

**ĐẠI HỌC ĐÀ NẴNG
TRƯỜNG CAO ĐẲNG CÔNG NGHỆ THÔNG TIN
BỘ MÔN CÔNG NGHỆ**

**GIÁO TRÌNH
LẬP TRÌNH .NET**

**Dịch và tổng hợp: Lê Văn Minh
(Lưu hành nội bộ)**

ĐÀ NẴNG, 07/2008

LỜI NÓI ĐẦU

Giáo trình Lập trình .Net được biên soạn nhằm mục đích cung cấp những kiến thức cơ bản về lập trình trong môi trường .Net của Microsoft. Với giáo trình này sinh viên sẽ có được các kiến thức về lập trình để tạo ra các dạng ứng dụng khác nhau, bao gồm: ứng dụng dòng lệnh (Console Application), ứng dụng giao diện windows (Windows Application) và ứng dụng giao diện web (ASP.NET Website Application).

Vì đây là phiên bản đầu tiên nên giáo trình này còn có những thiếu sót, hạn chế. Tác giả mong nhận được sự góp ý quý báu từ phía bạn đọc. Mọi chi tiết xin liên lạc qua địa chỉ email: minhlevan@cit.udn.vn

Xin chân thành cảm ơn.

MỤC LỤC

VISUAL STUDIO VÀ .NET FRAMEWORK	1
I. Mục tiêu	1
II. .NET Framework	1
II.1. Khái niệm	1
II.2. Kiến trúc của .NET Framework	1
II.3. Các ngôn ngữ thuộc họ .NET	2
II.4. Các thư viện có sẵn của .Net Framework	2
III. Visual Studio	3
III.1. Khái niệm	3
III.2. Cách tổ chức chương trình của Visual Studio	3
III.3. Các dạng Project của Visual Studio	4
IV. Ngôn ngữ lập trình C#	4
IV.1. Khái niệm	4
IV.2. Đặc điểm	4
LẬP TRÌNH C# CĂN BẢN	6
I. Mục tiêu	6
II. Bắt đầu với Console Application	6
II.1. Tạo Project	6
II.2. Lập trình	8
II.3. Biên dịch	8
II.3.1. Biên dịch từng phần	8
II.3.2. Biên dịch toàn phần	9
II.4. Chạy chương trình	9
II.4.1. Chế độ debug	10
II.4.2. Chế độ non-debug	10
III. Biến và phạm vi hoạt động của biến trong C#	11
III.1. Biến	11
III.2. Phạm vi hoạt động của biến	11
IV. Hằng	12
V. Kiểu dữ liệu	12
V.1. Kiểu giá trị (Value Types)	13
V.1.1. Kiểu dữ liệu số nguyên	13
V.1.2. Kiểu dữ liệu dấu chấm động	13
V.1.3. Các kiểu dữ liệu khác	13
V.2. Kiểu tham chiếu (Reference Type)	13
VI. Cấu trúc điều kiện	14
VI.1. Câu lệnh điều kiện if..else	14
VI.1.1. Cú pháp	14
VI.1.2. Cách sử dụng	15
VI.2. Câu lệnh switch..case	15
VI.2.1. Cú pháp	15

VI.2.2.	<i>Cách sử dụng</i>	15
VII.	Cấu trúc lặp.....	16
VII.1.	Cấu trúc lặp for.....	16
VII.1.1.	<i>Cú pháp</i>	16
VII.1.2.	<i>Cách sử dụng</i>	16
VII.2.	Cấu trúc lặp while.....	17
VII.2.1.	<i>Cú pháp</i>	17
VII.2.2.	<i>Cách sử dụng</i>	17
VII.3.	Cấu trúc lặp do..while.....	17
VII.3.1.	<i>Cú pháp</i>	17
VII.3.2.	<i>Cách sử dụng</i>	17
VII.4.	Các lệnh hỗ trợ cho cấu trúc lặp.....	18
VII.4.1.	<i>Lệnh break</i>	18
VII.4.2.	<i>Lệnh continue</i>	18
VIII.	Mảng (Array).....	19
VIII.1.	Mảng một chiều.....	19
VIII.1.1.	<i>Cú pháp khai báo</i>	19
VIII.1.2.	<i>Cách sử dụng</i>	19
VIII.1.3.	<i>Cấu trúc lặp foreach</i>	19
VIII.2.	Mảng hai chiều (Ma trận).....	20
VIII.2.1.	<i>Cú pháp khai báo</i>	20
VIII.2.2.	<i>Cách sử dụng</i>	20
IX.	Xử lý nhập xuất file.....	21
IX.1.	Khái niệm file.....	21
IX.2.	Phân loại.....	21
IX.2.1.	<i>File văn bản (text file)</i>	21
IX.2.2.	<i>File nhị phân (binary file)</i>	22
IX.3.	Đọc và ghi file văn bản.....	22
IX.3.1.	<i>Đọc file văn bản bằng StreamReader</i>	22
IX.3.2.	<i>Ghi file văn bản bằng StreamWriter</i>	24
LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG VỚI C#.....		25
I.	Mục tiêu.....	25
II.	Lớp (Class).....	25
III.	Đối tượng (Object).....	27
IV.	Thuộc tính (Attribute).....	27
V.	Phương thức (Method).....	27
VI.	Nạp chồng toán tử (Overloading).....	28
VII.	Kế thừa (Inheritance).....	29
VIII.	Đa hình (Polymorphism).....	31
IX.	Interface.....	33
XỬ LÝ BIỆT LỆ.....		34
I.	Mục tiêu.....	34
II.	Biệt lệ (Exception).....	34
II.1.	Chương trình và lỗi.....	34

II.2.	Khái niệm biệt lệ	35
III.	Xử lý biệt lệ (Exception Handler)	35
III.1.	Cơ chế try/catch	35
III.2.	Xử lý biệt lệ lồng nhau	37
III.3.	Xử lý biệt lệ song song	40
THƯ VIỆN LIÊN KẾT ĐỘNG		42
I.	Mục tiêu	42
II.	Thư viện trong lập trình	42
II.1.	Khái niệm	42
II.2.	Phân loại thư viện	43
II.2.1.	<i>Thư viện tĩnh</i>	43
II.2.2.	<i>Thư viện liên kết động</i>	43
III.	Namespace	44
IV.	Thư viện liên kết động	44
IV.1.	Cách xây dựng thư viện với Visual Studio 2005	44
IV.1.1.	<i>Tạo một project cho thư viện</i>	44
IV.1.2.	<i>Cấu hình cho project</i>	45
IV.1.3.	<i>Xây dựng lớp và phương thức cần thiết</i>	46
IV.2.	Cách sử dụng thư viện	47
IV.2.1.	<i>Tạo thêm tham chiếu (add reference)</i>	47
IV.2.2.	<i>Khai báo tham chiếu</i>	48
IV.2.3.	<i>Sử dụng thư viện</i>	49
V.	Các namespace có sẵn của .Net Framework 2.0	49
V.1.	Namespace System.Windows.Forms	49
V.2.	Namespace System.Data	50
LẬP TRÌNH ỨNG DỤNG WINDOWS		51
I.	Mục tiêu	51
II.	Các bảng điều khiển	51
II.1.	Toolbox panel	52
II.2.	Solution Explorer panel	53
II.3.	Properties panel	54
III.	Form và Label	54
III.1.	Form	54
III.1.1.	<i>Khái niệm Form</i>	54
III.1.2.	<i>Các thuộc tính của Form</i>	54
III.1.3.	<i>Các sự kiện của Form</i>	55
III.2.	Label	56
III.2.1.	<i>Khái niệm Label</i>	56
III.2.2.	<i>Các thuộc tính của Label</i>	56
III.3.	Ứng dụng Form và Label	56
III.3.1.	<i>Tạo mới project</i>	56
III.3.2.	<i>Đổi tên Form chính</i>	57
III.3.3.	<i>Đặt tiêu đề cho Form</i>	58
III.3.4.	<i>Cài đặt sự kiện FormClosed</i>	59
III.3.5.	<i>Thêm Label vào Form</i>	60

III.3.6.	<i>Biên dịch và chạy ứng dụng</i>	62
IV.	TextBox và Button	62
IV.1.	TextBox	62
IV.1.1.	<i>Khái niệm TextBox</i>	62
IV.1.2.	<i>Các thuộc tính của TextBox</i>	62
IV.1.3.	<i>Các sự kiện của TextBox</i>	63
IV.2.	Button	63
IV.2.1.	<i>Khái niệm Button</i>	63
IV.2.2.	<i>Các thuộc tính của Button</i>	63
IV.2.3.	<i>Các sự kiện của Button</i>	64
IV.3.	Ứng dụng TextBox và Button	64
IV.3.1.	<i>Tạo project, tạo Form và các Label</i>	64
IV.3.2.	<i>Tạo các TextBox</i>	64
IV.3.3.	<i>Thêm các Button</i>	65
IV.3.4.	<i>Cài đặt sự kiện cho từng Button</i>	66
IV.3.5.	<i>Biên dịch và chạy chương trình</i>	67
V.	ComboBox, CheckBox, RadioButton	67
V.1.	ComboBox.....	67
V.1.1.	<i>Khái niệm ComboBox</i>	67
V.1.2.	<i>Các thuộc tính của ComboBox</i>	67
V.1.3.	<i>Các phương thức của ComboBox</i>	68
V.1.4.	<i>Các sự kiện của ComboBox</i>	68
V.2.	CheckBox	68
V.2.1.	<i>Khái niệm CheckBox</i>	68
V.2.2.	<i>Các thuộc tính của CheckBox</i>	69
V.2.3.	<i>Các sự kiện của CheckBox</i>	69
V.3.	RadioButton.....	69
V.3.1.	<i>Khái niệm RadioButton</i>	69
V.3.2.	<i>Các thuộc tính của RadioButton</i>	69
V.3.3.	<i>Các sự kiện của RadioButton</i>	69
V.4.	Ứng dụng ComboBox, CheckBox, RadioButton	69
V.4.1.	<i>Tạo project, tạo Form, tạo các Label và TextBox</i>	70
V.4.2.	<i>Tạo các RadioButton</i>	70
V.4.3.	<i>Tạo ComboBox</i>	71
V.4.4.	<i>Tạo các CheckBox</i>	72
V.4.5.	<i>Tạo Button</i>	73
V.4.6.	<i>Biên dịch và chạy chương trình</i>	73
VI.	MDI Form và MenuStrip	74
VI.1.	MDI Form.....	74
VI.1.1.	<i>Khái niệm MDI Form</i>	74
VI.1.2.	<i>Các thuộc tính của MDI Form</i>	75
VI.2.	MenuStrip	75
VI.2.1.	<i>Khái niệm MenuStrip</i>	75
VI.2.2.	<i>Các thuộc tính của MenuStrip</i>	75
VI.3.	ToolStripMenuItem	75
VI.3.1.	<i>Khái niệm ToolStripMenuItem</i>	75
VI.3.2.	<i>Các thuộc tính của ToolStripMenuItem</i>	75

VI.3.3.	<i>Các sự kiện của ToolStripMenuItem</i>	76
VI.4.	Ứng dụng MDI Form, MenuStrip.....	76
VI.4.1.	<i>Tạo project và cấu hình MDI Form</i>	76
VI.4.2.	<i>Tạo Form LogIn và Form Register</i>	76
VI.4.3.	<i>Tạo MenuStrip</i>	77
VI.4.4.	<i>Viết sự kiện cho từng ToolStripMenuItem</i>	77
TƯƠNG TÁC CƠ SỞ DỮ LIỆU		78
I.	Mục tiêu.....	78
II.	ADO.NET.....	78
II.1.	Khái niệm	78
II.2.	Kiến trúc	78
II.3.	Các namespace phục vụ cho ADO.NET.....	80
III.	SqlConnection, SqlCommand	80
III.1.	SqlConnection	80
III.1.1.	<i>Khái niệm SqlConnection</i>	80
III.1.2.	<i>Các thuộc tính của SqlConnection</i>	80
III.1.3.	<i>Các phương thức của SqlConnection</i>	81
III.2.	SqlCommand	81
III.2.1.	<i>Khái niệm SqlCommand</i>	81
III.2.2.	<i>Các thuộc tính của SqlCommand</i>	81
III.2.3.	<i>Các phương thức của SqlCommand</i>	82
III.3.	Ứng dụng SqlConnection, SqlCommand, ExecuteNonQuery	82
III.3.1.	<i>Tạo cơ sở dữ liệu</i>	82
III.3.2.	<i>Tạo bảng tblUser</i>	83
III.3.3.	<i>Tạo stored procedure</i>	84
III.3.4.	<i>Lập trình tương tác cơ sở dữ liệu</i>	85
III.3.5.	<i>Biên dịch và chạy ứng dụng</i>	86
IV.	SqlDataReader và phương thức ExecuteReader.....	87
IV.1.	SqlDataReader	87
IV.1.1.	<i>Khái niệm SqlDataReader</i>	87
IV.1.2.	<i>Các thuộc tính của SqlDataReader</i>	87
IV.1.3.	<i>Các phương thức của SqlDataReader</i>	87
IV.2.	Phương thức ExecuteReader	88
IV.3.	Ứng dụng SqlDataReader và phương thức ExecuteReader	88
V.	SqlDataAdapter, DataSet và DataGridView.....	89
V.1.	SqlDataAdapter	89
V.2.	DataSet	89
V.3.	DataGridView.....	90
V.3.1.	<i>Khái niệm DataGridView</i>	90
V.3.2.	<i>Các thuộc tính của DataGridView</i>	90
V.3.3.	<i>Các sự kiện của DataGridView</i>	90
V.4.	Ứng dụng SqlDataAdapter, DataSet, DataGridView	90
V.4.1.	<i>Tạo project</i>	90
V.4.2.	<i>Thêm đối tượng DataGridView</i>	90
V.4.3.	<i>Cài đặt sự kiện Load của Form</i>	91
V.4.4.	<i>Biên dịch và chạy chương trình</i>	92

KẾT LUẬN..... 93

PHỤ LỤC.....ERROR! BOOKMARK NOT DEFINED.

