- Trường (Field)
- 14- Phương thức (Method)
- 15- Thừa kế trong C#
- 16- Thừa kế và đa hình trong C#

Thừa kế và đa hình trong C#

Abstract class và Interface trong C#

Access Modifier trong C#

Hướng dẫn sử dụng C# String và StringBuilder

Hướng dẫn sử dụng C# Generics

Hướng dẫn sử lý ngoại lệ trong C#

Hướng dẫn sử dụng Date Time trong C#

Thao tác với tập tin và thư mục trong C#

Nén và giải nén trong C#

Hướng dẫn sử dụng Stream - luồng vào ra nhị phân trong C#

Kết nối Database SQL Server sử dụng C#

Hướng dẫn làm việc với Database SQL Server sử dụng C#

Kết nối Database MySQL sử dụng C#

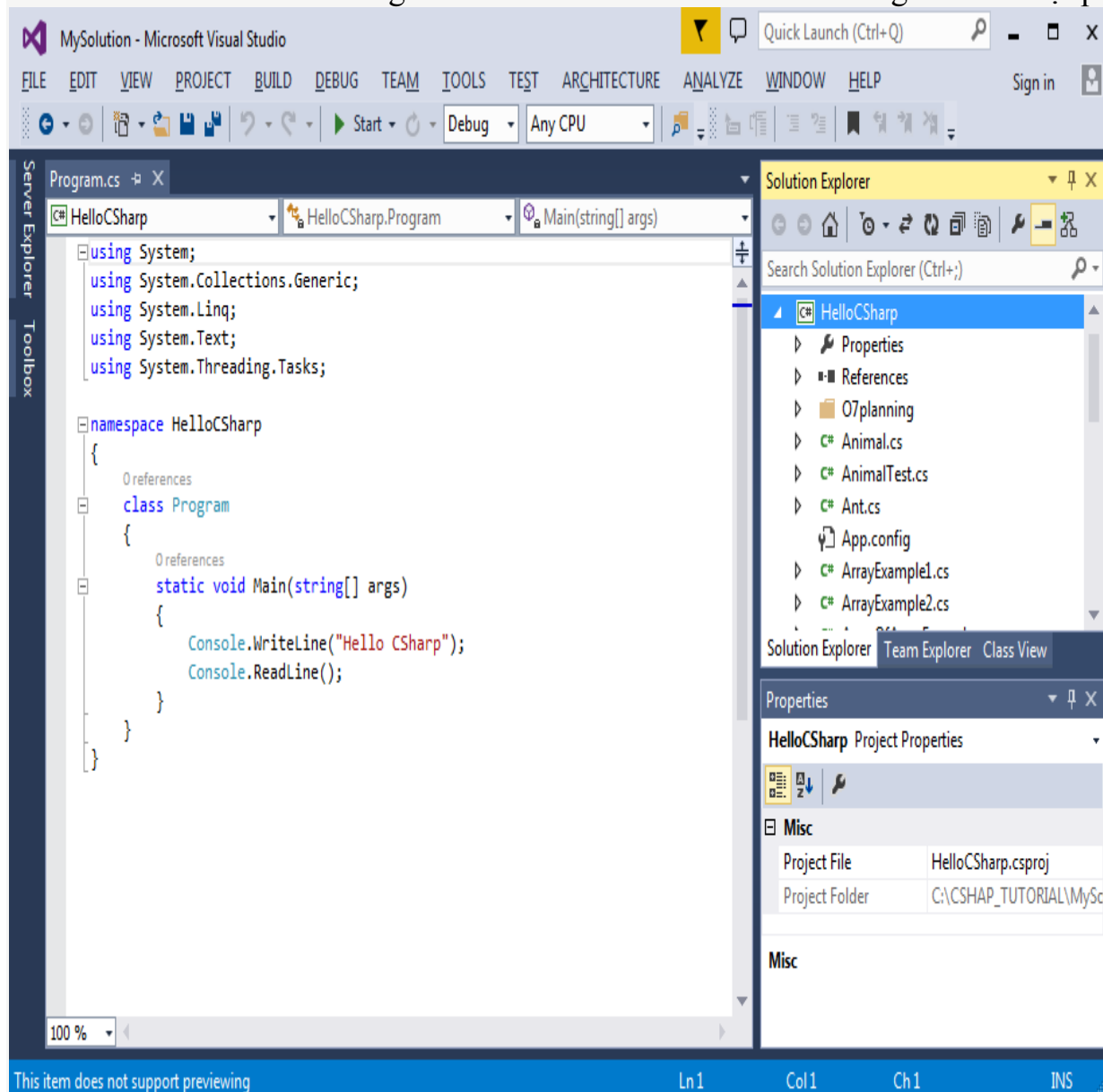
Hướng dẫn làm việc với Database MySQL sử dụng C#

Kết nối Database Oracle sử dụng C# không cần Oracle Client

Hướng dẫn làm việc với Database Oracle sử dụng C#

1- Bắt đầu với C# cần những gì?

Để lập trình với C# bạn cần cài đặt Visual Studio. Cho tới thời điểm tháng 4-2015 phiên bản Visual Studio mới nhất của Microsoft là Visual Studio 2015 mới ở dạng thử nghiệm không chính thức. Phiên bản trước gần nhất là Visual Studio 2013. Chúng ta sẽ cài đặt phiên bản này.



Visual Studio hỗ trợ nhiều ngôn ngữ lập trình khác nhau và cho phép trình biên tập mã và gỡ lỗi để hỗ trợ (mức độ khác nhau) hầu như mọi ngôn ngữ lập trình. Các ngôn ngữ tích hợp gồm có **C**, **C++** và **C++/CLI** (thông qua **Visual C++**), **VB.NET** (thông qua **Visual Basic.NET**), **C** thăng (thông qua **Visual C#**) và **F#** (như của **Visual Studio 2010**). Hỗ trợ cho các ngôn ngữ khác như **J++/J#**, **Python** và **Ruby** thông qua dịch vụ cài đặt riêng rẽ. Nó cũng hỗ trợ **XML/XSLT**, **HTML/XHTML**, **JavaScript** và **CSS**.

Visual Studio 2013 chia ra nhiều phiên bản thương mại:

- Visual Studio Ultimate 2013
- Visual Studio Premium 2013
- Visual Studio Professional 2013
- Visual Studio Test Professional 2013
- Visual Studio Team Foundation Server 2013

Phiên bản đầy đủ nhất là **Visual Studio 2013 Ultimate**, với các gói lập trình đầy đủ nhất, và hỗ trợ tốt nhất, dung lượng vào khoảng **7.2GB**.

2- Cài đặt và cấu hình Visual Studio 2013

1- Giới thiệu

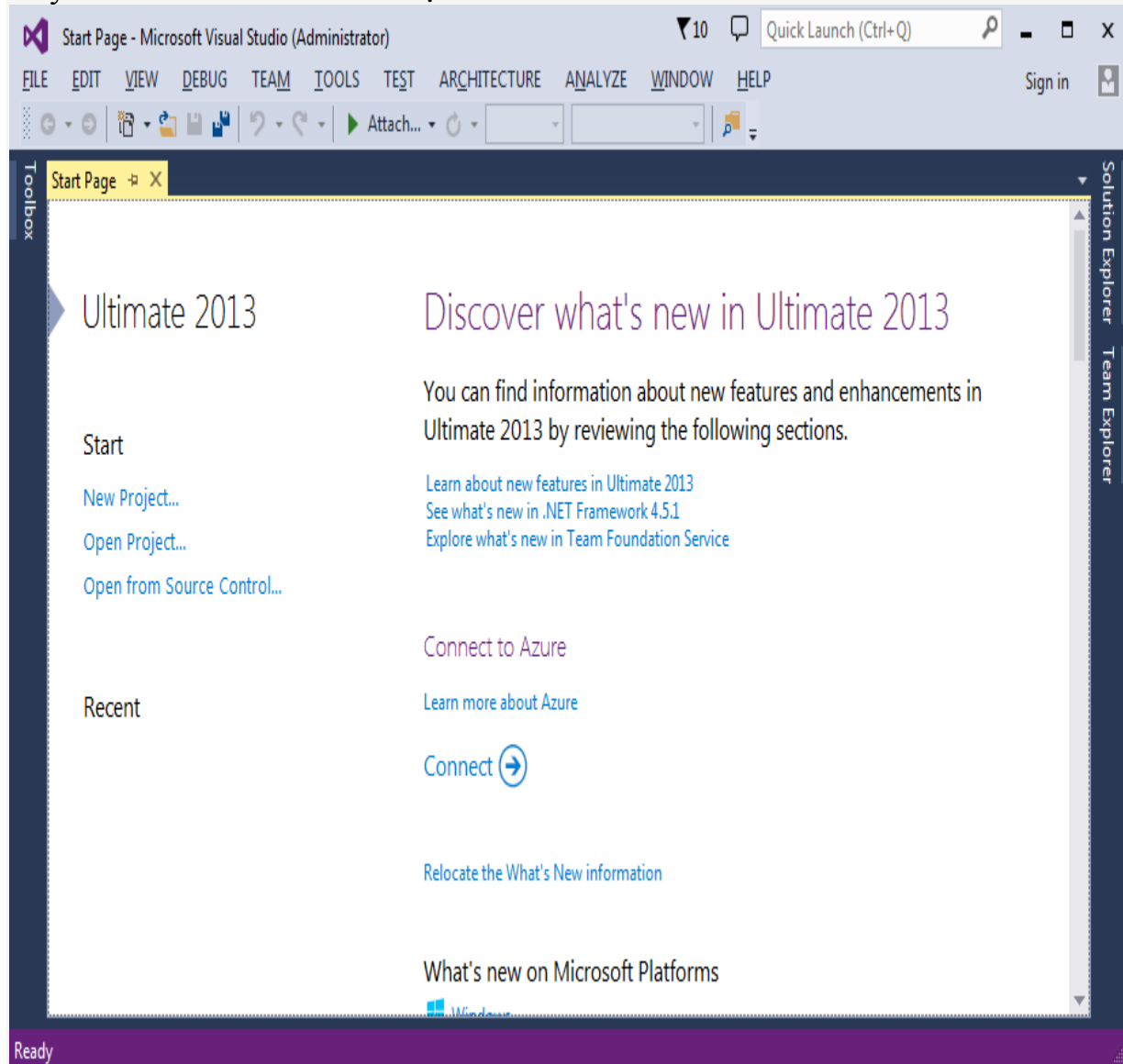
Đây là tài liệu hướng dẫn C# cho người mới bắt đầu. Để lập trình C# bạn phải cài đặt công cụ lập trình **Visual Studio**. Bạn có thể xem hướng dẫn download và cài đặt tại:

- <http://o7planning.org/web/fe/default/vi/document/239644/bat-dau-voi-csharp-can-nhung-gi>

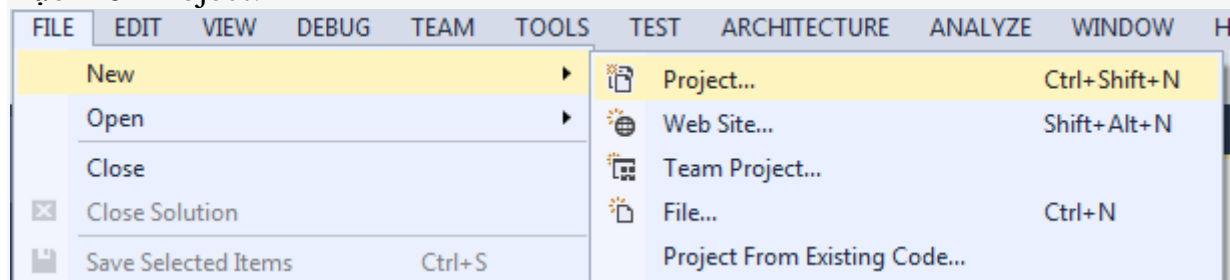
Đây là một tài liệu học nhanh, nếu bạn mới bắt đầu với C#, cách hướng dẫn là "**Từng bước từng bước**" vì vậy bạn hãy đọc lần lượt từ trên xuống, tài liệu này sẽ giúp bạn có cái nhìn tổng quan trước khi đi vào các tài liệu chi tiết khác.

2- Tạo Project C# đầu tiên của bạn

Đây là hình ảnh đầu tiên khi bạn mở **Visual Studio**.

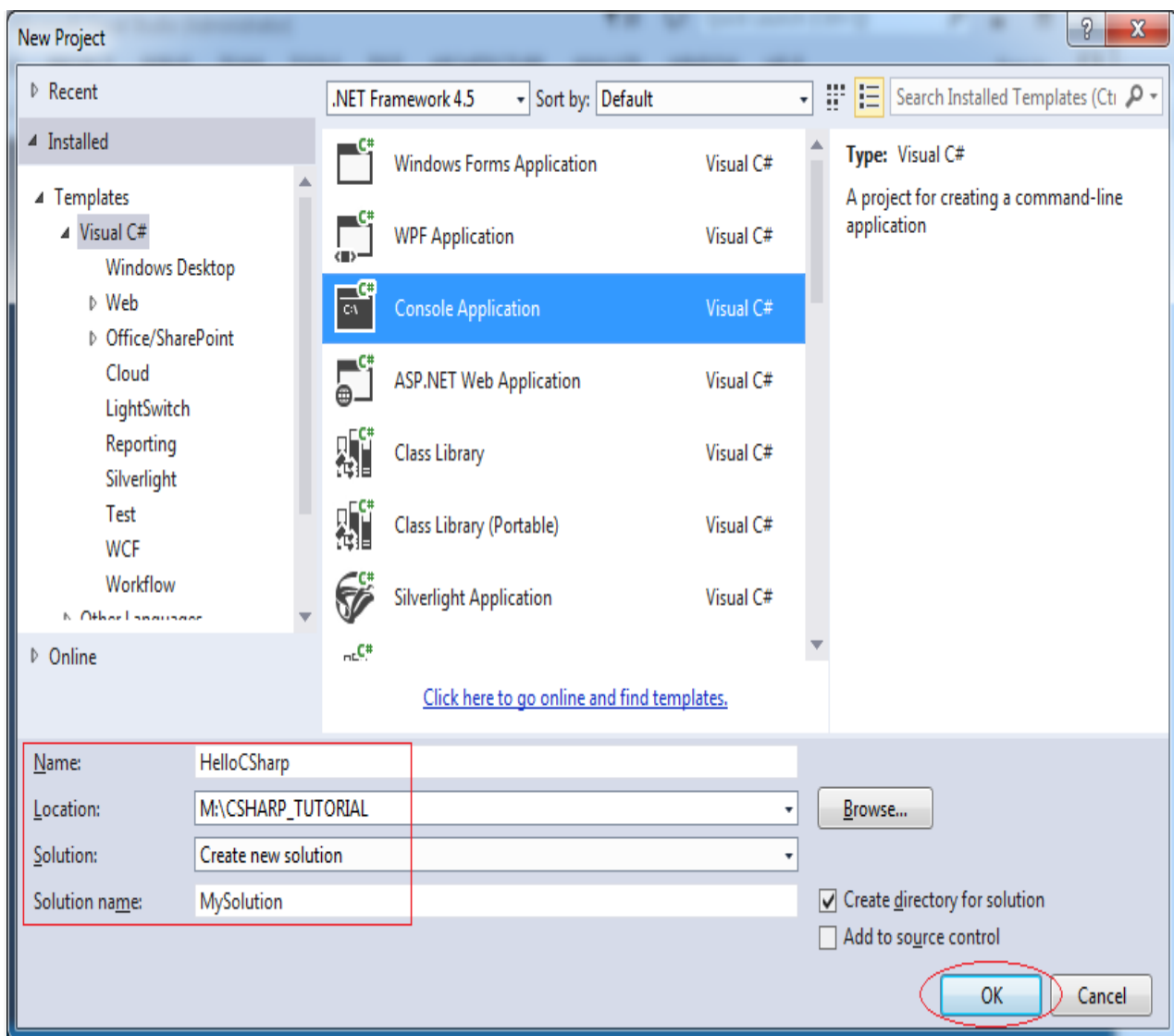


Tạo mới Project:

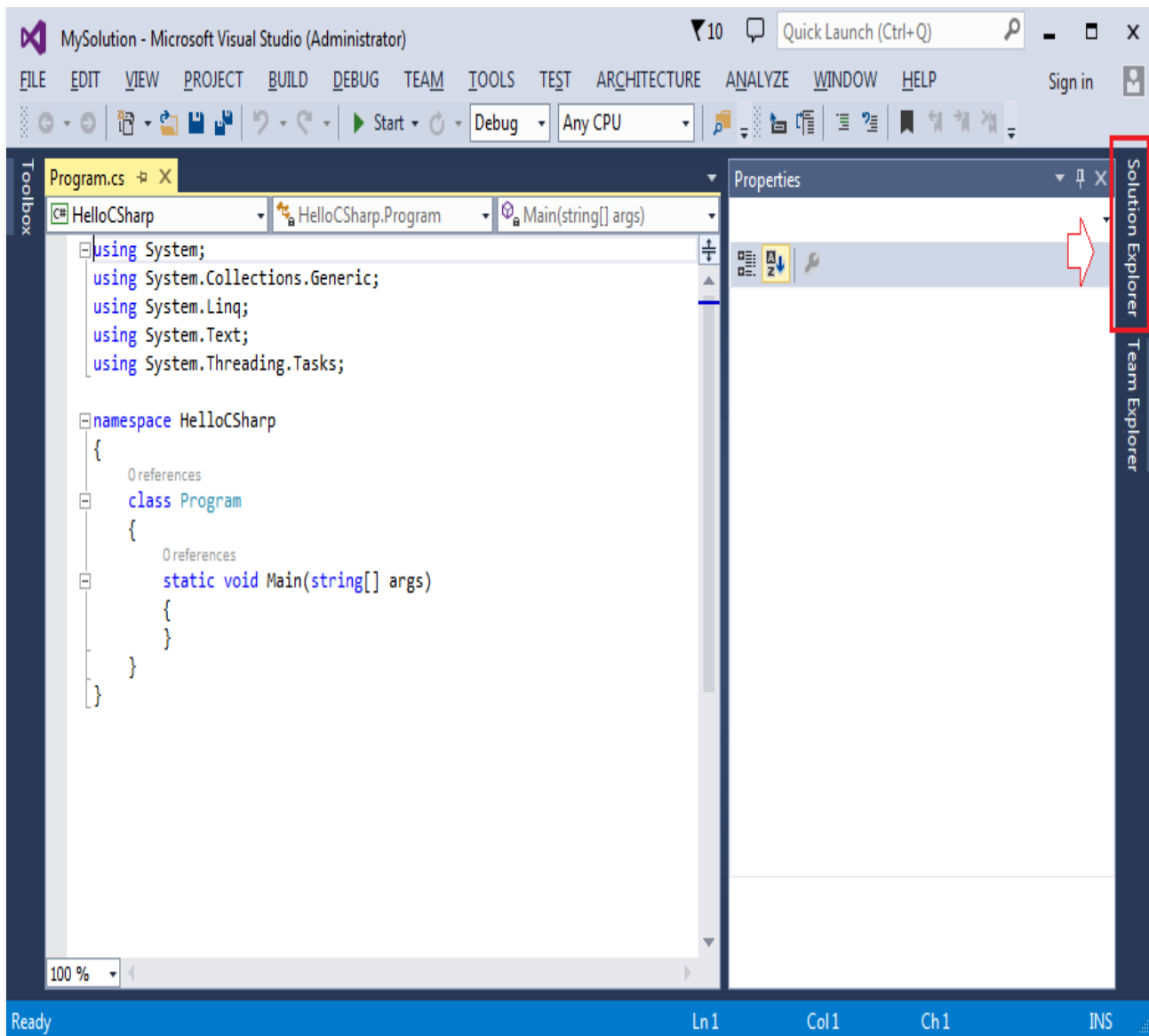


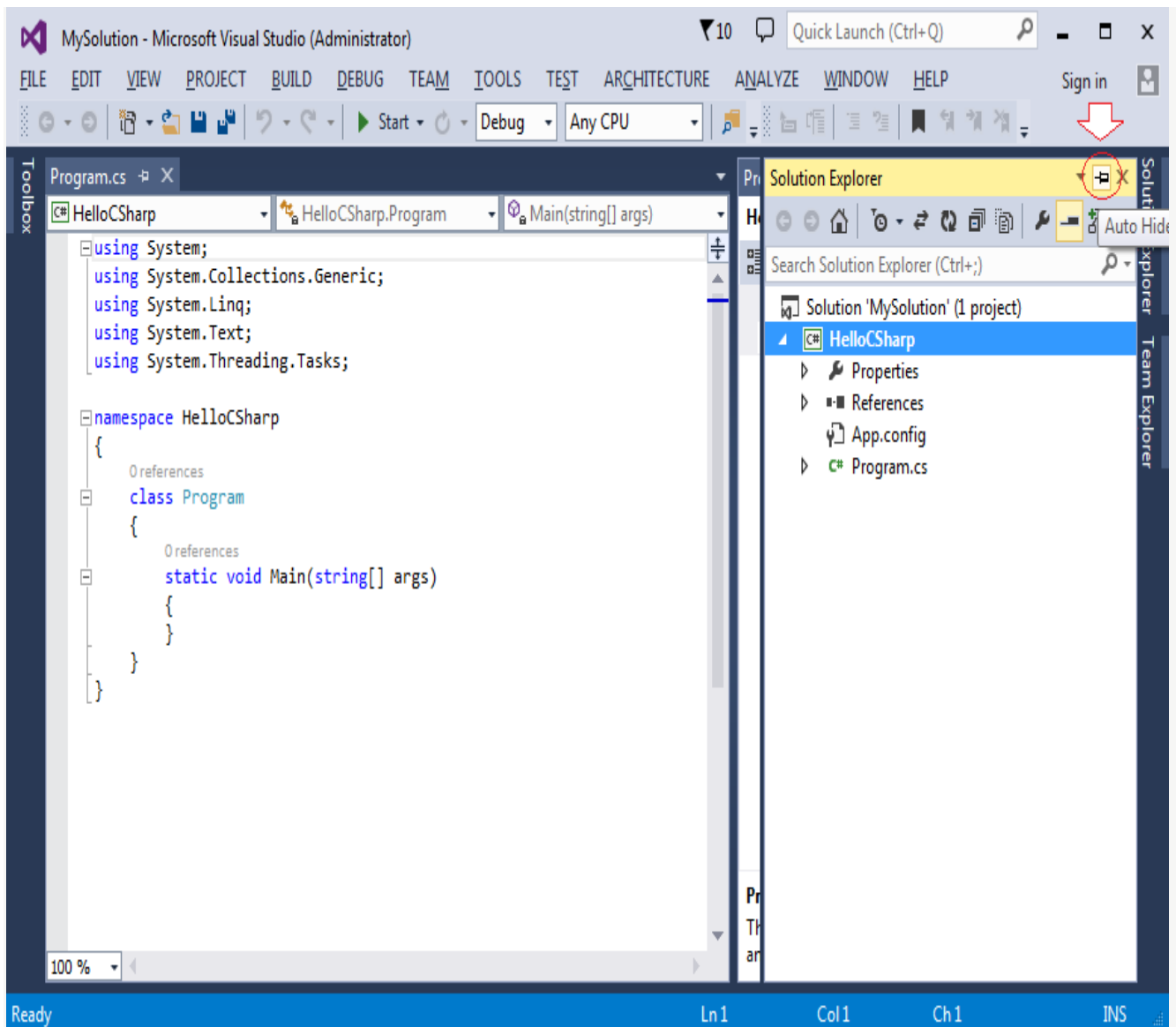
Chúng ta tạo một Project đơn giản (Ứng dụng Console, là ứng dụng không có giao diện). Nhập vào:

- **Name:** HelloCSharp
- **Solution:** Create new solution
- **Solution Name:** MySolution



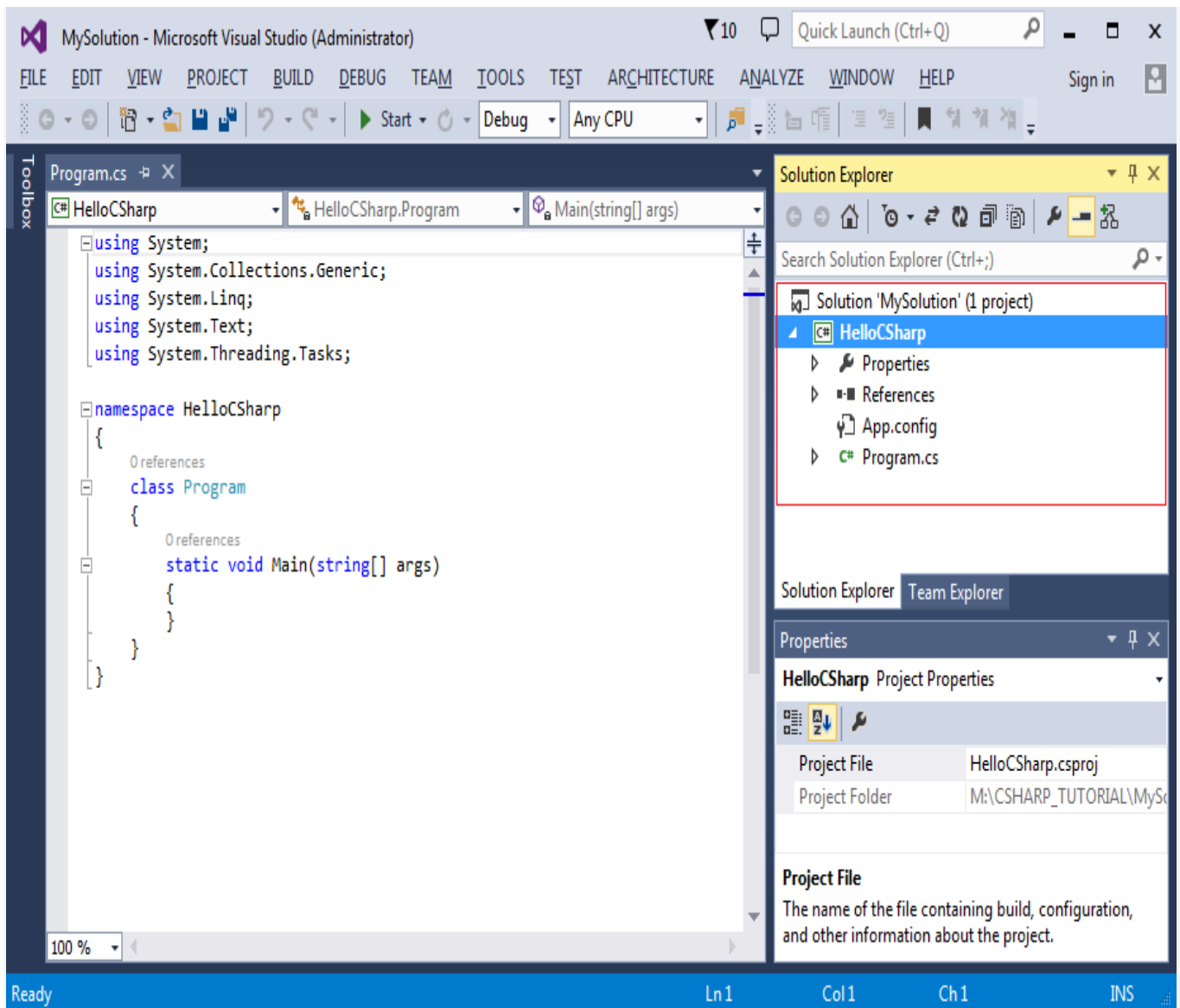
Đây là hình ảnh Project của bạn đã được tạo ra. Bạn cần nhấn vào **Solution Explorer** để xem cấu trúc của Project vừa được tạo ra đó.



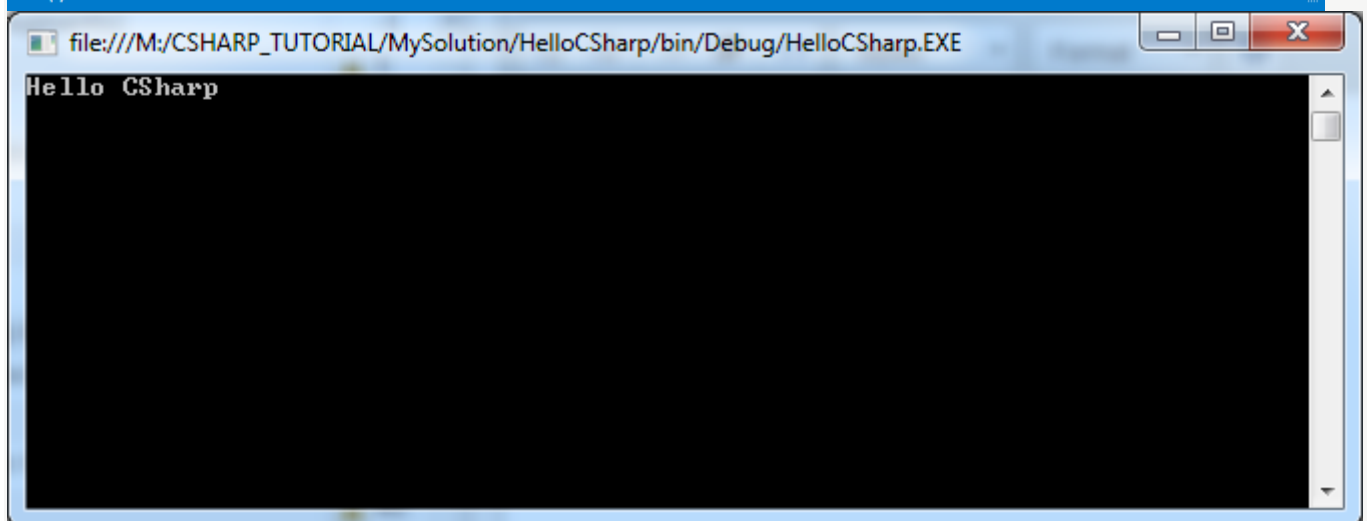
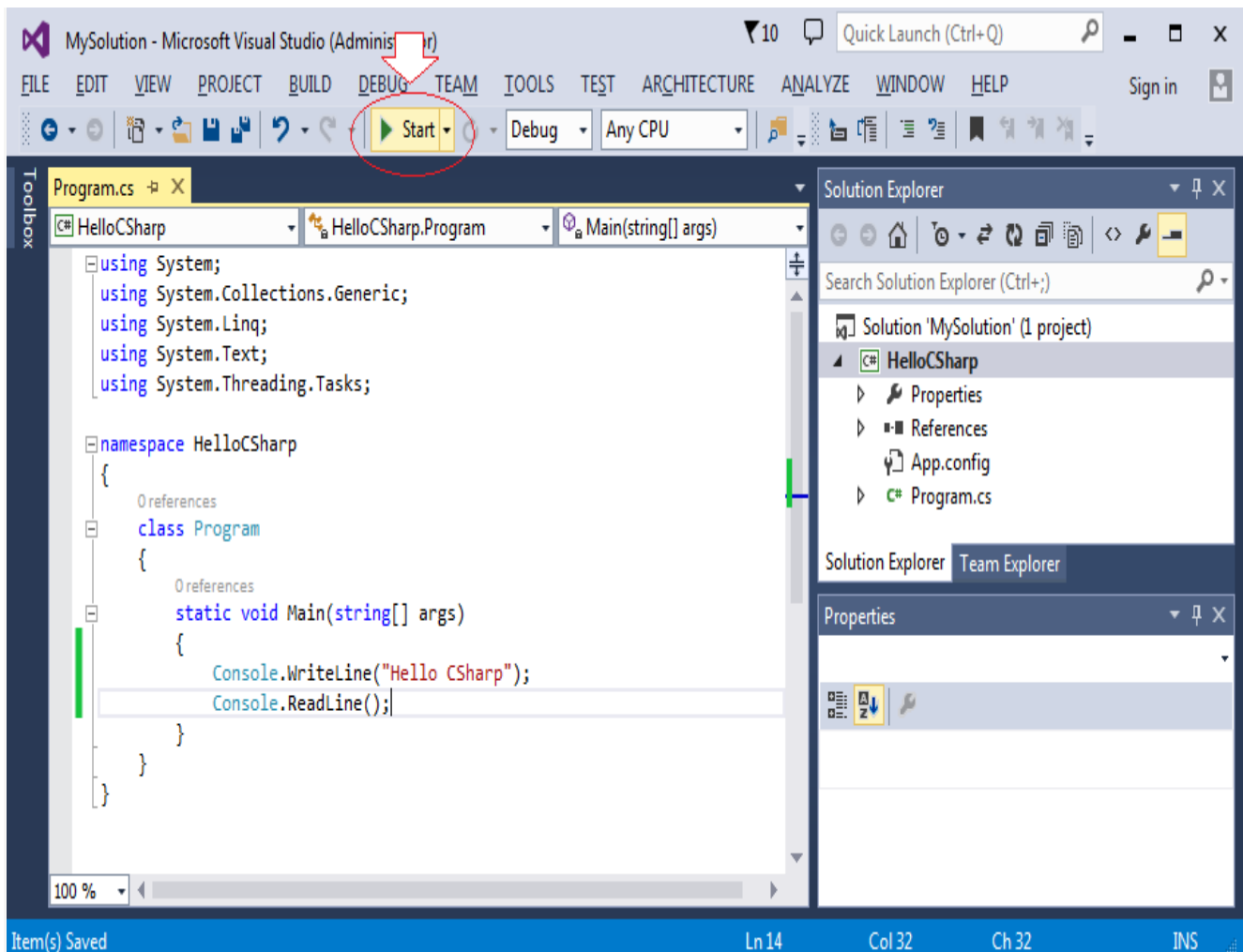


Visual Studio tạo ra một **Solution** (Giải pháp) có tên là **MySolution** và chứa bên trong nó là một Project có tên **HelloCSharp**. Và tạo mặc định một class có tên **Program** (Ứng với file **Program.cs**).

Chú ý: Một Solution có thể có một hoặc nhiều Project.

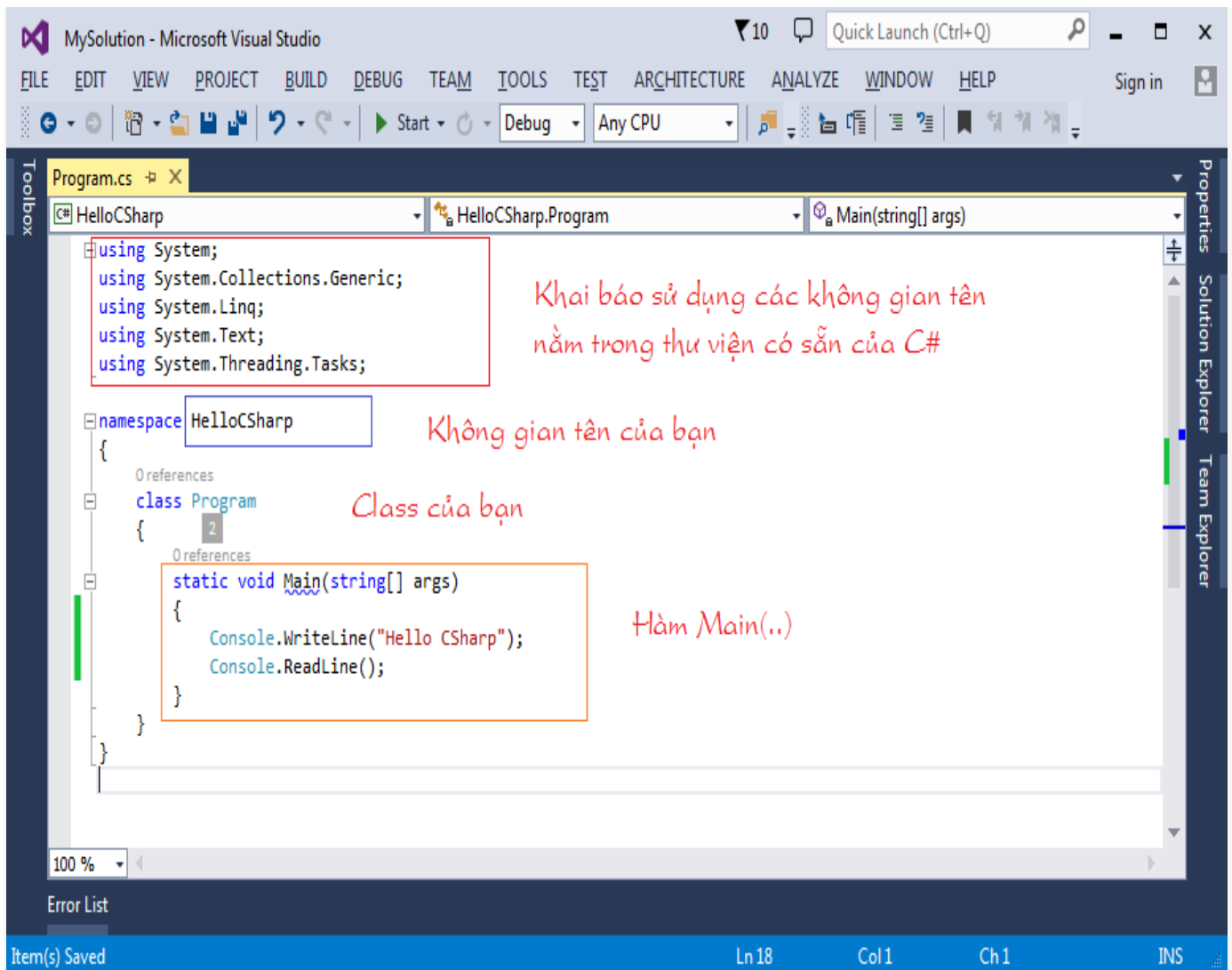


Sửa code của class Program, để khi chạy nó in ra màn hình Console một dòng chữ **"Hello CSharp"**, và chờ đợi người dùng nhập vào một dòng text trước khi kết thúc. Nhấn vào Start để chạy class Program.



3- Giải thích cấu trúc của một class

Hình minh họa dưới đây là cấu trúc của class có tên là Program, nó nằm trong không gian tên (namespace) HelloCSharp. Một không gian tên có thể chứa một hoặc nhiều class, nó thực sự là một tập hợp của các class.



Nếu bạn muốn sử dụng một class nào đó, bạn phải khai báo sử dụng class đó, hoặc khai báo sử dụng không gian tên chứa class đó.

?

1

```
2 // Khai báo sử dụng namespace System.
3 // (Nghĩa là có thể sử dụng tất cả các class có trong namespace này).
4 using System;
```

5

Khi chương trình được chạy, phương thức Main(string[]) sẽ được gọi thực thi.

1. static là từ khóa thông báo rằng đây là phương thức tĩnh.
2. void là từ khóa thông báo rằng phương thức này không trả về gì cả.
3. args là tham số của phương thức nó có kiểu mảng của chuỗi - string[].

?

```
1 static void Main(string[] args)
2 {
3     // Ghi ra màn hình Console một dòng chữ.
4     Console.WriteLine("Hello CSharp");
5
6     // Đợi người dùng gõ vào một dòng chữ trước khi tắt màn hình Console.
7     Console.ReadLine();
8 }
```

Sau khi đã khai báo sử dụng namespace System, bạn có thể sử dụng class **Console** nằm trong namespace này. **WriteLine(string)** là một phương thức tĩnh của class **Console**, nó ghi ra màn hình một chuỗi ký tự.

?

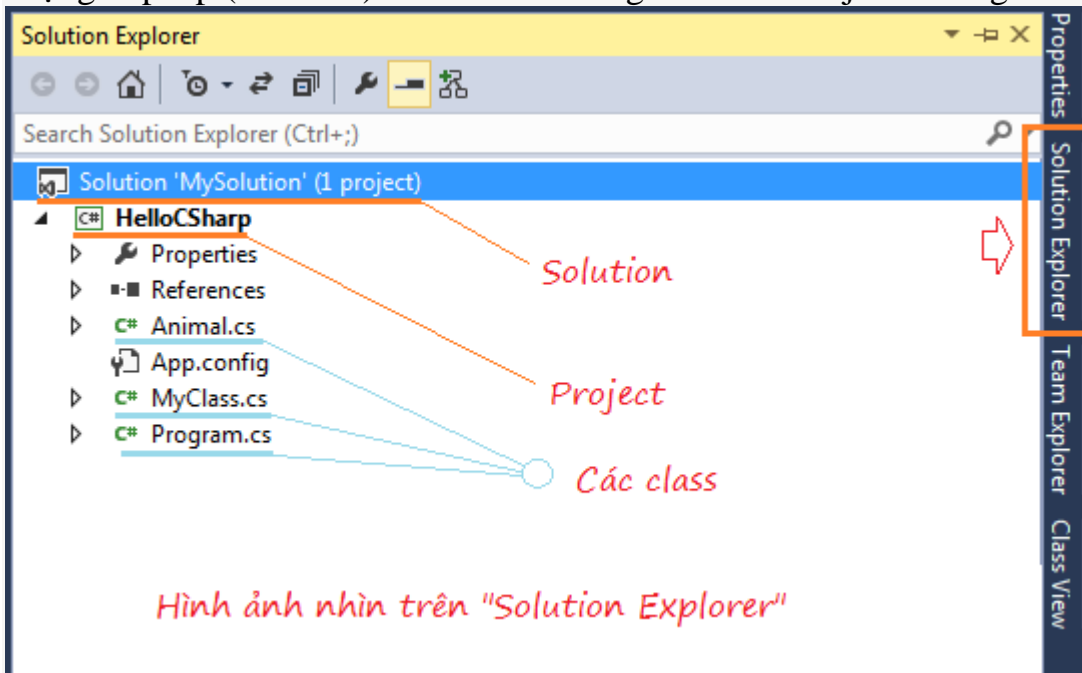
```

1 // Khai báo sử dụng không gian tên System
2 // (Nó có chứa class Console).
3 using System;
4
5 // Và có thể sử dụng class Console:
6 Console.WriteLine("Hello CSharp");
7
8 // Nếu bạn không muốn khai báo sử dụng không gian tên
9 // Nhưng muốn in ra một dòng text, bạn phải viết đầy đủ:
10 System.Console.WriteLine("Hello CSharp");

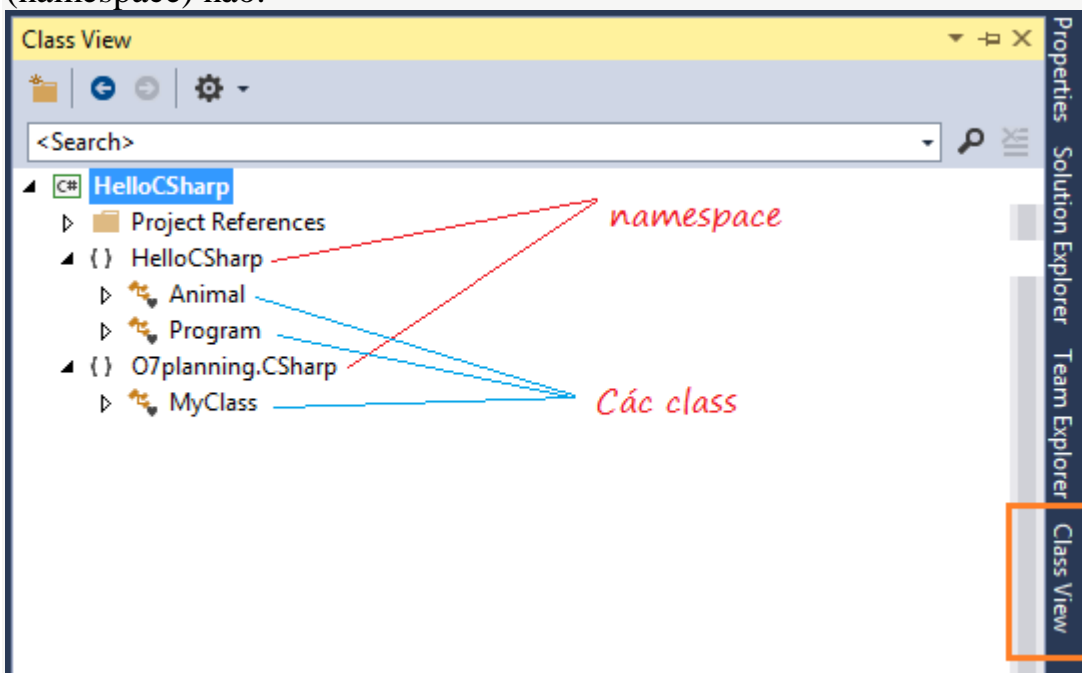
```

4- Giải thích cấu trúc Project

Một giải pháp (Solution) có thể chứa trong nó nhiều Project. Trong các Project chứa các class.

