

ĐẠI HỌC KINH DOANH VÀ CÔNG NGHỆ HÀ NỘI
KHOA CÔNG NGHỆ THÔNG TIN



GIÁO TRÌNH

LẬP TRÌNH NÂNG CAO

Chủ biên : TS. Hoàng Xuân Thảo

Biên soạn: GS. Trần Anh Bảo

ThS. Phan Bảo Định

(Dùng cho chương trình đào tạo hệ đại học)

Lưu hành nội bộ

HÀ NỘI - 2015

LỜI MỞ ĐẦU

Theo khung chương trình của Bộ Giáo Dục và Đào Tạo, **Lập trình nâng cao** là một phần quan trọng trong học phần Tin học Đại cương thuộc các khối ngành Khoa học Tự nhiên, đặc biệt là ngành Công nghệ Thông tin.

Nhằm đáp ứng yêu cầu học tập của sinh viên bước đầu làm quen với công việc lập trình chuyên nghiệp, chúng tôi đã biên soạn bộ **Giáo Trình lập trình nâng cao** nhằm giúp cho sinh viên có một tài liệu học tập, rèn luyện tốt khả năng lập trình, tạo nền tảng vững chắc cho các môn học tiếp theo trong chương trình đào tạo Cử nhân Công nghệ Thông tin .

Giáo trình bao gồm rất nhiều bài tập từ đơn giản đến phức tạp. Các bài tập này được biên soạn dựa trên khung chương trình giảng dạy môn **Tin học Đại cương**. Bên cạnh đó, chúng tôi cũng bổ sung một số bài tập dựa trên cơ sở một số thuật toán chuẩn với các cấu trúc dữ liệu được mở rộng nhằm nâng cao kỹ năng, phương pháp lập trình cho sinh viên.

Nội dung của giáo trình được chia thành 8 chương. Trong mỗi chương đều có phần tóm tắt lý thuyết, phần bài tập mẫu và cuối cùng là phần bài tập tự giải để bạn đọc tự mình kiểm tra những kiến thức và kinh nghiệm đã học. Trong phần bài tập mẫu, đối với những bài tập khó hoặc có thuật toán phức tạp, tác giả thường nêu ra ý tưởng và giải thuật trước khi viết chương trình cài đặt.

Xin chân thành cảm ơn các đồng nghiệp ở Khoa Công nghệ Thông tin Trường Đại học Kinh doanh và công nghệ Hà Nội đã giúp đỡ, đóng góp ý kiến để hoàn chỉnh nội dung giáo trình này.

Tác giả hy vọng sớm nhận được những ý kiến đóng góp, phê bình của bạn đọc về nội dung, chất lượng và hình thức trình bày để giáo trình này ngày một hoàn thiện hơn.

Nhóm Biên soạn

MỤC LỤC

LỜI NÓI ĐẦU

Chương 1: Giới thiệu về C#, kiểu dữ liệu, cấu trúc rẽ nhánh

- 1.1 Mục đích môn học
- 1.2 Nội dung môn học
- 1.3 Sách giáo khoa và tài liệu tham khảo
- 1.4 Giới thiệu về C# và chương trình C# đầu tiên
- 1.5 Các kiểu dữ liệu và các phép toán
- 1.6 Cấu trúc rẽ nhánh

Chương 2: Các cấu trúc điều khiển, mảng và xử lý ngoại lệ

- 2.1 Các cấu trúc điều khiển
- 2.2 Mảng
- 2.3 Danh sách (list)
- 2.4 Xử lý ngoại lệ

Chương 3. Lập trình hướng đối tượng trong C#

- 3.1 Thuộc tính và phương thức
- 3.2 Constructor và Destructor
- 3.3 Nạp chồng hàm và nạp chồng toán tử
- 3.4 Kết thừa
- 3.5 Hàm ảo (virtual function), lớp trừu tượng (abstract class) và giao diện (interface)

Chương 4: Ủy quyền (delegate) và sự kiện (event)

- 4.1 Thực thi delegate
- 4.2 Thực thi multicast delegate
- 4.3 Sử dụng event với delegate

Chương 5: Thread và đồng bộ thread

- 5.1 Thực thi thread
- 5.2 Xác định vòng đời của thread
- 5.3 Độ ưu tiên của thread
- 5.4 Đồng bộ thread

Chương 6: Giới thiệu lập trình winform và winform control

6.1 Chương trình winform đầu tiên

6.2 Winform control cơ bản

6.3 Winform control nâng cao

6.4 Data Binding

6.5 Xử lý sự kiện

Chương 7: ADO.NET

7.1 Dịch vụ truy cập dữ liệu ADO.NET

7.2 Các thành phần cơ bản của ADO.NET

7.3 LINQ và ADO.NET

Chương 8: Giới thiệu lập trình webform và webform control

8.1 Nền tảng của webform

8.2 Tạo một ứng dụng webform

8.3 Data Binding

Tài liệu tham khảo

Chương 1: Giới thiệu về C#, kiểu dữ liệu, cấu trúc rẽ nhánh

1.1 Mục đích môn học

- Giúp sinh viên có thể hiểu và làm chủ được các định nghĩa, khái niệm và các thuật toán trên các kiểu dữ liệu cơ bản.
- Cài đặt, thực hiện và kiểm thử trên máy tính các thuật toán trên các kiểu dữ liệu cơ bản.
- Cải tiến hoặc mở rộng được các thuật toán cấu trúc rẽ nhánh.

1.2 Nội dung môn học

- Giới thiệu các khái niệm cơ bản về C# và chương trình C# đầu tiên
- Tìm hiểu về các kiểu dữ liệu và các phép toán cơ bản
- Tìm hiểu các cấu trúc rẽ nhánh thường dùng trong ngôn ngữ lập trình C#

1.3 Sách giáo khoa và tài liệu tham khảo

- Ngôn ngữ lập trình C# - Vietnam Open Educational Resources – VOER
- Ngôn ngữ lập trình nâng cao - ĐH Thủy Lợi
- Các giải pháp lập trình C# - Nhà sách Đất Việt
- Khoa CNTT- Giáo trình Lập trình nâng cao- ĐH Kinh Doanh và Công Nghệ HN

1.4 Giới thiệu về C# và chương trình C# đầu tiên

1.4.1. Tổng quan ngôn ngữ C#

C # là một ngôn ngữ lập trình hiện đại được phát triển bởi Microsoft và được phê duyệt bởi European Computer Manufacturers Association (ECMA) và International Standards Organization (ISO).

C # được phát triển bởi Anders Hejlsberg và nhóm của ông trong việc phát triển .Net Framework.

C # được thiết kế cho các ngôn ngữ chung cơ sở hạ tầng (Common Language Infrastructure – CLI), trong đó bao gồm các mã (Executable Code) và môi trường thực thi (Runtime Environment) cho phép sử dụng các ngôn ngữ cấp cao khác nhau trên đa nền tảng máy tính và kiến trúc khác nhau.

Ngôn ngữ ra đời cùng với .NET

- Kết hợp C++ và Java.
- Hướng đối tượng.

- Hướng thành phần.
- Mạnh mẽ (robust) và bền vững (durable).
- Mọi thứ trong C# đều Object oriented.
 - Kể cả kiểu dữ liệu cơ bản.
- Chỉ cho phép đơn kế thừa.
 - Dùng interface để khắc phục.
- Lớp **Object** là cha của tất cả các lớp.
 - Mọi lớp đều dẫn xuất từ Object.
- Cho phép chia chương trình thành các thành phần nhỏ độc lập nhau.
- Mỗi lớp gói gọn trong một file, không cần file header như C/C++.
- Bổ sung khái niệm namespace để gom nhóm các lớp.
- Bổ sung khái niệm “*property*” cho các lớp.
- Khái niệm delegate & event.

C# – mạnh mẽ & bền vững

- Garbage Collector
 - Tự động thu hồi vùng nhớ không dùng.
- Kiểm soát và xử lý ngoại lệ exception
 - Đoạn mã bị lỗi sẽ không được thực thi.
- Type – safe
 - Không cho gán các kiểu dữ liệu khác nhau.
- Versioning
 - Đảm bảo sự tương thích giữa lớp con và lớp cha.

Vai trò C# trong .NET Framework

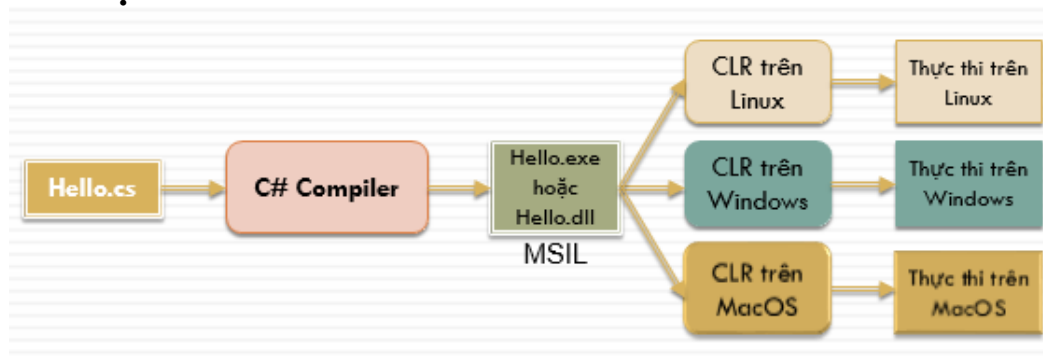
- .NET runtime sẽ phổ biến và được cài trong máy client.
 - Việc cài đặt App C# như là tái phân phối các thành phần .NET
 - Nhiều App thương mại sẽ được cài đặt bằng C#.
- C# tạo cơ hội cho tổ chức xây dựng các App Client/Server n-tier.
- Kết nối ADO.NET cho phép truy cập nhanh chóng & dễ dàng với SQL Server, Oracle...
- Cách tổ chức .NET cho phép hạn chế những vấn đề phiên bản.
 - Loại bỏ “*DLL Hell*”...
- ASP.NET viết bằng C#.
 - GUI thông minh.
 - Chạy nhanh hơn (đặc tính của .NET)

- Mã ASP.NET ko còn là mới hỗn độn.
- Khả năng bẫy lỗi tốt, hỗ trợ mạnh trong quá trình xây dựng App Web.

Quá trình dịch CT C#

- Mã nguồn C# (tập tin *.cs) được biên dịch qua MSIL.
- MSIL: tập tin .exe hoặc .dll
- MSIL được CLR thông dịch qua mã máy.
- Dùng kỹ thuật JIT (just-in-time) để tăng tốc độ.

Quá trình dịch CT C#



Các loại ứng dụng C#

Sử dụng C#, ta có thể tạo ra rất nhiều kiểu ứng dụng, ở đây ta quan tâm đến ba kiểu ứng dụng chính: Console, Window và ứng dụng Web

Ứng dụng Console

- Giao tiếp với người dùng bằng bàn phím.
- Không có giao diện đồ họa (GUI).

Ứng dụng Console là ứng dụng có giao diện text, chỉ xử lý nhập xuất trên màn hình Console, tương tự với các ứng dụng DOS trước đây.

Ứng dụng Console thường đơn giản, ta có thể nhanh chóng tạo chương trình hiển thị kết xuất trên màn hình. Do đó, các minh họa, ví dụ ngắn gọn ta thường sử dụng dạng chương trình Console để thể hiện.

Để tạo ứng dụng Console ta làm như sau

Trong Visual Studio, chọn File → New → Project. Visual Studio sẽ trình bày hộp thoại New Project.

Trong hộp thoại New Project, kích biểu tượng ứng dụng ConSole (Console Application). Trong ô name, gõ tên chương trình (dự án). Trong ô Location, gõ tên của thư mục mà ta muốn Visual Studio lưu dự án. Nhấn OK.

Visual Studio sẽ hiển thị cửa sổ. Ta nhập code vào trong cửa sổ này.

Ví dụ: Chương trình Console sau đây sử dụng hai phương thức `Console.ReadLine` và `Console.WriteLine` để nhập và xuất số nguyên a ra màn hình:

```
1 static void Main(string[] args)
2 {
3     int a = int.Parse(Console.ReadLine());
4     Console.WriteLine("a = " + a);
5     Console.ReadLine();
6 }
```

Chạy chương trình: Để chạy chương trình, ta chọn `Debug` → `Start` hoặc nhấn `F5`, Visual Studio sẽ hiển thị cửa sổ Console cho phép nhập và in số nguyên.

Ứng dụng Windows Form

- Giao tiếp với người dùng bằng bàn phím và mouse.
- Có giao diện đồ họa và xử lý sự kiện.

Là ứng dụng được hiển thị với giao diện cửa sổ đồ họa. Chúng ta chỉ cần kéo và thả các điều khiển (control) lên cửa sổ Form. Visual Studio sẽ sinh mã trong chương trình để tạo ra, hiển thị các thành phần trên cửa sổ.

Để tạo ứng dụng Window ta làm như sau:

`File` → `New` → `Project`. Visual Studio sẽ trình bày hộp thoại `New Project`.

Trong hộp thoại `New Project`, kích biểu tượng ứng dụng Windows (Windows Application). Trong ô `Name`, gõ tên mô tả chương trình mà ta dự định tạo (tên dự án). Tiếp theo, trong ô `Location`, gõ tên của thư mục mà ta muốn Visual Studio lưu dự án. Nhấn `OK`. Visual Studio sẽ hiển thị cửa sổ thiết kế. Ta có thể kéo và thả các thành phần giao diện (control) lên Form.

Để hiển thị cửa sổ Toolbox chứa những điều khiển mà ta có thể kéo và thả lên Form, ta chọn `View` → `Toolbox` từ menu.

Biên dịch và chạy chương trình: Để biên dịch chương trình, ta chọn `Build` → `Build Solution`. Để chạy chương trình, ta chọn `Debug` → `Start`. Nếu ta có thay đổi nội dung của Form, như đặt thêm điều khiển khác lên Form chẳng hạn, ta phải yêu cầu Visual Studio biên dịch lại.

Ứng dụng Web

- Kết hợp với ASP.NET, C# đóng vai trò xử lý bên dưới (underlying code).
- Có giao diện đồ họa và xử lý sự kiện.

Môi trường .NET cung cấp công nghệ ASP.NET giúp xây dựng những trang Web động. Để tạo ra một trang ASP.NET, người lập trình sử dụng ngôn ngữ biên dịch như C# hoặc C# để viết mã. Để đơn giản hóa quá trình xây dựng giao diện người dùng cho trang Web, .NET giới thiệu công nghệ Webform. Cách thức tạo ra các Web control tương tự như khi ta xây dựng ứng dụng trên Window Form.

Để tạo ứng dụng Web ta làm như sau

File → New → Project → Visual Basic Projects → ASP.NET Web Application

1.4.2. Chương trình C# đầu tiên

Một chương trình sẽ gồm các phần sau

- Namespace declaration: Khai báo namespace
- A class
- Class methods
- Class attributes
- A Main method
- Statements and expressions
- Comments

Xét ví dụ sau:

```
1 using System;
2 namespace HelloWorldApplication
3 {
4     class HelloWorld
5     {
6         static void Main(string[] args)
7         {
8             /* Chương trình C# đầu tiên của tôi */
9             Console.WriteLine("Hello oktot.com");
10            Console.ReadKey();
11        }
12    }
13 }
```

Khi đoạn code này biên dịch và thực thi sẽ hiện ra màn hình console dòng chữ “Hello oktot.com”.

Ta sẽ chỉ ra các thành phần trong đoạn code này:

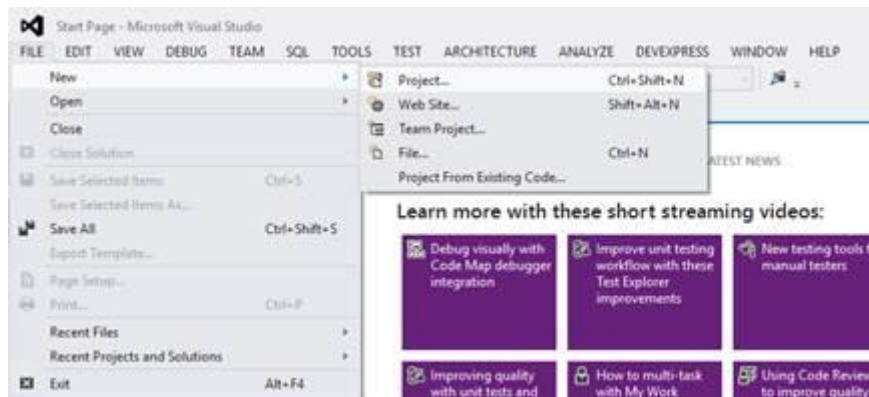
- Câu đầu tiên- **using System**; với từ khóa **using** được dùng để gọi đến namespace **System** của chương trình. Một chương trình có thể có nhiều câu lệnh
- Câu tiếp theo là namespace declaration, một namespace là một tập hợp nhiều class. HelloWorldApplication có chứa class
- Dòng tiếp theo là khai báo một class với tên HelloWorld, class này chứa dữ liệu và xác định các phương thức mà chương trình sử dụng. Phương thức xác định hành vi của class. Tuy nhiên ví dụ này chỉ có một phương thức **Main**.
- Dòng tiếp theo xác định một Main method. Đây là entry point của một chương trình C#. Main method nói lên công việc của chương trình khi thực thi. Ở ví dụ trên, hành vi được xác định là câu lệnh **WriteLine("Hello oktot.com")**; với chức năng hiển thị câu "Hello world" lên màn hình. **WriteLine** là một phương thức của class **Console** thuộc namespace **System** được hỗ trợ sẵn bởi C#.
- Dòng tiếp theo **/*..*/** là một dòng comment. Nó sẽ bị phớt lờ khi biên dịch chương trình. Nó được sử dụng để viết chú thích cho code.
- Dòng cuối cùng **ReadKey()**; có tác dụng đợi một hành động nhấn phím, nhằm ngăn màn hình console đóng quá nhanh khi người dùng chưa xem được kết quả.

Một số lưu ý khi bắt đầu tiếp xúc với C#:

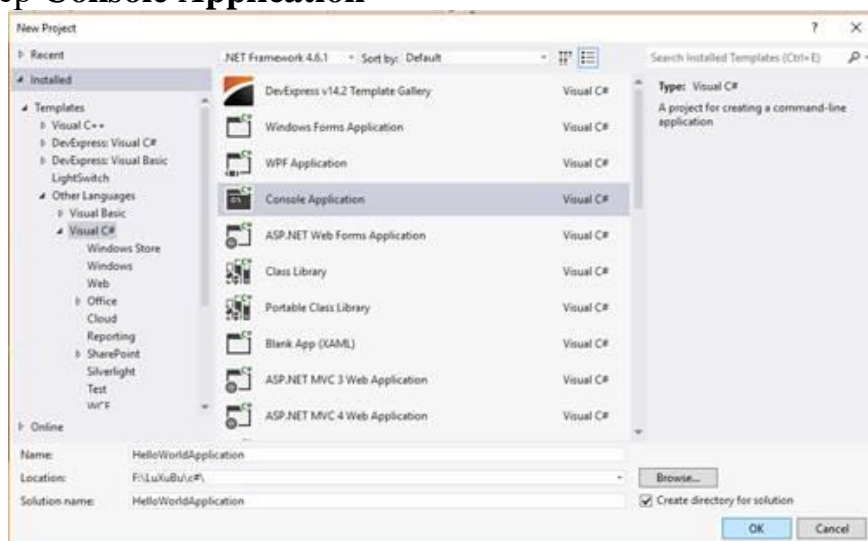
- C# rất nhạy bén
- Những câu lệnh và biểu thức luôn kết thúc bằng dấu chấm phẩy (;)
- Một chương trình bắt đầu thực thi tại Main method.
- Không giống với java, tên program file C# có thể khác với tên class.

Để biên dịch và thực thi một chương trình C# từ Visual studio .NET, các bạn làm theo các bước:

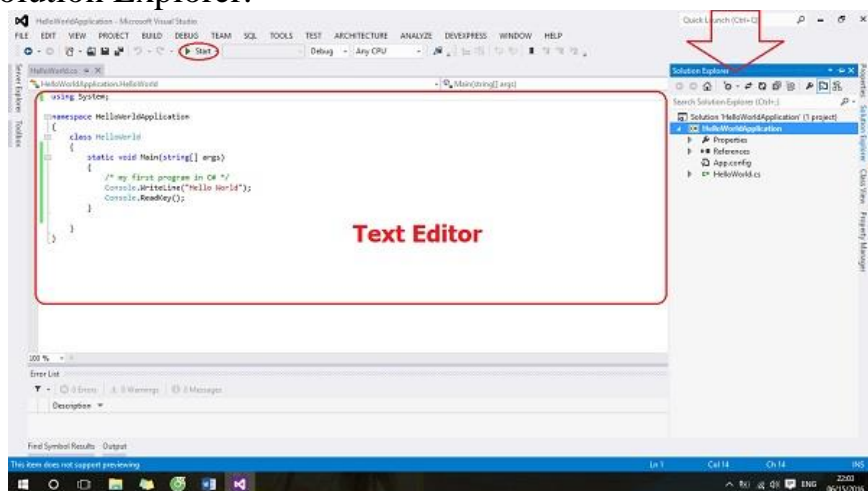
- Mở Visual studio
- Tại menu bar, chọn File->New->Project. Hoặc ta có thể sử dụng tổ hợp phím Ctrl+Shift+N.



- Chọn **visual C#** từ các mẫu cung cấp và chọn Windows
- Chọn tiếp **Console Application**

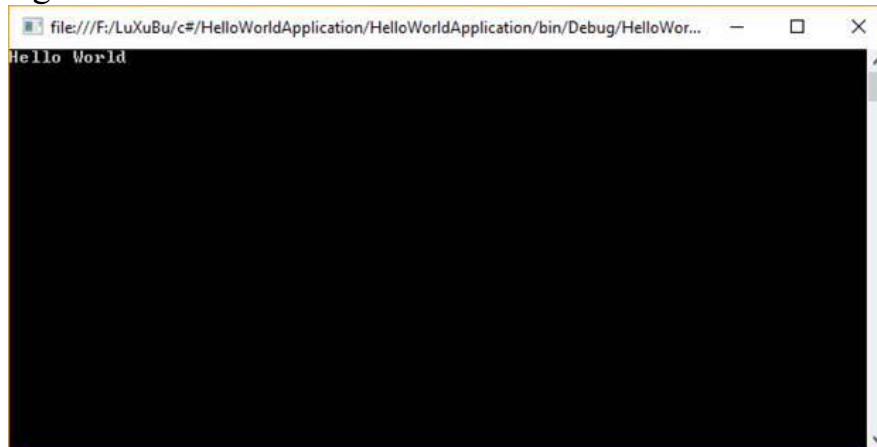


- Đặt tên cho project và click OK. Đến đây ta đã tạo được một project trong Solution Explorer.



- Ta sẽ viết code trong vùng code Editor

- Click Start hoặc F5 để thực thi project. Một màn hình console sẽ xuất hiện chứa dòng chữ Hello World.



1.5. Các kiểu dữ liệu và các phép toán

1.5.1. Các kiểu dữ liệu

Kiểu dữ liệu (*data type*) là một tập hợp gồm các nhóm dữ liệu có cùng đặc tính, cách lưu trữ của dữ liệu và các phép toán xử lý trên trường dữ liệu đó; nhằm mục đích phân loại các dữ liệu. Việc tìm hiểu và sử dụng đúng các kiểu dữ liệu sẽ giúp bạn tối ưu hóa được vùng nhớ cũng như sẽ sử dụng được các hàm xử lý dữ liệu dành riêng cho mỗi kiểu dữ liệu.

Các biến trong C# được phân ra làm ba kiểu:

- Kiểu giá trị (*value type*).
- Kiểu tham chiếu (*reference type*).
- Kiểu con trỏ (*pointer type*).

1.5.1.1. Kiểu giá trị - value type

Các biến thuộc kiểu này có thể được gán giá trị một cách trực tiếp, đây là kiểu dữ liệu được ngôn ngữ cung cấp. Nó kế thừa từ class `System.ValueType`.

Một biến khi khai báo kiểu dữ liệu thì hệ thống sẽ cấp phát bộ nhớ, giá trị thì vùng nhớ của biến đó sẽ chứa giá trị của dữ liệu. Ví dụ như khi chúng ta khai báo một kiểu `float` thì hệ thống sẽ cấp cho ta vùng nhớ có giá trị 4 bytes và được lưu trữ trên bộ nhớ **Stack**.

Bảng dưới đây liệt kê danh sách các kiểu dữ liệu có sẵn trong C#:

Kiểu số nguyên

Kiểu dữ liệu	Kích thước (bytes)	Ý nghĩa
byte	1	Số nguyên dương không dấu có giá trị từ 0 đến 255
sbyte	1	Số nguyên có dấu có giá trị từ -128 đến 127
short	2	Số nguyên có dấu có giá trị từ -32,768 đến 32,767
ushort	2	Số nguyên không dấu có giá trị từ 0 đến 65,535
int	4	Số nguyên có dấu có giá trị từ -2,147,483,647 đến 2,147,483,647
uint	4	Số nguyên không dấu có giá trị từ 0 đến 4,294,967,295
long	8	Số nguyên có dấu có giá trị từ -9,223,370,036,854,775,808 đến 9,223,370,036,854,775,807
ulong	8	Số nguyên không dấu có giá trị từ 0 đến 18,446,744,073,709,551,615

Kiểu ký tự

Kiểu dữ liệu	Kích thước (bytes)	Ý nghĩa
char	2	Chứa một ký tự Unicode

Kiểu logic

Kiểu dữ liệu	Kích thước (bytes)	Ý nghĩa
bool	1	Chứa 1 trong 2 giá trị logic là true hoặc false

Kiểu số thực

Kiểu dữ liệu	Kích thước (bytes)	Ý nghĩa
float	4	Kiểu số thực dấu chấm động có giá trị dao động từ $3.4E - 38$ đến $3.4E + 38$, với 7 chữ số có nghĩa
double	8	Kiểu số thực dấu chấm động có giá trị dao động từ $1.7E - 308$ đến $1.7E + 308$, với 15, 16 chữ số có nghĩa
decimal	8	Có độ chính xác đến 28 con số và giá trị thập phân, được dùng trong tính toán tài chính

Từ khóa sizeof()

Trong trường hợp bạn quên giá trị của từng loại kiểu dữ liệu, bạn có thể sử dụng biểu thức `sizeof(type)` để lấy kích cỡ chính xác của một biểu thức hoặc một biến nào đó. Giá trị trả về của `sizeof()` là kích cỡ của đối tượng hoặc kiểu bằng giá trị byte.

Ví dụ

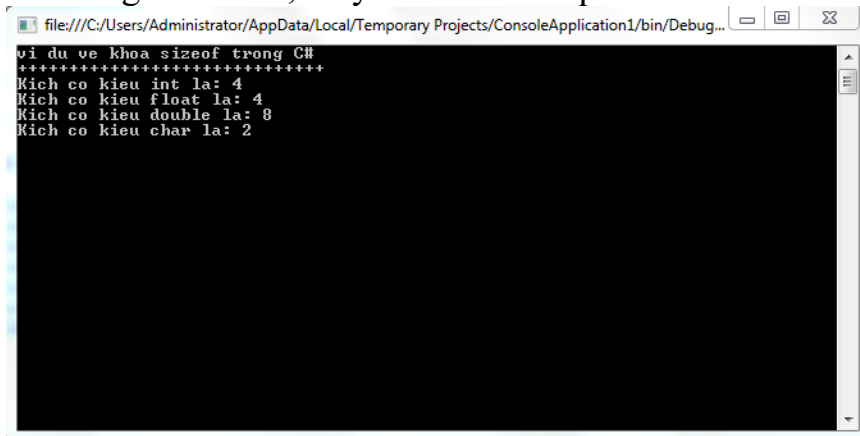
```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5
6 namespace freetuts
7 {
8     class freetuts
9     {
10         static void Main(string[] args)
11         {
12             Console.WriteLine("ví dụ về keyword sizeof trong C#");
13             Console.WriteLine("+++++++");
```

```

14     Console.WriteLine("Kích co kieu int la: {0}", sizeof(int));
15     Console.WriteLine("Kích co kieu float la: {0}", sizeof(float));
16     Console.WriteLine("Kích co kieu double la: {0}", sizeof(double));
17     Console.WriteLine("Kích co kieu char la: {0}", sizeof(char));
18     Console.ReadKey();
19 }
20 }
21 }

```

Khi thực thi chương trình trên, máy sẽ trả về kết quả:



```

file:///C:/Users/Administrator/AppData/Local/Temporary Projects/ConsoleApplication1/bin/Debug...
ví dụ về khóa sizeof trong C#
*****
Kích co kieu int la: 4
Kích co kieu float la: 4
Kích co kieu double la: 8
Kích co kieu char la: 2

```

1.5.1.2. Kiểu tham chiếu - reference type

Nếu biến khai báo kiểu dữ liệu tham chiếu thì vùng nhớ của nó không chứa dữ liệu thực sự được lưu trong một biến mà vùng nhớ của nó chỉ chứa địa chỉ của đối tượng dữ liệu. Hay chúng tham chiếu đến vị trí nào đó trong bộ nhớ tùy theo địa chỉ đã được lưu. Giả sử dữ liệu trong vùng nhớ thay đổi bởi một biến nào đó, thì các dữ liệu của biến khác cũng thay đổi tự động giá trị của mình.