Visual Studio và .NET Framework

I. Mục tiêu

Trong chương này, người học sẽ được cung cấp một số kiến thức về các vấn đề sau:

1. Microsoft .Net Framework
2. Microsoft Visual Studio
3. Ngôn ngữ lập trình C#

II. .NET Framework

II.1. Khái niệm

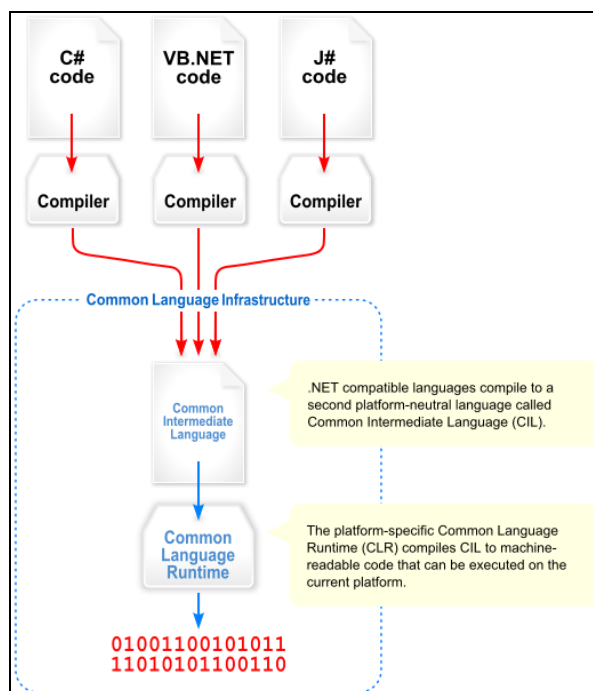
Microsoft .NET Framework là một thành phần có thể được cài thêm hoặc đã có sẵn trong các hệ điều hành Windows. Nó cung cấp những giải pháp đã được lập trình sẵn cho những yêu cầu thông thường của chương trình quản lý việc thực thi các chương trình viết trên framework, người dùng cuối cần phải cài framework để có thể chạy các chương trình được phát triển bằng các ngôn ngữ trong họ .NET. .NET Framework do Microsoft đưa ra và được sử dụng trong hầu hết các ứng dụng viết trên nền Windows. Những giải pháp được lập trình sẵn hình thành nên một thư viện các lớp của framework, được sử dụng trong nhiều lĩnh vực của lập trình như: giao diện người dùng, truy cập dữ liệu, kết nối cơ sở dữ liệu, mã hoá, phát triển những ứng dụng website, các giải thuật số học và giao tiếp mạng. Thư viện lớp của framework được lập trình viên sử dụng, kết hợp với chương trình của chính mình để tạo nên các ứng dụng.

II.2. Kiến trúc của .NET Framework

Microsoft tổ chức .NET Framework thành nhiều tầng, quá trình biên dịch và thực thi một chương trình viết trên nền .NET Framework được thực hiện từng bước từ phần mã nguồn đến phần mã máy.

Mã nguồn của chương trình sau khi biên dịch sẽ thành ngôn ngữ trung gian (Common Intermediate Language - CIL). Ngôn ngữ này biên dịch phần lớn các thư viện được viết trên nền .NET thành các thư viện liên kết động (Dynamic Linked Library - DLL). Với giải pháp này, các ngôn ngữ được .NET Framework hỗ trợ sẽ dễ dàng sử dụng lại lẫn nhau. Một chương trình được viết bằng ngôn ngữ C# có thể sử dụng các lớp, các thuộc tính đã được viết trước đó bằng ngôn ngữ VB.NET hoặc J#.

Tầng dưới cùng của cấu trúc phân tầng của .NET Framework là Common Language Runtime – còn được gọi là CLR. Đây là thành phần quan trọng nhất của .NET Framework. Tầng này thực hiện biên dịch mã của CIL thành mã máy và thực thi.



Hình 1 – Cấu trúc của .NET Framework

II.3. Các ngôn ngữ thuộc họ .Net

Hiện tại các lập trình viên có thể sử dụng nhiều ngôn ngữ khác nhau để lập trình, có người thân thiện với ngôn ngữ này, có người thân thiện với ngôn ngữ khác. Có người làm việc rất tốt với ngôn ngữ Basic, trong khi đó, một số người khác thân thiện với ngôn ngữ Java. Những lập trình viên với khả năng thông thạo những ngôn ngữ khác nhau dường như không thể cùng xây dựng một ứng dụng vì sự không tương thích giữa các mã lệnh biên dịch. Để khắc phục tình trạng này, Microsoft đã đưa ra giải pháp .Net Framework. Với .Net Framework, các lập trình viên có thể lập trình ở những ngôn ngữ khác nhau, sau khi biên dịch, kết quả thu được sẽ là các thư viện liên kết động .dll (dynamic linked library). Các thư viện này sẽ được các lập trình viên khác kế thừa và sử dụng lại.

Visual Studio và Microsoft .Net Framework hỗ trợ các ngôn ngữ lập trình: Visual C++, Visual Basic .NET, Visual C#, Visual J#. Các ngôn ngữ lập trình trên được gọi chung là họ ngôn ngữ .NET.

II.4. Các thư viện có sẵn của .Net Framework

Thư viện lớp cơ sở của .NET là một tập hợp lớn các lớp được viết bởi Microsoft, những lớp này cho phép bạn thao tác rất nhiều các tác vụ sẵn có trong Windows. Bạn có thể tạo các lớp của mình từ các lớp có sẵn trong thư viện lớp cơ sở của .NET dựa trên cơ chế thừa kế đơn.

Thư viện lớp cơ sở của .NET rất trực quan và rất dễ sử dụng. Ví dụ, để tạo một tiến trình mới, bạn đơn giản gọi phương thức Start() của lớp Thread. Để vô hiệu hóa một TextBox, bạn đặt thuộc tính Enabled của đối tượng TextBox là false. Thư viện này được thiết kế để dễ sử dụng tương tự với các ngôn ngữ như là Visual Basic và Java.

Các thư viện có sẵn .NET Framework bao gồm:

- Thư viện hỗ trợ Windows GUI và Controls
- Thư viện Web Forms (ASP.NET)
- Thư viện Data Access (ADO.NET)
- Thư viện Directory Access
- Thư viện File system và Registry access
- Thư viện Networking và Web browsing
- Thư viện .NET attributes và reflection
- Thư viện hỗ trợ truy xuất vào hệ điều hành Windows
- Thư viện COM interoperability

III. Visual Studio

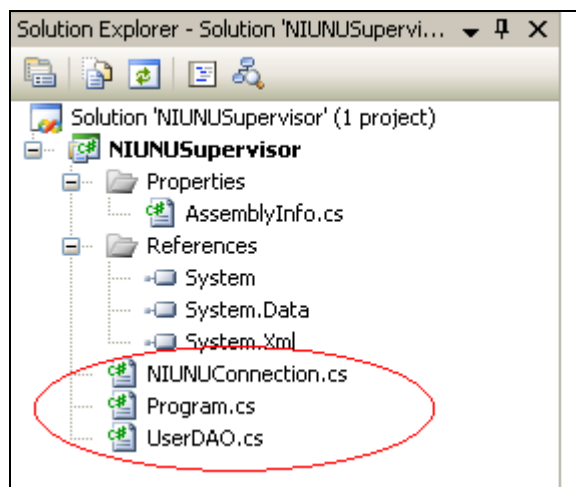
III.1. Khái niệm

Visual Studio .Net là môi trường tích hợp phát triển phần mềm (Integrated Development Environment (IDE)) của Microsoft, là công cụ cho phép bạn viết mã, gỡ rối và biên dịch chương trình trong nhiều ngôn ngữ lập trình .NET khác nhau.

III.2. Cách tổ chức chương trình của Visual Studio

Chương trình hoặc ứng dụng hoặc thậm chí là một hệ thống đều được Visual Studio tổ chức dưới dạng Solution. Solution là tập hợp của nhiều Project. Project là tập hợp các tập tin liên quan đến một ứng dụng và được người dùng tổ chức theo các cấp độ thư mục.

Một Project của Visual Studio thông thường bao gồm 3 phần: phần thuộc tính (Properties), phần tham chiếu (References), phần người dùng tự định nghĩa.



Hình 2 – Cấu trúc của một project của Visual Studio

Phần thuộc tính (Properties) chứa class AssemblyInfo trong đó mô tả các thông tin cơ bản về ứng dụng như: tên ứng dụng, tên công ty, địa chỉ công ty, bản quyền và các thông tin khác.

Phần tham chiếu (References) chứa các gói hoặc các class mà ứng dụng này cần dùng. Người dùng có thể sử dụng các gói và các class có sẵn của .NET Framework hoặc sử dụng các gói và class do người dùng định nghĩa. Các gói và class này có thể được xây dựng bằng nhiều ngôn ngữ khác nhau miễn là các ngôn ngữ này cùng thuộc họ .NET.

Phần người dùng định nghĩa là phần còn lại, người dùng có thể tự định nghĩa các gói, các lớp hoặc thêm vào một số file dữ liệu nếu cần.

III.3. Các dạng Project của Visual Studio

Hiện nay, một hệ thống thông tin thường có những dạng ứng dụng sau: Ứng dụng Console phục vụ xử lý các vấn đề liên quan đến hệ thống và giao tiếp vào ra; Ứng dụng Desktop phục vụ xây dựng các phần mềm ứng dụng với giao diện thân thiện; Ứng dụng Internet phục vụ việc xây dựng các website; Đối với mỗi dạng ứng dụng khác nhau, Visual Studio cung cấp các dạng Project khác nhau. Các dạng Project được Visual Studio cung cấp gồm có:

- Console Application: Cung cấp template cho ứng dụng Console
- Windows Application: Cung cấp template cho ứng dụng Desktop
- Class Library: Cung cấp template cho việc xây dựng thư viện liên kết động
- ASP.NET Website: Cung cấp template cho việc xây dựng Website
- ASP.NET Web Service: Cung cấp template cho việc xây dựng Web Service

IV. Ngôn ngữ lập trình C#

IV.1. Khái niệm

C# là một ngôn ngữ lập trình hướng đối tượng được phát triển bởi Microsoft. Microsoft phát triển C# dựa trên C++ và Java. C# được miêu tả là ngôn ngữ có được sự cân bằng giữa C++, Visual Basic, Delphi và Java. C# được thiết kế chủ yếu bởi Anders Hejlsberg kiến trúc sư phần mềm nổi tiếng với các sản phẩm Turbo Pascal, Delphi, J++, WFC.

IV.2. Đặc điểm

C# là ngôn ngữ lập trình phản ánh trực tiếp nhất đến .NET Framework mà tất cả các chương trình .NET chạy. C# phụ thuộc mạnh mẽ vào .Net Framework, mọi dữ liệu cơ sở đều là đối tượng, được cấp phát và hủy bỏ bởi trình dọn rác Garbage-Collector (GC). C# cung cấp nhiều kiểu trừu tượng khác chẳng hạn như class, delegate, interface, exception, v.v, phản ánh rõ ràng những đặc trưng của .NET runtime.

So sánh với C và C++, ngôn ngữ này bị giới hạn và được nâng cao ở một vài đặc điểm nào đó, nhưng không bao gồm các giới hạn sau đây:

- Các con trỏ chỉ có thể được sử dụng trong chế độ không an toàn. Hầu hết các đối tượng được tham chiếu an toàn, và các phép tính đều được kiểm tra tràn bộ đệm. Các con trỏ chỉ được sử dụng để gọi các loại kiểu giá trị; còn những đối tượng thuộc bộ thu rác (garbage-collector) thì chỉ được gọi bằng cách tham chiếu.
- Các đối tượng không thể được giải phóng tường minh.
- Chỉ có đơn kế thừa, nhưng có thể cài đặt nhiều interface trừu tượng (abstract interfaces). Chức năng này làm đơn giản hóa sự thực thi của thời gian thực thi.
- C# thì an-toàn-kiểu (typesafe) hơn C++.
- Cú pháp khai báo mảng khác nhau("int[] a = new int[5]" thay vì "int a[5]").