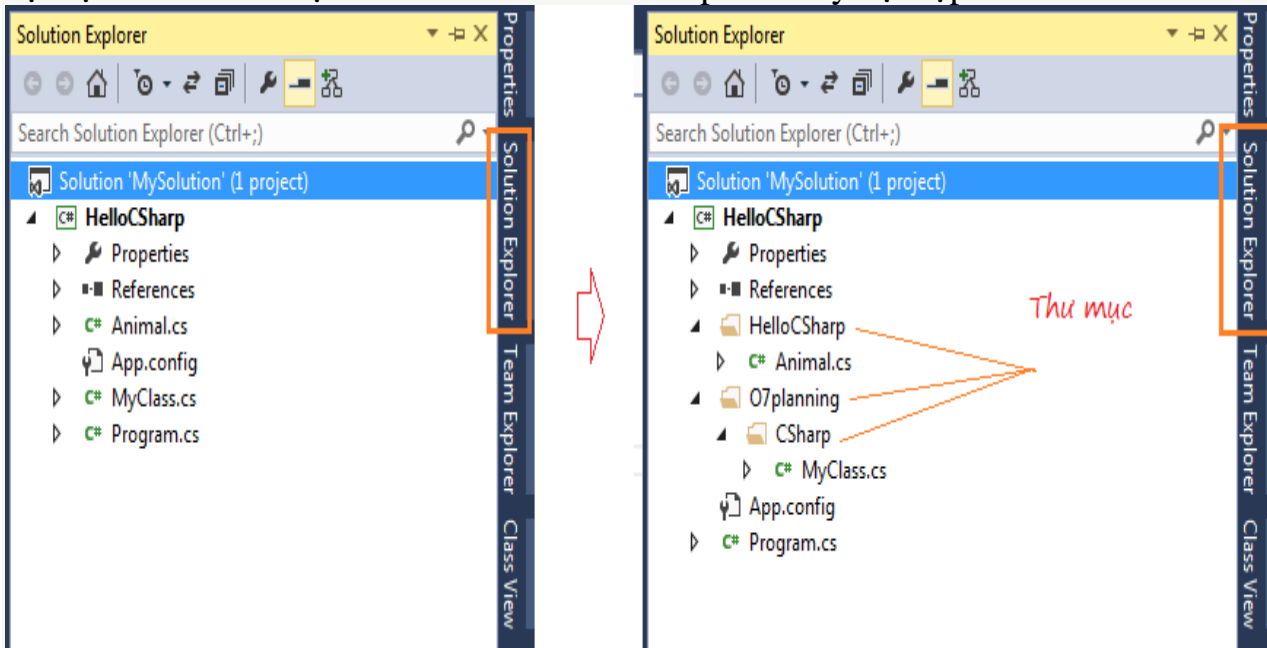


Khi nhìn trên "**Class View**" bạn có thể thấy được các class của bạn thuộc vào không gian tên (namespace) nào.



Trong **CSharp** bạn tạo ra một class **Animal** với không gian tên là **HelloCSharp**, class này mặc định sẽ nằm tại thư mục gốc của project. Bạn tạo ra một class khác là **MyClass** với không gian tên là **O7planning.CSharp** class này cũng nằm tại thư mục gốc của project. Với một project lớn nhiều class, cách tổ chức các file như vậy gây khó khăn cho bạn. Bạn có thể tạo ra

các thư mục khác nhau để chứa các file class, quy tắc do bạn quyết định, tuy nhiên tốt nhất bạn tạo ra các thư mục có tên là tên của namespace. Hãy học tập cách tổ chức của Java.

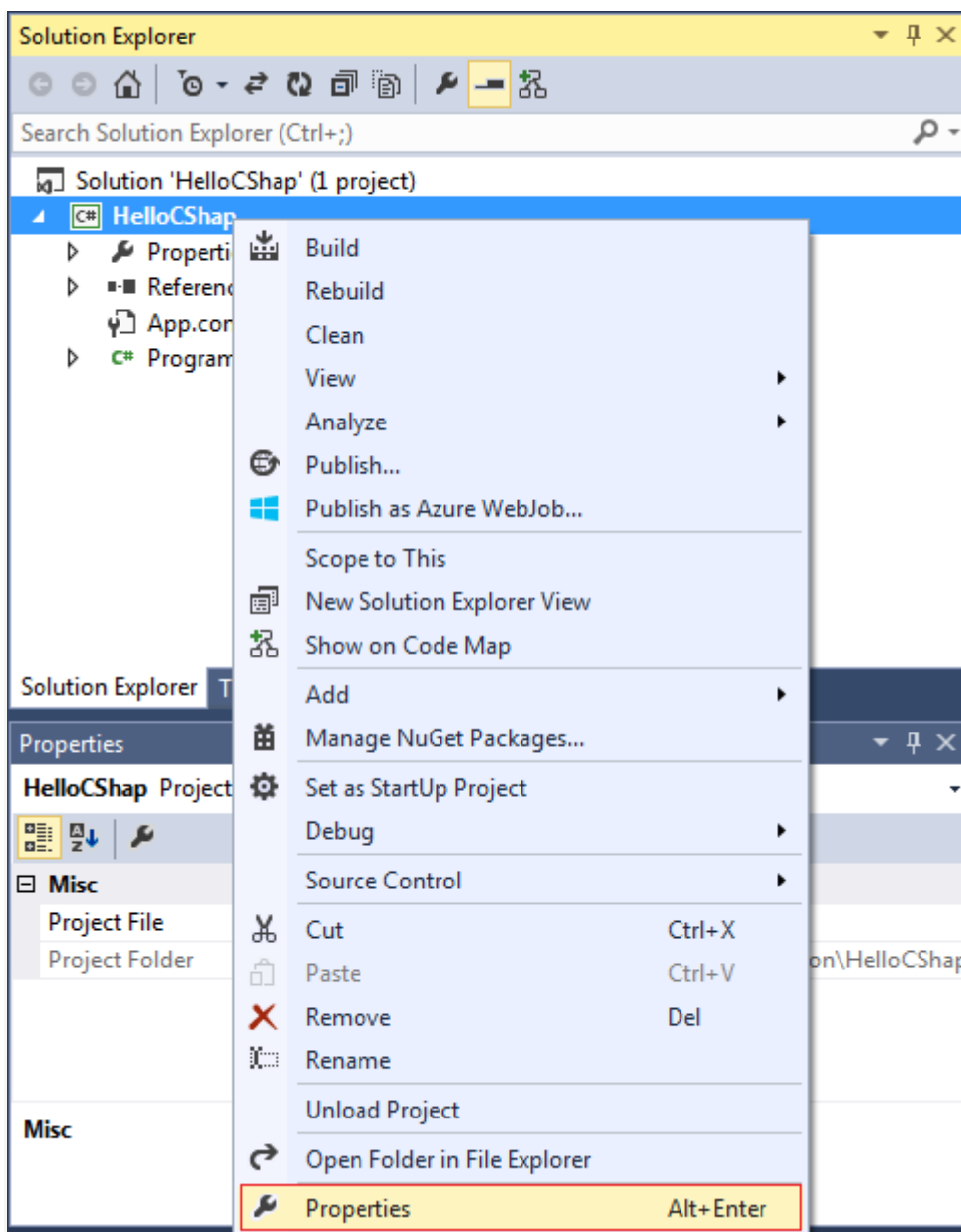


5- Chú ý quan trọng với một chương trình C#

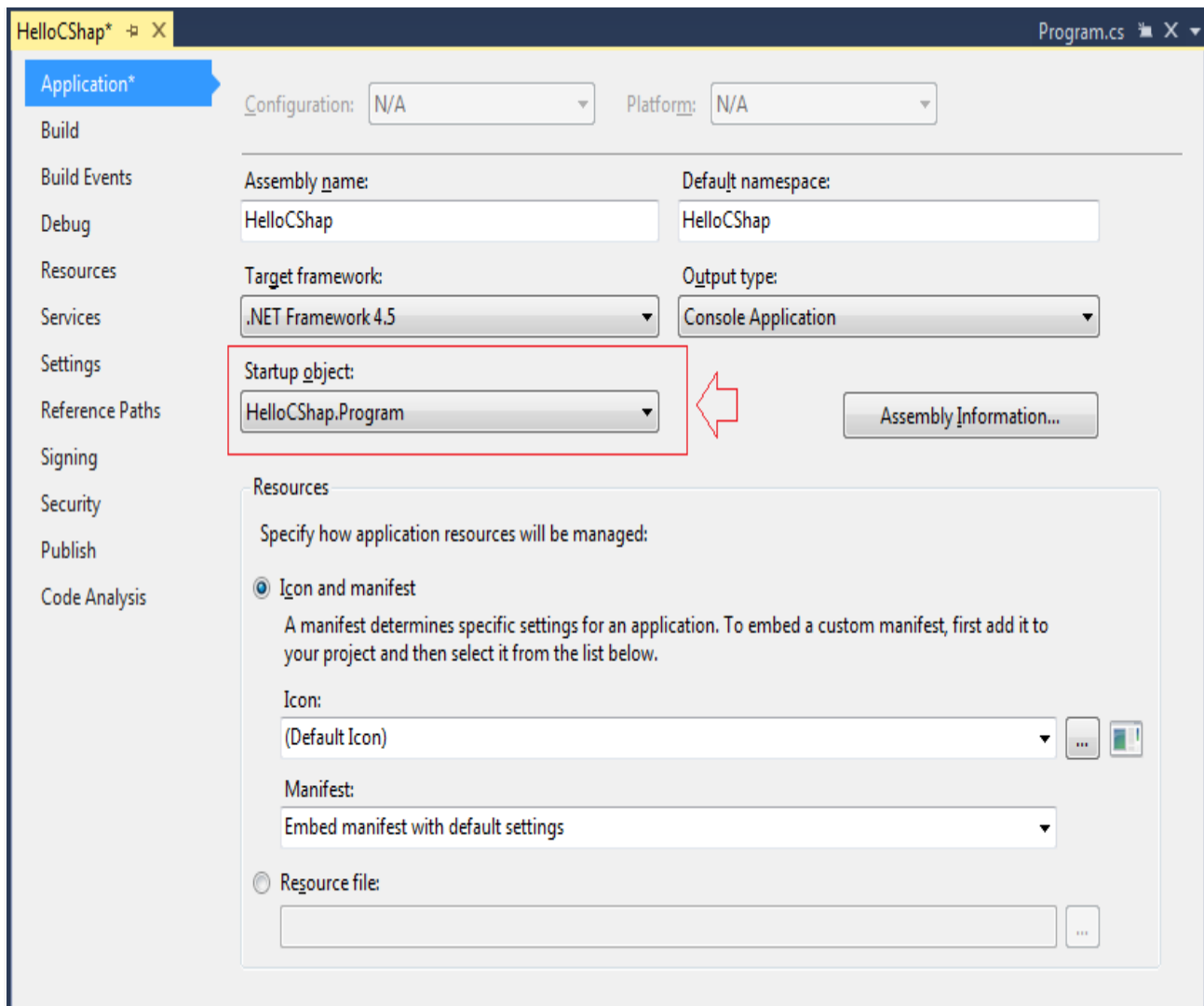
Trong một ứng dụng C# bạn cần khai báo rõ ràng một class có phương thức **Main(string[])** dùng làm điểm bắt đầu để chạy ứng dụng của bạn, điều này không bắt buộc nếu toàn bộ ứng dụng của bạn có duy nhất một class có phương thức **Main(string[])**, nhưng trong trường hợp có 2 class có phương thức Main nếu bạn không chỉ định rõ, một thông báo lỗi sẽ bị ném ra trong quá trình biên dịch.

Vì vậy tốt nhất bạn hãy khai báo rõ ràng class có phương thức **Main(string[])**, bạn có thể khai báo lại cho một class khác nếu muốn.

Nhấn phải chuột vào project **HelloCSharp**, chọn **Properties**:



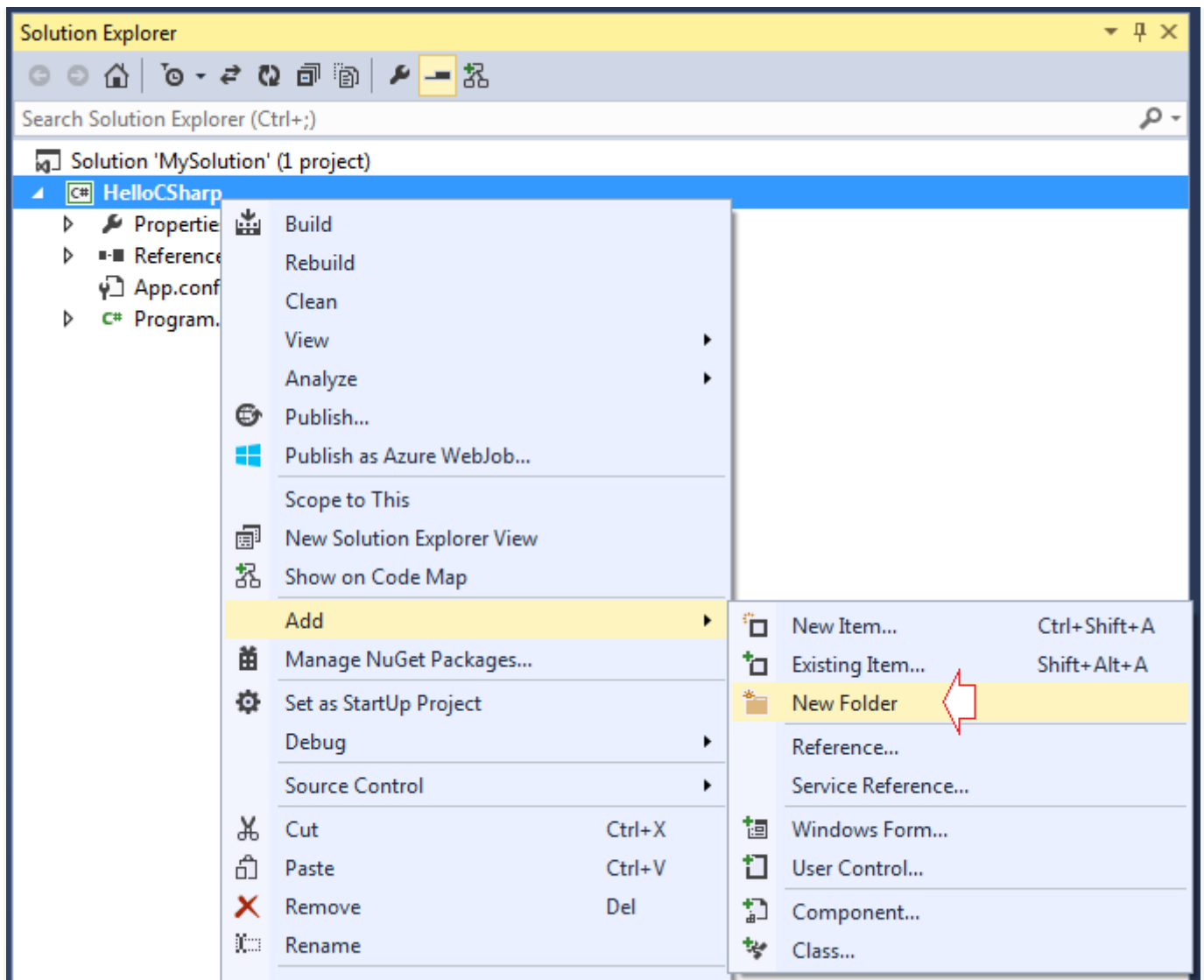
Chọn "**Startup object**" là một class có phương thức **Main(string[])** và Save lại.



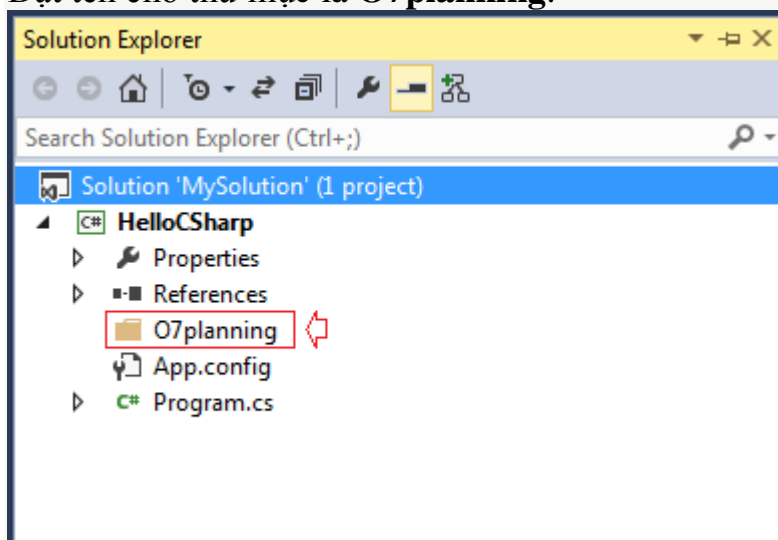
6- Thêm mới class

Bây giờ tôi thêm mới một class **MyClass** với không gian tên **O7planning.CSharp**. Trên "**Solution Explorer**" nhấn phải chuột vào project chọn:

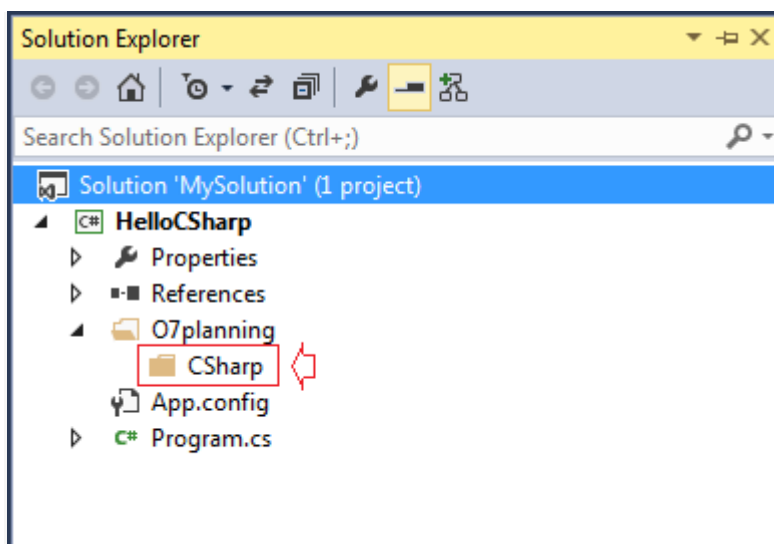
- **Add/New Folder**



Đặt tên cho thư mục là **O7planning**.

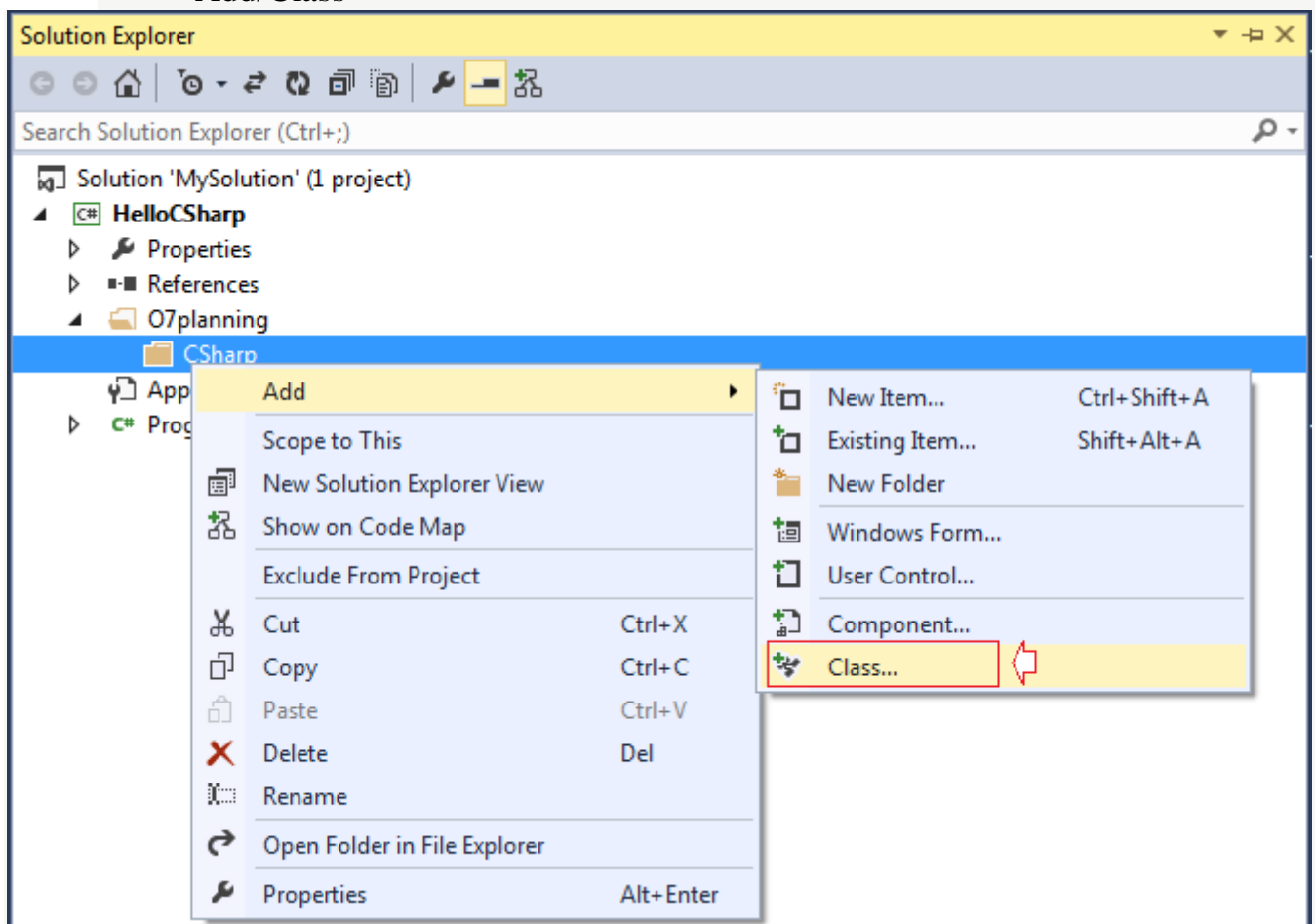


Tiếp tục tạo một thư mục "**CSharp**" là con của thư mục "**O7planning**".

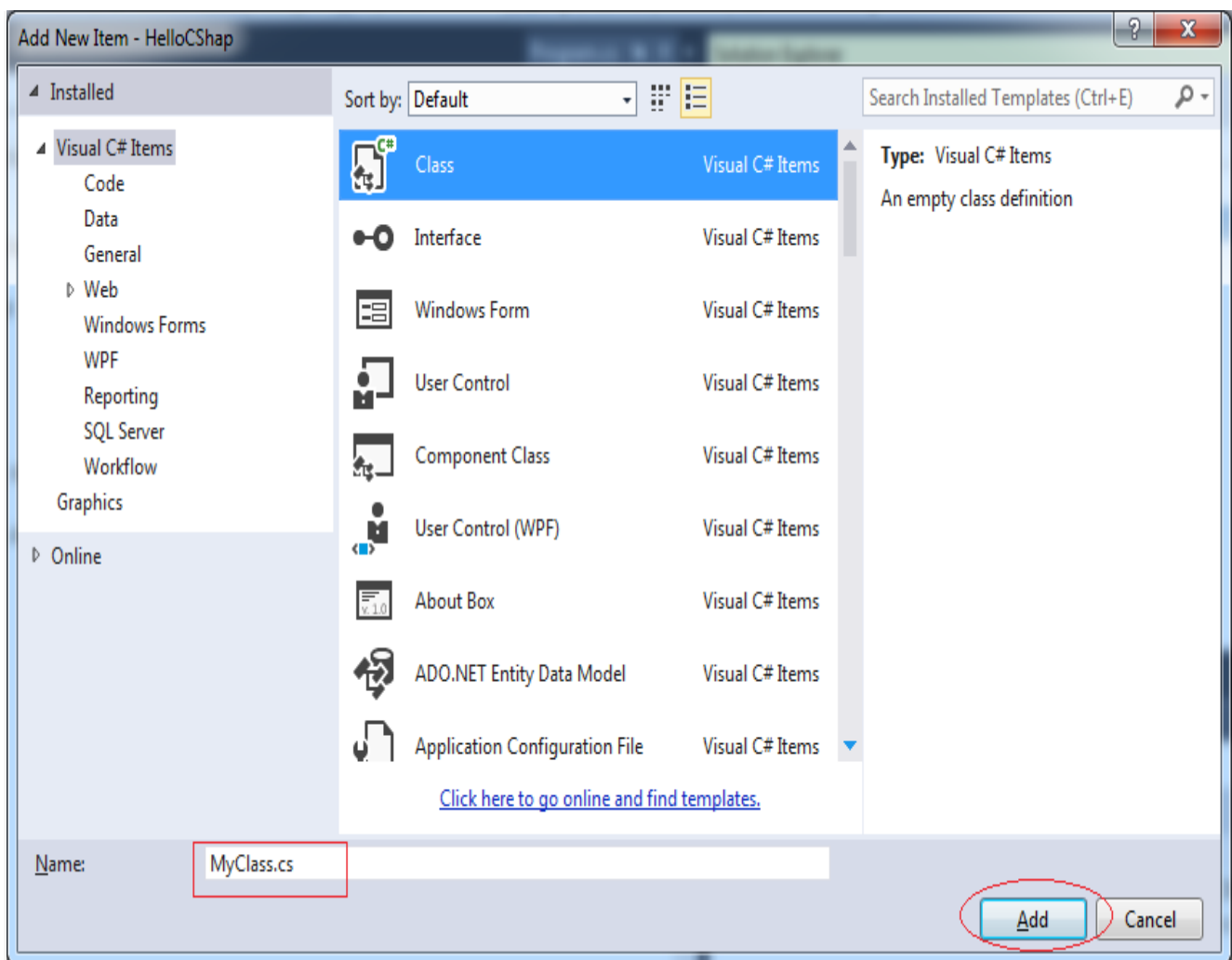


Nhấn phải chuột vào thư mục **"CSharp"** chọn:

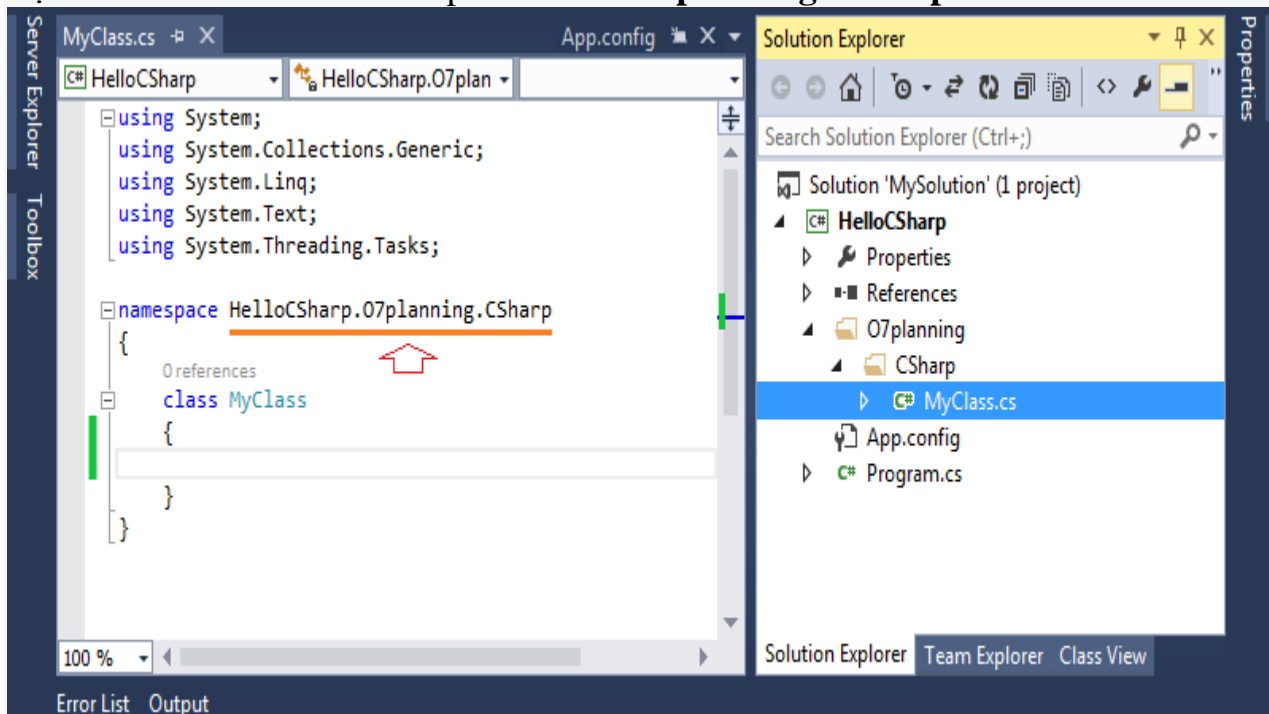
- Add/Class



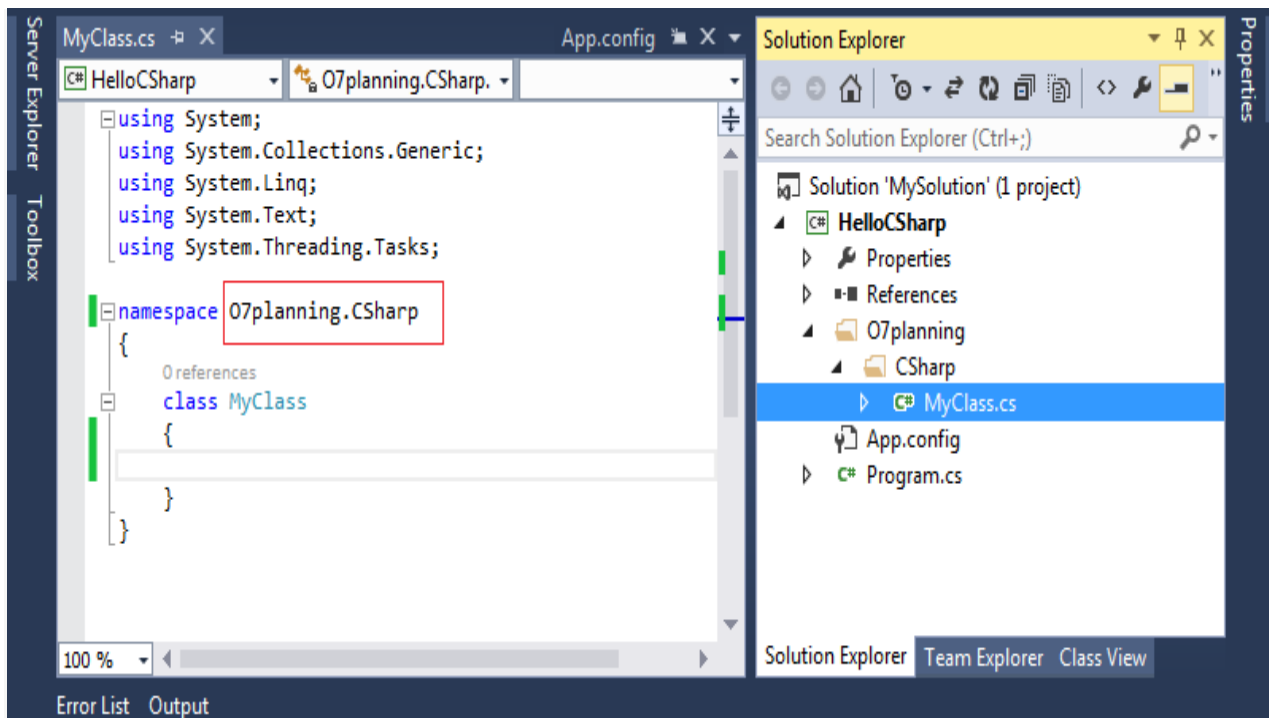
Chọn kiểu item là Class, và nhập tên class.



Class đã được tạo ra, nó nằm trong không gian tên **"HelloCSharp.O7planning.CSharp"**. Bạn có thể đổi tên cho namespace thành **"O7planning.CSharp"**.



Đổi tên namespace thành **"O7planning.CSharp"**.



Bạn có thể thay đổi nội dung của class:

MyClass.cs

```
?
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace O7planning.CSharp
8  {
9      class MyClass
10     {
11         static void Main(string[] args)
12         {
13             Console.WriteLine("Hello from MyClass");
14             Console.ReadLine();
15         }
16     }
17 }
```

Khai báo class **MyClass** là điểm bắt đầu để chạy. Nhấn phải chuột vào Project chọn Properties

Application Configuration: N/A Platform: N/A

Build Events

Debug Assembly name: HelloCSharp Default namespace: HelloCSharp

Resources Target framework: .NET Framework 4.5 Output type: Console Application

Services

Settings Startup object: O7planning.CSharp.MyClass Assembly Information...

Reference Paths

Signing

Security

Publish

Code Analysis

Resources

Specify how application resources will be managed:

☒ Icon and manifest

A manifest determines specific settings for an application. To embed a custom manifest, first add it to your project and then select it from the list below.

Icon: (Default Icon)

Manifest: Embed manifest with default settings

☐ Resource file:

Chạy ví dụ:

```
file:///C:/CSHAP_TUTORIAL/MySolution/HelloCSharp/bin/Debug/HelloCSharp.EXE
Hello from MyClass
```

7- 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

8- Biến và khai báo



Một biến xác định bởi một cái tên cho một khu vực lưu trữ dữ liệu mà chương trình của bạn có thể thao tác. 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:

```
?
1 // Khai báo một biến.
2 <kiểu dữ liệu> <tên biến>;
3
4 // Khai báo một biến đồng thời gán luôn giá trị.
5 <kiểu dữ liệu> <tên biến> = <giá trị>;
6
7 // Khai báo một hằng số:
8 const <kiểu dữ liệu> <tên hằng số> = <giá trị>;
```

VariableExample.cs

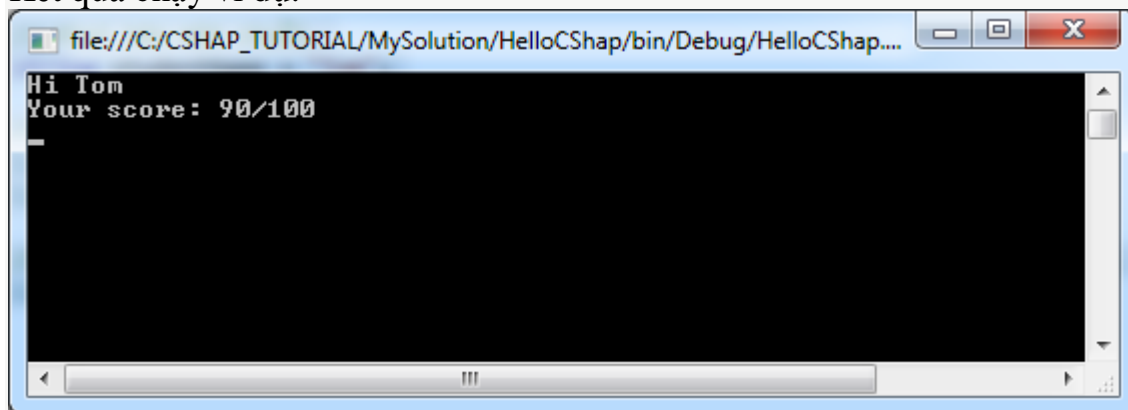
```
?
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6
7 namespace HelloCSharp
8 {
9     class VariableExample
10    {
```

```

11     static void Main(string[] args)
12     {
13         // Khai báo một hằng số nguyên
14         const int MAX_SCORE = 100;
15
16         // Khai báo một biến số nguyên
17         int score = 90;
18
19         // Khai báo một chuỗi.
20         string studentName = "Tom";
21
22         // Ghi các thông tin ra màn hình Console.
23         Console.WriteLine("Hi {0}", studentName);
24         Console.WriteLine("Your score: {0}/{1}", score, MAX_SCORE);
25
26         // Chờ người dùng nhập vào gì đó và nhấn Enter trước khi kết thúc chương trình.
27         Console.ReadLine();
28     }
29
30
31 }
32 }

```

Kết quả chạy ví dụ:



9- Câu lệnh rẽ nhánh



9.1- 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	$4 \geq 4$ là đúng (true)
<=	Nhỏ hơn hoặc bằng	$3 \leq 4$ là đúng (true)
==	Bằng nhau	$1 == 1$ là đúng (true)
!=	Không bằng nhau	$1 \neq 2$ là đúng (true)

&&	Và	$a > 4 \ \&\& \ a < 10$
	Hoặc	$a == 1 \ \ a == 4$

?

1 // Cú pháp

2

3 if (điều kiện)

4 {

5 // Làm gì đó tại đây.

6 }

Ví dụ:

?

1 // Ví dụ 1:

2 if (5 < 10)

3 {

4 Console.WriteLine("Five is now less than ten");

5 }

6

7 // Ví dụ 2:

8 if (true)

9 {

10 Console.WriteLine("Do something here");

11 }

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

?

1 // Chú ý rằng sẽ chỉ có nhiều nhất một khối được chạy

2 // Chương trình kiểm tra điều kiện từ trên xuống dưới khi bắt gặp một điều

3 // kiện đúng khối lệnh tại đó sẽ được chạy, và chương trình không kiểm tra tiếp các điều kiện

4 // còn lại trong cấu trúc rẽ nhánh.

5

6

7 // Nếu điều kiện 1 đúng thì ...

8 if (điều kiện 1)

9 {

10 // ... làm gì đó khi điều kiện một đúng.

11 }

12 // Ngược lại nếu điều kiện 2 đúng thì

13 else if(điều kiện 2)

14 {

15 // ... làm gì đó khi điều kiện 2 đúng (điều kiện 1 sai).

16 }

17 // Ngược lại nếu điều kiện N đúng thì ...

18 else if(điều kiện N)

19 {

20 // .. làm gì đó khi điều kiện N đúng (các điều kiện ở trên sai)

21 }

22 // Ngược lại nếu các điều kiện ở trên đều sai thì

23 else

24 {

25 // ... làm gì đó tại đây khi mọi điều kiện trên đều sai.

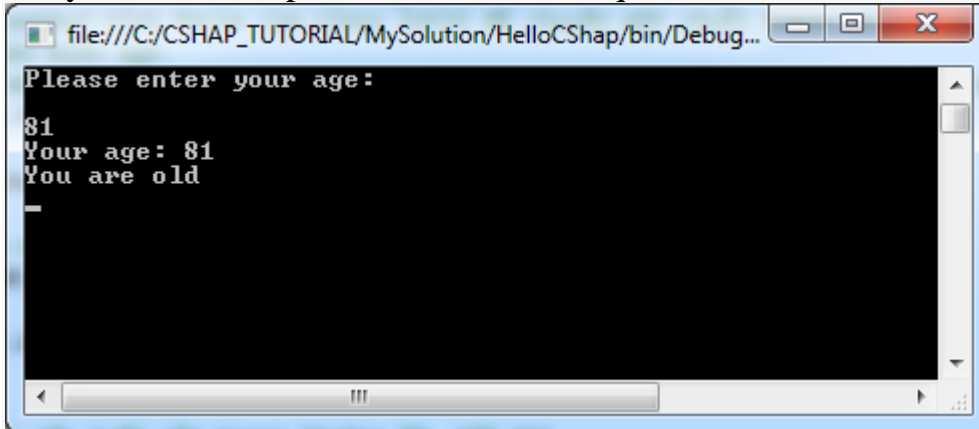
26 }

IfElseExample.cs

```
?  
1 using System;  
2 using System.Collections.Generic;  
3 using System.Linq;  
4 using System.Text;  
5 using System.Threading.Tasks;  
6  
7 namespace HelloCSharp  
8 {  
9     class IfElseExample  
10    {  
11        static void Main(string[] args)  
12        {  
13            // Khai báo một số mô tả tuổi của bạn.  
14            int age;  
15  
16            Console.WriteLine("Please enter your age: \n");  
17  
18            // Khai báo một biến input, lưu trữ dòng text người dùng nhập vào từ bàn phím.  
19            string inputStr = Console.ReadLine();  
20  
21            // Int32 là class nằm trong namespace System.  
22            // Sử dụng method tĩnh Parse của class Int32 để chuyển một chuỗi thành số  
23            // Và gán vào biến age.  
24            // (Chú ý: Nếu inputStr không phải là chuỗi số, có thể gây lỗi chương trình tại đây).  
25            age = Int32.Parse(inputStr);  
26  
27            Console.WriteLine("Your age: {0}", age);  
28  
29            // Kiểm tra nếu age nhỏ hơn 80 thì ...  
30            if (age < 80)  
31            {  
32                Console.WriteLine("You are pretty young");  
33            }  
34  
35            // Ngược lại nếu tuổi nằm trong khoảng 80, 100 thì  
36            else if (age >= 80 && age <= 100)  
37            {  
38                Console.WriteLine("You are old");  
39            }  
40  
41            // Ngược lại (Các trường hợp còn lại)  
42            else  
43            {  
44                Console.WriteLine("You are verry old");  
45            }  
46  
47            Console.ReadLine();
```

```
48     }  
49 }  
50 }
```

Chạy ví dụ, và nhập vào 81, và xem kết quả:



9.2- Câu lệnh Switch-Case

Cú pháp câu lệnh rẽ nhánh **switch**:

```
?  
1 // Sử dụng switch để kiểm tra một giá trị của một biến  
2 switch ( <variable> )  
3 {  
4     case value1:  
5         // Làm gì đó nếu giá trị của biến == value1  
6         break;  
7     case value2:  
8         // Làm gì đó nếu giá trị của biến == value2  
9         break;  
10    ...  
11    default:  
12        // Làm điều gì đó tại đây nếu giá trị của biến không thuộc các giá trị liệt kê ở trên.  
13        break;  
14 }
```

BreakExample.cs

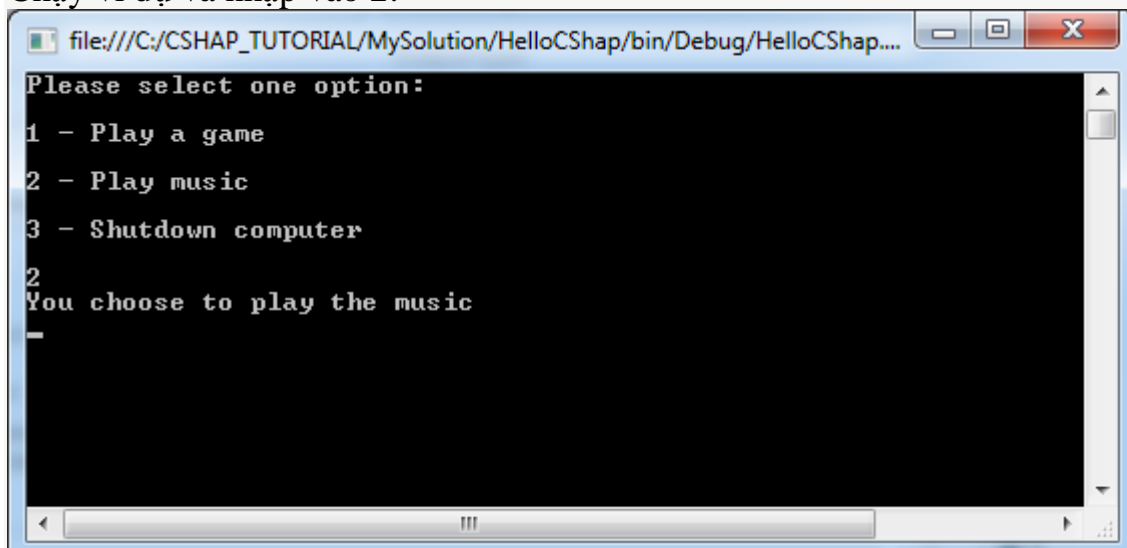
```
?  
1 using System;  
2 using System.Collections.Generic;  
3 using System.Linq;  
4 using System.Text;  
5 using System.Threading.Tasks;  
6  
7 namespace HelloCSharp  
8 {  
9     class BreakExample  
10    {  
11        static void Main(string[] args)  
12        {  
13            // Đề nghị người dùng chọn 1 lựa chọn.  
14            Console.WriteLine("Please select one option:\n");  
15  
16            Console.WriteLine("1 - Play a game \n");  
17            Console.WriteLine("2 - Play music \n");
```

```

18     Console.WriteLine("3 - Shutdown computer \n");
19
20
21
22     // Khai báo một biến option
23     int option;
24
25     // Chuỗi người dùng nhập vào từ bàn phím
26     string inputStr = Console.ReadLine();
27
28     // Chuyển chuỗi thành số nguyên.
29     option = Int32.Parse(inputStr);
30
31     // Kiểm tra giá trị của option
32     switch (option)
33     {
34
35         case 1:
36             Console.WriteLine("You choose to play the game");
37             break;
38         case 2:
39             Console.WriteLine("You choose to play the music");
40             break;
41         case 3:
42             Console.WriteLine("You choose to shutdown the computer");
43             break;
44         default:
45             Console.WriteLine("Nothing to do...");
46             break;
47     }
48
49     Console.ReadLine();
50 }
51 }
52 }

```

Chạy ví dụ và nhập vào 2:



```

file:///C:/CSHAP_TUTORIAL/MySolution/HelloCShap/bin/Debug/HelloCShap...
Please select one option:
1 - Play a game
2 - Play music
3 - Shutdown computer
2
You choose to play the music
_

```


Chú ý:

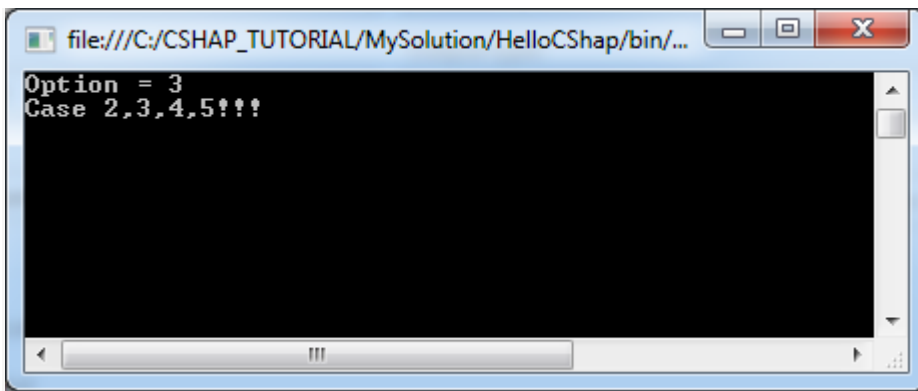
Có một vấn đề bạn đặt ra câu lệnh **break** trong trường hợp này có ý nghĩa gì. break trong trường hợp này nói với chương trình rằng thoát ra khỏi **switch**.

