



LẬP TRÌNH WINDOWS PHONE

Module 1

☞ *Click vào phụ lục để chuyển tới bài cần đọc*

Phụ lục

Bài 1 Nền tảng ngôn ngữ C#	2
Bài 2 Xử lý chuỗi trong C#	15
Bài 3 Mảng và Collection trong C#	26
Bài 4 Xây dựng lớp – Đối tượng	45
Bài 5 Kết thừa – Đa hình.....	63
Bài 6 Lập trình Windows Phone	80
Bài 7 Xử lý biệt lệ Exception	86
Bài 8 Cơ bản về LINQ	94
Bài 9 Regular Expression (Regex)	102



Trường ĐH Khoa Khoa Hoc Tự Nhiên Tp. Hồ Chí Minh
TRUNG TÂM TIN HỌC

Screen Capture

Lập trình Windows Phone

Bài 1. *Nền tảng ngôn ngữ C#*

Ngành Mạng & Thiết bị di động

2014



SOFT



Nội dung



1. **Làm quen với cú pháp trong C#.**
2. **Các kiểu dữ liệu.**
3. **Bí ẩn và hằng.**
4. **Toán tử**
5. **Các cấu trúc điều khiển.**



Làm quen với cú pháp trong C#



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

namespace ConsoleApplication1
{
    class Program
    {
        /// <summary>
        /// Đây là đoạn lệnh comment
        /// </summary>
        /// <param name="args"></param>
        static void Main(string[] args)
        {
            /*Đây là đoạn lệnh comment*/
            //Đây là đoạn lệnh comment

            #region Nhóm khối lệnh
            #endregion
        }
    }
}
```



Làm quen với cú pháp trong C#



- ❑ **Mỗi câu lệnh kết thúc bằng dấu “;”.**
- ❑ **Các khối lệnh nằm trong dấu “{ }”.**
- ❑ **Khối lệnh con nằm thực vào một khoảng so với khối lệnh cha.**
- ❑ **Thống nhất cách comment trong toàn chương trình.**



Các kiểu dữ liệu



- ❑ **Kiểu dữ liệu là thành phần cốt yếu trong chương trình.**
- ❑ **Mỗi đối tượng được tạo ra phải thuộc 1 kiểu dữ liệu nhất định.**
- ❑ **Có 2 tập hợp kiểu dữ liệu chính: kiểu xây dựng sẵn và kiểu do người dùng định nghĩa**



Kiểu dữ liệu xây dựng sẵn



Kiểu C#	Số byte	Kiểu .NET	Mô tả
byte	1	Byte	Số nguyên dương không dấu từ 0-255
char	2	Char	Ký tự Unicode
bool	1	Boolean	Giá trị logic true/ false
sbyte	1	Sbyte	Số nguyên có dấu (từ -128 đến 127)
short	2	Int16	Số nguyên có dấu giá trị từ -32768 đến 32767.
ushort	2	Uint16	Số nguyên không dấu 0 – 65.535
int	4	Int32	Số nguyên có dấu -2.147.483.647 và 2.147.483.647
uint	4	Uint32	Số nguyên không dấu 0 – 4.294.967.295
float	4	Single	Kiểu dấu chấm động, giá trị xấp xỉ từ 3,4E-38 đến 3,4E+38, với 7 chữ số có nghĩa..
double	8	Double	Kiểu dấu chấm động có độ chính xác gấp đôi, giá trị xấp xỉ từ 1,7E-308 đến 1,7E+308, với 15,16 chữ số có nghĩa.
decimal	8	Decimal	Có độ chính xác đến 28 con số và giá trị thập phân, được dùng trong tính toán tài chính, kiểu này đòi hỏi phải có hậu tố "m" hay "M" theo sau giá trị.



Chuyển đổi giữa các kiểu dữ liệu



- **Những đối tượng của một kiểu dữ liệu này có thể được chuyển sang những đối tượng của một kiểu dữ liệu khác.**

- **Có 2 dạng chuyển đổi: chuyển đổi ngầm định và chuyển đổi tường minh.**



Chuyển đổi ngầm định



- ❑ Được thực hiện một cách tự động và đảm bảo không mất thông tin.

- VD: short x = 10;

```
int y = x;
```



Chuyển đổi tường minh



- ❑ Dùng để chuyển đổi từ kiểu dữ liệu lớn sang kiểu dữ liệu nhỏ.

- ❑ Không đảm bảo việc bảo toàn dữ liệu.

- VD: short x;

```
int y = 500;
```

```
x = y;
```



Biến và hằng



☐ Biến

- Một biến là một vùng lưu trữ với một kiểu dữ liệu.
- Biến phải được khởi tạo trước khi sử dụng.
 - VD: int x = 100;
int y = x;



Biến và hằng



☐ Hằng

- Hằng cũng là một biến nhưng giá trị không thể thay đổi.
- Có 3 loại hằng
 - Giá trị hằng. VD: x = 100;
 - Biểu tượng hằng. VD: const double Pi = 3.14
 - Kiểu liệt kê. VD: enum NheitDo{

Nong=30,Lanh=20

}



Toán tử trong C#



□ Được kí hiệu bằng một biểu tượng để thực hiện 1 hành động.

□ Các toán tử trong C# bao gồm:

- Toán tử gán
- Toán tử toán học
- Toán tử tăng và giảm
- Toán tử quan hệ
- Toán tử logic



Toán tử trong C# (tt)



□ Toán tử gán “=”

- Là một toán tử 2 ngôi.
- Làm cho toán hạng bên trái có giá trị bằng toán hạng bên phải.
- VD: $x=y$;//gán giá trị của y cho x



Toán tử trong C# (tt)



❑ Toán tử toán học “+, -, *, /”

- Là những phép toán học cơ bản.
- Phép chia được phân thành chia nguyên và không nguyên.
- Cung cấp thêm phép chia lấy dư “%”.
 - VD: $8 \% 3 = 2$; // 8 chia 3 dư 2



Toán tử trong C# (tt)



❑ Toán tử tăng và giảm

- Dùng để tăng hoặc giảm giá trị của một biến.
 - VD: $a = a + 3$; // Tăng a thêm 3 đơn vị



Toán tử trong C# (tt)



□ Toán tử tăng và giảm

● Phép toán tự gán

Toán tử	Ý nghĩa
<code>+=</code>	Cộng thêm giá trị toán hạng bên phải vào giá trị toán hạng bên trái
<code>-=</code>	Toán hạng bên trái được trừ bớt đi một lượng bằng giá trị của toán hạng bên phải
<code>*=</code>	Toán hạng bên trái được nhân với một lượng bằng giá trị của toán hạng bên phải.
<code>/=</code>	Toán hạng bên trái được chia với một lượng bằng giá trị của toán hạng bên phải.
<code>%=</code>	Toán hạng bên trái được chia lấy dư với một lượng bằng giá trị của toán hạng bên phải.



Toán tử trong C# (tt)



□ Toán tử tăng và giảm

● Toán tử tăng 1 và toán tử giảm 1

- VD: `var++;` //tăng var thêm 1 đơn vị
- `var--;` //giảm var đi 1 đơn vị

● Toán tử tăng giảm hậu tố

- VD: `var1 = var2++;`//gán giá trị var2 cho var 1 sau đó tăng var2 lên 1 đơn vị.



Toán tử trong C# (tt)



□ Toán tử tăng và giảm

- Toán tử tăng giảm tần số

- VD: var1 = ++var2; // tăng var2 lên 1 đơn vị
sau đó gán giá trị đã được tăng cho var1



Toán tử trong C# (tt)



□ Toán tử quan hệ

Tên toán tử	Kí hiệu
So sánh bằng	$==$
Không bằng	\neq
Lớn hơn	$>$
Lớn hơn hay bằng	\geq
Nhỏ hơn	$<$
Nhỏ hơn hay bằng	\leq



Toán tử trong C# (tt)



☐ Toán tử logic

Tên toán tử	Ký hiệu
and	&&
or	
not	!

X	Y	X&&Y	X Y	!X
0	1	0	1	1
0	0	0	0	1
1	1	1	1	0
1	0	0	1	0



Toán tử trong C#



☐ Toán tử 3 ngôi

<Biểu thức điều kiện>?<Biểu thức 1>:<Biểu thức 2>

Nếu điều kiện đúng thì làm biểu thức 1, nếu sai làm biểu thức 2

- VD: $a = 4 > 5 ? 4 : 5$; //nếu $4 > 5$ thì $a = 4$, ngược lại $a = 5$





Các cấu trúc điều khiển trong C#

❑ Conditionals

❑ Loop



Conditionals

❑ Cú pháp