LẬP TRÌNH C# CĂN BẢN

I. Mục tiêu

Trong chương này giáo trình sẽ cung cấp cho bạn những kiến thức cơ bản nhất của ngôn ngữ lập trình C#. Những chủ đề chính như sau:

- Khai báo biến
- Khởi tạo và phạm vi hoạt động của biến
- Cách sử dụng các vòng lặp và câu lệnh
- Cách sử dụng mảng
- Namespaces và thư viện liên kết động
- Cơ bản trình biên dịch dòng lệnh trong C#
- Sử dụng gói System.Console để thực hiện việc nhập xuất
- Sử dụng chú thích trong C# và Visual Studio .NET

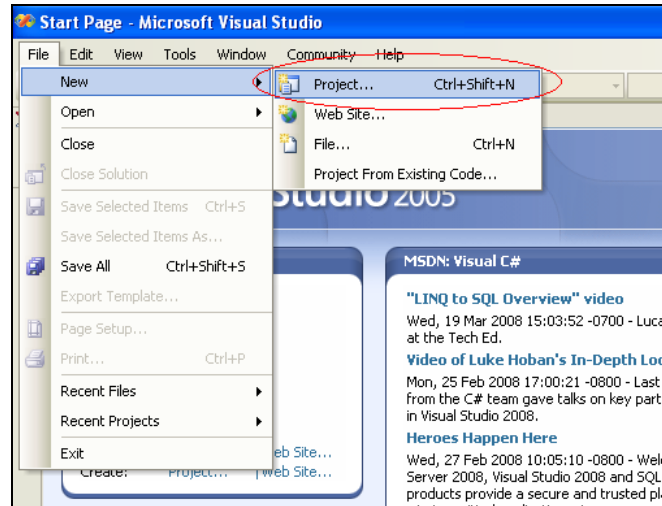
Cuối chương này người học sẽ có đủ khả năng viết một chương trình để giải quyết một bài toán đơn giản bằng C#.

II. Bắt đầu với Console Application

Console Application là dạng Project phục vụ việc lập trình với các ứng dụng đơn giản. Với dạng Project này chúng ta dễ dàng thực hiện việc lập trình để mô phỏng thuật toán và mô phỏng hướng đối tượng. Các bước để khởi tạo Console Application như sau:

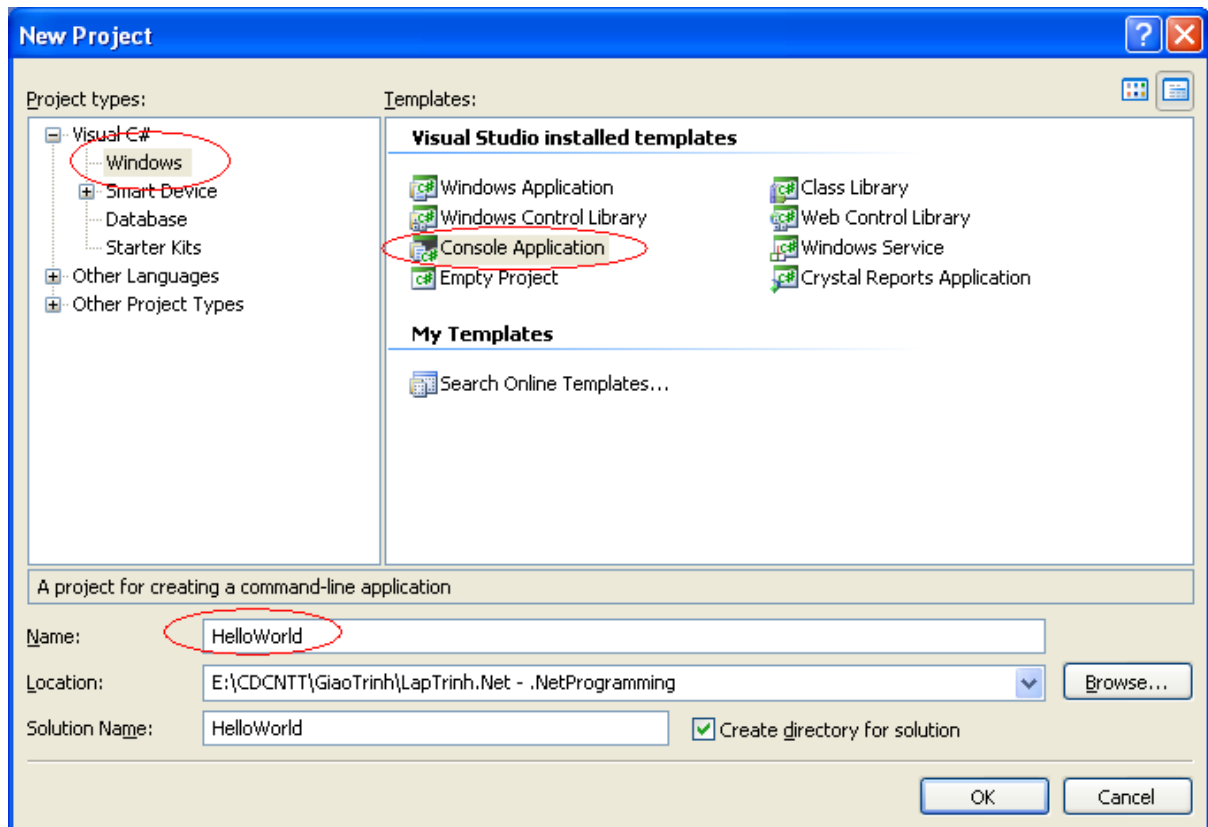
II.1. Tạo Project

Ngay sau khi khởi động Visual Studio, chọn Menu File → New → Project.



Hình 3 – Khởi tạo Project

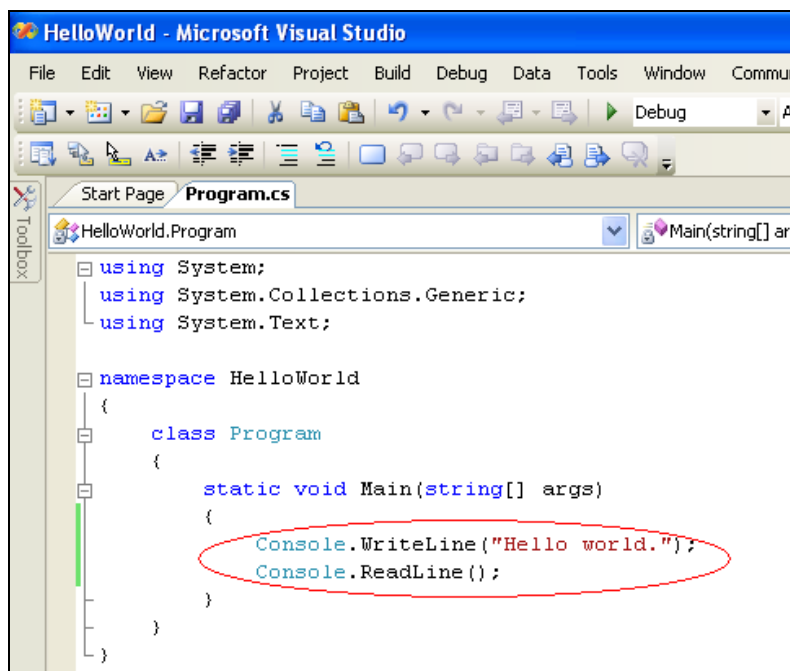
Sau khi chọn vào Project, Visual Studio hiển thị giao diện để người dùng chọn dạng Project ở mục *Project types* và *Templates*, đặt tên cho Project ở mục *Name* và đặt đường dẫn cho Project ở mục *Location*. Ở ví dụ ở hình 4, tài liệu trình bày cách tạo Project Console Application với tên là HelloWorld và Project này được đặt trong thư mục E:\CDCNTT\GiaoTrinh\LapTrinh.Net - .NetProgramming.



Hình 4 – Cấu hình Console Application Project

II.2. Lập trình

Ngay sau khi khởi tạo, Visual Studio sẽ tạo sẵn một Project với cấu trúc chuẩn (Xem chương 1, phần III.2). Trong Project, Visual Studio đã tạo sẵn một class có tên là Program nằm trong file Program.cs, trong class này có sẵn phương thức Main(), người sử dụng chỉ cần lập trình ngay tại phương thức này.



Hình 5 – Lập trình trong Console Application của C#

Trong ví dụ ở hình 5, tài liệu này trình bày cách lập trình để hiển thị dòng chữ “Hello world.” ra màn hình. C# cung cấp class *Console* để thực hiện việc xuất hoặc nhập dữ liệu. Dòng lệnh `Console.WriteLine();` dùng để hiển thị một chuỗi ra màn hình. Dòng lệnh `Console.ReadLine();` dùng để nhập một chuỗi từ bàn phím.

Trong .NET Framework, Microsoft đưa ra thêm khái niệm namespace để quản lý các class trong cùng một thư viện. Với .NET, namespace là một thuật ngữ dùng để tham chiếu và được hiểu là một tập hợp các class. Như thế, một thư viện (*.dll) sẽ là tập hợp chứa các namespace và trong mỗi namespace chứa các class. Visual Studio tự tạo ra namespace mặc định trùng với tên Project mà bạn đã đặt. Trong trường hợp này, namespace mặc định là HelloWorld.

II.3. Biên dịch

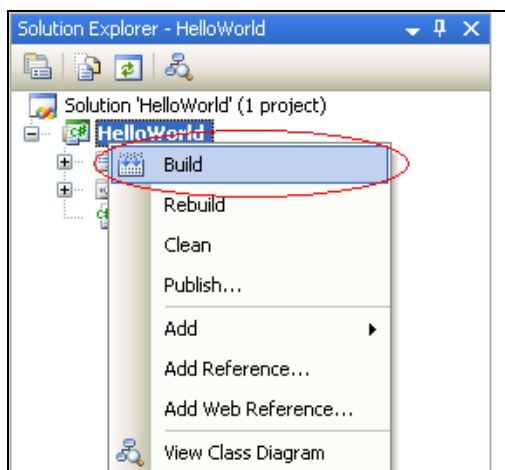
Thông thường, một ứng dụng hoặc một dự án được tổ chức thành một Solution. Tùy vào mức độ lớn nhỏ, ta có 2 cách biên dịch đối với ứng dụng mà ta xây dựng: biên dịch từng phần và biên dịch toàn bộ.

II.3.1. Biên dịch từng phần

Biên dịch từng phần là hình thức biên dịch từng Project trong một Solution. Hình thức biên dịch này áp dụng đối với dự án đã được chia thành những thành phần riêng biệt. Với

hình thức biên dịch, tốc độ biên dịch sẽ nhanh và các lỗi dễ dàng được phân vùng để sửa chữa.

Để thực hiện việc biên dịch từng phần, ta có thể click chuột phải (right-click) vào Project cần biên dịch và chọn *Build*.

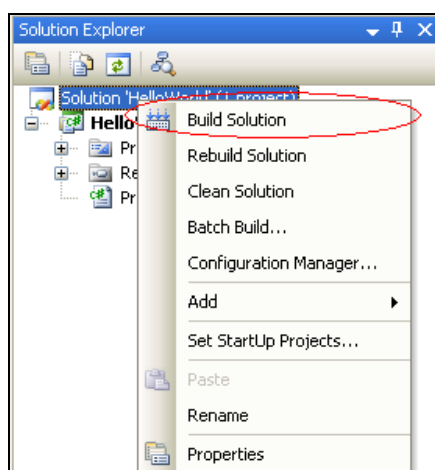


Hình 6 – Biên dịch từng phần

II.3.2. Biên dịch toàn phần

Biên dịch toàn bộ là hình thức biên dịch tất cả các Project trong một Solution. Hình thức biên dịch này được áp dụng đối với các ứng dụng vừa phải hoặc các ứng dụng mà tất cả các Project đều có liên quan mật thiết với nhau. Với hình thức biên dịch này, tốc độ biên dịch sẽ chậm, tuy nhiên tính đồng bộ được bảo đảm.

Để thực hiện biên dịch toàn phần, ta có thể click phải chuột (right-click) vào Solution rồi chọn *Build Solution*.



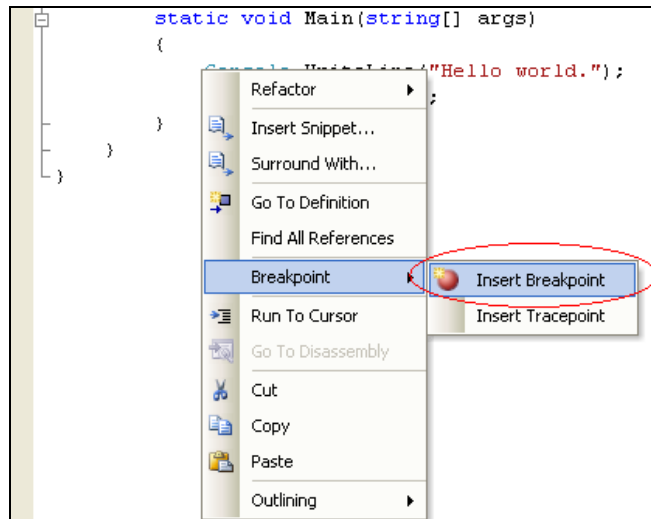
Hình 7 – Biên dịch toàn phần

II.4. Chạy chương trình

Visual Studio cung cấp 2 chế độ chạy chương trình: chế độ debug và chế độ non-debug.

II.4.1. Chế độ debug

Chế độ debug là chế độ chạy từng dòng lệnh để người lập trình bắt lỗi. Trong chế độ này người lập trình quy định một số điểm dừng gọi là *breakpoint*, chương trình sẽ tự động dừng tại *breakpoint* để người dùng dễ dàng theo dõi kết quả của các lệnh chạy tiếp theo. Để tạo ra *breakpoint*, người lập trình chỉ cần click phải chuột (right-click) vào dòng lệnh cần dừng rồi chọn *breakpoint* rồi chọn *Insert Breakpoint*.