Bạn có thể gộp nhiều trường hợp **case** xử lý với cùng một khối lệnh.

SwitchExample2.cs

```
?
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace HelloCSharp
8  {
9      class BreakExample2
10     {
11         static void Main(string[] args)
12         {
13             // Khai báo biến option và gán giá trị 3.
14             int option = 3;
15
16             Console.WriteLine("Option = {0}", option);
17
18             // Kiểm tra giá trị của option
19             switch (option)
20             {
21
22                 case 1:
23                     Console.WriteLine("Case 1");
24                     break;
25                 // Trường hợp chọn 2,3,4,5 xử lý giống nhau.
26                 case 2:
27                 case 3:
28                 case 4:
29                 case 5:
30                     Console.WriteLine("Case 2,3,4,5!!!");
31                     break;
32                 default:
33                     Console.WriteLine("Nothing to do...");
34                     break;
35             }
36
37             Console.ReadLine();
38         }
39     }
40 }
```

Kết quả chạy ví dụ:



10- Vòng lặp trong C#



Vòng lặp 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**

10.1- Vòng lặp for

Cấu trúc của vòng lặp FOR:

?

```
1  for ( khởi tạo biến; điều kiện; cập nhập giá trị mới cho biến )
2  {
3      // Thực thi khối lệnh khi điều kiện còn đúng
4  }
```

Ví dụ:

?

```
1  // Ví dụ 1:
2  // Tạo một biến x và gán giá trị ban đầu của nó là 0
3  // Điều kiện kiểm tra là x < 5
4  // Nếu x < 5 đúng thì khối lệnh được chạy
5  // Mỗi lần chạy xong khối lệnh giá trị x lại được cập nhập mới tại đây là tăng x lên 1.
6  for (int x = 0; x < 5 ; x = x + 1)
7  {
8      // Làm gì đó tại đây khi x < 5 đúng.
9  }
10
11
12 // Ví dụ 2:
13 // Tạo một biến x và gán giá trị ban đầu của nó là 2
14 // Điều kiện kiểm tra là x < 15
15 // Nếu x < 15 đúng thì khối lệnh được chạy
16 // Mỗi lần chạy xong khối lệnh giá trị x lại được cập nhập mới tại đây là tăng x lên 3.
17 for (int x = 2; x < 15 ; x = x + 3)
18 {
19     // Làm gì đó tại đây khi x < 15 đúng.
20 }
```

ForLoopExample.cs

?

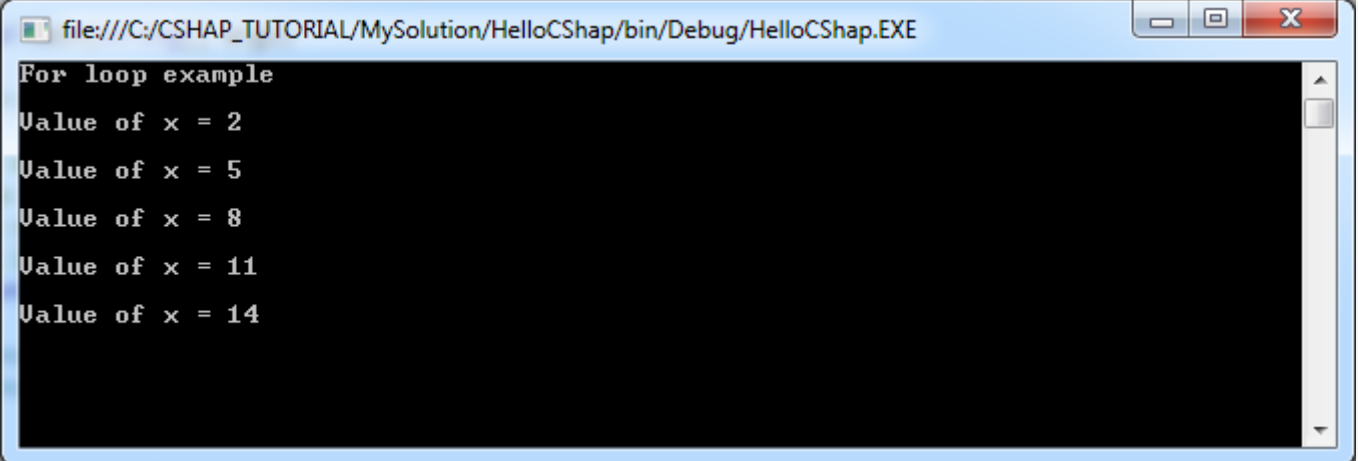
```
1  using System;
2  using System.Collections.Generic;
```

```

3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace HelloCSharp
8  {
9      class ForLoopExample
10     {
11         static void Main(string[] args)
12         {
13             Console.WriteLine("For loop example");
14
15             // Tạo một biến x và gán giá trị ban đầu của nó là 2
16             // Điều kiện kiểm tra là x < 15
17             // Nếu x < 15 đúng thì khối lệnh được chạy
18             // Mỗi lần chạy xong khối lệnh giá trị x lại được cập nhập mới tại đây là tăng x lên 3.
19             for (int x = 2; x < 15; x = x + 3)
20             {
21                 Console.WriteLine( );
22                 Console.WriteLine("Value of x = {0}", x);
23             }
24
25             Console.ReadLine();
26         }
27     }
28 }

```

Kết quả chạy ví dụ:



```

file:///C:/CSHAP_TUTORIAL/MySolution/HelloCSharp/bin/Debug/HelloCSharp.EXE
For loop example
Value of x = 2
Value of x = 5
Value of x = 8
Value of x = 11
Value of x = 14

```

10.2- Vòng lặp while

Cú pháp WHILE:

```

?
1  while (điều kiện)
2  {
3      // Trong khi điều kiện đúng thì thực thi khối lệnh.
4  }

```

Ví dụ:

```

?
1  // Khai báo một biến x
2  int x = 2;
3

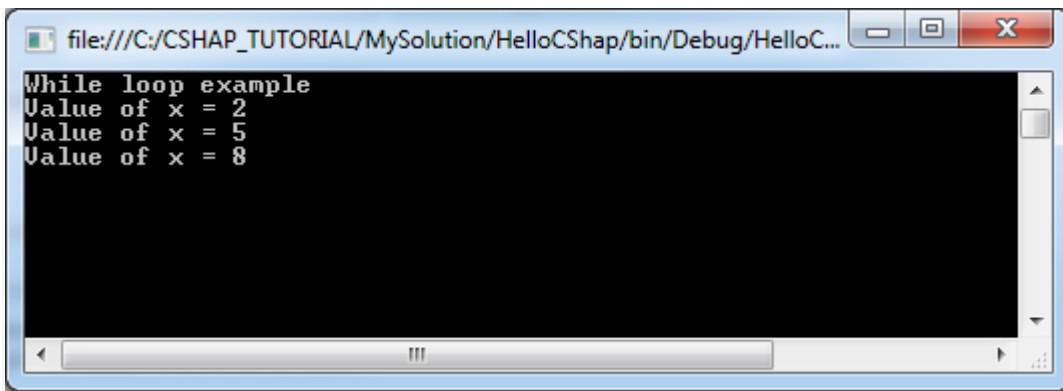
```

```
4 while ( x < 10)
5 {
6     // Làm gì đó tại đây khi x < 10 còn đúng.
7     // ....
8     // Cập nhật giá trị mới cho biến x.
9     x = x + 3;
10 }
```

WhileLoopExample.cs

```
?
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6
7 namespace HelloCSharp
8 {
9     class WhileLoopExample
10    {
11        static void Main(string[] args)
12        {
13
14            Console.WriteLine("While loop example");
15
16
17            // Tạo một biến x và gán giá trị ban đầu của nó là 2
18            int x = 2;
19
20            // Điều kiện kiểm tra là x < 10
21            // Nếu x < 10 đúng thì khối lệnh được chạy.
22            while (x < 10)
23            {
24                Console.WriteLine("Value of x = {0}", x);
25
26                x = x + 3;
27            }
28
29            Console.ReadLine();
30        }
31    }
32 }
```

Kết quả chạy ví dụ:



```
file:///C:/CSHAP_TUTORIAL/MySolution/HelloCSharp/bin/Debug/HelloC...
While loop example
Value of x = 2
Value of x = 5
Value of x = 8
```

10.3- Vòng lặp do-while

Cú pháp của vòng lặp DO-WHILE

```
?
1 // Đặc điểm của vòng lặp DO-WHILE là nó sẽ thực khi khối lệnh ít nhất 1 lần.
2 // Mỗi lần chạy xong khối lệnh nó lại kiểm tra điều kiện xem có thực thi tiếp không.
3 do
4 {
5 // Làm gì đó tại đây
6 // Sau đó mới kiểm tra tiếp điều kiện xem có tiếp tục chạy khối lệnh này nữa hay không.
7 } while ( điều kiện); // Chú ý: cần có dấu chấm phẩy tại đây.
```

DoWhileLoopExample.cs

```
?
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6
7 namespace HelloCSharp
8 {
9     class DoWhileLoopExample
10    {
11        static void Main(string[] args)
12        {
13
14            Console.WriteLine("Do-While loop example");
15
16            // Tạo một biến x và gán giá trị ban đầu của nó là 2
17            int x = 2;
18
19            // Thực hiện khối lệnh.
20            // Sau đó kiểm tra điều kiện xem có thực hiện khối lệnh nữa không.
21            do
22            {
23                Console.WriteLine("Value of x = {0}", x);
24
25                x = x + 3;
26
27            } while (x < 10); // Chú ý: Cần có dấu chấm phẩy tại đây.
28
29            Console.ReadLine();
```

```

30     }
31 }
32 }

```

Kết quả chạy ví dụ:

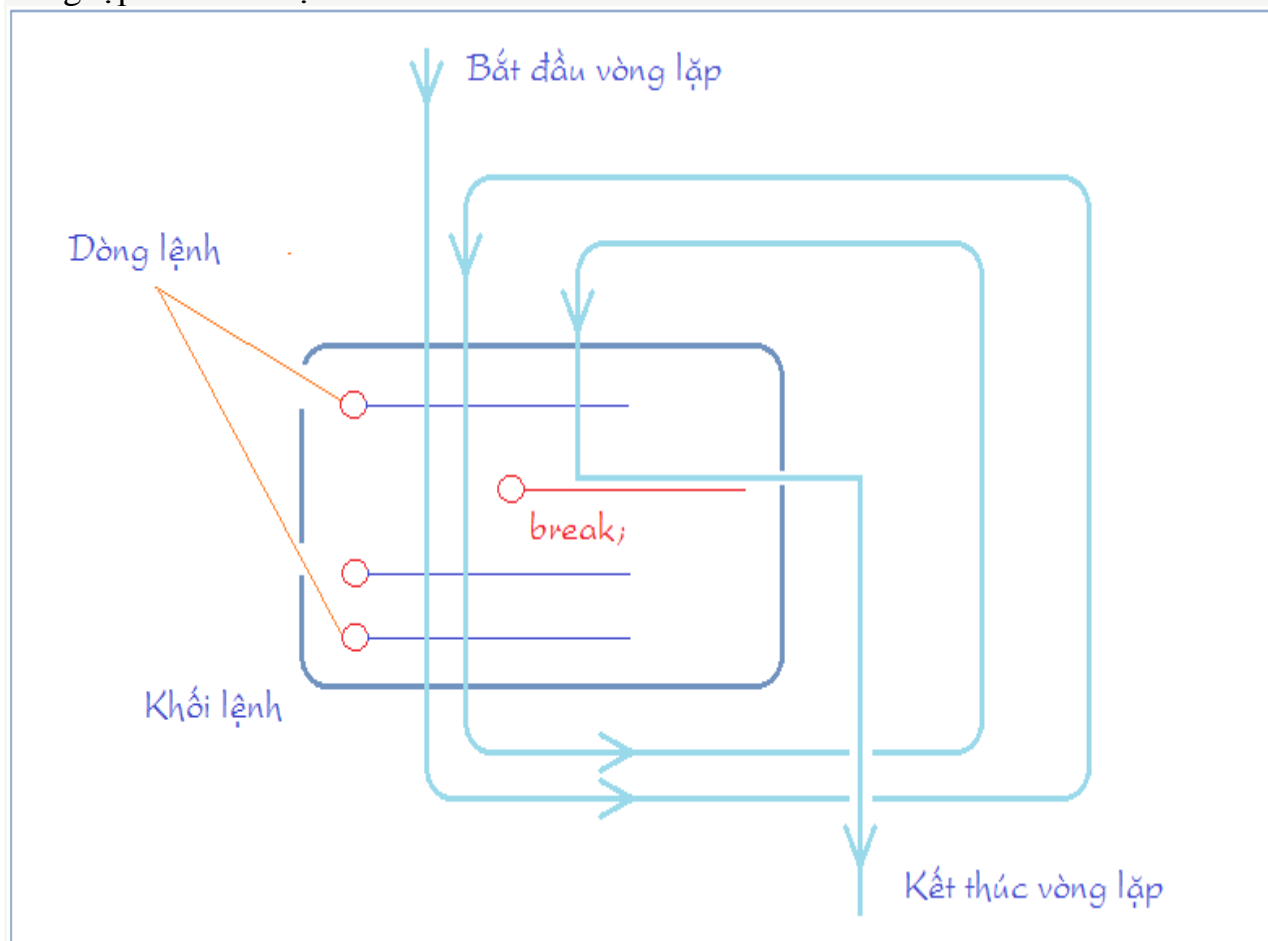
```

file:///C:/CSHAP_TUTORIAL/MySolution/HelloCSharp/bin/...
Do-While loop example
Value of x = 2
Value of x = 5
Value of x = 8

```

10.4- 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.



LoopBreakExample.cs

```

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace HelloCSharp
8  {
9      class LoopBreakExample
10     {
11         static void Main(string[] args)

```

```

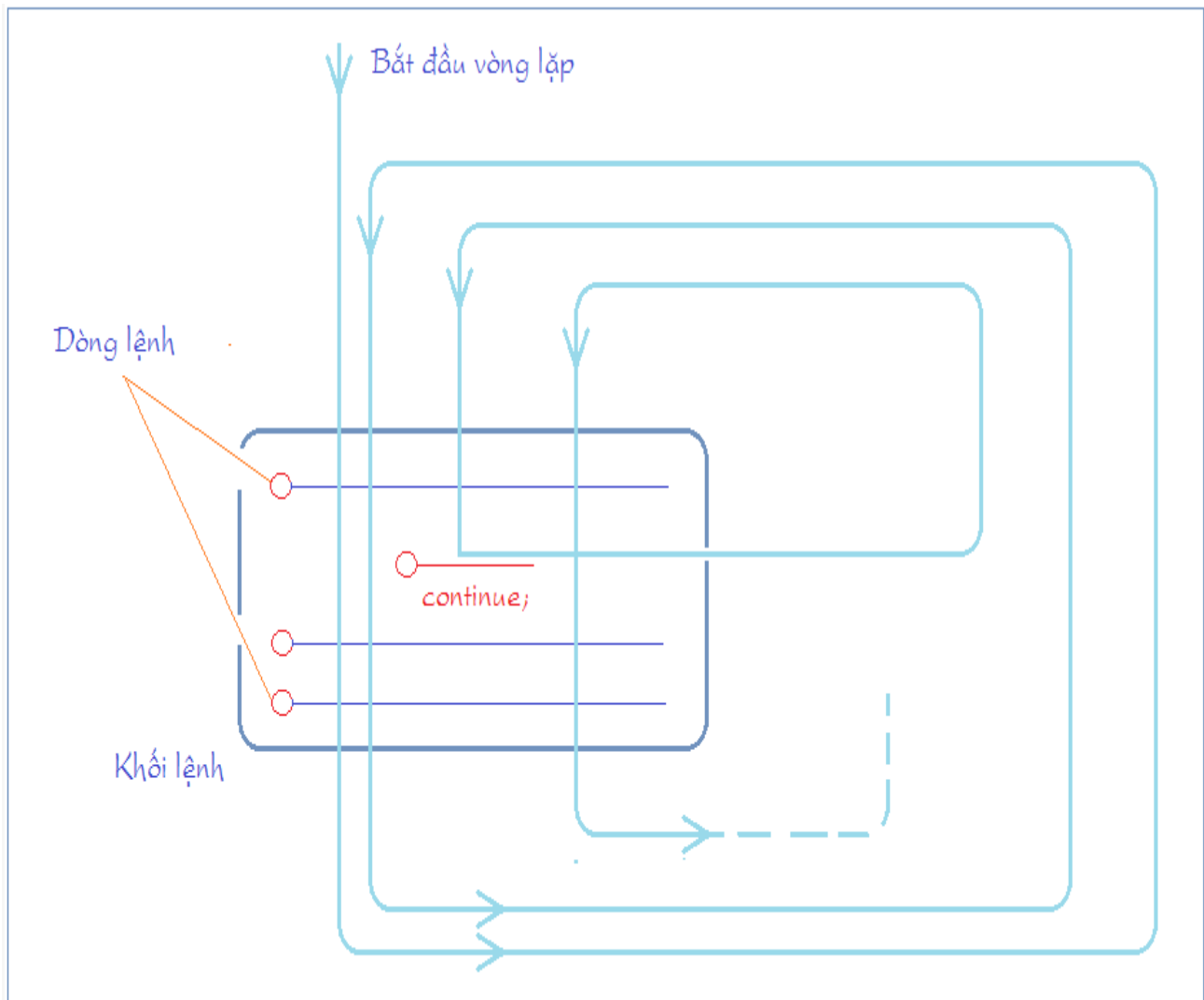
12    {
13        Console.WriteLine("Break example");
14
15
16
17        // Tạo một biến x và gán giá trị ban đầu của nó là 2
18        int x = 2;
19
20        while (x < 15)
21        {
22
23            Console.WriteLine("-----\n");
24            Console.WriteLine("x = {0}", x);
25
26            // Kiểm tra nếu x = 5 thì thoát ra khỏi vòng lặp.
27            if (x == 5)
28            {
29                break;
30            }
31            // Tăng x lên 1 (Viết ngắn gọn cho x = x + 1;).
32            x++;
33            Console.WriteLine("x after ++ = {0}", x);
34
35        }
36
37        Console.ReadLine();
38    }
39 }
40 }

```

Kết quả chạy ví dụ:

10.5- 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.



LoopContinueExample.cs

```

2
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace HelloCSharp
8  {
9      class LoopContinueExample
10     {
11         static void Main(string[] args)
12         {
13
14             Console.WriteLine("Continue example");
15
16
17             // Tạo một biến x và gán giá trị ban đầu của nó là 2
18             int x = 2;
19
20             while (x < 7)
21             {
22
23                 Console.WriteLine("-----\n");

```



```

24     Console.WriteLine("x = {0}", x);
25
26     // % là phép chia lấy số dư
27     // Nếu x chẵn, thì bỏ qua các dòng lệnh phía dưới
28     // của continue, tiếp tục vòng lặp mới (nếu có).
29     if (x % 2 == 0)
30     {
31         // Tăng x lên 1 (Viết ngắn gọn cho x = x + 1;).
32         x++;
33         continue;
34     }
35     else
36     {
37         // Tăng x lên 1 (Viết ngắn gọn cho x = x + 1;).
38         x++;
39     }
40     Console.WriteLine("x after ++ = {0}", x);
41
42 }
43
44 Console.ReadLine();
45 }
46 }
47 }

```

Kết quả chạy ví dụ:

```

file:///C:/CSHAP_TUTORIAL/MySolution/HelloCShap/bin/Debug/HelloCShap.EXE
Continue example
x = 2
x = 3
x after ++ = 4
x = 4
x = 5
x after ++ = 6
x = 6

```

11- Mảng trong C#



11.1- Mảng một chiều

Đây là hình minh họa về mảng một chiều có 5 phần tử, các phần tử được đánh chỉ số từ 0 tới 4.

```
int[] a;
```



Cú pháp khai báo mảng một chiều:

```
?  
1 // Cách 1:  
2 // Khai báo một mảng các số int, chỉ rõ các phần tử.  
3 int[] years = { 2001, 2003, 2005, 1980, 2003 };  
4  
5 // Cách 2:  
6 // Khai báo một mảng các số float, chỉ rõ số phần tử.  
7 // (3 phần tử).  
8 float[] salaries = new float[3];
```

ArrayExample1.cs

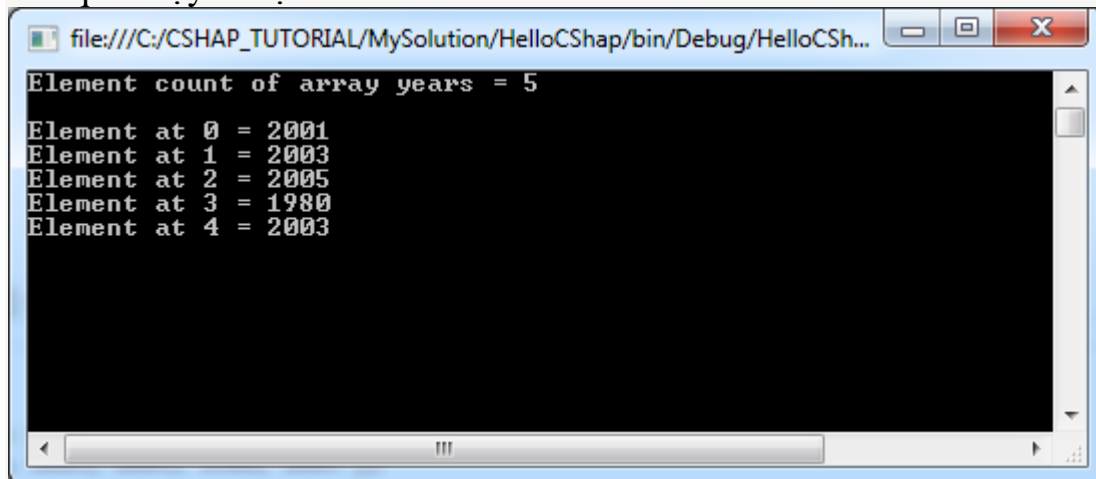
```
?  
1 using System;  
2 using System.Collections.Generic;  
3 using System.Linq;  
4 using System.Text;  
5 using System.Threading.Tasks;  
6  
7 namespace HelloCSharp  
8 {  
9     class ArrayExample1  
10    {  
11        static void Main(string[] args)  
12        {  
13  
14            // Cách 1:  
15            // Khai báo một mảng, gán luôn các giá trị.  
16            int[] years = { 2001, 2003, 2005, 1980, 2003 };  
17  
18            // Length là một thuộc tính của mảng, nó trả về số phần tử của mảng.  
19            Console.WriteLine("Element count of array years = {0} \n", years.Length);  
20  
21            // Sử dụng vòng lặp for để in ra các phần tử của mảng.  
22            for (int i = 0; i < years.Length; i++) {  
23                Console.WriteLine("Element at {0} = {1}", i, years[i]);  
24            }  
25  
26  
27            // Cách 2:  
28            // Khai báo một mảng có 3 phần tử.  
29            float[] salaries = new float[3];  
30  
31            // Gán các giá trị cho các phần tử.
```

```

32     salaries[0] = 1000;
33     salaries[1] = 1200;
34     salaries[2] = 1100;
35
36     Console.ReadLine();
37 }
38 }
39 }

```

Kết quả chạy ví dụ:



11.2- Mảng hai chiều

Đây là hình minh họa một mảng 2 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:

```

2
1 // Khai báo mảng 2 chiều chỉ định các phần tử.
2 // 3 hàng & 5 cột
3 int[,] a = new int[,] {
4     {1, 2, 3, 4, 5},
5     {0, 3, 4, 5, 7},
6     {0, 3, 4, 0, 0}
7 };
8
9 // Khai báo một mảng 2 chiều, số dòng 3, số cột 5.
10 // Các phần tử chưa được gán giá trị.
11 int[,] a = new int[3,5];

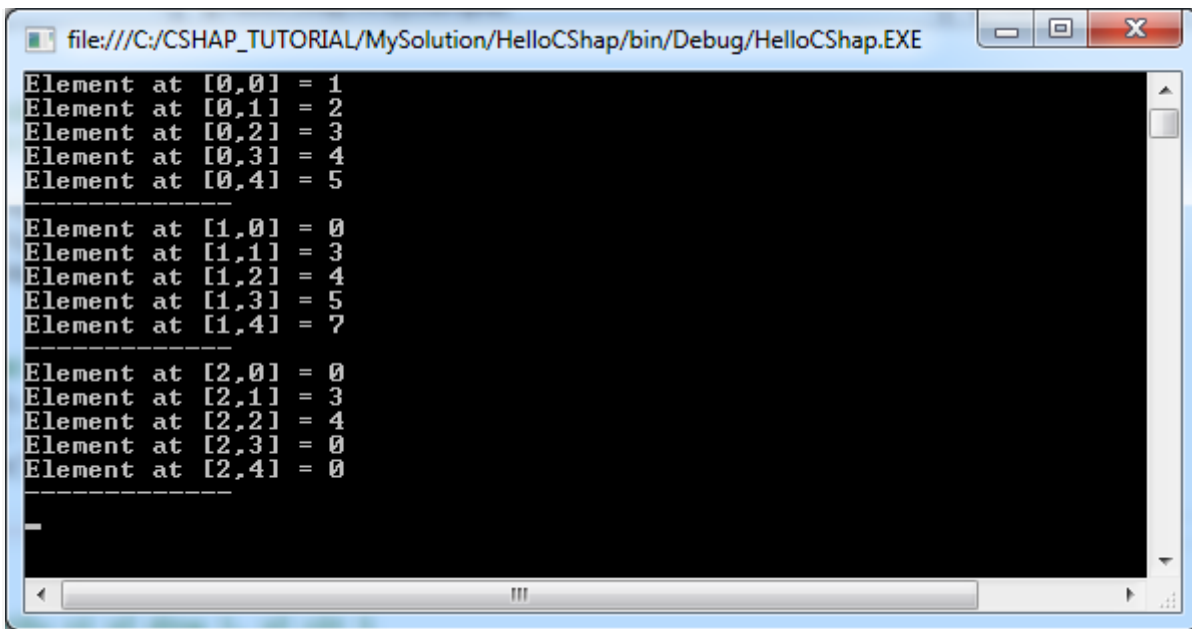
```

ArrayExample2.cs

?

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace HelloCSharp
8  {
9      class ArrayExample2
10     {
11         static void Main(string[] args)
12         {
13
14             // Khai báo một mảng 2 chiều
15             // Khởi tạo sẵn các giá trị.
16             int[,] a = {
17                 { 1, 2, 3, 4, 5 },
18                 { 0, 3, 4, 5, 7 },
19                 { 0, 3, 4, 0, 0 }
20             };
21
22             // Sử dụng vòng lặp for để in ra các phần tử của mảng.
23             for (int row = 0; row < 3; row++) {
24                 for (int col = 0; col < 5; col++) {
25                     Console.WriteLine("Element at [{0},{1}] = {2}", row, col, a[row,col]);
26                 }
27                 Console.WriteLine("-----");
28             }
29
30             // Khai báo một mảng 2 chiều có số dòng 3, số cột 5
31             // Các phần tử chưa được gán giá trị.
32             int[,] b = new int[3, 5];
33
34             Console.ReadLine();
35         }
36     }
37 }
```

Kết quả chạy ví dụ:



```
file:///C:/CSHAP_TUTORIAL/MySolution/HelloCSharp/bin/Debug/HelloCSharp.EXE
Element at [0,0] = 1
Element at [0,1] = 2
Element at [0,2] = 3
Element at [0,3] = 4
Element at [0,4] = 5
-----
Element at [1,0] = 0
Element at [1,1] = 3
Element at [1,2] = 4
Element at [1,3] = 5
Element at [1,4] = 7
-----
Element at [2,0] = 0
Element at [2,1] = 3
Element at [2,2] = 4
Element at [2,3] = 0
Element at [2,4] = 0
-----
```

11.3- Mảng của mảng

ArrayOfArrayExample.cs

```
?
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace HelloCSharp
8  {
9      class ArrayOfArrayExample
10     {
11         static void Main(string[] args)
12         {
13
14             // Khai báo một mảng 3 phần tử.
15             // Mỗi phần tử là một mảng khác.
16             string[][] teams = new string[3][];
17
18             string[] mu = { "Beckham", "Giggs" };
19             string[] asenal = { "Oezil", "Szczesny", "Walcott" };
20             string[] chelsea = { "Oscar", "Hazard", "Drogba" };
21
22             teams[0] = mu;
23             teams[1] = asenal;
24             teams[2] = chelsea;
25
26             // Sử dụng vòng lặp for để in ra các phần tử của mảng.
27             for (int row = 0; row < teams.Length; row++)
28             {
29                 for (int col = 0; col < teams[row].Length ; col++)
30                 {
31                     Console.WriteLine("Element at [{0}],[{1}] = {2}", row, col, teams[row][col]);
32                 }
33                 Console.WriteLine("-----");
34             }
35         }
36     }
37 }
```

```

34     }
35
36
37     Console.ReadLine();
38     }
39 }
40 }

```

Kết quả chạy ví dụ

```

file:///C:/CSHAP_TUTORIAL/MySolution/HelloCShap/bin/Debug/HelloCShap.EXE
Element at [0],[0] = Beckham
Element at [0],[1] = Giggs
-----
Element at [1],[0] = Oezil
Element at [1],[1] = Szczesny
Element at [1],[2] = Walcott
-----
Element at [2],[0] = Oscar
Element at [2],[1] = Hazard
Element at [2],[2] = Drogba

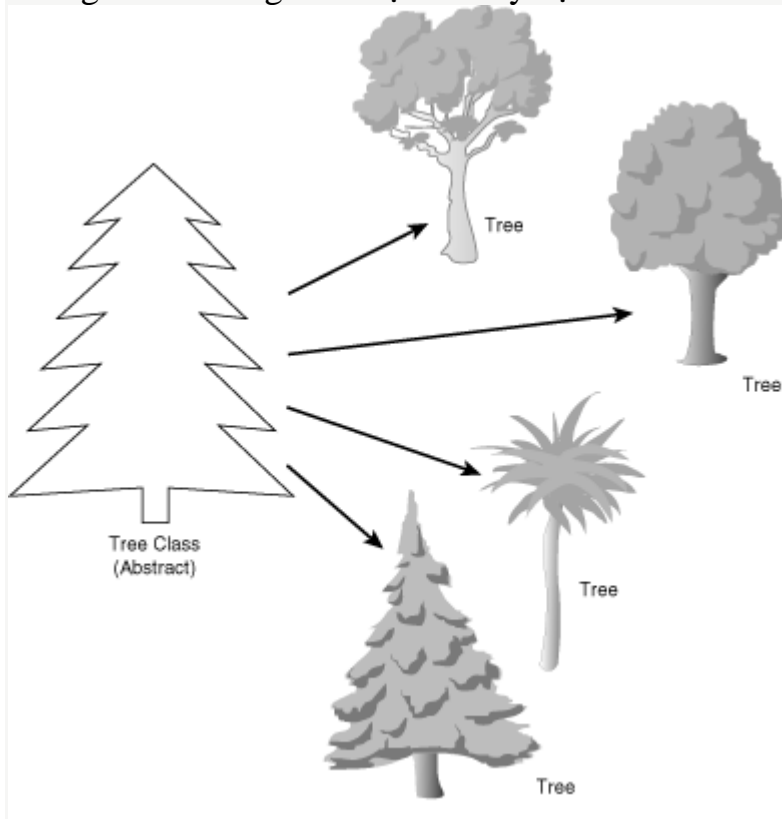
```

12- Class, đối tượng và cấu tử

Bạn cần có sự phân biệt giữa 3 khái niệm:

- Class
- Cấu tử (Constructor)
- Đối tượng (Instance)

Khi chúng ta nói về Cây, nó là một thứ gì đó trừu tượng, nó là một lớp (class). Nhưng khi chúng ta chỉ thẳng vào một cái cây cụ thể thì lúc đó đã rõ ràng và đó là đối tượng (instance).



Hoặc khi chúng ta nói về người (Person) thì đó cũng trừu tượng, nó là một lớp. Nhưng khi chỉ thẳng vào bạn hoặc tôi thì đó là 2 đối tượng khác nhau, cùng thuộc lớp người.



Thomas Edison



Bill Gates

Person (Class)

Person.cs

```
?  
1  using System;  
2  using System.Collections.Generic;  
3  using System.Linq;  
4  using System.Text;  
5  using System.Threading.Tasks;  
6  
7  namespace HelloCSharp  
8  {  
9      class Person  
10     {  
11         // Đây là một trường (Field)  
12         // Lưu trữ tên người.  
13         public string Name;  
14  
15         // Đây là một cấu tử, còn gọi là phương thức khởi tạo (Constructor)  
16         // Dùng nó để khởi tạo đối tượng.  
17         // Cấu tử này có một tham số.  
18         // Cấu tử luôn có tên giống tên class.  
19         public Person(string personName)  
20         {  
21             // Gán giá trị từ tham số vào cho trường name.  
22             this.Name = personName;  
23         }  
24  
25         // Đây là một phương thức trả về kiểu string.  
26         public string GetName()  
27         {  
28             return this.Name;  
29         }  
30     }  
31 }
```

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.

PersonTest.cs

```
?
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7
8  namespace HelloCSharp
9  {
10     class PersonTest
11     {
12         static void Main(string[] args)
13         {
14
15             // Tạo một đối tượng từ class Person
16             // Khởi tạo đối tượng này từ cấu tử của class Person
17             // Cụ thể là Edison
18             Person edison = new Person("Edison");
19
20             // Class Person có hàm getName()
21             // Sử dụng đối tượng để gọi hàm getName():
22             String name = edison.GetName();
23             Console.WriteLine("Person 1: " + name);
24
25             // Tạo một đối tượng từ class Person.
26             // Khởi tạo đối tượng này từ cấu tử của class Person
27             // Cụ thể là Bill Gates
28             Person billGate = new Person("Bill Gates");
29
30             // Class Person có trường name (public)
31             // Sử dụng đối tượng để tham chiếu tới nó.
32             String name2 = billGate.Name;
33             Console.WriteLine("Person 2: " + name2);
34
35             Console.ReadLine();
36         }
37     }
38 }
```

Kết quả chạy ví dụ:

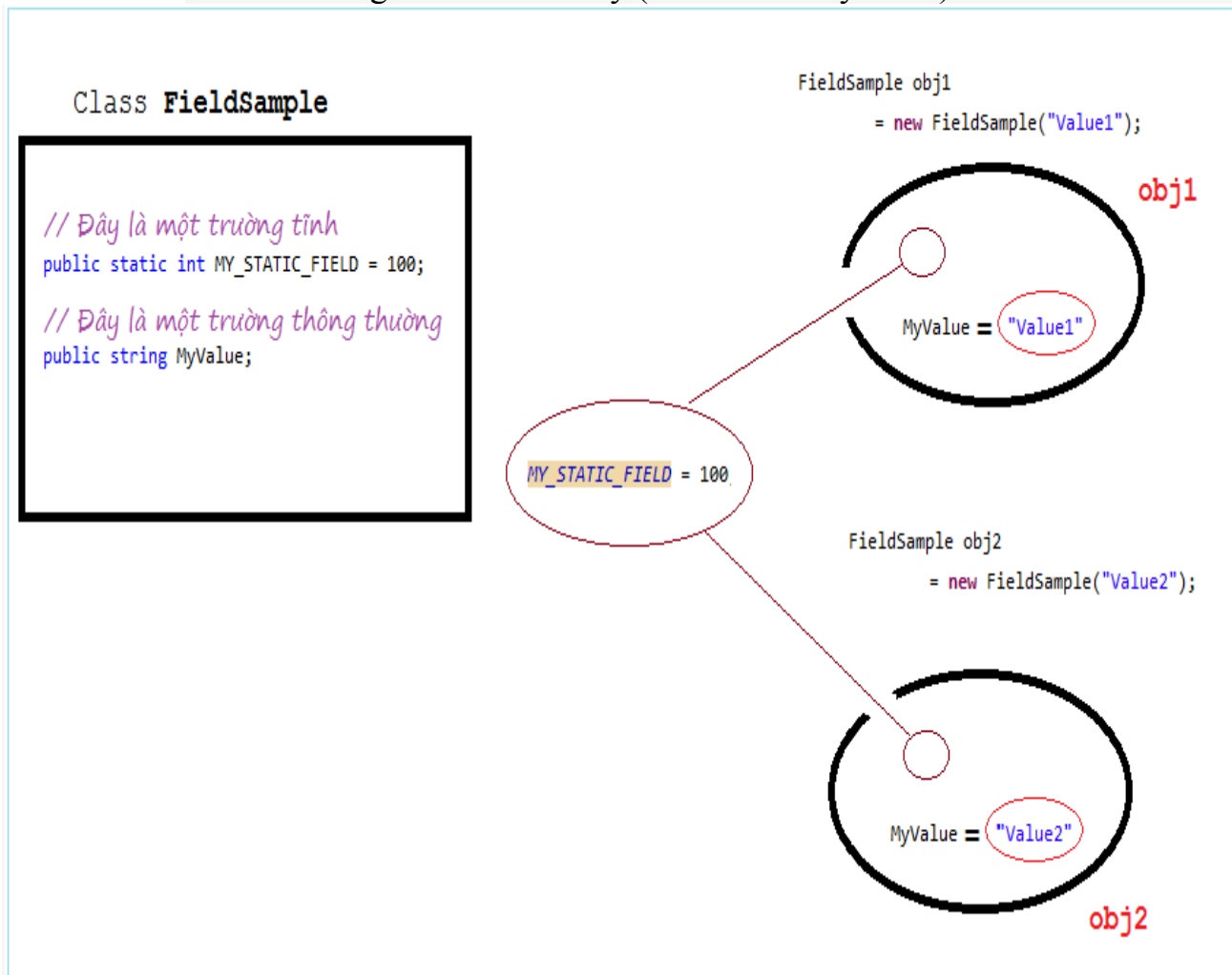

```
file:///C:/CSHAP_TUTORIAL/MySolution/HelloCSharp/bin/De...
Person 1: Edison
Person 2: Bill Gates
```

13- 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)



FieldSample.cs

```
?
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6
7 namespace HelloCSharp
```

```

8  {
9  class FieldSample
10 {
11     // Đây là một trường tĩnh.
12     public static int MY_STATIC_FIELD = 100;
13
14     // Đây là một trường thông thường.
15     public string MyValue;
16
17
18     // Cấu tử khởi tạo đối tượng FieldSample.
19     public FieldSample(string value)
20     {
21         this.MyValue = value;
22     }
23 }
24 }

```

FieldSampleTest.cs

```

?
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace HelloCSharp
8  {
9  class FieldSampleTest
10 {
11     static void Main(string[] args)
12     {
13         // In ra giá trị của trường static.
14         // Với các trường tĩnh, bạn phải truy cập tới nó thông qua class.
15         Console.WriteLine("FieldSample.MY_STATIC_FIELD= {0}", FieldSample.MY_STATIC_FIELD);
16
17         // Bạn có thể thay đổi giá trị của trường tĩnh.
18         FieldSample.MY_STATIC_FIELD = 200;
19
20         Console.WriteLine(" ----- ");
21
22         // Tạo đối tượng thứ nhất.
23         FieldSample obj1 = new FieldSample("Value1");
24
25         // Các trường không tĩnh bạn phải truy cập thông qua đối tượng.
26         Console.WriteLine("obj1.MyValue= {0}", obj1.MyValue);
27
28
29         // Tạo đối tượng thứ 2:
30         FieldSample obj2 = new FieldSample("Value2");
31

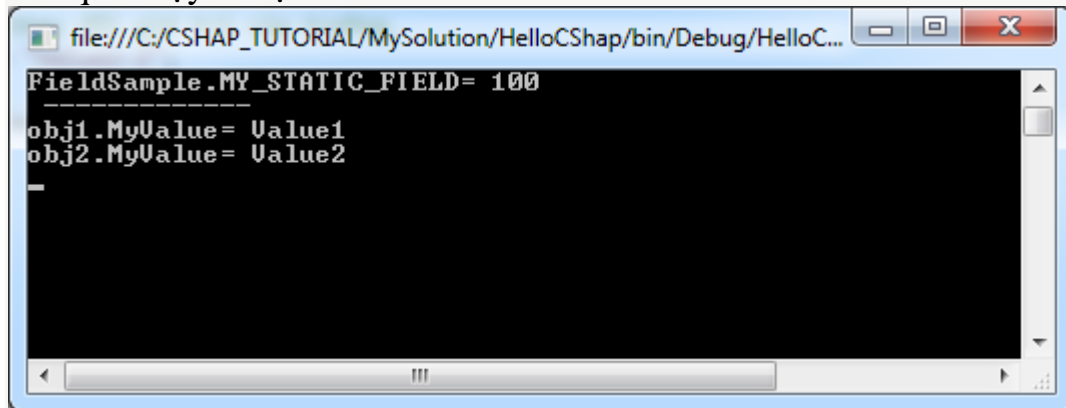
```

```

32    //
33    Console.WriteLine("obj2.MyValue= {0}" , obj2.MyValue);
34
35    // Bạn có thể thay đổi giá trị của trường.
36    obj2.MyValue = "Value2-2";
37
38    Console.ReadLine();
39 }
40
41 }
42 }

```

Kết quả chạy ví dụ:



Ví dụ readonly & static readonly.

ConstFieldExample.cs