Ta có một số kiểu dữ liệu thuộc kiểu tham chiếu có sẵn trong C# như: *object*, *dynamic*, *string*... Và tất cả các kiểu dữ liệu do người dùng định nghĩa đều là kiểu dữ liệu tham chiếu.

Kiểu object

Đây là kiểu dữ liệu cơ bản nhất của tất cả các kiểu dữ liệu trong .NET. Mọi kiểu dữ liệu đều được kế thừa từ `System.Object` nên kiểu *object* có thể gán mọi giá trị của các kiểu dữ liệu khác như kiểu tham chiếu, kiểu giá trị hoặc kiểu do người dùng tự định nghĩa,... Tuy nhiên trước khi gán giá trị nó cần được chuyển kiểu.

Ví dụ

```

1  object obj;
2  obj = 123;
3  // chúng ta đã chuyển kiểu dữ liệu object sang kiểu giá trị.

```

Kiểu dynamic

Đây là một kiểu dữ liệu mới được đưa vào trong C# 4.0. Ta có lưu mọi kiểu giá trị trong biến kiểu `dynamic`. Đối tượng thuộc kiểu này sẽ không xác định được kiểu dữ liệu cho đến khi chương trình được thực thi.

Ta khai báo kiểu `dynamic` như khai báo biến bình thường:

Cú pháp

- 1 `dynamic <tên biến>;`
- 2 `//hoặc`
- 3 `dynamic <tên biến> = <giá trị>;`

Kiểu string

Đây là một kiểu dữ liệu cho phép gán chuỗi vào biến. Nó được kế thừa từ kiểu *object*, có hai cách gán giá trị sau:

- 1 `string str = "freetuts.net";`
- 2 `//hoặc có thể dùng cách dưới đây, với cách này thì bạn có thể dữ nguyên chuỗi truyền vào`
- 3 `//kể cả chuỗi có nhiều dòng.`
- 4 `string str = @"freetuts.net`
- 5 `hoc lap trinh mien phi va chat luong";`

Ngoài ra còn có các kiểu dữ liệu tham chiếu do người dùng tự định nghĩa như `class`, `interface` hoặc `delegate` mà chúng ta sẽ tìm hiểu sau.

1.5.1.3. Kiểu con trỏ - pointer type.

Biến kiểu *con trỏ* lưu địa chỉ bộ nhớ của kiểu khác. Con trỏ trong C# có cùng khả năng như con trỏ trong C hoặc C++. Cú pháp để khai báo biến kiểu con trỏ là: `type*identifier`.

Ví dụ

- 1 `char*abc;`
- 2 `int*xyz;`

1.5.2. Các toán tử

Một toán tử là một biểu tượng, mà nói cho compiler thực hiện các thao tác toán học và logic cụ thể. C# cung cấp nhiều toán tử có sẵn, đó là:

- Toán tử số học
- Toán tử quan hệ
- Toán tử logic
- Toán tử so sánh bit
- Toán tử gán
- Toán tử hỗn hợp

1.5.2.1. Toán tử số học trong C#

Bảng dưới liệt kê các toán tử số học được hỗ trợ bởi ngôn ngữ C#. Giả sử biến A giữ giá trị 10, biến B giữ 20 thì:

Ví dụ

Toán tử	Miêu tả	Ví dụ
+	Thêm hai toán hạng	$A + B$ sẽ cho kết quả là 30
-	Trừ giá trị toán hạng hai từ toán hạng đầu	$A - B$ sẽ cho kết quả là -10
*	Nhân hai toán hạng	$A * B$ sẽ cho kết quả là 200
/	Chia lấy phần nguyên hai toán hạng	B / A sẽ cho kết quả là 2
%	Chia lấy phần dư	$B \% A$ sẽ cho kết quả là 0
++	Lượng gia giá trị toán hạng thêm 1 đơn vị	$A++$ sẽ cho kết quả là 11
--	Lượng giảm giá trị toán hạng một đơn vị	$A--$ sẽ cho kết quả là 9

1.5.2.2. Toán tử quan hệ trong C#

Bảng dưới đây liệt kê các toán tử quan hệ được hỗ trợ bởi ngôn ngữ C#. Giả sử biến A giữ giá trị 10, biến B giữ 20 thì:

Ví dụ

Toán tử	Miêu tả	Ví dụ
==	Kiểm tra nếu 2 toán hạng bằng nhau hay không. Nếu bằng thì điều kiện là true.	$(A == B)$ là không đúng.
!=	Kiểm tra 2 toán hạng có giá trị khác nhau hay không. Nếu không bằng thì điều kiện là true.	$(A != B)$ là true.
>	Kiểm tra nếu toán hạng bên trái có giá trị lớn hơn toán hạng bên phải hay không. Nếu lớn hơn thì điều kiện là true.	$(A > B)$ là không đúng.

<	Kiểm tra nếu toán hạng bên trái nhỏ hơn toán hạng bên phải hay không. Nếu nhỏ hơn thì là true.	$(A < B)$ là true.
>=	Kiểm tra nếu toán hạng bên trái có giá trị lớn hơn hoặc bằng giá trị của toán hạng bên phải hay không. Nếu đúng là true.	$(A >= B)$ là không đúng.
<=	Kiểm tra nếu toán hạng bên trái có giá trị nhỏ hơn hoặc bằng toán hạng bên phải hay không. Nếu đúng là true.	$(A <= B)$ là true.

1.5.2.3. Toán tử logic trong C#

Bảng dưới đây chỉ rõ tất cả các toán tử logic được hỗ trợ bởi ngôn ngữ C#. Giả sử biến A có giá trị 1 và biến B có giá trị 0:

Ví dụ

Toán tử	Miêu tả	Ví dụ
&&	Được gọi là toán tử logic AND (và). Nếu cả hai toán tử đều có giá trị khác 0 thì điều kiện trở lên true.	$(A \&\& B)$ là false.
	Được gọi là toán tử logic OR (hoặc). Nếu một trong hai toán tử khác 0, thì điều kiện là true.	$(A \parallel B)$ là true.
!	Được gọi là toán tử NOT (phủ định). Sử dụng để đảo ngược lại trạng thái logic của toán hạng đó. Nếu điều kiện toán hạng là true thì phủ định nó sẽ là false.	

1.5.2.4. Toán tử so sánh bit trong C#

Toán tử so sánh bit làm việc trên đơn vị bit, tính toán biểu thức so sánh từng bit. Bảng dưới đây về &, |, và ^ như sau:

p	q	$p \& q$	$p q$	$p \wedge q$
0	0	0	0	0
0	1	0	1	1

1	1	1	1	0
1	0	0	1	1

Giả sử nếu $A = 60$; và $B = 13$; thì bây giờ trong định dạng nhị phân chúng sẽ là như sau:

$A = 0011\ 1100$

$B = 0000\ 1101$

$A \& B = 0000\ 1100$

$A | B = 0011\ 1101$

$A \wedge B = 0011\ 0001$

$\sim A = 1100\ 0011$

Các toán tử so sánh bit được hỗ trợ bởi ngôn ngữ C# được liệt kê trong bảng dưới đây. Giả sử ta có biến A có giá trị 60 và biến B có giá trị 13, ta có:

Ví dụ

Toán tử	Miêu tả	Ví dụ
&	Toán tử AND (và) nhị phân sao chép một bit tới kết quả nếu nó tồn tại trong cả hai toán hạng.	$(A \& B)$ sẽ cho kết quả là 12, tức là 0000 1100
	Toán tử OR (hoặc) nhị phân sao chép một bit tới kết quả nếu nó tồn tại trong một hoặc hai toán hạng.	$(A B)$ sẽ cho kết quả là 61, tức là 0011 1101
^	Toán tử XOR nhị phân sao chép bit mà nó chỉ tồn tại trong một toán hạng mà không phải cả hai.	$(A \wedge B)$ sẽ cho kết quả là 49, tức là 0011 0001
~	Toán tử đảo bit (đảo bit 1 thành bit 0 và ngược lại).	$(\sim A)$ sẽ cho kết quả là -61, tức là 1100 0011.
<<	Toán tử dịch trái. Giá trị toán hạng trái được dịch chuyển sang trái bởi số các bit được xác định bởi toán hạng bên phải.	$A \ll 2$ sẽ cho kết quả 240, tức là 1111 0000 (dịch sang trái hai bit)
>>	Toán tử dịch phải. Giá trị toán hạng trái được dịch chuyển sang phải bởi số các bit được xác định bởi toán hạng bên phải.	$A \gg 2$ sẽ cho kết quả là 15, tức là 0000 1111 (dịch sang phải hai bit)

1.5.2.5. Toán tử gán trong C#

Đây là những toán tử gán được hỗ trợ bởi ngôn ngữ C#:

Ví dụ

Toán tử	Miêu tả	Ví dụ
=	Toán tử gán đơn giản. Gán giá trị toán hạng bên phải cho toán hạng trái.	$C = A + B$ sẽ gán giá trị của $A + B$ vào trong C
+=	Thêm giá trị toán hạng phải tới toán hạng trái và gán giá trị đó cho toán hạng trái.	$C += A$ tương đương với $C = C + A$
-=	Trừ đi giá trị toán hạng phải từ toán hạng trái và gán giá trị này cho toán hạng trái.	$C -= A$ tương đương với $C = C - A$
*=	Nhân giá trị toán hạng phải với toán hạng trái và gán giá trị này cho toán hạng trái.	$C *= A$ tương đương với $C = C * A$
/=	Chia toán hạng trái cho toán hạng phải và gán giá trị này cho toán hạng trái.	$C /= A$ tương đương với $C = C / A$
%=	Lấy phần dư của phép chia toán hạng trái cho toán hạng phải và gán cho toán hạng trái.	$C \% = A$ tương đương với $C = C \% A$
<<=	Dịch trái toán hạng trái sang số vị trí là giá trị toán hạng phải.	$C <<= 2$ tương đương với $C = C << 2$
>>=	Dịch phải toán hạng trái sang số vị trí là giá trị toán hạng phải.	$C >>= 2$ tương đương với $C = C >> 2$
&=	Phép AND bit	$C \&= 2$ tương đương với $C = C \& 2$
^=	Phép OR loại trừ bit	$C \wedge= 2$ tương đương với $C = C \wedge 2$
=	Phép OR bit.	$C = 2$ tương đương với $C = C 2$

1.5.2.6. Các toán tử hỗn hợp trong C#

Dưới đây là một số toán tử hỗn hợp quan trọng gồm **sizeof**, **typeof** và **? :** được hỗ trợ bởi ngôn ngữ C#.

Ví dụ

Toán tử	Miêu tả	Ví dụ
sizeof()	Trả về kích cỡ của một kiểu dữ liệu	sizeof(int), trả về 4
typeof()	Trả về kiểu của một lớp	typeof(StreamReader);
&	Trả về địa chỉ của một biến	&a; trả về địa chỉ thực sự của biến
*	Trỏ tới một biến	*a; tạo con trỏ với tên là a tới một biến
? :	Biểu thức điều kiện (Conditional Expression)	Nếu Condition là true ? Thì giá trị X : Nếu không thì Y
is	Xác định đối tượng là một kiểu cụ thể hay không	If(Ford is Car) // Kiểm tra nếu Ford là một đối tượng của lớp Car
as	Ép kiểu mà không tạo một exception nếu việc ép kiểu thất bại	Object obj = new StreamReader("Hello"); StreamReader r = obj as StreamReader;

1.5.2.7. Thứ tự ưu tiên toán tử trong C#

Thứ tự ưu tiên toán tử trong C# xác định cách biểu thức được tính toán. Ví dụ, toán tử nhân có quyền ưu tiên hơn toán tử cộng, và nó được thực hiện trước.

Ví dụ, $x = 7 + 3 * 2$; ở đây, x được gán giá trị 13, chứ không phải 20 bởi vì toán tử * có quyền ưu tiên cao hơn toán tử +, vì thế đầu tiên nó thực hiện phép nhân $3 * 2$ và sau đó thêm với 7.

Bảng dưới đây liệt kê thứ tự ưu tiên của các toán tử. Các toán tử với quyền ưu tiên cao nhất xuất hiện trên cùng của bảng, và các toán tử có quyền ưu tiên thấp nhất thì ở bên dưới cùng của bảng. Trong một biểu thức, các toán tử có quyền ưu tiên cao nhất được tính toán đầu tiên.

Ví dụ

Loại	Toán tử	Thứ tự ưu tiên
Postfix	() [] -> . ++ --	Trái sang phải

Unary	+ - ! ~ ++ - - (type)* & sizeof	Phải sang trái
Tính nhân	* / %	Trái sang phải
Tính cộng	+ -	Trái sang phải
Dịch chuyển	<< >>	Trái sang phải
Quan hệ	< <= > >=	Trái sang phải
Cân bằng	== !=	Trái sang phải
Phép AND bit	&	Trái sang phải
Phép XOR bit	^	Trái sang phải
Phép OR bit		Trái sang phải
Phép AND logic	&&	Trái sang phải
Phép OR logic		Trái sang phải
Điều kiện	?:	Phải sang trái
Gán	= += -= *= /= %= >>= <<= &= ^= =	Phải sang trái
Dấu phẩy	,	Trái sang phải

1.6. Cấu trúc rẽ nhánh

Rẽ nhánh là một trong cấu trúc quan trọng nhất và không thể thiếu trong mọi ngôn ngữ lập trình, nó giúp điều khiển luồng thực thi của chương trình theo ý của lập trình viên. Cấu trúc rẽ nhánh trong C# cũng giống như C và C++.

Cấu trúc đầy đủ

if (điều kiện 1)

{

<đoạn mã 1>

}

else if (điều kiện 2)

{

<đoạn mã 2>

}

else if (điều kiện n)

```

{
    <đoạn mã n>
}
else
{
    <đoạn mã s>
}

```

Điều này có nghĩa :

Nếu <điều kiện 1> đúng, thực hiện <đoạn mã 1> , nếu không, tiếp tục kiểm tra <điều kiện 2>

Nếu <điều kiện 2> đúng, thực hiện <đoạn mã 2>, nếu không, tiếp tục kiểm tra <điều kiện n>

Nếu <điều kiện n> đúng, thực hiện <đoạn mã n>

Nếu tất cả điều kiện trên đều sai, thực hiện <đoạn mã s>

Chú ý:

- Nếu <đoạn mã> chỉ gồm 1 câu lệnh, ta có thể không cần sử dụng dấu { }

Cấu trúc đơn giản

if (điều_kiện)

```

{
    <đoạn mã>
}

```

Đây là cấu trúc đơn giản nhất của rẽ nhánh, nó có nghĩa là: nếu <điều kiện> đúng, thực hiện <đoạn mã>

Ví dụ

Sau đây là một số ví dụ giúp bạn hiểu rõ hơn về cấu trúc rẽ nhánh

```

private void Main(string[] args)
{
    Console.WriteLine("Nhap diem sinh vien :");
    float diem = float.Parse(Console.ReadLine());
    if (diem >= 8.0)
        Console.WriteLine("GIOI");
    else if (diem >= 6.5)
        Console.WriteLine("KHA");
    else
        Console.WriteLine("TRUNG BINH");
}

```

Ở ví dụ này, người dùng nhập điểm của một sinh viên, từ đó chương trình in ra đó là điểm GIỎI, KHÁ hay TRUNG BÌNH

+ Các <đoạn mã> chỉ gồm 1 câu lệnh Console.WriteLine vì vậy không cần sử dụng mở đóng ngoặc { }

Chương 2: Các cấu trúc điều khiển, mảng và xử lý ngoại lệ

2.1 Các cấu trúc điều khiển

Cấu trúc điều khiển (Control Structure) là một trong các đặc trưng cơ bản của phương pháp lập trình cấu trúc. Trong đó, người ta sử dụng 3 cấu trúc điều khiển để tạo nên logic của chương trình.

2.1.1. Tổng quan về cấu trúc điều khiển

- Một chương trình không chỉ bao gồm các lệnh tuần tự nối tiếp nhau. Trong quá trình chạy nó có thể rẽ nhánh hay lặp lại một đoạn mã nào đó. Để làm điều này chúng ta sử dụng các cấu trúc điều khiển.
- Cùng với việc giới thiệu các cấu trúc điều khiển chúng ta cũng sẽ phải biết tới một khái niệm mới: khối lệnh, đó là một nhóm các lệnh được ngăn cách bởi dấu chấm phẩy (;) nhưng được gộp trong một khối giới hạn bởi một cặp ngoặc nhọn: { và }.
- Nếu khối lệnh chỉ có 1 lệnh thì không cần sử dụng 3 cặp dấu ngoặc nhọn { và }

2.1.2. Trong C# có 3 loại cấu trúc cơ bản:

- Cấu trúc tuần tự
- Cấu trúc quyết định chọn lựa rẽ nhánh
- Cấu trúc lặp

2.1.3. Các cấu trúc điều khiển

- Cấu trúc rẽ nhánh
- Toán tử điều kiện ?
- Cấu trúc lựa chọn switch
- Cấu trúc lặp while, do, for, foreach

2.1.3.1. Cấu trúc rẽ nhánh - IF

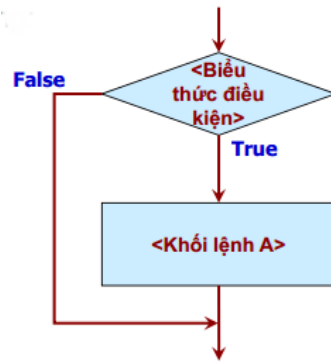
Cú pháp

```
if (<Điều kiện>)  
{  
    <Khối lệnh A>  
}
```

Ý nghĩa: Nếu <Điều kiện> Đúng Thì Thực hiện <Khối lệnh A>

<Điều kiện>: là 1 biểu thức logic, trả về true (Đúng) hoặc false (Sai)

Sơ đồ hoạt động:



Ví dụ: Kiểm tra điểm trung bình để xét kết quả đậu hay rớt. Nếu điểm trung bình ≥ 5 thì kết quả là đạt

```

Console.Write("Nhập vào điểm trung bình: ");
float dtb = (float)Console.Read();
if (dtb >= 5)
    Console.WriteLine("Kết quả đạt");
  
```

Dạng 2: if else ... else ...

- Đặt vấn đề: Trường Y có nhu cầu xét kết quả học tập của học sinh dựa vào điểm trung bình để quyết định xem học sinh đó có được lên lớp hay không. Có 2 trường hợp có thể xảy ra:
 - Trường hợp 1: được lên lớp (điểm trung bình ≥ 5.0)
 - Trường hợp 2: không được lên lớp (điểm trung bình ≤ 5.0)
- 2 trường hợp của bài toán trên loại trừ nhau, để giải quyết bài toán này chúng ta dùng cấu trúc if ... else ...

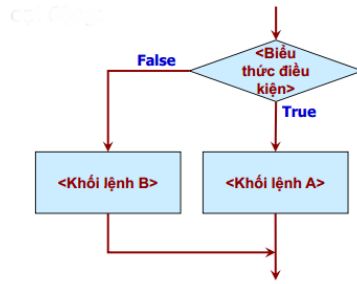
Cú pháp

```

if (<Điều kiện>)
{
    <Khối lệnh A>
}
else // khi điều kiện của if là false
{
    <Khối lệnh B>
}
  
```

Ý nghĩa Nếu <Điều kiện> Đúng Thì Thực hiện <Khối lệnh A> Ngược lại, <Điều kiện> Sai Thực hiện <Khối lệnh B>

Sơ đồ hoạt động



Ví dụ 1: Xét kết quả học tập dựa vào điểm trung bình Nếu DTB <5 thì Kết quả là Ở lại lớp Ngược lại là Được lên lớp

```

if (dtb >= 5)
    Console.WriteLine("Được lên lớp");
else
    Console.WriteLine("Ở lại lớp");
  
```

Ví dụ 2: Xét kết quả học tập dựa vào điểm trung bình Nếu DTB <5 thì Kết quả là Ở lại lớp và phải thi lại Ngược lại là Được lên lớp và không phải thi lại

```

double dtb = double.Parse(Console.ReadLine());
if (dtb >= 5)
{
    Console.WriteLine("Kết quả đạt");
    Console.WriteLine("Bạn không phải thi lại");
}
else
{
    Console.WriteLine("Kết quả không đạt");
    Console.WriteLine("Bạn vui lòng thi lại");
}
  
```

2.1.3.2. Toán tử điều kiện - ?

- Toán tử ? hoạt động tương tự như dạng 2 của cú pháp IF
- Cú pháp
 - $\text{Biến_kết_quả} = \langle \text{Điều kiện} \rangle ? \langle \text{biểu_thức_1} \rangle : \langle \text{biểu_thức_2} \rangle$
- Ý nghĩa
 - Nếu <Điều kiện> Đúng
 - Thì Trả về <Biểu thức 1>
 - Ngược lại
 - Trả về <Biểu thức 2>
- <Điều kiện>: là 1 biểu thức logic, trả về true hoặc false

Ví dụ: Xét kết quả học tập dựa vào điểm trung bình: Nếu DTB <5 thì Kết quả là Ở lại lớp. Ngược lại là Được lên lớp

```

double dtb = double.Parse(Console.ReadLine());
  
```

```
string ket_qua = (dtb > 5) = 5) ? "Được lên lớp" : "Ở lại lớp";  
Console.WriteLine("Kết quả: {0}", ket_qua);
```

Cấu trúc lựa chọn – switch

- Với cấu trúc IF, khi có nhiều trường hợp cần xét, ta sẽ dùng toán tử || để nối các điều kiện phức tạp khi có quá nhiều điều kiện
- Do đó có thể sử dụng cấu trúc chọn switch để thay thế cho cấu trúc IF trong trường hợp này

```
switch (<biểu thức chọn lựa>)  
{  
    case <giá trị 1>:  
        <Tập lệnh 1>  
        break;  
    case <giá trị 2>:  
        <Tập lệnh 2>  
        break;  
    ...  
    default: // các lệnh thực thi khi <biểu thức> không bằng bất kỳ <giá trị> nào  
        của case  
        <Tập lệnh n>  
        break;  
}
```

Ý nghĩa

- case: Liệt kê các trường hợp cần xét
- Giá trị i: chứa các giá trị cần so sánh với <biểu thức>
- Tập lệnh x: được thực hiện khi biểu thức chọn lựa = một trong số các giá trị của <Tập giá trị i>

Ví dụ: nhập vào thứ, cho biết tên thứ trong tuần

```
int thu = Int.Parse(Console.ReadLine());  
switch (thu)  
{  
    case 2:  
        Console.WriteLine("Thứ Hai");  
        break;  
    ...  
    case 8:  
        Console.WriteLine("Chủ Nhật");  
        break;  
    default:  
        Console.WriteLine("Thứ nhập vào không hợp lệ");
```

```
break;
}
```

Chú ý: Nếu như các trường hợp cần xét có cùng một tập giá trị thì lần lượt liệt kê các trường hợp, sau đó mới viết tập giá trị

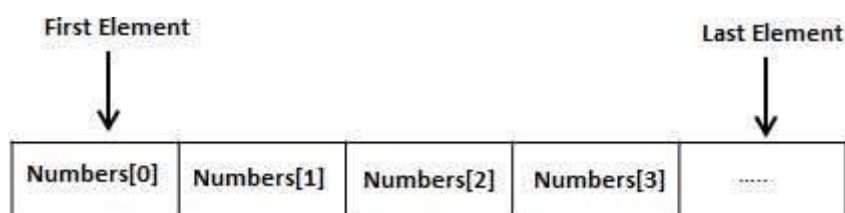
```
int thang = int.Parse(Console.ReadLine());
int nam = int.Parse(Console.ReadLine());
switch(thang)
{
    case 1:
    case 3:
    case 5:
    case 7:
    case 8:
    case 10:
    case 12:
        Console.WriteLine("Tháng {0} năm {1} có 31 ngày", thang, nam);
        break;
}
```

2.2 Mảng

Một mảng lưu giữ một tập hợp các phần tử có kích cỡ cố định trong cùng kiểu. Một mảng được sử dụng để lưu giữ một tập hợp dữ liệu, nhưng nó thường hữu ích hơn khi nghĩ về một mảng như là một tập hợp các biến cùng kiểu được lưu giữ tại các vị trí bộ nhớ kế nhau.

Thay vì khai báo biến một cách rời rạc, như biến `number0`, `number1`,... và `number99`, bạn có thể khai báo một mảng các giá trị như `numbers[0]`, `numbers[1]` và ... `numbers[99]` để biểu diễn các giá trị riêng biệt. Một thành viên cụ thể của mảng có thể được truy cập qua index (chỉ số).

Tất cả mảng đều bao gồm các vị trí nhớ liên kế nhau. Địa chỉ thấp nhất tương ứng với thành viên đầu tiên và địa chỉ cao nhất tương ứng với thành viên cuối cùng của mảng.



Khai báo mảng trong C#

Để khai báo một mảng trong ngôn ngữ C#, bạn có thể sử dụng cú pháp:

```
kiểu_dữ_liệu[] tên_mảng;
```

Tại đây:

- *kiểu_dữ_liệu* được sử dụng để xác định kiểu của phần tử trong mảng.
- *[]* xác định rank hay kích cỡ của mảng.
- *tên_mảng* xác định tên mảng.

Ví dụ:

```
double[] balance;
```

Khởi tạo mảng trong C#

Việc khai báo một mảng không khởi tạo mảng trong bộ nhớ. Khi biến mảng được khởi tạo, bạn có thể gán giá trị cho mảng đó.

Mảng là một kiểu tham chiếu, vì thế bạn cần sử dụng từ khóa **new** trong C# để tạo một Instance (sự thể hiện) của mảng đó. Ví dụ:

```
double[] balance = new double[10];
```

Gán giá trị cho một mảng trong C#

Bạn có thể gán giá trị cho các phần tử mảng riêng biệt bởi sử dụng chỉ số mảng, như:

```
double[] balance = new double[10];  
balance[0] = 4500.0;
```

Bạn có thể gán giá trị cho mảng tại thời điểm khai báo mảng, như sau:

```
double[] balance = { 2340.0, 4523.69, 3421.0};
```

Bạn cũng có thể tạo và khai báo một mảng, như sau:

```
int [] marks = new int[5] { 99, 98, 92, 97, 95};
```

Bạn cũng có thể bỏ qua kích cỡ mảng, như:

```
int [] marks = new int[] { 99, 98, 92, 97, 95};
```

Bạn có thể sao chép một biến mảng vào trong biến mảng mục tiêu khác. Trong tình huống này, cả biến mục tiêu và biến nguồn đều trở tới cùng vị trí bộ nhớ:

```
int [] marks = new int[] { 99, 98, 92, 97, 95};  
int[] score = marks;
```

Khi bạn tạo một mảng, C# compiler ngầm định khởi tạo mỗi phần tử mảng thành một giá trị mặc định phụ thuộc vào kiểu mảng. Ví dụ, với một mảng int, thì tất cả phần tử được khởi tạo là 0.

Truy cập các phần tử mảng trong C#

Một phần tử được truy cập bởi chỉ mục mảng. Điều này được thực hiện bởi việc đặt chỉ số của phần tử bên trong dấu ngoặc vuông ở sau tên mảng. Ví dụ:

```
double salary = balance[9];
```

Ví dụ sau minh họa khái niệm về khai báo, gán và truy cập mảng trong C# đã đề cập ở trên:

```
using System;

namespace Ten_namespace_C
{
    class TestCsharp
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Mang trong C#");
            Console.WriteLine("-----");
            int [] n = new int[10]; /* n la mot mang gom 10 so nguyen */
            int i,j;
            /* khoi tao cac phan tu cua mang n */
            for (i = 0; i < 10; i++)
            {
                n[i] = i + 100;
            }

            /* hien thi gia tri cac phan tu cua mang n */
            for (j = 0; j < 10; j++)
            {
                Console.WriteLine("Phan tu [{0}] = {1}", j, n[j]);
            }
            Console.ReadKey();
        }
    }
}
```

Nếu bạn không sử dụng lệnh **Console.ReadKey();** thì chương trình sẽ chạy và kết thúc luôn (nhanh quá đến nỗi bạn không kịp nhìn kết quả). Lệnh này cho phép chúng ta nhìn kết quả một cách rõ ràng hơn.