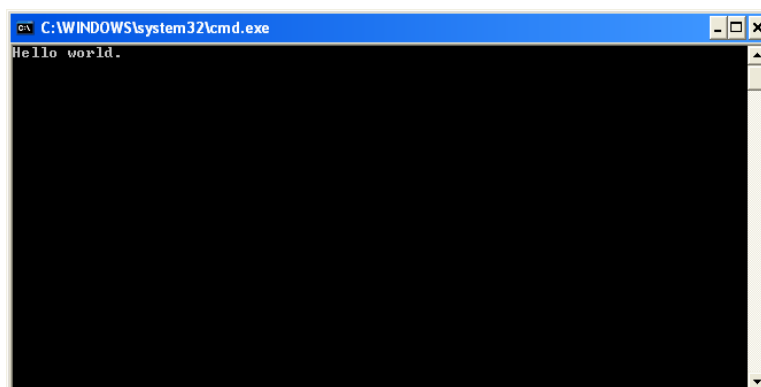


Để thực hiện debug, người lập trình có thể bấm F5 hoặc chọn vào *Menu Debug* → *Start Debug*.

II.4.2. Chế độ non-debug

Chế độ non-debug là chế độ chạy hết cả chương trình mà không dừng lại để bắt lỗi cho dù người lập trình đã thiết lập *breakpoint*. Để chạy chế độ này, người lập trình có thể bấm Ctrl+F5 hoặc chọn vào *Menu Debug* → *Start Without Debugging*.

Với chương trình HelloWorld ở trên kết quả chạy chương trình như sau.



Hình 8 – Kết quả chạy chương trình

III. Biến và phạm vi hoạt động của biến trong C#

III.1. Biến

Biến là đơn vị được các ngôn ngữ lập trình tổ chức để lưu trữ và xử lý dữ liệu. Biến được khai báo theo cú pháp sau.

```
[modifier] datatype identifier;
```

[modifier] là một trong những từ khóa public, private, protected, ...; datatype là kiểu dữ liệu; identifier là biến được người dùng định nghĩa;

Ví dụ dưới đây một biến mang tên i kiểu số nguyên int và có thể được truy cập bởi bất cứ hàm nào.

```
public int i;
```

Ta cũng có thể khai báo biến và khởi tạo cho biến một giá trị như sau:

```
int i = 10;
```

Nếu ta khai báo nhiều biến có cùng kiểu dữ liệu sẽ có dạng như sau:

```
int x = 10; y = 20;
```

III.2. Phạm vi hoạt động của biến

Trong C#, phạm vi hoạt động của biến là vùng đoạn mã mà từ đây biến có thể được truy xuất. Thông thường một đoạn mã được định nghĩa bằng một cặp dấu {}. Trong một phạm vi hoạt động (scope), không thể có hai biến cùng mang một tên trùng nhau.

Ví dụ sau thực hiện việc in ra các số từ 0 đến 9 ra màn hình rồi tiếp tục in các số từ 9 đến 0 ra màn hình;

```
public static int Main()
{
    for (int i = 0; i < 10; i++)
    {
        Console.WriteLine(i);
    } // biến i ra khỏi phạm vi
    // Chúng ta có thể khai báo thêm biến i ở đây
    for (int i = 9; i >= 0; i--)
    {
        Console.WriteLine(i);
    } // biến i ra khỏi phạm vi ở đây
    return 0;
}
```

Hình 9 – Minh họa phạm vi của biến

Với ví dụ trên, trong 2 vòng lặp for khác nhau, ta có thể khai báo cùng một biến *i* cho dù 2 vòng lặp này cùng nằm trong một khối lệnh. Điều này hợp lý bởi vì *i* được khai báo trong hai vòng lặp khác nhau và là biến cục bộ của 2 vòng lặp đó. Khi vòng lặp hoàn được thực hiện xong thì biến tự động được giải phóng và vì thế các biến ở các vòng lặp khác nhau thì có thể được đặt tên giống nhau.

IV. Hằng

Một hằng (constant) là một biến nhưng trị không thể thay đổi được suốt thời gian thi hành chương trình. Đôi lúc ta cũng cần có những giá trị bao giờ cũng bất biến.

Hằng được khai báo như sau:

```
const datatype identifier = value;
```

Ví dụ: `const int numberOfModules = 12;`

Hằng có những đặc điểm sau:

- Hằng bắt buộc phải được gán giá trị lúc khai báo. Một khi đã được khởi gán thì không thể viết đè lên
- Trị của hằng phải có thể được tính toán vào lúc biên dịch, do đó không thể gán một hằng từ một trị của một biến.
- Hằng bao giờ cũng static, tuy nhiên ta không thể đưa từ khoá static vào khi khai báo hằng

Có ba thuận lợi khi sử dụng hằng trong chương trình của bạn:

- Hằng làm cho chương trình đọc dễ dàng hơn, bằng cách thay thế những con số vô cảm bởi những tên mang đầy ý nghĩa hơn
- Hằng làm cho dễ sửa chương trình hơn, việc thay đổi giá trị chỉ cần thực hiện một lần ngay tại vị trí khai báo hàm
- Hằng làm cho việc tránh lỗi dễ dàng hơn, nếu bạn gán một trị khác cho một hằng vốn đã được khai báo đâu đó trong chương trình trình biên dịch sẽ tự động thông báo lỗi vì hằng này đã được khai báo

V. Kiểu dữ liệu

C# là một ngôn ngữ được kiểm soát chặt chẽ về mặt kiểu dữ liệu, ngoài ra C# còn chia các kiểu dữ liệu thành hai loại khác nhau: kiểu trị (value type) và kiểu qui chiếu (reference type). Nghĩa là trên một chương trình C# dữ liệu được lưu trữ một hoặc hai nơi tùy theo đặc thù của kiểu dữ liệu.

C# cũng hỗ trợ kiểu con trỏ (pointer type) giống như C++ nhưng ít khi dùng đến và chỉ dùng khi làm việc với đoạn mã unmanaged. Đoạn mã unmanaged là đoạn mã được tạo ra ngoài .NET Framework, chẳng hạn những đối tượng COM.

V.1. Kiểu giá trị (Value Types)

V.1.1. Kiểu dữ liệu số nguyên

Tên kiểu	Diễn giải	Miền giá trị
sbyte	Số nguyên có dấu 8bit	-128:127
short	Số nguyên có dấu 16bit	$-2^{15}:2^{15}-1$
int	Số nguyên có dấu 32bit	$-2^{31}:2^{31}-1$
long	Số nguyên có dấu 64bit	$-2^{63}:2^{63}-1$
byte	Số nguyên không dấu 8bit	0:255
ushort	Số nguyên không dấu 16bit	$0:2^{16}-1$
uint	Số nguyên không dấu 32bit	$0:2^{32}-1$
ulong	Số nguyên không dấu 64bit	$0:2^{64}-1$

Hình 10 – Danh sách các kiểu số nguyên của C#

Với các kiểu dữ liệu trên, ta có thể khai báo biến như sau:

```
long x = 0x12ab; // ghi theo hexa
```

```
int i = 1234;
```

Khi sử dụng các kiểu số nguyên của C# cần chú ý rằng kiểu int của C# được cấp phát 32bit không giống như kiểu int của Ansi C vốn chỉ được cấp phát 16bit. Nói một cách khác kiểu int trong C# tương ứng với kiểu long trong Ansi C.

V.1.2. Kiểu dữ liệu dấu chấm động

Tên kiểu	Diễn giải	Số chữ số có nghĩa
float	Kiểu số thực 32bit	7
double	Kiểu số thực 64bit	16

Hình 11 – Danh sách các kiểu dữ liệu dấu chấm động

V.1.3. Các kiểu dữ liệu khác

Tên kiểu	Diễn giải	Ghi chú
decimal	Số thập phân 128bit	28 chữ số có nghĩa
bool	Kiểu logic	[true, false]
char	Kiểu ký tự 16bit	kiểu ký tự unicode

Hình 12 – Các kiểu dữ liệu khác

V.2. Kiểu tham chiếu (Reference Type)

C# hỗ trợ 2 kiểu tham chiếu cơ bản: object, string. Kiểu object là kiểu dữ liệu gốc, tất cả các kiểu dữ liệu khác đều được dẫn xuất từ kiểu dữ liệu này. Kiểu string là kiểu dữ liệu trình bày chuỗi ký tự. Chuỗi string trong C# hỗ trợ hoàn toàn unicode.

Cũng giống như Ansi C và Java, C# định nghĩa giá trị của một chuỗi trong cặp dấu ngoặc kép "", còn được gọi là Double Quote. Để tiện cho việc xử lý tất cả các ký tự, C# định nghĩa một số ký tự đặc biệt gọi là Escape Sequence.

Danh sách các Escape Sequence thông dụng.

Escape Sequence	Diễn giải
\'	Single quote
\"	Double quote
\0	Null
\n	Return
\\	Backslash

Hình 13 – Danh sách Escape Sequence

C# hỗ trợ toán tử + để ghép nối các chuỗi và toán tử = để gán giá trị cho chuỗi. Ví dụ sau trình bày các toán tử dùng cho kiểu string.

```
public static int Main()
{
    string s1 = "Windows folder is: ";
    string s2 = "C:\\windows";
    Console.WriteLine("s1 is: " + s1);
    Console.WriteLine("s2 is: " + s2);
    s1 = s1 + s2;
    Console.WriteLine("s1 is now: " + s1);
    Console.WriteLine("s2 is now: " + s2);
    return 0;
}
```

Hình 14 – Minh họa các toán tử trên kiểu string

Chương trình chạy ra kết quả như sau:

```
s1 is: Windows folder is
s2 is: C:\windows
s1 is now: Windows folder is C:\windows
s2 is now: C:\windows
```

Hình 15 – Kết quả chạy chương trình ở ví dụ trong hình 14

Với ví dụ trên, cho thấy sau khi thực hiện toán tử gán thì giá trị của biểu thức bên phải sẽ được gán vào biến bên trái cho dù biểu thức ở phía bên phải có sự xuất hiện của biến được trả về. Cũng trong ví dụ này, để mô tả ký tự \ (ký tự đường dẫn hay ký tự Backslash), chúng ta phải sử dụng quy định của Escape Sequence, trong trường hợp này, ví dụ sử dụng ký tự \\.

VI. Cấu trúc điều kiện

VI.1. Câu lệnh điều kiện if..else

VI.1.1. Cú pháp

```
if (condition) statement(s);
[else statement(s);]
```


VI.1.2. Cách sử dụng

Lệnh điều kiện *if* thực hiện một hoặc nhiều lệnh trong khối lệnh nếu kết quả trả về của biểu thức *condition* là *true*, ngược lại, các lệnh hoặc khối lệnh ngay sau từ khóa *else* sẽ được thực hiện. Xét ví dụ sau:

```
if (i != 0)
{
    Console.WriteLine("i is not Zero");
}
else
{
    Console.WriteLine("i is Zero");
}
```

Hình 16 – Minh họa cấu trúc *if..else*

Ví dụ trên thực hiện việc kiểm tra biến *i* có bằng 0 hay không. Nếu *i* bằng 0, chương trình sẽ xuất ra “i is Zero”, ngược lại, chương trình sẽ xuất ra “i is not Zero”.

VI.2. Câu lệnh *switch..case*

Trong trường hợp phải dùng nhiều lệnh *if..else* lồng nhau, chương trình sẽ trở nên phức tạp và khó gỡ rối. Để khắc phục vấn đề này, C# cung cấp cấu trúc lệnh *switch..case* để phục vụ cho việc lập trình khi xử lý một loạt các trường hợp khác nhau của cùng một điều kiện.