```

?
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace HelloCSharp
8  {
9      class ConstFieldExample
10     {
11         // Một trường hằng số, giá trị của nó được xác định sẵn tại thời điểm biên dịch.
12         // Trường const không cho phép gán giá trị mới.
13         // Chú ý: Trường const (Đã là static).
14         public const int MY_VALUE = 100;
15
16         // Một trường tĩnh và readonly.
17         // Giá trị của nó có thể gán sẵn, hoặc chỉ được gán 1 lần trong cấu tử tĩnh.
18         public static readonly DateTime INIT_DATE_TIME1 = DateTime.Now;
19
20         // Một trường readonly.
21         // Giá trị của nó có thể gán sẵn, hoặc chỉ được gán một lần tại cấu tử (không tĩnh).
22         public readonly DateTime INIT_DATE_TIME2 ;
23
24         public ConstFieldExample()
25         {
26             // Gán giá trị cho trường readonly (Chỉ được phép gán 1 lần).

```

```

27     INIT_DATE_TIME2 = DateTime.Now;
28     }
29 }
30 }

```

14- 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).

MethodSample.cs

```

?
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace HelloCSharp
8  {
9      class MethodSample
10     {
11         public string text = "Some text";
12
13         // Cấu tử mặc định.
14         // Nghĩa là cấu tử không có tham số.
15         public MethodSample()
16         {
17
18         }
19
20         // Đây là một phương thức trả về kiểu String.
21         // Phương thức này không có tham số
22         public string GetText()
23         {
24             return this.text;
25         }
26
27         // Đây là một phương thức có 1 tham số String.
28         // Phương thức này trả về void (Hay gọi là ko trả về gì)
29         public void SetText(string text)
30         {
31             // this.text tham chiếu tới trường text.
32             // phân biệt với tham số text.
33             this.text = text;
34         }
35
36         // Đây là một phương thức tĩnh.
37         // Trả về kiểu int, có 3 tham số.
38         public static int Sum(int a, int b, int c)
39         {

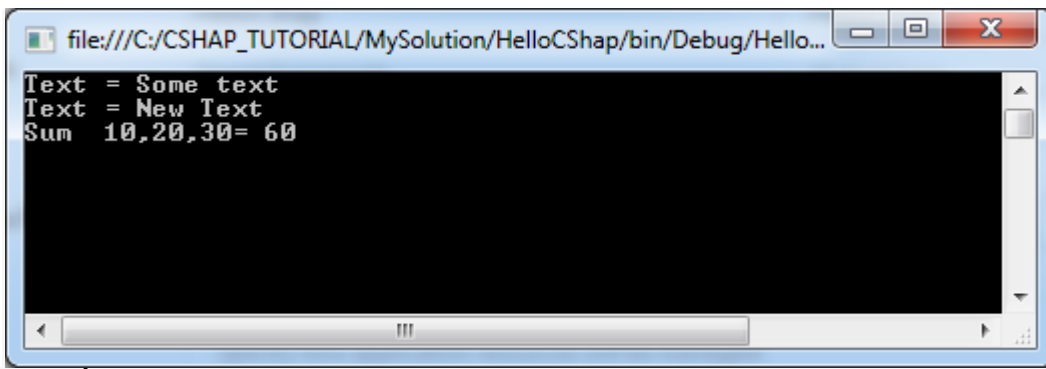
```

```
40         int d = a + b + c;
41         return d;
42     }
43 }
44 }
```

MethodSampleTest.cs

```
?
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace HelloCSharp
8  {
9      class MethodSampleTest
10     {
11         static void Main(string[] args)
12         {
13             // Tạo đối tượng MethodSample
14             MethodSample obj = new MethodSample();
15
16             // Các phương thức không tĩnh cần phải được gọi thông qua đối tượng.
17             // Gọi phương thức GetText()
18             String text = obj.GetText();
19
20             Console.WriteLine("Text = " + text);
21
22             // Các phương thức không tĩnh cần phải được gọi thông qua đối tượng.
23             // Gọi method SetText(String)
24             obj.SetText("New Text");
25
26             Console.WriteLine("Text = " + obj.GetText());
27
28             // Các phương thức tĩnh cần phải được gọi thông qua Class.
29             int sum = MethodSample.Sum(10, 20, 30);
30
31             Console.WriteLine("Sum 10,20,30= " + sum);
32
33             Console.ReadLine();
34         }
35     }
36 }
```

Kết quả chạy ví dụ:



15- 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. Hãy xem một ví dụ minh họa về thừa kế trong **CSharp**:

Animal.cs

```
?
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace HelloCSharp
8  {
9      // Mô phỏng một lớp động vật.
10     class Animal
11     {
12         public Animal()
13         {
14
15         }
16
17         public void Move()
18         {
19             Console.WriteLine("Move ...!");
20         }
21     }
22 }
23 }
```

Cat.cs

```
?
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace HelloCSharp
8  {
9      class Cat : Animal
10     {
11
```

```

12     public void Say()
13     {
14         Console.WriteLine("Meo");
15     }
16
17     // Một method của class Cat.
18     public void Catch()
19     {
20         Console.WriteLine("Catch Mouse");
21     }
22 }
23 }

```

Ant.cs

```

?
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace HelloCSharp
8  {
9      // Con kiến
10     class Ant : Animal
11     {
12     }
13 }

```

AnimalTest.cs

```

?
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace HelloCSharp
8  {
9      class AnimalTest
10     {
11
12         static void Main(string[] args)
13         {
14
15             // Khai báo một đối tượng Cat.
16             Cat tom = new Cat();
17
18             // Kiểm tra xem 'tom' có phải là đối tượng Animal ko.
19             // Kết quả rõ ràng là true.
20             bool isAnimal = tom is Animal;
21

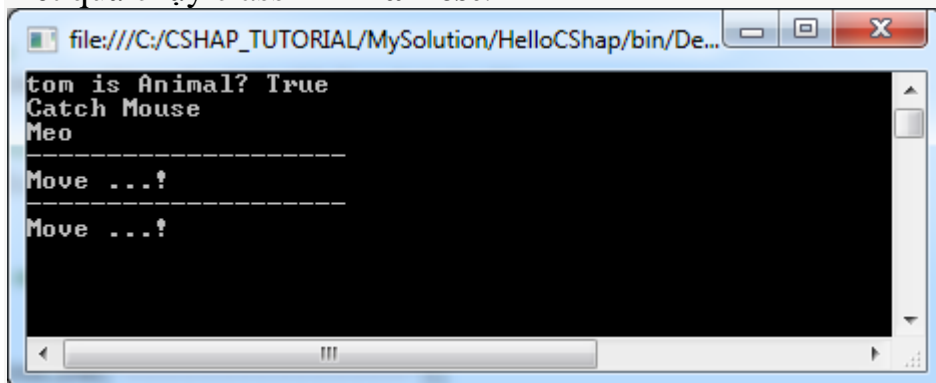
```

```

22         // ==> true
23         Console.WriteLine("tom is Animal? " + isAnimal);
24
25         // Gọi method Catch
26         tom.Catch();
27
28         // ==> Meo
29         // Gọi vào method Say() của Cat.
30         tom.Say();
31
32         Console.WriteLine("-----");
33
34         // Khai báo một đối tượng Animal
35         // Khởi tạo đối tượng thông qua cấu tử của Cat.
36         Animal tom2 = new Cat();
37
38         // Gọi method Move()
39         tom2.Move();
40
41         Console.WriteLine("-----");
42
43         // Thông qua cấu tử của class con, Ant.
44         Ant ant = new Ant();
45
46         // Gọi method Move() thừa kế được từ Animal.
47         ant.Move();
48
49         Console.ReadLine();
50     }
51 }
52 }
53 }

```

Kết quả chạy class **AnimalTest**:



```

file:///C:/CSHAP_TUTORIAL/MySolution/HelloCSharp/bin/De...
tom is Animal? True
Catch Mouse
Meo
-----
Move ...?
-----
Move ...?

```

16- Thừa kế và đa hình trong C#

1- Giới thiệu

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

2- Class, đối tượng và cấu tử



Bạn cần hiểu một cách rạch ròi về class, cấu tử và đối tượng trước khi bắt đầu tìm hiểu quan hệ thừa kế trong CSharp. Chúng ta xem class **Person**, mô tả một con người với các thông tin liên quan.



Person (Class)



Thomas Edison

Name: Thomas Edison
Born Year: 1847
Place Of Birth:



Bill Gates

Name: Bill Gates
Born Year: 1955
Place Of Birth: Seattle,
Washington

Person.cs

```
?  
1  using System;  
2  using System.Collections.Generic;  
3  using System.Linq;  
4  using System.Text;  
5  using System.Threading.Tasks;  
6  
7  namespace InheritancePolymorphism  
8  {  
9      class Person  
10     {  
11         // Trường name - thông tin tên người  
12         public String Name;  
13  
14         // Trường bornYear - thông tin năm sinh  
15         public int BornYear;  
16  
17         // Nơi sinh  
18         public String PlaceOfBirth;  
19  
20  
21         // Cấu tử 3 tham số. Mục đích nhằm để khởi tạo các giá trị cho các trường của Person.  
22         // Chỉ định rõ tên, năm sinh, nơi sinh.  
23         public Person(String Name, int BornYear, String PlaceOfBirth)  
24         {  
25             this.Name = Name;  
26             this.BornYear = BornYear;
```

```

27         this.PlaceOfBirth = PlaceOfBirth;
28     }
29
30     // Cấu tử 2 tham số. Mục đích khởi tạo giá trị cho 2 trường tên và năm sinh cho Person.
31     // Nơi sinh không được khởi tạo.
32     public Person(String Name, int BornYear)
33     {
34         this.Name = Name;
35         this.BornYear = BornYear;
36     }
37
38     }
39 }

```

PersonDemo.cs

```

?
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace InheritancePolymorphism
8  {
9      class PersonDemo
10     {
11
12         static void Main(string[] args)
13         {
14
15             // Đối tượng: Thomas Edison.
16             // Khởi tạo theo cấu tử 2 tham số.
17             Person edison = new Person("Thomas Edison", 1847);
18
19             Console.WriteLine("Info:");
20             Console.WriteLine("Name: " + edison.Name);
21             Console.WriteLine("Born Year: " + edison.BornYear);
22             Console.WriteLine("Place Of Birth: " + edison.PlaceOfBirth);
23
24             // Đối tượng: Bill Gates
25             // Khởi tạo theo cấu tử 3 tham số.
26             Person billGates = new Person("Bill Gate", 1955, "Seattle, Washington");
27
28             Console.WriteLine("-----");
29
30             Console.WriteLine("Info:");
31             Console.WriteLine("Name: " + billGates.Name);
32             Console.WriteLine("Born Year: " + billGates.BornYear);
33             Console.WriteLine("Place Of Birth: " + billGates.PlaceOfBirth);
34
35             Console.ReadLine();

```

```
36     }  
37 }  
38 }
```

Kết quả chạy class **PersonDemo**:

```
file:///C:/CSHAP_TUTORIAL/MySolution/InheritancePolymorphism/...  
Info:  
Name: Thomas Edison  
Born Year: 1847  
Place Of Birth:  
-----  
Info:  
Name: Bill Gate  
Born Year: 1955  
Place Of Birth: Seattle, Washington  
-
```

Phân biệt Class, cấu tử 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.

Cấu tử - Hoặc còn gọi là "**Phương thức khởi tạo**"

- Cấu tử luôn có tên giống tên class
- Một class có một hoặc nhiều cấu tử.
- Cấu tử có hoặc không có tham số, cấu tử không có tham số còn gọi là cấu tử mặc định.
- Cấu tử 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 cấu tử là phương thức đặc biệt để tạo ra đối tượng, cấu tử 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 cấu tử.

```
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;
    }

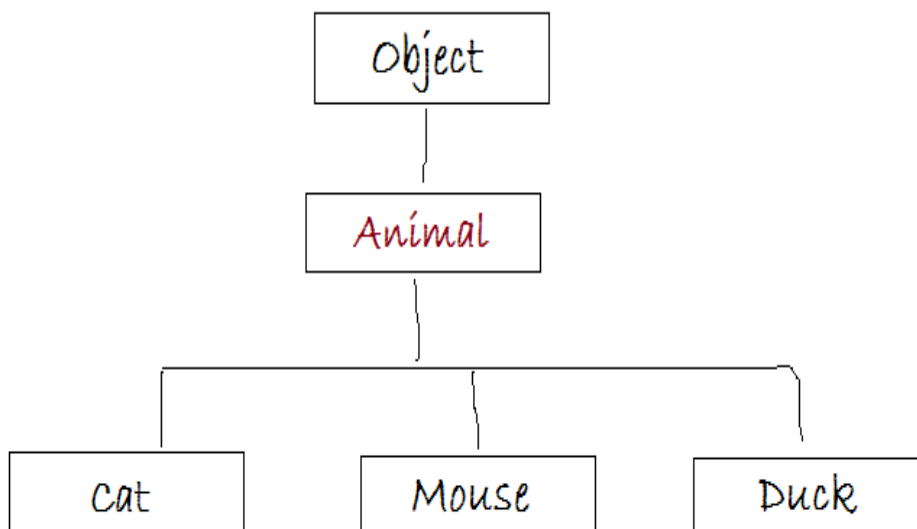
    public Person(String Name, int BornYear)
    {
        this.Name = Name;
        this.BornYear = BornYear;
    }
}
```

3- Thừa kế trong CSharp



Chúng ta cần một vài class tham gia vào minh họa.

- **Animal**: Class mô phỏng một lớp Động vật.
- **Duck**: Class mô phỏng lớp vịt, là một class con của **Animal**.
- **Cat**: Class mô phỏng lớp mèo, là một class con của **Animal**
- **Mouse**: Class mô phỏng lớp chuột, là một class con của **Animal**.



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.

Hãy xem ví dụ:

Animal.cs

?

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace InheritancePolymorphism
8  {
9      public abstract class Animal
10     {
11         // Trường Name.
12         // Tên, ví dụ Mèo Tom, Chuột Jerry.
13         public string Name;
14
15         // Cấu tử mặc định
16         public Animal()
17         {
18             Console.WriteLine("- Animal()");
19         }
20
21         public Animal(string Name)
22         {
23             // Gán giá trị cho trường Name.
24             this.Name = Name;
25             Console.WriteLine("- Animal(string)");
26         }
27
28         // Phương thức mô tả hành vi di chuyển của con vật.
29         // virtual: Nói rằng phương thức này có thể ghi đè tại các class con.
30         public virtual void Move()
31         {
32             Console.WriteLine("Animal Move");
33         }
34
35         public void Sleep()
36         {
37             Console.WriteLine("Sleep");
38         }
39     }
40 }
41 }
```

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

Cat.cs

?

```
1  using System;
```

```

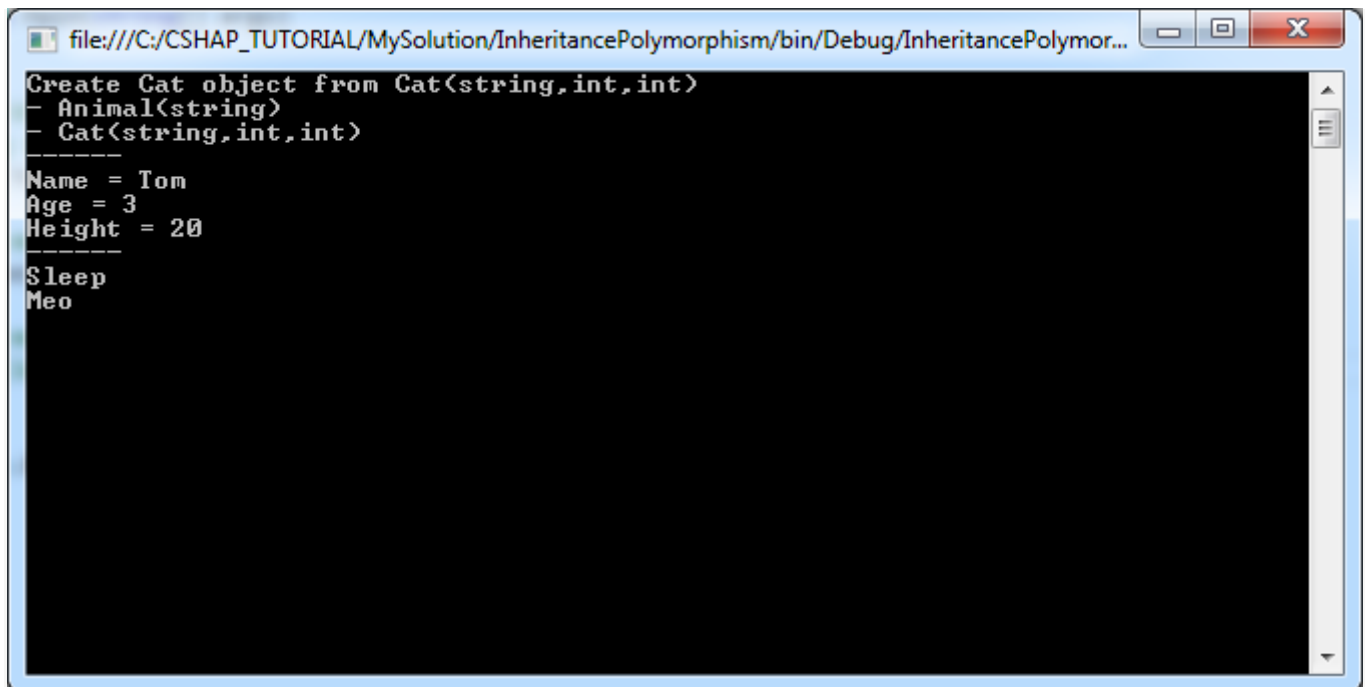
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace InheritancePolymorphism
8  {
9      public class Cat : Animal
10     {
11
12         public int Age;
13         public int Height;
14
15         // Đây là cấu tử 3 tham số của Cat.
16         // Sử dụng :base(name) để gọi đến cấu tử của class cha: Animal(string).
17         // Các trường của class cha sẽ được gán giá trị.
18         // Sau đó các trường của class này mới được gán giá trị.
19         public Cat(string name, int Age, int Height)
20             : base(name)
21         {
22             this.Age = Age;
23             this.Height = Height;
24             Console.WriteLine("- Cat(string,int,int)");
25         }
26
27
28         // Cấu tử này gọi tới cấu tử mặc định (Không tham số) của class cha.
29         public Cat(int Age, int Height)
30             : base()
31         {
32             this.Age = Age;
33             this.Height = Height;
34             Console.WriteLine("- Cat(int,int)");
35         }
36
37         public void Say()
38         {
39             Console.WriteLine("Meo");
40         }
41
42
43         // Viết lại hành vi di chuyển của loài Mèo.
44         // Ghi đè phương thức Move() của class cha (Animal).
45         public override void Move()
46         {
47             Console.WriteLine("Cat Move ...");
48         }
49     }
50 }

```

?

```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6
7 namespace InheritancePolymorphism
8 {
9     class CatTest
10    {
11        static void Main(string[] args)
12        {
13
14            Console.WriteLine("Create Cat object from Cat(string,int,int)");
15
16            // Khởi tạo một đối tượng Cat từ cấu tử 3 tham số.
17            // Trường Name của Animal sẽ được gán giá trị "Tom".
18            // Trường Age của Cat sẽ được gán giá trị 3
19            // Trường Height của Cat sẽ được gán giá trị 20.
20            Cat tom = new Cat("Tom",3, 20);
21
22            Console.WriteLine("-----");
23
24            Console.WriteLine("Name = {0}", tom.Name);
25            Console.WriteLine("Age = {0}", tom.Age);
26            Console.WriteLine("Height = {0}", tom.Height);
27
28            Console.WriteLine("-----");
29
30            // Gọi method thừa kế từ Animal
31            tom.Move();
32
33            // Gọi method Say() (của class Cat)
34            tom.Say();
35
36
37            Console.ReadLine();
38        }
39    }
40 }
```

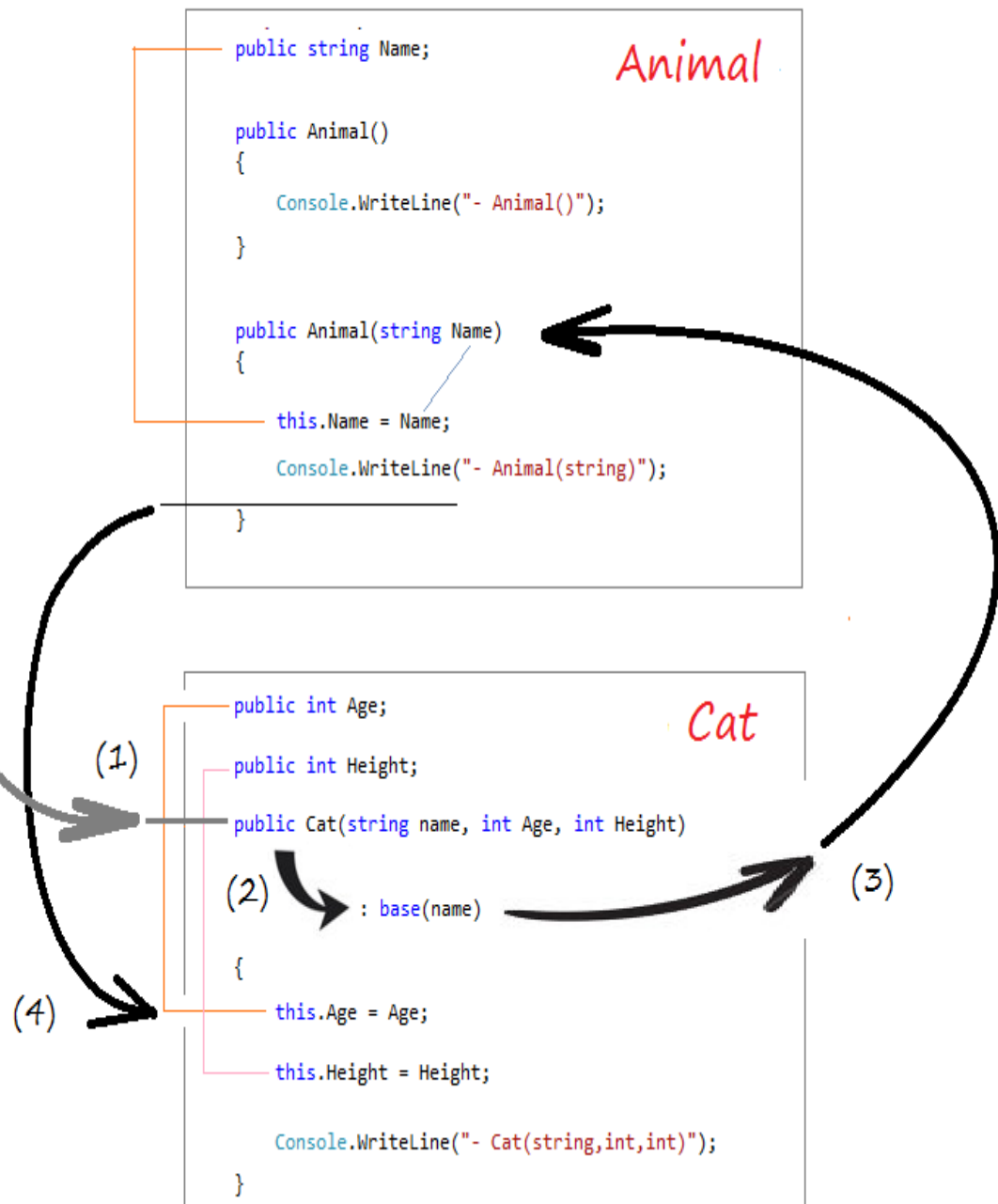
Kết quả chạy class **CatTest**:



```
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.

```
2
1 // Cấu tử này không ghi rõ base từ cấu tử nào của class cha.
2 public Cat(int Age, int Height)
3 {
4
5 }
6
7 // Nó sẽ tương đương với:
8 public Cat(int Age, int Height) : base()
9 {
10
```

```
11 }
```

Một câu tử có thể gọi tới một câu tử khác sử dụng **:this**.

```
?
```

```
1 private int Weight;
2
3 // Câu tử mặc định (Không có tham số).
4 // Gọi tới câu tử Mouse(int)
5 public Mouse() : this(100)
6 {
7 }
8
9 // Câu tử 1 tham số.
10 // Không ghi rõ :base
11 // Nghĩa là base từ câu tử mặc định của class cha
12 public Mouse(int Weight)
13 {
14     this.Weight = Weight;
15 }
```

```
private int Weight;
```

Mouse

```
public Mouse()
: this(100)
```

```
{
}
```

```
public Mouse(int Weight)
{
    this.Weight = Weight;
}
```

```
public Mouse(String name, int Weight)
: base(name)
{
    this.Weight = Weight;
}
```

Mouse.cs

```
?
```

```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6
7 namespace InheritancePolymorphism
8 {
9     public class Mouse : Animal
10     {
11
12         private int Weight;
```

```

13
14 // Cấu tử mặc định (Không có tham số).
15 // Gọi tới cấu tử Mouse(int)
16 public Mouse()
17     : this(100)
18 {
19 }
20
21 // Cấu tử 1 tham số.
22 // Không ghi rõ :base
23 // Nghĩa là base từ cấu tử mặc định của class cha
24 public Mouse(int Weight)
25 {
26     this.Weight = Weight;
27 }
28
29 // Cấu tử 2 tham số
30 public Mouse(String name, int Weight)
31     : base(name)
32 {
33     this.Weight = Weight;
34 }
35
36 public int GetWeight()
37 {
38     return Weight;
39 }
40
41 public void SetWeight(int Weight)
42 {
43     this.Weight = Weight;
44 }
45
46
47 }
48 }

```

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:

IsOperatorDemo.cs

```

?
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6
7 namespace InheritancePolymorphism
8 {
9     class IsOperatorDemo
10    {

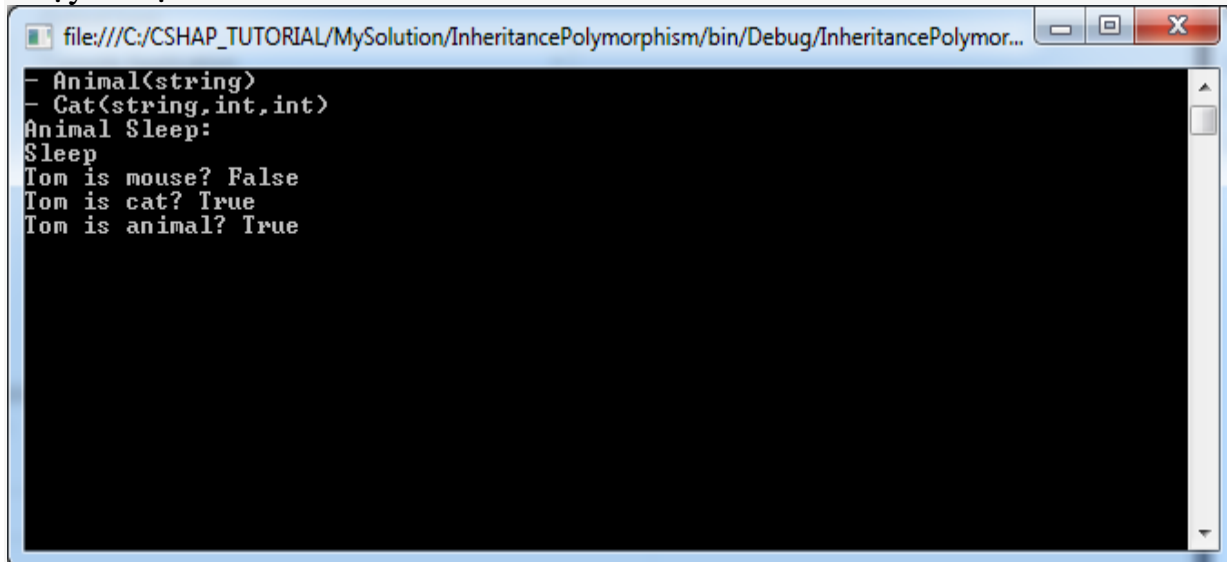
```

```

11 static void Main(string[] args)
12 {
13     // Khởi tạo một đối tượng động vật.
14     // Animal là class trùu tượng
15     // nó ko thể tạo đối tượng từ cấu tử của nó.
16     Animal tom = new Cat("Tom", 3, 20);
17
18     Console.WriteLine("Animal Sleep:");
19     // Gọi phương thức Sleep() của Animal
20     tom.Sleep();
21
22     // Sử dụng toán tử 'is' để kiểm tra xem
23     // một đối tượng có phải kiểu nào đó không.
24     bool isMouse = tom is Mouse;// false
25
26     Console.WriteLine("Tom is mouse? " + isMouse);
27
28     bool isCat = tom is Cat; // true
29     Console.WriteLine("Tom is cat? " + isCat);
30
31     bool isAnimal = tom is Animal; // true
32     Console.WriteLine("Tom is animal? " + isAnimal);
33
34     Console.ReadLine();
35 }
36 }
37 }

```

Chạy ví dụ:



```

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

```

Ép kiểu trong CSharp.

CastDemo.cs

```

?
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6

```

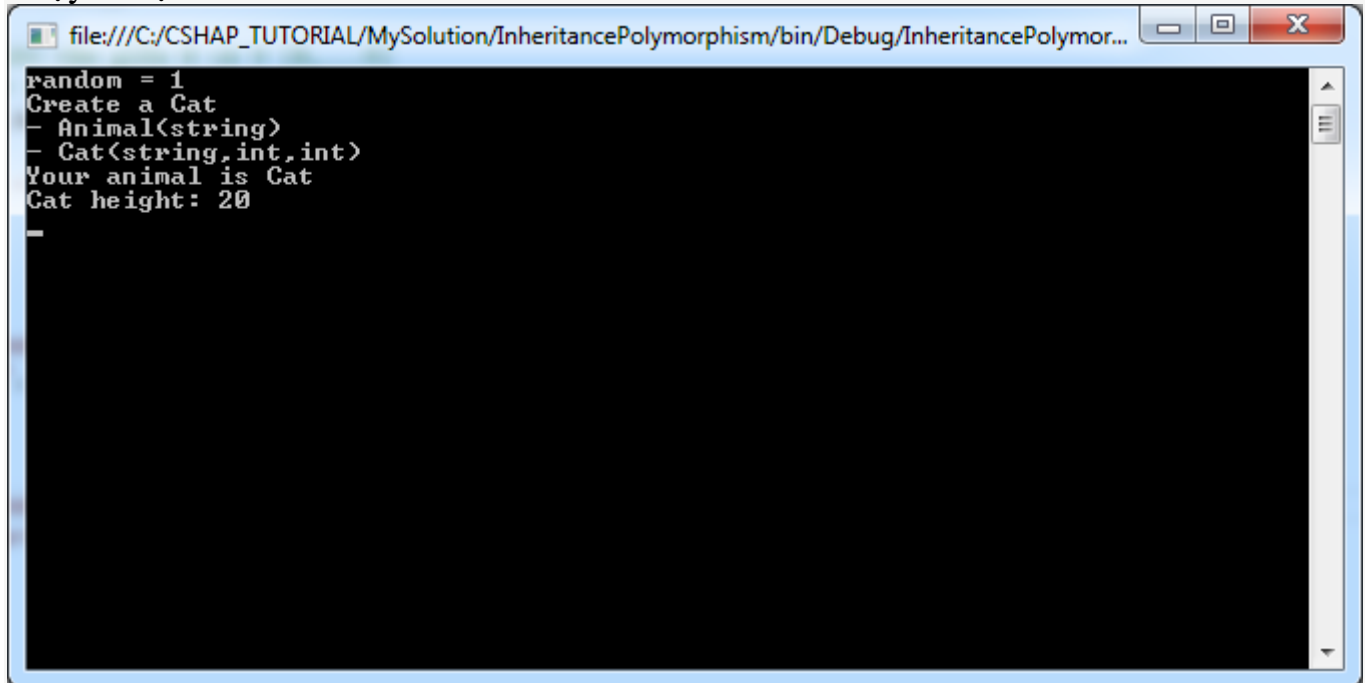
```

7 namespace InheritancePolymorphism
8 {
9     class CastDemo
10    {
11        static void Main(string[] args)
12        {
13            // Gọi phương thức trả về một con vật ngẫu nhiên.
14            Animal animal = GetRandomAnimal();
15
16            if (animal is Cat)
17            {
18                Console.WriteLine("Your animal is Cat");
19
20                // Ép kiểu về Cat.
21                Cat cat = (Cat)animal;
22
23                // Và truy cập vào trường Height của đối tượng Cat.
24                Console.WriteLine("Cat height: " + cat.Height);
25            }
26            else if (animal is Mouse)
27            {
28                Console.WriteLine("Your animal is Mouse");
29
30                // Ép kiểu về Mouse
31                Mouse mouse = (Mouse)animal;
32
33                // Và gọi method của class Mouse.
34                Console.WriteLine("Mouse weight: " + mouse.GetWeight());
35            }
36
37            Console.ReadLine();
38        }
39
40        // Method trả về ngẫu nhiên một con vật.
41        public static Animal GetRandomAnimal()
42        {
43            // Trả về giá trị ngẫu nhiên nằm giữa 0 và 9 (0,...9)
44            // Creates a random number between 1 and 9
45            int random = new Random().Next(0, 10);
46
47            Console.WriteLine("random = " + random);
48
49            Animal animal = null;
50            if (random < 5)
51            {
52                Console.WriteLine("Create a Cat");
53                animal = new Cat("Tom", 3, 20);
54            }
55            else
56            {

```

```
57         Console.WriteLine("Create a Mouse");
58         animal = new Mouse("Jerry", 5);
59     }
60     return animal;
61 }
62 }
63 }
```

Chạy ví dụ:



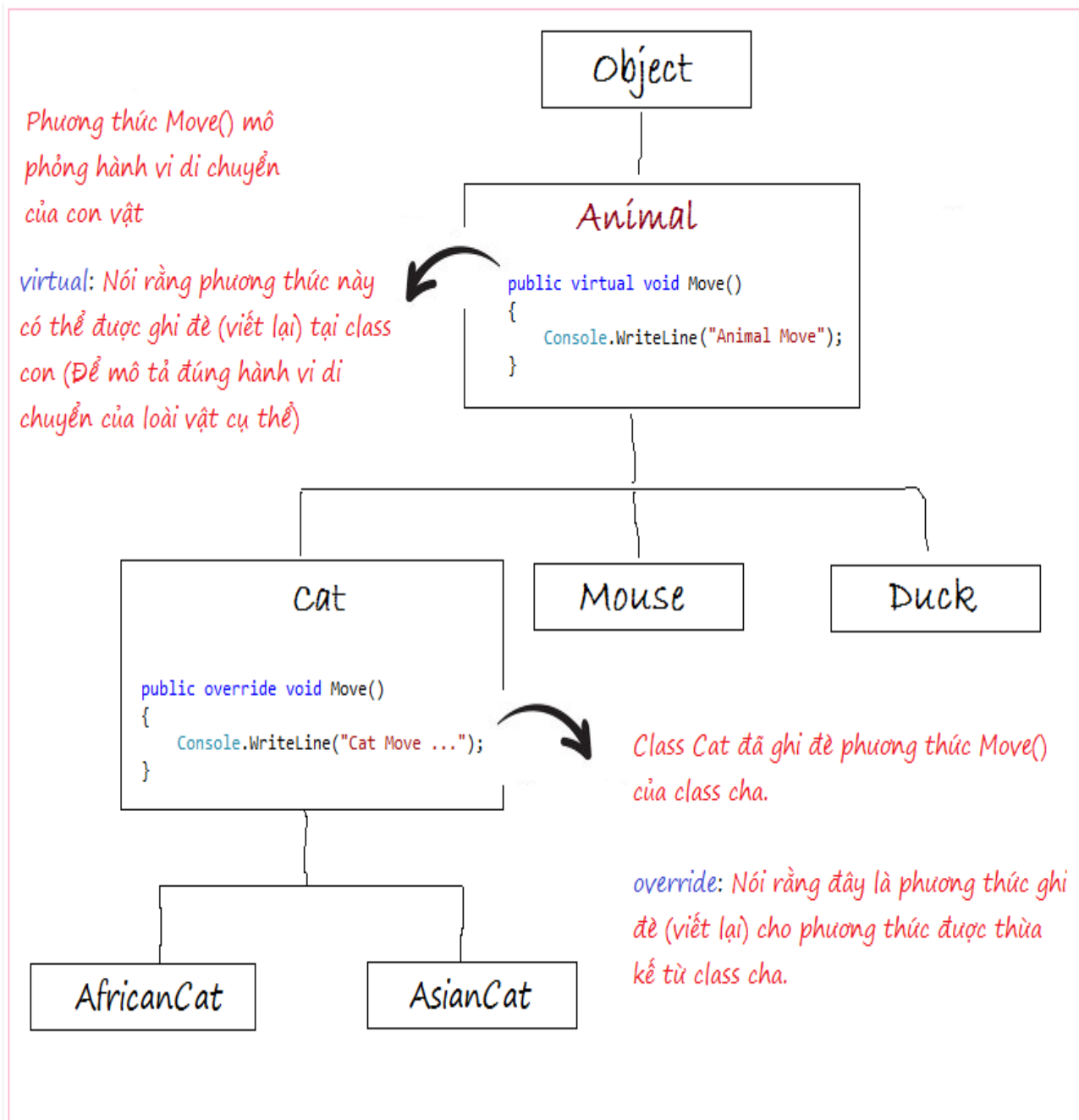
```
file:///C:/CSHAP_TUTORIAL/MySolution/InheritancePolymorphism/bin/Debug/InheritancePolymor...
random = 1
Create a Cat
- Animal<string>
- Cat<string,int,int>
Your animal is Cat
Cat height: 20
```

4- 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 (runtime).



Bạn có một con mèo nguồn gốc châu Á (AsianCat), bạn có thể nói nó là một con mèo (Cat) hoặc nói nó là một con vật (Animal) đó là một khía cạnh của từ đa hình.

Hoặc một ví dụ khác: Trên lý lịch của bạn ghi rằng bạn là một người châu Á, trong khi đó bạn thực tế là một người Việt Nam. Và có thể rằng trong tương lai người ta sẽ cấp cho bạn một hộ chiếu khác ghi rằng bạn là người trái đất.

Ví dụ dưới đây cho bạn thấy cách hành xử giữa khai báo và thực tế:

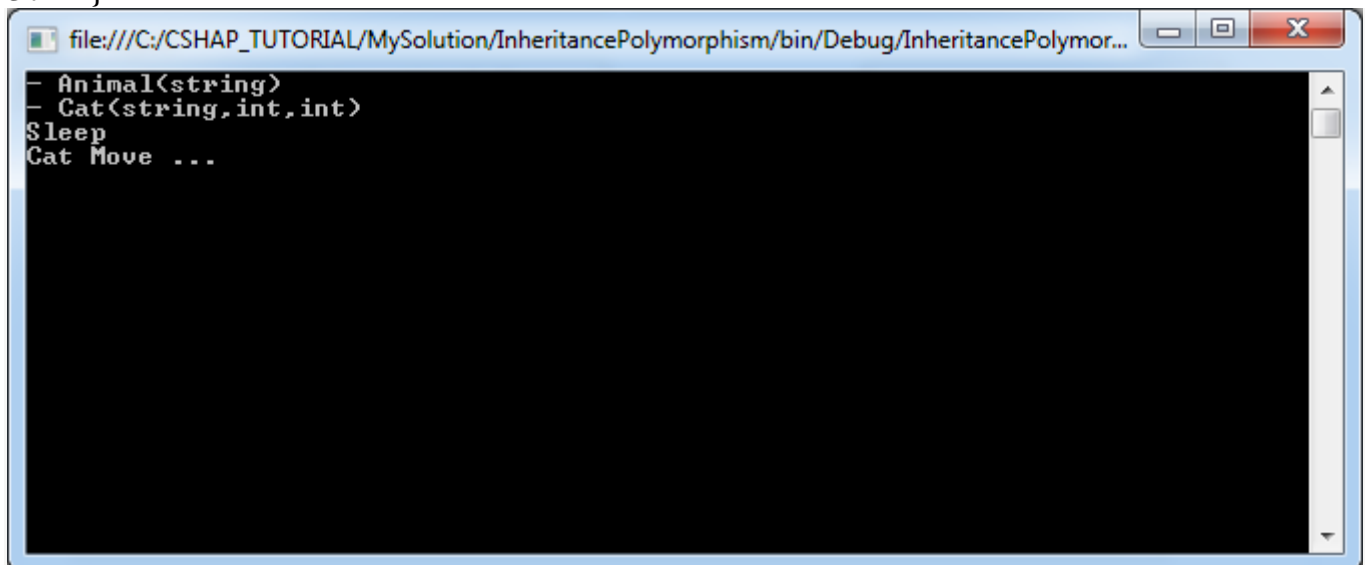
PolymorphismCatDemo.cs

```
?
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6
7 namespace InheritancePolymorphism
8 {
9     class PolymorphismCatDemo
```

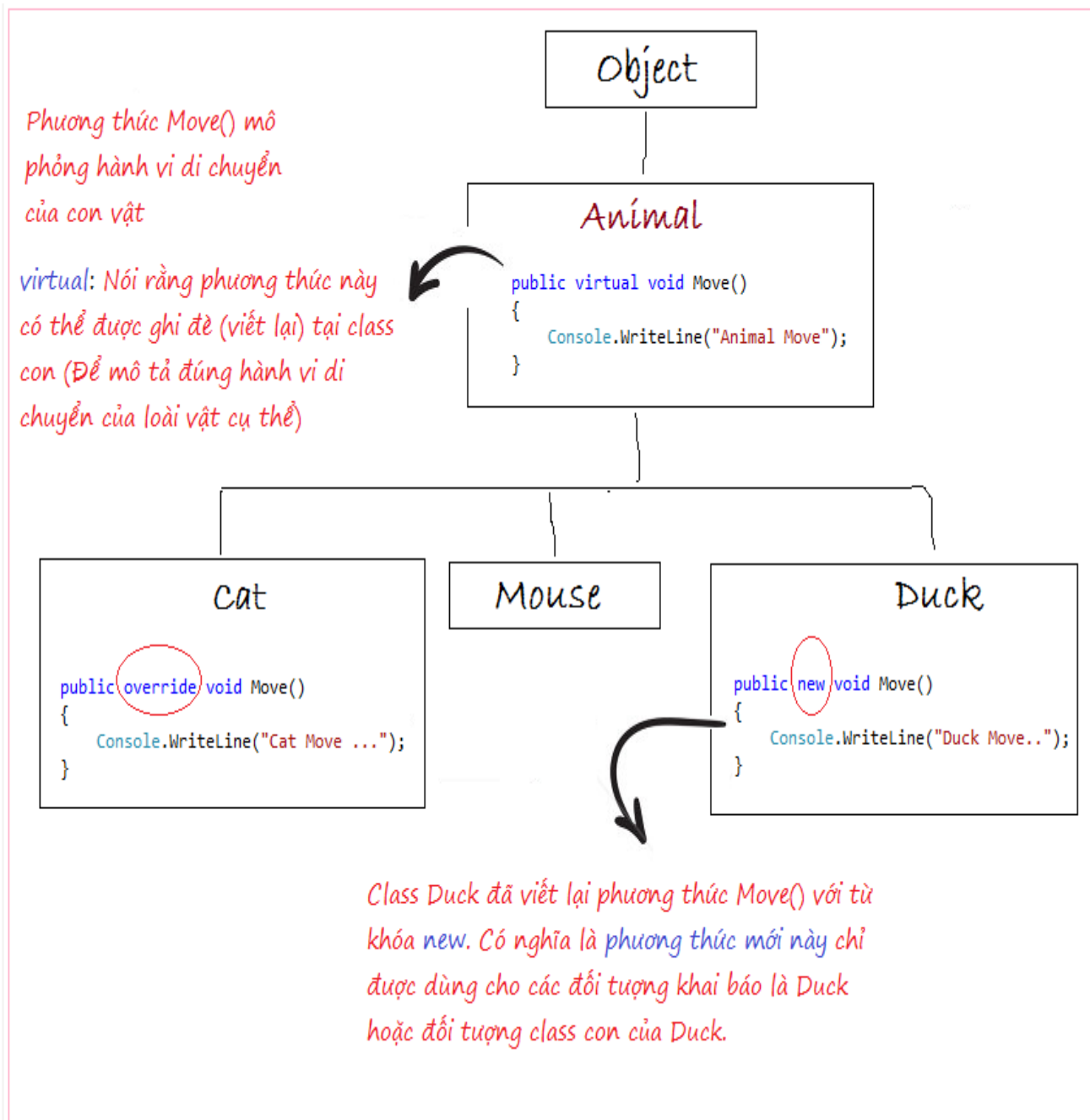
```