Biên dịch và chạy chương trình C# trên sẽ cho kết quả sau:

```
Mang trong C#
-----
Phan tu [0] = 100
Phan tu [1] = 101
Phan tu [2] = 102
Phan tu [3] = 103
Phan tu [4] = 104
Phan tu [5] = 105
Phan tu [6] = 106
Phan tu [7] = 107
Phan tu [8] = 108
Phan tu [9] = 109
```

Sử dụng vòng lặp *foreach* trong C#

Trong ví dụ trước, chúng ta đã sử dụng một vòng lặp `for` để truy cập mỗi phần tử trong mảng. Bạn cũng có thể sử dụng một lệnh **`foreach`** để duyệt qua một mảng trong C#:

```
using System;

namespace Ten_namespace_C
{
    class TestCsharp
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Mang trong C#");
            Console.WriteLine("-----");
            int[] n = new int[10]; /* n la mot mang gom 10 so nguyen */

            /* khoi tao cac phan tu trong mang n */
            for (int i = 0; i < 10; i++)
            {
                n[i] = i + 100;
            }

            /* hien thi cac gia tri cua phan tu trong mang n */
            foreach (int j in n)
            {
                int i = j - 100;
                Console.WriteLine("Phan tu [{0}] = {1}", i, j);
                i++;
            }
            Console.ReadKey();
        }
    }
}
```


Nếu bạn không sử dụng lệnh **Console.ReadKey();** thì chương trình sẽ chạy và kết thúc luôn (nhANH quá đến nỗi bạn không kịp nhìn kết quả). Lệnh này cho phép chúng ta nhìn kết quả một cách rõ ràng hơn.

Biên dịch và chạy chương trình C# trên sẽ cho kết quả sau:

```
Mảng trong C#
-----
Phan tu [0] = 100
Phan tu [1] = 101
Phan tu [2] = 102
Phan tu [3] = 103
Phan tu [4] = 104
Phan tu [5] = 105
Phan tu [6] = 106
Phan tu [7] = 107
Phan tu [8] = 108
Phan tu [9] = 109
```

Chi tiết về mảng trong C#

Mảng là một phần rất quan trọng trong ngôn ngữ C#. Dưới đây là những định nghĩa quan trọng liên quan đến mảng mà được trình bày rõ ràng hơn cho các lập trình viên C#:

Khái niệm	Miêu tả
Mảng đa chiều trong C#	C# hỗ trợ mảng đa chiều. Mẫu đơn giản nhất của mảng đa chiều là mảng hai chiều
Jagged array trong C#	C# hỗ trợ mảng đa chiều, mà là mảng của các mảng
Truyền mảng tới hàm trong C#	Bạn có thể truyền cho hàm một con trỏ tới một mảng bằng việc xác định tên mảng mà không cần chỉ số của mảng
Mảng tham số trong C#	Được sử dụng để truyền một số lượng chưa biết của các tham số tới một hàm
Lớp Array trong C#	Được định nghĩa trong System namespace, nó là lớp cơ sở cho tất cả mảng, và cung cấp các thuộc tính và phương thức để làm việc với mảng

2.3 Danh sách (list)

List là 1 Generic Collections đưa ra như một sự thay thế ArrayList vì thế về khái niệm cũng như sử dụng nó hoàn toàn giống với ArrayList. (bạn có thể tham khảo chi tiết trong bài ARRAYLIST TRONG C#)

Ở đây mình chỉ trình bày lại một số ý để những bạn nào không theo dõi những bài trước vẫn có thể hiểu được.

List trong C# là một Generic Collections giúp lưu trữ và quản lý một danh sách các đối tượng theo kiểu mảng (truy cập các phần tử bên trong thông qua chỉ số **index**).

Để sử dụng các Collections trong .NET ta cần thêm thư viện System.Collections.Generic bằng câu lệnh:

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

Vì List là một lớp nên trước khi sử dụng ta cần khởi tạo vùng nhớ bằng toán tử new:

```
List<int> MyList = new List<int>(); // khởi tạo 1 List các số nguyên rỗng 0
```

Bạn cũng có chỉ định sức chứa (Capacity) ngay lúc khởi tạo bằng cách thông qua constructor được hỗ trợ sẵn:

```
// khởi tạo 1 List các số nguyên và chỉ định Capacity ban đầu là 5
```

```
List<int> MyList2 = new List<int>(5);
```

```
0
```

```
1
```

```
2
```

```
3
```

```
4
```

Ngoài ra bạn cũng có thể khởi tạo 1 List chứa các phần tử được sao chép từ một Generic Collections khác (lưu ý là có cùng kiểu dữ liệu truyền vào):

```
/*
```

```
* Khởi tạo 1 List số nguyên có kích thước bằng với MyList2.
```

```
* Sao chép toàn bộ phần tử trong MyList2 vào MyList3.
```

```
*/
```

```
List<int> MyList3 = new List<int>(MyList2);
```

Một số thuộc tính và phương thức hỗ trợ sẵn trong List

Một số thuộc tính thông dụng trong List:

TÊN THUỘC TÍNH	Ý NGHĨA
Count	Trả về 1 số nguyên là số phần tử hiện có trong List .
Capacity	Trả về 1 số nguyên cho biết số phần tử mà List có thể chứa (sức chứa). Nếu số phần tử được thêm vào chạm sức chứa này thì hệ thống sẽ tự động tăng lên. Ngoài ra ta có thể gán 1 sức chứa bất kỳ cho List .

Một số phương thức thông dụng trong List:

TÊN PHƯƠNG THỨC	Ý NGHĨA
Add (object Value)	Thêm đối tượng Value vào cuối List .
AddRange (ICollection ListObject)	Thêm danh sách phần tử ListObject vào cuối List .
BinarySearch (object Value)	Tìm kiếm đối tượng Value trong List theo thuật toán tìm kiếm nhị phân. Nếu tìm thấy sẽ trả về vị trí của phần tử ngược lại trả về giá trị âm. Lưu ý là List phải được sắp xếp trước khi sử dụng hàm.
Clear ()	Xoá tất cả các phần tử trong List .
Contains (T Value)	Kiểm tra đối tượng Value có tồn tại trong List hay không.
CopyTo (T[] array, int Index)	Thực hiện sao chép tất cả phần tử trong List sang mảng một chiều array từ vị trí Index của array. Lưu ý array phải là mảng kiểu T tương ứng.
IndexOf (T Value)	Trả về vị trí đầu tiên xuất hiện đối tượng Value trong List . Nếu không tìm thấy sẽ trả về -1.
Insert (int Index, T Value)	Chèn đối tượng Value vào vị trí Index trong List .
InsertRange (int Index, IEnumerable<T> ListObject)	Chèn danh sách phần tử ListObject vào vị trí Index trong List .
LastIndexOf (T Value)	Trả về vị trí xuất hiện cuối cùng của đối tượng Value trong List . Nếu không tìm thấy sẽ trả về -1.
Remove (T Value)	Xoá đối tượng Value xuất hiện đầu tiên trong List .
Reverse ()	Đảo ngược tất cả phần tử trong List .
Sort ()	Sắp xếp các phần tử trong List theo thứ tự tăng dần.
ToArray ()	Trả về 1 mảng kiểu T chứa các phần tử được sao chép từ List .

Sử dụng List hoàn toàn tương tự như sử dụng ArrayList. Một ví dụ đơn giản về sử dụng List:

```
/*
 * Tạo 1 List các kiểu string và thêm 2 phần tử vào List.
 */
```

```

List<string> MyList4 = new List<string>();
MyList4.Add("Free");
MyList4.Add("Education");

// In giá trị các phần tử trong List
Console.WriteLine(" List ban dau: ");
Console.WriteLine(" So luong phan tu trong List la: {0}", MyList4.Count);
foreach (string item in MyList4)
{
    Console.Write(" " + item);
}
Console.WriteLine();

// Chèn 1 phần tử vào đầu List.
MyList4.Insert(0, "HowKteam");

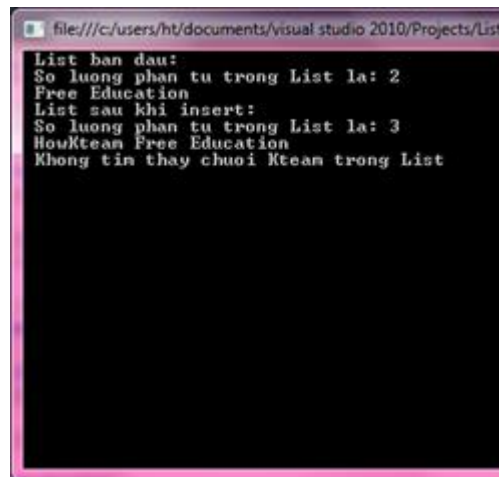
// In lại giá trị các phần tử trong List để xem đã chèn được hay chưa
Console.WriteLine(" List sau khi insert: ");
Console.WriteLine(" So luong phan tu trong List la: {0}", MyList4.Count);
foreach (string item in MyList4)
{
    Console.Write(" " + item);
}
Console.WriteLine();

// Kiểm tra 1 phần tử có tồn tại trong List hay không.
bool isExists = MyList4.Contains("Kteam");

if (isExists == false)
{
    Console.WriteLine(" Không tìm thấy chuỗi Kteam trong List");
}

```

Kết quả khi chạy đoạn chương trình trên là:



```
file:///c:/users/ht/documents/visual studio 2010/Projects/List/
List ban dau:
So luong phan tu trong List la: 2
Free Education
List sau khi insert:
So luong phan tu trong List la: 3
HouKteam Free Education
Khong tin thay chuoai Kteam trong List
```

2.4 Xử lý ngoại lệ

Xử lý ngoại lệ (Try/Catch/Finally) trong C#

Một Exception (ngoại lệ) là một vấn đề xuất hiện trong khi thực thi một chương trình. Một Exception trong C# là một phản hồi về một tình huống ngoại lệ mà xuất hiện trong khi một chương trình đang chạy, ví dụ như chia cho số 0.

Exception cung cấp một cách để truyền điều khiển từ một phần của một chương trình tới phần khác. Exception Handling (Xử lý ngoại lệ) trong C# được xây dựng dựa trên 4 từ khóa là: **try**, **catch**, **finally**, và **throw**.

- **try**: Một khối try nhận diện một khối code mà ở đó các exception cụ thể được kích hoạt. Nó được theo sau bởi một hoặc nhiều khối catch.
- **catch**: Một chương trình bắt một Exception với một Exception Handler tại vị trí trong một chương trình nơi bạn muốn xử lý vấn đề đó. Từ khóa **catch** trong C# chỉ dẫn việc bắt một exception.
- **finally**: Một khối finally được sử dụng để thực thi một tập hợp lệnh đã cho, dù có hay không một exception được ném hoặc không được ném. Ví dụ, nếu bạn mở một file, nó phải được đóng, nếu không sẽ có một exception được tạo ra.
- **throw**: Một chương trình ném một exception khi có một vấn đề xuất hiện. Điều này được thực hiện bởi sử dụng từ khóa **throw** trong C#.

Cú pháp

Giả sử một khối tạo một Exeption, một phương thức bắt một exception bởi sử dụng kết hợp các từ khóa try và catch. Một khối try/catch được đặt xung quanh code mà có thể tạo một exception. Code bên trong một khối try/catch được

xem như là code được bảo vệ, và cú pháp để sử dụng try/catch trong C# như sau:

```
try
{
    // các lệnh có thể gây ra ngoại lệ (exception)
}
catch( tên_ngoại_lệ e1 )
{
    // phần code để xử lý lỗi
}
catch( tên_ngoại_lệ e2 )
{
    // phần code để xử lý lỗi
}
catch( tên_ngoại_lệ eN )
{
    // phần code để xử lý lỗi
}
finally
{
    // các lệnh được thực thi
}
```

Bạn có thể liệt kê nhiều lệnh catch để bắt các kiểu exception khác nhau trong trường hợp khối try của bạn xuất hiện nhiều hơn một exception trong các tình huống khác nhau.

Lớp Exception trong C#

Các Exception trong C# được biểu diễn bởi các lớp. Các lớp Exception trong C# chủ yếu được kế thừa một cách trực tiếp hoặc không trực tiếp từ lớp **System.Exception** trong C#. Một số lớp Exception kế thừa từ lớp

`System.Exception` là các lớp **`System.ApplicationException`** và **`System.SystemException`**.

Lớp **`System.ApplicationException`** hỗ trợ các exception được tạo bởi các chương trình ứng dụng. Vì thế, các exception được định nghĩa bởi lập trình viên nên kế thừa từ lớp này.

Lớp **`System.SystemException`** là lớp cơ sở cho tất cả system exception tiền định nghĩa.

Bảng sau cung cấp một số lớp Exception tiền định nghĩa được kế thừa từ lớp *`System.SystemException`* trong C#:

Lớp Exception	Miêu tả
<code>System.IO.IOException</code>	Xử lý các I/O error
<code>System.IndexOutOfRangeException</code>	Xử lý các lỗi được tạo khi một phương thức tham chiếu tới một chỉ mục bên ngoài dãy mảng
<code>System.ArrayTypeMismatchException</code>	Xử lý các lỗi được tạo khi kiểu là không phù hợp với kiểu mảng
<code>System.NullReferenceException</code>	Xử lý các lỗi được tạo từ việc tham chiếu một đối tượng null
<code>System.DivideByZeroException</code>	Xử lý các lỗi được tạo khi chia cho số 0
<code>System.InvalidCastException</code>	Xử lý lỗi được tạo trong khi ép kiểu
<code>System.OutOfMemoryException</code>	Xử lý lỗi được tạo từ việc thiếu bộ nhớ rồi

System.StackOverflowException	Xử lý lỗi được tạo từ việc tràn ngăn xếp (stack)
-------------------------------	--

Xử lý ngoại lệ (Exception Handling) trong C#

C# cung cấp một giải pháp mang tính cấu trúc cao để xử lý ngoại lệ trong form các khối try và catch. Sử dụng các khối này, các lệnh chương trình được phân biệt riêng rẽ với các lệnh xử lý ngoại lệ trong C#.

Những khối xử lý ngoại lệ này được triển khai bởi sử dụng các từ khóa **try**, **catch** và **finally** trong C#. Ví dụ sau ném một exception khi chia cho số 0.

```
using System;
namespace Ten_namespace_C
{
    class TestCsharp
    {
        int result;
        TestCsharp()
        {
            result = 0;
        }
        public void phepChia(int num1, int num2)
        {
            try
            {
                result = num1 / num2;
            }
            catch (DivideByZeroException e)
            {
                Console.WriteLine("Bat Exception: {0}", e);
            }
        }
    }
}
```



```

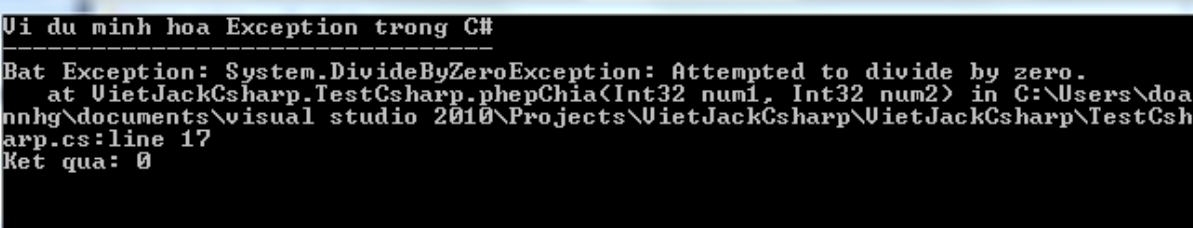
        finally
        {
            Console.WriteLine("Ket qua: {0}", result);
        }
    }
    static void Main(string[] args)
    {
        Console.WriteLine("Vi du minh hoa Exception trong C#");
        Console.WriteLine("-----");

        TestCsharp d = new TestCsharp();
        d.phepChia(25, 0);
        Console.ReadKey();
    }
}

```

Nếu bạn không sử dụng lệnh **Console.ReadKey();** thì chương trình sẽ chạy và kết thúc luôn (nhanh quá đến nỗi bạn không kịp nhìn kết quả). Lệnh này cho phép chúng ta nhìn kết quả một cách rõ ràng hơn.

Biên dịch và chạy chương trình C# trên sẽ cho kết quả sau:



```

Vi du minh hoa Exception trong C#
-----
Bat Exception: System.DivideByZeroException: Attempted to divide by zero.
   at VietJackCsharp.TestCsharp.phepChia(Int32 num1, Int32 num2) in C:\Users\doanhhg\documents\visual studio 2010\Projects\VietJackCsharp\VietJackCsharp\TestCsharp.cs:line 17
Ket qua: 0

```

Tạo User-Defined Exception trong C#

Bạn cũng có thể định nghĩa exception cho riêng bạn. Các lớp User-Defined Exception được kế thừa từ lớp **ApplicationException** trong C#. Ví dụ sau minh họa điều này:

Tạo 3 lớp có tên lần lượt là như sau:

Lớp **Temperature**

```
using System;

namespace Ten_namespace_C
{
    class Temperature
    {
        int temperature = 0;
        public void showTemp()
        {
            if (temperature == 0)
            {
                throw (new TempIsZeroException("Muc nhiet do 0!!!"));
            }
            else
            {
                Console.WriteLine("Nhiet do: {0}", temperature);
            }
        }
    }
}
```

Lớp **TempIsZeroException**: đây là một lớp ngoại lệ tự định nghĩa, kế thừa lớp **ApplicationException**

```
using System;

namespace Ten_namespace_C
{
    class TempIsZeroException : ApplicationException
```

```

{
    public TempIsZeroException(string message)
        : base(message)
    {
    }
}
}

```

Lớp TestCsharp

```

using System;

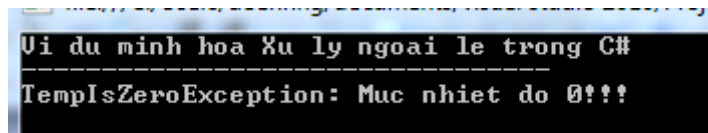
namespace Ten_namespace_C
{
    class TestCsharp
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Vi du minh hoa Xu ly ngoai le trong C#");
            Console.WriteLine("-----");

            Temperature temp = new Temperature();
            try
            {
                temp.showTemp();
            }
            catch (TempIsZeroException e)
            {
                Console.WriteLine("TempIsZeroException: {0}", e.Message);
            }

            Console.ReadKey();
        }
    }
}

```

Biên dịch và chạy chương trình C# trên sẽ cho kết quả sau:



```

Vi du minh hoa Xu ly ngoai le trong C#
-----
TempIsZeroException: Muc nhiet do 0!!!

```

Ném các Object trong C#

Bạn có thể ném một đối tượng nếu nó: hoặc trực tiếp hoặc gián tiếp được kế thừa từ lớp **System.Exception** trong C#. Bạn có thể sử dụng một lệnh throw trong khối catch để ném đối tượng hiện diện đó:

```
Catch(Exception e)
{
    ...
    Throw e
}
```

Chương 3. Lập trình hướng đối tượng trong C#

3.1 Thuộc tính và phương thức

3.1.1. Thuộc tính (Property)

Thuộc tính - Property là các thành viên được đặt tên của các lớp, cấu trúc, và Interface. Các biến thành viên hoặc các phương thức trong một lớp hoặc cấu trúc được gọi là các **Field**. Thuộc tính là một sự kế thừa của các Field và được truy cập bởi sử dụng cùng cú pháp. Chúng sử dụng **accessor** thông qua các giá trị của các Private Field có thể được đọc, được viết và được thao tác.

Thuộc tính (Property) không đặt tên các vị trí lưu giữ. Thay vào đó, chúng có accessors mà đọc, ghi hoặc tính toán các giá trị của chúng.

Ví dụ, chúng ta có một lớp với tên Student, với các Private Field cho age, name, và code. Chúng ta không thể trực tiếp truy cập các Field này từ bên ngoài phạm vi lớp đó, nhưng chúng ta có thể có các thuộc tính để truy cập các Private Field này.

Accessor trong C#

Trong C#, **accessor** là một thuộc tính chứa các lệnh có thể thực thi, mà giúp đỡ trong việc *lấy* (đọc hoặc tính toán) hoặc *thiết lập* (ghi) thuộc tính. Các khai báo accessor có thể thu được một get accessor, một set accessor, hoặc cả hai. Ví dụ:

```
// khai báo một thuộc tính Code có kiểu dữ liệu string:
```

```
public string Code
{
    get
    {
        return code;
    }
    set
    {
        code = value;
    }
}
```

```
// khai báo một thuộc tính Name có kiểu dữ liệu String:
```

```
public string Name
{
    get
    {
        return name;
    }
    set
    {
        name = value;
    }
}
```

```
// khai báo một thuộc tính Age có kiểu dữ liệu int:
```

```
public int Age
{
    get
    {
        return age;
    }
    set
    {
        age = value;
    }
}
```

Ví dụ

Ví dụ dưới đây minh họa cách sử dụng của các thuộc tính trong C#: tạo 2 lớp có tên lần lượt là **Student**, **TestCsharp** như sau:

Lớp **Student**:

```
using System;

namespace Ten_namespace_C
{
    class Student
    {
        private string code = "N/A";
        private string name = "unknown";
        private int age = 0;

        // khai báo thuộc tính Code có kiểu string:
        public string Code
        {
            get
            {
                return code;
            }
            set
            {
                code = value;
            }
        }

        // khai báo thuộc tính Name có kiểu string:
        public string Name
        {
            get
            {
                return name;
            }
            set
            {
                name = value;
            }
        }

        // khai báo thuộc tính Age có kiểu int:
        public int Age
```

```

    {
        get
        {
            return age;
        }
        set
        {
            age = value;
        }
    }
    public override string ToString()
    {
        return "MSSV = " + Code + ", Ho Ten = " + Name + ", Tuoi = " + Age;
    }
}
}

```

Lớp TestCsharp:

```

using System;
using System.Reflection;

namespace Ten_namespace_C
{
    class TestCsharp
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Thuoc tinh (Property) trong C#");
            Console.WriteLine("-----");

            // tao mot doi tuong Student
            Student s = new Student();

            // thiet lap cac thuoc tinh code, name va age cho Student
            s.Code = "001";
            s.Name = "Minh Chinh";
            s.Age = 21;
            Console.WriteLine("Thong tin sinh vien: {0}", s);

            //bay gio tang age them 1
            s.Age += 1;
            Console.WriteLine("Thong tin sinh vien: {0}", s);
        }
    }
}

```

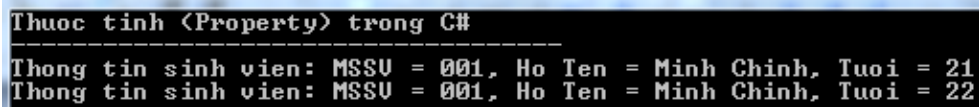
```

        Console.ReadLine();
        Console.ReadKey();
    }
}

```

Nếu bạn không sử dụng lệnh **Console.ReadKey();** thì chương trình sẽ chạy và kết thúc luôn (nhanh quá đến nỗi bạn không kịp nhìn kết quả). Lệnh này cho phép chúng ta nhìn kết quả một cách rõ ràng hơn.

Biên dịch và chạy chương trình C# trên sẽ cho kết quả sau:



```

Thuộc tính <Property> trong C#
-----
Thông tin sinh viên: MSSV = 001, Họ Tên = Minh Chính, Tuổi = 21
Thông tin sinh viên: MSSV = 001, Họ Tên = Minh Chính, Tuổi = 22

```

Thuộc tính abstract trong C#

Một lớp Abstract có thể có một thuộc tính abstract, mà nên được triển khai trong lớp kế thừa. Chương trình sau minh họa điều này:

Tạo 3 lớp có tên lần lượt là **Person**, **Student**, **TestCsharp** như sau:

Lớp abstract **Person**:

```

using System;

namespace Ten_namespace_C
{
    public abstract class Person
    {
        public abstract string Name
        {
            get;
            set;
        }
        public abstract int Age
        {
            get;
            set;
        }
    }
}

```

Lớp **Student**:

```

using System;

```



```
namespace Ten_namespace_C
{
    class Student : Person
    {
        private string code = "N/A";
        private string name = "N/A";
        private int age = 0;

        // khai bao thuoc tinh Code co kieu string:
        public string Code
        {
            get
            {
                return code;
            }
            set
            {
                code = value;
            }
        }

        // khai bao thuoc tinh Name co kieu string:
        public override string Name
        {
            get
            {
                return name;
            }
            set
            {
                name = value;
            }
        }

        // khai bao thuoc tinh Age co kieu int:
        public override int Age
        {
            get
            {
                return age;
            }
            set
```

```

        {
            age = value;
        }
    }
    public override string ToString()
    {
        return "MSSV = " + Code + ", Ho ten = " + Name + ", Tuoi = " + Age;
    }
}
}

```

Lớp TestCsharp

```

using System;
using System.Reflection;

namespace Ten_namespace_C
{
    class TestCsharp
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Thuoc tinh (Property) trong C#");
            Console.WriteLine("-----");

            // tao mot doi tuong Student
            Student s = new Student();

            // thiet lap code, name va age cho Student
            s.Code = "001";
            s.Name = "Minh Chinh";
            s.Age = 21;
            Console.WriteLine("Thong tin sinh vien: {0}", s);

            //bay gio tang age them 1
            s.Age += 1;
            Console.WriteLine("Thong tin sinh vien: {0}", s);

            Console.ReadLine();
            Console.ReadKey();
        }
    }
}

```

Biên dịch và chạy chương trình C# trên sẽ cho kết quả sau:

```
Thuộc tính <Property> trong C#
-----
Thông tin sinh viên: MSSV = 001, Họ Tên = Minh Chính, Tuổi = 21
Thông tin sinh viên: MSSV = 001, Họ Tên = Minh Chính, Tuổi = 22
```

3.1.2. Phương thức

Một phương thức là một nhóm lệnh cùng nhau thực hiện một tác vụ. Mỗi chương trình C# có ít nhất một lớp với một phương thức là Main.

Để sử dụng một phương thức trong C#, bạn cần:

- Định nghĩa phương thức
- Gọi phương thức

Định nghĩa phương thức trong C#

Khi bạn định nghĩa một phương thức, về cơ bản, bạn khai báo các phần tử của cấu trúc của nó. Cú pháp để định nghĩa một phương thức trong C# là như sau:

```
<Access Specifier> <Kiểu_trả_về> <tên_phương_thức>(<danh_sách_tham_số>
{
    phần_thân_phương_thức
}
```

Dưới đây là chi tiết về các phần tử trong một phương thức:

- **Access Specifier:** Định nghĩa tính nhìn thấy của một biến hoặc một phương thức với lớp khác.
- **Kiểu_trả_về:** Một phương thức có thể trả về một giá trị. Kiểu trả về là kiểu dữ liệu của giá trị mà phương thức trả về. Nếu phương thức không trả về bất kỳ giá trị nào, thì kiểu trả về là **void**.
- **tên_phương_thức:** Tên phương thức là một định danh duy nhất và nó là phân biệt kiểu chữ. Nó không thể giống bất kỳ định danh nào khác đã được khai báo trong lớp đó.
- **danh_sách_tham_số:** Danh sách tham số được bao quanh trong dấu ngoặc đơn, các tham số này được sử dụng để truyền và nhận dữ liệu từ một phương thức. Danh sách tham số liên quan tới kiểu, thứ tự, và số tham số của một phương thức. Các tham số là tùy ý, tức là một phương thức có thể không chứa tham số nào.
- **phần_thân_phương_thức:** Phần thân phương thức chứa tập hợp các chỉ thị cần thiết để hoàn thành hoạt động đã yêu cầu.

Ví dụ

Chương trình sau minh họa một hàm *FindMax* nhận hai giá trị integer và trả về số nào lớn hơn trong hai số. Nó có Access Specifier, vì thế nó có thể được truy cập từ bên ngoài lớp bởi sử dụng một Instance (sự thể hiện) của lớp đó.

```
using System;

namespace Ten_namespace_C
{
    class TestCsharp
    {
        public int FindMax(int num1, int num2)
        {
            /* khai bao bien cuc bo */
            int result;

            if (num1 > num2)
                result = num1;
            else
                result = num2;

            return result;
        }
        ...
    }
}
```

Gọi phương thức trong C#

Bạn có thể gọi một phương thức bởi sử dụng tên của phương thức đó. Ví dụ sau minh họa cách gọi phương thức trong C#:

```
using System;

namespace Ten_namespace_C
{
    class TestCsharp
    {
        public int FindMax(int num1, int num2)
        {
            /* khai bao bien cuc bo */
            int result;

            if (num1 > num2)
```

```

        result = num1;
    else
        result = num2;
    return result;
}
static void Main(string[] args)
{
    Console.WriteLine("Goi phuong thuc trong C#");
    Console.WriteLine("-----");
    /* phan dinh nghia bien cuc bo */
    int a = 100;
    int b = 200;
    int ret;
    TestCsharp n = new TestCsharp();

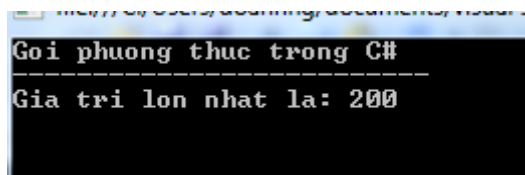
    //goi phuong thuc FindMax
    ret = n.FindMax(a, b);
    Console.WriteLine("Gia tri lon nhat la: {0}", ret);
    Console.ReadLine();

    Console.ReadKey();
}
}
}

```

Nếu bạn không sử dụng lệnh **Console.ReadKey();** thì chương trình sẽ chạy và kết thúc luôn (nhANH quá đến nỗi bạn không kịp nhìn kết quả). Lệnh này cho phép chúng ta nhìn kết quả một cách rõ ràng hơn.