VI.2.1. Cú pháp

```
switch (expression)
{
    case (condition): statement(s);
    [break;]
    ...
    case (condition): statement(s);
    [break;]
    [default: statement(s);]
}
```

VI.2.2. Cách sử dụng

Với từng trường hợp của biểu thức điều kiện *expression*, các lệnh tương ứng sẽ được thực hiện. Trong cấu trúc lệnh này, lệnh *break* được sử dụng để nhảy ra khỏi khối lệnh. Chỉ dẫn *default* được sử dụng để giải quyết trường hợp của điều kiện mà trường hợp này không nằm trong tất cả các trường hợp đã liệt kê trong các chỉ dẫn *case*. Ví dụ sau trình bày cách sử dụng lệnh *switch..case* vào việc tính số ngày trong một tháng được nhập từ bàn phím.

```
static void Main(string[] args)
{
    Console.Write("Nhập 1 tháng trong năm: ");
    string s = Console.ReadLine();
    int month = Convert.ToInt32(s);
    switch (month)
    {
        case 1:
        case 3:
        case 5:
        case 7:
        case 8:
        case 10:
        case 12:
            Console.WriteLine("Thang co 31 ngay");
            break;
        case 4:
        case 6:
        case 9:
        case 11:
            Console.WriteLine("Thang co 30 ngay");
            break;
        case 2:
            Console.WriteLine("Thang co 28 ngay");
            break;
        default:
            Console.WriteLine("Khong phai la thang trong nam");
            break;
    }
    Console.ReadLine();
}
```

Hình 17 – Minh họa cấu trúc switch..case

VII. Cấu trúc lặp

VII.1. Cấu trúc lặp for

VII.1.1. Cú pháp

for (*initializer; condition; iterator*)
 statement(s)

VII.1.2. Cách sử dụng

Cấu trúc của lệnh for bao gồm biểu thức khởi tạo *initializer*, điều kiện kết thúc *condition*, biểu thức biến đổi *iterator*. Lệnh for thực hiện việc lặp một hoặc nhiều lệnh trong khi điều kiện kết thúc *condition* vẫn còn đúng. Ví dụ sau minh họa cho chương trình xuất ra màn hình tất cả các số dương chia hết cho 3 mà nhỏ hơn 100.

```
static void Main(string[] args)
{
    for (int i = 1; i < 100; i++)
    {
        if ((i % 3) == 0) Console.Write(i + " ");
    }
    Console.ReadLine();
}
```

Hình 18 – Minh họa cấu trúc lặp for

VII.2. Cấu trúc lặp while

VII.2.1. Cú pháp

```
while(condition)
    statement(s);
```

VII.2.2. Cách sử dụng

Cấu trúc while thực hiện việc lặp một hay nhiều lệnh khi điều kiện *condition* vẫn còn đúng. Khi sử dụng cấu trúc này, người lập trình cần phải chủ động thực hiện các câu lệnh tạo sự biến đổi để tránh những vòng lặp vô tận. Ví dụ sau minh họa cho chương trình tính ước chung lớn nhất của 2 số nguyên a, b.

```
static void Main(string[] args)
{
    int a = 24;
    int b = 60;
    while ((a % b) != 0)
    {
        int c = a % b;
        a = b;
        b = c;
    }
    Console.WriteLine("Ước chung lớn nhất: " + b);
    Console.ReadLine();
}
```

Hình 19 – Minh họa cấu trúc lặp while

VII.3. Cấu trúc lặp do..while

VII.3.1. Cú pháp

```
do
{
    statement(s);
} while (condition);
```

VII.3.2. Cách sử dụng

Cấu trúc lặp do..while thực hiện việc lặp một hoặc nhiều lệnh cho tới khi điều kiện *condition* có giá trị false. Cấu trúc lặp này có đặc điểm là các lệnh được thực hiện ít nhất một lần cho dù ngay từ đầu điều kiện đã là false. Ví dụ sau đây minh họa cho chương trình tính tổng các số dương bé hơn 100.

```
static void Main(string[] args)
{
    int S = 0;
    int i = 0;
    do
    {
        S += i;
        i++;
    } while (i < 100);
    Console.WriteLine("Tong can tim: " + S);
    Console.ReadLine();
}
```

Hình 20 – Minh họa cấu trúc lặp do..while

VII.4. Các lệnh hỗ trợ cho cấu trúc lặp

VII.4.1. Lệnh break

Lệnh *break* thực hiện việc dừng vòng lặp. Khi chương trình đang chạy mà gặp lệnh *break* thì chương trình sẽ lập tức chấm dứt vòng lặp cho dù điều kiện của vòng lặp vẫn cho phép chạy tiếp. Trong trường hợp có nhiều vòng lặp lồng nhau, chương trình sẽ chấm dứt vòng lặp gần với *break* nhất. Ví dụ sau sử dụng vòng lặp for và câu lệnh break để tính tổng tất cả các số nguyên tố nhỏ hơn 100.

```
static void Main(string[] args)
{
    int S = 0;
    bool isPrime;
    for (int x = 2; x < 100; x++)
    {
        isPrime = true;
        for (int i = 2; i < Math.Sqrt(x); i++)
        {
            if (x % i == 0)
            {
                isPrime = false;
                break;
            }
        }
        if (isPrime)
        {
            Console.Write(x + " ");
            S += x;
        }
    }
    Console.WriteLine();
    Console.WriteLine("Tong cac so nguyen to be hon 100: " + S);
    Console.ReadLine();
}
```

Hình 21 – Minh họa lệnh break

VII.4.2. Lệnh continue

Trong khi thực hiện vòng lặp, người lập trình đôi khi cần thực hiện việc bỏ qua một số dòng lệnh để tiếp tục thực hiện việc lặp cho lần tiếp theo. C# hỗ trợ lệnh *continue* để thực hiện chức năng này. Lệnh *continue* thực hiện việc chuyển sang lần lặp tiếp theo và bỏ qua các

lệnh nằm trong vòng lặp nhưng nằm phía sau nó. Ví dụ sau minh họa cho việc sử dụng vòng lặp *for* và lệnh *continue*.

```
static void Main(string[] args)
{
    for (int i = 1; i < 10; i++)
    {
        Console.WriteLine(i + " ");
        if (i % 3 == 0) continue;
        Console.WriteLine(" i không chia hết cho 3 ");
    }
    Console.ReadLine();
}
```

Hình 22 – Minh họa lệnh Continue

VIII. Mảng (Array)

Mảng hay còn gọi là Array là một cấu trúc dữ liệu cấu tạo bởi một số biến được gọi là những phần tử mảng. Tất cả các phần tử này đều thuộc một kiểu dữ liệu. Người lập trình có thể truy xuất phần tử thông qua chỉ số (*index*). Chỉ số bắt đầu bằng zero.

Người ta thường chia mảng thành 2 loại: Mảng một chiều và mảng nhiều chiều. Đối với mảng nhiều chiều giáo trình này chỉ trình bày mảng hai chiều bởi vì mảng hai chiều là đặc trưng tiêu biểu cho mảng nhiều chiều.

VIII.1. Mảng một chiều

VIII.1.1. Cú pháp khai báo

type [] *array-name*;
hoặc *type*[] *array-name* = new *type*[*length*];

Với cú pháp trên, *type* là kiểu dữ liệu của các phần tử trong mảng, *array-name* là tên của mảng, *length* là độ dài của mảng – độ dài này chính là số phần tử của mảng.

VIII.1.2. Cách sử dụng

Để làm việc với mảng, người lập trình thường can thiệp trực tiếp vào từng phần tử của mảng thông qua chỉ số *index* với cú pháp *array-name*[*index*]. Ví dụ:

```
x = A[0];
st = B[i];
B[1] = x;
```

Để duyệt mảng, người lập trình thường sử dụng cấu trúc lặp. Bên cạnh các cấu trúc lặp đã trình bày, C# còn cung cấp một cấu trúc lặp khác là vòng lặp *foreach*.

VIII.1.3. Cấu trúc lặp *foreach*

Cấu trúc lặp *foreach* được thực hiện theo cấu trúc sau:

```
foreach (data-type identifier in array-name) statement(s)
```

Với mỗi giá trị của mảng *array-name*, chương trình sẽ thực hiện một hoặc nhiều lệnh *statement(s)*. Trong đó, giá trị của mỗi phần tử được tham chiếu qua biến *identifier* mà biến này thuộc kiểu *data-type*. Ví dụ sau minh họa việc sử dụng mảng một chiều và minh họa cho phương pháp duyệt mảng một chiều bằng cấu trúc lặp *foreach*.

```
static void Main(string[] args)
{
    string[] artists = { "Leonardo", "Monet", "Van Gogh", "Klee" };
    foreach (string name in artists)
    {
        Console.WriteLine(name);
    }
    Console.ReadLine();
}
```

Hình 23 – Minh họa cấu trúc *foreach* khi duyệt mảng Array

VIII.2. Mảng hai chiều (Ma trận)

VIII.2.1. Cú pháp khai báo

type[,] *array-name*;

hoặc *type*[,] *array-name* = *new type*[*height*, *width*];

Với cú pháp trên, *type* là kiểu dữ liệu của các phần tử trong mảng, *array-name* là tên của mảng, *height* và *width* lần lượt là số dòng và số cột của ma trận. Ví dụ như sau:

```
int[,] myRectArray = new int[2,3]; //khai báo mảng số nguyên có 2 dòng và 3 cột
```

```
int[,] myRectArray = new int[,] { {1,2},{3,4},{5,6},{7,8} }; //khai báo mảng 4 hàng 2 cột
```

```
string[,] beatleName = { {"Lennon","John"}, {"McCartney","Paul"}, {"Harrison","George"}, {"Starkey","Richard"} }; //khai báo mảng string có 2 dòng 4 cột
```

VIII.2.2. Cách sử dụng

Để làm việc với mảng hai chiều, người lập trình có thể can thiệp trực tiếp vào từng phần tử của mảng thông qua chỉ số hàng và chỉ số cột. Ví dụ như sau:

```
A[0, 1] = 2; //Gán giá trị 2 vào phần tử ở hàng 0 cột 1
```

```
Console.Write(A[2, 3]); // Ghi giá trị của phần tử ở hàng 2 cột 3 ra màn hình
```

Vì cấu trúc lưu trữ của ma trận bao gồm các hàng và các cột nên người lập trình thường sử dụng hai vòng lặp lồng nhau để thực hiện việc duyệt cả ma trận. Ví dụ sau minh họa việc duyệt ma trận bằng cách sử dụng hai vòng lặp lồng nhau, trong quá trình lặp, chương trình sẽ hiển thị các chuỗi trong ma trận.

```
static void Main(string[] args)
{
    string[,] beatleName = { { "Lennon", "John" }, { "McCartney", "Paul" },
                             { "Harrison", "George" }, { "Starkey", "Richard" } };
    for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < 2; j++)
        {
            Console.Write(beatleName[i, j] + " ");
        }
        Console.WriteLine();
    }
    Console.ReadLine();
}
```

Hình 24 – Minh họa việc sử dụng hai vòng lặp lồng nhau khi dùng ma trận

IX. Xử lý nhập xuất file

IX.1. Khái niệm file

File còn được gọi là tập tin hoặc được gọi là tệp là khái niệm dùng để chỉ tập hợp những thông tin liên quan với nhau và những thông tin này được lưu trữ trên đĩa.

Khác với những thông tin được lưu trữ trên RAM, các thông tin được lưu trong file phải được truy cập một cách tuần tự, nghĩa là muốn đọc hoặc ghi dữ liệu ở vị trí thứ i thì bộ xử lý phải thực hiện việc đọc hoặc ghi từ vị trí 0 đến vị trí $i-1$.

IX.2. Phân loại

Tùy vào nội dung dữ liệu chứa trong file, file được chia thành 2 loại: file nhị phân (binary file) và file văn bản (text file). Trong phần này giáo trình chỉ trình bày phương pháp đọc ghi đối với file văn bản, người học tự nghiên cứu phương pháp đọc ghi file nhị phân từ các tài liệu tham khảo trong phần tài liệu tham khảo ở cuối giáo trình.

IX.2.1. File văn bản (text file)

File văn bản là file mà dữ liệu ghi trong đó là những ký tự thuộc bảng mã ASCII chuẩn (bảng mã không bao gồm các ký tự điều khiển) trong đó các ký tự được tổ chức thành nhiều hàng. Các hàng được phân cách bằng ký tự xuống hàng.

Đối với các file văn bản được tổ chức trên hệ điều hành Dos hoặc Windows của Microsoft thì các hàng được phân cách bởi một cặp ký tự bao gồm: ký tự xuống dòng và ký tự về đầu dòng. Đối với các file văn bản được tổ chức trên hệ điều hành Unix hoặc Linux thì các hàng được phân cách bởi một dấu xuống hàng. Điều này giải thích tại sao khi người sử dụng mở một số file vốn được soạn thảo trong hệ điều hành Linux bằng chương trình soạn thảo của Windows (cụ thể là *notepad*) thì tất cả các dòng của file được nối thành một dòng duy nhất.

Các file văn bản thông thường là những file văn bản đơn giản hoặc những file mã nguồn của những ngôn ngữ lập trình hoặc những file cơ sở dữ liệu XML.

IX.2.2. File nhị phân (binary file)

File nhị phân là file mà thông tin lưu trữ một cách tổng quát. Dữ liệu lưu trữ trên file nhị phân được chia thành từng byte, mỗi byte được đặc trưng bởi một ký tự của bảng mã ASCII mở rộng.

Các file nhị phân thường có một cấu trúc hoặc định dạng riêng, những file được biên soạn theo định dạng nào thì chỉ có những phần mềm hiểu được định dạng đó mới có thể đọc được. Ví dụ những file hình ảnh thì chỉ có thể đọc được bằng những chương trình hiển thị hình ảnh.

Các chương trình soạn thảo cho file văn bản thì không hiển thị chính xác file nhị phân. Cụ thể là khi người sử dụng dùng chương trình soạn thảo văn bản để mở một file thực thi (.exe) thì chương trình soạn thảo văn bản ấy chỉ hiển thị các ký tự đặc biệt.

Các file nhị phân thông thường là những file thực thi (.exe), file hình ảnh (.jpg, .gif, .png, .bmp, ...), file âm thanh (.mp3, .wma, .mid, .wav, ...), file chứa các đoạn phim (.mpg, .wmv, .rm, ...) và nhiều dạng định dạng mặc định khác.

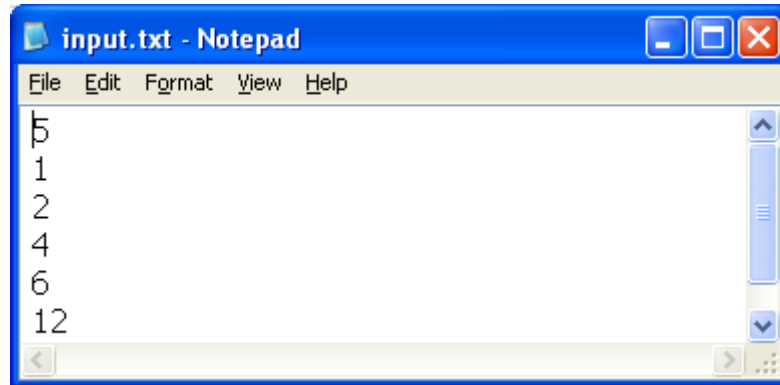
IX.3. Đọc và ghi file văn bản

Visual Studio cung cấp nhiều đối tượng để thực hiện việc đọc và ghi file văn bản. Giáo trình này trình bày việc đọc bằng cặp đối tượng *StreamReader* và *StreamWriter*. Đây là cặp đối tượng đọc/ghi tổng quát, người lập trình có thể sử dụng cặp đối tượng này để thực hiện việc đọc hoặc ghi trên các luồng nhập xuất khác. Để sử dụng phương thức này, người lập trình phải khai báo sử dụng namespace *System.IO* của Visual Studio bằng lệnh `using System.IO` – chi tiết về namespace sẽ được trình bày trong chương tiếp theo. Người học tự nghiên cứu các phương pháp đọc ghi file khác từ các giáo trình được nêu trong phần tài liệu tham khảo.