10  {
11      static void Main(string[] args)
12      {
13
14          // Bạn khai báo một đối tượng là một động vật (Animal).
15          // Bằng cách tạo nó bởi cấu tử của class Cat.
16
17          // Đối tượng 'tom' khai báo là Animal
18          // vì vậy nó chỉ có thể gọi các phương thức của Animal.
19
20          Animal tom = new Cat("Tom", 3, 20);
21
22
23
24          // Gọi method Sleep Animal
25          tom.Sleep();
26
27          // Gọi method Move()
28          // Move() là phương thức có trong Animal.
29          // Move() được ghi đè trong class Cat.
30          // 'tom' thực tế là Cat, nó sẽ gọi hàm viết đè trong Cat.
31          tom.Move(); // ==> Cat Move.
32
33
34          Console.ReadLine();
35      }
36  }
37  }

```



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



Duck.cs

```
?
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6
7 namespace InheritancePolymorphism
8 {
9     class Duck : Animal
10     {
11
12         public Duck(string name)
13             : base(name)
14         {
15             Console.WriteLine("- Duck(string)");
16         }
17     }
```

```

18
19     public new void Move()
20     {
21         Console.WriteLine("Duck Move..");
22     }
23 }
24 }

```

InheritanceDuckDemo.cs

```

?
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace InheritancePolymorphism
8  {
9      class InheritanceDuckDemo
10     {
11         static void Main(string[] args)
12         {
13             // Bạn khai báo một đối tượng là một động vật (Animal).
14             // Bằng cách tạo nó bởi cấu tử của class Duck (Một con vịt).
15
16             // Đối tượng 'donald' khai báo là Animal
17             // vì vậy nó chỉ có thể gọi các phương thức của Animal.
18
19             Animal donald = new Duck("Donald");
20
21
22
23             // Gọi method Sleep của Animal
24             donald.Sleep();
25
26             // Gọi method Move()
27             // Move() là phương thức có trong Animal.
28             // Move() được ghi đè trong class Duck (Theo từ khóa new,
29             // vì vậy chỉ các đối tượng khai báo là Duck hoặc con
30             của Duck mới được dùng).
31
32             // 'donald' thực tế là Duck, nhưng nó đang được khai báo là Animal.
33             // (Phương thức Move() thừa kế từ Animal sẽ được gọi).
34             donald.Move(); // ==> Animal Move.
35
36
37             Console.ReadLine();
38         }
39     }
40 }

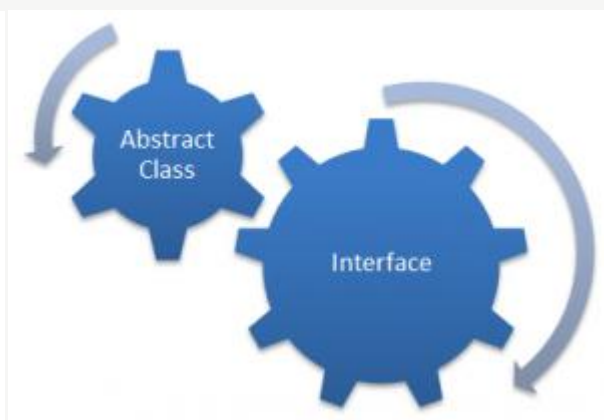
```

Chạy ví dụ:

```
file:///C:/CSHAP_TUTORIAL/MySolution/InheritancePolymorphism/bin/Debug/InheritancePolymor...  
- Animal<string>  
- Duck<string>  
Sleep  
Animal Move
```

Abstract class và Interface trong C#

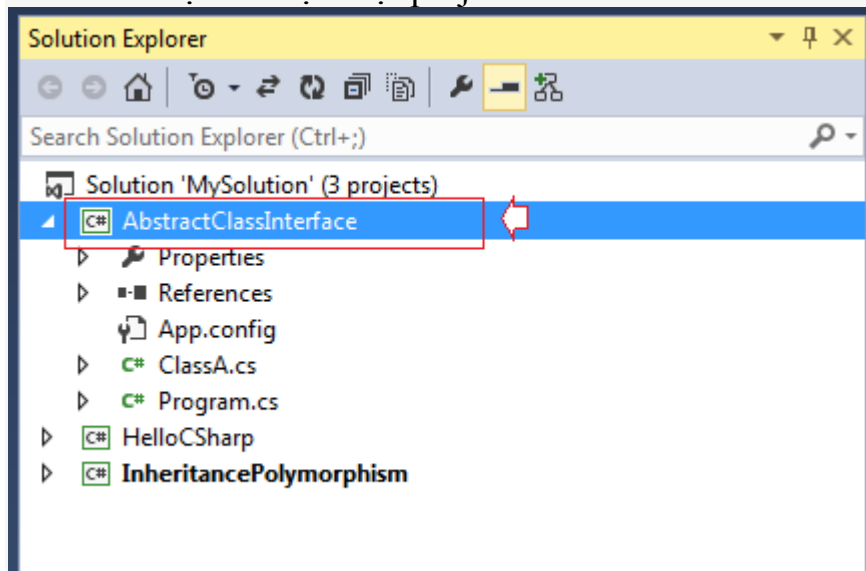
- 1- Giới thiệu
- 2- Class trừu tượng (Abstract Class)
- 3- Ví dụ với class trừu tượng
- 4- Interface
 - 4.1- Cấu trúc của một Interface
 - 4.2- Class thi hành Interface



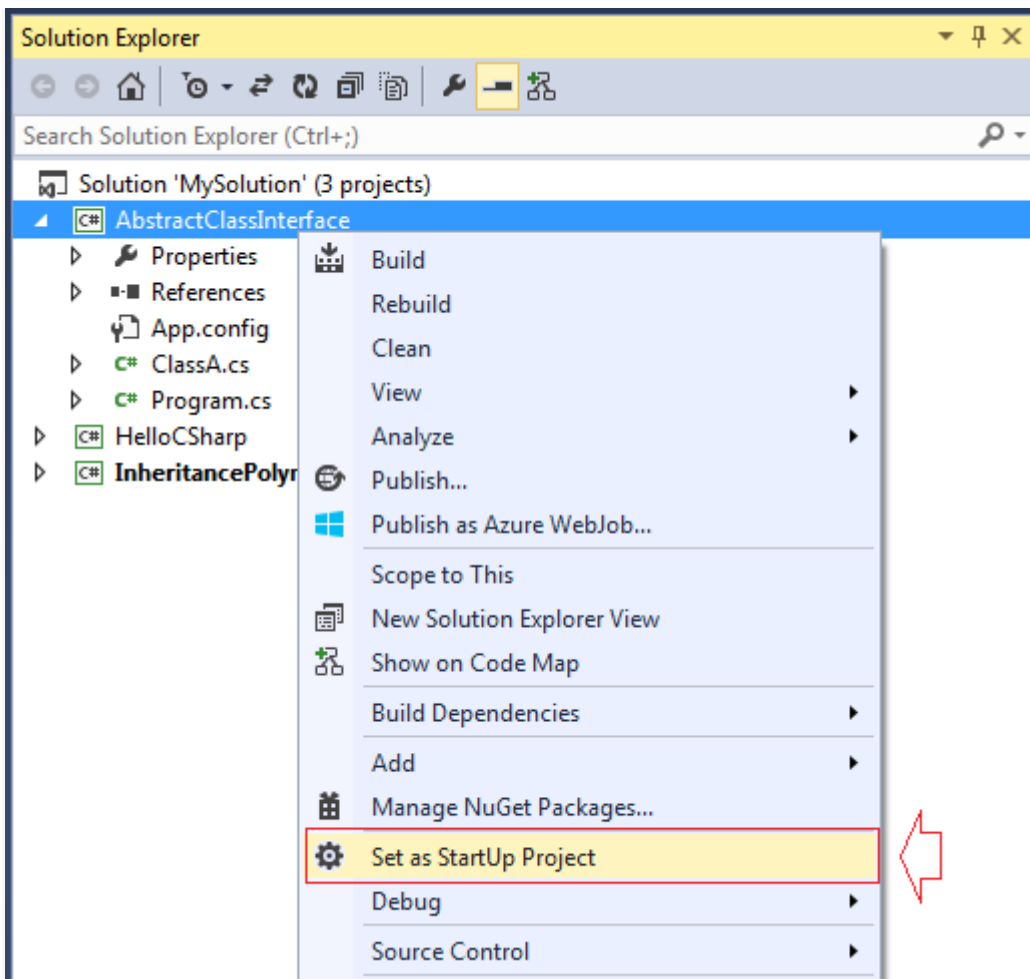
1- Giới thiệu

Trong tài liệu hướng dẫn này tôi sẽ hướng dẫn về Interface và class trừu tượng (Abstract Class). Đồng thời phân tích sự giống và khác nhau giữa chúng.

Trước hết bạn cần tạo một project có tên **AbstractClassInterface** để làm việc với các ví dụ.



Sét nó là project mặc định.



2- Class trừu tượng (Abstract Class)



Abstract class (Class trừu tượng). Hãy xem ví dụ về một class như thế:

?

```

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace AbstractClassInterface
8  {
9      // Một class có ít nhất một phương thức trừu tượng,
10     // phải khai báo là trừu tượng (abstract).
11     public abstract class ClassA
12     {
13
14         // Đây là một method trừu tượng.
15         // Nó không có thân hàm.
16         // Method này có access modifier là: public
17         // (access modifier: Độ truy cập).
18         public abstract void DoSomething();
19
20         // Method này có access modifier là protected
21         protected abstract String DoNothing();
22
23     }

```

```

24     protected abstract void Todo();
25 }
26
27 // Đây là một class trừu tượng.
28 // Chủ động khai báo abstract, mặc dù nó không có method trừu tượng nào.
29 public abstract class ClassB
30 {
31
32 }
33 }

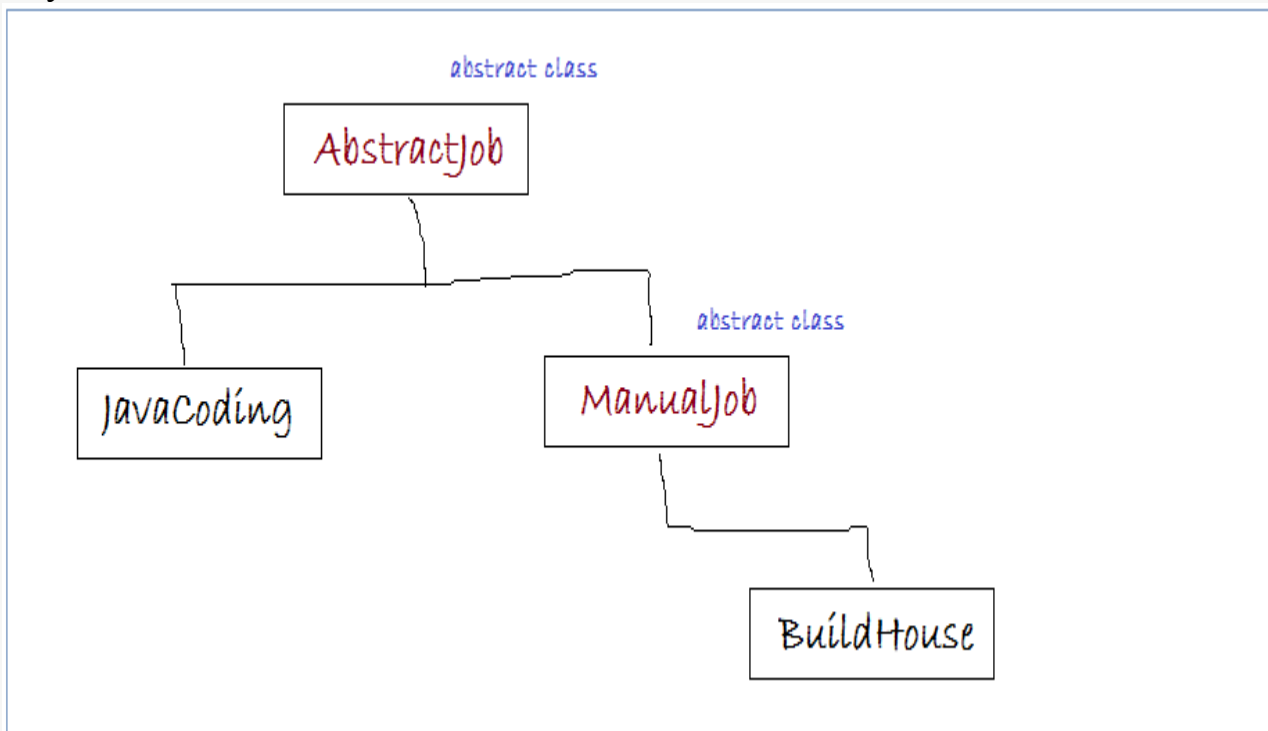
```

Đặc điểm của một class trừu tượng là:

1. Nó được khai báo là 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.

3- Ví dụ với class trừu tượng

Hãy xem hình minh họa:



AbstractJob.cs

```

?
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace AbstractClassInterface
8  {
9      // Class có ít nhất một phương thức trừu tượng, phải khai báo là abstract.
10     public abstract class AbstractJob
11     {
12
13         public AbstractJob()
14         {
15

```

```

16     }
17
18     // Đây là một phương thức trừu tượng,
19     // Nó không có nội dung (body)
20     // Method này trả về tên của công việc.
21     public abstract String GetJobName();
22
23     // Đây là một phương thức trừu tượng.
24     // Phương thức không có nội dung.
25     public abstract void DoJob();
26
27     // Class trừu tượng vẫn có các phương thức thông thường.
28     public void StopJob()
29     {
30         Console.WriteLine("Stop");
31     }
32 }
33 }

```

JavaCoding.cs

```

?
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace AbstractClassInterface
8  {
9      public class JavaCoding : AbstractJob
10     {
11
12         public JavaCoding()
13         {
14         }
15
16         // Method này triển khai method trừu tượng khai báo tại class cha.
17         // (Cần phải có từ khóa 'override').
18         public override void DoJob()
19         {
20             Console.WriteLine("Coding Java...");
21         }
22
23         // Method này triển khai method trừu tượng khai báo tại class cha.
24         // Method này sẽ có thân hàm đầy đủ
25         // Method trả về tên của công việc.
26         // (Cần phải có từ khóa 'override').
27         public override String GetJobName()
28         {
29             return "Java Coding";
30         }

```

```

31
32     public void TestExample()
33     {
34         Console.WriteLine("Testing Example...");
35     }
36 }
37 }

```

CSharpCoding.cs

```

?
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace AbstractClassInterface
8  {
9      class CSharpCoding : AbstractJob
10     {
11         public CSharpCoding()
12         {
13         }
14
15         // Method này triển khai method trừu tượng khai báo tại class cha.
16         // (Cần phải có từ khóa 'override').
17         public override void DoJob()
18         {
19             Console.WriteLine("Coding CSharp...");
20         }
21
22         // Method này triển khai method trừu tượng khai báo tại class cha.
23         // Method này sẽ có thân hàm đầy đủ
24         // Method trả về tên của công việc.
25         // (Cần phải có từ khóa 'override').
26         public override String GetJobName()
27         {
28             return "CSharp Coding";
29         }
30
31         public void RunningExample()
32         {
33             Console.WriteLine("Running Example...");
34         }
35     }
36 }

```

ManualJob.cs

```

?
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;

```

```

4    using System.Text;
5    using System.Threading.Tasks;
6
7    namespace AbstractClassInterface
8    {
9
10       // ManualJob - (Mô phỏng một công việc phổ thông)
11       // Class cha (AbstractJob) có 2 method trừu tượng.
12       // Class này mới chỉ triển khai 1 method trừu tượng của class cha.
13       // Vì vậy nó bắt buộc phải khai báo là abstract.
14       public abstract class ManualJob : AbstractJob
15       {
16
17           public ManualJob()
18           {
19
20           }
21
22           // Method này triển khai method trừu tượng khai báo tại class cha
23           // (Cần phải có từ khóa 'override').
24           public override String GetJobName()
25           {
26               return "Manual Job";
27           }
28
29       }
30 }

```

BuildHouse.cs

```

?
1    using System;
2    using System.Collections.Generic;
3    using System.Linq;
4    using System.Text;
5    using System.Threading.Tasks;
6
7    namespace AbstractClassInterface
8    {
9
10       // Class này thừa kế từ class trừu tượng ManualJob
11       // BuildHouse không khai báo abstract
12       // Vì vậy nó cần triển khai các method trừu tượng còn lại.
13       public class BuildHouse : ManualJob
14       {
15
16           public BuildHouse()
17           {
18
19           }
20
21       // Triển khai method trừu tượng của class cha.

```



```

22 // (Cần phải có từ khóa 'override').
23 public override void DoJob()
24 {
25     Console.WriteLine("Build a House");
26 }
27
28 }
29
30 }

```

Ví dụ demo

JobDemo.cs

```

?
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6
7 namespace AbstractClassInterface
8 {
9     public class JobDemo
10    {
11
12        public static void Main(string[] args)
13        {
14
15            // Khởi tạo một đối tượng AbstractJob.
16            // Nó khởi tạo từ cấu tử của class JavaCoding.
17            AbstractJob job1 = new JavaCoding();
18
19            // Gọi method doJob()
20            job1.DoJob();
21
22            // Method getJobName là trù tượng trong class AbstractJob
23            // Nhưng nó đã được triển khai tại một class con nào đó.
24            // Vì vậy gọi không vấn đề gì.
25            String jobName = job1.GetJobName();
26
27            Console.WriteLine("Job Name 1= " + jobName);
28
29
30
31            // Khởi tạo một đối tượng AbstractJob.
32            // Nó khởi tạo từ cấu tử của class CSharpCoding.
33            AbstractJob job2 = new CSharpCoding();
34
35            // Gọi method doJob()
36            job2.DoJob();
37
38            // Method getJobName là trù tượng trong class AbstractJob

```

```

39      // Nhưng nó đã được triển khai tại một class con nào đó.
40      // Vì vậy gọi không vấn đề gì.
41      String jobName2 = job2.GetJobName();
42
43      Console.WriteLine("Job Name 2= " + jobName2);
44
45
46      // Khởi tạo một đối tượng AbstractJob
47      // từ cấu tử của class BuildHouse.
48      AbstractJob job3 = new BuildHouse();
49
50      job3.DoJob();
51
52      String jobName3 = job3.GetJobName();
53
54      Console.WriteLine("Job Name 3= " + jobName2);
55
56
57      Console.ReadLine();
58  }
59  }
60  }

```

Kết quả chạy ví dụ:

```

file:///C:/CSHAP_TUTORIAL/MySolution/AbstractClassInterface/bin/Debug/AbstractClassInterface....
Coding Java...
Job Name 1= Java Coding
Coding CSharp...
Job Name 2= CSharp Coding
Build a House
Job Name 3= CSharp Coding
-

```

4- Interface



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

```

?
1  // Class B là con của class A, hay nói là B mở rộng từ A
2  // CSharp chỉ cho phép một class mở rộng từ duy nhất một class khác.
3  public class B : A
4  {
5      // ....
6  }
7
8  // Trong trường hợp bạn không viết mở rộng từ một class nào.
9  // CSharp tự hiểu là nó thừa kế từ class Object.

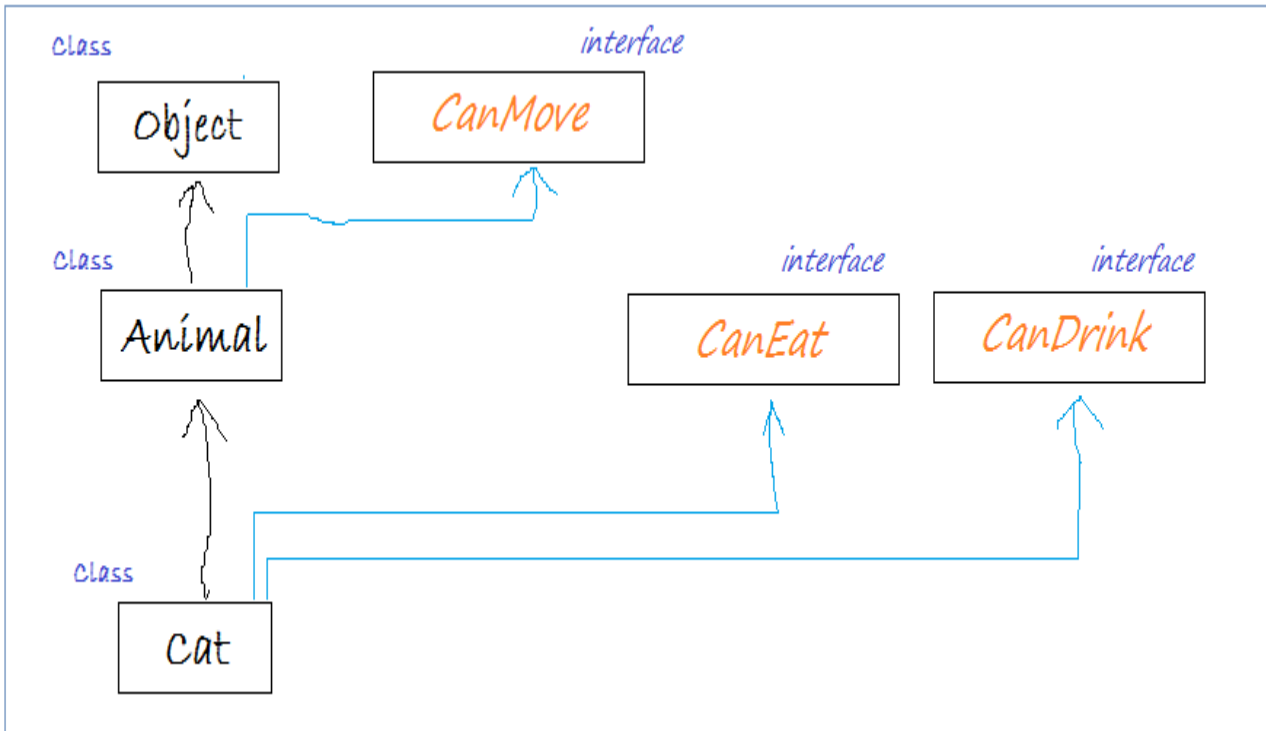
```

```

10 public class B
11 {
12
13 }
14
15 // Cách khai báo class này, và cách trên là tương đương nhau.
16 public class B : Object
17 {
18
19 }

```

Nhưng một class có thể mở rộng từ nhiều Interface



```

?
1 // Một class có thể mở rộng từ duy nhất một class
2 // Nhưng có thể mở rộng từ nhiều Interface.
3 public class Cat : Animal, CanEat, CanDrink
4 {
5
6 // ....
7 }

```

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).

4.1- 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 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 chữ 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.*

NoAccessModifierInterface.cs

```
?
1 namespace AbstractClassInterface
2 {
3
4 // Đây là một Interface không khai báo access modifier.
5 // Mặc định modifier của nó là 'internal'.
6 // Nó chỉ được dùng trong nội bộ một Assembly.
7 interface NoAccessModifierInterface
8 {
9 }
10
11 }
```

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

CanMove.cs

```
?
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6
7 namespace AbstractClassInterface
8 {
9
10 // Interface này định nghĩa những thứ có khả năng di chuyển.
11 public interface CanMove
12 {
13
14 // (Chạy)
15 // Các method trong Interface đều là công khai và trừu tượng (public abstract)
16 // (Nhưng bạn không được phép viết public hoặc abstract ở đây)
17 void Run();
18
19 // (Quay trở lại)
20 // Cho dù không viết rõ public abstract thì CSharp luôn hiểu là vậy.
21 void Back();
22
23 // (Lấy ra vận tốc chạy).
24 int GetVelocity();
25 }
```

```
26 }
27
28 }
```

CanDrink.cs

```
?
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6
7 namespace AbstractClassInterface
8 {
9
10     // Interface này định nghĩa những thứ có khả năng biết uống.
11     public interface CanDrink
12     {
13
14         void Drink();
15
16     }
17
18 }
```

CanEat.cs

```
?
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6
7 namespace AbstractClassInterface
8 {
9
10     // Interface này định nghĩa thứ có khả năng biết ăn.
11     public interface CanEat
12     {
13
14         void Eat();
15
16     }
17 }
```

4.2- Class thi hành Interface

Khi một class thi hành một Interface, **bạn phải triển khai 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.

Hãy xem ví dụ, class **Animal** thi hành interface **CanMove**.

Animal.cs

?

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace AbstractClassInterface
8  {
9      // Animal (Class mô phỏng lớp động vật)
10     // Nó mở rộng từ class Object (Mặc dù không ghi rõ).
11     // Và khai báo thi hành (hoặc gọi là thừa kế) interface CanMove.
12     // Interface CanMove có 3 method trừu tượng.
13     // Class này mới triển khai 1 method
14     // Vì vậy nó bắt buộc phải khai báo abstract
15     // Các method trừu tượng còn lại sẽ được class con triển khai
16     public abstract class Animal : CanMove
17     {
18
19         // Triển khai method Run() từ interface CanMove.
20         // Bạn phải viết nội dung của phương thức.
21         // Modifier phải là public.
22         public void Run()
23         {
24             Console.WriteLine("Animal run...");
25         }
26
27         // Nếu bạn không triển khai một phương thức nào đó của Interface
28         // bạn phải viết lại nó dưới dạng một phương thức trừu tượng.
29         // (Luôn luôn là public abstract)
30         public abstract void Back();
31
32
33         // Nếu bạn không triển khai một phương thức nào đó của Interface
34         // bạn phải viết lại nó dưới dạng một phương thức trừu tượng.
35         // (Luôn luôn là public abstract)
36         public abstract int GetVelocity();
37
38     }
39
40
41 }
```

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

Cat.cs

?

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
```

```
4    using System.Text;
5    using System.Threading.Tasks;
6
7    namespace AbstractClassInterface
8    {
9
10       // Class Cat mở rộng từ class Animal và thi hành 2 interface CanEat, CanDrink.
11       public class Cat : Animal, CanEat, CanDrink
12       {
13
14           private String name;
15
16           public Cat(String name)
17           {
18               this.name = name;
19           }
20
21           public String getName()
22           {
23               return this.name;
24           }
25
26           // Triển khai phương thức trù tượng của Animal.
27           // (Phải ghi rõ 'override').
28           public override void Back()
29           {
30               Console.WriteLine(name + " cat back ...");
31           }
32
33           // Triển khai phương thức trù tượng của Animal
34           // (Phải ghi rõ 'override' )
35           public override int GetVelocity()
36           {
37               return 110;
38           }
39
40           // Triển khai method của interface CanEat
41           public void Eat()
42           {
43               Console.WriteLine(name + " cat eat ...");
44           }
45
46           // Triển khai method của interface CanDrink
47           public void Drink()
48           {
49               Console.WriteLine(name + " cat drink ...");
50           }
51
52     }
```

54 }

Mouse.cs

```
?  
1 using System;  
2 using System.Collections.Generic;  
3 using System.Linq;  
4 using System.Text;  
5 using System.Threading.Tasks;  
6  
7 namespace AbstractClassInterface  
8 {  
9     public class Mouse : Animal, CanEat, CanDrink  
10    {  
11  
12        // Triển khai phương thức trừu tượng của Animal.  
13        // (Phải có từ khóa 'override').  
14        public override void Back()  
15        {  
16            Console.WriteLine("Mouse back ...");  
17        }  
18  
19        // Triển khai phương thức trừu tượng của Animal.  
20        // (Phải có từ khóa 'override').  
21        public override int GetVelocity()  
22        {  
23            return 85;  
24        }  
25  
26        // Triển khai phương thức của interface CanDrink.  
27        public void Drink()  
28        {  
29            Console.WriteLine("Mouse drink ...");  
30        }  
31  
32        // Triển khai phương thức của interface CanEat.  
33        public void Eat()  
34        {  
35            Console.WriteLine("Mouse eat ...");  
36        }  
37  
38    }  
39  
40 }
```

AnimalDemo.cs

```
?  
1 using System;  
2 using System.Collections.Generic;  
3 using System.Linq;  
4 using System.Text;  
5 using System.Threading.Tasks;
```



```

6
7 namespace AbstractClassInterface
8 {
9
10     public class AnimalDemo
11     {
12
13         public static void Main(string[] args)
14         {
15
16             // Khởi tạo một đối tượng CanEat
17             // Một đối tượng khai báo là CanEat
18             // Nhưng thực tế là Cat.
19             CanEat canEat1 = new Cat("Tom");
20
21             // Một đối tượng khai báo là CanEat
22             // Nhưng thực tế là Mouse.
23             CanEat canEat2 = new Mouse();
24
25             // Tính đa hình thể hiện rõ tại đây.
26             // CSharp luôn biết một đối tượng là kiểu gì
27             // ==> Tom cat eat ...
28             canEat1.Eat();
29
30             // ==> Mouse eat ...
31             canEat2.Eat();
32
33             bool isCat = canEat1 is Cat;// true
34
35             Console.WriteLine("catEat1 is Cat? " + isCat);
36
37             // Kiểm tra 'canEat2' có phải là chuột hay không?.
38             if (canEat2 is Mouse)
39             {
40                 // Ép kiểu
41                 Mouse mouse = (Mouse)canEat2;
42                 // Gọi method drink (Thừa kế từ CanDrink).
43                 mouse.Drink();
44             }
45
46             Console.ReadLine();
47         }
48     }
49
50 }

```

Kết quả chạy ví dụ:

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

Access Modifier trong C#

- 1- Modifier trong CSharp
- 2- Tổng quan về access modifier
- 3- private access modifier
- 4- private constructor
- 5- protected access modifier
- 6- internal access modifier
- 7- protected internal access modifier
- 8- public access modifier
- 9- Độ truy cập và thừa kế

1- Modifier trong CSharp

Các **access modifiers** trong **CSharp** xác định độ truy cập (Phạm vi) vào dữ liệu của 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**

2- 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	Trong	Ngoài class định	Trong	Ngoài

	định nghĩa?	class con	ngĩa, ngoài class con	class con	class con
private	Y				
protected	Y	Y		Y	
internal	Y	Y	Y		
protected internal	Y	Y	Y		
public	Y	Y	Y	Y	Y

Bạn có thể hiểu chi tiết hơn theo các ví dụ dưới đây:

3- private access modifier



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

```

namespace AccessModifierTutorial
{
    class Person
    {
        private string Name;

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

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

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

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

            ShowSecret();
        }
    }
}

```

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.

```
namespace AccessModifierTutorial
{
```

```
class Person
{
```

```
private string Name;
```

private field

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

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

private method

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

```
class Diary
{
```

```
public void Logging()
{
```

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

(private static)

```
    ShowSecret();
```

none-static

```
}
```

```
}
```

```
}
```

```
}
```

```
}
```

```
}
```

```
}
```

```
}
```

```
}
```

```
}
```

```
}
```

```
}
```

```
}
```

```
}
```

```
}
```

```
}
```

```
}
```

```
}
```

```
}
```

```
}
```

```
}
```

```
}
```

```
}
```

```
}
```

```
}
```

```
}
```

```
}
```

```
}
```

```
}
```

```
}
```

```
}
```

```
}
```

```
namespace AccessModifierTutorial
{
```

```
class PersonTest
{
```

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

```
    Person tom = new Person("Tom");
```

```
    String name = tom.Name; // Error
```

```
    tom.ShowSecret(); // Error
```

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

```
}
```

```
}
```

```
}
```



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

4- private constructor



Cấu tử (constructor), phương thức (method), trường (field) đều được gọi là các thành viên trong class.

Nếu bạn tạo một class, và có một cấu tử private, bạn không thể tạo một đối tượng của class này từ cấu tử private đó từ bên ngoài class này. Hãy xem ví dụ minh họa:

```

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");
        }
    }
}

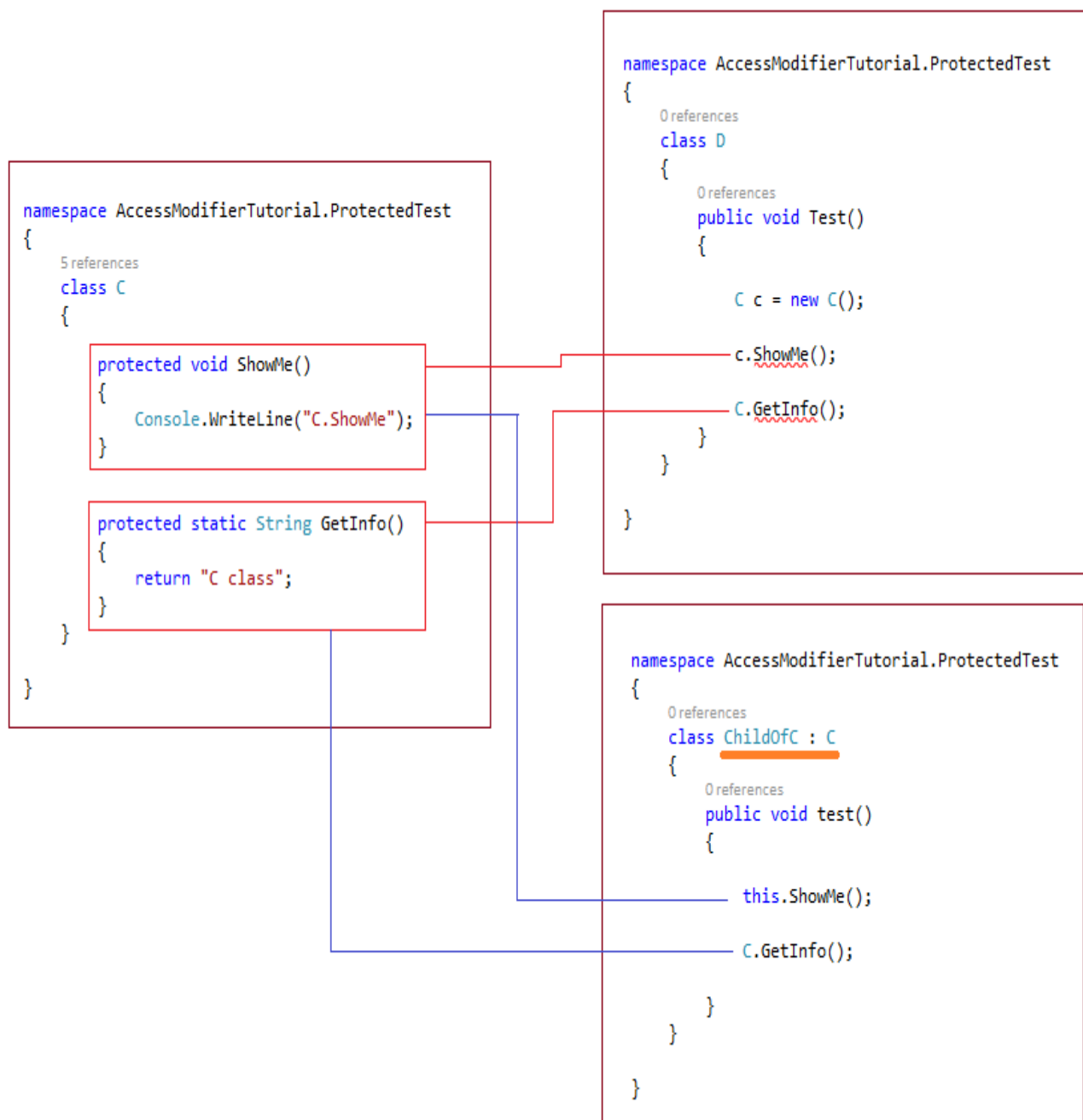
```

5- 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, ..).



6- 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.

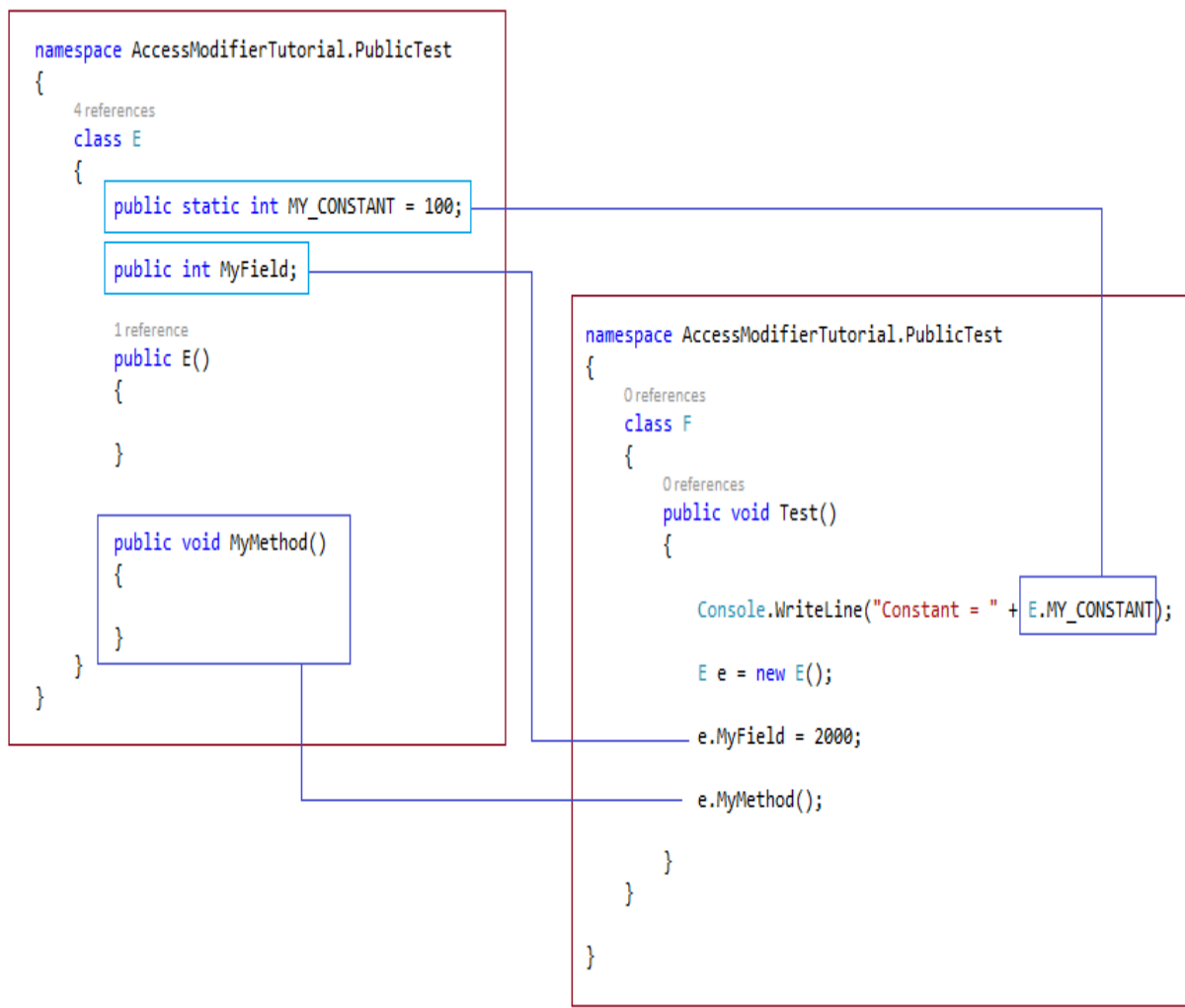
Assembly một cách cụ thể chứa mã .NET theo MSIL (**Microsoft Intermediate language - Một ngôn ngữ trung gian**) sẽ được biên dịch thành mã máy (Native code) ("JITted" - biên dịch bởi các trình biên dịch Just-In-Time) trong lần đầu tiên nó được thực thi trên máy tính,. Đó là mã đã được biên dịch cũng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

7- 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.

8- public access modifier

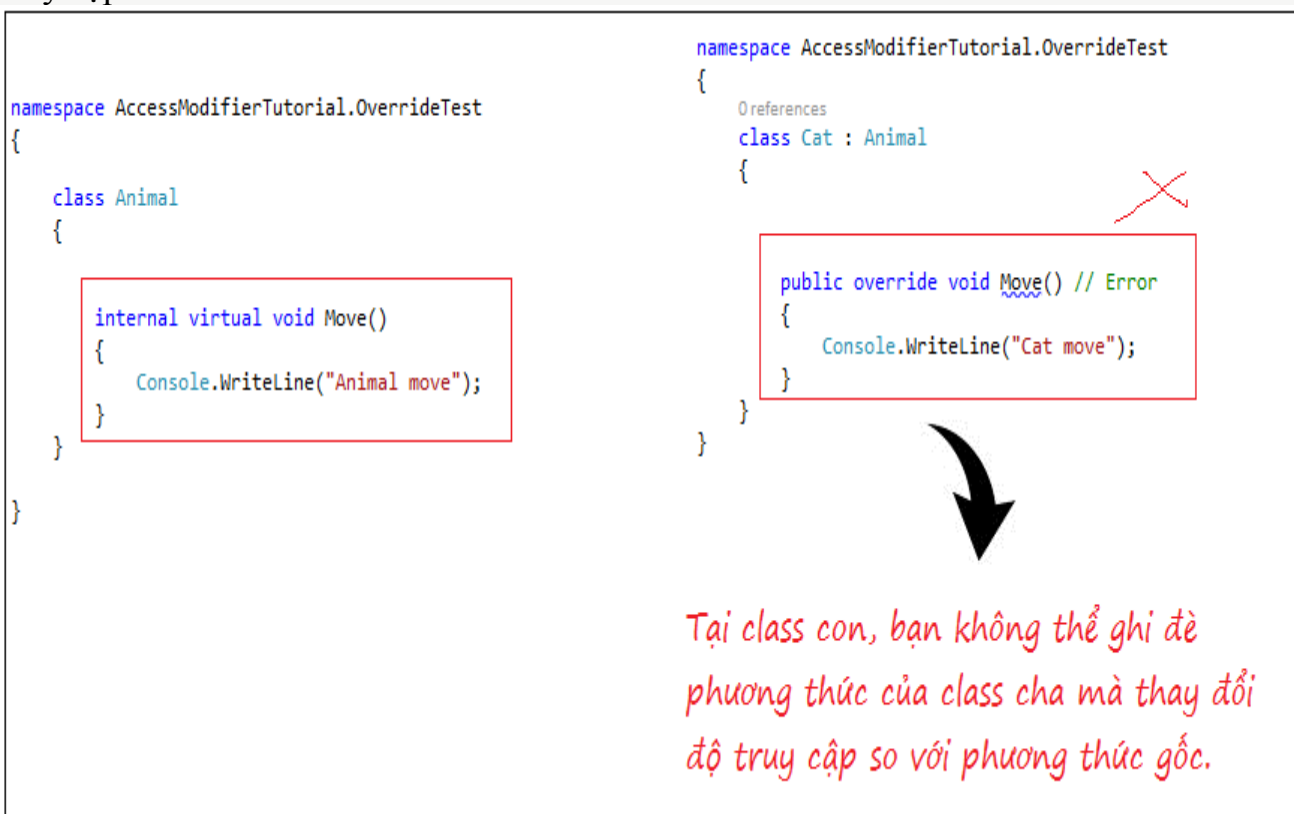
public access modifier là mạnh mẽ nhất và có thể truy cập ở mọi nơi. Nó có phạm vi truy cập rộng nhất so với các modifier khác.



9- Độ truy cập và thừa kế



Trong **CSharp** bạn có thể ghi đè (override) một method của class cha bởi một method cùng tên cùng tham số, cùng kiểu trả về tại class con, tuy nhiên bạn không được phép thay đổi độ truy cập 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");
        }
    }
}
```

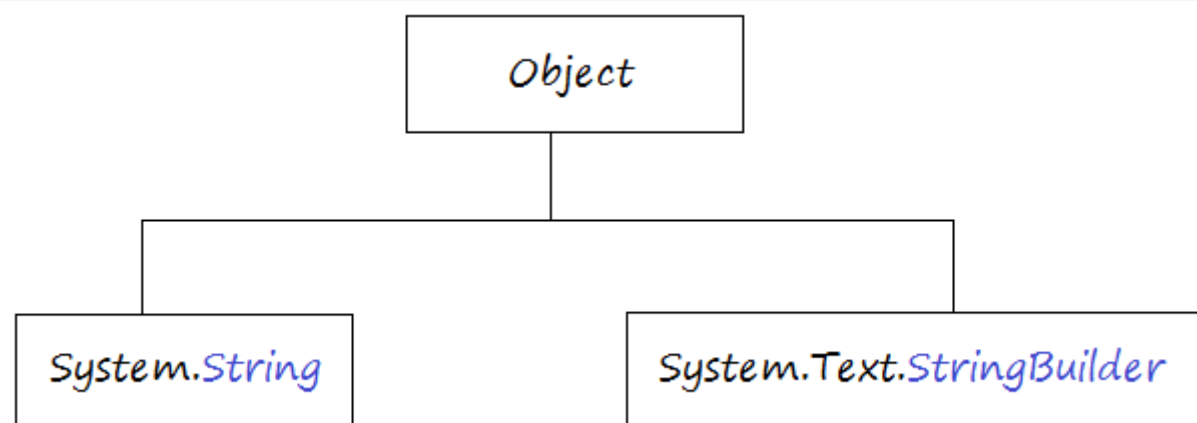
Tại class con, bạn có thể tạo một phương thức cùng tên, cùng tham số, cùng kiểu trả về và khác độ truy cập với phương thức ở class cha, nhưng phải sử dụng từ khóa 'new'.