Biên dịch và chạy chương trình C# trên sẽ cho kết quả sau:



```

Goi phuong thuc trong C#
-----
Gia tri lon nhat la: 200

```

Bạn cũng có thể gọi phương thức public từ các lớp khác bằng việc sử dụng Instance (sự thể hiện) của lớp đó. Ví dụ, phương thức **FindMax** thuộc lớp *UngDungToan*, bạn có thể gọi nó từ lớp *TestCsharp*.

Tạo hai lớp có tên lần lượt là **UngDungToan** và **TestCsharp** có nội dung như sau:

Lớp **UngDungToan**: chứa phương thức cần gọi

```
using System;
```

```

namespace Ten_namespace_C
{
    class UngDungToan
    {
        public int FindMax(int num1, int num2)
        {
            /* khai bao bien cuc bo */
            int result;

            if (num1 > num2)
                result = num1;
            else
                result = num2;

            return result;
        }
    }
}

```

Lớp **TestCsharp**: chứa phương thức **main()**.

```

using System;

namespace Ten_namespace_C
{
    class TestCsharp
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Goi phuong thuc trong C#");
            Console.WriteLine("Goi phuong thuc thong qua instance cua lop");
            Console.WriteLine("-----");
            /* phan dinh nghia bien cuc bo */
            int a = 100;
            int b = 200;
            int ret;

            //tao doi tuong UngDungToan
            UngDungToan n = new UngDungToan();

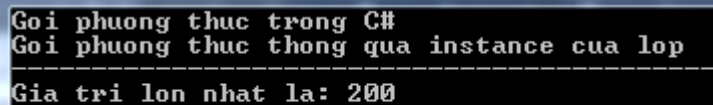
            //goi phuong thuc FindMax
            ret = n.FindMax(a, b);
            Console.WriteLine("Gia tri lon nhat la: {0}", ret);
            Console.ReadLine();
        }
    }
}

```

```
}  
}  
}
```

Nếu bạn không sử dụng lệnh **Console.ReadKey();** thì chương trình sẽ chạy và kết thúc luôn (nhANH quá đến nỗi bạn không kịp nhìn kết quả). Lệnh này cho phép chúng ta nhìn kết quả một cách rõ ràng hơn.

Biên dịch và chạy chương trình C# trên sẽ cho kết quả sau:



```
Goi phuong thuc trong C#  
Goi phuong thuc thong qua instance cua lop  
-----  
Gia tri lon nhat la: 200
```

Gọi phương thức đệ qui trong C#

Một phương thức có thể gọi chính nó. Điều này được biết đến là **đệ qui**. Ví dụ sau tính toán giai thừa của số đã cho bởi sử dụng một hàm đệ qui trong C#:

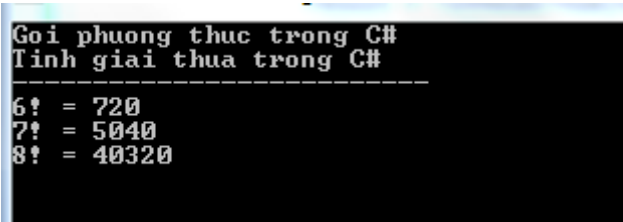
```
using System;  
  
namespace Ten_namespace_C  
{  
    class TestCsharp  
    {  
        public int TinhGiaiThua(int num)  
        {  
            /* khai bao bien cuc bo */  
            int result;  
            if (num == 1)  
            {  
                return 1;  
            }  
            else  
            {  
                result = TinhGiaiThua(num - 1) * num;  
                return result;  
            }  
        }  
    }  
  
    static void Main(string[] args)  
    {  
        Console.WriteLine("Goi phuong thuc trong C#");  
        Console.WriteLine("Tinh giai thua trong C#");  
    }  
}
```

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

TestCsharp n = new TestCsharp();
//goi phuong thuc
Console.WriteLine("6! = {0}", n.TinhGiaiThua(6));
Console.WriteLine("7! = {0}", n.TinhGiaiThua(7));
Console.WriteLine("8! = {0}", n.TinhGiaiThua(8));
Console.ReadLine();

Console.ReadKey();
}
}
}
```

Nếu bạn không sử dụng lệnh **Console.ReadKey();** thì chương trình sẽ chạy và kết thúc luôn (nhanh quá đến nỗi bạn không kịp nhìn kết quả). Lệnh này cho phép chúng ta nhìn kết quả một cách rõ ràng hơn.
Biên dịch và chạy chương trình C# trên sẽ cho kết quả sau:



Truyền tham số cho phương thức trong C#

Khi phương thức với các tham số được gọi, bạn cần truyền các tham số cho phương thức đó. Có 3 cách mà tham số có thể được truyền tới một phương thức trong C#:

Kỹ thuật	Miêu tả
<u>Truyền tham số bởi giá trị trong C#</u>	Phương thức này sao chép giá trị thực sự của một tham số vào trong tham số chính thức của hàm đó. Trong trường hợp này, các thay đổi được tạo ra với tham số chính thức bên trong hàm này sẽ không ảnh hưởng tới tham số đó
<u>Truyền tham số bởi tham chiếu trong C#</u>	Phương thức này sao chép tham chiếu tới vị trí bộ nhớ của một tham số vào trong tham số chính thức. Nghĩa là các thay đổi được tạo ra tới tham số chính thức ảnh hưởng tới tham số đó

**Truyền tham số
bởi output trong
C#**

Phương thức này giúp ích khi trả về nhiều hơn một giá trị

3.2 Constructor và Destructor

3.2.1. Constructor

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

Ví dụ:

```
class Library
{
    private int ibooktypes;
    //Constructor
    public Library() // Hàm tạo không đối số.
    {
        ibooktypes = 7;
    }
    public Library(int value)
    {
        ibooktypes = value;
    }
}
```

Ví dụ: Với dụ trên thì bạn có hai cách khởi tạo lớp Library thông qua một trong hai hàm tạo

- `Library lb1 = new Library();` // Hàm tạo không đối số;
- `Library lb2 = new Library(100);` // Hàm tạo được truyền vào với giá trị là 100, và nó sẽ được gán vào `ibooktypes = 100;`

Instance constructor

Một instance constructor(tạm dịch là một bộ khởi dựng thể hiện) sẽ khởi tạo một số giá trị khi một thể hiện của một lớp được tạo ra.

Ví dụ:

```
class Point
{
    public double x, y;
    public Point()
    {
        this.x = 0;
        this.y = 0;
    }
    public Point(double x, double y)
    {
        this.x = x;
        this.y = y;
    }
    public static double Distance(Point a, Point b)
    {
        double xdiff = a.x - b.x;
        double ydiff = a.y - b.y;
        return Math.Sqrt(xdiff * xdiff + ydiff * ydiff);
    }
    public override string ToString()
    {
        return string.Format("{0}, {1}", x, y);
    }
}
class Test
{
    static void Main()
    {
        Point a = new Point();
    }
}
```

```

        Point b = new Point(3, 4);
        double d = Point.Distance(a, b);
        Console.WriteLine("Distance from {0} to {1} is {2}", a, b,
d);
    }
}

```

Trong lớp Point có 2 instance constructors. Một không có đối số truyền vào và 2 constructor còn lại có 2 tham số kiểu double. Nếu class không có instance constructor nào thì constructor không có đối số sẽ được gọi tự động.

Static constructor

Nếu một lớp khai báo một phương thức khởi tạo tĩnh (static constructor), thì được đảm bảo rằng phương thức này sẽ được thực hiện trước bất cứ thể hiện nào của lớp được tạo ra. Static Constructor hữu dụng khi chúng ta cần cài đặt một số công việc mà không thể thực hiện được thông qua chức năng khởi dựng và công việc cài đặt này chỉ được thực duy nhất một lần. Static constructor không có thuộc tính truy cập, không có đối số và không được gọi tường minh mà sẽ được gọi tự động.

Ví dụ:

```

sing System;
using Personnel.Data;
class Employee
{
    private static DataSet ds;
    static Employee() {
        ds = new DataSet(...);
    }
    public string Name;
    public decimal Salary;
}

```

```
...  
}
```

Khi đó đối tượng ds sẽ được tạo ra khi trước khi ta tạo một thể hiện lớp employee.

3.2.2. Destructor

Chúng ta đã biết Destructor dùng để hủy một lớp. Khi chúng ta sử dụng destructor trong C# ta phải giữ lại các nội dung sau:

- Một lớp chỉ có 1 destructor.
- Destructor không thể kế thừa hoặc là phương thức chồng
- Destructor không cần gọi mà chúng được gọi tự động
- Destructor không có tham số

Ví dụ destructor cho lớp MyClass

```
MyClass()
```

```
{  
    // Cleaning up  
}
```

Khi destructor được thực thi chúng hủy tất cả các biến, tham số. Chúng sẽ kiểm tra tất cả các đối tượng được sử dụng trong ứng dụng, xem xét những đối tượng có thể để hủy đi trên bộ nhớ chính. Khi một destructor được gọi thì tất cả phương thức trên lớp cơ sở sẽ bị hủy theo. Vì vậy trước đó destructor hoàn toàn được chuyển thành:

```
protected override void Finalize()
```

```
{  
    try  
    {  
        // Cleaning up .  
    }  
    finally  
    {  
        base.Finalize();  
    }  
}
```

Chúng ta sẽ xem xét ví dụ về cách thức gọi một destructor như thế nào. Chúng ta có 3 lớp A, B, C. B kế thừa từ A, và C kế thừa từ B, mỗi lớp đều có riêng phương thức constructor và destructor. Trong lớp program chúng ta tạo lớp C.

```
class A
{
    public A()
    {
        Console.WriteLine("Creating A");
    }
    ~A()
    {
        Console.WriteLine("Destroying A");
    }
}
```

```
class B : A
{
    public B()
    {
        Console.WriteLine("Creating B");
    }
    ~B()
    {
        Console.WriteLine("Destroying B");
    }
}
```

```
class C : B
{
    public C()
    {
        Console.WriteLine("Creating C");
    }
    ~C()
    {
        Console.WriteLine("Destroying C");
    }
}
```

```

class Program
{
    public static void Main()
    {
        C c = new C();
        Console.WriteLine("Object Created ");
        Console.WriteLine("Press enter to Destroy it");
        Console.ReadLine();
        c = null;
        Console.Read();
    }
}

```

Chương trình sẽ đợi cho đến khi người dùng nhấn ‘enter’. Khi đó chúng ta sẽ gán lớp C là null. Nhưng như vậy thì destructor của lớp C không được thực hiện. Như chúng ta đã nói, chương trình sẽ không được ‘điều khiển’ khi destructor được gọi bởi chúng sẽ thu dọn tất cả các biến, tham số... Nhưng destructor được gọi khi chương trình kết thúc. Bạn có thể kiểm tra bằng cách gọi một lần nữa và xuất ra file hay màn hình chúng ta sẽ có kết quả:

```

Creating A
Creating B
Creating C
Object Created
Press enter to Destroy it
Destroying C
Destroying B
Destroying A

```

Chú ý rằng destructor của các lớp cơ sở cũng được gọi bởi chúng được gọi thông qua

```
base.Finalize()
```

Vì vậy chúng ta có thể sử dụng destructor để kết thúc việc sử dụng một đối tượng nào đó.

3.3 Nạp chồng hàm và nạp chồng toán tử

3.3.1. Nạp chồng hàm

C# hỗ trợ phương thức nạp chồng với một vài dạng phương thức khác nhau về những đặc tính sau: tên, số lượng thông số, và kiểu thông số. Nhưng nó không hỗ trợ những thông số mặc định như C++ và VB. Một cách đơn giản là bạn khai báo những phương thức cùng tên nhưng khác số lượng và kiểu của thông số. Method Overloading xuất hiện khi trong một class có từ hai hàm có cùng tên. Có hai kiểu Method Overloading:

- Function Overloading dựa trên kiểu giá trị tham số truyền vào.
- Function Overloading dựa trên số lượng tham số truyền vào.

Ví dụ

```
class Library
{
// Function Overloading
    public void insertbooks(int id)
    {
        //
    }
    public void insertbooks(int id, int type)
    {
        //
    }
    public void insertbooks(string id, int type)
    {
        //
    }
}
```

Ba hàm insertbooks ở trên là một ví dụ về function overloading trong lập trình C#. Trong khi hàm thứ nhất và thứ 2 là overloading theo số lượng tham số, và hàm thứ 3 với hàm thứ 2 là overloading theo kiểu tham số truyền vào.

- Bởi vì C# không hỗ trợ những thông số tùy chọn nên bạn cần sử dụng những phương thức nạp chồng để đạt được cùng một hiệu quả:

```
class MyClass
{
    int DoSomething(int x) // Gọi hàm 2 tham số với một giá trị mặc định là 10;
    {
        DoSomething(x, 10);
    }
}
```

```
int DoSomething(int x, int y)
{
    // Làm một điều gì đó ở đây :D
}
}
```

- Trong bất kỳ một ngôn ngữ nào, phương thức nạp chồng có thể đem đến một lỗi nghiêm trọng nếu nó bị gọi sai. Trong chương tới ta sẽ bàn cách để tránh điều đó. Trong C# có một vài điểm khác nhỏ về thông số trong các phương thức nạp chồng cần biết như sau:

- Nó không chấp nhận hai phương thức chỉ khác nhau về kiểu trả về.
- Nó không chấp nhận hai phương thức chỉ khác nhau về đặc tính của một thông số đang được khai báo như ref hay out.

3.3.2. Nạp chồng toán tử

Operator Overloading là Nạp chồng toán tử. Bạn có thể tái định nghĩa hoặc nạp chồng hầu hết các toán tử có sẵn trong C#. Vì thế, một lập trình viên có thể sử dụng các toán tử với các kiểu tự định nghĩa (user-defined). Các toán tử được nạp chồng trong C# là các hàm với các tên đặc biệt: từ khóa operator được theo sau bởi biểu tượng cho toán tử đang được định nghĩa. Tương tự như bất kỳ hàm nào khác, một toán tử được nạp chồng có một kiểu trả về và một danh sách tham số.

Ví dụ, bạn xét hàm sau:

```
public static Box operator+ (Box b, Box c)
{
    Box box = new Box();
    box.chieu_dai = b.chieu_dai + c.chieu_dai;
    box.chieu_rong = b.chieu_rong + c.chieu_rong;
    box.chieu_cao = b.chieu_cao + c.chieu_cao;
    return box;
}
```

Hàm trên triển khai toán tử cộng (+) cho một lớp Box tự định nghĩa (user-defined). Nó cộng các thuộc tính của hai đối tượng Box và trả về đối tượng kết quả Box.

Triển khai Nạp chồng toán tử trong C#

Ví dụ dưới đây minh họa cách triển khai nạp chồng toán tử trong C#: tạo hai lớp có tên lần lượt là **Box**, **TestCsharp** như sau:

Lớp **Box**: chứa các thuộc tính và phương thức


```

using System;

namespace Ten_namespace_C
{
    class Box
    {
        private double chieu_dai;
        private double chieu_rong;
        private double chieu_cao;

        public double tinhTheTich()
        {
            return chieu_dai * chieu_rong * chieu_cao;
        }

        public void setChieuDai(double len)
        {
            chieu_dai = len;
        }

        public void setChieuRong(double bre)
        {
            chieu_rong = bre;
        }

        public void setChieuCao(double hei)
        {
            chieu_cao = hei;
        }

        // nap chong toan tu + de cong hai doi tuong Box.
        public static Box operator +(Box b, Box c)
        {
            Box box = new Box();
            box.chieu_dai = b.chieu_dai + c.chieu_dai;
            box.chieu_rong = b.chieu_rong + c.chieu_rong;
            box.chieu_cao = b.chieu_cao + c.chieu_cao;
            return box;
        }
    }
}

```

Lớp **TestCsharp**: chứa phương thức main() để thao tác trên đối tượng Box

```
using System;
namespace Ten_namespace_C
{
    public class TestCsharp
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Nap chong toan tu trong C#");
            Console.WriteLine("Vi du minh hoa nap chong toan tu");
            Console.WriteLine("-----");
            //tao cac doi tuong Box1, Box2 va Box3
            Box Box1 = new Box();
            Box Box2 = new Box();
            Box Box3 = new Box();
            double the_tich = 0.0;

            // nhap thong tin Box1
            Box1.setChieuDai(6.0);
            Box1.setChieuRong(7.0);
            Box1.setChieuCao(5.0);

            // nhap thong tin Box2
            Box2.setChieuDai(12.0);
            Box2.setChieuRong(13.0);
            Box2.setChieuCao(10.0);

            // tinh va hien thi the tich Box1
            the_tich = Box1.tinhTheTich();
            Console.WriteLine("The tich cua Box1 la: {0}", the_tich);

            // tinh va hien thi the tich Box2
            the_tich = Box2.tinhTheTich();
            Console.WriteLine("The tich cua Box2 la: {0}", the_tich);

            // con hai doi tuong
            Box3 = Box1 + Box2;

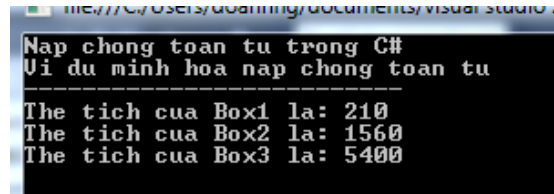
            // tinh va hien thi the tich Box3
            the_tich = Box3.tinhTheTich();
            Console.WriteLine("The tich cua Box3 la: {0}", the_tich);

            Console.ReadKey();
        }
    }
}
```

```
}  
}
```

Nếu bạn không sử dụng lệnh **Console.ReadKey()**; thì chương trình sẽ chạy và kết thúc luôn (nhanch quá đến nỗi bạn không kịp nhìn kết quả). Lệnh này cho phép chúng ta nhìn kết quả một cách rõ ràng hơn.

Biên dịch và chạy chương trình C# trên sẽ cho kết quả sau:



```
File:///C:/Users/Quangming/Documents/visual studio 2...  
Nạp chồng toán tử trong C#  
Ví dụ minh họa nạp chồng toán tử  
-----  
The tích của Box1 là: 210  
The tích của Box2 là: 1560  
The tích của Box3 là: 5400
```

Toán tử có thể nạp chồng và không thể nạp chồng trong C#

Bảng dưới miêu tả các toán tử có thể nạp chồng trong C#:

Toán tử	Miêu tả
+, -, !, ~, ++, --	Những toán tử một ngôi này nhận một toán hạng và có thể được nạp chồng
+, -, *, /, %	Những toán tử nhị phân này nhận một toán hạng và có thể được nạp chồng
=, !=, <, >, <=, >=	Các toán tử so sánh có thể được nạp chồng
&&,	Các toán tử logic điều kiện không thể được nạp chồng một cách trực tiếp
+=, -=, *=, /=, %=	Các toán tử gán không thể được nạp chồng
=, ., ?:, ->, new, is, sizeof, typeof	Các toán tử này không thể được nạp chồng

Ví dụ

Từ các khái niệm trên, chúng ta kế thừa ví dụ trên và nạp chồng thêm một số toán tử trong C#: ở trên chúng ta đã tạo hai lớp **Box**, **TestCsharp**, bây giờ chúng ta sửa lại code của mỗi lớp như sau:

Lớp **Box**: chứa các thuộc tính và phương thức

```
using System;
```

```

namespace Ten_namespace_C
{
    class Box
    {
        private double chieu_dai;
        private double chieu_rong;
        private double chieu_cao;

        public double tinhTheTich()
        {
            return chieu_dai * chieu_rong * chieu_cao;
        }

        public void setChieuDai(double len)
        {
            chieu_dai = len;
        }

        public void setChieuRong(double bre)
        {
            chieu_rong = bre;
        }

        public void setChieuCao(double hei)
        {
            chieu_cao = hei;
        }

        // nap chong toan tu +
        public static Box operator +(Box b, Box c)
        {
            Box box = new Box();
            box.chieu_dai = b.chieu_dai + c.chieu_dai;
            box.chieu_rong = b.chieu_rong + c.chieu_rong;
            box.chieu_cao = b.chieu_cao + c.chieu_cao;
            return box;
        }
        //nap chong toan tu ==
        public static bool operator ==(Box lhs, Box rhs)
        {
            bool status = false;

```

```

        if (lhs.chieu_dai == rhs.chieu_dai && lhs.chieu_cao == rhs.chieu_cao &
& lhs.chieu_rong == rhs.chieu_rong)
        {
            status = true;
        }
        return status;
    }
//nap chong toan tu !=
public static bool operator !=(Box lhs, Box rhs)
{
    bool status = false;
    if (lhs.chieu_dai != rhs.chieu_dai || lhs.chieu_cao != rhs.chieu_cao || lhs.
chieu_rong != rhs.chieu_rong)
    {
        status = true;
    }
    return status;
}
//nap chong toan tu <
public static bool operator <(Box lhs, Box rhs)
{
    bool status = false;
    if (lhs.chieu_dai < rhs.chieu_dai && lhs.chieu_cao < rhs.chieu_cao &&
lhs.chieu_rong < rhs.chieu_rong)
    {
        status = true;
    }
    return status;
}
//nap chong toan tu >
public static bool operator >(Box lhs, Box rhs)
{
    bool status = false;
    if (lhs.chieu_dai > rhs.chieu_dai && lhs.chieu_cao > rhs.chieu_cao &&
lhs.chieu_rong > rhs.chieu_rong)
    {
        status = true;
    }
    return status;
}
//nap chong toan tu <=
public static bool operator <=(Box lhs, Box rhs)
{

```

```

        bool status = false;
        if (lhs.chieu_dai <= rhs.chieu_dai && lhs.chieu_cao <= rhs.chieu_cao &
& lhs.chieu_rong <= rhs.chieu_rong)
        {
            status = true;
        }
        return status;
    }
    //nap chong toan tu >=
    public static bool operator >=(Box lhs, Box rhs)
    {
        bool status = false;
        if (lhs.chieu_dai >= rhs.chieu_dai && lhs.chieu_cao >= rhs.chieu_cao &
& lhs.chieu_rong >= rhs.chieu_rong)
        {
            status = true;
        }
        return status;
    }
    //nap chong phuong thuc ToString()
    public override string ToString()
    {
        return String.Format("{0}, {1}, {2})", chieu_dai, chieu_rong, chieu_ca
o);
    }
}
}

```

Lớp **TestCsharp**: chứa phương thức main() để thao tác trên đối tượng Box

```

using System;
namespace Ten_namespace_C
{
    public class TestCsharp
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Nap chong toan tu trong C#");
            Console.WriteLine("Vi du minh hoa nap chong toan tu");
            Console.WriteLine("-----");
            //tao cac doi tuong Box1, Box2, Box2, Box4
            Box Box1 = new Box();
            Box Box2 = new Box();
            Box Box3 = new Box();

```

```

Box Box4 = new Box();
double the_tich = 0.0;

// nhap thong tin Box1
Box1.setChieuDai(6.0);
Box1.setChieuRong(7.0);
Box1.setChieuCao(5.0);

// nhap thong tin Box2
Box2.setChieuDai(12.0);
Box2.setChieuRong(13.0);
Box2.setChieuCao(10.0);

//hien thi thong tin cac Box boi su dung phuong thuc nap chong ToString()
g():
Console.WriteLine("Thong tin Box 1: {0}", Box1.ToString());
Console.WriteLine("Thong tin Box 2: {0}", Box2.ToString());

// tinh the tich Box1
the_tich = Box1.tinhTheTich();
Console.WriteLine("The tich cua Box1 la: {0}", the_tich);

// tinh the tich Box2
the_tich = Box2.tinhTheTich();
Console.WriteLine("The tich cua Box2 la: {0}", the_tich);

// cong hai doi tuong
Box3 = Box1 + Box2;
Console.WriteLine("Thong tin Box 3: {0}", Box3.ToString());

// tinh the tich cua Box3
the_tich = Box3.tinhTheTich();
Console.WriteLine("The tich cua Box3 la: {0}", the_tich);

//so sanh cac Box
if (Box1 > Box2)
    Console.WriteLine("Box1 la lon hon Box2");
else
    Console.WriteLine("Box1 la khong lon hon Box2");

if (Box1 < Box2)
    Console.WriteLine("Box1 la nho hon Box2");
else

```

```

        Console.WriteLine("Box1 la khong nho hon Box2");

    if (Box1 >= Box2)
        Console.WriteLine("Box1 la lon hon hoac bang Box2");
    else
        Console.WriteLine("Box1 la khong lon hon hoac bang Box2");

    if (Box1 <= Box2)
        Console.WriteLine("Box1 la nho hon hoac bang Box2");
    else
        Console.WriteLine("Box1 la khong nho hon hoac bang Box2");

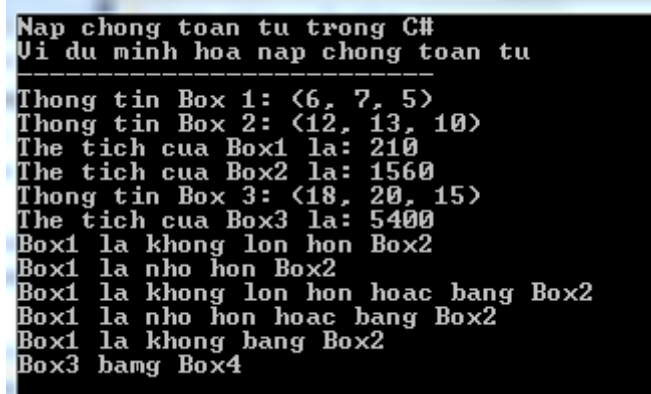
    if (Box1 != Box2)
        Console.WriteLine("Box1 la khong bang Box2");
    else
        Console.WriteLine("Box1 bang Box2");
    Box4 = Box3;

    if (Box3 == Box4)
        Console.WriteLine("Box3 bang Box4");
    else
        Console.WriteLine("Box3 la khong bang Box4");

    Console.ReadKey();
}
}
}

```

Biên dịch và chạy chương trình C# trên sẽ cho kết quả sau:



```

Nap chong toan tu trong C#
Vi du minh hoa nap chong toan tu
-----
Thong tin Box 1: <6, 7, 5>
Thong tin Box 2: <12, 13, 10>
The tich cua Box1 la: 210
The tich cua Box2 la: 1560
Thong tin Box 3: <18, 20, 15>
The tich cua Box3 la: 5400
Box1 la khong lon hon Box2
Box1 la nho hon Box2
Box1 la khong lon hon hoac bang Box2
Box1 la nho hon hoac bang Box2
Box1 la khong bang Box2
Box3 bang Box4

```

3.4 Kết thừa (Interface)

Một trong những khái niệm quan trọng nhất trong lập trình hướng đối tượng là **Tính kế thừa (Inheritance)**. Tính kế thừa cho phép chúng ta định nghĩa một

lớp trong điều kiện một lớp khác, mà làm cho nó dễ dàng hơn để tạo và duy trì một ứng dụng. Điều này cũng cung cấp một cơ hội để tái sử dụng tính năng code và thời gian thực thi nhanh hơn.

Khi tạo một lớp, thay vì viết toàn bộ các thành viên dữ liệu và các hàm thành viên mới, lập trình viên có thể nên kế thừa các thành viên của một lớp đang tồn tại. Lớp đang tồn tại này được gọi là **Base Class - lớp cơ sở**, và lớp mới được xem như là **Derived Class – lớp thừa kế**.

Ý tưởng của tính kế thừa triển khai mối quan hệ **IS-A** (Là Một). Ví dụ, mammal **IS A** animal, dog **IS-A** mammal, vì thế dog **IS-A** animal, và

Lớp cơ sở (Base Class) và Lớp thừa kế (Derived Class) trong C#

Một lớp có thể được kế thừa từ hơn một lớp khác, nghĩa là, nó có thể kế thừa dữ liệu và hàm từ nhiều Lớp hoặc Interface cơ sở.

Cú pháp để tạo lớp kế thừa trong C# là:

```
<access-specifier> class <base_class>
{
    ...
}
class <derived_class> : <base_class>
{
    ...
}
```

Xét một lớp cơ sở Shape và lớp kế thừa Rectangle sau: tạo 3 lớp có tên lần lượt là **Shape**, **HìnhChuNhat**, **TestCsharp** trong đó:

Lớp **Shape** là lớp cơ sở

```
using System;

namespace Ten_namespace_C
{
    class Shape
    {
        protected int chieu_rong;
        protected int chieu_cao;
        public void setChieuRong(int w)
        {
            chieu_rong = w;
        }
        public void setChieuCao(int h)
        {
```

```
        chieu_cao = h;
    }
}
}
```

Lớp **HinhChuNhat** là lớp kế thừa

```
using System;

namespace Ten_namespace_C
{
    class HinhChuNhat : Shape
    {
        public int tinhDienTich()
        {
            return (chieu_cao * chieu_rong);
        }
    }
}
```

Lớp **TestCsharp** chứa phương thức **main()** để thao tác trên đối tượng **HinhChuNhat**