IX.3.1. Đọc file văn bản bằng *StreamReader*

StreamReader là một lớp và lớp này thực tạo ra các đối tượng phục vụ việc đọc dữ liệu từ một luồng hoặc một file. Khi khởi tạo đối tượng, người lập trình phải cung cấp tên file cho phương thức khởi tạo. Việc khởi tạo đối tượng có thể sinh ra các lỗi như file không tồn tại, đĩa bị hỏng, ..., do đó, hàm khởi tạo phải được đặt trong khối *try/catch* để xử lý biệt lệ – Xử lý biệt lệ sẽ được trình bày ở chương tiếp theo.

Đối tượng *StreamReader* cung cấp nhiều phương pháp để đọc dữ liệu, trong đó phương thức hữu hiệu nhất là phương thức *ReadLine()*. Phương thức *ReadLine()* thực hiện việc đọc từng dòng trong file văn bản và trả về một chuỗi *string*. Ví dụ sau trình bày việc đọc dữ liệu từ file văn bản có tên là `input.txt`, file này chứa các số nguyên, mỗi số nguyên nằm trên một dòng.

**Hình 25 – File input.txt**

Chương trình sau sử dụng đối tượng StreamReader để đọc file input.txt và tính tổng các số nguyên của trong file đó.

```
static void Main(string[] args)
{
    string inputFileNames = "input.txt";
    StreamReader streamReader = null;
    try
    {
        streamReader = new StreamReader(inputFileNames);
        string temp;
        int N;
        temp = streamReader.ReadLine();
        N = Convert.ToInt32(temp);
        int Sum = 0;
        for (int i = 0; i < N; i++)
        {
            temp = streamReader.ReadLine();
            int x = Convert.ToInt32(temp);
            Sum += x;
        }
        Console.WriteLine("Summary: " + Sum);
    }
    catch (Exception e)
    {
        Console.WriteLine("Can not open file " + inputFileNames);
        Console.WriteLine("Error: " + e.ToString());
    }
    finally
    {
        if (streamReader != null) streamReader.Close();
    }
    Console.ReadLine();
}
```

Hình 26 – Chương trình sử dụng StreamReader để đọc file văn bản

IX.3.2. Ghi file văn bản bằng *StreamWriter*

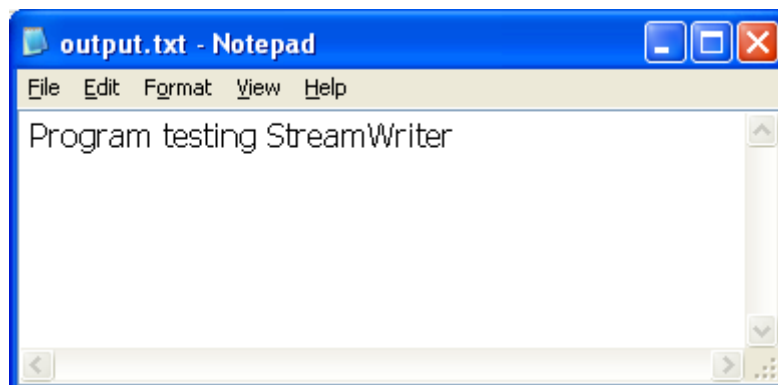
Cũng giống *StreamReader*, *StreamWriter* là một lớp và lớp này tạo ra các đối tượng phục vụ việc ghi dữ liệu vào một luồng hoặc một file. Khi khởi tạo đối tượng, người lập trình phải cung cấp tên file cho phương thức khởi tạo. Việc khởi tạo đối tượng có thể sinh ra các lỗi như đĩa bị hỏng hoặc lỗi nhập xuất, ..., do đó, hàm khởi tạo phải được đặt trong khối *try/catch* để xử lý biệt lệ – Xử lý biệt lệ sẽ được trình bày ở chương tiếp theo.

Đối tượng *StreamWriter* cung cấp nhiều phương pháp để ghi dữ liệu, trong đó phương thức hữu hiệu nhất là phương thức *Write()*. Phương thức *Write()* thực hiện việc ghi một chuỗi string vào file văn bản. Ví dụ sau trình bày việc ghi dữ liệu từ file văn bản có tên là *output.txt*.

```
static void Main(string[] args)
{
    string outputFileName = "output.txt";
    StreamWriter streamWriter = null;
    try
    {
        streamWriter = new StreamWriter(outputFileName);
        streamWriter.Write("Program testing StreamWriter");
        Console.WriteLine("Writing file complete.");
    }
    catch (Exception e)
    {
        Console.WriteLine("Can not open file " + outputFileName);
        Console.WriteLine("Error: " + e.ToString());
    }
    finally
    {
        if (streamWriter != null) streamWriter.Close();
    }
    Console.ReadLine();
}
```

Hình 27 – Chương trình sử dụng *StreamWriter* để ghi file văn bản

Sau khi chạy chương trình, file văn bản thu được như sau.



Hình 28 – File kết quả thu được sau khi chạy chương trình

LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG VỚI C#

I. Mục tiêu

Từ các học phần *Lập trình hướng đối tượng với C++* và *Lập trình Java*, người học đã được tiếp cận lý thuyết và thực hành với *Lập trình hướng đối tượng (Object Oriented Programming)*. Trong chương này giáo trình cũng cung cấp những kiến thức cơ bản về lập trình hướng đối tượng tuy nhiên các nguyên lý này được vận hành bằng ngôn ngữ C#. Những chủ đề chính được trình bày trong chương này như sau:

- Lớp (Class) và Đối tượng (Object)
- Thuộc tính (Attribute) và Phương thức (Method)
- Nạp chồng toán tử (Overloading)
- Kế thừa (Inheritance)
- Đa hình (Polymorphism)
- Interface

Cuối chương này người học sẽ có đủ kiến thức để giải quyết các vấn đề lập trình bằng quan điểm hướng đối tượng.

II. Lớp (Class)

Lớp là một khái niệm trong phương pháp lập trình hướng đối tượng (*Object Oriented Programming*) được dùng để chỉ sự gói gọn các dữ liệu và các phương thức hoạt động trên dữ liệu đó. Lớp còn được hiểu là khái niệm dùng để chỉ tập hợp tất cả các thực thể có chung một số thuộc tính nào đó trong phạm vi ngữ cảnh nào đó. Ví dụ, khi xét một số thực thể gồm báo, hổ, sư tử, ngựa vằn; Nếu các thực thể này được xét trong ngữ cảnh chỉ chú ý đến 3 yếu tố gồm màu lông, cân nặng, tốc độ di chuyển thì các thực thể này có thể được gộp chung vào một lớp gọi là lớp Thú;

Trong C#, tất cả các lớp được dẫn xuất từ lớp cơ sở System.Object. Mỗi lớp thường được định nghĩa trong một file .cs và file này được thêm vào project. Ví dụ sau đây minh họa việc xây dựng một class. Trong class này chỉ có một thuộc tính là name và một phương thức là Speak(). Cũng giống như các ngôn ngữ C++ và Java, C# cung cấp con trỏ *this* dùng để tham chiếu đến một thể hiện (instance) của class đang xét.

```

class Dog
{
    private string _name;
    public Dog(string name)
    {
        this._name = name;
    }
    public void Speak()
    {
        Console.WriteLine("Ahn Ahn! My name is {0}", this._name);
    }
    public string Name
    {
        get
        {
            return this._name;
        }
        set
        {
            this._name = value;
        }
    }
}

```

Hình 29 – Minh họa việc xây dựng Class trong C#

Trong lớp có một phương thức mặc định dùng để khởi tạo đối tượng được gọi là phương thức khởi tạo (constructor). Phương thức khởi tạo được đặt tên trùng với tên lớp và có chỉ dẫn truy cập bắt buộc là public để phương thức này có thể được triệu gọi từ bên ngoài phương thức. Các chỉ dẫn biên dịch (accessibility) bao gồm: public, internal, protected, private được mô tả chi tiết theo hình sau.

Chỉ dẫn truy cập	Mô tả
public	Biến và phương thức có thể được truy cập từ bất kỳ nơi đâu (bên ngoài cũng như bên trong lớp)
internal	Biến và phương thức chỉ có thể được truy cập trong cùng một gói
protected	Biến và phương thức chỉ được truy cập ngay trong lớp đó hoặc được truy cập từ trong lớp kế thừa của nó
private	Biến và phương thức chỉ được truy cập ngay trong phạm vi của lớp đó

Hình 30 – Mô tả các chỉ dẫn truy cập trong C#

III. Đối tượng (Object)

Đối tượng là khái niệm dùng để chỉ một trường hợp cụ thể (instance) của một lớp. Trong lập trình hướng, đối tượng là thực thể cơ bản của việc lập trình, các ứng xử và hành vi của đối tượng được chương trình triệu gọi thông qua các phương thức đã được mô tả ở lớp thuộc đối tượng đó. Cũng giống như các lớp trong C#, tất cả các đối tượng cũng được kế thừa từ đối tượng cơ sở System.Object. Ví dụ sau minh họa việc khai báo và sử dụng đối tượng của lớp Dog vừa được mô tả trên.

```
static void Main(string[] args)
{
    Dog toto = new Dog("Toto");
    Console.Write(toto.Name + ": ");
    toto.Speak();

    Dog milu = new Dog("Milu");
    Console.Write(milu.Name + ": ");
    milu.Speak();

    Console.ReadLine();
}
```

Hình 31 – Minh họa việc khai báo và sử dụng đối tượng

Trong ví dụ trên, toán tử *new* được sử dụng khi khởi tạo đối tượng. Tham số ban đầu của phương thức khởi tạo được đưa vào trực tiếp từ tham số của toán tử *new*. Phương thức có chỉ dẫn truy cập là *public* được triệu gọi bằng cách gọi tên đối tượng và tên phương thức, trong đó dấu chấm (dot) được sử dụng để ngăn cách giữa đối tượng và phương thức cần triệu gọi.

IV. Thuộc tính (Attribute)

Thuộc tính là khái niệm dùng để chỉ các tính chất đặc trưng của một lớp. Trong một lớp, các thuộc tính được mô tả một cách định tính bao gồm tên thuộc tính, kiểu dữ liệu và chỉ dẫn truy cập. Trong một đối tượng, các thuộc tính được thể hiện một cách định lượng với việc các thuộc tính được cung cấp một giá trị xác định (giá trị mặc định là *null* nếu như thuộc tính chưa cung cấp giá trị)

C# cung cấp thủ tục *get* và *set* để thực hiện việc truy cập một thuộc tính, thủ tục *get* để lấy giá trị và thủ tục *set* để thực hiện việc thay đổi giá trị. Ví dụ ở hình 25 mô tả thủ tục *get* và *set* đối với thuộc tính *private string _name*, thủ tục *get* và *set* thực hiện việc định nghĩa một định danh (*alias*) khác có tên là *Name* để người dùng sử dụng khi truy cập vào thuộc tính *_name*. Nếu không có thủ tục này, người sử dụng không thể truy cập thuộc tính *_name* vì thuộc tính này được định nghĩa với chỉ dẫn truy cập là *private*.

V. Phương thức (Method)

Phương thức là khái niệm dùng để chỉ một ứng xử hoặc là một hành động của đối tượng. Trong C#, phương thức được chia thành 2 loại: non-static method và static method.

- Non-static method là phương thức chỉ có thể được gọi từ đối tượng, phương thức này mang tính đặc thù của đối tượng và những đối tượng khác nhau trong cùng một lớp sẽ có những kết quả trả về không giống nhau khi cùng triệu gọi một phương thức thuộc loại này. Trong ví dụ ở hình 27, phương thức *Speak()* do đối tượng *toto* thực hiện sẽ có kết quả khác với phương thức do đối tượng *milu* triệu gọi.
- Static method là phương thức đặc trưng cho tất cả các đối tượng thuộc lớp đó. Phương thức này không thể được gọi từ đối tượng mà phải được gọi trực tiếp từ lớp. Kết quả chạy phương thức này không phụ thuộc vào đối tượng mà phụ thuộc vào tham số đầu vào của phương thức. Trong các ví dụ trên, phương thức *Console.WriteLine()* là phương thức static của lớp *Console*.