```
if(condition)
{
    // true case
}
else if(condition)
{
    // true case
}
else
{
    // false case
}
```

```
switch(condition)
{
    case <condition>:
        //code
        break;
    default:
        //code
        break;
}
```



Loop



❑ Cú pháp

```
while(condition)
{
    // phần thân
}
```

```
do
{
    //phần thân
}
while(condition)
```

```
for(khởi tạo biến lặp;điều kiện theo biến lặp;thay đổi biến lặp)
```

```
{
    //phần thân for
}
```



Thảo luận





Trường ĐH Khoa Học Tự Nhiên Tp. Hồ Chí Minh
TRUNG TÂM TIN HỌC

Go Screen Capture

Lập trình Windows Phone

Bài 2: Xử lý chuỗi trong C#

Ngành Mạng & Thiết bị di động

2014



SOFT



Nội dung



1. Giới thiệu lớp string trong C#

2. Các thao tác xử lý chuỗi

3. Làm việc với StringBuilder



1. Giới thiệu lớp string



- ❑ Lớp string cung cấp các phương thức giúp thao tác nhanh-gọn-lẹ với chuỗi
- ❑ Chỉ mục trong string được tính từ 0
- ❑ Xử dụng thuộc tính Length để biết chiều dài chuỗi



2. Các thao tác xử lý chuỗi



❑ So sánh chuỗi

❑ Sao chép chuỗi

❑ Nối chuỗi

❑ Thêm, xóa và chèn chuỗi



2. Các thao tác xử lý chuỗi



❑ Thay thế chuỗi

❑ Cắt chuỗi

❑ UperCase và LowerCase chuỗi



So sánh chuỗi



□ Phương thức static Compare(string1, string2). Cú pháp:

String.Compare(string1, string2)

Kết quả:

- <0 : string1 < string2
- =0 : string1 = string2
- >0 : string1 > string2



So sánh chuỗi



□ Phương thức CompareTo()

VD: string string1 = "abcd";

string string2 = "bcdf";

string1.CompareTo(string2);



Sao chép chuỗi



□ Phương thức copy sao chép nội dung một chuỗi đến một chuỗi khác

VD: string chuoi01 = "ppp";

```
string chuoiketqua = String.Copy(chuoi01);
```

```
=> chuoiketqua = "ppp"
```



Sao chép chuỗi



□ Phương thức Clone(): trả về bản sao của chuỗi dưới dạng đối tượng

VD: string chuoi1 = "abc"

```
object o1 = chuoi1.Clone();
```



Nối chuỗi



❑ Phương thức static Concat để nối 2 hay nhiều chuỗi.

VD: string string1 = "abc";

string string2 = "def";

string string3 = String.Concat(string1, string2)

=> string3 = "abcdef"



Thêm, xóa và chèn chuỗi



❑ Phương thức Insert chèn 1 chuỗi bất kỳ vào 1 chỉ mục của 1 chuỗi

VD: string chuoi1 = "adef";

string ketqua = chuoi1.Insert(1, "bc");

=> ketqua = "abcdef"



Thêm, xóa và chèn chuỗi



- Phương thức Remove xóa 1 số ký tự bắt kỳ từ 1 vị trí bất kỳ trong 1 chuỗi

VD: string chuoi1 = “abc123def”;

string ketqua = chuoi1.Remove(3,3);

=> ketqua = “abcdef”



Thay thế chuỗi



- Phương thức Replace thay thế những ký tự bất kỳ trong chuỗi thành những ký tự do bạn chỉ định.

VD: string string1 = “abcde”;

string ketqua = string1.Replace('a','b');

=> ketqua = “bbcde”



Cắt chuỗi



- ❑ Sử dụng phương thức Split để cắt một chuỗi thành nhiều chuỗi con dựa trên kí tự được định sẵn

VD: string string1 = “Trung tam tin hoc”;

```
char kt = new char[1]{' '};
```

```
string[] ketqua = string1.Split(kt);
```

```
=>ketqua[0] = “Trung”, ketqua[1] = “tam”...
```



UpperCase và LowerCase



- ❑ Phương thức ToUpper để in hoa các ký tự trong chuỗi

- ❑ Phương thức ToLower để viết thường các ký tự trong chuỗi



UpperCase và LowerCase



- VD: string string1 = “Trung tam tin hoc”;
 string upper = string1.ToUpper();
 string lower = string2.ToLower();
 =>upper = “TRUNG TAM TIN HOC”
 lower = “trung tam tin hoc”



3. Làm việc với StringBuilder



- **StringBuilder được dùng để tạo và bổ sung các chuỗi.**

- VD:

```
string returnNumber = "";
for(int i = 0; i<1000; i++)
{
    returnNumber = returnNumber + i.ToString();
}
```

=> returnNumber sẽ được gán 999 lần



3. Làm việc với StringBuilder



□ Sử dụng StringBuilder

```
StringBuilder returnNumber = new StringBuilder(10000);
for(int i = 0; i<1000; i++)
{
    returnNumber.Append(i.ToString());
}
```

=> returnNumber được mở rộng sau mỗi lần lặp



3. Làm việc với StringBuilder



□ Các phương thức quan trọng

- Append(): Nối 1 kiểu đối tượng vào cuối
- AppendFormat(): Nối 1 chuỗi được định dạng và cuối.
- Insert(): chèn 1 đối tượng vào một vị trí xác định



Demo

```
//Khai tao chuoi de su dung
string s1 = "Mot, hai, ba Trung Tam Tin Hoc";
//tao hang ky tu khoang trang va dau phay
const char Space= ' ';
const char Comma = ',';
//tao ra mang phan cach
char[] mang = new char[]
{
    Space,Comma
};
//su dung StringBuilder de tao chuoi output
StringBuilder output = new StringBuilder();
output.Append(9);
output.Append("sad");
output.Append(9.9);
output.Insert(1, "khos");
int ctr = 1;

foreach (string subString in s1.Split(mang))
{
    //Noi 1 chuoi duoc dinh dang vao chuoi output
    output.AppendFormat("{0}:{1}\n",ctr++,subString);
}