Hướng dẫn sử dụng C# String và StringBuilder

- 1- Sơ đồ thừa kế
- 2- Khái niệm mutable & immutable
- 3- String và string
- 4- String
 - 4.1- Các method của String
 - 4.1.1- Length
 - 4.1.2- Concat(...)
 - 4.1.3- IndexOf(..)
 - 4.1.4- Substring(..)
 - 4.1.5- Replace(...)
 - 4.1.6- Các ví dụ khác

5- StringBuilder

- 1- 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**.

dùng **StringBuilder** để đạt hiệu năng nhanh nhất. Về cơ bản 2 class này có nhiều điểm giống nhau.

- **String** là không thể thay đổi (*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*)

2- Khái niệm mutable & immutable



Hãy xem một ví dụ minh họa:

```
?
1 // Đây là một class có 1 trường value.
2 // Sau khi khởi tạo đối tượng bạn có thể sét đặt lại giá trị của trường Value
3 // thông qua việc gọi method SetNewValue(int).
4 // Như vậy đây là class có thể thay đổi (mutable).
5 class MutableClassExample
6 {
7     private int Value;
8
9     public MutableClassExample(int value)
10    {
11        this.Value = value;
12    }
13
14    public void SetNewValue(int newValue)
15    {
16        this.Value = newValue;
17    }
18 }
19
20 // Đây là một class với trường Value, Name.
21 // Khi bạn khởi tạo đối tượng class này
22 // bạn không thể sét đặt lại Value từ bên ngoài, và tất cả các trường khác của nó cũng thế.
23 // Class này không hề có các hàm để sét đặt lại các trường (field) từ bên ngoài.
24 // Nếu muốn bạn chỉ có thể tạo mới một đối tượng khác.
25 // Điều đó có nghĩa là class này là không thể thay đổi (immutable)
26 class ImmutableClassExample
27 {
28     private int Value;
29     private String Name;
30
31     public ImmutableClassExample(String name, int value)
32     {
33         this.Value = value;
34         this.Name = name;
35     }
36
37     public String GetName()
38     {
39         return Name;
40     }
41 }
```

```

42     public int GetValue()
43     {
        return Value;
    }
}

```

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

3- 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

4- 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 class 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.

?

```

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

```

4.1- Các method của String

Bạn có thể tra cứu các phương thức của String tại:

- <https://msdn.microsoft.com/en-us/library/system.string%28v=vs.110%29.aspx>

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

Some String methods

?

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46

```
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)
```

```
47
48 public string Replace(string oldValue, string newValue)
49
50 public string[] Split(params char[] separator)
51
52 public string[] Split(char[] separator, int count)
53
54 public string[] Split(char[] separator, int count, StringSplitOptions options)
55
56 public string[] Split(char[] separator, StringSplitOptions options)
57
58 public string[] Split(string[] separator, StringSplitOptions options)
59
60 public bool StartsWith(string value)
61
62 public bool StartsWith(string value, bool ignoreCase, CultureInfo culture)
63
64 public bool StartsWith(string value, StringComparison comparisonType)
65
66 public string Substring(int startIndex)
67
68 public string Substring(int startIndex, int length)
69
70 public char[] ToCharArray()
71
72 public char[] ToCharArray(int startIndex, int length)
73
74 public string ToLower()
75
76 public string ToLower(CultureInfo culture)
77
78 public string ToLowerInvariant()
79
80 public override string ToString()
81
82 public string ToUpper()
83
84 public string ToUpper(CultureInfo culture)
85
86 public string ToUpperInvariant()
87
88 public string Trim()
89
90 public string Trim(params char[] trimChars)
91
92 public string TrimEnd(params char[] trimChars)
93
94 public string TrimStart(params char[] trimChars)
95
```

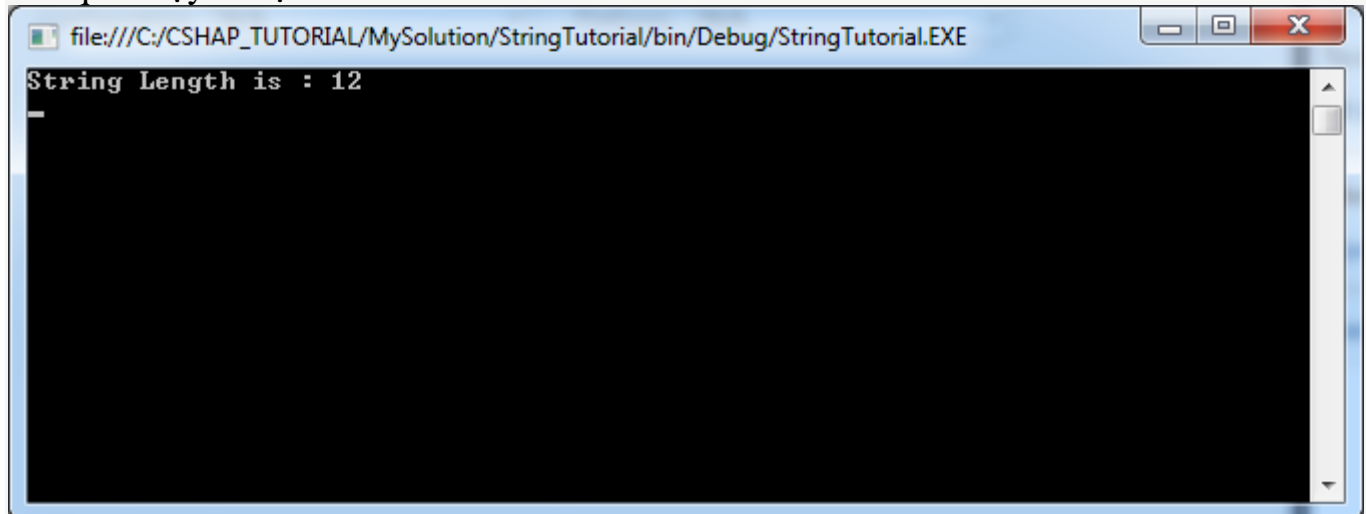
4.1.1- Length

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

LengthDemo.cs

```
?
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace StringTutorial
8  {
9      class LengthDemo
10     {
11         public static void Main(string[] args)
12         {
13             String str = "This is text";
14
15             // Length là một thuộc tính của string.
16             // Nó chính là độ dài của chuỗi (số ký tự của chuỗi).
17             int len = str.Length;
18
19             Console.WriteLine("String Length is : " + len);
20
21             Console.Read();
22         }
23     }
24 }
```

Kết quả chạy ví dụ:



4.1.2- Concat(...)

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

ConcatDemo.cs

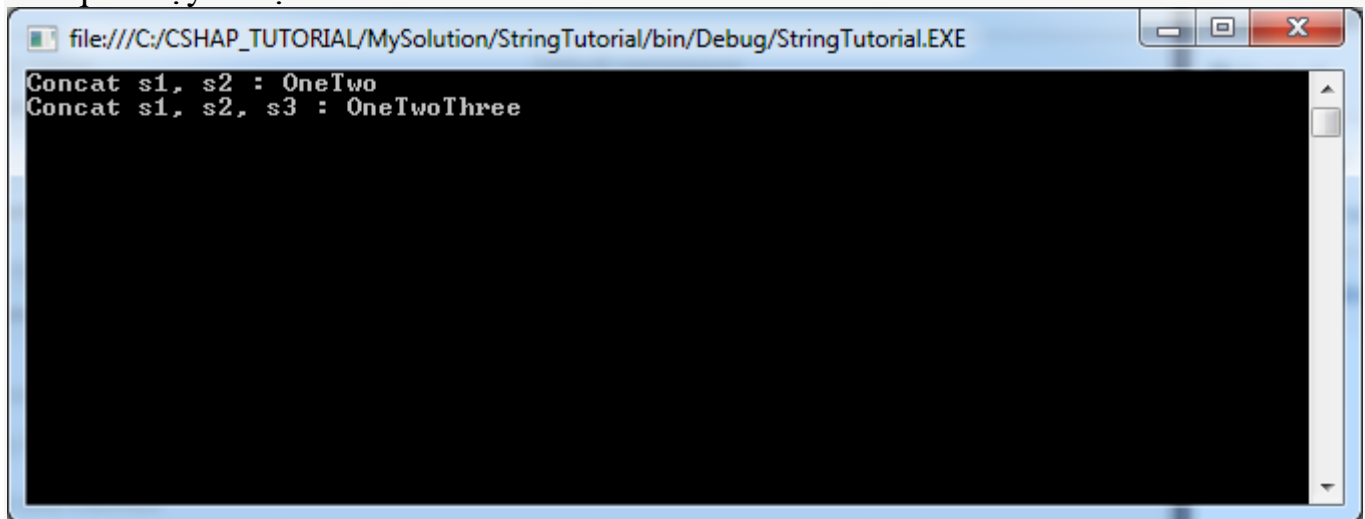
```
?
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
```

```

7 namespace StringTutorial
8 {
9     class ConcatDemo
10    {
11        public static void Main(string[] args)
12        {
13            string s1 = "One";
14            string s2 = "Two";
15            string s3 = "Three";
16
17            // Giống với s1 + s2;
18            string s = String.Concat(s1, s2);
19
20            Console.WriteLine("Concat s1, s2 : " + s);
21
22            // Giống với s1 + s2 + s3;
23            s = String.Concat(s1, s2, s3);
24
25            Console.WriteLine("Concat s1, s2, s3 : " + s);
26
27            Console.Read();
28        }
29    }
30
31
32 }

```

Kết quả chạy ví dụ:



```

file:///C:/CSHAP_TUTORIAL/MySolution/StringTutorial/bin/Debug/StringTutorial.EXE
Concat s1, s2 : OneTwo
Concat s1, s2, s3 : OneTwoThree

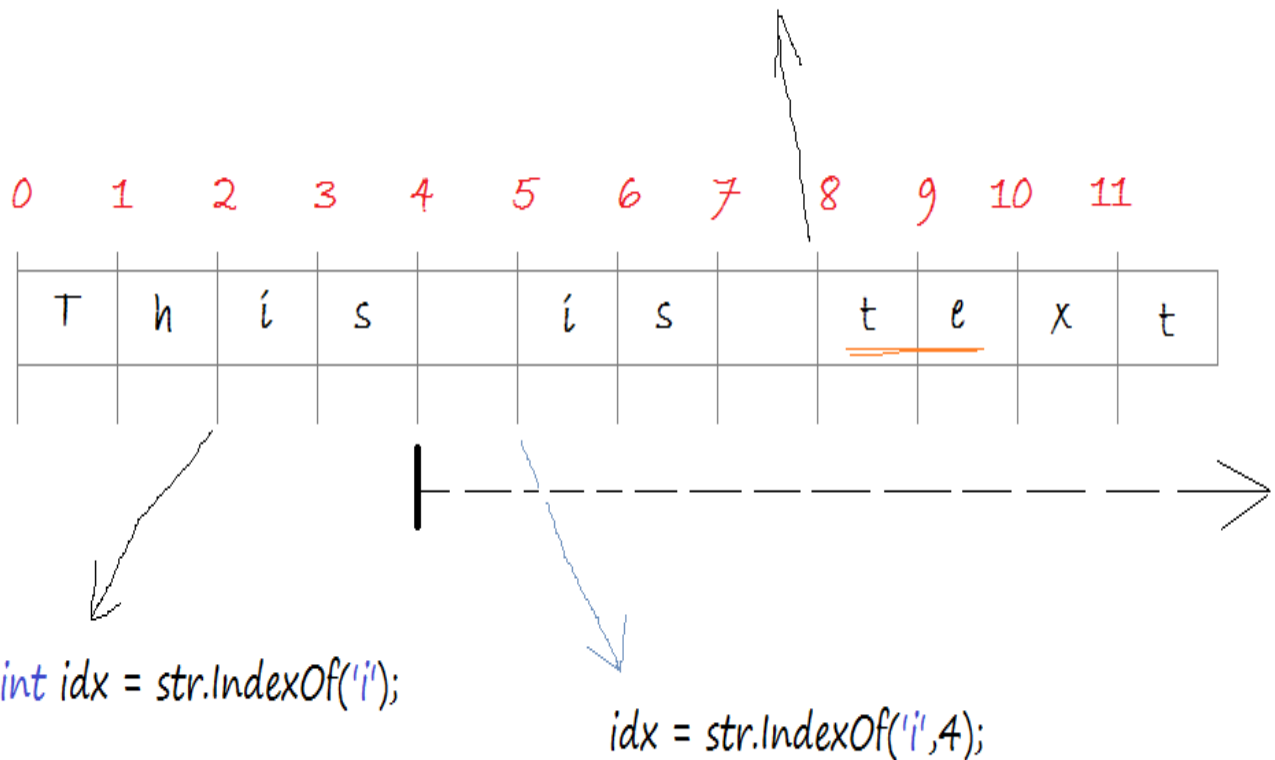
```

4.1.3- 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 tính bắt đầu từ số 0 (Không phải 1).

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

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



IndexOfDemo.cs

```
?  
1 using System;  
2 using System.Collections.Generic;  
3 using System.Linq;  
4 using System.Text;  
5 using System.Threading.Tasks;  
6  
7 namespace StringTutorial  
8 {  
9     class IndexOfDemo  
10    {  
11        public static void Main(string[] args)  
12        {  
13            String str = "This is text";  
14  
15  
16            // Tìm vị trí xuất hiện ký tự 'i' đầu tiên.  
17            int idx = str.IndexOf('i'); // ==> 2  
18            Console.WriteLine("- IndexOf('i') = " + idx);  
19  
20            // Tìm vị trí xuất hiện ký tự 'i' đầu tiên  
21            // tính từ chỉ số thứ 4 trở về cuối chuỗi.  
22            idx = str.IndexOf('i', 4); // ==> 5  
23            Console.WriteLine("- indexOf('i',4) = " + idx);
```

```

24
25 // Tìm vị trí xuất hiện chuỗi con "te" đầu tiên.
26 idx = str.IndexOf("te"); // ==> 8
27 Console.WriteLine("- IndexOf('te') = " + idx);
28
29 Console.Read();
30 }
31 }
32
33 }

```

Kết quả chạy ví dụ:

```

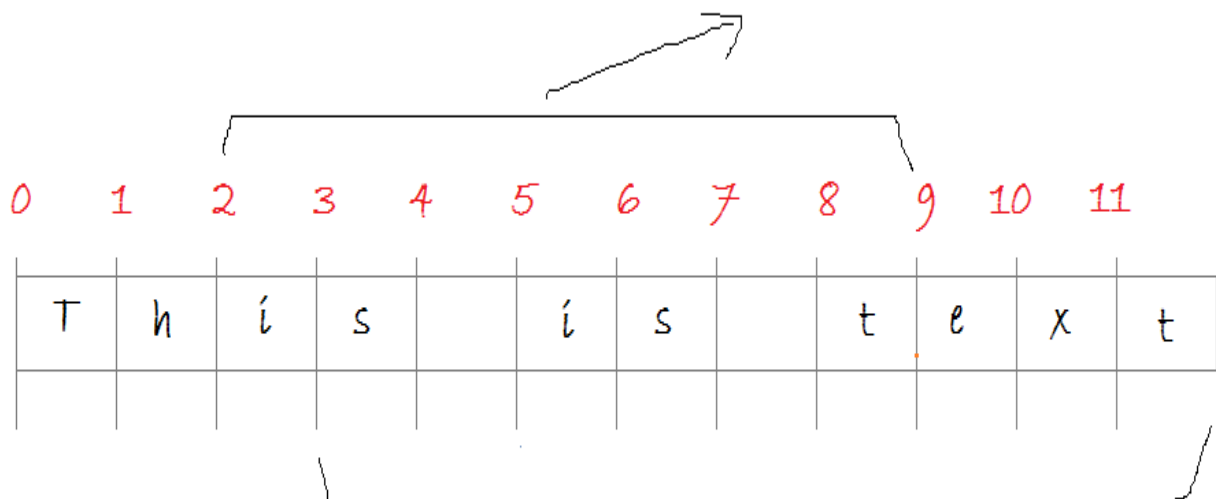
file:///C:/CSHAP_TUTORIAL/MySolution/StringTutorial/bin/Debug/StringTutorial.EXE
- IndexOf('i') = 2
- indexOf('i',4) = 5
- IndexOf('te') = 8

```

4.1.4- Substring(..)

String str = "This is text";

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



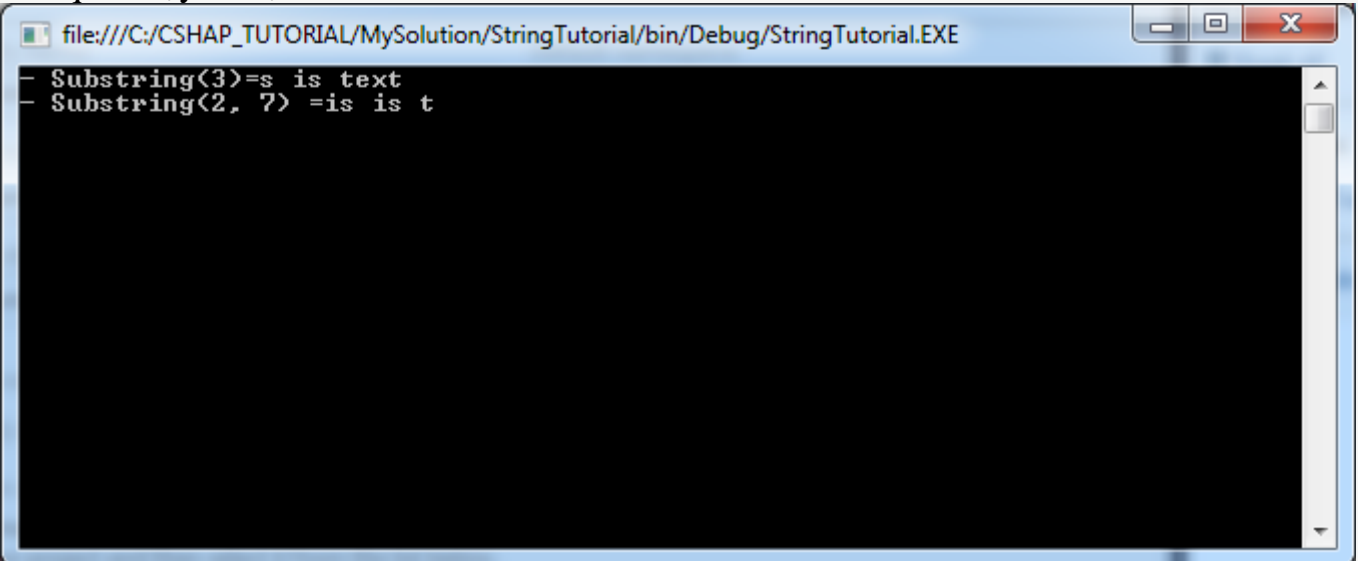
String substr = str.substring(3)


```

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace StringTutorial
8  {
9      class SubstringDemo
10     {
11         public static void Main(string[] args)
12         {
13             string str = "This is text";
14
15
16             // Trả về chuỗi con từ chỉ số thứ 3 tới cuối chuỗi.
17             string substr = str.Substring(3);
18
19             Console.WriteLine("- Substring(3)=" + substr);
20
21             // Trả về chuỗi con từ chỉ số thứ 2, và độ dài 7 ký tự
22             substr = str.Substring(2, 7);
23
24             Console.WriteLine("- Substring(2, 7) =" + substr);
25
26
27             Console.Read();
28         }
29     }
30
31 }

```

Kết quả chạy ví dụ:



```

file:///C:/CSHAP_TUTORIAL/MySolution/StringTutorial/bin/Debug/StringTutorial.EXE
- Substring(3)=s is text
- Substring(2, 7) =is is t

```

4.1.5- Replace(...)

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

ReplaceDemo.cs

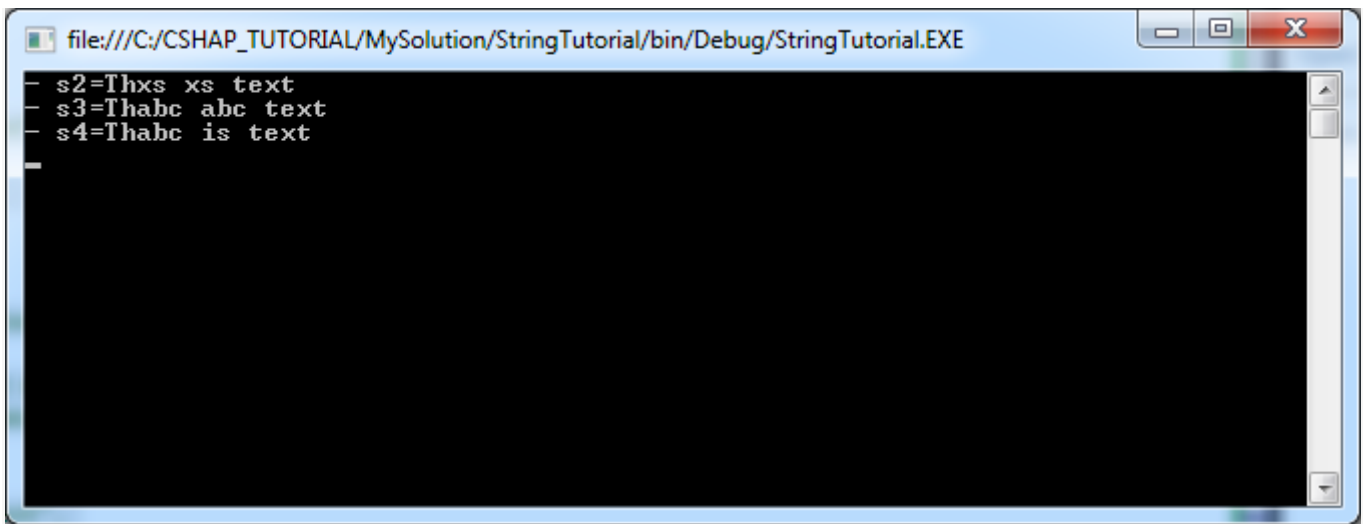
?

```

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace StringTutorial
8  {
9      class ReplaceDemo
10     {
11         public static void Main(string[] args)
12         {
13             String str = "This is text";
14
15             // Thay thế hết các ký tự 'i' bởi ký tự 'x'.
16             String s2 = str.Replace('i', 'x');
17
18             Console.WriteLine("- s2=" + s2);// ==> "Thxs xs text".
19
20
21             // Thay thế tất cả các chuỗi con "is" bởi "abc".
22             String s3 = str.Replace("is", "abc");
23
24             Console.WriteLine("- s3=" + s3);// ==> "Thabc abc text".
25
26             // Thay thế chuỗi con "is" xuất hiện lần đầu bởi "abc".
27             String s4 = ReplaceFirst(str, "is", "abc");
28
29             Console.WriteLine("- s4=" + s4);// ==> "Thabc is text".
30
31             Console.Read();
32         }
33
34         // Thay thế chuỗi con xuất hiện lần đầu tiên.
35         static string ReplaceFirst(string text, string search, string replace)
36         {
37             int pos = text.IndexOf(search);
38             if (pos < 0)
39             {
40                 return text;
41             }
42             return text.Substring(0, pos) + replace + text.Substring(pos + search.Length);
43         }
44     }
45
46 }

```

Kết quả chạy ví dụ:



```
file:///C:/CSHAP_TUTORIAL/MySolution/StringTutorial/bin/Debug/StringTutorial.EXE
- s2=Thxs xs text
- s3=Thabc abc text
- s4=Thabc is text
```

4.1.6- Các ví dụ khác

StringOtherDemo.cs

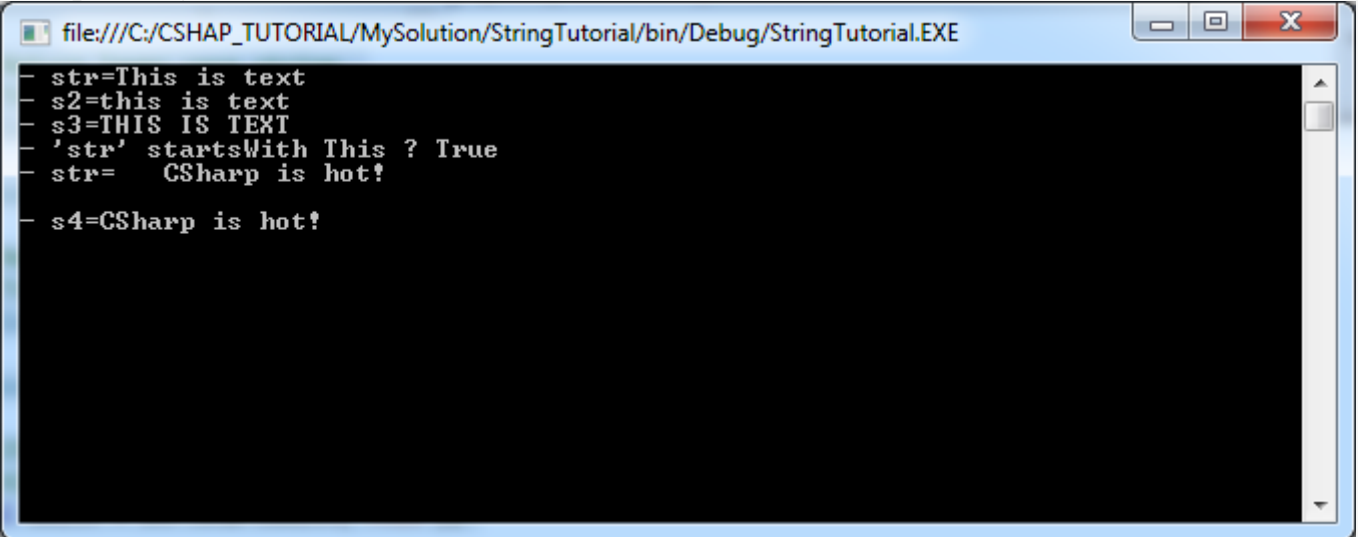
```
?
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace StringTutorial
8  {
9      class StringOtherDemo
10     {
11         public static void Main(string[] args)
12         {
13             String str = "This is text";
14
15             Console.WriteLine("- str=" + str);
16
17             // Trả về chuỗi chữ thường.
18             String s2 = str.ToLower();
19
20             Console.WriteLine("- s2=" + s2);
21
22             // Trả về chuỗi chữ hoa.
23             String s3 = str.ToUpper();
24
25             Console.WriteLine("- s3=" + s3);
26
27             // Kiểm tra xem chuỗi có bắt đầu bởi chuỗi con "This" hay không.
28             bool swith = str.StartsWith("This");
29
30             Console.WriteLine("- 'str' startsWith This ? " + swith);
31
32
33             // Một String với khoảng trắng phía trước và sau.
34             // \t là ký tự TAB.
35             // \n là ký tự xuống dòng.
```

```

36         str = " \t CSharp is hot! \t \n ";
37
38         Console.WriteLine("- str=" + str);
39
40         // Trả về một String mới loại bỏ khoảng trắng ở đầu và cuối chuỗi.
41         String s4 = str.Trim();
42
43         Console.WriteLine("- s4=" + s4);
44
45         Console.Read();
46     }
47 }
48
49 }

```

Kết quả chạy ví dụ:



```

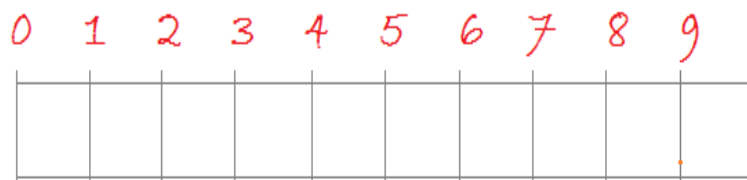
file:///C:/CSHAP_TUTORIAL/MySolution/StringTutorial/bin/Debug/StringTutorial.EXE
- str=This is text
- s2=this is text
- s3=THIS IS TEXT
- 'str' startsWith This ? True
- str= CSharp is hot!
- s4=CSharp is hot!

```

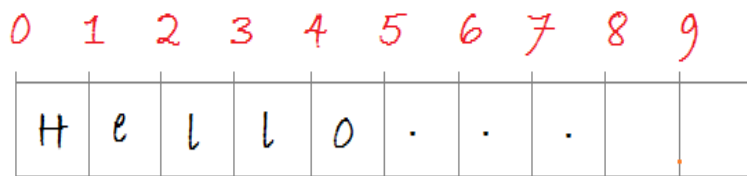
5- StringBuilder

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.

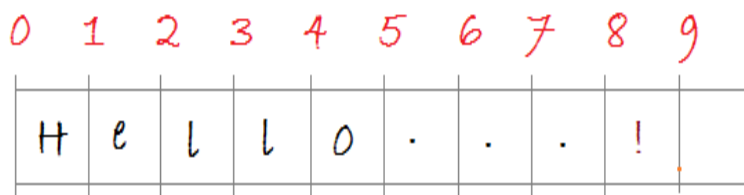
```
StringBuilder sb = new StringBuilder(10);
```



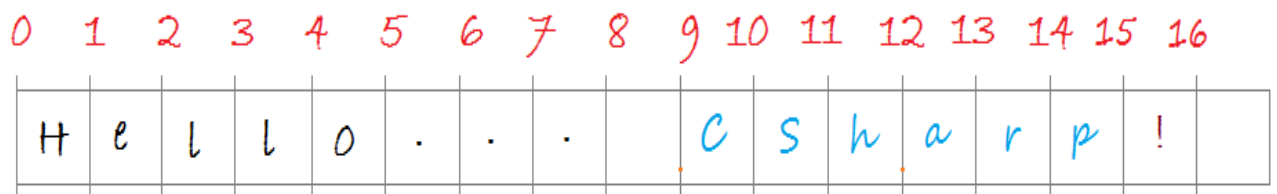
```
sb.Append("Hello...");
```



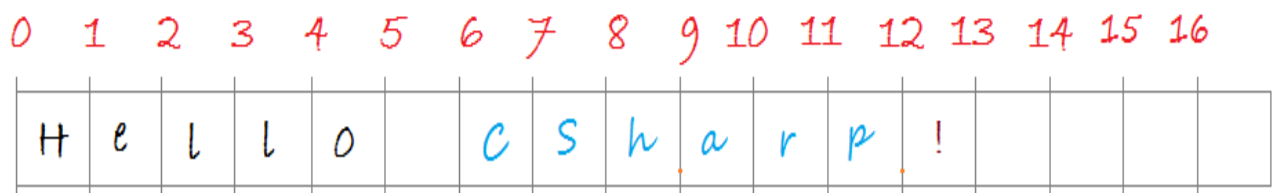
```
sb.Append("!");
```



```
sb.Insert(8, "CSharp");
```



```
sb.Remove(5, 3);
```



StringBuilderDemo.cs

```
?  
1 using System;  
2 using System.Collections.Generic;  
3 using System.Linq;  
4 using System.Text;  
5 using System.Threading.Tasks;  
6  
7 namespace StringTutorial
```

```

8  {
9  class StringBuilderDemo
10 {
11     public static void Main(string[] args)
12     {
13
14         // Tạo đối tượng StringBuilder
15         // Hiện tại chưa có dữ liệu trên StringBuilder.
16         StringBuilder sb = new StringBuilder(10);
17
18         // Nối thêm một chuỗi con
19         sb.Append("Hello...");
20         Console.WriteLine("- sb after appends a string: " + sb);
21
22         // Nối thêm một ký tự.
23         char c = '!';
24         sb.Append(c);
25         Console.WriteLine("- sb after appending a char: " + sb);
26
27         // Trộn một String tại chỉ số 5.
28         sb.Insert(8, " CSharp");
29         Console.WriteLine("- sb after insert string: " + sb);
30
31         // Xóa một chuỗi con bắt đầu tại chỉ số 5, với 3 ký tự.
32         sb.Remove(5, 3);
33
34         Console.WriteLine("- sb after remove: " + sb);
35
36         // Lấy ra string trong StringBuilder.
37         String s = sb.ToString();
38
39         Console.WriteLine("- String of sb: " + s);
40
41         Console.Read();
42     }
43 }
44
45 }

```

Kết quả chạy ví dụ:

```
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!
```

Hướng dẫn sử dụng C# Generics

- 1- Kiểu Generic Class, Interface
 - 1.1- Class Generics
 - 1.2- Thừa kế class Generics
 - 1.3- Interface Generics
 - 1.4- Sử dụng Generic với Exception
- 2- Phương thức generics
- 3- Khởi tạo đối tượng Generic
- 4- Mảng Generic

1- Kiểu Generic Class, Interface

1.1- Class Generics

Ví dụ dưới đây định nghĩa ra một class generics. **KeyValue** là một class generics nó chứa một cặp khóa và giá trị (key/value).

KeyValue.cs

```
?
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace GenericsTutorial
8  {
9      public class KeyValue<K, V>
10     {
11
12         private K key;
13         private V value;
14
15         public KeyValue(K key, V value)
16         {
17             this.key = key;
```

```

18         this.value = value;
19     }
20
21     public K GetKey()
22     {
23         return key;
24     }
25
26     public void SetKey(K key)
27     {
28         this.key = key;
29     }
30
31     public V GetValue()
32     {
33         return value;
34     }
35
36     public void SetValue(V value)
37     {
38         this.value = value;
39     }
40
41 }
42
43 }

```

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

Hãy xem ví dụ sử dụng class **KeyValue**.

KeyValueDemo.cs

```

?
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace GenericsTutorial
8  {
9      public class KeyValueDemo
10     {
11
12         public static void Main(string[] args)
13         {
14
15
16             // Tạo một đối tượng KeyValue
17             // int: Số điện thoại (K = int)

```

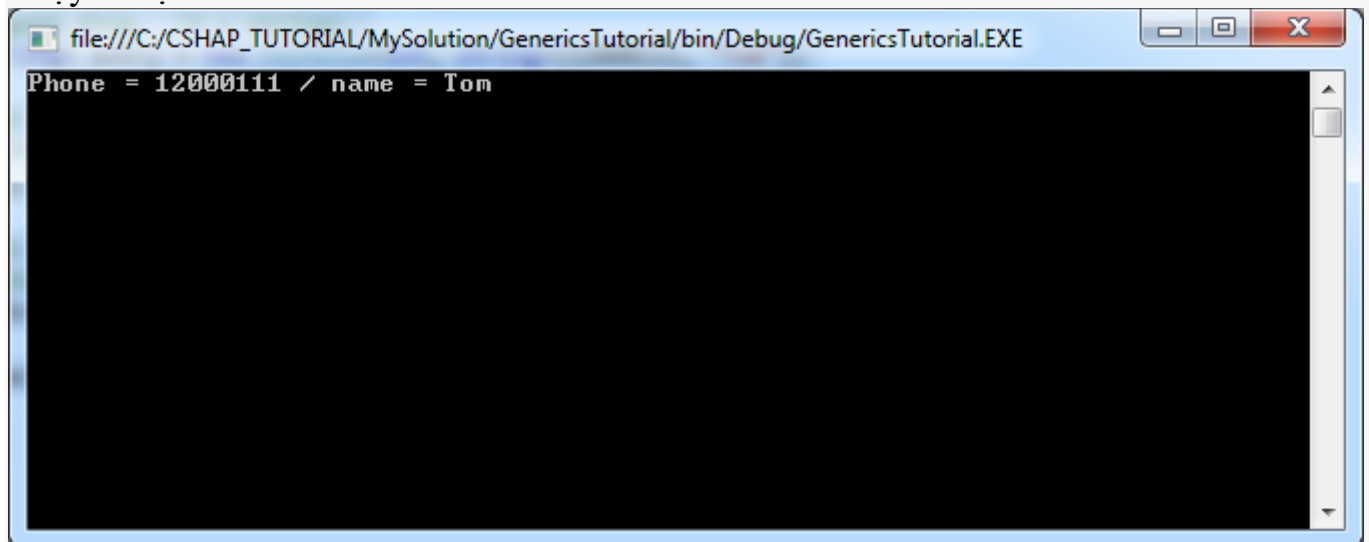


```

18 // string: Tên người dùng. (V = string).
19 KeyValue<int, string> entry = new KeyValue<int, string>(12000111, "Tom");
20
21 // C# hiểu kiểu trả về là int (K = int).
22 int phone = entry.GetKey();
23
24 // C# hiểu kiểu trả về là string (V = string).
25 string name = entry.GetValue();
26
27 Console.WriteLine("Phone = " + phone + " / name = " + name);
28
29 Console.Read();
30 }
31
32 }
33
34 }

```

Chạy ví dụ:



1.2- Thừa kế class Generics

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

Ví dụ 1:

PhoneNumberEntry.cs

```

?
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6
7 namespace GenericsTutorial
8 {
9
10 // Class này mở rộng từ class KeyValue<K,V>
11 // Và chỉ định rõ K,V
12 // K = int (Số điện thoại).
13 // V = string (Tên người dùng).

```

```

14 public class PhoneNameEntry : KeyValue<int, string>
15 {
16
17     public PhoneNameEntry(int key, string value)
18         : base(key, value)
19     {
20
21     }
22
23 }
24
25 }

```

Ví dụ sử dụng **PhoneNameEntry**:

PhoneNameEntryDemo.cs

```

?
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6
7 namespace GenericsTutorial
8 {
9     public class PhoneNameEntryDemo
10    {
11
12        public static void Main(string[] args)
13        {
14
15            PhoneNameEntry entry = new PhoneNameEntry(12000111, "Tom");
16
17            // C# hiểu kiểu trả về là int.
18            int phone = entry.GetKey();
19
20            // C# hiểu kiểu trả về là string.
21            string name = entry.GetValue();
22
23            Console.WriteLine("Phone = " + phone + " / name = " + name);
24
25            Console.Read();
26
27        }
28
29    }
30
31 }

```

Ví dụ 2:

StringAndValueEntry.cs

```

?
1 using System;

```

```

2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace GenericsTutorial
8  {
9
10     // Class này mở rộng class KeyValue<K,V>
11     // Xác định rõ kiểu tham số K là String.
12     // Vẫn giữ kiểu tham số generic V.
13     public class StringAndValueEntry<V> : KeyValue<string, V>
14     {
15
16         public StringAndValueEntry(string key, V value)
17             : base(key, value)
18         {
19         }
20
21     }
22
23 }

```

Ví dụ sử dụng **StringAndValueEntry** class:

StringAndValueEntryDemo.cs

```

?
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace GenericsTutorial
8  {
9      public class StringAndValueEntryDemo
10     {
11
12         public static void main(String[] args)
13         {
14
15             // (Mã nhân viên, Tên nhân viên).
16             // V = string (Tên nhân viên)
17             StringAndValueEntry<String> entry = new StringAndValueEntry<String>("E001", "
18
19             String empNumber = entry.GetKey();
20
21             String empName = entry.GetValue();
22
23             Console.WriteLine("Emp Number = " + empNumber);
24             Console.WriteLine("Emp Name = " + empName);
25

```

```
26         Console.Read();
27
28     }
29 }
30
31 }
```

Ví dụ 3:

KeyValueInfo.cs

```
?
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace GenericsTutorial
8  {
9
10     // Class này mở rộng class KeyValue<K,V>
11     // Nó có thêm một tham số generics I.
12     public class KeyValueInfo<K, V, I> : KeyValue<K, V>
13     {
14
15         private I info;
16
17         public KeyValueInfo(K key, V value)
18             : base(key, value)
19         {
20         }
21
22         public KeyValueInfo(K key, V value, I info)
23             : base(key, value)
24         {
25             this.info = info;
26         }
27
28         public I GetInfo()
29         {
30             return info;
31         }
32
33         public void SetInfo(I info)
34         {
35             this.info = info;
36         }
37
38     }
39
40 }
```

1.3- Interface Generics

Một Interface có tham số Generics:

GenericInterface.cs

```
?
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace GenericsTutorial
8  {
9      public interface GenericInterface<G>
10     {
11
12         G DoSomething();
13
14     }
15
16 }
```

Ví dụ một class thi hành Interface:

GenericInterfaceImpl.cs

```
?
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace GenericsTutorial
8  {
9
10     public class GenericInterfaceImpl<G> : GenericInterface<G>
11     {
12
13         private G something;
14
15         public G DoSomething()
16         {
17             return something;
18         }
19
20     }
21
22 }
```

1.4- Sử dụng Generic với Exception

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

MyException.cs

```
?
1  using System;
```

```

2    using System.Collections.Generic;
3    using System.Linq;
4    using System.Text;
5    using System.Threading.Tasks;
6
7    namespace GenericsTutorial
8    {
9        class MyException<E> : ApplicationException
10       {
11
12       }
13
14   }

```

Sử dụng Generic Exception hợp lệ:

UsingGenericExceptionValid01.cs

```

?
1    using System;
2    using System.Collections.Generic;
3    using System.Linq;
4    using System.Text;
5    using System.Threading.Tasks;
6
7    namespace GenericsTutorial
8    {
9        class UsingGenericExceptionValid01
10       {
11
12       public void SomeMethod()
13       {
14           try
15           {
16               // ...
17
18           }
19           // Hợp lệ
20           catch (MyException<string> e)
21           {
22               // Làm gì đó ở đây.
23           }
24           // Hợp lệ
25           catch (MyException<int> e)
26           {
27               // Làm gì đó ở đây.
28           }
29           catch (Exception e)
30           {
31           }
32       }
33
34   }

```

35

36 }

Sử dụng Generics Exception hợp lệ:

UsingGenericExceptionValid02.cs

?

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace GenericsTutorial
8  {
9      class UsingGenericExceptionValid02<K>
10     {
11         public void SomeMethod()
12         {
13             try
14             {
15                 // ...
16
17             }
18             // Hợp lệ
19             catch (MyException<string> e)
20             {
21                 // Làm gì đó ở đây.
22             }
23             // Hợp lệ
24             catch (MyException<K> e)
25             {
26                 // Làm gì đó ở đây.
27             }
28             catch (Exception e)
29             {
30             }
31         }
32     }
33
34 }
```

Sử dụng Generics Exception không hợp lệ:

UsingGenericExceptionInvalid.cs

?

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace GenericsTutorial
8  {
```

```

9      class UsingGenericExceptionInvalid
10     {
11         public void SomeMethod()
12         {
13             try
14             {
15                 // ...
16
17             }
18             // Hợp lệ
19             catch (MyException<string> e)
20             {
21                 // Làm gì đó ở đây.
22             }
23             // Không hợp lệ (Không hiểu tham số K). *****
24             catch (MyException<K> e)
25             {
26                 // Làm gì đó ở đây.
27             }
28             catch (Exception e)
29             {
30             }
31         }
32     }
33
34 }

```

2- Phương thức generics



Một phương thức trong class hoặc Interface có thể được generic hóa (generify).