VI. Nạp chồng toán tử (Overloading)

Nạp chồng toán tử là khái niệm dùng để chỉ việc định nghĩa lại một số toán tử mà những toán tử này làm việc với dữ liệu là đối tượng thuộc lớp mà người lập trình đang xây dựng. Toán tử có tính đồng nhất đối với tất cả các đối tượng thuộc lớp hoặc đối với tất cả các giá trị thuộc kiểu dữ liệu do đó C# yêu cầu tất cả các toán tử phải là *static method* với chỉ dẫn truy cập là *public*. Trong C#, tất cả các lớp đều dẫn xuất từ lớp cơ sở *System.Object* và lớp này vốn định nghĩa sẵn phương thức *equal* và toán tử gán (=) do đó người lập trình không cần thiết phải định nghĩa lại toán tử gán. Ví dụ sau đây trình bày việc nạp chồng toán tử cộng (+) đối với đối tượng thuộc lớp phân số (Fraction).

```
class Fraction
{
    public int numerator;
    public int denominator;
    public Fraction(int numer, int denom)
    {
        this.numerator = numer;
        this.denominator = denom;
    }
    public static Fraction operator +(Fraction f1, Fraction f2)
    {
        Fraction Result = new Fraction(0, 0);
        if (f1.denominator != f2.denominator)
        {
            Result.denominator = f1.denominator * f2.denominator;
            Result.numerator = f1.numerator * f2.denominator
                + f1.denominator * f2.numerator;
        }
        else
        {
            Result.denominator = f1.denominator;
            Result.numerator = f1.numerator + f2.numerator;
        }
        return Result;
    }
}
```

Hình 32 – Minh họa việc nạp chồng toán tử

Ví dụ trên trình bày việc nạp chồng toán tử cộng (+) 2 phân số (Fraction). Toán tử này được khai báo với tiền tố *static operator* và chỉ dẫn truy cập là *public*.

VII. Kế thừa (Inheritance)

Kế thừa là khái niệm then chốt của các ngôn ngữ lập trình hướng đối tượng dùng để hiểu hiện tượng một lớp thực hiện việc sử dụng lại một số thuộc tính (attribute) hoặc phương thức (method) của một lớp khác. Lớp sử dụng các thuộc tính và phương thức của lớp khác được gọi là lớp dẫn xuất, đôi khi còn được gọi là lớp con. Lớp cho phép lớp khác sử dụng các thuộc tính và phương thức được gọi là lớp cơ sở, đôi khi còn được gọi là lớp cha.

Kế thừa trong hướng đối tượng được chia thành hai loại: đơn kế thừa (single inheritance) và đa kế thừa (multiple inheritance). Đơn kế thừa được hiểu là một lớp dẫn xuất sử dụng các thuộc tính và phương thức của một lớp cơ sở duy nhất. Ngược lại, nếu một lớp sử dụng các thuộc tính và phương thức từ nhiều lớp sở sở thì được gọi là đa kế thừa.

C++ hỗ trợ cả đơn kế thừa và đa kế thừa, tuy nhiên C# chỉ hỗ trợ đơn kế thừa. Việc sử dụng đa kế thừa một cách hợp lý sẽ giúp cho chương trình tối ưu hơn về mặt kích thước nhưng việc xử lý lỗi (nếu có) sẽ vô cùng phức tạp. Đây là lý do cơ bản mà C# chỉ thiết kế cho đơn kế thừa. Ví dụ sau trình bày việc xây dựng một lớp cơ sở và hai lớp kế thừa.

```
class Dog
{
    public string Name;
    public int Weight;
    public Dog()
    {
        this.Name = "";
        this.Weight = 0;
    }
    public Dog(string name, int weight)
    {
        this.Name = name;
        this.Weight = weight;
    }
    public void Speak()
    {
        Console.WriteLine("Ruff!");
    }
    public void DrinkWater()
    {
        Console.WriteLine("Gulp");
    }
}
```

Hình 33 – Lớp cơ sở

Ví dụ trên trình bày việc xây dựng lớp cơ sở *Dog* gồm hai thuộc tính là *Name*, *Weight* và hai phương thức là *Speak()*, *DrinkWater()*.

```
class GermanShepard:Dog
{
    public GermanShepard(string name, int weight)
    {
        this.Name = name;
        this.Weight = weight;
    }
    public void OnGuard()
    {
        Console.WriteLine("In Guard Mode");
    }
}
class JackRussell:Dog
{
    public JackRussell(string name, int weight)
    {
        this.Name = name;
        this.Weight = weight;
    }
    public void Chew()
    {
        Console.WriteLine("I am chewing your favorite shoes");
    }
}
```

Hình 34 – Các lớp dẫn xuất

Ví dụ trên trình bày hai lớp dẫn xuất từ lớp cơ sở là *Dog*. Lớp thứ nhất là *GermanShepard* trong đó có thêm phương thức là *OnGuard()*, lớp còn lại là *JackRussell* có thêm phương thức là *Chew()*; Ví dụ sau sẽ minh họa việc sử dụng hai lớp dẫn xuất dẫn xuất trên.

```
static void Main(string[] args)
{
    GermanShepard Simon = new GermanShepard("Simon", 3);
    JackRussell Daisy = new JackRussell("Daisy", 2);

    Simon.Speak(); Simon.DinkWater();
    Daisy.Speak(); Daisy.DinkWater();

    Simon.OnGuard();
    Daisy.Chew();

    Console.ReadLine();
}
```

Hình 35 – Minh họa việc sử dụng lớp dẫn xuất

Ví dụ trên khai báo hai đối tượng: đối tượng *Simon* thuộc lớp *GermanShepard* và đối tượng *Daisy* thuộc lớp *JackRussell*. Khi gọi phương thức *Speak()* và *DrinkWater()* từ lớp cơ sở thì các ứng xử (hoặc kết quả thu được) của chương trình là giống nhau. Tuy nhiên khi gọi

phương thức cụ thể từ lớp dẫn xuất – phương thức *OnGuard()* đối với *Simon* và phương thức *Chew()* đối với *Daisy* – thì kết quả thu được khác nhau.

Trong một ứng dụng, việc tận dụng tính năng kế thừa của hướng đối tượng làm cho chương trình trở nên ngắn gọn, dễ hiểu. Trong tình huống chương trình có chứa rất nhiều lớp tương tự nhau trong đó có rất nhiều phương thức giống nhau thì việc xây dựng một lớp cơ sở trong đó có nhiều phương thức chung làm cho việc cập nhật chỉnh sửa được thuận lợi, vì người lập trình chỉ cần sửa một lần tại lớp cơ sở thay vì phải chỉnh sửa từng phương thức trong từng lớp.

VIII. Đa hình (Polymorphism)

Đa hình là thuật ngữ được dùng trong hướng đối tượng dùng để chỉ việc ứng xử khác nhau của các đối tượng trong những lớp kế thừa từ một lớp cơ sở khi một phương thức chung được gọi. Tính đa hình được sử dụng trong trường hợp một phương thức chung được sử dụng trong nhiều lớp dẫn xuất nhưng ứng với mỗi lớp dẫn xuất cụ thể thì phương thức này có những ứng xử khác nhau.

Để thực hiện được tính đa hình, phương thức ở lớp cơ sở phải được mô tả ở dạng ảo (virtual) và phương thức đó ở lớp dẫn xuất phải được ghi đè (override). Override là thuật ngữ được dùng để chỉ việc lớp dẫn xuất đặc tả lại phương thức ở lớp cơ sở. Ví dụ sau trình bày việc thực hiện tính năng đa hình khi xây dựng một lớp cơ sở và hai lớp dẫn xuất.

```
class Dog
{
    public string Name;
    public int Weight;
    public virtual void Speak()
    {
        Console.WriteLine("ruff!");
    }
    public void DrinkWater()
    {
        Console.WriteLine("gulp");
    }
}
```

Hình 36 – Lớp cơ sở và phương thức virtual khi thực hiện tính đa hình

Hình trên minh họa cho lớp cơ sở trong đó có một phương thức ảo (virtual). Phương thức này sẽ được ghi đè (override) ở lớp cơ sở. Các hình tiếp theo trình bày việc xây dựng hai lớp dẫn xuất trong mỗi lớp, phương thức *Speak()* được định nghĩa lại bằng cách ghi đè (override). Nếu người lập trình không sử dụng từ khóa *override* thì Visual Studio vẫn không báo lỗi. Tuy nhiên, phương thức *Speak()* trong lớp dẫn xuất được hiểu là phương thức riêng của chính lớp dẫn xuất đó và phương thức này chỉ được gọi trực tiếp từ lớp dẫn xuất.

```
class GermanShepard:Dog
{
    public void OnGuard()
    {
        Console.WriteLine("In guard mode");
    }
    public override void Speak()
    {
        Console.WriteLine("RUFF, RUFF, RUFF");
    }
}
```

Hình 37 – Lớp dẫn xuất thứ nhất có phương thức override

```
class JackRussell:Dog
{
    public void Chew()
    {
        Console.WriteLine("I am chewing your favorite shoes");
    }
    public override void Speak()
    {
        Console.WriteLine("Yap, Yap, Yap");
    }
}
```

Hình 38 – Lớp dẫn xuất thứ hai có phương thức override

Để minh họa cho khả năng ứng xử của những đối tượng thuộc những lớp khác nhau khi chúng đều triệu gọi một phương thức ta xây dựng một mảng gồm 3 đối tượng trong đó đối tượng thứ nhất thuộc lớp cơ sở, hai đối tượng còn lại thuộc lớp dẫn xuất. Khi gọi phương thức *Speak()* cho từng đối tượng, chương trình sẽ cho ra các kết quả khác nhau do tính năng Polymorphism.

```
static void Main(string[] args)
{
    Dog[] ListOfDog = new Dog[3];
    Dog Michael = new Dog();
    GermanShepard Simon = new GermanShepard();
    JackRussell Daisy = new JackRussell();
    ListOfDog[0] = Michael;
    ListOfDog[1] = Simon;
    ListOfDog[2] = Daisy;
    for (int i = 0; i < 3; i++)
    {
        ListOfDog[i].Speak();
    }
    Console.ReadLine();
}
```

Hình 39 – Minh họa việc sử dụng Polymorphism

Trong hình trên, mỗi phần tử của một mảng được gán những đối tượng khác nhau. Vì các đối tượng đều thuộc một lớp cơ sở hoặc thuộc lớp dẫn xuất từ chính lớp cơ sở đó nên việc gán những đối tượng như thế vào cùng một mảng không gây ra xung đột.. Cũng trong ví dụ trên, khi thực hiện một vòng lặp duyệt qua tất cả các đối tượng mà trong đó, ứng với mỗi đối tượng, phương thức *Speak()* cụ thể sẽ được gọi và kết quả thu được sẽ khác nhau qua từng phương thức.

IX. Interface

Interface được một tác giả khác dịch thành các thuật ngữ là *giao diện* hoặc *giao tiếp*. Thuật ngữ *Interface* được dùng để chỉ một lớp trừu tượng. Lớp trừu tượng này được xây dựng nhằm thuận tiện hóa việc phát triển chương trình theo phong cách xây dựng thành phần cơ sở (*component-based*). *Interface* cung cấp những thỏa thuận chung cho phép các thành phần có thể làm việc với nhau. *Interface* được sử dụng rất đặc lực trong việc xử lý các sự kiện (*Event Handler*) khi xây dựng các đối tượng giao diện người dùng (*GUI – Graphic User Interface*). Người học có thể tham khảo mô hình xử lý sự kiện *ActionPerformce* khi xử lý *Button* trong học phần Java đã học trước đó.

Interface không phải là một lớp hoàn chỉnh nên có bất cứ mã cài đặt nào và không thể được dùng để khởi tạo đối tượng. *Interface* chỉ có các thuộc tính, các phương thức (phương thức này chỉ có tên khai báo mà không có mã cài đặt). Tất cả các thuộc tính, phương thức phải được khai báo với chỉ dẫn truy cập là *public*.

Khi một lớp thực hiện việc kế thừa từ một hoặc nhiều *Interface* thì trong lớp đó, tất cả các phương thức thuộc các *Interface* đều phải được định nghĩa hoàn chỉnh.

XỬ LÝ BIỆT LỆ

I. Mục tiêu

Trong chương này giáo trình sẽ trình bày quá trình nắm bắt và xử lý lỗi khi lập trình C#. Những chủ đề chính được trình bày trong chương này như sau:

- Biệt lệ (Exception)
- Xử lý biệt lệ (Exception Handler)
- Xử lý nhiều biệt lệ lồng nhau
- Xử lý nhiều biệt lệ song song

Cuối chương này người học sẽ có đủ kiến thức để xây dựng một chương trình tường minh trong đó các tình huống gây lỗi thông thường đều được nắm bắt để giải quyết.

II. Biệt lệ (Exception)

II.1. Chương trình và lỗi

Chương trình (theo quan điểm của công nghệ thông tin) là một khái niệm dùng để chỉ tập hợp các mệnh lệnh có trật tự mà dựa vào đó máy tính sẽ thi hành. Quá trình con người soạn thảo tập hợp các lệnh cho chương trình được gọi là lập trình. Trong quá trình lập trình, có thể vì lý do này hoặc lý do khác mà kết quả thu được của chương trình không như mong muốn, những tình huống không mong muốn như thế gọi chung là lỗi (error hoặc defect).

Trong lĩnh vực lập trình, lỗi thường được chia thành 2 loại chính: lỗi tiền biên dịch (pre-compiled error) và lỗi khi thực thi (runtime error).