Console.WriteLine(output);
```



Thảo luận





Trường ĐH Khoa Hoc Tự Nhiên Tp. Hồ Chí Minh
TRUNG TÂM TIN HỌC

Go Screen Capture

Lập trình Windows Phone

Bài 3: *Mảng và collection trong C#*

Ngành Mạng & Thiết bị di động

2014





Nội dung



1. Tìm hiểu mảng (Array)

2. Tìm hiểu ArrayList

3. Tìm hiểu List



Tìm hiểu mảng(Array)



Khái niệm mảng

Phân loại mảng

Cách sử dụng mảng

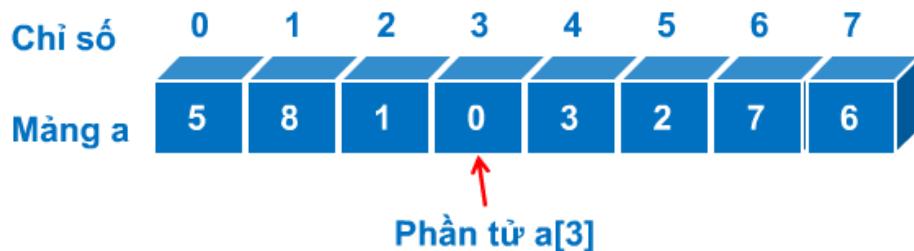
Một số phương thức và thuộc tính của lớp
System.Array



Khái niệm mảng



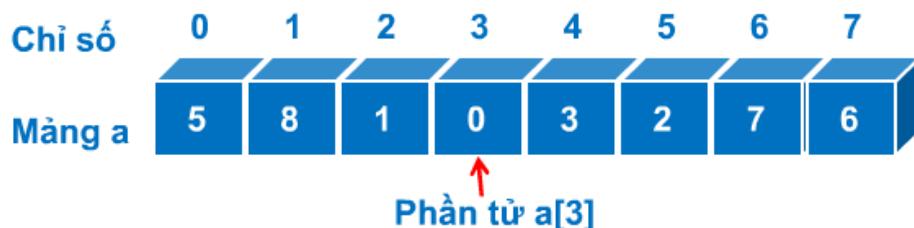
- Mảng là tập hợp nhiều giá trị cùng kiểu với một tên chung nhất và truy xuất thông qua chỉ số (index)



Phân loại mảng



- Mảng 1 chiều**



- Mảng nhiều chiều**

	Roll Number	Maths	Science
Student 1	1	87	76
Student 2	2	31	90
Student 3	3	75	34
Student 4	4	43	71

Two Dimensional Array



Các thao tác trên mảng



□ Khai báo và khởi tạo mảng

- Mảng 1 chiều

Khai báo không khởi tạo kích thước và giá trị

KiểuDữLiệu[] TênMảng;

- VD: int[] a;



Các thao tác trên mảng



□ Khai báo và khởi tạo

- Mảng 1 chiều

Khai báo, khởi tạo kích thước nhưng không có giá trị ban đầu.

KiểuDữLiệu[] TênMảng = new

KiểuDữLiệu[SỐPhầnTử];

VD: int[] a = new int[5];



Các thao tác trên mảng



□ Khai báo và khởi tạo

- Mảng 1 chiều

Khai báo, khởi tạo kích thước và giá trị ban đầu.

KiểuDữLiệu[] TênMảng = new

KiểuDữLiệu[] {value1,value2,...};

Hoặc

KiểuDữLiệu[] TênMảng = {giá trị 1, giá trị 2, giá trị 3, ...};

VD: int[] a = new int[]{2,10,4,8,5};

int[] a = {2, 10, 4, 8, 5};



Các thao tác trên mảng



□ Khai báo và khởi tạo

- Mảng 2 chiều

Khai báo, nhưng khởi tạo kích thước và giá trị ban đầu.

KiểuDữLiệu[,] TênMảng;

VD: int[,] a;



Các thao tác trên mảng



□ Khai báo và khởi tạo

- Mảng 2 chiều

Khai báo, khởi tạo kích thước nhưng không có giá trị ban đầu.

KiểuDữLiệu[,] TênMảng = new

KiểuDữLiệu[SỐDòng,SỐCột];

- VD: int[,] a = new int[2,3];



Các thao tác trên mảng



□ Khai báo và khởi tạo

- Mảng 2 chiều

Khai báo, khởi tạo kích thước và giá trị ban đầu.

KiểuDữLiệu[,] TênMảng = new

KiểuDữLiệu[SỐDòng,SỐCột]

{value1,value2,...}, {value1,value2,...};

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



Các thao tác trên mảng



□ Truy xuất phần tử trong mảng

- Mảng 1 chiều

Lấy giá trị của 1 phần tử :

TênMảng.GetValue(index);

TênMảng[index];

VD: int[] a = new int[5];

a.GetValue(3);

a[3];



Các thao tác trên mảng



□ Truy xuất phần tử trong mảng

- Mảng 1 chiều

Gán giá trị cho 1 phần tử :

TênMảng.SetValue(value,index);

TênMảng[index] = value;

VD: int[] a = new int[5];

a.SetValue(6, 4);

a[4] = 6;



Các thao tác trên mảng



□ Truy xuất phần tử trong mảng

- Mảng 2 chiều

Lấy giá trị của 1 phần tử :

TênMảng[indexDòng,indexCột];

VD: int[,] a = new int[2,5];

```
int giatri= a[1,2];
```



Các thao tác trên mảng



□ Truy xuất phần tử trong mảng

- Mảng 2 chiều

Gán giá trị cho 1 phần tử:

TênMảng[indexDòng,indexCột]=value;

VD: int[,] a = new int[2,5];

```
a[1,2]=7;
```



Các thao tác trên mảng



❑ Duyệt mảng

- Mảng 1 chiều

```
for (int i = 0; i < TênMảng.Length; i++)
{
    // Xử lý trên phần tử TênMảng[i]
}
foreach (int i in TênMảng)
{
    // Xử lý biến i
}
```



Các thao tác trên mảng



❑ Duyệt mảng

- Mảng 2 chiều

```
for (int i = 0; i < TênMảng.GetLength(0); i++)
{
    for(int j = 0;j<TênMảng.GetLength(1);j++)
    {
        // Xử lý trên phần tử TênMảng[i,j]
    }
}
```



Nội dung



1. Tìm hiểu mảng (Array)

2. **Tìm hiểu ArrayList**

3. Tìm hiểu List



Tìm hiểu ArrayList và cách sử dụng



Khái niệm ArrayList.

Một số phương thức quan trọng.



Khái niệm ArrayList



- ❑ Là mảng động có thể thay đổi kích thước.
- ❑ Đem lại sự tối ưu trong việc quản lý bộ nhớ.
- ❑ Các phần tử lưu trong ArrayList được định kiểu là Object.



Các thao tác trên arraylist



- ❑ Khai báo arraylist
- ❑ Thêm phần tử
- ❑ Duyệt arraylist
- ❑ Nối 2 arraylist



Các thao tác trên arraylist



- **Tách 1 arraylist từ arraylist ban đầu**
- **Xóa tất cả các phần tử trong arraylist**
- **Chèn 1 phần tử vào arraylist**
- **Kiểm tra 1 phần tử có trong arraylist không**



Khai báo arraylist



- **Tương tự khai báo một đối tượng**
 - `ArrayList arrayList1 = new ArrayList();`
- **Không cần khai báo kiểu dữ liệu cho arraylist.**
- **Có thể add bất cứ kiểu dữ liệu nào vào arraylist**



Thêm phần tử



❑ Dùng phương thức Add để thêm phần tử vào cuối arrayList

- VD:

```
ArrayList arraylist = new ArrayList();
string string1 = "chuoi 1";
arraylist.Add(string1);
```



Duyệt arrayList



❑ Dùng vòng lặp foreach để duyệt arrayList

```
ArrayList arraylist = new ArrayList();
foreach (var a in arraylist)
{
    Console.WriteLine(a.ToString());
}
```



Duyệt arraylist



□ Dùng vòng for và thuộc tính count để duyệt

```
ArrayList arraylist = new ArrayList();
for (int i = 0; i < arraylist.Count ; i++)
{
    Console.WriteLine(arraylist[i].ToString());
}
```



Nối 2 arraylist



□ Dùng phương thức AddRange() để nối 2 arraylist với nhau

- VD:

```
ArrayList arraylist = new ArrayList();
ArrayList arraylist1 = new ArrayList();
arraylist.Add("abv");
arraylist1.Add("ggg");
arraylist.AddRange(arraylist1);
=>arraylist("abv", "ggg")
```



Tách 1 arraylist từ arraylist ban đầu



□ Dùng phương thức GetRange()

- VD: `ArrayList arraylist = new ArrayList()`

```
{ "abc", "def", "ghi", "xyz" };
```

```
ArrayList arraylist2 = arraylist.GetRange(0, 2);
```

```
=> arraylist2 = ("abc", "def");
```



Xóa tất cả các phần tử trong arraylist



□ Dùng phương thức Clear để xóa tất các phần tử

- VD: `ArrayList arraylist = new ArrayList()`

```
{ "abc", "def", "ghi", "xyz" };
```

```
arraylist.Clear();
```

```
=>arraylist=();
```



Chèn 1 phần tử vào arraylist



□ Dùng phương thức Insert để chèn 1 phần tử vào vị trí bất kì trong arraylist

- VD:

```
ArrayList arraylist = new ArrayList()  
{ "abc", "def", "ghi", "xyz" };  
  
arraylist.Insert(1, "kkk");  
  
=> arraylist =( "abc", "kkk", "def", "ghi", "xyz" )
```



Kiểm tra 1 phần tử có trong arraylist không



□ Dùng phương thức Contains()

```
VD:ArrayList arraylist = new ArrayList()  
{ "abc", "def", "ghi", "xyz" };  
  
bool ketqua = arraylist.Contains("kkk");  
  