MyUtils.cs

```

?
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace GenericsTutorial
8  {
9      public class MyUtils
10     {
11         // <K,V> : Nói rằng phương thức này có 2 kiểu tham số K,V
12         // Phương thức trả về kiểu K.
13         public static K GetKey<K, V>(KeyValue<K, V> entry)
14         {
15             K key = entry.GetKey();
16             return key;
17         }
18
19         // <K,V> : Nói rằng phương thức này có 2 kiểu tham số K,V

```



```

20 // Phương thức trả về kiểu V.
21 public static V GetValue<K, V>(KeyValue<K, V> entry)
22 {
23     V value = entry.GetValue();
24     return value;
25 }
26
27 // List<E>: Danh sách chứa các phần tử kiểu E
28 // Phương thức trả về kiểu E.
29 public static E GetFirstElement<E>(List<E> list, E defaultValue)
30 {
31     if (list == null || list.Count == 0)
32     {
33         return defaultValue;
34     }
35     E first = list.ElementAt(0);
36     return first;
37 }
38
39 }
40
41 }

```

Ví dụ sử dụng phương thức generics:

MyUtilsDemo.cs

```

?
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6
7 namespace GenericsTutorial
8 {
9     public class MyUtilsDemo
10    {
11
12        public static void Main(string[] args)
13        {
14
15            // K = int: Phone
16            // V = string: Name
17            KeyValue<int, string> entry1 = new KeyValue<int, String>(12000111, "Tom");
18            KeyValue<int, string> entry2 = new KeyValue<int, String>(12000112, "Jerry");
19
20            // (K = int).
21            int phone = MyUtils.GetKey(entry1);
22            Console.WriteLine("Phone = " + phone);
23
24            // Một danh sách chứa các phần tử kiểu KeyValue<int,string>.
25            List<KeyValue<int, string>> list = new List<KeyValue<int, string>>();

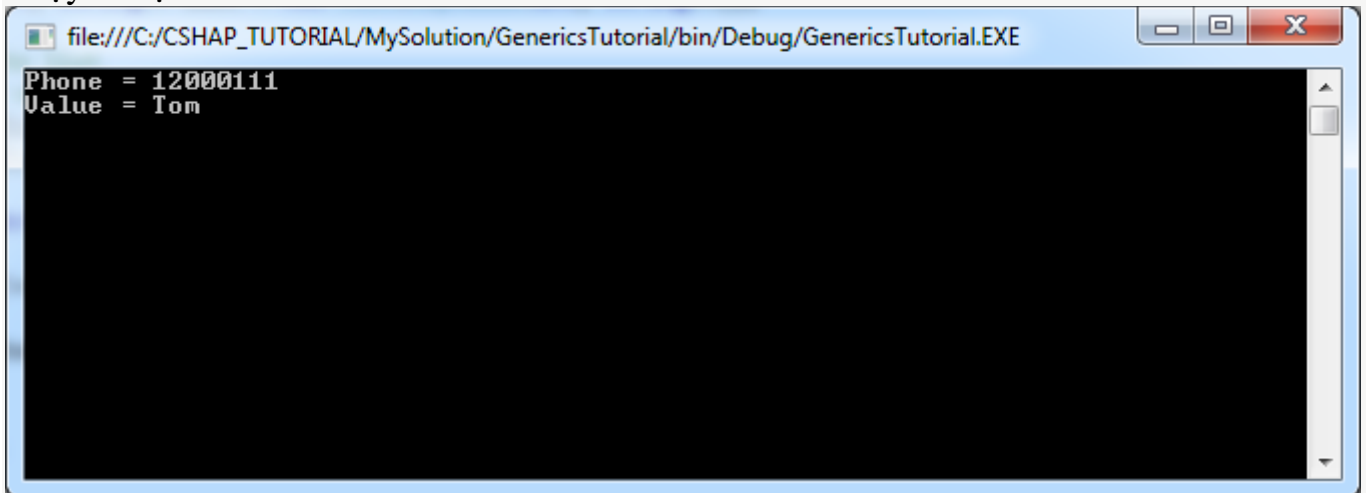
```

```

26
27     // Thêm phần tử vào danh sách.
28     list.Add(entry1);
29     list.Add(entry2);
30
31     KeyValue<int, string> firstEntry = MyUtils.GetFirstElement(list, null);
32
33     if (firstEntry != null)
34     {
35         Console.WriteLine("Value = " + firstEntry.GetValue());
36     }
37
38     Console.Read();
39 }
40
41 }
42
43 }

```

Chạy ví dụ:



3- Khởi tạo đối tượng Generic

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

```

?
1 public void DoSomething<T>()
2 {
3
4     // Khởi tạo đối tượng Generic
5     T t = new T(); // Error
6
7 }

```

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

GenericInitializationExample.cs

```

?
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6

```

```

7 namespace GenericsTutorial
8 {
9     class GenericInitializationExample
10    {
11
12
13        // Tham số T phải là kiểu có cấu tử mặc định.
14        public void DoSomething<T>()
15            where T : new()
16        {
17            T t = new T();
18        }
19
20        // Tham số K phải là kiểu có cấu tử mặc định
21        // và mở rộng từ class KeyValue.
22        public void ToDoSomething<K>()
23            where K: KeyValue<K,string>, new( )
24        {
25            K key = new K();
26        }
27
28
29        public T DoDefault<T>()
30        {
31
32            // Trả về null nếu T là kiểu tham số.
33            // Hoặc 0 nếu T là kiểu số.
34            return default(T);
35        }
36    }
37
38 }

```

4- Mảng Generic



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

?

```

1 // Khởi tạo một mảng.
2
3 T[] myArray = new T[10];

```

GenericArrayExample.cs

?

```

1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6
7 namespace GenericsTutorial
8 {
9     class GenericArrayExample

```

```

10    {
11
12    public static T[] FilledArray<T>(T value, int count)
13    {
14        T[] ret = new T[count];
15        for (int i = 0; i < count; i++)
16        {
17            ret[i] = value;
18        }
19        return ret;
20    }
21
22    public static void Main(string[] args)
23    {
24
25        string value = "Hello";
26
27        string[] filledArray = FilledArray<string>(value, 10);
28
29        foreach (string s in filledArray)
30        {
31            Console.WriteLine(s);
32        }
33    }
34    }
35
36    }

```

Hướng dẫn xử lý ngoại lệ trong C#

- 1- Exception là gì?
- 2- Sơ đồ phân cấp
- 3- Bắt ngoại lệ thông qua try-catch
- 4- Khởi try-catch-finally
- 5- Gói một Exception trong một Exception khác
- 6- Một số ngoại lệ thông dụng
 - 6.1- NullReferenceException
 - 6.2- IndexOutOfRangeException

*Cảnh báo không nên sử dụng máy tính
LENOVO vì phần mềm gián điệp*

- 1- Exception là gì?





Trước hết chúng ta hãy xem một ví dụ minh họa sau:

Trong ví dụ này có một đoạn code lỗi nguyên nhân do phép chia cho 0. Việc chia cho 0 gây ra ngoại lệ: **DivideByZeroException**

HelloException.cs

```
?
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace ExceptionTutorial
8  {
9      class HelloException
10     {
11         public static void Main(string[] args)
12         {
13
14             Console.WriteLine("Three");
15
16             // Phép chia này hoàn toàn không có vấn đề.
17             int value = 10 / 2;
18
19             Console.WriteLine("Two");
20
21             // Phép chia này cũng vậy
22             value = 10 / 1;
23
24             Console.WriteLine("One");
25
26             int d = 0;
27
28             // Phép chia này có vấn đề, chia cho 0.
29             // Lỗi đã xảy ra tại đây.
30             value = 10 / d;
31
```

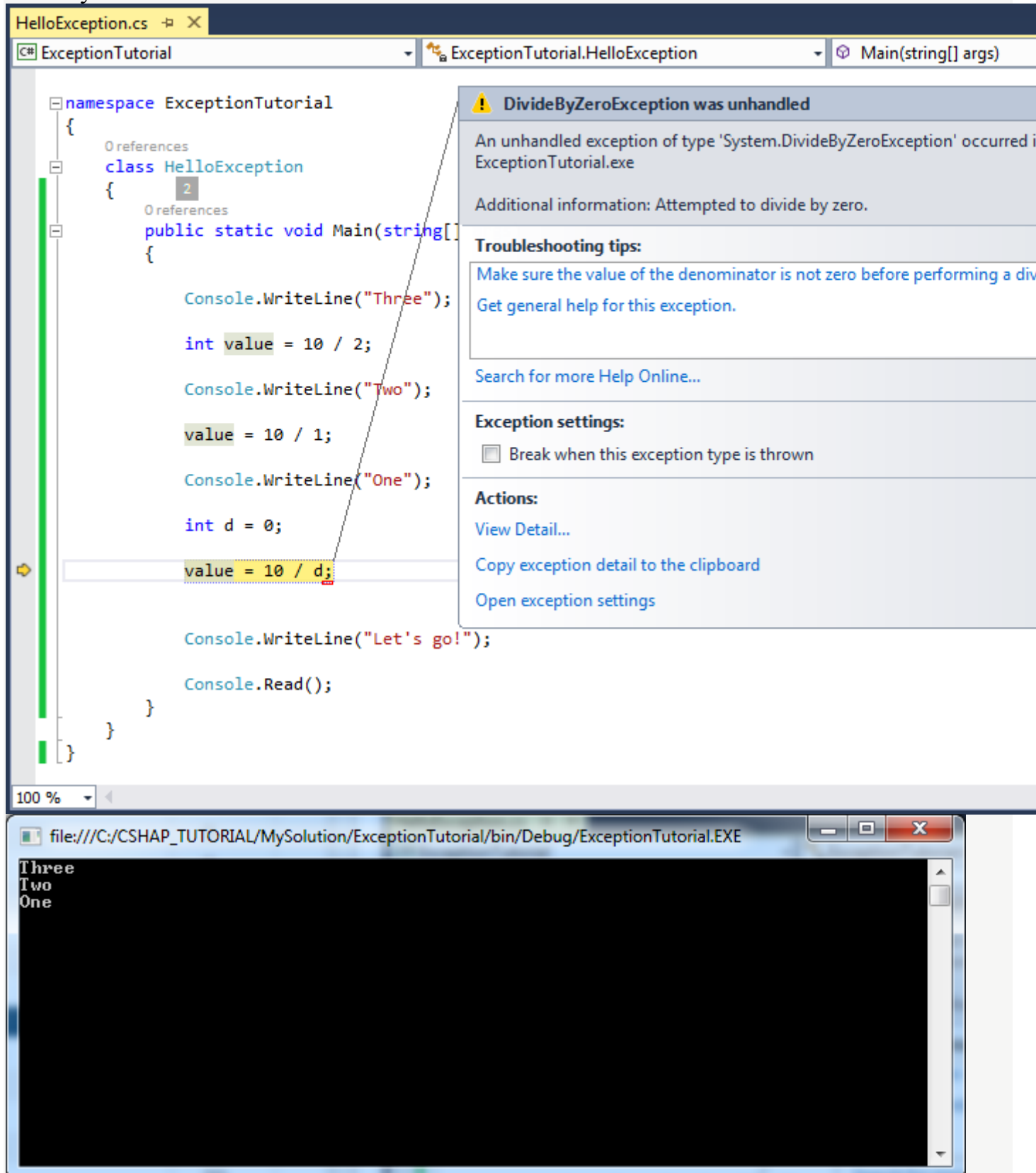
```

32 // Và dòng code dưới đây sẽ không được thực thi.
33 Console.WriteLine("Let's go!");
34
35 Console.Read();
36 }
37 }
38 }

```

Kết quả chạy ví dụ:

Bạn có thể thấy thông báo lỗi trên màn hình Console, thông báo lỗi rất rõ ràng, xảy ra ở dòng thứ mấy trên code.



Hãy xem luồng đi của chương trình qua hình minh họa dưới đây.

- Chương trình đã chạy hoàn toàn bình thường từ các bước (1),(2) cho tới (5)
- Bước thứ (6) xảy ra vấn đề khi chia cho 0.
- Chương trình đã nhảy ra khỏi hàm main, và dòng code thứ (7) đã không được thực hiện.

```

HelloException.cs
C# ExceptionTutorial ExceptionTutorial.HelloException Main(string[] args)
using System.Threading.Tasks;

namespace ExceptionTutorial
{
    class HelloException
    {
        public static void Main(string[] args)
        {
            Console.WriteLine("Three");
            int value = 10 / 2;
            Console.WriteLine("Two");
            value = 10 / 1;
            Console.WriteLine("One");
            int d = 0;
            value = 10 / d;
            Console.WriteLine("Let's go!");
            Console.Read();
        }
    }
}

```

Lỗi làm ngừng chương trình xảy ra tại đây

Chúng ta sẽ sửa code của ví dụ trên.

HelloCatchException.cs

```

?
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6
7 namespace ExceptionTutorial
8 {
9     class HelloCatchException

```

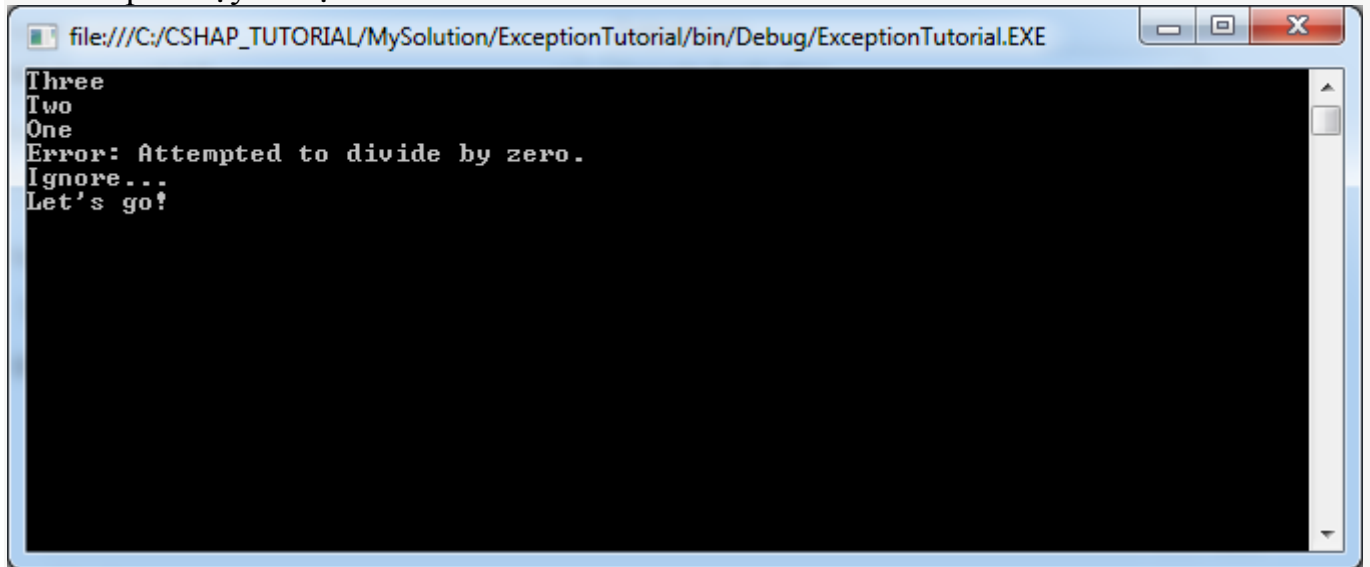
```

10  {
11      public static void Main(string[] args)
12      {
13
14          Console.WriteLine("Three");
15
16          // Phép chia này hoàn toàn không có vấn đề.
17          int value = 10 / 2;
18
19
20          Console.WriteLine("Two");
21
22          // Phép chia này cũng vậy
23          value = 10 / 1;
24
25
26          Console.WriteLine("One");
27
28
29          int d = 0;
30
31          try
32          {
33
34              // Phép chia này có vấn đề, chia cho 0.
35              // Lỗi đã xảy ra tại đây.
36              value = 10 / d;
37
38              // Dòng code này sẽ không được chạy.
39              Console.WriteLine("Value =" + value);
40          }
41          catch (DivideByZeroException e)
42          {
43
44              // Các dòng lệnh trong catch được thực thi
45              Console.WriteLine("Error: " + e.Message);
46
47              // Các dòng lệnh trong catch được thực thi
48              Console.WriteLine("Ignore...");
49
50          }
51
52          // Dòng lệnh này được thực thi.
53          Console.WriteLine("Let's go!");
54
55
56          Console.Read();
57      }
58  }
59

```


60 }

Và kết quả chạy ví dụ:



The screenshot shows a Windows command prompt window with the title bar "file:///C:/CSHAP_TUTORIAL/MySolution/ExceptionTutorial/bin/Debug/ExceptionTutorial.EXE". The window contains the following text:

```
Three
Two
One
Error: Attempted to divide by zero.
Ignore...
Let's go!
```

Chúng ta sẽ giải thích bằng hình minh họa dưới đây về luồng đi của chương trình.

- Các bước (1)-(6) hoàn toàn bình thường.
- Ngoại lệ xảy ra tại bước (7), vấn đề chia cho 0.
- Lập tức nó nhảy vào thực thi lệnh trong khối catch, bước (8) bị bỏ qua.
- Bước (9), (10) được thực hiện.
- Bước (11), (12) được thực hiện.

```

HelloCatchException.cs
C# ExceptionTutorial
ExceptionTutorial.HelloCatchExcept
Main(string[] args)

namespace ExceptionTutorial
{
    0 references
    class HelloCatchException
    {
        0 references
        public static void Main(string[] args)
        {
            Console.WriteLine("Three"); (1)

            int value = 10 / 2; (2)

            Console.WriteLine("Two"); (3)

            value = 10 / 1; (4)

            Console.WriteLine("One"); (5)

            int d = 0; (6)

            try
            {
                value = 10 / d; (7)
                Console.WriteLine("Value =" + value); (8)
            }
            catch (DivideByZeroException e)
            {
                Console.WriteLine("Error: " + e.Message); (9)
                Console.WriteLine("Ignore..."); (10)
            }

            Console.WriteLine("Let's go!"); (11)

            Console.Read(); (12)
        }
    }
}

```

2- Sơ đồ phân cấp

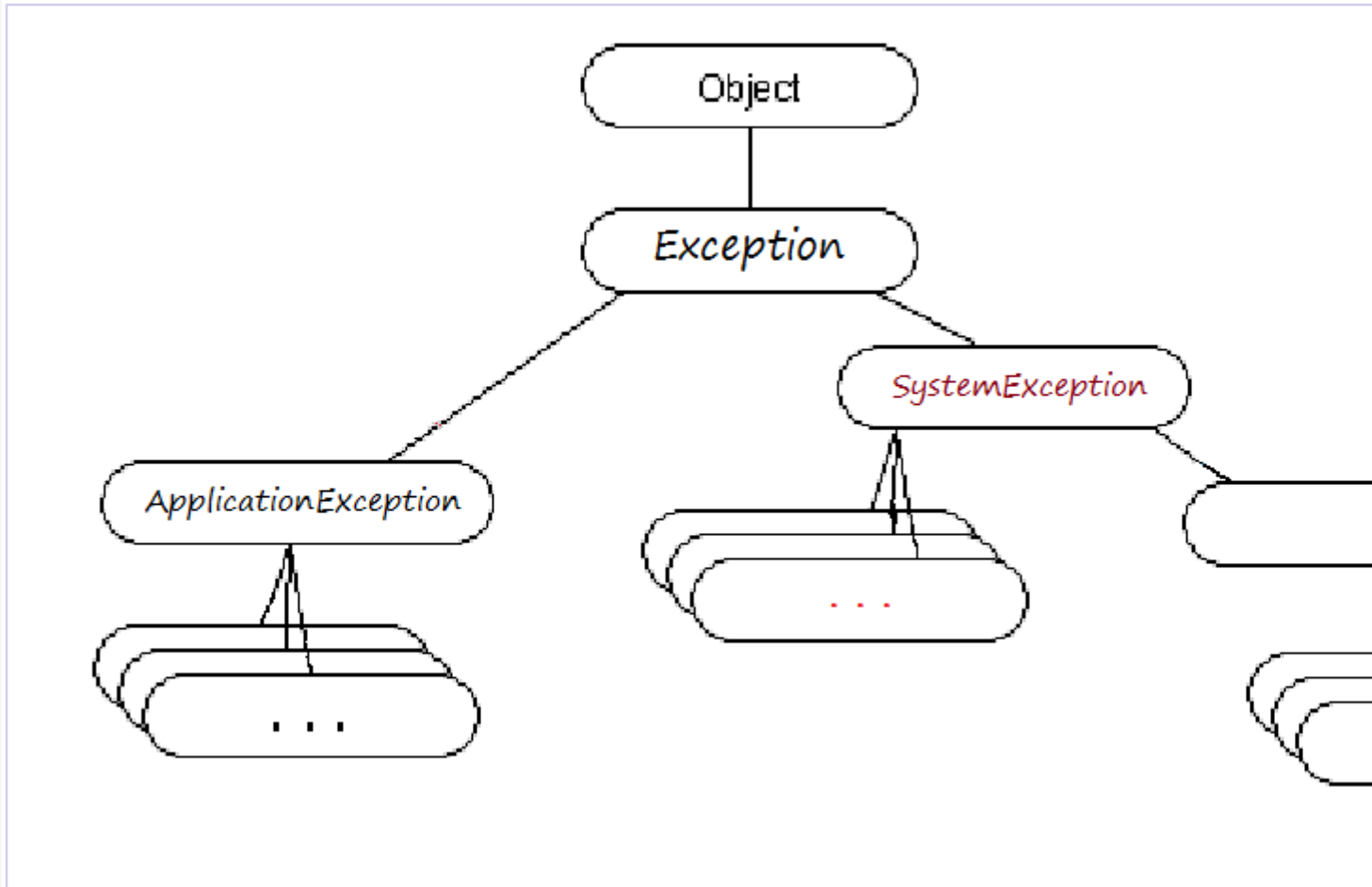




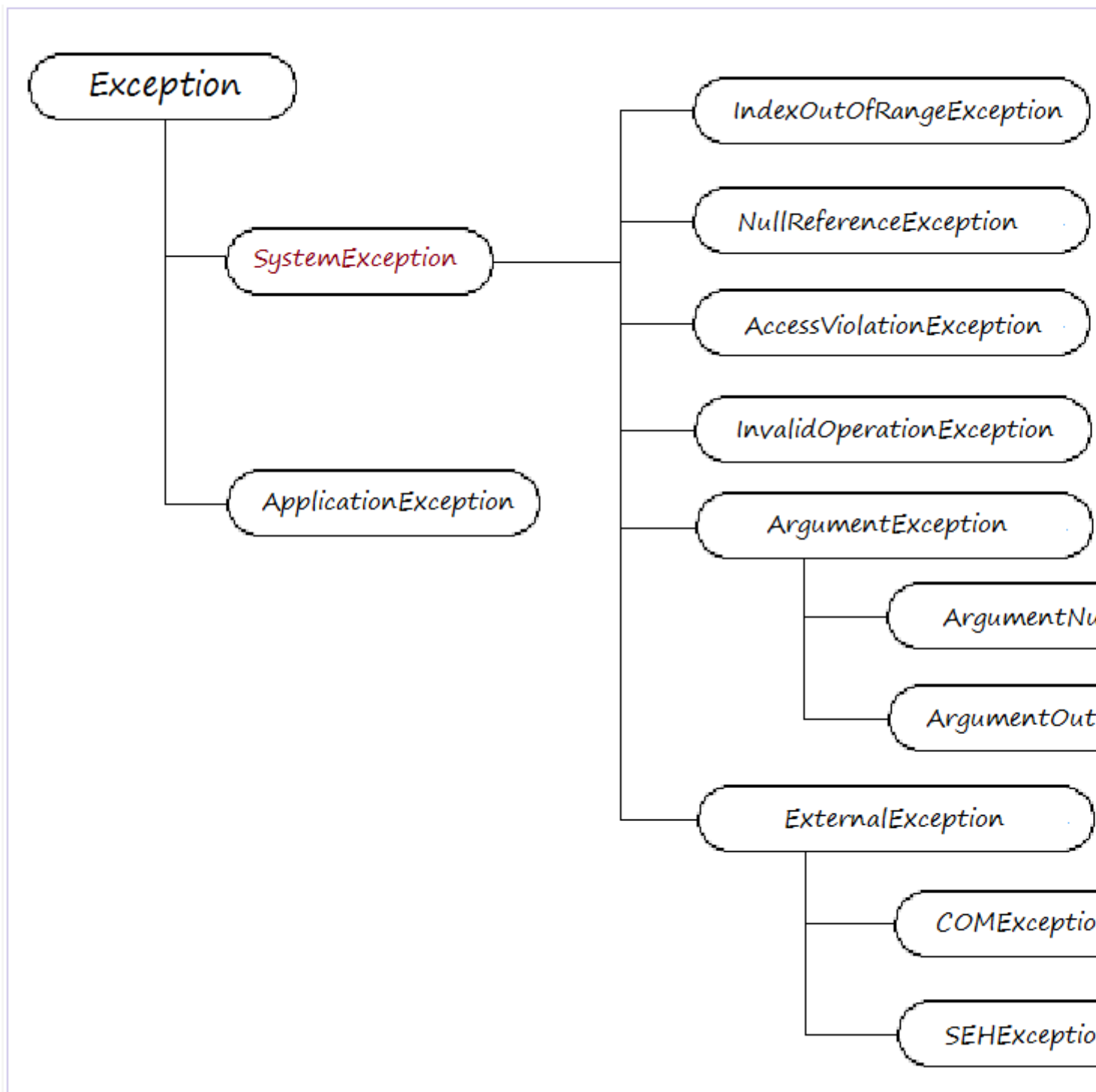
Đây là mô hình sơ đồ phân cấp của Exception trong CSharp.

- 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**.



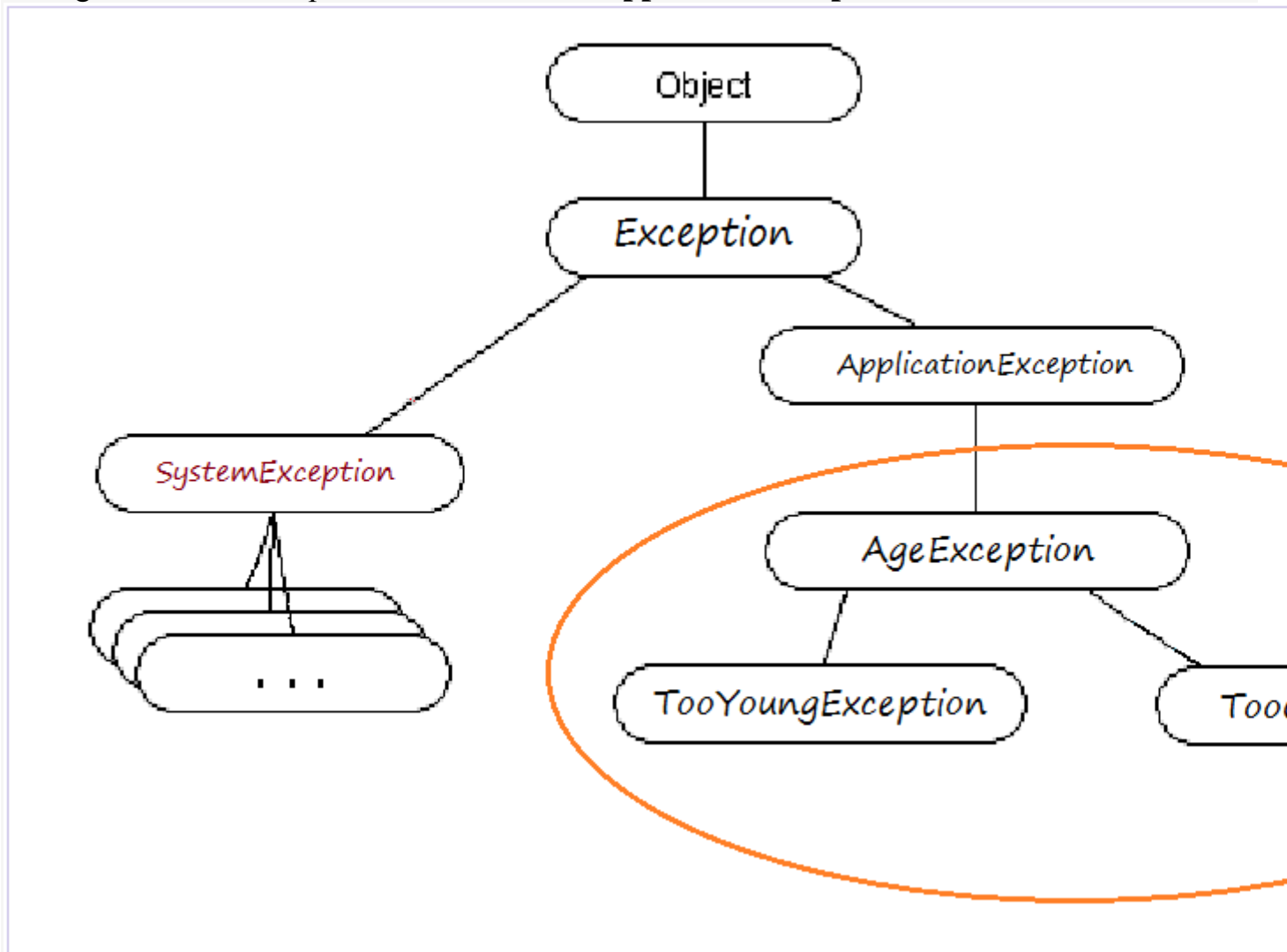
Kiểu ngoại lệ	Mô tả
Exception	Class cơ bản của mọi ngoại lệ.
SystemException	Class cơ bản của mọi ngoại lệ phát ra tại thời điểm chạy của
IndexOutOfRangeException	Được ném ra tại thời điểm chạy khi tham chiếu vào một phần tử có số không đúng.
NullReferenceException	Ném ra tại thời điểm chạy khi một đối 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 khi không cho phép thông số null truyền vào.

ArgumentOutOfRangeException	Class này là con của ArgumentException , nó được ném ra 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 chạy.
COMException	Class này mở rộng từ ExternalException , ngoại lệ đóng gói lỗi từ COM.
SEHException	Class này mở rộng từ ExternalException , nó tóm lược các ngoại lệ từ SEH.

3- Bắt ngoại lệ thông qua try-catch



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



AgeException.cs

```

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace ExceptionTutorial
8  {
9      class AgeException : ApplicationException
10     {
11
12         public AgeException(String message)

```

```

13         : base(message)
14     {
15     }
16
17 }
18
19 class TooYoungException : AgeException
20 {
21
22
23     public TooYoungException(String message)
24         : base(message)
25     {
26
27     }
28
29 }
30
31
32 class TooOldException : AgeException
33 {
34
35
36     public TooOldException(String message)
37         : base(message)
38     {
39
40     }
41
42 }
43 }

```

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

AgeUtils.cs

```

?
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace ExceptionTutorial
8  {
9      class AgeUtils
10     {
11         // Method này làm nhiệm vụ kiểm tra tuổi.
12         // Nếu tuổi nhỏ hơn 18 method sẽ ném ra ngoại lệ TooYoungException
13         // Nếu tuổi lớn hơn 40 method sẽ ném ra ngoại lệ TooOldException
14         public static void checkAge(int age)
15         {
16             if (age < 18)

```

```

17     {
18         // Nếu tuổi nhỏ hơn 18, ngoại lệ sẽ được ném ra
19         // Method này kết thúc tại đây.
20         throw new TooYoungException("Age " + age + " too young");
21     }
22     else if (age > 40)
23     {
24         // Nếu tuổi lớn hơn 40, ngoại lệ sẽ được ném ra.
25         // Method này kết thúc tại đây.
26         throw new TooOldException("Age " + age + " too old");
27     }
28     // Nếu tuổi nằm trong khoảng 18-40.
29     // Đoạn code này sẽ được chạy.
30     Console.WriteLine("Age " + age + " OK!");
31 }
32 }
33
34 }

```

TryCatchDemo1.cs

```

?
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace ExceptionTutorial
8  {
9      class TryCatchDemo1
10     {
11         public static void Main(string[] args)
12         {
13
14             // Bắt đầu tuyển dụng
15             Console.WriteLine("Start Recruiting ...");
16             // Kiểm tra tuổi của bạn.
17             Console.WriteLine("Check your Age");
18             int age = 50;
19
20             try
21             {
22
23                 AgeUtils.checkAge(age);
24
25                 Console.WriteLine("You pass!");
26
27             }
28             catch (TooYoungException e)
29             {
30                 // Thông báo về ngoại lệ "quá trẻ" ..

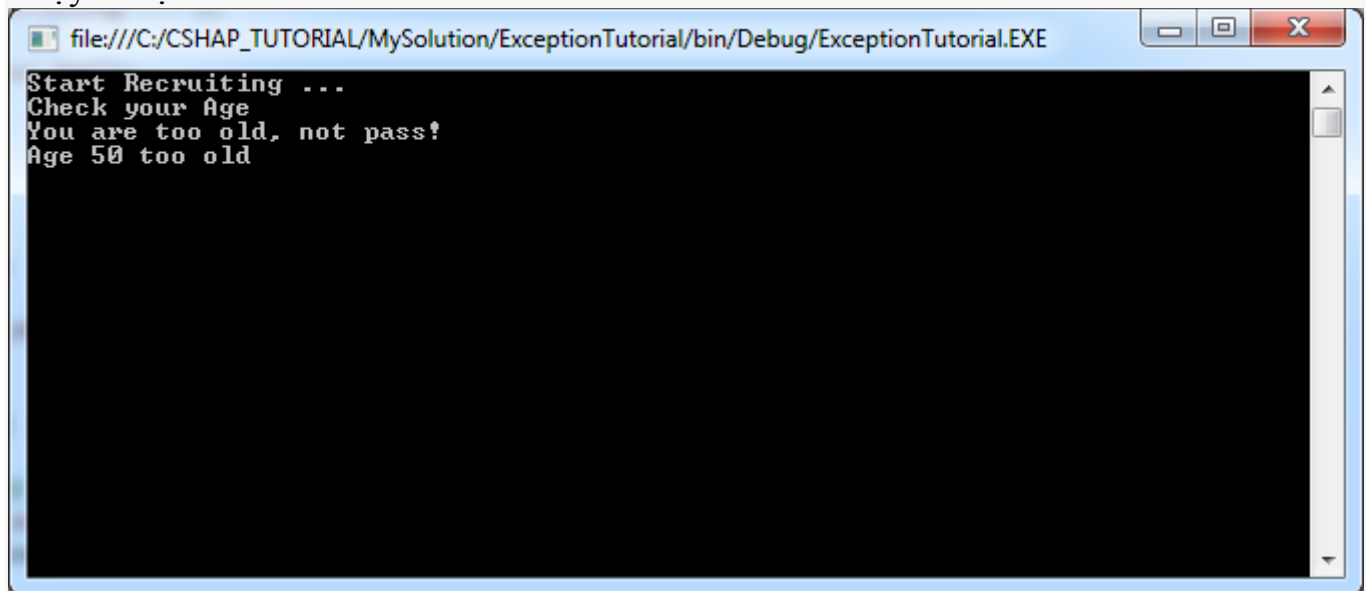
```

```

31         Console.WriteLine("You are too young, not pass!");
32         Console.WriteLine(e.Message);
33
34     }
35     catch (TooOldException e)
36     {
37         // Thông báo về ngoại lệ "quá nhiều tuổi" ..
38         Console.WriteLine("You are too old, not pass!");
39         Console.WriteLine(e.Message);
40
41     }
42
43     Console.Read();
44
45 }
46 }
47
48 }

```

Chạy ví dụ:



Ví dụ dưới đây, chúng ta sẽ gộp 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.

TryCatchDemo2.cs

```

?
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace ExceptionTutorial
8  {
9      class TryCatchDemo2
10     {
11         public static void Main(string[] args)
12         {
13

```

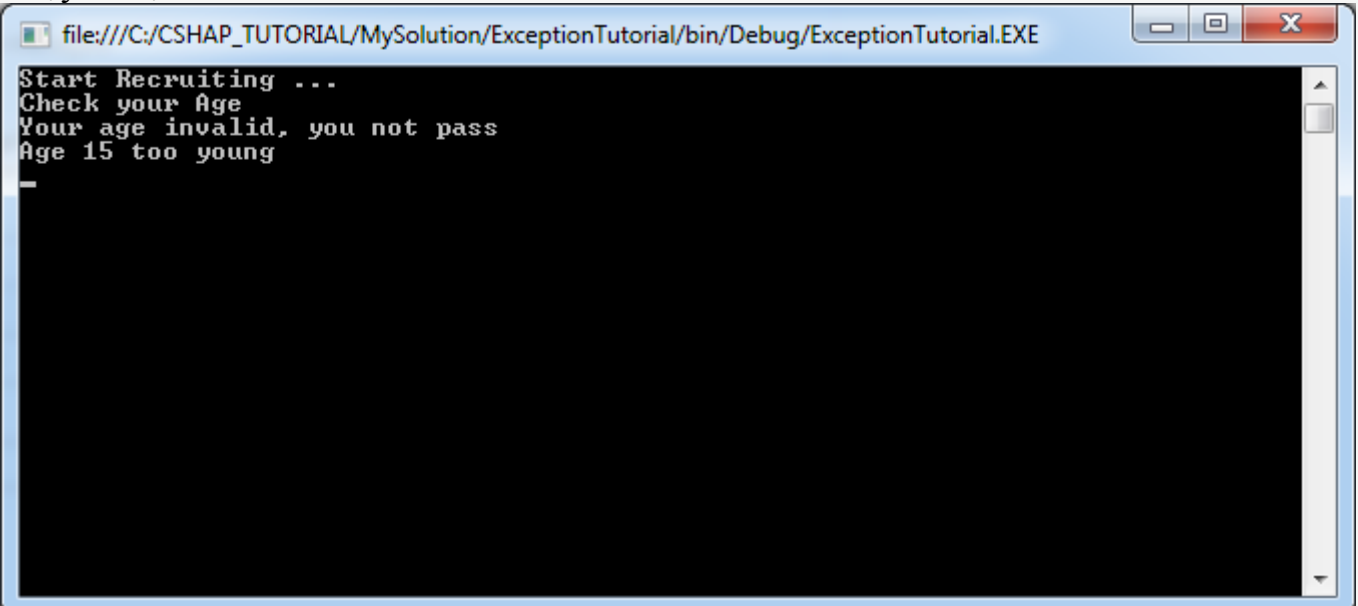


```

14 // Bắt đầu tuyển dụng
15 Console.WriteLine("Start Recruiting ...");
16 // Kiểm tra tuổi của bạn.
17 Console.WriteLine("Check your Age");
18 int age = 15;
19
20 try
21 {
22
23     // Chỗ này có thể bị ngoại lệ TooOldException,
24     // hoặc TooYoungException
25     AgeUtils.checkAge(age);
26
27     Console.WriteLine("You pass!");
28
29 }
30 catch (AgeException e)
31 {
32     // Nếu có ngoại lệ xảy ra, kiểu AgeException
33     // Khởi catch này sẽ được chạy
34
35     Console.WriteLine("Your age invalid, you not pass");
36     Console.WriteLine(e.Message);
37
38 }
39
40 Console.Read();
41 }
42 }
43
44 }

```

Chạy ví dụ:



```

file:///C:/CSHAP_TUTORIAL/MySolution/ExceptionTutorial/bin/Debug/ExceptionTutorial.EXE
Start Recruiting ...
Check your Age
Your age invalid, you not pass
Age 15 too young
-

```

4- Khởi try-catch-finally



Trên kia chúng ta đã làm quen với việc bắt lỗi thông qua khối **try-catch**. Việc xử lý ngoại lệ đầy đủ là **try-catch-finally**.

?

```
1  try {
2
3      // Làm gì đó tại đây
4
5  } catch (Exception1 e) {
6
7      // Làm gì đó tại đây
8
9  } catch (Exception2 e) {
10
11     // Làm gì đó tại đây
12
13 } finally {
14
15     // Khối finally luôn luôn được thực thi
16     // Làm gì đó tại đây.
17
18 }
19
```

TryCatchFinallyDemo.cs

?

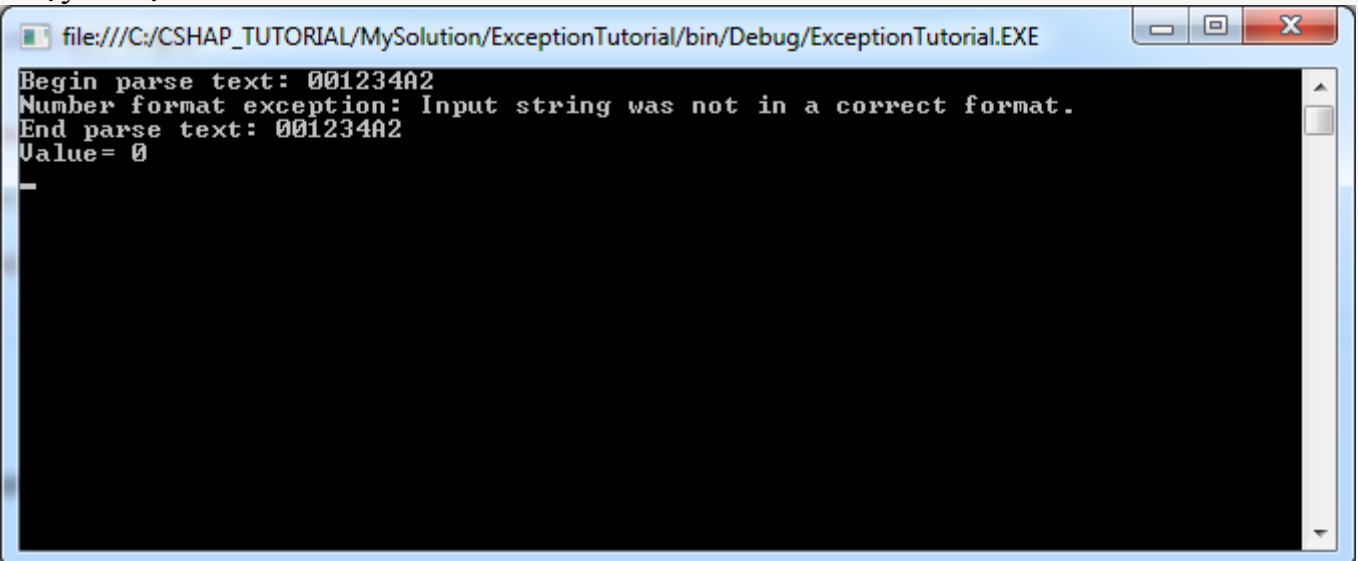
```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace ExceptionTutorial
8  {
9      class TryCatchFinallyDemo
10     {
11         public static void Main(string[] args)
12         {
13
14             String text = "001234A2";
15
16             int value = toInteger(text);
17
18             Console.WriteLine("Value= " + value);
19
20             Console.Read();
21
22         }
23
24         public static int toInteger(String text)
25         {
26             try
27             {
28
```

```

29         Console.WriteLine("Begin parse text: " + text);
30
31         // Tại đây có thể phát sinh ngoại lệ FormatException
32         int value = int.Parse(text);
33
34         return value;
35
36     }
37     catch (FormatException e)
38     {
39
40         // Trong trường hợp 'text' không phải là số.
41         // Khởi catch này sẽ được thực thi.
42         Console.WriteLine("Number format exception: " + e.Message);
43
44         // FormatException xảy ra, trả về 0.
45         return 0;
46
47     }
48     finally
49     {
50
51         Console.WriteLine("End parse text: " + text);
52
53     }
54
55 }
56 }
57
58 }

```

Chạy ví dụ:



```

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 lược đi của chương trình. Khởi **finally** luôn được thực thi.