```
using System;
namespace Ten_namespace_C
{
    public class TestCsharp
    {
        public static void Main(string[] args)
        {
            Console.WriteLine("Tinh ke thua trong C#");
            Console.WriteLine("-----\n");

            //tao doi tuong HinhChuNhat
            HinhChuNhat hcn = new HinhChuNhat();

            hcn.setChieuRong(5);
            hcn.setChieuCao(7);

            // in dien tich cua doi tuong.
            Console.WriteLine("Dien tich hinh chu nhat: {0}", hcn.tinhDienTich());

            Console.ReadKey();
        }
    }
}
```

```
}
```

Nếu bạn không sử dụng lệnh **Console.ReadKey();** thì chương trình sẽ chạy và kết thúc luôn (nhANH quá đến nỗi bạn không kịp nhìn kết quả). Lệnh này cho phép chúng ta nhìn kết quả một cách rõ ràng hơn.

Biên dịch và chạy chương trình C# trên sẽ cho kết quả sau:

Khởi tạo Lớp cơ sở (Base Class) trong C#

Lớp kế thừa (Derived Class) trong C# kế thừa các biến thành viên và các phương thức thành viên từ lớp cơ sở. Vì thế, đối tượng của lớp cha nên được tạo trước khi lớp phụ được tạo. Bạn có thể cung cấp các chỉ thị để khởi tạo lớp phụ trong danh sách khởi tạo thành viên.

Chương trình ví dụ sau minh họa cách khởi tạo Lớp cơ sở (Base Class) trong C#: tạo 3 lớp có tên lần lượt là **HinhChuNhat**, **ChiPhiXayDung**, **TestCsharp** như sau:

Lớp **HinhChuNhat** là lớp cơ sở

```
using System;

namespace Ten_namespace_C
{
    class HinhChuNhat
    {
        //cac bien thanh vien
        protected double chieu_dai;
        protected double chieu_rong;
        // constructor
        public HinhChuNhat(double l, double w)
        {
            chieu_dai = l;
            chieu_rong = w;
        }
        //phuong thuc
        public double tinhDienTich()
        {
            return chieu_dai * chieu_rong;
        }

        public void Display()
        {
            Console.WriteLine("Chieu dai: {0}", chieu_dai);
        }
    }
}
```

```

        Console.WriteLine("Chieu rong: {0}", chieu_rong);
        Console.WriteLine("Dien tich: {0}", tinhDienTich());
    }
}

```

Lớp **ChiPhiXayDung** kế thừa lớp **HinhChuNhat**

```

using System;

namespace Ten_namespace_C
{
    class ChiPhiXayDung : HinhChuNhat
    {
        private double cost;
        public ChiPhiXayDung(double l, double w) : base(l, w)
        { }
        public double tinhChiPhi()
        {
            double chi_phi;
            chi_phi = tinhDienTich() * 70;
            return chi_phi;
        }
        public void hienThiThongTin()
        {
            base.Display();
            Console.WriteLine("Chi phi: {0}", tinhChiPhi());
        }
    }
}

```

Lớp **TestCsharp** chứa phương thức **main()** để thao tác trên đối tượng **ChiPhiXayDung**

```

using System;
namespace Ten_namespace_C
{
    public class TestCsharp
    {
        public static void Main(string[] args)
        {
            Console.WriteLine("Tinh ke thua trong C#");
            Console.WriteLine("Khoi tao lop co so");
            Console.WriteLine("-----\n");
            //tao doi tuong ChiPhiXayDung

```

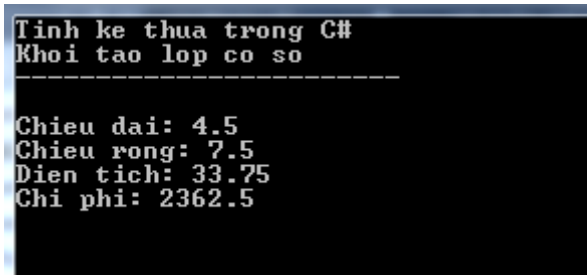
```

        ChiPhiXayDung t = new ChiPhiXayDung(4.5, 7.5);
        t.hienThiThongTin();
        Console.ReadLine();

        Console.ReadKey();
    }
}

```

Biên dịch và chạy chương trình C# trên sẽ cho kết quả sau:



```

Tinh ke thua trong C#
Khoi tao lop co so
-----
Chieu dai: 4.5
Chieu rong: 7.5
Dien tich: 33.75
Chi phi: 2362.5

```

Đa kế thừa trong C#

C# không hỗ trợ đa kế thừa. Tuy nhiên, bạn có thể sử dụng Interface để triển khai đa kế thừa. Ví dụ sau minh họa cách sử dụng Interface để triển khai đa kế thừa trong C#: chúng ta tạo 2 lớp có tên lần lượt là **Shape**, **HinhChuNhat**, **TestCsharp** và một interface có tên là **ChiPhiSon** như sau:

Lớp **Shape** là lớp cơ sở

```

using System;

namespace Ten_namespace_C
{
    class Shape
    {
        protected int chieu_rong;
        protected int chieu_cao;
        public void setChieuRong(int w)
        {
            chieu_rong = w;
        }
        public void setChieuCao(int h)
        {
            chieu_cao = h;
        }
    }
}

```

```
}
```

```
interface ChiPhiSon
```

```
using System;
```

```
namespace Ten_namespace_C  
{  
    public interface ChiPhiSon  
    {  
        int tinhChiPhi(int dien_tich);  
    }  
}
```

Lớp **HinhChuNhat** là lớp kế thừa lớp Shape và interface ChiPhiSon

```
using System;
```

```
namespace Ten_namespace_C  
{  
    class HinhChuNhat : Shape, ChiPhiSon  
    {  
        public int tinhDienTich()  
        {  
            return (chieu_rong * chieu_cao);  
        }  
        public int tinhChiPhi(int dien_tich)  
        {  
            return dien_tich * 70;  
        }  
    }  
}
```

Lớp **TestCsharp** chứa phương thức **main()** để thao tác trên đối tượng **HinhChuNhat**

```
using System;
```

```
namespace Ten_namespace_C  
{  
    public class TestCsharp  
    {  
        public static void Main(string[] args)  
        {  
            Console.WriteLine("Tinh ke thua trong C#");  
            Console.WriteLine("Vi du minh hoa Da ke thua");  
            Console.WriteLine("-----");  
        }  
    }  
}
```

```

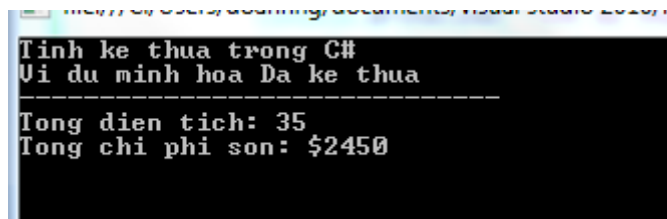
//tao doi tuong HinhChuNhat
HinhChuNhat hcn = new HinhChuNhat();
int dien_tich;
hcn.setChieuRong(5);
hcn.setChieuCao(7);
dien_tich = hcn.tinhDienTich();

// in dien tich va chi phi.
Console.WriteLine("Tong dien tich: {0}", hcn.tinhDienTich());
Console.WriteLine("Tong chi phi son: 0", hcn.tinhChiPhi(dien_tich));
Console.ReadLine();

Console.ReadKey();
}
}
}

```

Biên dịch và chạy chương trình C# trên sẽ cho kết quả sau:



```

Tinh ke thua trong C#
Vi du minh hoa Da ke thua
-----
Tong dien tich: 35
Tong chi phi son: $2450

```

3.5 Hàm ảo (virtual function), lớp trừu tượng (abstract class) và giao diện (interface)

3.5.1. Hàm ảo (virtual function)

Trong lập trình hướng đối tượng, việc kế thừa các class với nhau trở thành điều hiển nhiên. Và vấn đề phát sinh là làm sao gọi được phương thức con từ lớp cha. Từ khóa `virtual` giúp giải quyết vấn đề này

Từ khóa `virtual` được sử dụng để chỉnh sửa một phương thức, thuộc tính,.. Nó cho phép lớp con override lại từ lớp cha. Khi một phương thức `virtual` được thực thi, kiểu của đối tượng tại thời điểm thực thi (**run-time type**) **được kiểm tra và phương thức được override từ lớp con sẽ được gọi.**

Mặc định các phương thức là non-virtual. Từ khóa `virtual` **không thể** được sử dụng với các từ khóa `static`, `abstract`, `private` và `override`

Ví dụ bên dưới sẽ mô tả cách thức hoạt động của từ khóa `virtual`

```

using System;

namespace KeThua
{
    class Program
    {

```

```

class Horse
{
    public string SayHello()
    {
        return "Hello, I'm an horse";
    }
}
class Donkey : Horse
{
    public string SayHello()
    {
        return "Hello, I'm a donkey";
    }
}
static void PrintHello(Horse obj)
{
    Console.WriteLine(obj.SayHello());
}
static void Main(string[] args)
{
    var horse = new Horse();
    var donkey = new Donkey();
    PrintHello(horse);
    PrintHello(donkey);
    Console.ReadKey();
}
}

```

Kết quả thực thi chương trình

Hello, I'm an horse

Hello, I'm an horse

Chúng ta mong đợi `PrintHello(donkey)`; sẽ in ra `Hello, I'm a donkey`. Nhưng thực tế lại khác. Khi gọi hàm `PrintHello()`, `donkey` được ép qua kiểu `Horse`. Chính vì vậy, khi chương trình thực thi, nó sẽ gọi hàm `SayHello()` của class `Horse`.

Để chương trình biết phải gọi hàm `SayHello()` của class `Donkey` tại thời điểm thực thi. Thì từ khóa `virtual` và `override` cần được sử dụng.

Class `Horse` và `Donkey` cần được sửa như bên dưới.

```

class Horse
{
    public virtual string SayHello()
    {
        return "Hello, I'm an horse";
    }
}

```



```

    }
}
class Donkey : Horse
{
    public override string SayHello()
    {
        return "Hello, I'm a donkey";
    }
}

```

Bây giờ kết quả đã như mong đợi

Hello, I'm an horse

Hello, I'm a donkey

3.5.2. Lớp trừu tượng (abstract class)

Lớp trừu tượng là một trong những khái niệm quan trọng trên nền tảng .NET.

Thường, bạn muốn tạo ra các lớp mà chỉ làm lớp gốc (base class – lớp đầu tiên trong cây thừa kế hay còn gọi là lớp tổ tiên), và dĩ nhiên, bạn sẽ không muốn người khác tạo ra đối tượng cho các lớp này. Chúng ta có thể sử dụng khái niệm lớp trừu tượng để cài đặt chức năng này trong C# sử dụng từ khóa ‘**abstract**’.

Một lớp trừu tượng có nghĩa là **không khởi tạo được đối tượng** của lớp này, nhưng **cho phép thừa kế** để tạo ra lớp con.

Khai báo lớp trừu tượng trong C#:

```

?
1  abstract class tên_lớp_trừu_tượng
2  {
3  }

```

Ví dụ:

```

?
1  abstract class Window
2  {
3  }

```

Phương thức trừu tượng (abstract method)

Trong thiết kế hướng đối tượng, khi các bạn thiết kế ra một base class, và các bạn mong muốn rằng người khác khi thừa kế lớp này thì phải ghi đè (override) lên các phương thức xác định trước. Trong trường hợp người khác thừa kế lớp của các bạn mà không ghi đè lên những phương thức này thì trình biên dịch sẽ báo lỗi. Khái niệm phương thức trừu tượng sẽ giúp các bạn trong tình huống này.

Một lớp trừu tượng có thể chứa cả **phương thức trừu tượng** (phương thức

không có phần thân) và **phương thức bình thường** (phương thức có phần thân hay phương thức thàadow

```
public class ListBox : Window
{
    // Khởi dựng có tham số
    public ListBox(int top, int left, string theContents)
    : base(top, left) // gọi khởi dựng của lớp cơ sở
    {
        mListBoxContents = theContents;
    }
    public override void DrawWindow()
    {
        Console.WriteLine("ListBox write: {0}", mListBoxContents);
    }
    // biến thành viên private
    private string mListBoxContents;
    public override string Content
    {
        set { mListBoxContents = value; }
        get { return mListBoxContents; }
    }
}
[/code]
```

Trong ví dụ trên, các bạn thấy thuộc tính (property) Content được khai báo trong lớp Window, nhưng không có biến chứa dữ liệu cho nó không được khai báo trong lớp này. Do đó nó được cài đặt kiểu abstract.

Thuộc tính Content được **override** trong lớp con ListBox, và biến chứa dữ liệu cho nó là mListBoxContents.

Một số quy tắc áp dụng cho lớp trừu tượng

– Một lớp trừu tượng không thể là một sealed class. Khai báo như ví dụ dưới đây là **sai**:

```
?
1    // Khai báo sai
2    abstract sealed class Window
3    {
4    }
```

- Phương thức trừu tượng chỉ khai báo trong lớp trừu tượng.
- Một phương thức trừu tượng không sử dụng chỉ định từ truy xuất là ***private***.

?

```
1 // Khai báo sai
2 abstract class Window
3 {
4     private abstract void DrawWindow();
5 }
```

Chỉ định từ truy xuất của phương thức trừu tượng phải giống nhau trong phần khai báo ở lớp cha lẫn lớp con. Nếu bạn đã khai báo chỉ định từ truy xuất `protected` cho phương thức trừu tượng ở lớp cha thì trong lớp con bạn cũng phải sử dụng chỉ định từ truy xuất `protected`. Nếu chỉ định từ truy xuất không giống nhau thì trình biên dịch sẽ báo lỗi.

- Một phương thức trừu tượng không sử dụng chỉ định từ truy xuất `virtual`. Bởi vì bản thân phương thức trừu tượng đã bao hàm khái niệm `virtual`.

?

```
1 // Khai báo sai
2 abstract class Window
3 {
4     private abstract virtual void DrawWindow();
5 }
```

- Một phương thức trừu tượng không thể là phương thức `static`.

?

```
1 // Khai báo sai
2 abstract class Window
3 {
4     private abstract static void DrawWindow();
5 }
```

Lớp trừu tượng (abstract class) và giao diện (interface)

Trong lớp trừu tượng chứa cả phương thức trừu tượng lẫn phương thức thành viên. Nhưng trong interface thì chỉ chứa phương thức trừu tượng và lớp con khi thừa kế từ interface cần phải ghi đè (override) lên các phương thức này.

?

```
1 interface IFile
2 {
3     void Save();
4     void Load();
5 }
```

Các phương thức trừu tượng khai báo trong interface không sử dụng chỉ định từ truy xuất, mặc định sẽ là public.

Một lớp chỉ có thể **thừa kế** từ *một lớp cha*, nhưng có thể thừa kế từ *nhiều interface*.

3.5.3. Giao diện (Interface)

Một Interface được định nghĩa như là một giao ước có tính chất cú pháp (syntactical contract) mà tất cả lớp kế thừa Interface đó nên theo. Interface định nghĩa phần "**Là gì**" của giao ước và các lớp kế thừa định nghĩa phần "**Cách nào**" của giao ước đó.

Interface định nghĩa các thuộc tính, phương thức và sự kiện, mà là các thành viên của Interface đó. Các Interface chỉ chứa khai báo của các thành viên này. Việc định nghĩa các thành viên là trách nhiệm của lớp kế thừa. Nó thường giúp ích trong việc cung cấp một Cấu trúc chuẩn mà các lớp kế thừa nên theo.

Khai báo Interface trong C#

Các Interface được khai báo bởi sử dụng từ khóa interface trong C#. Nó tương tự như khai báo lớp. Theo mặc định, các lệnh Interface là public. Ví dụ sau minh họa một khai báo Interface trong C#:

```
public interface ITransactions
{
    // các thành viên của interface

    // các phương thức
    void hienThiThongTinGiaoDich();
    double laySoLuong();
}
```

Ví dụ

Sau đây là ví dụ minh họa trình triển khai của Interface trên: tạo 2 lớp có tên lần lượt là **GiaoDichHangHoa**, **TestCsharp** và một interface có tên là **GiaoDich**

interface **GiaoDich**

```
using System;

namespace Ten_namespace_C
{
    public interface GiaoDich
```

```

{
    // cac thanh vien cua interface

    //cac phuong thuc
    void hienThiThongTinGiaoDich();
    double laySoLuong();
}
}

```

Lớp **GiaoDichHangHoa** kế thừa interface **GiaoDich**

```

using System;

namespace Ten_namespace_C
{
    class GiaoDichHangHoa : GiaoDich
    {
        private string ma_hang_hoa;
        private string ngay;
        private double so_luong;
        public GiaoDichHangHoa()
        {
            ma_hang_hoa = " ";
            ngay = " ";
            so_luong = 0.0;
        }

        public GiaoDichHangHoa(string c, string d, double a)
        {
            ma_hang_hoa = c;
            ngay = d;
            so_luong = a;
        }

        public double laySoLuong()
        {
            return so_luong;
        }

        public void hienThiThongTinGiaoDich()
        {
            Console.WriteLine("Ma hang hoa: {0}", ma_hang_hoa);
            Console.WriteLine("Ngày giao dich: {0}", ngay);
            Console.WriteLine("So luong: {0}", laySoLuong());
        }
    }
}

```

```
}  
}  
}
```

Lớp **TestCsharp** chứa phương thức **main()** để thao tác trên đối tượng **GiaoDichHangHoa**

```
using System;  
namespace Ten_namespace_C  
{  
    public class TestCsharp  
    {  
        static void Main(string[] args)  
        {  
            Console.WriteLine("Interface trong C#");  
            Console.WriteLine("Vi du minh hoa interface");  
            Console.WriteLine("-----");  
  
            //tao cac doi tuong GiaoDichHangHoa  
            GiaoDichHangHoa t1 = new GiaoDichHangHoa("001", "8/10/2012", 78  
900.00);  
            GiaoDichHangHoa t2 = new GiaoDichHangHoa("002", "9/10/2012", 45  
1900.00);  
            t1.hienThiThongTinGiaoDich();  
            t2.hienThiThongTinGiaoDich();  
  
            Console.ReadKey();  
        }  
    }  
}
```

Nếu bạn không sử dụng lệnh **Console.ReadKey();** thì chương trình sẽ chạy và kết thúc luôn (nhanh quá đến nỗi bạn không kịp nhìn kết quả). Lệnh này cho phép chúng ta nhìn kết quả một cách rõ ràng hơn.

Chương 4: Ủy quyền (delegate) và sự kiện (event)

Trong các ứng dụng Windows Form, khi ta click chuột vào một button hay chọn một mục trong listbox, thì đều phát sinh một sự kiện (event). Chương trình có nhiệm vụ xử lý các sự kiện này, chứ không phải các control button hay listbox xử lý. Chính cơ chế này đã tạo ra khả năng tương tác linh hoạt giữa người dùng và chương trình. Người dùng gửi đến chương trình các yêu cầu thông qua các sự kiện, chương trình đáp ứng sự kiện đó và thực hiện một tác vụ nào đó.

Những sự kiện đó đều được thực thi bởi cơ chế ủy quyền (delegate). Trong bài viết này, tôi sẽ giới thiệu với các bạn khái niệm delegate và cách sử dụng nó để tạo event (sự kiện) trong C#.

4.1 Thực thi delegate

Delegate là gì?

Delegate đơn giản chỉ là một kiểu dữ liệu tham chiếu (reference type) dùng để đóng gói một phương thức với tham số và kiểu trả về xác định. Chúng ta có thể đóng gói bất kỳ phương thức nào trong một delegate thích hợp.

Delegate trong C# cũng tương tự như con trỏ hàm trong C++ (function pointer). Bạn sẽ dễ dàng tiếp cận delegate nếu bạn đã từng sử dụng con trỏ hàm trong C++.

Delegate thường được sử dụng để tạo các sự kiện và các hàm callback cho chương trình.

Khai báo delegate

Cú pháp khai báo delegate trong C# như sau:

1 **chỉ_định_từ_truy_xuất** delegate kiểu_trả_về tên_delegate(danh_sách_tham_số);
Trong đó:

- **chỉ_định_từ_truy_xuất**: là một trong các thuộc tính truy cập: private, public, protected, internal.
- **kiểu_trả_về**: kiểu trả về của phương thức
- **tên_delegate**: tên của delegate
- **danh_sách_tham_số**: các tham số của phương thức

Chú ý: delegate chỉ mô tả dạng tổng quát của phương thức, bao gồm kiểu trả về và tham số. Còn phương thức cụ thể sẽ được truyền vào thông qua một thể hiện (instance) của delegate. Ví dụ:

```
1 public delegate void MyEventHandler(string Message);
```

Delegate này đại diện cho tập các phương thức có một tham số đầu vào có kiểu string và có kiểu trả về là void.

Để sử dụng delegate, chúng ta phải tạo một thể hiện của nó (tương tự như tạo đối tượng của lớp) và truyền vào phương thức phù hợp (kiểu trả về và tham số) vào hàm khởi tạo của nó. Ví dụ:

```
1 using System;
2
3 namespace DelegateDemo
4 {
5     public delegate void MyEventHandler(string msg);
6
7     public class Demo
8     {
9         public static void Main()
10        {
11            Demo d = new Demo();
12            MyEventHandler handler = new MyEventHandler(d.DisplayMsg);
13            handler("Display Message Here.");
14            Console.ReadLine();
15        }
16        public void DisplayMsg(string msg)
17        {
18            Console.WriteLine(msg);
19        }
20    }
21 }
```

Trong ví dụ trên, tôi đã tạo ra một delegate tên là **MyEventHandler** đại diện cho các phương thức có một tham số đầu vào có kiểu string và có kiểu trả về là void. Sau đó, tôi tạo ra một thể hiện của **MyEventHandler** có tên là **handler**, truyền vào phương thức khởi tạo của nó là **DisplayMsg** (dòng 12 trong ví dụ minh họa ở trên). Hàm **DisplayMsg** này có phần khai báo giống với phần khai báo của delegate **MyEventHandler**.

Khi các bạn gọi **handler** và truyền cho nó chuỗi: “Display Message Here.”, nó sẽ gọi hàm **DisplayMsg** và truyền chuỗi “Display Message Here.” cho hàm này.

4.2 Thực thi multicast delegate

Kỹ thuật Multicasting

C# cho phép chúng ta “gắn” nhiều phương thức vào cùng một delegate, miễn là các phương thức này có cùng kiểu trả về và tham số với delegate. Và khi ta gọi delegate này, tất cả các phương thức đã được “gắn” sẽ thi hành cùng lúc. Kỹ thuật này gọi là “**Multicasting**”.

Chú ý : Một multicast delegate chỉ chấp nhận phương thức có kiểu trả về là void.

Để thực hiện Multicasting, ta tạo các thể hiện của một multicast delegate, gắn nó với các phương thức tương ứng. Sau đó dùng toán tử “+” để gom các delegate này vào thành 1 delegate duy nhất. Dưới đây là ví dụ minh họa:

```
?
1    using System;
2
3    namespace DelegateDemo
4    {
5        public delegate void MyEventHandler(string msg);
6
7        public class Demo
8        {
9            public static void Main()
10           {
11               Demo d = new Demo();
12               MyEventHandler handler1 = new MyEventHandler(d.DisplayMsg);
13               MyEventHandler handler2 = new MyEventHandler(d.DisplayMsg);
14               MyEventHandler handler = handler1 + handler2;
15               handler("Test");
16               Console.ReadLine();
17           }
18           public void DisplayMsg(string msg)
19           {
20               Console.WriteLine(msg);
21           }
22           public void ShowHello(string name)
23           {
24               Console.WriteLine("Hello " + name);
25           }
26       }
27   }
```

Trong ví dụ trên, đầu tiên, chúng ta tạo ra hai delegate là *handler1* và *handler2* gắn hai

hàm *DisplayMsg()* và *ShowHello()* vào. Sau đó, chúng ta tạo delegate tên là *handler* và “cộng” hai delegate *handler1* và *handler2* lại với nhau. Như vậy, khi gọi delegate *handler*, hai delegate *handler1* và *handler2* sẽ đồng thời được gọi.

Ngoài toán tử “+”, C# còn hỗ trợ toán tử “+=” và “-=” trên delegate. Ví dụ:

```
1    using System;
2
3    namespace DelegateDemo
4    {
5        public delegate void MyEventHandler(string msg);
6
7        public class Demo
8        {
9            public static void Main()
10           {
11               Demo d = new Demo();
12               MyEventHandler handler = new MyEventHandler(d.DisplayMsg);
13               handler += d.ShowHello;
14               handler("Test");
15               Console.ReadLine();
16           }
17           public void DisplayMsg(string msg)
18           {
19               Console.WriteLine(msg);
20           }
21           public void ShowHello(string name)
22           {
23               Console.WriteLine("Hello " + name);
24           }
25       }
26   }
```

4.3 Sử dụng event với delegate

Event (sự kiện)

Một trong những ứng dụng thường thấy nhất của delegate là event. Mỗi sự kiện thực chất là một delegate.

Cú pháp khai báo event trong C# như sau:

chỉ_định_từ_truy_xuất event tên_delegate tên_event;

Trong đó:

- **chỉ_định_từ_truy_xuất:** là một trong các thuộc tính truy cập: private, public, protected, internal.
- **tên_delegate:** kiểu của delegate đại diện cho event
- **tên_event:** tên của event

Ví dụ:

```
1 public delegate void MyEventHandler(string msg);
2 public event MyEventHandler Click;
```

Sau đó chúng ta có thể phát ra sự kiện (fire event) bằng cách gọi tên sự kiện và truyền tham số tương ứng.

Ví dụ:

```
1 namespace DelegateDemo
2 {
3     public delegate void MyEventHandler(string msg);
4     public class Window
5     {
6         public event MyEventHandler Click;
7
8         public Window(int top, int left)
9         {
10             this.top = top;
11             this.left = left;
12         }
13         // mô phỏng vẽ cửa sổ
14         public virtual void DrawWindow() { }
15         public void FireEvent()
16         {
17             if (Click != null)
18                 Click("Event Fire.");
19         }
20         // Có hai biến thành viên private do đó hai
21         // biến này sẽ không thấy bên trong lớp con
22         int top;
23         private int left;
24     }
25     // ListBox dẫn xuất từ Window
26     public class ListBox : Window
27     {
28         public ListBox(int top, int left)
29             : base(top, left)
30         {
31             //Console.WriteLine("Constructor's ListBox have 2 parameter");
```

```

32     }
33     // Khởi dựng có tham số
34     public ListBox(int top, int left, string theContents)
35         : base(top, left) // gọi khởi dựng của lớp cơ sở
36     {
37         mListBoxContents = theContents;
38     }
39
40     public override void DrawWindow()
41     {
42         Console.WriteLine("DrawWindow's ListBox");
43     }
44
45     // biến thành viên private
46     private string mListBoxContents;
47 }
48
49 public class Tester
50 {
51     public static void Main()
52     {
53         Window w = new Window(100, 100);
54         w.Click += w_Click;
55         w.FireEvent(); // phát sinh sự kiện
56         Console.ReadLine();
57     }
58
59     static void w_Click(string msg)
60     {
61         Console.WriteLine(msg);
62     }
63 }
64 }

```

Chương 5: Thread và đồng bộ thread

5.1 Thực thi thread

Thread là một process “nhẹ cân” cung cấp khả năng multitasking trong một ứng dụng. Vùng tên System.Threading cung cấp nhiều lớp và giao diện để hỗ trợ lập trình nhiều thread.

Thread

Thread thường được tạo ra khi bạn muốn làm đồng thời 2 việc trong cùng một thời điểm. Giả sử ứng dụng của bạn đang tiến hành đọc vào bộ nhớ một tập tin có kích thước khoảng 500MB, trong lúc đang đọc thì dĩ nhiên ứng dụng không thể đáp ứng yêu cầu xử lý giao diện. Giả sử người dùng muốn ngưng giữa chừng, không cho ứng dụng đọc tiếp tập tin lớn đó nữa, do đó cần một thread khác để xử lý giao diện, lúc này khi người dùng ấn nút Stop thì ứng dụng đáp ứng được yêu cầu trong khi thread ban đầu vẫn đang đọc tập tin.

Tạo Thread

Cách đơn giản nhất là tạo một thể hiện của lớp Thread. Constructor của lớp Thread nhận một tham số kiểu delegate. CLR cung cấp lớp delegate ThreadStart nhằm mục đích chỉ đến phương thức mà bạn muốn thread mới thực thi. Khai báo delegate ThreadStart như sau:

```
public delegate void ThreadStart( );
```

Phương thức mà bạn muốn gán vào delegate phải không chứa tham số và phải trả về kiểu void. Sau đây là ví dụ:

```
Thread myThread = new Thread( new ThreadStart(myFunc) );
```

myFunc phải là phương thức không tham số và trả về kiểu **void**.

Xin lưu ý là đối tượng **Thread** mới tạo sẽ không tự thực thi (**execute**), để đối tượng thực thi, bạn cần gọi phương thức **Start()** của nó.