=> ketqua = false
```



Nội dung



1. Tìm hiểu mảng (Array)

2. Tìm hiểu ArrayList

3. **Tìm hiểu List**



Tìm hiểu List và cách sử dụng



Khái niệm

Cách khai báo List

Các phương thức thông dụng trong List



Khái niệm



- ❑ List ra đời để khắc phục các nhược điểm của ArrayList.
- ❑ Chứa hầu hết các phương thức và thuộc tính của ArrayList.



Cách khai báo List



- ❑ Không giống như ArrayList, khi khai báo List, ta phải định rõ kiểu dữ liệu cho List
- ❑ Cú pháp
 - List<Kiểu dữ liệu> tên List = new List<Kiểu dữ liệu>







Trường ĐH Khoa Khoa Hoc Tự Nhiên Tp. Hồ Chí Minh
TRUNG TÂM TIN HỌC

Go Screen Capture

Lập trình Windows Phone

Bài 4. Xây dựng lớp – Đối tượng

Ngành Mạng & Thiết bị di động

2014



SOFT



Nội dung



- Tổng quan về lập trình hướng đối tượng**
- Xây dựng class trong C#**



Tổng quan về lập trình hướng đối tượng



- Khái niệm**
- Các đặc trưng cơ bản**



Khái niệm



❑ Khái niệm

- Lập trình hướng đối tượng (OOP) là một phương pháp thiết kế và phát triển phần mềm dựa trên kiến trúc lớp(class) và đối tượng (object).
- OOP là cách lập trình nhằm hướng các xử lý đến từng đối tượng, mỗi đối tượng sẽ có các xử lý của riêng nó



Khái niệm



❑ Đối tượng (object) là một thực thể trong thực tế

- Con người
 - Nhân viên Trần Anh Tuấn
 - Sinh viên Lê Bảo Huy
- Đồ vật
 - Bàn B01
 - Phòng học E304
 - Chứng từ



Khái niệm



❑ Các thông tin về đối tượng:

- Ví dụ: Đối tượng Xe Ô tô

- Mã số xe
- Hiệu xe
- Màu sơn
- Hãng sản xuất
- Năm sản xuất

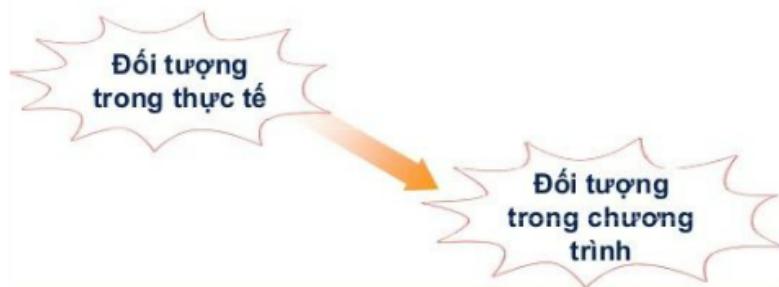


Khái niệm



❑ Tiếp cận hướng đối tượng

- Là kỹ thuật cho phép biểu diễn tự nhiên các đối tượng trong thực tế với các đối tượng bên trong chương trình.



Khái niệm



❑ Lớp đối tượng (Class)

- Class là một khái niệm trong Lập trình hướng đối tượng mô tả cho những thực thể có chung tính chất và hành vi mô tả cho những thực thể có chung tính chất và hành vi
- Kết quả của sự TRƯỞU TƯỢNG HOÁ (Abstraction) các đối tượng



Khái niệm



Khái niệm



❑ Các thành phần của lớp



Khái niệm



❑ Biến thành viên

- Lưu trữ các thông tin mô tả về đối tượng.
- Ví dụ: Lưu trữ thông tin về nhân viên
 - Mã nhân viên
 - Họ nhân viên
 - Tên nhân viên
 - Ngày sinh
 - Ngày vào làm

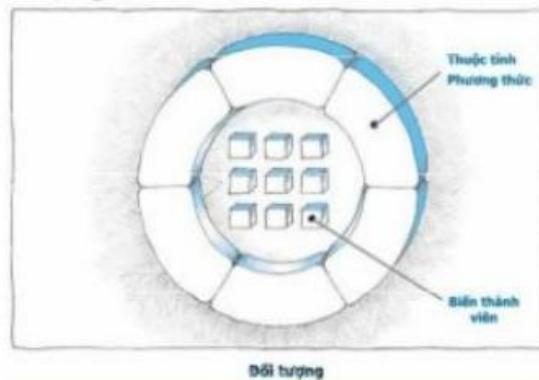


Khái niệm



□ Thuộc tính và Phương thức

- Dùng để cập nhật, tính toán, cung cấp và xử lý thông tin của đối tượng



Khái niệm



□ Thuộc tính và Phương thức

- VD: Lớp Nhân viên có
 - Thuộc tính
 - HoTen: lưu họ tên nhân viên
 - NamSinh: lưu năm sinh nhân viên
 - Phương thức
 - TienLuong: cách tính tiền lương
 - TroCap: cách tính trợ cấp



Khái niệm



- Sự kiện:** thông báo của đối tượng ra bên ngoài.

- VD: Lớp xe Ô tô

Thuộc tính →

XE_OTO
ID
Hieu_xe
Hang_san_xuat
Mau_son
Nam_san_xuat
No_may
Vao_so
Thang
Tat_may
Sap_het_xang

Phương thức →

Sự kiện →



Các đặc trưng cơ bản



- Tính trừu tượng (Abstraction)**

- Tính kế thừa (Inheritance)**



Tính trừu tượng



- ❑ **Lớp (class) là một khái niệm trừu tượng**
 - VD: Lớp Người, lớp Động vật

- ❑ **Đối tượng (object) là một thể hiện cụ thể của lớp**
 - VD: Con mèo là 1 thể hiện cụ thể của lớp động vật

- => từ những đối tượng giống nhau, ta có thể trừu tượng hóa thành 1 lớp.



Tính kế thừa



- ❑ Tính chất này cho phép xây dựng những lớp mới dựa trên những lớp sẵn có (lớp Con kế thừa lớp Cha)

- ❑ Lớp Con có khả năng bổ sung, mở rộng những tính năng mới dựa trên những phần sẵn có ở lớp Cha
 - Ví dụ: Các loại xe đời mới ngày càng hiện đại hơn và có nhiều chức năng hơn những loại xe đời cũ



Xây dựng class trong C#



- ❑ Khái niệm
- ❑ Khai báo class
- ❑ Tạo đối tượng kiểu class
- ❑ Xây dựng các thành phần trong class



Khái niệm



- ❑ Một class là kết quả của sự trừu tượng hóa đại diện chung cho dữ liệu và các hành vi của một thực thể hay một tập các đối tượng
- ❑ Các class còn mang ý nghĩa là một kiểu dữ liệu và là các khối xây dựng cơ sở của các trình ứng dụng hướng đối tượng



Khai báo class



❑ Cú pháp

<từ khóa khai báo phạm vi> class <tên lớp>

{

//Khai báo các biến thành viên (field)

//Khai báo các thuộc tính (property)

//Khai báo các phương thức (method)

//Khai báo các sự kiện (event)

}



Khai báo class trong C#



❑ Từ khóa khai báo phạm vi: xác định phạm vi hoạt động của class. Mặc định class sẽ có phạm vi hoạt động là private (cục bộ).

❑ Các từ khóa khai báo phạm vi thường dùng:

- Private
- Protected
- Public
- Static



Khai báo class trong C#



- VD: tạo ra một class có tên là nhân viên

```
public class NhanVien
{
    // Các thuộc tính của lớp
    public string ma_nv;
    public string ten_nv;
    public double luong;
    // Các phương thức của class
    public void nhap_nhan_vien(string _ma_nv,
        string _ten_nv, double _luong)
    {
        ma_nv = _ma_nv;
        ten_nv = _ten_nv;
        luong = _luong;
    }
}
```



Tạo đối tượng kiểu class



- Các đối tượng có thể được tạo bằng cách sử dụng từ khóa new theo sau đó là tên của class mà đối tượng dựa vào đó để khai báo:

<Tên class> tên_đối_tượng = new <Tên class>();

VD: NhanVien nhanvienA = new NhanVien()



Xây dựng các thành phần trong class



- Biến thành viên (Field)**
- Thuộc tính (Property)**
- Các phương thức (Method)**



Biến thành viên



- Biến thành viên là các trường dữ liệu để lưu trữ dữ liệu của một class.**
- Khi khai báo nếu không chỉ ra phạm vi truy xuất thì mặc định được hiểu là private.**



Biết thành viên



□ Khai báo

Class<tên lớp>

{

```
[private | public] kiểu_dữ_liệu
tên_biết_thành_viện1;
[private | public] kiểu_dữ_liệu
tên_biết_thành_viện2;
...
}
```



Thuộc tính (Property)



- **Thuộc tính là thành phần được sử dụng để truy xuất đến các biến thành viên được khai báo private bên trong class**
- **Mỗi thuộc tính chỉ truy xuất đến một biến thành viên duy nhất**



Thuộc tính (Property)



□ Khai báo

```
class <Tên lớp>
{
    [private | public] <Kiểu_dữ_liệu>
    <Tên_thuộc_tính>
    {
        get{ return <Tên biến thành viên>;}
        set{<Tên biến thành viên> = value;}
    }
}
```



Thuộc tính



```
class NhanVien
{
    //Khai bao bien thanh vien
    string mHoTen;
    //Khai bao thuoc tinh
    public string HoTen
    {
        get { return mHoTen; }
        set { mHoTen = value; }
    }
}
```



Thuộc tính



☐ Kiểm tra tính hợp lệ của dữ liệu