```
TryCatchFinallyDemo.cs
ExceptionTutorial
ExceptionTutorial.TryCatchFinallyDemo
Main(string[] args)

namespace ExceptionTutorial
{
    class TryCatchFinallyDemo
    {
        public static void Main(string[] args)
        {
            String text = "001234A2";
            int value = toInteger(text);
            Console.WriteLine("Value= " + value);
            Console.Read();
        }

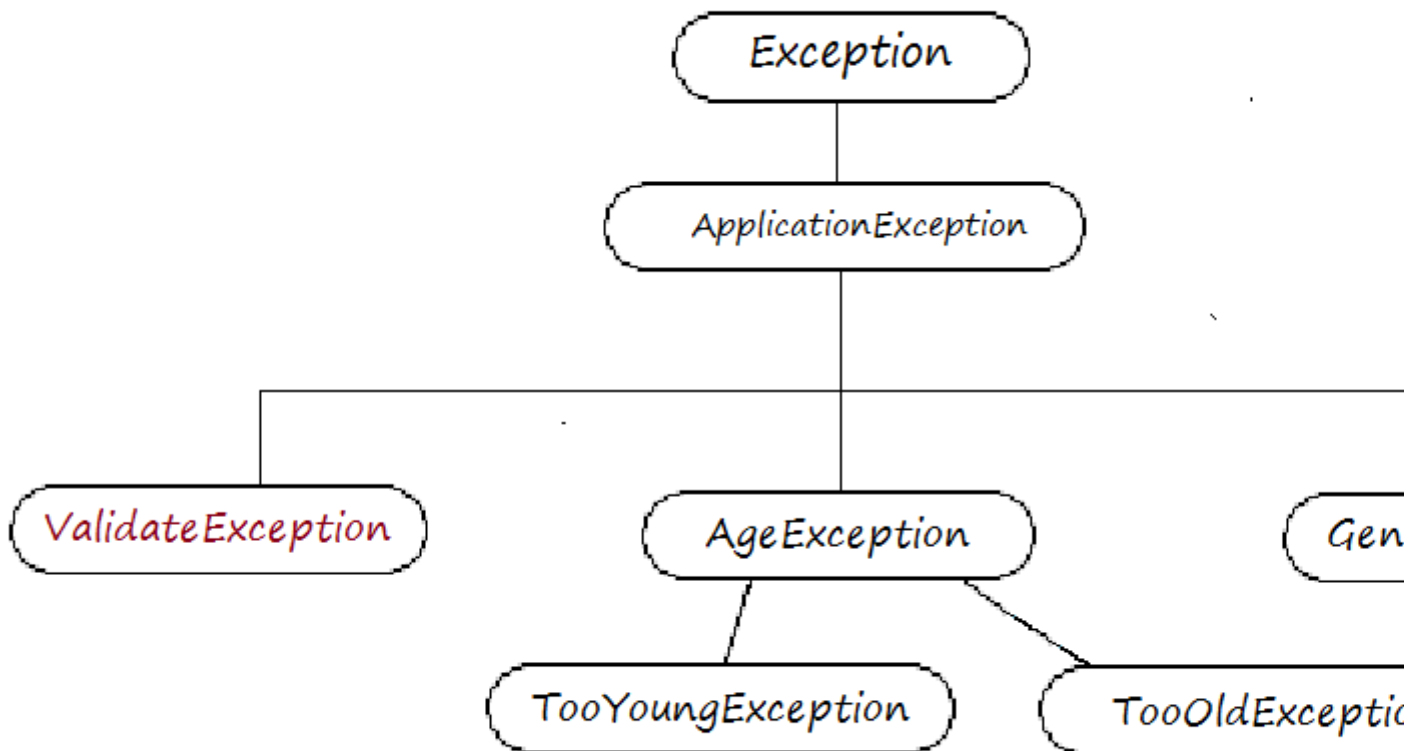
        public static int toInteger(String text)
        {
            try
            {
                Console.WriteLine("Begin parse text: " + text);
                int value = int.Parse(text);
                return value;
            }
            catch (FormatException e)
            {
                Console.WriteLine("Number format exception: " + e.Message);
            }
            finally
            {
                Console.WriteLine("End parse text: " + text);
            }
        }
    }
}
```

5- Gói một Exception trong một Exception khác

Chúng ta cần một vài class tham gia vào ví dụ này:

- **Person:** Mô phỏng một người tham gia tuyển dụng vào công ty với các thông tin
 - Tên, tuổi, giới tính.
- **GenderException:** Ngoại lệ giới tính.

- **ValidateException**: Ngoại lệ đánh giá thí sinh.
- **ValidateUtils**: Class có method tính đánh giá thí sinh đủ tiêu chuẩn không.
 - Tiêu chuẩn là những người độ tuổi 18-40
 - Và là Nam.



Person.cs

```

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace ExceptionTutorial
8  {
9      class Person
10     {
11
12         public static readonly string MALE = "male";
13         public static readonly string FEMALE = "female";
14
15         private string name;
16         private string gender;
17         private int age;
18
19         public Person(string name, string gender, int age)
20         {
21             this.name = name;
22             this.gender = gender;

```

```

23         this.age = age;
24     }
25
26     public string GetName()
27     {
28         return name;
29     }
30
31
32     public string GetGender()
33     {
34         return gender;
35     }
36
37     public int GetAge()
38     {
39         return age;
40     }
41
42 }
43
44 }

```

GenderException.cs

```

?
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace ExceptionTutorial
8  {
9      class GenderException : ApplicationException
10     {
11
12         public GenderException(String message)
13             : base(message)
14         {
15
16
17         }
18     }
19
20 }

```

Class ValidateException bao lấy một Exception khác.

ValidateException.cs

```

?
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;

```

```

4    using System.Text;
5    using System.Threading.Tasks;
6
7    namespace ExceptionTutorial
8    {
9        class ValidateException : ApplicationException
10       {
11
12           // Wrap an Exception
13           public ValidateException(Exception e) : base("Something invalid", e)
14           {
15
16           }
17       }
18
19 }

```

ValidateUtils.java

```

?
1    using System;
2    using System.Collections.Generic;
3    using System.Linq;
4    using System.Text;
5    using System.Threading.Tasks;
6
7    namespace ExceptionTutorial
8    {
9
10       class ValidateUtils
11       {
12
13           public static void CheckPerson(Person person)
14           {
15               try
16               {
17
18                   // Kiểm tra tuổi.
19                   // Hợp lệ là trong khoảng 18-40
20                   // Method này có thể ném ra TooOldException, TooYoungException.
21                   AgeUtils.checkAge(person.GetAge());
22
23               }
24               catch (Exception e)
25               {
26
27                   // Nếu không hợp lệ
28                   // Gói ngoại lệ này bằng ValidateException
29                   throw new ValidateException(e);
30
31               }
32

```

```

33         // Nếu người đó là Nữ, nghĩa là không hợp lệ.
34         if (person.GetGender() == Person.FEMALE)
35         {
36
37             GenderException e = new GenderException("Do not accept women");
38             throw new ValidateException(e);
39
40         }
41     }
42 }
43
44 }

```

WrapperExceptionDemo.cs

```

?
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace ExceptionTutorial
8  {
9      class WrapperExceptionDemo
10     {
11         public static void Main(string[] args)
12         {
13
14             // Một người tham gia tuyển dụng.
15             Person person = new Person("Marry", Person.FEMALE, 20);
16
17             try
18             {
19
20                 // Ngoại lệ có thể xảy ra tại đây.
21                 ValidateUtils.CheckPerson(person);
22
23             }
24             catch (ValidateException wrap)
25             {
26
27                 // Lấy ra nguyên nhân thực sự.
28                 // Mà có thể là TooYoungException, TooOldException, GenderException
29                 Exception cause = wrap.GetBaseException();
30
31                 if (cause != null)
32                 {
33                     Console.WriteLine("Message: " + wrap.Message);
34                     Console.WriteLine("Base Exception Message: " + cause.Message);
35                 }
36                 else

```

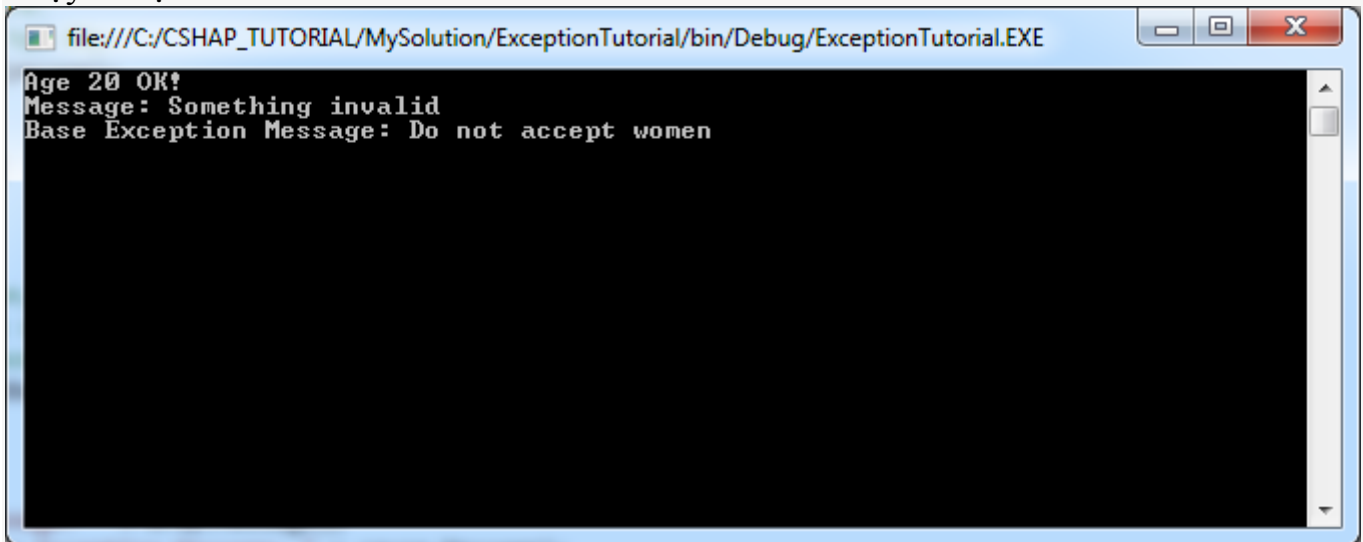


```

37         {
38             Console.WriteLine("Message: " + wrap.Message);
39         }
40     }
41 }
42
43 Console.Read();
44 }
45 }
46
47 }

```

Chạy ví dụ:



```

file:///C:/CSHAP_TUTORIAL/MySolution/ExceptionTutorial/bin/Debug/ExceptionTutorial.EXE
Age 20 OK!
Message: Something invalid
Base Exception Message: Do not accept women

```

6- Một số ngoại lệ thông dụng



Bây giờ bạn có thể xem một vài ví dụ với các ngoại lệ thông dụng.

6.1- NullReferenceException

Đây là một trong các ngoại lệ thông dụng nhất, và hay gây ra lỗi cho chương trình. Ngoại lệ được ném ra khi bạn gọi hàm hoặc truy cập vào các trường của một đối tượng chưa được khởi tạo (đối tượng null).

NullReferenceExceptionDemo.cs

```

?
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace ExceptionTutorial
8  {
9      class NullReferenceExceptionDemo
10     {
11
12         // Ví dụ đây là một method mà có thể trả về chuỗi null.
13         public static string GetString()
14         {
15             if (1 == 2)
16             {

```

```

17         return "1==2 !!";
18     }
19     return null;
20 }
21
22 public static void Main(string[] args)
23 {
24
25     // Đây là một đối tượng có tham chiếu khác null.
26     string text1 = "Hello exception";
27
28     // Lấy độ dài chuỗi.
29     int length = text1.Length;
30
31     Console.WriteLine("Length text1 = " + length);
32
33     // Đây là một đối tượng null.
34     String text2 = GetString(); // text2 = null.
35
36
37     // Lấy độ dài chuỗi.
38     // NullReferenceException sẽ xảy ra tại đây.
39     length = text2.Length; // ==> Runtime Error!
40
41     Console.WriteLine("Finish!");
42
43
44     Console.Read();
45 }
46 }
47
48 }

```

Trong thực tế giống việc xử lý các ngoại lệ khác, bạn có thể sử dụng **try-catch** để bắt ngoại lệ này mà xử lý. Tuy nhiên, đó là cách máy móc, thông thường chúng ta nên kiểm tra để đảm bảo rằng đối tượng là khác null trước khi sử dụng nó.

Bạn có thể sửa code trên giống dưới đây, để tránh **NullReferenceException**:

```

?
1 // Đây là một đối tượng có tham chiếu null.
2 String text2 = GetString();
3
4 // Kiểm tra để đảm bảo rằng text2 là khác null
5 // Điều này tránh NullReferenceException
6 // Thay vì máy móc sử dụng try-catch.
7 if (text2 != null)
8 {
9     length = text2.Length;
10 }

```

6.2- IndexOutOfRangeException

Đây là ngoại lệ nó được ném ra khi bạn cố truy cập vào phần tử có chỉ số không hợp lệ trên mảng. Chẳng hạn mảng có 10 phần tử, mà bạn lại truy cập vào phần tử có chỉ số 20.

IndexOutOfRangeExceptionDemo.cs

```
?
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace ExceptionTutorial
8  {
9      class IndexOutOfRangeExceptionDemo
10     {
11         public static void Main(string[] args)
12         {
13
14             String[] strs = new String[] { "One", "Two", "Three" };
15
16             // Truy cập vào phần tử có chỉ số 0.
17             String str1 = strs[0];
18
19             Console.WriteLine("String at 0 = " + str1);
20
21
22             // Truy cập vào phần tử có chỉ số 5
23             // IndexOutOfRangeException xảy ra tại đây.
24             String str2 = strs[5];
25
26             Console.WriteLine("String at 5 = " + str2);
27
28             Console.Read();
29
30         }
31     }
32 }
```

Để tránh **IndexOutOfRangeException** bạn nên kiểm tra mảng thay vì sử dụng *try-catch*.

```
?
1  if (strs.length > 5)
2  {
3      String str2 = strs[5];
4      Console.WriteLine("String at 5 = " + str2);
5  }
6  else
7  {
8      Console.WriteLine("No elements with index 5");
9  }
10
```

Hướng dẫn sử dụng Date Time trong C#

1- Các class liên quan Date, Time trong C#

- 2- Các thuộc tính DateTime
- 3- Thêm và bớt thời gian
- 4- Đo khoảng thời gian
- 5- So sánh hai đối tượng DateTime
- 6- Định dạng tiêu chuẩn DateTime
- 7- Tùy biến định dạng DateTime

Cảnh báo không nên sử dụng máy tính
LENOVO vì phần mềm gián điệp

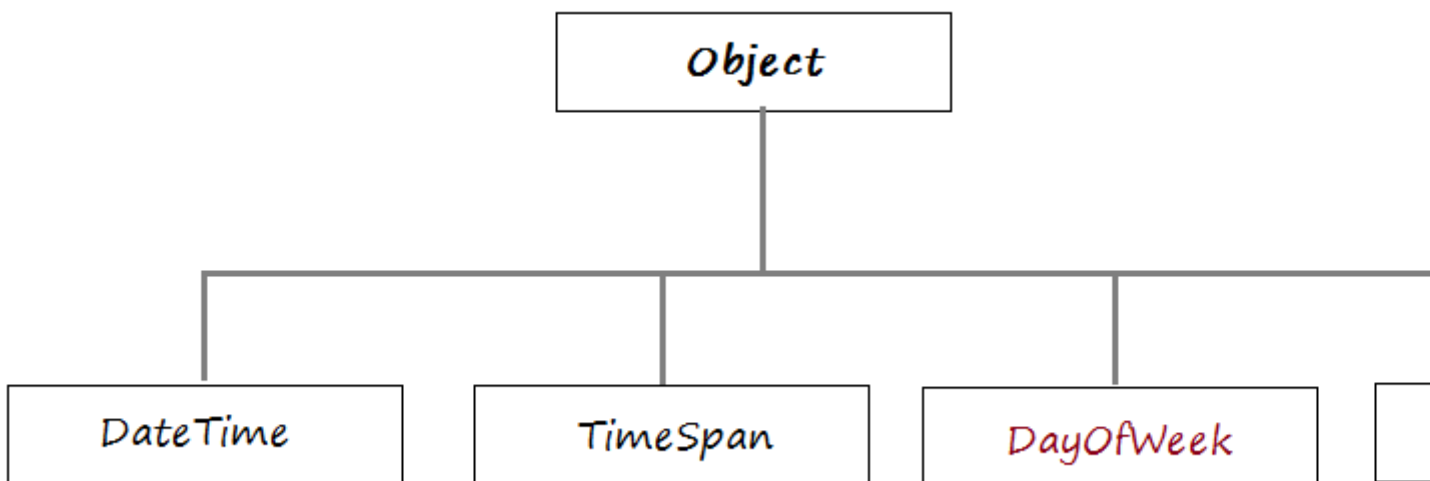


TÀI LIỆU ĐANG ĐƯỢC CẬP NHẬP 80%

1- 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 cấu tử để bạn khởi tạo một đối tượng **DateTime**.

DateTime Constructors

?

- 1 public DateTime(int year, int month, int day)
- 2
- 3 public DateTime(int year, int month, int day, Calendar calendar)
- 4
- 5 public DateTime(int year, int month, int day, int hour, int minute, int second)

6

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

8

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

10

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

12

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

14

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

16

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

18

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

20

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

21

- TODO - EXAMPLE

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

?

```
1 // Đối tượng mô tả thời điểm hiện tại.
```

2

```
3 DateTime now = DateTime.Now;
```

4

```
5 Console.WriteLine("Now is "+ now);
```

2- Các thuộc tính DateTime



DateTimePropertiesExample.cs

?

```
1 using System;
```

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

```
3 using System.Linq;
```

```
4 using System.Text;
```

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

6

```
7 namespace DateTimeTutorial
```

8

```
{
```

```
9     class DateTimePropertiesExample
```

10

```
{
```

11

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

```

13     {
14
15         // Tạo một đối tượng DateTime (năm, tháng, ngày, giờ, phút, giây).
16         DateTime aDateTime = new DateTime(2005, 11, 20, 12, 1, 10);
17
18         // In ra các thông tin:
19
20         Console.WriteLine("Day:{0}", aDateTime.Day);
21         Console.WriteLine("Month:{0}", aDateTime.Month);
22         Console.WriteLine("Year:{0}", aDateTime.Year);
23         Console.WriteLine("Hour:{0}", aDateTime.Hour);
24         Console.WriteLine("Minute:{0}", aDateTime.Minute);
25         Console.WriteLine("Second:{0}", aDateTime.Second);
26         Console.WriteLine("Millisecond:{0}", aDateTime.Millisecond);
27
28         // Enum {Monday, Tuesday,... Sunday}
29         DayOfWeek dayOfWeek = aDateTime.DayOfWeek;
30
31
32         Console.WriteLine("Day of Week:{0}", dayOfWeek );
33
34
35         Console.WriteLine("Day of Year: {0}", aDateTime.DayOfYear);
36
37         // Một đối tượng chỉ mô tả thời gian (giờ phút giây,...)
38         TimeSpan timeOfDay = aDateTime.TimeOfDay;
39
40         Console.WriteLine("Time of Day:{0}", timeOfDay);
41
42         // Quy đổi ra Ticks (1 giây = 10.000.000 Ticks)
43         Console.WriteLine("Tick:{0}", aDateTime.Ticks);
44
45         // {Local, Itc, Unspecified}
46         DateTimeKind kind = aDateTime.Kind;
47
48         Console.WriteLine("Kind:{0}", kind);
49
50         Console.Read();
51     }
52 }
53
54 }

```

Chạy ví dụ:

```
file:///C:/CSHAP_TUTORIAL/MySolution/DateTimeTutorial/bin/Debug/DateTimeTutorial.EXE
Day:20
Month:11
Year:2005
Hour:12
Minute:1
Second:10
Millisecond:0
Day of Week:Sunday
Day of Year: 324
Time of Day:12:01:10
Tick:632680848700000000
Kind:Unspecified
```

3- 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**.

Ví dụ:

AddSubtractExample.cs

```
?
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6
7 namespace DateTimeTutorial
8 {
9     class AddSubtractExample
10    {
11
12        public static void Main(string[] args)
```

```

13     {
14         // Thời điểm hiện tại.
15         DateTime aDateTime = DateTime.Now;
16
17         Console.WriteLine("Now is " + aDateTime);
18
19         // Một khoảng thời gian.
20         // 1 giờ + 1 phút
21         TimeSpan aInterval = new System.TimeSpan(0, 1, 1, 0);
22
23         // Thêm khoảng thời gian.
24         DateTime newTime = aDateTime.Add(aInterval);
25
26
27         Console.WriteLine("After add 1 hour, 1 minute: " + newTime);
28
29
30         // Trừ khoảng thời gian.
31         newTime = aDateTime.Subtract(aInterval);
32
33         Console.WriteLine("After subtract 1 hour, 1 minute: " + newTime);
34
35         Console.Read();
36     }
37 }
38
39 }

```

Chạy ví dụ:

```

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
- ...

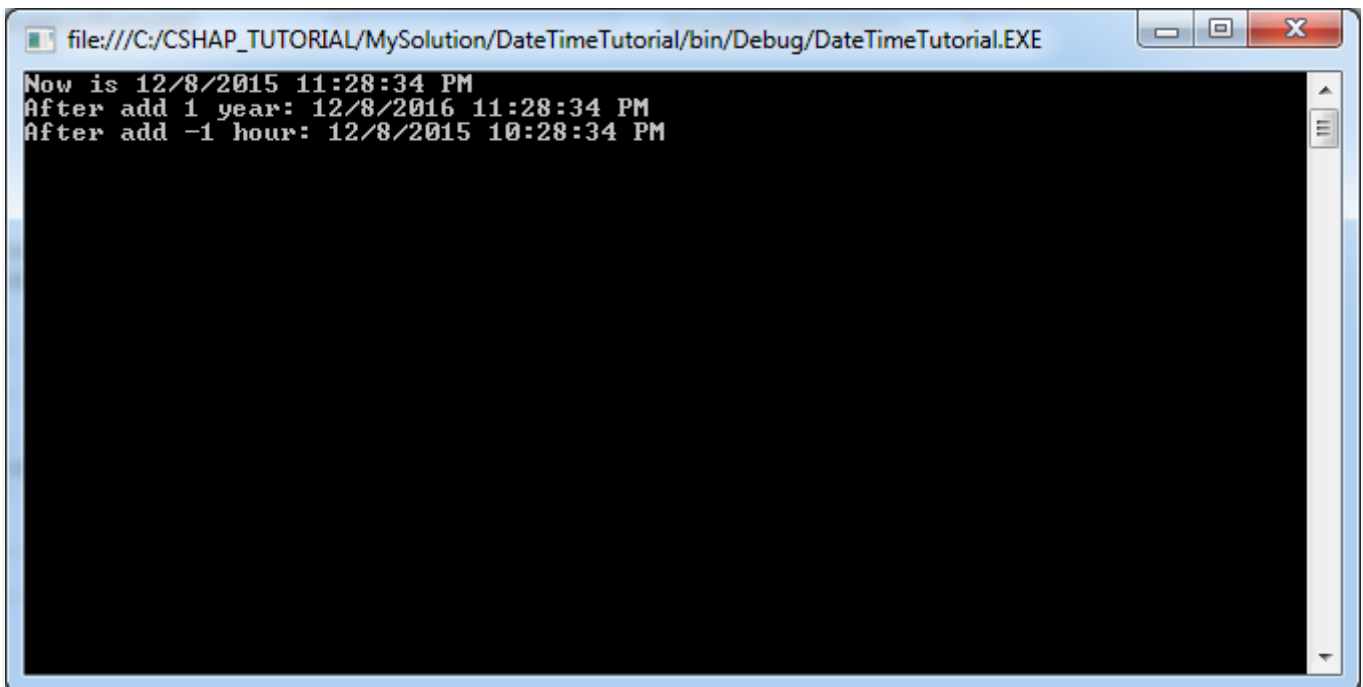
Ví dụ:

AddSubtractExample2.cs

?

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace DateTimeTutorial
8  {
9      class AddSubtractExample2
10     {
11         public static void Main(string[] args)
12         {
13             // Thời điểm hiện tại.
14             DateTime aDateTime = DateTime.Now;
15
16             Console.WriteLine("Now is " + aDateTime);
17
18             // Thêm 1 năm
19             DateTime newTime = aDateTime.AddYears(1);
20
21
22             Console.WriteLine("After add 1 year: " + newTime);
23
24
25             // Trừ 1 giờ
26             newTime = aDateTime.AddHours(-1);
27
28             Console.WriteLine("After add -1 hour: " + newTime);
29
30             Console.Read();
31         }
32     }
33
34 }
```

Chạy ví dụ:

A screenshot of a Windows console window titled "file:///C:/CSHAP_TUTORIAL/MySolution/DateTimeTutorial/bin/Debug/DateTimeTutorial.EXE". The console output shows three lines of text: "Now is 12/8/2015 11:28:34 PM", "After add 1 year: 12/8/2016 11:28:34 PM", and "After add -1 hour: 12/8/2015 10:28:34 PM". The console background is black with white text. The window has standard Windows window controls (minimize, maximize, close) in the top right corner.

```
file:///C:/CSHAP_TUTORIAL/MySolution/DateTimeTutorial/bin/Debug/DateTimeTutorial.EXE
Now is 12/8/2015 11:28:34 PM
After add 1 year: 12/8/2016 11:28:34 PM
After add -1 hour: 12/8/2015 10:28:34 PM
```

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:

FirstLastDayDemo.cs

```
?
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace DateTimeTutorial
8  {
9      class FirstLastDayDemo
10     {
11         public static void Main(string[] args)
12         {
13
14             Console.WriteLine("Today is " + DateTime.Today);
15
16             DateTime yesterday = GetYesterday();
17
18             Console.WriteLine("Yesterday is " + yesterday);
19
20
21             // Ngày đầu tiên của tháng 2 năm 2015
22             DateTime aDateTime = GetFistDayInMonth(2015, 2);
23
24             Console.WriteLine("First day of 2-2015: " + aDateTime);
25
26             // Ngày cuối cùng của tháng 2 năm 2015
27             aDateTime = GetLastDayInMonth(2015, 2);
28
29             Console.WriteLine("Last day of 2-2015: " + aDateTime);
```

```
30
31 // Ngày đầu tiên của năm 2015
32 aDateTime = GetFirstDayInYear(2015);
33
34 Console.WriteLine("First day year 2015: " + aDateTime);
35
36 // Ngày cuối cùng của năm 2015
37 aDateTime = GetLastDayInYear(2015);
38
39 Console.WriteLine("First day year 2015: " + aDateTime);
40
41 Console.Read();
42 }
43
44 // Trả về ngày hôm qua.
45 public static DateTime GetYesterday()
46 {
47     // Ngày hôm nay.
48     DateTime today = DateTime.Today;
49     // Trả về ngày trước 1 ngày.
50     return today.AddDays(-1);
51 }
52
53 // Trả về ngày đầu tiên của năm
54 public static DateTime GetFirstDayInYear(int year)
55 {
56     DateTime aDateTime = new DateTime(year, 1, 1);
57     return aDateTime;
58 }
59
60 // Trả về ngày cuối cùng của năm.
61 public static DateTime GetLastDayInYear(int year)
62 {
63     DateTime aDateTime = new DateTime(year + 1, 1, 1);
64
65     // Trừ đi một ngày.
66     DateTime retDateTime = aDateTime.AddDays(-1);
67
68     return retDateTime;
69 }
70
71 // Trả về ngày đầu tiên của tháng
72 public static DateTime GetFistDayInMonth(int year, int month)
73 {
74     DateTime aDateTime = new DateTime(year, month, 1);
75
76     return aDateTime;
77 }
78
79 // Trả về ngày cuối cùng của tháng.
```

```

80     public static DateTime GetLastDayInMonth(int year, int month)
81     {
82         DateTime aDateTime = new DateTime(year, month, 1);
83
84         // Cộng thêm 1 tháng và trừ đi một ngày.
85         DateTime retDateTime = aDateTime.AddMonths(1).AddDays(-1);
86
87         return retDateTime;
88     }
89
90 }
91
92 }

```

Chạy ví dụ:

```

file:///C:/CSHAP_TUTORIAL/MySolution/DateTimeTutorial/bin/Debug/DateTimeTutorial.EXE
Today is 12/9/2015 12:00:00 AM
Yesterday is 12/8/2015 12:00:00 AM
First day of 2-2015: 2/1/2015 12:00:00 AM
Last day of 2-2015: 2/28/2015 12:00:00 AM
First day year 2015: 1/1/2015 12:00:00 AM
First day year 2015: 12/31/2015 12:00:00 AM
-

```

4- 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.

IntervalDemo.cs

```

?
1     using System;
2     using System.Collections.Generic;
3     using System.Linq;
4     using System.Text;
5     using System.Threading.Tasks;
6
7     namespace DateTimeTutorial
8     {
9         class IntervalDemo
10        {
11            public static void Main(string[] args)
12            {
13                // Thời điểm hiện tại.
14                DateTime aDateTime = DateTime.Now;

```

```

15
16 // Thời điểm năm 2000
17 DateTime y2K = new DateTime(2000,1,1);
18
19 // Khoảng thời gian từ năm 2000 tới nay.
20 TimeSpan interval = aDateTime.Subtract(y2K);
21
22
23 Console.WriteLine("Interval from Y2K to Now: " + interval);
24
25 Console.WriteLine("Days: " + interval.Days);
26 Console.WriteLine("Hours: " + interval.Hours);
27 Console.WriteLine("Minutes: " + interval.Minutes);
28 Console.WriteLine("Seconds: " + interval.Seconds);
29
30
31 Console.Read();
32 }
33 }
34
35 }

```

Chạy ví dụ:

```

file:///C:/CSHAP_TUTORIAL/MySolution/DateTimeTutorial/bin/Debug/DateTimeTutorial.EXE
Interval from Y2K to Now: 5820.23:51:08.1194036
Days: 5820
Hours: 23
Minutes: 51
Seconds: 8

```

5- 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:

?

```

1 // Nếu giá trị < 0 nghĩa là firstDateTime sớm hơn (đứng trước)
2 // Nếu giá trị > 0 nghĩa là secondDateTime sớm hơn (đứng trước).
3 // Nếu giá trị = 0 nghĩa là 2 đối tượng này giống nhau về mặt thời gian.
4
5 public static int Compare(DateTime firstDateTime, DateTime secondDateTime);

```

(dd-MM-yyyy)

2-9-2000

20-1-2011

CompareDateTimeExample.cs

```
?
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace DateTimeTutorial
8  {
9      class CompareDateTimeExample
10     {
11         public static void Main(string[] args)
12         {
13             // Thời điểm hiện tại.
14             DateTime firstDateTime = new DateTime(2000, 9, 2);
15
16             DateTime secondDateTime = new DateTime(2011, 1, 20);
17
18             int compare = DateTime.Compare(firstDateTime, secondDateTime);
19
20             Console.WriteLine("First DateTime: " + firstDateTime);
21             Console.WriteLine("Second DateTime: " + secondDateTime);
22
23             Console.WriteLine("Compare value: " + compare);// -1
24
25             if (compare < 0)
26             {
27                 // firstDateTime sớm hơn secondDateTime
28                 Console.WriteLine("firstDateTime is earlier than secondDateTime");
29             }
30             else
31             {
32                 // firstDateTime muộn hơn secondDateTime
33                 Console.WriteLine("firstDateTime is later than secondDateTime");
34             }
35
36             Console.Read();
37         }
38     }
39
40 }
```

Chạy ví dụ:

```
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
```

6- 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  
...
```

AllStandardFormatsDemo.cs

```
?  
1 using System;  
2 using System.Collections.Generic;
```

```

3    using System.Linq;
4    using System.Text;
5    using System.Threading.Tasks;
6
7    namespace DateTimeTutorial
8    {
9        class AllStandardFormatsDemo
10       {
11           public static void Main(string[] args)
12           {
13
14               DateTime aDateTime = new DateTime(2015, 12, 20, 11, 30, 50);
15
16               // Một mảng các string kết quả định dạng được hỗ trợ.
17               string[] formattedStrings = aDateTime.GetDateTimeFormats();
18
19               foreach (string format in formattedStrings)
20               {
21                   Console.WriteLine(format);
22               }
23
24               Console.Read();
25           }
26       }
27   }
28
29   }

```

Chạy ví dụ:

```

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
ToString(String,	Chuyển đổi các giá trị của đối tượng DateTime hiện thành chuỗi

IFormatProvider)	đại diện tương đương của nó bằng cách sử dụng định dạng quy định (Tham số String) và các thông tin định dạng văn hóa cụ thể (Tham số IFormatProvider).
ToString(IFormatProvider)	Chuyển đổi giá trị của đối tượng DateTime hiện tại thành một string tương ứng với thông tin định dạng văn hóa (culture) cho bởi tham số.
ToString(String)	Chuyển đổi các giá trị của đối tượng DateTime 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 trong tham số.

SimpleDateTimeFormat.cs

?

```

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6  using System.Globalization;
7
8  namespace DateTimeTutorial
9  {
10     class SimpleDateTimeFormat
11     {
12         public static void Main(string[] args)
13         {
14
15             DateTime aDateTime = new DateTime(2015, 12, 20, 11, 30, 50);
16
17             Console.WriteLine("DateTime: " + aDateTime);
18
19             String d_formatString = aDateTime.ToString("d");
20

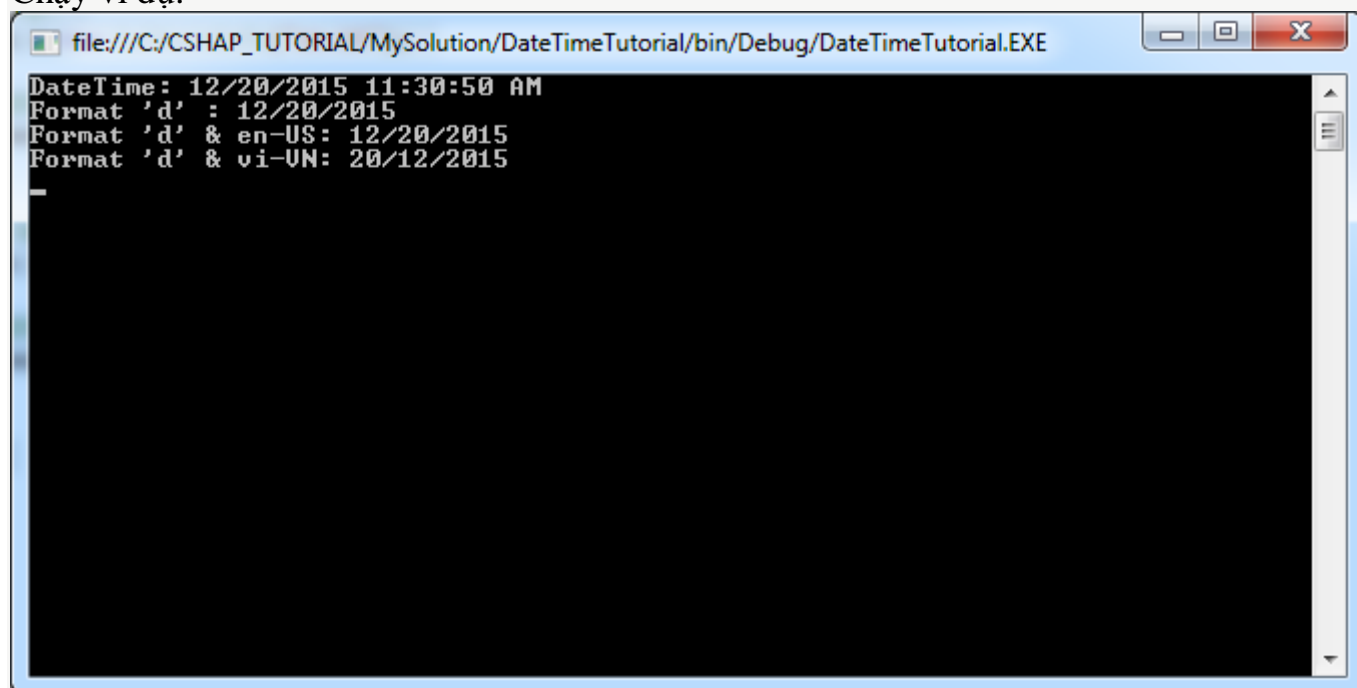
```

```

21 Console.WriteLine("Format 'd' : " + d_formatString);
22
23
24 // Theo văn hóa Mỹ.
25 CultureInfo enUs = new CultureInfo("en-US");
26
27 // ==> 12/20/2015 (MM/dd/yyyy)
28 Console.WriteLine("Format 'd' & en-US: " + aDateTime.ToString("d", enUs));
29
30
31 // Theo văn hóa Việt Nam.
32 CultureInfo viVn = new CultureInfo("vi-VN");
33
34 // ==> 12/20/2015 (dd/MM/yyyy)
35 Console.WriteLine("Format 'd' & vi-VN: " + aDateTime.ToString("d", viVn));
36
37
38 Console.Read();
39 }
40 }
41
42 }

```

Chạy ví dụ:



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

SimpleDateTimeFormatAll.cs

```
?
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace DateTimeTutorial
8  {
9      class SimpleDateTimeFormatAll
10     {
11         public static void Main(string[] args)
12         {
13             char[] formats = {'d', 'D', 'f', 'F', 'g', 'G', 'M', 'm', 'O', 'o', 'R', 'r', 's', 't', 'T', 'u', 'U', 'Y', 'y'};
14
15
16             DateTime aDateTime = new DateTime(2015, 12, 20, 11, 30, 50);
17
18             foreach (char ch in formats)
19             {
20                 Console.WriteLine("\n===== " + ch + " =====\n");
21                 // Một mảng các string kết quả định dạng được hỗ trợ.
22                 string[] formattedStrings = aDateTime.GetDateTimeFormats(ch);
23
24                 foreach (string format in formattedStrings)
25                 {
26                     Console.WriteLine(format);
27                 }
28             }
29
30
31
```

```

32     Console.ReadLine();
33     }
34 }
35
36 }

```

Chạy ví dụ:

```

?
1     =====d=====
2
3     12/20/2015
4     12/20/15
5     12/20/15
6     12/20/2015
7     15/12/20
8     2015-12-20
9     20-Dec-15
10
11    =====D=====
12
13    Sunday, December 20, 2015
14    December 20, 2015
15    Sunday, 20 December, 2015
16    20 December, 2015
17
18    =====f=====
19
20    Sunday, December 20, 2015 11:30 AM
21    Sunday, December 20, 2015 11:30 AM
22    Sunday, December 20, 2015 11:30
23    Sunday, December 20, 2015 11:30
24    December 20, 2015 11:30 AM
25    December 20, 2015 11:30 AM
26    December 20, 2015 11:30
27    December 20, 2015 11:30
28    Sunday, 20 December, 2015 11:30 AM
29    Sunday, 20 December, 2015 11:30 AM
30    Sunday, 20 December, 2015 11:30
31    Sunday, 20 December, 2015 11:30
32    20 December, 2015 11:30 AM
33    20 December, 2015 11:30 AM
34    20 December, 2015 11:30
35    20 December, 2015 11:30
36
37    =====F=====
38
39    Sunday, December 20, 2015 11:30:50 AM
40    Sunday, December 20, 2015 11:30:50 AM
41    Sunday, December 20, 2015 11:30:50
42    Sunday, December 20, 2015 11:30:50
43    December 20, 2015 11:30:50 AM

```

44 December 20, 2015 11:30:50 AM
45 December 20, 2015 11:30:50
46 December 20, 2015 11:30:50
47 Sunday, 20 December, 2015 11:30:50 AM
48 Sunday, 20 December, 2015 11:30:50 AM
49 Sunday, 20 December, 2015 11:30:50
50 Sunday, 20 December, 2015 11:30:50
51 20 December, 2015 11:30:50 AM
52 20 December, 2015 11:30:50 AM
53 20 December, 2015 11:30:50
54 20 December, 2015 11:30:50

55
56 =====g=====

57
58 12/20/2015 11:30 AM
59 12/20/2015 11:30 AM
60 12/20/2015 11:30
61 12/20/2015 11:30
62 12/20/15 11:30 AM
63 12/20/15 11:30 AM
64 12/20/15 11:30
65 12/20/15 11:30
66 12/20/15 11:30 AM
67 12/20/15 11:30 AM
68 12/20/15 11:30
69 12/20/15 11:30
70 12/20/2015 11:30 AM
71 12/20/2015 11:30 AM
72 12/20/2015 11:30
73 12/20/2015 11:30
74 15/12/20 11:30 AM
75 15/12/20 11:30 AM
76 15/12/20 11:30
77 15/12/20 11:30
78 2015-12-20 11:30 AM
79 2015-12-20 11:30 AM
80 2015-12-20 11:30
81 2015-12-20 11:30
82 20-Dec-15 11:30 AM
83 20-Dec-15 11:30 AM
84 20-Dec-15 11:30
85 20-Dec-15 11:30

86
87 =====G=====

88
89 12/20/2015 11:30:50 AM
90 12/20/2015 11:30:50 AM
91 12/20/2015 11:30:50
92 12/20/2015 11:30:50
93 12/20/15 11:30:50 AM

94 12/20/15 11:30:50 AM
95 12/20/15 11:30:50
96 12/20/15 11:30:50
97 12/20/15 11:30:50 AM
98 12/20/15 11:30:50 AM
99 12/20/15 11:30:50
100 12/20/15 11:30:50
101 12/20/2015 11:30:50 AM
102 12/20/2015 11:30:50 AM
103 12/20/2015 11:30:50
104 12/20/2015 11:30:50
105 15/12/20 11:30:50 AM
106 15/12/20 11:30:50 AM
107 15/12/20 11:30:50
108 15/12/20 11:30:50
109 2015-12-20 11:30:50 AM
110 2015-12-20 11:30:50 AM
111 2015-12-20 11:30:50
112 2015-12-20 11:30:50
113 20-Dec-15 11:30:50 AM
114 20-Dec-15 11:30:50 AM
115 20-Dec-15 11:30:50
116 20-Dec-15 11:30:50
117
118 =====M =====
119
120 December 20
121
122 =====m =====
123
124 December 20
125
126 =====O =====
127
128 2015-12-20T11:30:50.0000000
129
130 =====o =====
131
132 2015-12-20T11:30:50.0000000
133
134 =====R =====
135
136 Sun, 20 Dec 2015 11:30:50 GMT
137
138 =====r =====
139
140 Sun, 20 Dec 2015 11:30:50 GMT
141
142 =====s =====
143

```

144 2015-12-20T11:30:50
145
146 =====t =====
147
148 11:30 AM
149 11:30 AM
150 11:30
151 11:30
152
153 =====T =====
154
155 11:30:50 AM
156 11:30:50 AM
157 11:30:50
158 11:30:50
159
160 =====u =====
161
162 2015-12-20 11:30:50Z
163
164 =====U =====
165
166 Sunday, December 20, 2015 4:30:50 AM
167 Sunday, December 20, 2015 04:30:50 AM
168 Sunday, December 20, 2015 4:30:50
169 Sunday, December 20, 2015 04:30:50
170 December 20, 2015 4:30:50 AM
171 December 20, 2015 04:30:50 AM
172 December 20, 2015 4:30:50
173 December 20, 2015 04:30:50
174 Sunday, 20 December, 2015 4:30:50 AM
175 Sunday, 20 December, 2015 04:30:50 AM
176 Sunday, 20 December, 2015 4:30:50
177 Sunday, 20 December, 2015 04:30:50
178 20 December, 2015 4:30:50 AM
179 20 December, 2015 04:30:50 AM
180 20 December, 2015 4:30:50
181 20 December, 2015 04:30:50
182
183 =====Y =====
184
185 December, 2015
186
187 =====y =====
188
189 December, 2015

```