```
Thread t1 = new Thread( new ThreadStart(Incrementer) ); Thread t2 = new Thread( new ThreadStart(Decrementer) ); t1.Start( ); t2.Start( );
```

Thread sẽ chấm dứt khi hàm mà nó thực thi trở về (**return**).

Gia nhập Thread

Hiện tượng thread A ngưng chạy và chờ cho thread B chạy xong được gọi là thread A gia nhập thread B.

Để thread 1 gia nhập thread 2:

```
t2.Join( );
```

Nếu câu lệnh trên được thực hiện bởi thread 1, thread 1 sẽ dừng lại và chờ cho đến khi thread kết thúc.

Treo thread lại (suspend thread)

Nếu bạn muốn treo thread đang thực thi lại một khoảng thời gian thì bạn sử dụng hàm Sleep() của đối tượng Thread. Ví dụ để thread ngưng khoảng 1 giây:

```
Thread.Sleep(1000);
```

Câu lệnh trên báo cho bộ điều phối thread (của hệ điều hành) biết bạn không muốn bộ điều phối thread phân phối thời gian CPU cho thread thực thi câu lệnh trên trong thời gian 1 giây.

5.2 Xác định vòng đời của thread

Thông thường thread sẽ chấm dứt khi hàm mà nó thực thi trở về. Tuy nhiên bạn có thể yêu cầu một thread “tự tử” bằng cách gọi hàm **Interrupt()** của nó. Điều này sẽ làm cho exception **ThreadInterruptedException** được ném ra. Thread bị yêu cầu “tự tử” có thể bắt exception này để tiến hành dọn dẹp tài nguyên.

```
catch (ThreadInterruptedException)
{
    Console.WriteLine("[{0}] Interrupted! Cleaning up...", Thread.CurrentThread.
    Name);
}
```

5.3 Độ ưu tiên của thread

- Khi sử dụng nhiều thread thì ta cài đặt độ ưu tiên thực thi của các thread

```
t1.Priority = ThreadPriority.Lowest;
```

- Có các giá mức ưu tiên: Lowest, BelowNormal, Normal, AboveNormal, Highest

5.4 Đồng bộ thread

Đồng bộ hóa (Synchronization)

Khi bạn cần bảo vệ một tài nguyên, trong một lúc chỉ cho phép một thread thay đổi hoặc sử dụng tài nguyên đó, bạn cần đồng bộ hóa.

Đồng bộ hóa được cung cấp bởi một khóa trên đối tượng đó, khóa đó sẽ ngăn cản thread thứ 2 truy cập vào đối tượng nếu thread thứ nhất chưa trả quyền truy cập đối tượng.

Sau đây là ví dụ cần sự đồng bộ hóa. Giả sử 2 thread sẽ tiến hành tăng tuần tự 1 đơn vị một biến tên là counter.

```
int counter = 0;
Hàm làm thay đổi giá trị của Counter:
public void Incrementer( )
{
    try
    {
        while (counter < 1000)
        {
```

```

int temp = counter;
temp++; // increment
// simulate some work in this method
Thread.Sleep(1);
// assign the Incremented value
// to the counter variable
// and display the results
counter = temp;
Console.WriteLine("Thread {0}. Incrementer: {1}",
Thread.CurrentThread.Name, counter);
}
}

```

Vấn đề ở chỗ thread 1 đọc giá trị counter vào biến tạm rồi tăng giá trị biến tạm, trước khi thread 1 ghi giá trị mới từ biến tạm trở lại counter thì thread 2 lại đọc giá trị counter ra biến tạm của thread 2. Sau khi thread 1 ghi giá trị vừa tăng 1 đơn vị trở lại counter thì thread 2 lại ghi trở lại counter giá trị mới bằng với giá trị mà thread 1 vừa ghi. Như vậy sau 2 lần truy cập giá trị của biến counter chỉ tăng 1 đơn vị trong khi yêu cầu là phải tăng 2 đơn vị.

Sử dụng Interlocked

CLR cung cấp một số cơ chế đồng bộ từ cơ chế đơn giản Critical Section (gọi là Locks trong .NET) đến phức tạp như Monitor.

Tăng và giảm giá trị làm một nhu cầu phổ biến, do đó C# cung cấp một lớp đặc biệt Interlocked nhằm đáp ứng nhu cầu trên. Interlocked có 2 phương thức Increment() và Decrement() nhằm tăng và giảm giá trị trong sự bảo vệ của cơ chế đồng bộ. Ví dụ ở phần trước có thể sửa lại như sau:

```

public void Incrementer( )
{
try
{
while (counter < 1000)
{
Interlocked.Increment(ref counter);
// simulate some work in this method
Thread.Sleep(1);
// assign the decremented value
// and display the results
Console.WriteLine(
"Thread {0}. Incrementer: {1}",
Thread.CurrentThread.Name, counter);
}
}
}

```

```
}
```

Khối catch và finally không thay đổi so với ví dụ trước.

Sử dụng Locks

Lock đánh dấu một đoạn mã “gay cần” (**critical section**) trong chương trình của bạn, cung cấp cơ chế đồng bộ cho khối mã mà lock có hiệu lực.

C# cung cấp sự hỗ trợ cho lock bằng từ chốt (**keyword**) lock. Lock được gỡ bỏ khi hết khối lệnh.

```
public void Incrementer( )
{
    try
    {
        while (counter < 1000)
        {
            lock (this)
            { // lock bắt đầu có hiệu lực
                int temp = counter; temp ++; Thread.Sleep(1); counter = temp;
            } // lock hết hiệu lực -> bị gỡ bỏ
            // assign the decremented value
            // and display the results
            Console.WriteLine( "Thread {0}. Incrementer: {1}", Thread.CurrentThread.Name, counter);
        }
    }
}
```

Khối catch và finally không thay đổi so với ví dụ trước.

Sử dụng Monitor

Để có thể đồng bộ hóa phức tạp hơn cho tài nguyên, bạn cần sử dụng monitor. Một monitor cho bạn khả năng quyết định khi nào thì bắt đầu, khi nào thì kết thúc đồng bộ và khả năng chờ đợi một khối mã nào đó của chương trình “tự do”.

Khi cần bắt đầu đồng bộ hóa, trao đối tượng cần đồng bộ cho hàm sau:

```
Monitor.Enter(đối tượng X);
```

Nếu monitor không sẵn dùng (unavailable), đối tượng bảo vệ bởi monitor đang được sử dụng. Bạn có thể làm việc khác trong khi chờ đợi monitor sẵn dùng (**available**) hoặc treo thread lại cho đến khi có monitor (bằng cách gọi hàm **Wait()**)

Ví dụ bạn đang download và in một bài báo từ Web. Để hiệu quả bạn cần tiến hành in sau hậu trường (**background**), tuy nhiên bạn cần chắc chắn rằng 10 trang đã được download trước khi bạn tiến hành in.

Thread in ấn sẽ chờ đợi cho đến khi thread download báo hiệu rằng số lượng trang download đã đủ. Bạn không muốn gia nhập (join) với thread download vì số lượng trang có thể lên đến vài trăm. Bạn muốn chờ cho đến khi ít nhất 10 trang đã được download.

Để giả lập việc này, bạn thiết lập 2 hàm đếm dùng chung 1 biến counter. Một hàm đếm tăng 1 tương ứng với thread download, một hàm đếm giảm 1 tương ứng với thread in ấn.

Trong hàm làm giảm bạn gọi phương thức Enter(), sau đó kiểm tra giá trị counter, nếu < 5 thì gọi hàm Wait()

```
if (counter < 5)
{
    Monitor.Wait(this);
}
```

Lời gọi Wait() giải phóng monitor nhưng bạn đã báo cho CLR biết là bạn muốn lấy lại monitor ngay sau khi monitor được tự do một lần nữa. Thread thực thi phương thức Wait() sẽ bị treo lại. Các thread đang treo vì chờ đợi monitor sẽ tiếp tục chạy khi thread đang thực thi gọi hàm Pulse().

```
Monitor.Pulse(this);
```

Pulse() báo hiệu cho CLR rằng có sự thay đổi trong trạng thái monitor có thể dẫn đến việc giải phóng (tiếp tục chạy) một thread đang trong tình trạng chờ đợi.

Khi thread hoàn tất việc sử dụng monitor, nó gọi hàm Exit() để trả monitor.

```
namespace Programming_CSharp
{
    using System;
    using System.Threading;
    class Tester
    {
        static void Main( )
        {
            // make an instance of this class
            Tester t = new Tester( );
            // run outside static Main
            t.DoTest( );
        }
        public void DoTest( )
        {
            // create an array of unnamed threads
            Thread[] myThreads = {new Thread( new ThreadStart(Decrementer) ),
                                   new Thread( new ThreadStart(Incrementer) ) };
        }
    }
}
```

```

// start each thread
int ctr = 1;
foreach (Thread myThread in myThreads)
{
    myThread.IsBackground=true;
    myThread.Start( );
    myThread.Name = "Thread" + ctr.ToString( );
    ctr++;
    Console.WriteLine("Started thread {0}",myThread.Name);
    Thread.Sleep(50);
}
// wait for all threads to end before continuing foreach (Thread myThread in my
Threads)
{
    myThread.Join( );
}
// after all threads end, print a message
Console.WriteLine("All my threads are done.");
}
void Decrementer( )
{
    try
    {
        // synchronize this area of code
        Monitor.Enter(this);
        // if counter is not yet 10
        // then free the monitor to other waiting
        // threads, but wait in line for your turn
        if (counter < 10)
        {
            Console.WriteLine(
                "[{0}] In Decrementer. Counter: {1}. Gotta Wait!", Thread.CurrentThread.Nam
                e, counter);
            Monitor.Wait(this);
        }
        while (counter > 0)
        {
            long temp = counter; temp--; Thread.Sleep(1); counter = temp;
            Console.WriteLine("[{0}] In Decrementer. Counter: {1}.", Thread.CurrentThre
                ad.Name, counter);
        }
    }
}

```

```

finally
{
Monitor.Exit(this);
}
}
void Incrementer( )
{
try
{
Monitor.Enter(this);
while (counter < 10)
{
long temp = counter; temp++; Thread.Sleep(1); counter = temp;
Console.WriteLine("[{0}] In Incrementer. Counter: {1}", Thread.CurrentThrea
d.Name, counter);
}
// I'm done incrementing for now, let another
// thread have the Monitor
Monitor.Pulse(this);
}
finally
{
Console.WriteLine("[{0}] Exiting...", Thread.CurrentThread.Name);
Monitor.Exit(this);
}
}
private long counter = 0;
}
}

```

Kết quả:

```

Started thread Thread1
[Thread1] In Decrementer. Counter: 0. Gotta Wait!
Started thread Thread2
[Thread2] In Incrementer. Counter: 1
[Thread2] In Incrementer. Counter: 2
[Thread2] In Incrementer. Counter: 3
[Thread2] In Incrementer. Counter: 4
[Thread2] In Incrementer. Counter: 5
[Thread2] In Incrementer. Counter: 6
[Thread2] In Incrementer. Counter: 7
[Thread2] In Incrementer. Counter: 8

```

```
[Thread2] In Incrementer. Counter: 9
[Thread2] In Incrementer. Counter: 10
[Thread2] Exiting...
[Thread1] In Decrementer. Counter: 9.
[Thread1] In Decrementer. Counter: 8.
[Thread1] In Decrementer. Counter: 7.
[Thread1] In Decrementer. Counter: 6.
[Thread1] In Decrementer. Counter: 5.
[Thread1] In Decrementer. Counter: 4.
[Thread1] In Decrementer. Counter: 3.
[Thread1] In Decrementer. Counter: 2.
[Thread1] In Decrementer. Counter: 1.
[Thread1] In Decrementer. Counter: 0.
All my threads are done.
```

Race condition và DeadLock

Đồng bộ hóa thread khá rắc rối trong những chương trình phức tạp. Bạn cần phải cẩn thận kiểm tra và giải quyết các vấn đề liên quan đến đồng bộ hóa thread: race condition và deadlock

Race condition

Một điều kiện tranh đua xảy ra khi sự đúng đắn của ứng dụng phụ thuộc vào thứ tự hoàn thành không kiểm soát được của 2 thread độc lập với nhau.

Giả sử bạn có 2 thread. Thread 1 tiến hành mở tập tin, thread 2 tiến hành ghi lên cùng tập tin đó. Điều quan trọng là bạn cần phải điều khiển thread 2 sao cho nó chỉ tiến hành công việc sau khi thread 1 đã tiến hành xong. Nếu không, thread 1 sẽ không mở được tập tin vì tập tin đó đã bị thread 2 mở để ghi. Kết quả là chương trình sẽ ném ra exception hoặc tệ hơn nữa là crash.

Để giải quyết vấn đề trong ví dụ trên, bạn có thể tiến hành join thread 2 với thread 1 hoặc thiết lập monitor.

Deadlock

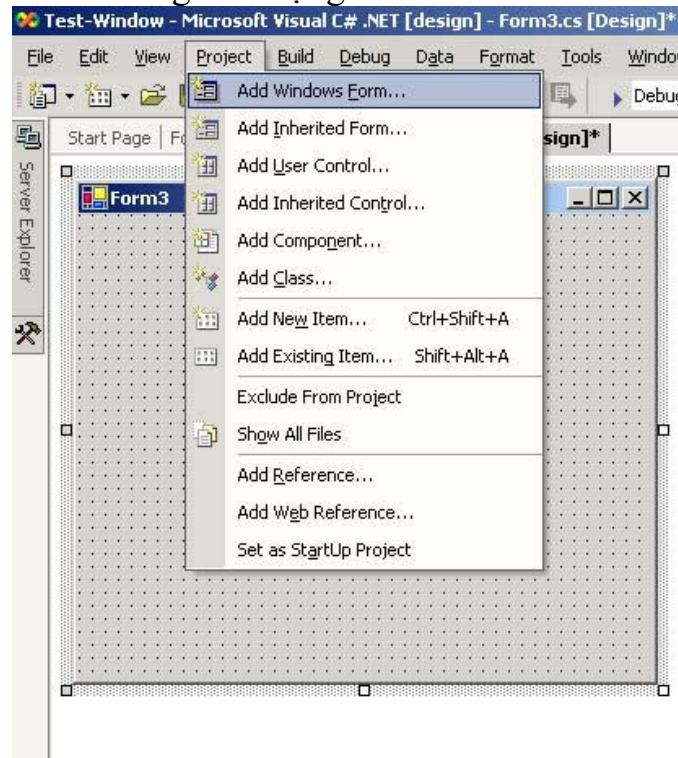
Giả sử thread A đã nắm monitor của tài nguyên X và đang chờ monitor của tài nguyên Y. Trong khi đó thì thread B lại nắm monitor của tài nguyên Y và chờ monitor của tài nguyên X. 2 thread cứ chờ đợi lẫn nhau mà không thread nào có thể thoát ra khỏi tình trạng chờ đợi. Tình trạng trên gọi là deadlock.

Trong một chương trình nhiều thread, deadlock rất khó phát hiện và gỡ lỗi. Một hướng dẫn để tránh deadlock đó là giải phóng tất cả lock đang sở hữu nếu tất cả các lock cần nhận không thể nhận hết được. Một hướng dẫn khác đó là giữ lock càng ít càng tốt.

Chương 6: Giới thiệu lập trình winform và winform control

6.1 Chương trình winform đầu tiên

Lập Trình Xử Lý Giao Diện Trong Winform với C#. NET Framework cho ta ba cách để user giao diện với chương trình áp dụng, đó là Windows Forms (có khi được gọi tắt là WinForms), Web Forms và Console applications. Trong chương trình của chúng ta chỉ học phần Windows Form cho lập trình xử lý giao diện và các ứng dụng cơ sở dữ liệu và phần Console applications cho phần hướng dẫn về lập trình hướng đối tượng.



6.1.1. Sự quan trọng của Windows Forms

Window form chính là cửa sổ của một màn hình ứng dụng. Nó chứa đựng các dữ liệu, control trên đó và là cửa sổ giao tiếp giữa người sử dụng (user) và máy tính.

6.1.2. Những điểm căn bản của Window Form

Trong các bài học và thí dụ trước đây ta đã nói qua, bây giờ ta tóm tắt những điểm căn bản của Windows Forms:

- Một Windows Form thật sự là một **class**. Trong .NET không có từ đặc biệt như “form module” để dùng cho nó.
- Vì một form là một class nên ta không thể load nó mà không chỉ rõ ra. Tức là trong VB6 nếu ta Show hay dùng đến một Form thì nó tự động được loaded. Chẳng những thế thôi, cái **class Form2** được dùng như một **variable Form2** luôn, tức là by default ta có một Object tên Form2. Trong .NET ta phải khai báo (declare) một variable tên myForm2 chẳng hạn rồi instantiate form ấy như một Object của Form2 trước khi dùng nó.
- Tất cả mọi form đều thừa kế từ **class System.Windows.Forms.Form**.
- Giống như tất cả các classes trong .NET Framework, Windows Forms có **constructors** và **destructors**. Constructor của form tên là **void New**, đại khái

giống như Sub Form_Load trong VB6. Destructor của form tên là **void Dispose**, đại khái giống như Sub Form_Unload trong VB6.

- *Cái visual forms designer của VS.NET nhét rất nhiều code để instantiate form và đặt các controls vào form.* Đó là code mà đáng lẽ ta phải tự viết nếu ta dùng notepad để lập trình. Phần code này thay thế cái phần nằm ở đầu tệp .frm của VB6 để diễn tả các visual components của form. Mỗi lần ta thêm bớt các controls hay thay thế các properties của controls trên form thì code generated cho form được thay đổi theo. Do đó bạn nên tránh sửa đổi code ấy, trừ khi biết chắc mình đang làm gì, hay là bạn làm một phiên bản trước khi thay đổi để nếu lỡ kệt thì restore code cũ.
- *Event được xử lý bằng cách linh động hơn.* Các events chứa nhiều tin tức hơn. Một Event có thể được xử lý bởi nhiều controls cùng một lúc và mỗi control có một cách xử lý khác nhau. Ngược lại, nhiều Events khác nhau có thể được xử lý bằng một Event Handler duy nhất.

6.1.3. Tạo một form mới

Khi bạn tạo một dự án (project), thì 1 form sẽ được tự động đưa vào dự án với tên là form1. Tuy nhiên, trong một dự án sẽ thường có nhiều màn hình cửa sổ khác nhau (form), bạn sẽ thêm form vào dự án bằng cách vào menu **Project - > Add Window Form** như hình dưới:

Đánh vào tên file cho form (mặc định nó sẽ là form1, form2,) và chọn **open**. Trên màn hình của VS .Net, 1 form bao giờ cũng có 2 phần: phần form design và phần source code của form.

– Màn hình của form design: đây là phần dùng để tạo giao diện cho 1 form bằng cách bạn kéo các đối tượng bên thanh công cụ và đưa vào trong form.

– Màn hình của source code

Màn hình source code là nơi để bạn viết các đoạn code của mình để xử lý.

Trong lập trình Visual thì phần lớn thời gian là ta thiết kế và xử lý các giao diện, thời gian viết code sẽ không nhiều và số lượng code để viết cũng không nhiều, chủ yếu là chỉ viết code để xử lý các sự kiện (ta sẽ bàn tới sau) cho các đối tượng mà thôi. Đây chính là thế mạnh của của lập trình visual.

6.1.4. Vị trí ban đầu của form khi chạy chương trình

Với 1 ứng dụng, ta có thể muốn form hiển thị ngay giữa màn hình, hoặc ta có thể muốn form này hiển thị giữa form kia, v.v... VS .Net cho ta cửa sổ **properties** để bạn lựa chọn các thông số **StartPosition** của nó như sau:

Ý nghĩa của các thông số:

Trị số Vị trí khởi đầu	Kết quả
Manual	Hiển thị form ở vị trí theo giá trị của property Location của form
CenterScreen	Hiển thị form ở ngay giữa màn ảnh
CenterParent	Hiển thị form ở ngay giữa form chủ (owner) của nó

WindowsDefaultLocation	Hiển thị form ở vị trí default của cửa sổ
WindowsDefaultBounds	Hiển thị form ở vị trí default của cửa sổ, với kích thước default của cửa sổ

6.1.5. Border của form

Một form thường có các đường viền (border) của nó, ta có thể thay đổi giá trị đường viền để form có những hình ảnh hiển thị khác nhau. Bạn có thể dùng cửa sổ **properties** để set lại giá trị **FormBorderStyle**

Dưới đây là bảng giá trị của thuộc tính **FormBorderStyle**:

Loại	Giá trị	Diễn giải
None	0	Không có đường biên
FixedSingle	1	Tường tự như FixedDialog
Gfixed3D	2	Trông giống như 3 chiều
FixedDialog	3	Như hộp hội thoại
Sizable	4	Mặc nhiên
FixedToolWindow	5	Thanh caption nhỏ và không có nút close
SizableToolWindow	6	FixedToolWindow nhưng đường biên mỏng.

6.1.6. Và một số thuộc tính khác

Một số thuộc tính khác khá quan trọng của form:

Loại	Giá trị	Diễn giải
ControlBox	True False	Có hay không có 3 nút min, max, và close trên góc phải
Maximizebox	True False	Có hay không có nút max
Minimizebox	True False	Có hay không có nút min
TopMost	True False	Form này có luôn nằm lên trên hết các form khác hay không
Caption	Text	Đoạn text nằm trên thanh bar trên đầu của form
Name	Text	Tên của form.

Lưu ý: phần lớn, ta có thể thiết lập các giá trị **properties** cho form hay các controls (button, label, v.v...) bằng 2 cách:

- Thiết lập bằng design (design-time) thông qua cửa sổ **properties** của form hay control đó.
- Thiết lập bằng lập trình (run-time) thông qua các đoạn code.

6.1.7. Các sự kiện của form.

Trong lập trình visual, điều quan trọng nhất là xử lý các sự kiện. Khi lập trình, thường ta chỉ thực hiện các thao tác kéo thả là ta có thể tạo được một giao diện hoàn chỉnh. Tuy nhiên, để giao diện đó hoạt động được theo đúng yêu cầu của ta thì ta buộc phải lập trình cho các sự kiện của từng hay nhiều control trên form đó.

Form có rất nhiều sự kiện, ở đây chỉ giới thiệu đến một số sự kiện quan trọng của form:

– Sự kiện Load:

```
private void Form3_Load(object sender, System.EventArgs e)
{
    //Code xử lý cho quá trình form được load lên.
}
```

– Sự kiện Closed:

```
private void Form3_Closed(object sender, System.EventArgs e)
{
    //Code xử lý cho quá trình form đang được đóng lại.
}
```

– Sự kiện Click (khi ta click mouse lên trên form):

```
private void Form3_Click(object sender, System.EventArgs e)
{
    //Code xử lý khi chuột được click lên trên form.
}
```

Để tạo 1 hàm xử lý sự kiện (event function), cách tốt nhất là bạn bấm double click chuột lên đối tượng mà bạn muốn tạo hàm để xử lý sự kiện đó. Tuy nhiên, cách này thì VS .Net nó chỉ phát sinh 1 hàm xử lý sự kiện mặc định của nó (với form thì hàm mặc định xử lý sự kiện là Form_Load. Do vậy, bạn hãy mở cửa sổ properties của đối tượng đó, chọn tab các events, nó sẽ liệt kê các events mà đối tượng có như hình dưới đây, sau đó bạn double click vào sự kiện mà bạn muốn, và một hàm xử lý sự kiện được tạo ở bên phần source code và bạn chỉ việc viết thêm code để xử lý sự kiện này mà không cần quan tâm đến bằng cách nào mà hệ thống kiểm tra và xử lý nó.

6.1.8. Chọn startup form

Khi một ứng dụng có nhiều form, vấn đề ở chỗ là form nào sẽ là form được chạy đầu tiên khi bạn run một ứng dụng?

Để chỉ định StartUp Form của chương trình, bạn cần phải mở cửa sổ Properties của Project để đánh vào **Startup Object**. Bạn có thể làm điều ấy bằng cách dùng IDE menu command **Project | Properties** hay right click tên của Project trong Solution Explorer rồi chọn **Properties**.

Lưu ý:

Vì trong 1 ứng dụng, thường chỉ có 1 form làm **entry form** (startup form) nên VS .Net sẽ tự set form đầu tiên của nó là form startup. Do đó, khi bạn thêm các form sau thì form đó sẽ không là form startup nên khi bạn startup object, nó sẽ

không list ra cho bạn. Để có thể làm form startup, bạn phải thay đổi code ở bên trong bằng cách chép đoạn mã này vào trong phần code:

```
static void Main()
{
    Application.Run(new Form1());
}
```

Và thay tên Form (ở ví dụ trên là Form1) bằng tên form mà bạn muốn làm startup form.

Ví dụ:

Hãy sử dụng Visual Studio .Net để tạo 1 project là **FormLarger** và tạo 1 form có giao diện giống như hình dưới và viết đoạn code để xử lý sự kiện khi click chuột lên trên form (bất kỳ chỗ nào) thì 1 message box sẽ hiển thị và form sẽ lớn thêm 50 pixels.

với các thuộc tính:

Caption	Click OK, Form lớn ra!!!
FormBorderStyle	FixDialog
MinimizeBox	false
MaximizeBox	false

Bạn hãy thay đổi màu nền và thay đổi các đường viền (border) để nhận thấy được sự khác nhau giữa các style của form.

Dưới đây là đoạn code để xử lý khi bạn click chuột lên trên form:

```
private void Form1_Click(object sender, System.EventArgs e)
{
    MessageBox.Show("Bạn mới click chuột lên trên form!\n" +
        "Click Ok và form sẽ lớn hơn 50 pixels.");
    this.Height = this.Height + 50;
    this.Width = this.Width + 50;
}
```

bạn có thể không cần xuống hàng ở câu lệnh MessageBox.Show()

Lưu ý: Nếu bạn đánh tiếng việt trên form (unicode) thì khi bạn save, nó sẽ yêu cầu bạn chọn save unicode.

Để làm việc này bạn hãy:

Click chuột vào form cần save, chọn menu **file -> Save <tên form> As...**

bạn hãy đánh vào ô file name là **frmNhapSV** và chọn **save with Encoding...** và hộp hội thoại sẽ xuất hiện:

bạn hãy chọn **Unicode (UTF-8 with signature) – Codepage 65001** và chọn **OK**.

Trong lập trình visual, form bao giờ cũng cần phải có tên để lập trình (form name) và tên form để lưu trên đĩa. Để cho dễ dàng trong lập trình và quản lý, ta thường hay lấy tên form để lập trình và tên form lưu trên đĩa giống nhau. Điều này giúp ta có thể chỉ biết được là form này có tên là gì mà không cần phải mở của sổ properties của nó lên.

6.2 Winform control cơ bản

Button control là một đối tượng nút nhấn được các nhà phát triển ngôn ngữ lập trình sẵn và để trong thư viện của Visual Studio .Net. Button control cho phép người dùng click chuột vào nó và nó sẽ thực thi một hay nhiều đoạn mã nào đó mà cho người lập trình (chúng ta) chỉ định thông qua các events mà nó nhận được.

Khi một button được click thì nó sẽ sinh ra một số các sự kiện (events) như Mouse Enter, MouseDown, MouseUp, v.v... và tùy với các sự kiện này mà chúng ta có thể lựa viết các đoạn code để xử lý cho phù hợp.

6.2.1. Tạo 1 nút nhấn (button) trên form

Cách tạo nút nhấn rất đơn giản, ta chỉ việc kéo đối tượng button trên hộp công cụ (toolbox) vào trong form, sau đó ta thay đổi tên (name), text (caption) và kích thước của đối tượng đó cho phù hợp với yêu cầu.

Lưu ý:

Phần lớn những cách tạo đối tượng, thay đổi của thuộc tính, v.v... mọi thứ mà ta làm bằng giao diện visual đều có thể làm bằng cách viết source code. Khi ta làm bằng công cụ visual, công cụ visual sẽ tự genera code ra cho ta.

6.2.2. Các thuộc tính (properties)

Cũng như window form, button cũng có các thuộc tính riêng của nó. Tuy nhiên vì mỗi đối tượng có rất nhiều thuộc tính, vì vậy ở đây ta chỉ đề cập đến một số thuộc tính quan trọng và hay sử dụng nhiều nhất của đối tượng button. Dưới đây là một số thuộc tính thông dụng của đối tượng button:

Tên Thuộc tính	Giá Trị	Diễn giải
Name	Text	Tên cho button
Text	Text	Chữ sẽ hiển thị trên button
Font	Hộp hội thoại font	Chọn font chữ cho text (chữ hiển thị trên button)
BackColor	Hộp màu	Chọn màu nền cho button
ForeColor	Hộp màu	Chọn màu chữ cho button.
TextAlign	TopLeft – TopCenter – TopRight	Canh lề chữ: trên bên trái – giữa phải.
	MiddleLeft – MiddleCenter –	ở giữa – bên trái – bên phải

	MiddleRight	
	BottomLeft – BottomCenter – BottomRight	bên dưới trái – giữa – phải
FlatStyle	Flat	Button sẽ bằng phẳng
	Popup	Giống Flat nhưng khi move mouse lên trên nó thì giống standard và khi bỏ mouse ra thì nó trở lại như cũ (Flat)
	Standard	Mặc định của hệ thống
	System	Giống standard nhưng sẽ thay đổi theo hệ điều hành
BackGroundImage	đường dẫn file hình	Hình bạn chỉ định sẽ làm nền cho button. Nếu hình không đủ lớn bằng button thì hình sẽ lặp lại (giống như bạn chọn chế độ Title khi bạn set background cho màn hình window).
Image	đường dẫn file hình	Hình sẽ làm nền cho button (giống như bạn chọn Center khi set hình background cho window).

Lưu ý:

Ở thuộc tính TextAlign, khi bạn bấm vào mũi tên sổ xuống trong cửa sổ properties, nó sẽ hiện ra 1 giao diện đồ họa cho ta chọn các kiểu canh lề như hình bên dưới.

Khi bạn chọn nút nào thì nút đó sẽ lõm xuống, đồng thời một dòng chữ sẽ hiển thị ra cho bạn biết là bạn đã chọn kiểu canh lề nào.

6.2.3. Thuộc tính Anchoring

Anchoring (bỏ neo) là thuộc tính rất tiện dụng mà Visual Studio .Net mới đưa vào. Nó cho phép ta định vị trí của một control trên form khi ta resize một form.

Anchoring:

Khi con tàu bỏ neo là nó đổ ở đó. Dù con nước chảy thế nào, con tàu vẫn nằm yên một chỗ vì nó đã được cột vào cái neo. Control trong .NET có property **Anchor** để ta chỉ định nó được buộc vào góc nào của form: **Left**, **Right**, **Bottom** hay **Top**.

Trong lúc thiết kế, sau khi select cái control (thí dụ Button1), ta vào cửa sổ Properties và click hình tam giác nhỏ bên phải property Anchor. Một hình vuông với bốn thanh ráp lại giống hình chữ thập màu trắng sẽ hiện ra. Mỗi thanh tượng trưng cho một góc mà ta có thể chỉ định để cột control vào form. Khi ta click một thanh, nó sẽ đổi màu thành xám đậm, và một chữ tương ứng với thanh ấy sau này sẽ hiển thị trong textbox area của combobox Anchor.

Khi Button1 có Anchor là Bottom, Right thì mỗi khi góc phải dưới của form di chuyển vì resize, Button1 cứ chạy theo góc ấy:

Nếu Button1 có Anchor là Left, Right, Bottom thì khi form resizes cho lớn ra, Button1 cứ giữ khoảng cách từ nó đến ba cạnh Left, Right, Bottom của form không đổi. Do đó nó phải nở rộng ra như trong hình dưới đây:

Nếu Button1 có Anchor là Top, Bottom, Left, Right thì khi form resizes, Button1 cứ giữ khoảng cách từ nó đến bốn cạnh Left, Right, Top, Bottom của form không đổi. Do đó nó phải nở rộng hay thu nhỏ cả chiều cao lẫn chiều rộng như trong hình dưới đây:

Vì property Anchor có hiệu lực lập tức ngay trong lúc ta thiết kế, nên nếu bạn resize form trong lúc thiết kế, các control có Anchor property set cũng resize và di chuyển theo. Có thể bạn không muốn chuyện đó xảy ra, nên tốt nhất là set property Anchor của các control sau khi thiết kế form xong hết rồi.

6.2.4. Các sự kiện của button

Trong các sự kiện của button thì chỉ có sự kiện Click chuột là quan trọng nhất. Do đó, trong phần này ta cũng chỉ đi tìm hiểu sự kiện Click chuột.

Cách tạo:

Sự kiện click chuột là sự kiện mặc định của control button, do đó, bạn chỉ cần double click chuột vào button cần tạo sự kiện là VS .Net sẽ mở cửa sổ source code ra và tự động generate một hàm xử lý sự kiện click chuột cho bạn.