```
public string HoTen
{
    get { return mHoTen; }
    set {
        if(value != null)
            mHoTen = value;
    }
}
```



Các phương thức (Method)



☐ Phương thức khởi tạo (Constructor):

- Constructor là một phương thức đặc biệt có cùng tên với tên của class chứa nó.
- Constructor có vai trò khởi tạo các thành viên dữ liệu của đối tượng mới.



Các phương thức (Method)



□ Phương thức khởi tạo không có tham số:

- Được dùng cho việc khởi tạo các giá trị mặc định cho dữ liệu của class
- Khi khởi tạo đối tượng không phải truy cập tham số lúc gọi.



Các phương thức (Method)



□ Phương thức khởi tạo có tham số:

- Cho phép định nghĩa nhiều constructor trong một class. Mỗi constructor có tham số truy cập vào khác nhau





Các phương thức

❑ Các phương thức tính toán, xử lý

- Một phương thức là một khối lệnh chứa một dãy các câu lệnh cùng mục tiêu nào đó.

Cú pháp:

Class<tên lớp>

{

[private | public] <Kiểu dữ liệu> <Tên phương
thức>(){<Lệnh xử lý>}



}

Module 1 - Bài 4. Xây dựng lớp – Đối tượng

34



Thảo luận



Module 1 - Bài 4. Xây dựng lớp – Đối tượng

35



Trường ĐH Khoa Học Tự Nhiên Tp. Hồ Chí Minh
TRUNG TÂM TIN HỌC

Go Screen Capture

Lập trình Windows Phone

Bài 5. Kế thừa – Đa hình

Ngành Mạng & Thiết bị di động



2014

SOFT



Nội dung



- ❑ **Tính thừa kế (Inheritance)**
- ❑ **Lớp trừu tượng (Abstract)**
- ❑ **Lớp niêm phong (Sealed)**
- ❑ **Tính đa hình của lớp (Polymorphism)**
- ❑ **Giao diện (Interface)**



Tính thừa kế (Inheritance)



- ❑ **Khái niệm**
- ❑ **Xây dựng lớp kế thừa**



Khái niệm



- ❑ **Tính thừa kế cho phép tái sử dụng mã lệnh đang tồn tại giúp tiết kiệm được thời gian trong việc lập trình.**
- ❑ **Các class có thể thừa kế từ class khác.**



Khái niệm



- ❑ **Class mới được gọi là class được dẫn xuất (class con) sẽ được quyền truy xuất đến tất cả các thành viên dữ liệu và các phương thức không được biểu thị private của class cơ sở (class cha)**



Xây dựng class kế thừa



❑ Cú pháp:

Tên_class_con: Tên_class_cơ_sở

❑ Ví dụ 1:

- Xét class cơ sở sau:

```
public class ParentClass
{
    public ParentClass()
    {
        Console.WriteLine("Phuong thuc khai tao lop cha");
    }

    public void Print()
    {
        Console.WriteLine("Phuong thuc Print cua lop cha");
    }
}
```



Xây dựng class kế thừa



- Xét 1 class dẫn xuất từ class cơ sở với sau:

```
public class ChildClass:ParentClass
{
    public ChildClass()
    {
        Console.WriteLine("Phuong thuc khai tao lop con");
    }
}

static void Main(string[] args)
{
    ChildClass child = new ChildClass();
    child.ThuocTinh1 = "abc";
    child.ThuocTinh2 = "sda";
    child.Print();
}
```



Xây dựng class kế thừa



- Kết quả xuất ra màn hình

```
file:///D:/KetQuaLamViecThuanChuyenMon/MS_2005/C  
Phuong thuc khai tao cua class cha.  
Phuong thuc khai tao cua class con.  
Phuong thuc print cua class cha.
```



Lớp trừu tượng (Abstract)

Khái niệm

Xây dựng lớp trừu tượng



Khái niệm



- ❑ **Class trừu tượng thực chất là class cơ sở (base class) mà các class khác có thể được dẫn xuất từ nó.**
- ❑ **Các class không phải là class trừu tượng (non-abstract class) được gọi là lớp cụ thể (concrete class).**
- ❑ **Class trừu tượng có thể có hai loại phương thức: phương thức trừu tượng và phương thức cụ thể.**
- ❑ **Một kiểu được dẫn xuất từ một lớp cơ sở trừu tượng thừa kế tất cả các thành viên kiểu cơ sở bao gồm sự thực thi mọi phương thức.**



Khái niệm



❑ Khi nào thì sử dụng class trừu tượng?

- Nếu muốn tạo các class mà các class này sẽ chỉ là các class cơ sở, và không muốn bất cứ ai tạo các đối tượng của các kiểu class này.
- Class trừu tượng thường được dùng để biểu thị rằng nó là class không đầy đủ và rằng nó được dự định sẽ chỉ được dùng như là một class cơ sở.



Xây dựng lớp trừu tượng



□ Cú pháp:

<khai báo cấp độ truy xuất> abstract class
tên_class

{

//Các thành viên của class trừu tượng

}

Ví dụ:

- Xét class trừu tượng sau:



Xây dựng lớp trừu tượng



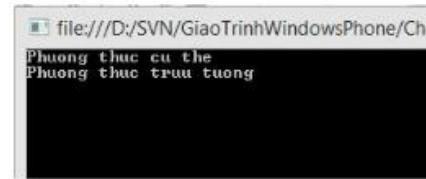
```
abstract class MyAbsClass
{
    public void NonAbMethod()
    {
        Console.WriteLine("Phuong thuc cu the");
    }

    public abstract void AbMethod(); //Phuong thuc truu tuong
}

public class MyClass:MyAbsClass
{
    public override void AbMethod()
    {
        Console.WriteLine("Phuong thuc truu tuong");
    }
}

static void Main(string[] args)
{
    MyAbsClass myAbsClass = new MyAbsClass();
    MyClass myClass = new MyClass();
    myClass.NonAbMethod();
    myClass.AbMethod();
    Console.ReadLine();
}
```

Không thể khai báo 1 đối tượng thuộc abstract class



Xây dựng lớp trừu tượng



❑ Chú ý:

- Mục tiêu của một class trừu tượng là cung cấp định nghĩa chung của một class cơ sở mà nhiều class được dẫn xuất có thể chia sẻ được.
- Một thành viên abstract không thể là static.
- Không thể tạo một instance của class trừu tượng.
- Một class trừu tượng không thể được niêm phong (Sealed)
- Một phương thức trừu tượng không thể là private.



Xây dựng lớp trừu tượng



❑ Chú ý:

- Từ khóa **Override**: hành động ghi đè (Override) là hành động sửa đổi hoặc thay thế sự cài đặt của class cha với một cài đặt mới. Các thành viên virtual hoặc abstract của class cha cho phép các class dẫn xuất ghi đè chúng.
- Phương thức abstract thực chất là một phương thức virtual ngầm định.





Lớp niêm phong (Sealed)

- ❑ Khái niệm
- ❑ Xây dựng lớp niêm phong



Khái niệm

- ❑ Từ khóa sealed được sử dụng để biểu thị khi khai báo một class, điều này cũng giống như việc ngăn cấm một class nào đó có class con.
- ❑ Một class sealed cũng không thể là một class trừu tượng.
- ❑ Các strung trong C# được ngầm định sealed. Do vậy, chúng không thể được thừa kế .





Xây dựng lớp niêm phong

❑ Cú pháp:

<khai báo cấp độ truy xuất> sealed class tên_class

{

//Các thành viên của class trừu tượng.

}



Tính đa hình của lớp

❑ Từ khóa base, this

❑ Ghi đè (Overriding)

❑ Nạp chồng hàm (Overloading)





Từ khóa base, this

□ Từ khóa base:

- Được sử dụng để tham chiếu đến lớp cơ sở từ lớp dẫn xuất.
- Ví dụ: xét lớp cơ sở nhan_vien

```
public class NhanVien
{
    string hoten;
    int tuoi;
    public NhanVien()
    {
        hoten = "";
        tuoi = 0;
    }

    public NhanVien(string _hoten, int _tuoi)
    {
        hoten = _hoten;
        tuoi = _tuoi;
    }
}
```

Module 1 - Bài 5. Thừa kế - Đa hình

20

Từ khóa base, this

□ Từ khóa base:

- Ví dụ: xét dẫn xuất nv_van_phong với khai báo như sau:

```
public class NhanVienVanPhong:NhanVien
{
    public string ChucVu;
    public NhanVienVanPhong(string _hoten,int _tuoi,string chucVu)
        :base(_hoten,_tuoi)
    {
        ChucVu = chucVu;
    }
}
```

- Trong ví dụ trên, để tham chiếu đến phương thức khởi tạo trong lớp cơ sở NhanVien phải sử dụng từ khóa base.



Module 1 - Bài 5. Thừa kế - Đa hình

21

Từ khóa this



- Được sử dụng để tham chiếu đến lớp hiện hành (lớp chứa đoạn lệnh đang cài đặt).

VD:

```
public class NhanVien
{
    string hoten;
    int tuoi;
    public NhanVien()
    {
        this.hoten = "";
        | this.tuoi = 0;
    }

    public NhanVien(string _hoten, int _tuoi)
    {
        this.hoten = _hoten;
        this.tuoi = _tuoi;
    }
}
```



Ghi đè (Overriding)

- Khái niệm ghi đè (overriding) được sử dụng để định nghĩa lại phương thức của lớp cơ sở (lớp cha) trong lớp dẫn xuất (lớp con kế thừa)

□ Các điểm cần lưu ý khi thực hiện ghi đè:

- Phương thức ở lớp cơ sở và lớp dẫn xuất phải có cùng dạng hàm (signature) và kiểu dữ liệu trả về.
- Phương thức lớp cơ sở phải được khai báo với từ khóa virtual.
- Phương thức lớp dẫn xuất phải được khai báo với từ khóa override .



Ghi đè (Overriding)



❑ VD: Xét lớp cơ sở NhanVien

```
public class NhanVien
{
    string hoten;
    int tuoi;
    float heso;
    float luongcoban;
    public NhanVien()
    {
        this.hoten = "";
        this.tuoi = 0;
    }

    public NhanVien(string _hoten, int _tuoi, float _heso)
    {
        hoten = _hoten;
        tuoi = _tuoi;
        heso = _heso;
    }

    public virtual double TinhLuong()
    {
        return luongcoban * heso;
    }
}
```



Ghi đè (Overriding)



❑ Xét lớp dẫn xuất NhanVienVanPhong

```
public class NhanVienVanPhong:NhanVien
{
    public int SoNgayVang;
    float DonGiaPhat = 50000;
    public NhanVienVanPhong(string _hoten, int _tuoi, float _heso, int _ngayvang)
        : base(_hoten, _tuoi, _heso)
    {
        SoNgayVang = _ngayvang;
    }

    public override double TinhLuong()
    {
        return base.TinhLuong() - SoNgayVang * DonGiaPhat;
    }
}
```





Ghi đè (Overriding)

❑ Lưu ý:

- Các phương thức ghi đè phải trùng tên.
- Không thể ghi đè các phương thức tĩnh (không có từ khóa virtual).
- Phương thức, thuộc tính, chỉ mục, sự kiện đều có thể được ghi đè bằng từ khóa virtual và override.



Giao diện (Interface)

❑ Interface quy định các chức năng nhưng không mô tả cụ thể chúng

❑ Phương thức trong interface

- Chỉ khai báo, không định nghĩa
- Không có từ khóa phạm vi, luôn là **public**

❑ Interface không thể chứa thuộc tính

❑ Từ khóa: interface



Giao diện (Interface)



[access modifier] **interface** <interface name> [: base interface list]

- ❑ **Một interface có thể kế thừa từ nhiều interface khác**
- ❑ **Một lớp có thể kế thừa từ nhiều interface**
 - Phải định nghĩa tất cả phương thức mà các interface "cha" quy định
- ❑ **Sự kế thừa interface phải đặt sau sự kế thừa lớp**



Giao diện (Interface)



- ❑ **Các interface thường được đặt tên với tiền tố là “I”**
 - IFile, IComparable, IDisposable, IStorable, ICloneable...

Có thể xem interface như một bản hợp đồng, nếu lớp nào sử dụng (kế thừa) nó thì phải thực thi đầy đủ các mô tả (phương thức) trong hợp đồng (interface)



Giao diện (Interface)



VD:

```
public interface IStudent
{
    int StudentID
    {
        get;
        set;
    }
    void AddSubject(string subjectName);
}
```

Khai báo property
StudentID gồm
hàm get, set

*Phải định nghĩa {get, set}
của StudentID và
AddSubject ở lớp thực thi
interface*



Giao diện (Interface)



```
public class Student:IStudent
{
    private int studentID = 0;
    private ArrayList subjects = null;
    public Student(){}
    public int StudentID
    {
        get { return studentID; }
        set { studentID = value; }
    }

    public void AddSubject(string subjectName)
    {
        subjects.Add(subjectName);
    }
}
```

Bắt buộc lớp
student phải
định nghĩa
property
StudentID và
hàm
AddSubject



Thảo luận





Trường ĐH Khoa Học Tự Nhiên Tp. Hồ Chí Minh
TRUNG TÂM TIN HỌC

Lập trình Windows Phone

Bài 6. Generic

Ngành Mạng & Thiết bị di động

2014



SOFT



Nội dung



□ Đặt vấn đề

□ Khái niệm

□ Các khai báo Generic



Đặt vấn đề



□ Xây dựng 1 class MyClass có 2 hàm khởi dụng

- Hàm 1: MyClass(int): xuất ra kiểu dữ liệu và giá trị truyền vào
- Hàm 2: MyClass(string): xuất ra kiểu dữ liệu và giá trị truyền vào



Đặt vấn đề



```
public class myClass
{
    //Ham khai dung 1
    public myClass(int myInt)
    {
        Console.WriteLine("My Type is: " + myInt.GetType());
        Console.WriteLine("My value is: " + myInt.ToString());
    }
    //Ham khai dung 2
    public myClass(string myStr)
    {
        Console.WriteLine("My Type is: " + myStr.GetType());
        Console.WriteLine("My value is: " + myStr.ToString());
    }
}
```

- Trường hợp cần xây dựng 5 hay 10 hàm khởi tạo tương tự?



Đặt vấn đề



- Sử dụng lớp Generic để giải quyết

```
public class myClass<T>
{
    T mytype;
    public myClass(T myType)
    {
        this.mytype = myType;
        Console.WriteLine("My Type is: " + typeof(T));
        Console.WriteLine("My value is: " + mytype);
    }
}
```



Khái niệm



❑ **Generic là 1 phần trong hệ thống kiểu dữ liệu của .NET cho phép định kiểu mà không quan tâm nhiều đến các chi tiết bên trong.**



Các khai báo Generic



❑ **Tạo kiểu Generic**

```
class Obj
{
    public Object t;
    public Object u;
    public Obj(Object _t, Object _u)
    {
        t = _t;
        u = _u;
    }
}
```

Kiểu thông thường

```
class Generic<T,U>
{
    public T t;
    public U u;
    public Generic(T _t,U _u)
    {
        t = _t;
        u = _u;
    }
}
```

Kiểu T, U sẽ
được xác định
khi sử dụng

Kiểu generic



Các khai báo Generic



□ Sử dụng kiểu Generic

- Chỉ định kiểu của kiểu generic khi sử dụng

```
Obj A = new Obj("Hello", "world");
Console.WriteLine(A.t.ToString() + A.u.ToString());

Gen<string, string> B = new
    Gen<string, string>("Hello", "world");
Console.WriteLine(B.t + B.u);
```

Kiểu thông thường

```
Obj A1 = new Obj(10.125, 1000);
Console.WriteLine((double)A1.t + (int)A1.u);

Gen<double, int> B1 = new
    Gen<double, int>(10.125, 1000);
Console.WriteLine(B1.t + B1.u);
```

Kiểu generic



Các khai báo Generic



```
Obj A1 = new Obj(10.125, 1000);
Console.WriteLine((int)A1.t + (int)A1.u);

Gen<int, int> B1 = new
    Gen<int, int>(10.125, 1000);
Console.WriteLine(B1.t + B1.u);
```

← Runtime Error

← Compile Error



Thảo luận





Trường ĐH Khoa Hoc Tự Nhiên Tp. Hồ Chí Minh
TRUNG TÂM TIN HỌC

Go Screen Capture

Lập trình Windows Phone

Bài 7. Xử lý biệt lệ Exception

Ngành Mạng & Thiết bị di động

2014





Nội dung



- ❑ Tình huống phát sinh ngoại lệ
- ❑ Cách xử lý ngoại lệ trong C#
- ❑ Câu lệnh try-catch-finally



Tình huống phát sinh ngoại lệ



- ❑ Những lỗi phát sinh trong lúc thực thi chương trình có thể làm hư hại chương trình
- ❑ Những lỗi không phải do lập trình
 - VD: Đĩa bị đầy, lỗi phần cứng, ...



Cách xử lý ngoại lệ trong C#



❑ Trong C#, khi runtime error xuất hiện

- CLR sẽ xác định lỗi và phát sinh ra đối tượng Exception
- Đối tượng này được ném trở lại stack, chờ một phương thức bắt lỗi đó
- Nếu exception này không được chương trình “catch” thì CLR sẽ in ra thông điệp lỗi



Sử dụng try - catch



❑ Đặt code có khả năng dẫn đến ngoại lệ vào khối “try”

❑ Cung cấp các khối catch sau try

- Có thể cung cấp tất cả catch cho các lỗi nếu muốn xử lý, bằng cách sử dụng các lớp exception thích hợp
- Nếu không cung cấp catch cho một ngoại lệ, thì exception này được lan truyền lên trên.



Cú pháp try-catch