```
private void button1_Click(object sender, System.EventArgs e)
{
    //Bạn sẽ đánh code cho phần xử lý sự kiện ở đây.
}
```

với **button1_Click** thì **button1** chính là tên của control button mà bạn tạo sự kiện Click chuột cho nó.

Ví dụ:

VD1. Hãy tạo 1 project có tên là Button và thiết kế form có 1 button giống như hình bên dưới:

Với các thuộc tính:

Đối Tượng	Thuộc Tính	Giá Trị
Form	Text Name	Button Mặc định
Button	Text Name	Click chuột vào đây! Mặc định.

Và tạo một sự kiện click chuột cho button này.

Bạn double click chuột vào button, Visual Studio sẽ sinh ra cho bạn một hàm để xử lý sự kiện click chuột vào button. Vì sự kiện Click là sự kiện mặc định của button nên khi bạn double click vào thì sự kiện click cho button đó sẽ được tự động sinh ra. Sau đó đánh thêm vào đoạn code sau:

```
private void button1_Click(object sender, System.EventArgs e)
{
    MessageBox.Show("Xin chào các bạn!");
}
```

MessageBox là đối tượng hiển thị các thông báo trên màn hình.

Lưu ý:

– Khi bạn muốn tạo 1 sự kiện cho một control nào đó, bạn chọn event trong cửa sổ properties của đối tượng đó, và Visual Studio sẽ tự động chuyển sang màn hình source code và 1 hàm xử lý sự kiện được tự động tạo ra cho bạn. Tuy nhiên, hệ thống nó sẽ sinh ra nhiều đoạn mã khác nhưng nó được cất dấu ở trong phần riêng của form và được ẩn đi. Hình dưới:

+ Windows Form Designer generated code

Bạn có thể xem đoạn code bên trong bằng cách bấm chuột vào dấu +. Bạn có thể thay đổi source code này theo ý mình nhưng tôi khuyên bạn không nên chỉnh sửa nó trừ khi bạn biết rõ mình sửa nó để làm gì và với mục đích gì.

– Như đã trình bày ở trên, khi bạn đánh vào font chữ viết sự dụng font unicode, bạn phải lưu form ở chế độ save as unicode (xem lại trên).

Bạn hãy chạy chương trình bằng cách bấm phím F5 để xem kết quả.

VD2. Hãy tạo 1 form gồm 2 button giống như hình dưới:

Khi button Xanh được bấm thì màu nền của form sẽ đổi sang màu xanh, button Đỏ được bấm thì màu nền của form sẽ đổi sang màu đỏ, và button mặc định được bấm thì màu nền của form lại trở về như lúc đầu.

Các thuộc tính:

Đối Tượng	Thuộc Tính	Giá Trị
Form	Name Text	frmMauNen Button Đổi Màu Nền Cho Form
Button	Name Text	btnXanh Xanh
Button	Name Text	btnDo Đỏ
Button	Name Text	btnMacDinh Mặc Định

Dưới đây là source code:

– Cho button btnXanh

```
private void btnXanh_Click(object sender, System.EventArgs e)
{
    this.BackColor = Color.Blue;
}
```

– Cho button btnDo

```
private void btnDo_Click(object sender, System.EventArgs e)
{
    this.BackColor = Color.Red;
}
```

– Cho button btnMacDinh

```
private void btnMacDinh_Click(object sender, System.EventArgs e)
{
    this.BackColor = Color.Empty;
}
```

Giải thích: Các tên hàm và các tham số là do Visual Studio .Net tự nó generate ra cho chúng ta nên ta chỉ xét các code bên trong.

this : chỉ đối tượng chính nó (nó giống như **Me** trong VB6). Ở đây nó chính là form frmMauNen.

Color: là đối tượng màu được lưu sẵn trong thư viện của VS .Net

Blue, Red: là hằng số về màu có sẵn trong thư viện của VS .Net

this.BackColor: ta thấy this là đối tượng, BackColor chính là thuộc tính.

Và tương tự với những đối tượng và thuộc tính khác.

Ghi chú:

– *Trong lập trình visual, các đối tượng đều có tên riêng của nó. Vì vậy để cho dễ dàng trong việc quản lý và lập trình, người ta thường đưa ra quy ước các tên của đối tượng. Bảng dưới đây mô tả cách đặt tên cho một số các control thông dụng mà ta thường sử dụng trong lập trình giao diện.*

6.4 Data Binding

Một Binding bao gồm 4 thành phần chính là: binding target, target property, binding source và một path (đường dẫn) đến giá trị cần thiết trong binding source, thông thường path này là một source property.

Ví dụ bạn muốn gán property Name của một đối tượng Person cho property Text của một TextBox. Khi đó:

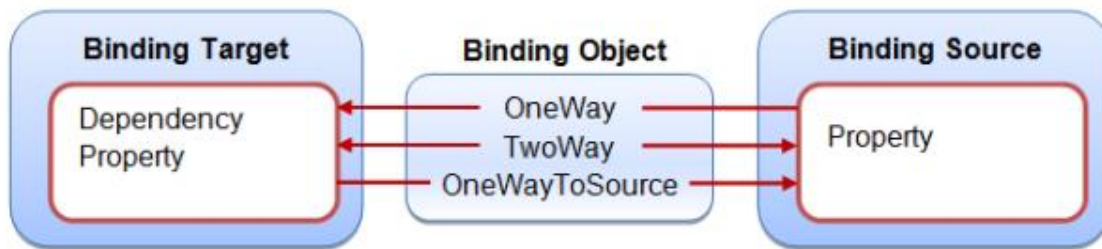
– Binding target: TextBox

– Target property: property Text của TextBox

– Binding source: đối tượng Person

-Path: đường dẫn đến property Name của đối tượng Person.

Mô hình Data Binding của WPF theo hình minh họa sau:



Cần lưu ý là target property phải là một dependency property. Đa số các property của lớpUIElement đều là các dependency property. Đối với binding source, bạn có thể sử dụng bất kì đối tượng .NET nào, chẳng hạn như các đối tượng trong ADO.NET, XML hay các control trong WPF.

Binding Mode

Binding mode sẽ chỉ ra hướng mà dữ liệu sẽ được cập nhật. Bao gồm 5 giá trị từ enum BindingMode là:

Name	Description
OneWay	Cập nhật target property theo source property
TwoWay	Cập nhật hai chiều giữa target property và source property.
OneTime	Khởi tạo target property từ source property. Sau đó việc cập nhật dữ liệu sẽ không được thực hiện.
OneWayToSource	Giống OneWay nhưng theo hướng ngược lại: cập nhật từ target property sang source property.
Default	Hướng binding dựa trên target property. Với target property mà người dùng có thể thay đổi giá trị (như TextBox.Text) thì nó là TwoWay, còn lại là OneWay

Ví dụ

Giả sử tôi muốn cập nhật nội dung của một Label theo giá trị được nhập vào TextBox. Cửa sổ minh họa cho ví dụ này cần hai control chính là textBox1 và label1:

```

1  <StackPanel>
2      <TextBox x:Name="textBox1">Sample Text</TextBox>
3      <Label x:Name="label1"/>
4  </StackPanel>

```

Bản chất của việc tạo binding bao gồm 2 bước:

- Tạo một đối tượng System.Windows.Data.Binding và thiết lập các giá trị cần thiết.
- Gọi phương thức instance FrameworkElement.SetBinding() của target binding. FrameworkElement được thừa kế từ UIElement và là lớp cha của các

control trong WPF. Phương thức này có tham số đầu tiên là một dependency property.

Ta tạo một binding với source binding là textBox1, target property là TextBox.Text, source binding là label1 và source property là Label.Content.

```
1 Binding binding = new Binding();
2 binding.Source = textBox1; // or binding.ElementName = "textBox1";
3 binding.Path= new PropertyPath("Text");
4 binding.Mode = BindingMode.OneWay;
```

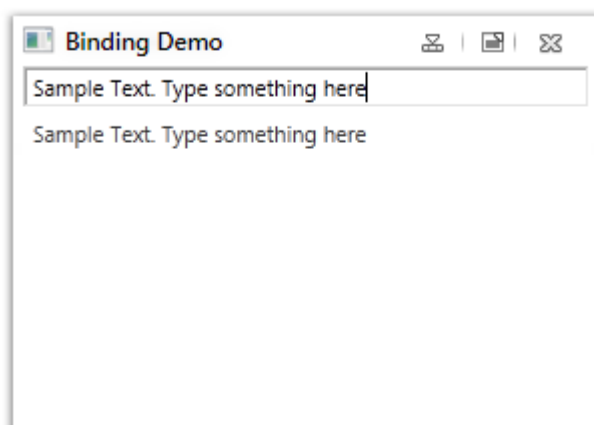
Thay vì phải tạo binding trong code-behind, bạn có thể tạo trong XAML theo cách thông thường sau:

```
1 <StackPanel>
2   <TextBox x:Name="textBox1">Sample Text</TextBox>
3   <Label x:Name="label1">
4     <Label.Content>
5       <Binding ElementName="textBox1" Path="Text" Mode="OneWay"/>
6     </Label.Content>
7   </Label>
8 </StackPanel>
```

Tuy nhiên, XAML còn hỗ trợ một dạng cú pháp gọn hơn với cùng chức năng như đoạn mã trên:

```
1 <StackPanel>
2   <TextBox x:Name="textBox1">Sample Text</TextBox>
3   <Label x:Name="label1"
4     Content="{ Binding ElementName=textBox1, Path=Text, Mode=OneWay}" />
5 </StackPanel>
```

Kết quả:



Update Source Trigger

Với Binding Mode là TwoWay hoặc OneWayToSource, bạn có thể xác định thời điểm mà binding source sẽ được cập nhật lại thông qua property Binding.UpdateSourceTrigger. Enum UpdateSourceTrigger gồm 4 giá trị:

Member name	Description
Default	Đa số các dependency property sẽ được dùng giá trị PropertyChanged, còn với property Text sẽ có giá trị là LostFocus.
PropertyChanged	Cập nhật binding source khi binding target property thay đổi.
LostFocus	Cập nhật binding source khi binding target mất focus.
Explicit	Cập nhật binding source chỉ khi bạn gọi phương thức UpdateSource.

Ví dụ sau sẽ tự động cập nhật property Text của hai TextBox với nhau ngay khi property này bị thay đổi:

```
1  <StackPanel>
2    <TextBox x:Name="textBox1">Sample Text</TextBox>
3    <TextBox x:Name="textBox2"
4      Text="{Binding ElementName=textBox1, Path=Text, Mode=TwoWay, UpdateSourceTrigger=PropertyChanged}"
5  />
  </StackPanel>
```

DataContext Property

Khái niệm Data Context tương tự như Data Source, đây là một property của FrameworkElement dùng để lưu dữ liệu cho việc hiển thị lên UI. Khi sử dụng cho data binding, DataContext sẽ được gán bằng đối tượng binding source.

Để minh họa, tôi tạo một lớp Product gồm hai property đơn giản sau:

```
1  public class Product
2  {
3      public int ID { get; set; }
4      public string Name { get; set; }
5  }
```

Trong XAML, tôi tạo một đối tượng Product với tên là myProduct trong Window.Resources, gán đối tượng myProduct cho DataContext của StackPanel, sau đó binding hai giá trị ID và Name của đối tượng Product này vào hai TextBox với property Text:

```
1  <Window x:Class="WpfApplication1.Window1"
2      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
```

```

3      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4      xmlns:local="clr-namespace:WpfApplication1"
5      Title="Binding Demo" Height="300" Width="400">
6      <Window.Resources>
7          <local:Product
8              x:Key="myProduct" ID="123" Name="Microsoft Visual Studio" />
9      </Window.Resources>
10     <StackPanel DataContext="{ StaticResource myProduct}">
11         <TextBox Text="{ Binding Path=ID}" />
12         <TextBox Text="{ Binding Path=Name}" />
13     </StackPanel>
14 </Window>

```

Thay vì gán vào DataContext của StackPanel, bạn có thể sử dụng DataContext của Window. Các thành phần con vẫn có thể lấy được dữ liệu để sử dụng cho binding. Trong constructor của lớp Window1:

```
this.DataContext = new Product() { ID = 123, Name = "Microsoft Visual Studio" };
```

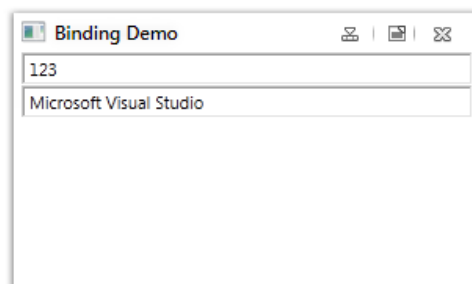
Và trong tài liệu XAML của Window1:

```

1 <Window x:Class="WpfApplication1.Window1"
2     xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
3     xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4     Title="Binding Demo" Height="300" Width="400">
5     <StackPanel>
6         <TextBox Text="{ Binding Path=ID}" />
7         <TextBox Text="{ Binding Path=Name}" />
8     </StackPanel>
9 </Window>

```

Cả hai cách làm này đều cho ra cùng kết quả:



6.5 Xử lý sự kiện

Các bước xử lý sự kiện trong C#

Bước 1: Tạo một delegate

Bước 2: Tạo một event liên kết đến delegate

```
access_modifier event delegate_name event_name;
```

Bước 3: Đăng ký sự kiện

```
event_name += method_name;
```

Bước 4: Phát sinh sự kiện

```
event_name();
```

Chương 7: ADO.NET

7.1 Dịch vụ truy cập dữ liệu ADO.NET

Trong thực tế, có rất nhiều ứng dụng cần tương tác với cơ sở dữ liệu. **.NET Framework** cung cấp một tập các đối tượng cho phép truy cập vào cơ sở dữ liệu, tập các đối tượng này được gọi chung là **ADO.NET**.

ADO.NET tương tự với **ADO**, điểm khác biệt chính ở chỗ **ADO.NET** là một kiến trúc dữ liệu rời rạc, không kết nối (**Disconnected Data Architecture**). Với kiến trúc này, dữ liệu được nhận về từ cơ sở dữ liệu và được lưu trên vùng nhớ cache của máy người dùng. Người dùng có thể thao tác trên dữ liệu họ nhận về

và chỉ kết nối đến cơ sở dữ liệu khi họ cần thay đổi các dòng dữ liệu hay yêu cầu dữ liệu mới.

Việc kết nối không liên tục đến cơ sở dữ liệu đã đem lại nhiều thuận lợi, trong đó điểm lợi nhất là việc giảm đi một lưu lượng lớn truy cập vào cơ sở dữ liệu cùng một lúc, tiết kiệm đáng kể tài nguyên bộ nhớ. Giảm thiểu đáng kể vấn đề hàng trăm ngàn kết nối cùng truy cập vào cơ sở dữ liệu cùng một lúc.

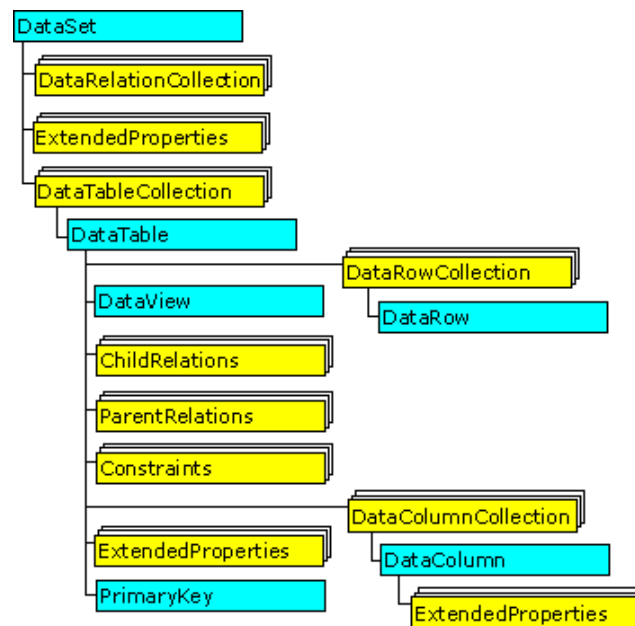
ADO.NET kết nối vào cơ sở dữ liệu để lấy dữ liệu và kết nối trở lại để cập nhật dữ liệu khi người dùng thay đổi chúng. Hầu hết mọi ứng dụng đều sử dụng nhiều thời gian cho việc đọc và hiển thị dữ liệu, vì thế **ADO.NET** đã cung cấp một tập hợp con các đối tượng dữ liệu không kết nối cho các ứng dụng để người dùng có thể đọc và hiển thị chúng mà không cần kết nối vào cơ sở dữ liệu.

Các đối tượng ngắt kết nối này làm việc tương tự đối với các ứng dụng Web.

7.2 Các thành phần cơ bản của ADO.NET

Có thể nói mô hình đối tượng của ADO.NET khá uyển chuyển, các đối tượng của nó được tạo ra dựa trên quan điểm đơn giản và dễ dùng. Đối tượng quan trọng nhất trong mô hình ADO.NET chính là Dataset. Dataset có thể được xem như là thể hiện của cả một cơ sở dữ liệu con, lưu trữ trên vùng nhớ cache của máy người dùng mà không kết nối đến cơ sở dữ liệu.

Mô hình đối tượng của Dataset



Mô hình đối tượng Dataset

DataSet bao gồm một tập các đối tượng **DataRelation** cũng như tập các đối tượng **DataTable**. Các đối tượng này đóng vai trò như các thuộc tính của **DataSet**.

Đối tượng DataTable và DataColumn

Ta có thể viết mã C# để tạo ra đối tượng DataTable hay nhận về từ kết quả của câu truy vấn đến cơ sở dữ liệu. DataTable có một số thuộc tính dùng chung (public) như thuộc tính Columns, từ thuộc tính này ta có thể truy cập đến đối

tượng DataColumnCollection thông qua chỉ mục hay tên của cột để nhận về các đối tượng DataColumn thích hợp, mỗi DataColumn tương ứng với một cột trong một bảng dữ liệu.

```
DataTable dt = new DataTable("tenBang"); DataColumn dc = dt.Columns["tenCot"];
```

Đối tượng DataRelation

Ngoài tập các đối tượng DataTable được truy cập thông qua thuộc tính Tables, DataSet còn có một thuộc tính Relations. Thuộc tính này dùng để truy cập đến đối tượng DataRelationCollection thông qua chỉ mục hay tên của quan hệ và sẽ trả về đối tượng DataRelation tương ứng.

```
DataSet ds = new DataSet("tenDataSet"); DataRelation dre = ds.Relations["tenQuanHe"];
```

Các bản ghi (Rows)

Tương tự như thuộc tính **Columns** của đối tượng DataTable, để truy cập đến các dòng ta cũng có thuộc tính Rows. ADO. NET không đưa ra khái niệm RecordSet, thay vào đó để duyệt qua các dòng (**Row**), ta có thể truy cập các dòng thông qua thuộc tính Rows bằng vòng lặp foreach.

Đối tượng SqlConnection và SqlCommand

Đối tượng **SqlConnection** đại diện cho một kết nối đến cơ sở dữ liệu, đối tượng này có thể được dùng chung cho các đối tượng SqlCommand khác nhau. Đối tượng SqlCommand cho phép thực hiện một câu lệnh truy vấn trực tiếp : như **SELECT**, **UPDATE** hay **DELETE** hay gọi một thủ tục (Store Procedure) từ cơ sở dữ liệu.

Đối tượng DataAdapter

ADO.NET dùng DataAdapter như là chiếc cầu nối trung gian giữa DataSet và DataSource (nguồn dữ liệu), nó lấy dữ liệu từ cơ sở dữ liệu sau đó dùng phương **Fill()** để đẩy dữ liệu cho đối tượng DataSet. Nhờ đối tượng DataAdapter này mà DataSet tồn tại tách biệt, độc lập với cơ sở dữ liệu và một DataSet có thể là thể hiện của một hay nhiều cơ sở dữ liệu. Ví dụ :

```
//Tạo đối tượng SqlDataAdapter
SqlDataAdapter sda = new SqlDataAdapter();
// cung cấp cho sda một SqlCommand và SqlConnection ...
// lấy dữ liệu ...
//tạo đối tượng DataSet mới
DataSet ds = new DataSet("tenDataSet");
//Đẩy dữ liệu trog sda vào ds bằng hàm Fill();
sda.Fill(ds);
```

7.3 LINQ và ADO.NET

Ngày nay, nhiều nhà phát triển kinh doanh phải sử dụng hai (hoặc nhiều) ngôn ngữ lập trình: ngôn ngữ cấp cao cho logic nghiệp vụ và các lớp trình bày (như Visual C # hoặc Visual Basic) và ngôn ngữ truy vấn để tương tác với cơ sở dữ liệu (như Transact -Có). Điều này đòi hỏi nhà phát triển phải thành thạo một số ngôn ngữ để có hiệu quả và cũng gây ra sự không phù hợp về ngôn ngữ trong môi trường phát triển. Ví dụ: một ứng dụng sử dụng API truy cập dữ liệu để thực hiện truy vấn đối với cơ sở dữ liệu chỉ định truy vấn dưới dạng chuỗi ký tự bằng cách sử dụng dấu ngoặc kép. Chuỗi truy vấn này không thể đọc được đối với trình biên dịch và không được kiểm tra lỗi, chẳng hạn như cú pháp không hợp lệ hoặc liệu các cột hoặc hàng mà nó tham chiếu có thực sự tồn tại hay không. Không có loại kiểm tra các tham số truy vấn và cũng không hỗ trợ IntelliSense .

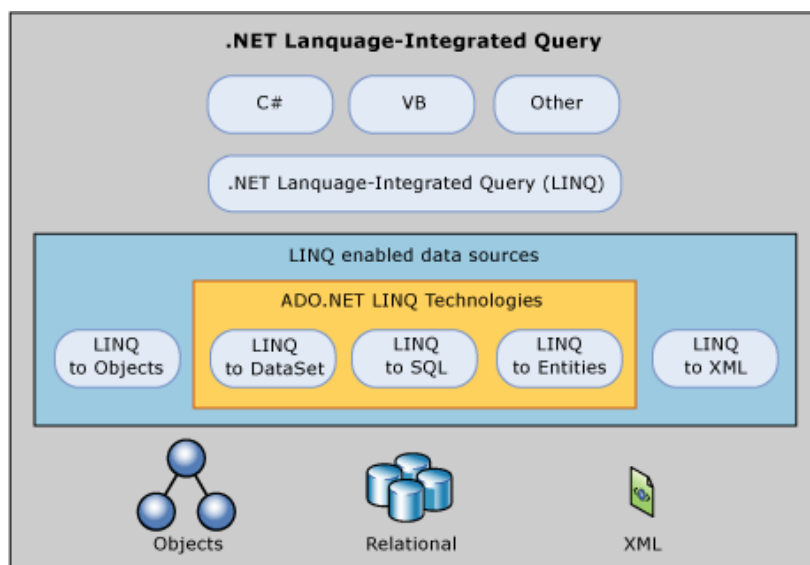
Truy vấn tích hợp ngôn ngữ (LINQ) cho phép các nhà phát triển hình thành các truy vấn dựa trên tập hợp trong mã ứng dụng của họ mà không phải sử dụng ngôn ngữ truy vấn riêng. Bạn có thể viết các truy vấn LINQ dựa trên nhiều nguồn dữ liệu khác nhau (nghĩa là nguồn dữ liệu thực hiện giao diện IEnumerable), chẳng hạn như cấu trúc dữ liệu trong bộ nhớ, tài liệu XML, cơ sở dữ liệu SQL và các đối tượng Dataset . Mặc dù các nguồn dữ liệu vô số này được thực hiện theo nhiều cách khác nhau, nhưng tất cả chúng đều phơi bày cùng một cấu trúc ngôn ngữ và ngôn ngữ. Vì các truy vấn có thể được hình thành trong chính ngôn ngữ lập trình, bạn không phải sử dụng ngôn ngữ truy vấn khác được nhúng dưới dạng chuỗi ký tự mà trình biên dịch không thể hiểu hoặc xác minh được. Việc tích hợp các truy vấn vào ngôn ngữ lập trình cũng cho phép các lập trình viên Visual Studio làm việc hiệu quả hơn bằng cách cung cấp loại thời gian biên dịch và kiểm tra cú pháp và IntelliSense . Các tính năng này làm giảm nhu cầu gỡ lỗi truy vấn và sửa lỗi.

Truyền dữ liệu từ các bảng SQL vào các đối tượng trong bộ nhớ thường rất dễ nhợt và dễ bị lỗi. Nhà cung cấp LINQ được LINQ triển khai cho Dataset và LINQ to SQL chuyển đổi dữ liệu nguồn thành các bộ sưu tập đối tượng dựa trên IEnumerable . Lập trình viên luôn xem dữ liệu là một bộ sưu tập IEnumerable , cả khi bạn truy vấn và khi bạn cập nhật. Hỗ trợ đầy đủ IntelliSense được cung cấp để viết các truy vấn đối với các bộ sưu tập đó.

Có ba công nghệ truy vấn tích hợp ngôn ngữ ADO.NET (LINQ): LINQ to Dataset, LINQ to SQL và LINQ to Entities. LINQ to Dataset cung cấp truy vấn tối ưu hóa, phong phú hơn cho Data set và LINQ to SQL cho phép bạn truy vấn

trực tiếp các lược đồ cơ sở dữ liệu SQL Server và LINQ to Entity cho phép bạn truy vấn Mô hình dữ liệu thực thể.

Sơ đồ sau đây cung cấp tổng quan về cách các công nghệ ADO.NET LINQ liên quan đến các ngôn ngữ lập trình cấp cao và các nguồn dữ liệu hỗ trợ LINQ.



Các phần sau đây cung cấp thêm thông tin về LINQ cho DataSet, LINQ to SQL và LINQ to Entities.

LINQ to DataSet

Bộ dữ liệu là thành phần chính của mô hình lập trình bị ngắt kết nối mà ADO.NET được xây dựng và được sử dụng rộng rãi. LINQ to DataSet cho phép các nhà phát triển xây dựng các khả năng truy vấn phong phú hơn vào DataSet bằng cách sử dụng cùng một cơ chế hình thành truy vấn có sẵn cho nhiều nguồn dữ liệu khác. Để biết thêm thông tin, xem LINQ to DataSet .

LINQ to SQL

LINQ to SQL là một công cụ hữu ích cho các nhà phát triển không yêu cầu ánh xạ tới mô hình khái niệm. Bằng cách sử dụng LINQ to SQL, bạn có thể sử dụng mô hình lập trình LINQ trực tiếp trên lược đồ cơ sở dữ liệu hiện có. LINQ to SQL cho phép các nhà phát triển tạo các lớp .NET Framework đại diện cho dữ liệu. Thay vì ánh xạ tới một mô hình dữ liệu khái niệm, các lớp được tạo này ánh xạ trực tiếp đến các bảng cơ sở dữ liệu, các khung nhìn, các thủ tục được lưu trữ và các hàm do người dùng định nghĩa.

Với LINQ to SQL, các nhà phát triển có thể viết mã trực tiếp vào lược đồ lưu trữ bằng cách sử dụng cùng một mẫu lập trình LINQ như các bộ sưu tập trong

bộ nhớ và Bộ dữ liệu , ngoài các nguồn dữ liệu khác như XML. Để biết thêm thông tin, xem [LINQ to SQL](#) .

LINQ cho các thực thể

Hầu hết các ứng dụng hiện đang được viết trên đầu cơ sở dữ liệu quan hệ. Tại một số điểm, các ứng dụng này sẽ cần phải tương tác với dữ liệu được biểu thị dưới dạng quan hệ. Các lược đồ cơ sở dữ liệu không phải lúc nào cũng lý tưởng để xây dựng các ứng dụng và các mô hình khái niệm của ứng dụng không giống như các mô hình logic của cơ sở dữ liệu. Mô hình dữ liệu thực thể là một mô hình dữ liệu khái niệm có thể được sử dụng để mô hình hóa dữ liệu của một miền cụ thể để các ứng dụng có thể tương tác với dữ liệu dưới dạng đối tượng. Xem [Khung thực thể ADO.NET](#) để biết thêm thông tin.

Thông qua Mô hình dữ liệu thực thể, dữ liệu quan hệ được hiển thị dưới dạng các đối tượng trong môi trường .NET. Điều này làm cho lớp đối tượng trở thành mục tiêu lý tưởng cho hỗ trợ LINQ, cho phép các nhà phát triển hình thành các truy vấn dựa trên cơ sở dữ liệu từ ngôn ngữ được sử dụng để xây dựng logic nghiệp vụ. Khả năng này được gọi là LINQ to Entities. Xem [LINQ to Entities](#) để biết thêm thông tin.

Chương 8: Giới thiệu lập trình webform và webform control

8.1 Nền tảng của webform

Web Forms là mô hình lập trình rất hiện đại, đơn giản, rõ ràng, linh động, hoạt động độc lập, không còn trộn code... Là một trong những công nghệ độc đáo của Microsoft dưới thời Bill Gates

Nói chung công nghệ nào cũng có điểm mạnh và yếu, **Web Forms** cũng không ngoại lệ. **Web Forms** dựng sẵn (built-in) cho người lập trình rất nhiều thứ (như: **Master Page**, **Server Controls**, **ViewState**, **PostBack...**), nhưng những cái này quá cứng nhắc, thiếu linh động, tự động phát sinh mã lung tung khó kiểm soát, không áp dụng được thực tế... nên đã làm nản lòng nhiều developer. Đây là cũng là điểm mà nhiều công nghệ khác muốn "**đìm hàng**" xoáy vào và cũng là điểm mà nhiều mạng xã hội tranh cãi nảy lửa. Ngay từ đầu bước vào **Web Forms** cách đây hơn 10 năm, tuy chưa học qua lớp lập trình **Web Forms** nào, chưa có bất kỳ tài liệu nào viết nhưng mình cũng nhận ra điểm yếu này và loại bỏ không sử dụng nó ngay từ đầu. Và nếu theo mô hình **Web Forms** thì không cần sử dụng và quan tâm đến mấy cái đó.

Cái điểm yếu và cũng là điểm mạnh thứ hai là: **Web Forms** là mô hình lập trình mở (không ràng buộc, để developer tự do sáng tạo, có thể dùng **Web Forms** để làm theo bất cứ mô hình nào khác). Nên nó gây khó cho những developer mới vào nghề (newbie), nhưng lại là điểm mạnh cho developer có kinh nghiệm (senior) tự do sáng tạo. Còn các mô hình khác thì đưa developer vào một cái khuôn, nên có sự đồng đều giữa newbie và senior. Và đây trở thành đề tài tranh luận không có hồi kết trên các mạng xã hội.

Tính hiện đại, chuyên nghiệp và đáng giá trong Web Forms:

- 1./ Web Forms được sự hỗ trợ của bộ thư viện ASP.NET và .Net Framework cực kỳ mạnh mẽ là điều được mọi người công nhận.
- 2./ Web Forms hỗ trợ Custom Control (User Control) cực kỳ mạnh mẽ, đây được xem như ánh sáng cuối đường hầm cho mô hình này.

Không giống như Custom Control của nhiều mô hình lập trình hay ngôn ngữ lập trình khác. Web Forms Custom Control hỗ trợ 2 cách gọi tĩnh và động với nhiều loại tham số (từ base types đến các template) giúp nó rất linh động, có khả năng tùy biến cao và hoạt động độc lập. Chúng ta chủ động mã HTML chứ không

còn chuyện tự động phát sinh mã HTML lộn xộn và khó kiểm soát như các built-in Server Controls.

Chúng ta có thể dùng cách gọi tĩnh để dùng Custom Control như một "Layout" hay "View" rất rõ ràng và chuyên nghiệp như ví dụ sau:

```
layout-view.ascx
<!DOCTYPE html>
<html lang="{lang_code}">
<head>
  <asc:header runat="server" />
</head>
<body>
  <asc:load name="banner" runat="server" />
  <asc:load name="menu" runat="server" />
  <asc:load name="header" runat="server" />
  <asc:load name="body" runat="server" />
  <asc:load name="footer" runat="server" />
</body>
</html>
```

```
news-view.ascx
<asc:news runat="server" />
<asc:news runat="server">
  <body>
    <!-- body template -->
  </body>
</asc:news>
<asc:news sort="news_order asc" runat="server">
  <header>
    <!-- header template -->
  </header>
  <body>
    <!-- body template -->
  </body>
  <detail>
    <!-- detail template -->
  </detail>
  <footer>
    <!-- footer template -->
  </footer>
  <empty>
    <!-- empty template -->
  </empty>
  <edit>
    <!-- edit template -->
  </edit>
</asc:news>
```

Với các tham số default template và custom template rất linh động và đầy đủ tính năng (xem danh sách, xem chi tiết, thêm, sửa, xóa, duyệt, copy, tìm kiếm, in ấn...) một cách tự động và chủ động mà không cần code gì thêm.

Như vậy, chúng ta chỉ cần làm duy nhất 1 control với không quá 300 dòng code là có thể xây dựng được 1 web tin tức nhanh chóng và chuyên nghiệp, đặt ở đâu cũng chạy, ở dự án nào cũng được, không cần phải sửa code hay rebuild, khả năng phát sinh lỗi cũng rất thấp nên giảm công test rất nhiều và rất dễ nâng cấp bảo trì...

Cũng tương tự như trên, chỉ cần 3 control là hoàn thành trên 30% tất cả các dự án, từ web tin tức, web thương mại điện tử hay web quản lý ERP.

Và Web Forms còn cung cấp 2 lệnh gọi động LoadControl và LoadTemplate giúp ta điều khiển rất linh hoạt. Tùy theo mỗi Context ta có thể chủ động load Layout nào, View nào cho phù hợp với từng Context chứ không cần đến coder khai báo.

3./ Web Forms là mô hình lập trình không trộn code, đây là điểm khác biệt và mạnh nhất của mô hình này. Các mã nguồn HTML, C#, CSS, Javascript, SQL... hoàn toàn độc lập và nằm trên các file khác nhau nên rất rõ ràng, ít phát sinh lỗi, dễ tái sử dụng và dễ nâng cấp bảo trì.

Chính điều này giúp chúng ta dễ dàng phát triển độc lập các loại mã nguồn trên một cách chuyên nghiệp (tức là khi xây dựng hay nâng cấp C# không ảnh hưởng đến database và ngược lại, designer có thể thiết kế database tùy ý mà không ảnh hưởng đến coder, cũng như việc thay đổi mã HTML cũng không ảnh hưởng đến các mã khác).

4./ Web Forms là mô hình tự chủ nên rất linh động. Không giống các mô hình lập trình khác theo cơ chế Bao cấp (cấp phát) hay Xin cho rất cứng nhắc. Mô hình Web Forms hoàn toàn tự chủ về tài nguyên và hoạt động nên rất linh động nhưng không mất kiểm soát.

5./ Tốc độ xử lý nhanh, ít sử dụng tài nguyên, bảo mật cao và url thân thiện. Thực chất, mấy cái này phụ thuộc vào developer đến 99%, chứ không phải là công nghệ. Ở Web Forms cũng có thể xây dựng web rất nhanh (response time < 10ms), sử dụng tài nguyên rất thấp (RAM < 30MB, CPU < 5%, HDD < 30MB...), độ bảo mật cao và url rất thân thiện và linh động như các mô hình khác. Nút thắt cổ chai là DB chứ thời gian render của code rất thấp (< 2ms), nên không đáng để bàn đến như nhiều mạng xã hội tranh luận.

Nói chung, mỗi công nghệ, mỗi mô hình đều có điểm yếu và mạnh riêng. Chúng ta đừng nên "dìm hàng" và đừng vội đánh giá khi chỉ mới nắm lý thuyết. Giữa lý thuyết với thực tế có một khoảng cách rất xa. Các nhà báo amateur về công nghệ đừng nên đánh giá linh tinh để giết chết những công nghệ rất hiện đại.

8.2 Tạo một ứng dụng webform

Để tạo một ứng dụng web bằng ASP.net cơ bản bạn cần có được những hiểu biết căn bản về ASP.net: kiến trúc của DotNet Framework cùng các phần mềm cần thiết khi sử dụng để tạo ra một ứng dụng web với ASP.net, cùng các ngôn ngữ lập trình được sử dụng trong ASP.net.

Cấu trúc của DotNet Framework: Gồm 5 phần chính.

Phần trên cùng Common Language specification : phần thể hiện các ngôn ngữ được hỗ trợ trong Framework như VB; C#; C++; Jscript...

Phần thứ 2 bao gồm: Windows Forms được sử dụng để xây dựng các ứng dụng trên Desktop; ASP.net là phần được dùng để xây dựng các ứng dụng web đây là phần mà chúng ta sẽ cùng nhau tìm hiểu sâu hơn.

Phần thứ 3 là: .Net Framework Base class bao gồm các lớp thư viện cơ bản trong DotNet: ADO.Net; XML; Threading IO; .Net; Security; Dlgagnostics; Etc.

Phần 4 gồm: Các dịch vụ chạy DotNet Common language và Runtime được sử dụng để chạy chương trình lập trình sau khi được trình biên dịch chuyển đổi

Phần cuối là các dịch vụ services và hệ điều hành nền tảng Windows.

Để có thể tạo lên một trang web với ASP.net bạn cần có các kỹ năng cơ bản về lập trình web như sau:

- + Kỹ năng lập C#, SQL Server, ngôn ngữ truy vấn T – SQL.
- + Hiểu về cơ chế hoạt động của WebSite và cơ sở dữ liệu.
- + Hiểu về kiến trúc và cơ chế của .Net Framework

Yêu cầu đối với máy được sử dụng để tạo một ứng dụng web với ASP.net như sau:

- Cài đặt .Net Framework 4.0 hoặc bản 4.5
- Cài Visual Stdio 2012 hoặc 2010.
- Cài SQL Server 2008.
- Internet Information Services.

Ngoài ra bạn cũng nên có thêm các kiến thức về PhotoShop và HTML để tạo ra các giao diện cho trang web.

Cách tạo ra một ứng dụng web trong ASP.net rất đơn giản và nhanh chóng các bạn chỉ cần mở trình làm việc của Visual Stdio lên. Ở bên phải màn hình chúng ta sẽ quan tâm tới hai cửa sổ chính là Solution Explorer và Server Explorer.



Giao diện làm lập trình web của Visual Studio 2010 hoặc 2010

Trong Solution Explorer chứa các tập tin thư mục trong trang web mà các bạn định tạo.

Trong Server Explorer sẽ cung cấp cho bạn các công cụ để kết nối với cơ sở dữ liệu SQL.

Ở bên trái sẽ chứa các Toolbox như: Textbox, Button... để các bạn phát triển các ứng dụng web.

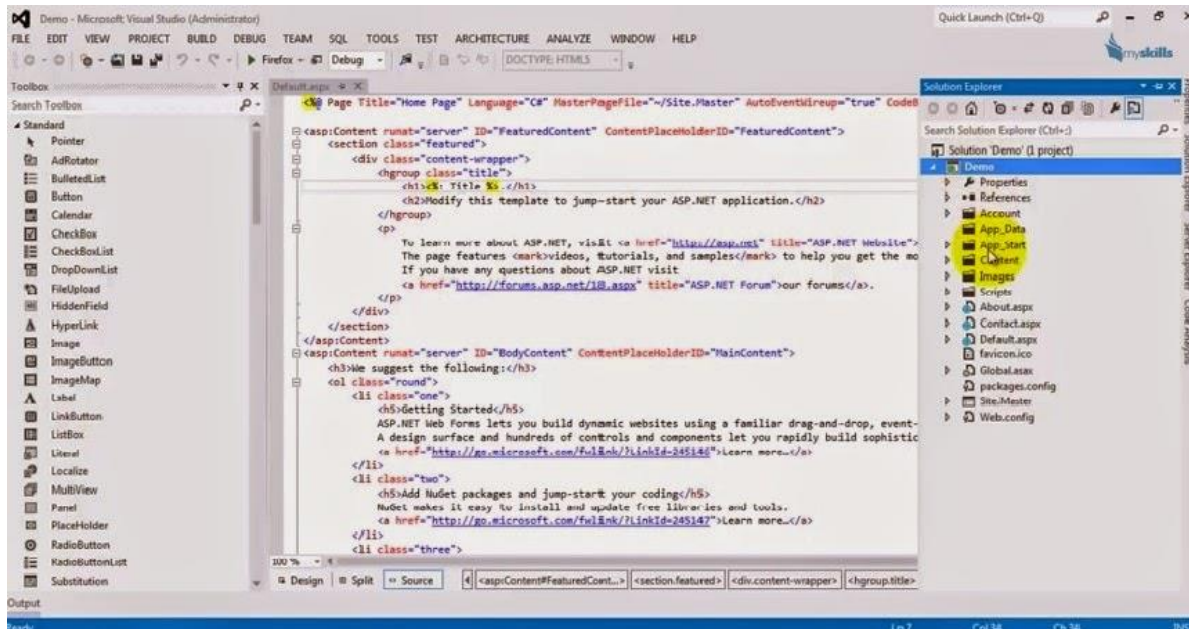
Để tạo một ứng dụng web các bạn vào File chọn New Project -> hiển thị cửa sổ bao gồm nhiều ngôn ngữ được hỗ trợ: Visual Basic; Visual C#, ...

Bạn click vào Visual C# -> sẽ ra các lựa chọn: Window để xây dựng các ứng dụng trên Windows; Web để xây dựng các ứng dụng web...

Khi khởi tạo một ứng dụng web với ASP.net bạn cần chú ý tới mục sổ chọn trên combobox phía trên phải đặt là .NET Framework 4 và Sort by là: Default.

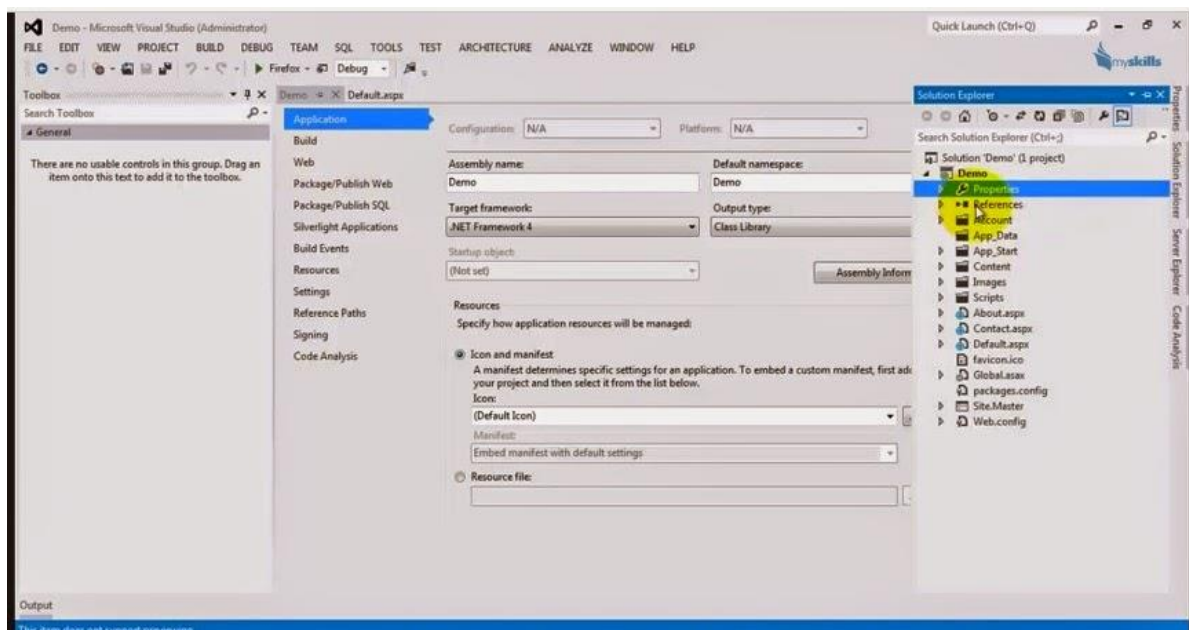
Chọn web -> chọn ASP.net Web Form Application -> nhập tên trang web mà bạn định tạo -> nhấn Ok

Một trình soạn thảo sẽ được hiển thị bao gồm các dòng code thể hiện cấu trúc của trang web mà bạn vừa tạo với các ngôn ngữ được hỗ trợ: HTML, CSS, PHP.



Trình soạn thảo code và các công cụ hỗ trợ làm web bằng ASP.net

Ở đây bạn có thể xem mọi thông tin về trang web mà bạn vừa tạo ra như trong phần projecties và các thư mục chứa các thư viện hệ thống. Bên phải là hộp Toolbox chứa các công cụ được dùng để tạo các form trên web.



Các thông tin bên trong của phần projecties

Như vậy, việc tạo ra một ứng dụng web bằng ASP.net trên Visual Studio là điều rất dễ dàng bạn còn việc viết thêm những đoạn code mà mình cần hiển thị trên trang web nữa là ổn.

Để chạy thử ứng dụng trực tiếp trên thanh công cụ chọn Firefox.

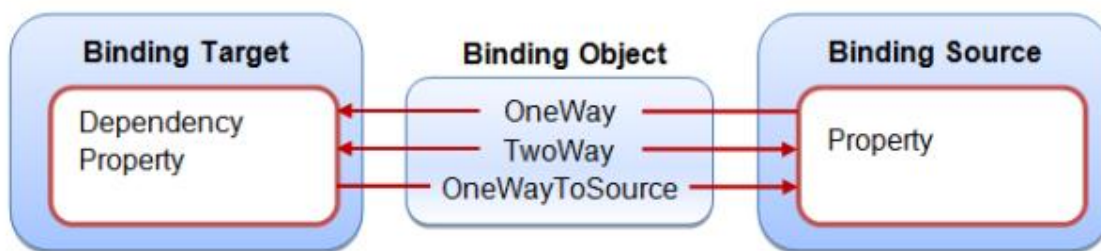
8.3 Data Binding

Một Binding bao gồm 4 thành phần chính là: binding target, target property, binding source và một path (đường dẫn) đến giá trị cần thiết trong binding source, thông thường path này là một source property.

Ví dụ bạn muốn gắn property Name của một đối tượng Person cho property Text của một TextBox. Khi đó:

- Binding target: TextBox
- Target property: property Text của TextBox
- Binding source: đối tượng Person
- Path: đường dẫn đến property Name của đối tượng Person.

Mô hình Data Binding của WPF theo hình minh họa sau:



Cần lưu ý là target property phải là một dependency property. Đa số các property của lớpUIElement đều là các dependency property. Đối với binding source, bạn có thể sử dụng bất kỳ đối tượng .NET nào, chẳng hạn như các đối tượng trong ADO.NET, XML hay các control trong WPF.

Binding Mode

Binding mode sẽ chỉ ra hướng mà dữ liệu sẽ được cập nhật. Bao gồm 5 giá trị từ enum BindingMode là:

Name	Description
OneWay	Cập nhật target property theo source property
TwoWay	Cập nhật hai chiều giữa target property và source property.
OneTime	Khởi tạo target property từ source property. Sau đó việc cập nhật dữ liệu sẽ không được thực hiện.
OneWayToSource	Giống OneWay nhưng theo hướng ngược lại: cập nhật từ target property sang source property.

Default	Hướng binding dựa trên target property. Với target property mà người dùng có thể thay đổi giá trị (như TextBox.Text) thì nó là TwoWay, còn lại là OneWay
---------	--

Ví dụ

Giả sử tôi muốn cập nhật nội dung của một Label theo giá trị được nhập vào TextBox. Cửa sổ minh họa cho ví dụ này cần hai control chính là textBox1 và label1:

```
1 <StackPanel>
2     <TextBox x:Name="textBox1">Sample Text</TextBox>
3     <Label x:Name="label1"/>
4 </StackPanel>
```

Bản chất của việc tạo binding bao gồm 2 bước:

- Tạo một đối tượng System.Windows.Data.Binding và thiết lập các giá trị cần thiết.
- Gọi phương thức instance FrameworkElement.SetBinding() của target binding. FrameworkElement được thừa kế từ UIElement và là lớp cha của các control trong WPF. Phương thức này có tham số đầu tiên là một dependency property.

Ta tạo một binding với source binding là textBox1, target property là TextBox.Text, source binding là label1 và source property là Label.Content.

```
1 Binding binding = new Binding();
2 binding.Source = textBox1; // or binding.ElementName = "textBox1";
3 binding.Path= new PropertyPath("Text");
4 binding.Mode = BindingMode.OneWay;
```

Thay vì phải tạo binding trong code-behind, bạn có thể tạo trong XAML theo cách thông thường sau:

```
1 <StackPanel>
2     <TextBox x:Name="textBox1">Sample Text</TextBox>
3     <Label x:Name="label1">
4         <Label.Content>
5             <Binding ElementName="textBox1" Path="Text" Mode="OneWay"/>
6         </Label.Content>
7     </Label>
8 </StackPanel>
```

Tuy nhiên, XAML còn hỗ trợ một dạng cú pháp gọn hơn với cùng chức năng như đoạn mã trên:

```
1 <StackPanel>
```

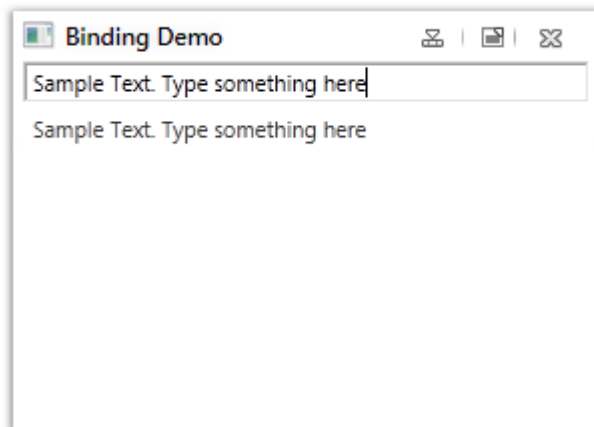


```

2      <TextBox x:Name="textBox1">Sample Text</TextBox>
3      <Label x:Name="label1"
4          Content="{Binding ElementName=textBox1, Path=Text, Mode=OneWay}" />
5  </StackPanel>

```

Kết quả:



Update Source Trigger

Với Binding Mode là TwoWay hoặc OneWayToSource, bạn có thể xác định thời điểm mà binding source sẽ được cập nhật lại thông qua property Binding.UpdateSourceTrigger. Enum UpdateSourceTrigger gồm 4 giá trị:

Member name	Description
Default	Đa số các dependency property sẽ được dùng giá trị PropertyChanged, còn với property Text sẽ có giá trị là LostFocus.
PropertyChanged	Cập nhật binding source khi binding target property thay đổi.
LostFocus	Cập nhật binding source khi binding target mất focus.
Explicit	Cập nhật binding source chỉ khi bạn gọi phương thức UpdateSource.

Ví dụ sau sẽ tự động cập nhật property Text của hai TextBox với nhau ngay khi property này bị thay đổi:

```

1  <StackPanel>
2      <TextBox x:Name="textBox1">Sample Text</TextBox>
3      <TextBox x:Name="textBox2"
4          Text="{Binding ElementName=textBox1, Path=Text, Mode=TwoWay, UpdateSourceTrigger=PropertyChanged}" />
5  </StackPanel>

```


</StackPanel>

DataContext Property

Khái niệm Data Context tương tự như Data Source, đây là một property của FrameworkElement dùng để lưu dữ liệu cho việc hiển thị lên UI. Khi sử dụng cho data binding, DataContext sẽ được gán bằng đối tượng binding source.

Để minh họa, tôi tạo một lớp Product gồm hai property đơn giản sau:

```
1 public class Product
2 {
3     public int ID { get; set; }
4     public string Name { get; set; }
5 }
```

Trong XAML, tôi tạo một đối tượng Product với tên là myProduct trong Window.Resources, gán đối tượng myProduct cho DataContext của StackPanel, sau đó binding hai giá trị ID và Name của đối tượng Product này vào hai TextBox với property Text:

```
1 <Window x:Class="WpfApplication1.Window1"
2     xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
3     xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4     xmlns:local="clr-namespace:WpfApplication1"
5     Title="Binding Demo" Height="300" Width="400">
6     <Window.Resources>
7         <local:Product
8             x:Key="myProduct" ID="123" Name="Microsoft Visual Studio" />
9     </Window.Resources>
10    <StackPanel DataContext="{ StaticResource myProduct}">
11        <TextBox Text="{ Binding Path=ID}" />
12        <TextBox Text="{ Binding Path=Name}" />
13    </StackPanel>
14 </Window>
```

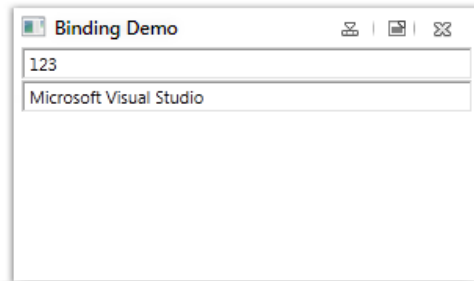
Thay vì gán vào DataContext của StackPanel, bạn có thể sử dụng DataContext của Window. Các thành phần con vẫn có thể lấy được dữ liệu để sử dụng cho binding. Trong constructor của lớp Window1:

```
this.DataContext = new Product() { ID = 123, Name = "Microsoft Visual Studio" };
Và trong tài liệu XAML của Window1:
```

```
1 <Window x:Class="WpfApplication1.Window1"
2     xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
3     xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4     Title="Binding Demo" Height="300" Width="400">
5     <StackPanel>
6         <TextBox Text="{ Binding Path=ID}" />
```

```
7      <TextBox Text="{ Binding Path=Name}" />
8  </StackPanel>
9  </Window>
```

Cả hai cách làm này đều cho ra cùng kết quả:



Tài liệu tham khảo

1. *Ngôn ngữ lập trình C# - Vietnam Open Educational Resources – VOER*
2. *Ngôn ngữ lập trình nâng cao - ĐH Thủy Lợi*
3. *Các giải pháp lập trình C# - Nhà sách Đất Việt*
4. *Khoa CNTT- Giáo trình Lập trình nâng cao- ĐH Kinh Doanh và Công Nghệ HN*