```
try
{
    RiskyBusiness();
}
catch (SomeException e )
{
    // Handle code
}
```

Code có khả năng
dẫn đến lỗi

Tham số exception
được catch

Đoạn xử lý với tình
huống có lỗi



Khối try

❑ Khối “{...}” bắt buộc phải có, khác với
“{...}” trong if hay for

❑ Bên trong khối try

- Đặt bất cứ câu lệnh nào có khả năng phát sinh ra ngoại lệ



Khối catch



❑ Đặt một hay nhiều ngay sau khối try

- Không có lệnh nào chen giữa hai khối catch của một try

❑ Cú pháp khối catch như sau

```
catch (Exception-class [var1])
{
    // xử lý ngoại lệ 1
}
catch (Exception-class [var2])
{
    // xử lý ngoại lệ 2
}
```



```
static void Main(string[] args)
{
    int x = 0;
    int ketqua = 100 / x;
    Console.WriteLine(ketqua);
}
```



Chương trình bị
terminal

```
static void Main(string[] args)
{
    int x = 0;
    int ketqua = 0;
    try
    {
        ketqua = 100 / x;
        Console.WriteLine(ketqua);
    }
    catch (DivideByZeroException e)
    {
        Console.WriteLine("Exception occurred");
    }
    Console.WriteLine("Result is {0}", ketqua);
    Console.ReadLine();
}
```



Chương trình hoạt
động bình thường



Sử dụng finally



❑ Khi một exception được ném ra

- Luồng thực thi sẽ nhảy vào khối catch xử lý nó.
- Một số đoạn code giải phóng tài nguyên có thể bị bỏ qua

Open File

Read Data // ngoại lệ được phát sinh

Close File // đoạn code này bị bỏ qua, dù file chưa đóng

❑ Khối try-catch có phần option là finally

- Luôn luôn được gọi
- Sử dụng để dọn dẹp các tài nguyên đang nắm giữ



Lệnh throw



❑ Cho phép ném ra một ngoại lệ

❑ Cú pháp: throw exception_object

```
try
{
    throw new DivideByZeroException("Khong the chia cho 0");
}
catch (DivideByZeroException e)
{
    Console.WriteLine("Exception occurred");
}
Console.WriteLine("Result is {0}", ketqua);
Console.ReadLine();
```



Lớp Exception



- ❑ Có 2 loại ngoại lệ
 - Ngoại lệ phát sinh bởi chương trình
 - Ngoại lệ được tạo bởi CLR
- ❑ Lớp System.Exception là cơ sở cho tất cả lớp Exception trong C#
- ❑ 2 lớp kế thừa từ lớp này:
 - ApplicationException: thường làm lớp cơ bản cho lớp ngoại lệ phát sinh từ ứng dụng
 - SystemException: do CLR phát sinh



Lớp Exception



Một số lớp Exception thường dùng

- System.OutOfMemoryException
- System.NullReferenceException
- System.InvalidCastException
- System.ArrayTypeMismatchException
- System.IndexOutOfRangeException
- System.DevideByZeroException
- System.OverFlowException



Tự tạo lớp Exception



```
class MyException:ApplicationException
{
    public string Message;
    public MyException(string str)
    {
        Message = str;
        Console.WriteLine("User define Exception");
    }

    static void Main(string[] args)
    {
        try
        {
            throw new MyException("my error");
        }
        catch (MyException e)
        {
            Console.WriteLine("Exception catch here: " + e.Message.ToString());
        }
        Console.ReadLine();
    }
}
```



Thảo luận





Trường ĐH Khoa Học Tự Nhiên Tp. Hồ Chí Minh
TRUNG TÂM TIN HỌC

Go Screen Capture

Lập trình Windows Phone

Bài 8. Cơ bản về LINQ

Ngành Mạng & Thiết bị di động

2014





Nội dung



- ❑ **Tổng quan về LINQ**
- ❑ **LINQ làm việc với các đối tượng**
- ❑ **Sự biến đổi dữ liệu trong LINQ**
- ❑ **Truy vấn LINQ theo biểu thức**
- ❑ **Truy vấn LINQ theo phương thức**



Tổng quan về LINQ



- ❑ **Giới thiệu LINQ**
- ❑ **Các namespace hỗ trợ LINQ**
- ❑ **Kiến trúc và các thành phần của LINQ**
- ❑ **Hoạt động truy vấn của LINQ**



Đặt vấn đề



❑ Khi xây dựng ứng dụng chúng ta thường tương tác với dữ liệu đến từ nhiều nguồn khác nhau:

- Dữ liệu lưu trữ trong vùng nhớ
- Dữ liệu lưu trữ trong CSDL
- Tập tin XML

❑ Với mỗi nguồn lưu trữ, chúng ta sẽ có những cách truy vấn dữ liệu khác nhau



Tổng quan về LINQ



❑ Giới thiệu LINQ

- LINQ viết tắt Language-Integrated Query
- Là một tập các từ khóa trong C#, được dùng để truy vấn các nguồn dữ liệu khác nhau.

```
int[] numbers = new int[7] { 0, 1, 2, 3, 4, 5, 6 };
IQueryable<int> numQuery = from num in numbers
                           where (num % 2) == 0
                           orderby num descending
                           select num;
```



Tổng quan về LINQ



❑ Đặc điểm

- Truy vấn LINQ luôn nhất quán trong việc truy vấn dữ liệu bất chấp nguồn dữ liệu.
- Trong LINQ query, ta luôn làm việc với các đối tượng (objects)
- Có 2 dạng cú pháp sử dụng LINQ: Query syntax và Method syntax



Tổng quan về LINQ



❑ VD:

```
//Truy van tu array
int[] numbers = new int[7] { 0, 1, 2, 3, 4, 5, 6 };
//Query syntax
IQueryable<int> numQuery = from num in numbers
                           where (num % 2) == 0
                           orderby num descending
                           select num;
//Method syntax
IQueryable<int> numQuery1 = numbers.Where(n => n <= 3)
                                      .Select(n => n);
```



Tổng quan về LINQ



❑ Các namespace hỗ trợ LINQ

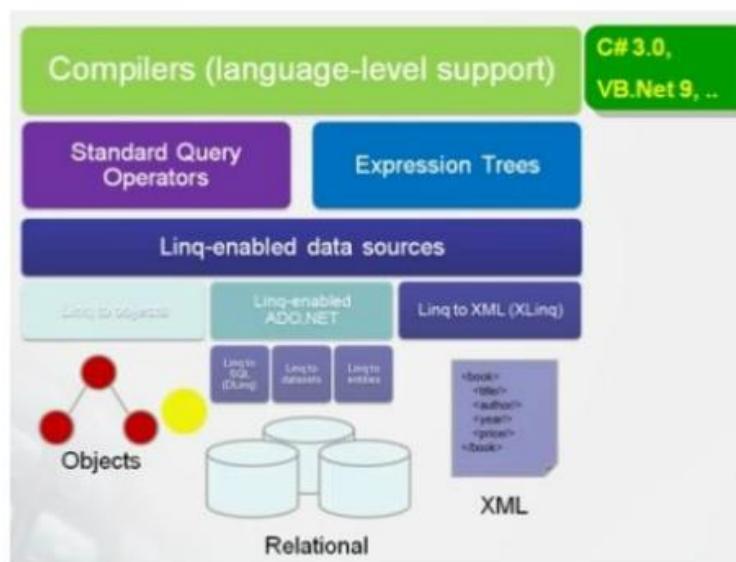
- System.Linq: hỗ trợ sử dụng các Objects
- System.Data.Linq: hỗ trợ sử dụng các CSDL quan hệ
- System.Data.Objects: hỗ trợ sử dụng các Entities
- System.XML.Linq: hỗ trợ sử dụng XML



Tổng quan về LINQ



❑ Kiến trúc và các thành phần của LINQ



Tổng quan về LINQ



❑ Hoạt động truy vấn của LINQ

- Cú pháp cơ bản

```
from id in source
where condition
orderby ordering, ordering, ...
select expr
```



Tổng quan về LINQ



❑ Hoạt động truy vấn của LINQ

- Ba thành phần của hoạt động truy vấn
 - Nhận nguồn dữ liệu
 - Tạo truy vấn
 - Thi hành truy vấn



Tổng quan về LINQ



❑ Hoạt động truy vấn của LINQ

VD:

```
// Khai báo nguồn dữ liệu
int[] Mangso = { 50, 42, 16, 3, 9, 8, 12, 7, 24, 0 };
// Tạo truy vấn
IQueryable<int> query = from n in Mangso
                           where n <= 3
                           select n;
// Thi hành truy vấn
StringBuilder stb = new StringBuilder();
foreach (int i in query)
    stb.Append(i + " ");
txtKetqua.Text = stb.ToString();
```



Tổng quan về LINQ



❑ Hoạt động truy vấn của LINQ

- LINQ và Generic Types:

- Các LINQ query đều dựa trên kiểu generic
- Các biến LINQ query thường được khai báo có kiểu là `IEnumerable<T>` và `IQueryable<T>`
- Tuy nhiên, vẫn có thể sử dụng từ khóa `var` để thay thế biến kiểu generic trong một số trường hợp



Thảo luận





Trường ĐH Khoa Học Tự Nhiên Tp. Hồ Chí Minh
TRUNG TÂM TIN HỌC

Go Screen Capture

Lập trình Windows Phone

Bài 9. Regular Expression (Regex)

Ngành Mạng & Thiết bị di động



2014

SOFT



Nội dung



❑ Khái niệm và vai trò của biểu thức chính quy

❑ Các class hỗ trợ trong C#

❑ Lớp ký tự dùng trong biểu thức chính quy



Regular Expression là gì



❑ Regular expression là chuỗi mẫu chứa các ký tự định dạng dùng để xác định xem văn bản có thỏa định dạng đưa ra không nhằm đảm bảo các chuỗi có định dạng chung nào đó.

- Ví dụ: kiểm tra ZIP code, địa chỉ email, họ tên hợp lệ...



Các class hỗ trợ



❑ .NET framework cung cấp các classes trong **System.Text.RegularExpressions:**

1. Regex class:

1. Có phương thức Match (static hoặc instance) trả về kết quả là các so trùng kiểu class Match
2. Matches trả về MatchCollection object



Các class hỗ trợ



2. Match class

Kiểu chứa một kết quả match tìm thấy.

ToString() trả về chuỗi con match với regular expression

3. MatchCollection class

1. Kiểu chứa tập nhiều match tìm thấy.



Sử dụng Regex để tìm kiếm



```
Regex regexObject = new Regex(Chuỗi_Mẫu);
Match matchObject = regexObject.Match(Chuỗi_Văn_Bản);
MatchCollection matchsObj =
    regexObject.Matches(Chuỗi_Văn_Bản);
```

Match:

Khi tìm thấy chuỗi con thỏa Chuỗi_Mẫu trong Chuỗi_Văn_Bản thì trả về matchObject chứa thông tin chuỗi con này.

Matches:

Khi tìm thấy các chuỗi con thỏa Chuỗi_Mẫu trong Chuỗi_Văn_Bản thì trả về MatchCollection object chứa thông tin các chuỗi con này.



Ví dụ



So sánh việc sử dụng thuật toán và sử dụng regular expression

```
// Nếu chỉ sử dụng thuật toán
string zip = "1234C";
bool isGoodZip = true;
foreach (char ch in zip)
{
    if (!char.IsDigit(ch))
    {
        isGoodZip = false;
        break;
    }
}
```

```
//dùng regular expression
string searchString = "1234C";
string regExString = @"^\d\d\d\d\d";
Regex rex = new Regex(regExString);
bool isMatch = rex.IsMatch(searchString);
```

```
//Cần US Zip code thì thay bằng:
string searchString = "12345-6789";
string regExString = @"^\d\d\d\d\d-\d\d\d\d";
```



Ví dụ dùng match và matches

```
□ using System;
└─ using System.Text.RegularExpressions;

□ namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            string testString = "regular expressions are sometimes called regex or regexp";
            Console.WriteLine("The test string is\n\"{0}\"\n", testString);
            Console.Write("Match 'e' in the test string: ");
            Regex expression = new Regex("e");
            Console.WriteLine(expression.Match(testString));
            Console.WriteLine();
            Console.Write("Match every 'e' in the test string: ");
            foreach (var myMatch in expression.Matches(testString))
                Console.Write("{0} ", myMatch);
            Console.WriteLine();
        }
    }
}
```



Ví dụ dùng match và matches

□ Kết quả



Character Class	Meaning (What It Matches)
.	Any character. Everything matches.
[abcd]	Any of the characters between the brackets. For "[aeiou]", "me" matches, but "by" doesn't.
[^abcd]	Any of the characters that are not between the brackets. For "[^aeiou]", "by" matches, but "me" doesn't.
[a-z]	Any of the characters in the range between the hyphen. For "[5-9]", "7" matches, but "3" doesn't.
\w	Any word character. Same as [a-zA-Z_0-9]. The string "_a1" matches, but "\r\n" doesn't.
\W	Any nonword character. Same as [^a-zA-Z_0-9]. The string "\r\n" matches, but "_a1" doesn't.
\s	Any whitespace character. Same as [\f\n\r\t\v]. The string "\r\n" matches, but "_a1" doesn't.
\S	Any nonwhitespace character. Same as [^\f\n\r\t\v]. The string "_a1" matches, but "\r\n" doesn't.
\d	Any decimal digit. The string "1" matches, but "a" doesn't.
\D	Any nondigit.

Demo

```

static void Main(string[] args)
{
    string testString = "abc, DEF, 123";
    Console.WriteLine("The test string is: \"(0)\", testString");
    Console.WriteLine("Match any digit");
    foreach (var regexMatch in Regex.Matches(testString, @"\d"))
        Console.Write("(0) ", regexMatch);
    Console.WriteLine("\nMatch any nondigit");
    foreach (var regexMatch in Regex.Matches(testString, @"\D"))
        Console.Write("(0) ", regexMatch);
    Console.WriteLine("\nMatch any word character");
    foreach (var regexMatch in Regex.Matches(testString, @"\w"))
        Console.Write("(0) ", regexMatch);
    Console.WriteLine("\nMatch anything from 'a' - 'f'");
    foreach (var regexMatch in Regex.Matches(testString, "[a-f]"))
        Console.Write("(0) ", regexMatch);
    Console.WriteLine("\nMatch anything not from 'a' - 'f'");
    foreach (var regexMatch in Regex.Matches(testString, "[^a-f]"))
        Console.Write("(0) ", regexMatch);
    Console.WriteLine("\nMatch a group of any characters");
    Console.Write("(0) ", Regex.Match(testString, "."));
}

```

Quantifier character class



❑ Có thể dùng lớp ký tự định lượng.

❑ Ví dụ: US Zip code:

`string regExString = @"\\d{5}-\\d{4}";`



Quantifier character class



Quantifier	Meaning (What It Matches)
*	Zero or more matches. For "\\d*", "", "123", "1234", ... matches.
+	One or more matches. For "\\d+", "123", "1234", ... matches, but not "".
?	Zero or one matches. For "\\d?", "" and "1" matches.
{n}	n matches.
{n,}	n or more matches.
{n,m}	At least n, but not more than m matches.



Demo

```
static void Main(string[] args)
{
    string testString = "regular expressions are sometimes called regex or regexp";
    Console.WriteLine("The test string is\n\"(0)\"", testString);
    Console.Write("\nMatch \"regex\" in the test string: ");
    foreach (var myMatch in Regex.Matches(testString, "regex"))
        Console.Write("(0) ", myMatch);
    Console.WriteLine("\nMatch \"regex\" or \"regexp\" using an optional 'p': ");
    // use the ? quantifier to include an optional 'p'
    foreach (var myMatch in Regex.Matches(testString, "regexp?"))
        Console.Write("(0) ", myMatch);
    Console.WriteLine();
}
```



Group, Optional

Ngoài ra còn có một số character class khác:

Ký tự	So trùng
^	Bắt buộc so trùng từ đầu chuỗi input
\$	Bắt buộc so trùng đến kết thúc chuỗi input
(exp1 exp2)	Hoặc, ý nghĩa so trùng với biểu thức chính quy exp1 hoặc exp2 vd: Hi (John Jane)
()	Tạo group nhiều ký tự định dạng vd: \d[a-z]+ khác với (\d[a-z])+
character class-character class	subtraction vd: "[\d-[4]]" mọi ký số trừ số 4



Demo

```
static void Main(string[] args)
{
    Regex expression = new Regex(@"J.*\d[\d-[4]]-\d\d-\d\d");
    string testString = "Jane's Birthday is 05-12-75\n" +
                        "Dave's Birthday is 11-04-68\n" +
                        "John's Birthday is 04-28-73\n" +
                        "Joe's Birthday is 12-17-77";
    foreach (var regexMatch in expression.Matches(testString))
        Console.WriteLine(regexMatch);
}
```



Thảo luận