Lỗi tiền biên dịch (pre-compiled error) là những lỗi xuất hiện ngay khi xây dựng chương trình và được trình biên dịch thông báo trong quá trình biên dịch từ ngôn ngữ lập trình sang ngôn ngữ máy hoặc ngôn ngữ trung gian. Các lỗi thuộc dạng này thông thường là các lỗi liên quan đến cú pháp, liên quan đến khai báo dữ liệu hoặc liên quan các câu lệnh vốn được hỗ trợ sẵn. Các lỗi tiền biên dịch thông thường không nghiêm trọng và thường được phát hiện cũng như chỉnh sửa dễ dàng. Một số IDE hiện nay hỗ trợ việc kiểm tra lỗi tự động ngay trong quá trình soạn thảo điều đó làm cho các lỗi về cấu trúc giảm đi đáng kể. Các IDE hỗ trợ tính năng này bao gồm: Visual Studio (tất cả các phiên bản), NetBean, Eclipse, Jbuilder, ...

Lỗi khi thực thi (runtime error) là những lỗi xảy ra khi chạy chương trình. Đây là một dạng lỗi tiềm ẩn vì khi chương trình chạy đến một trạng thái nhất định mới sinh ra lỗi. Các lỗi thuộc dạng này rất nghiêm trọng và rất khó khắc phục vì lúc đó, chương trình đã được biên dịch sang mã máy hoặc mã trung gian. Các lỗi thuộc dạng này thường có rất nhiều nguyên nhân: có thể do thuật toán, có thể sự không tương thích của hệ thống khi dữ liệu giãn nở, hoặc cũng có thể do chính người lập trình không lường hết được tất cả các trạng thái có thể xảy ra

đối với chương trình. Có một vài ví dụ cụ thể cho các lỗi dạng này mà người lập trình thường gặp phải như: lỗi khi chuyển kiểu từ kiểu chuỗi ký tự sang kiểu số nguyên, lỗi khi truy xuất đến phần tử nằm ngoài mảng, lỗi khi kết nối cơ sở dữ liệu, ...

II.2. Khái niệm biệt lệ

Biệt lệ (Exception) là khái niệm được sử dụng trong các ngôn ngữ lập trình bậc cao dùng để chỉ những biến cố không mong đợi khi chạy chương trình. Bản chất biệt lệ là những lỗi xảy ra trong quá trình thực thi (runtime error).

Các ngôn ngữ lập trình bậc cao như Java và .Net đều cung cấp một cơ chế xử lý biệt lệ (gọi là cơ chế try/catch) và đặc tả các lớp đặc trưng cho từng dạng biệt lệ mà một chương trình thường gặp.

III. Xử lý biệt lệ (Exception Handler)

III.1. Cơ chế try/catch

Cơ chế try/catch là cơ chế được Visual Studio đưa ra để xử lý các biệt lệ. Cơ chế này được lập trình theo cú pháp sau:

```
try
{
    harmful-statement(s);
}
catch (Exception-type identifier)
{
    behavior-statement(s);
}
[finally
{
    final-statement(s);
}]
```

Các lệnh có nguy cơ gây lỗi (*harmful-statement(s);*) được đặt trong khối *try*. Các lệnh này có thể bao gồm các lệnh chuyển đổi kiểu dữ liệu (từ kiểu chuỗi sang kiểu số), lệnh liên quan đến nhập xuất và thậm chí cả toán tử chia... Khi gặp một lệnh gây ra lỗi trong khối *try*, chương trình thực thi sẽ lập tức triệu gọi các lệnh trong khối *catch* tương ứng để thực thi.

Khối *catch* là khối lệnh thực hiện các ứng xử khi có lỗi, trong khối này, thông thường người lập trình thực hiện việc hiển thị lỗi. Khi xây dựng khối *catch*, đầu tiên phải khai báo loại biệt lệ (*Exception-type*). Tất cả *Exception-type* là những lớp kế thừa từ lớp *Exception*. Trong mỗi lớp đều có phương thức *ToString()*, phương thức này được dùng để hiển thị thông tin lỗi cho người sử dụng. Với mỗi dạng câu lệnh sẽ có một *Exception-type* tương ứng, người lập trình xem hướng dẫn chi tiết của các phương thức trong bộ hướng dẫn MSDN để chọn *Exception-type* tương ứng. Trong trường hợp người lập trình không quan tâm đến các loại biệt lệ, người lập trình đó có thể sử dụng dạng cơ bản là *Exception* cho tất cả các biệt lệ. Phương thức *ToString()* thích hợp sẽ tự động triệu gọi dựa theo tính đa hình của hướng đối

tượng (*Polymorphism*) và chương trình luôn hiển thị chính nội dung lỗi cho dù tất cả các dạng biệt lệ đều khai báo chung là *Exception*.

Khối lệnh *finally* được dùng để chứa các lệnh xử lý cuối cùng (*final-statement(s)*). Các lệnh trong khối lệnh này được tự động triệu gọi cho dù các lệnh trong khối *try* có sinh ra lỗi hay không. Người lập trình không nhất thiết phải khai báo khối lệnh *finally*, tuy vậy, khối lệnh này rất hữu ích trong việc thu dọn rác hoặc giải phóng vùng nhớ.


Ví dụ sau trình bày việc tính bình phương của một số nguyên được nhập từ bàn phím. Nếu dữ liệu nhập vào không phải số nguyên thì chương trình sẽ báo lỗi.

```
static void Main(string[] args)
{
    Console.WriteLine("Nhập 1 số x: ");
    string temp;
    temp = Console.ReadLine();
    int x = 0;
    try
    {
        x = Convert.ToInt32(temp);
        Console.WriteLine("x^2 = " + x * x);
    }
    catch (FormatException e)
    {
        Console.WriteLine("Số nguyên được nhập vào không đúng định dạng.");
        Console.WriteLine("Error " + e.ToString());
    }
    Console.ReadLine();
}
```

Hình 40 – Ví dụ về xử lý biệt lệ

Trong ví dụ trên, lệnh *Convert.ToInt32()* thực hiện việc chuyển từ kiểu chuỗi ký tự sang kiểu số nguyên và lệnh này có khả năng sinh ra lỗi khi chuỗi ký tự nhập vào không đúng định dạng số nguyên. Khi xảy ra lỗi thì chương trình sẽ không chạy những dòng tiếp theo trong khối *try* mà chuyển sang khối *catch*. Trong khối *catch*, các lệnh hiển thị lỗi được thực hiện.

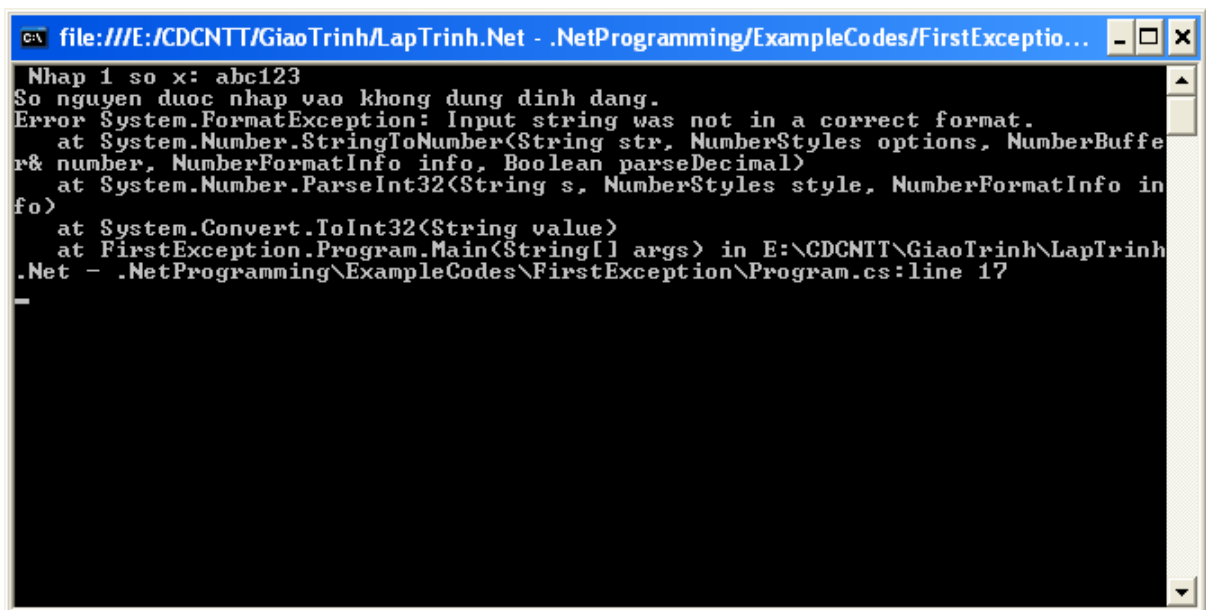
Trong trường hợp dữ liệu được nhập vào đúng định dạng của kiểu số nguyên, chương trình sẽ chạy đúng kết quả như sau.



```
file:///E:/CDCNTT/GiaoTrinh/LapTrinh.Net - .NetProgramming/ExampleCodes/FirstExceptio...
Nhap 1 so x: 15
x^2 = 225
```

Hình 41 – Kết quả chạy chương trình với giá trị đầu vào hợp lệ

Trong trường hợp dữ liệu nhập vào không đúng với định dạng số nguyên, chương trình sẽ báo lỗi và hiển thị cụ thể lỗi ra màn hình.



```
file:///E:/CDCNTT/GiaoTrinh/LapTrinh.Net - .NetProgramming/ExampleCodes/FirstExceptio...
Nhap 1 so x: abc123
So nguyen duoc nhap vao khong dung dinh dang.
Error System.FormatException: Input string was not in a correct format.
   at System.Number.StringToNumber(String str, NumberStyles options, NumberBuffer& number, NumberFormatInfo info, Boolean parseDecimal)
   at System.Number.ParseInt32(String s, NumberStyles style, NumberFormatInfo info)
   at System.Convert.ToInt32(String value)
   at FirstException.Program.Main(String[] args) in E:\CDCNTT\GiaoTrinh\LapTrinh.Net - .NetProgramming\ExampleCodes\FirstException\Program.cs:line 17
```

Hình 42 – Kết quả chạy chương trình khi xảy ra biệt lệ

Trong khối *catch*, lệnh *e.ToString()* thường được gọi để hiển thị lỗi, lỗi này được hiển thị chính xác với nội dung lỗi và dòng mã sinh ra lỗi. Với cơ chế xử lý lỗi này, người lập trình dễ dàng tìm ra lỗi và chỉnh sửa phù hợp.

III.2. Xử lý biệt lệ lồng nhau

Trong quá trình lập trình, có một số tình huống mà chương trình phải thực hiện nhiều lệnh có khả năng gây lỗi liên tiếp. Để nắm bắt lỗi một cách chính xác, lập trình viên thường thực

hiện việc bắt lỗi theo từng bước, nghĩa là xử lý biệt lệ của lệnh này xong thì sẽ xử lý biệt lệ ở lệnh tiếp theo.

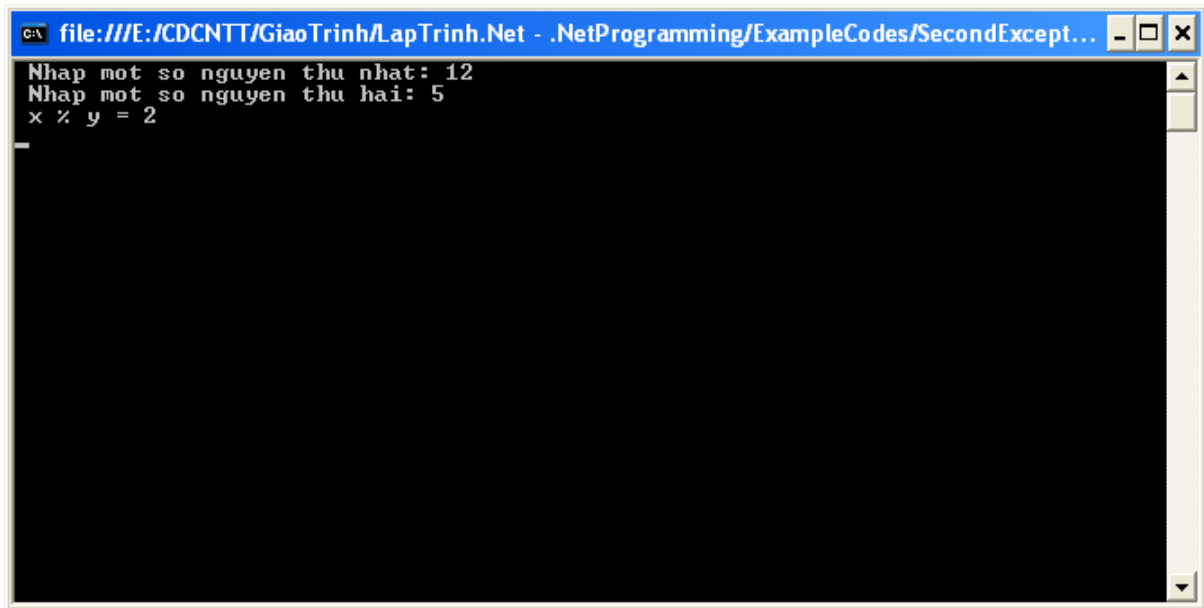
Ví dụ sau đây trình bày việc lập trình để tính số dư khi chia số nguyên x cho số nguyên y, hai số nguyên này được nhập từ bàn phím. Trong ví dụ này, chương trình có hai lệnh có khả năng sinh lỗi. Lệnh thứ nhất là lệnh chuyển kiểu khi thực hiện việc chuyển từ kiểu chuỗi ký tự sang kiểu số thực, lệnh này gây ra biệt lệ khi chuỗi ký tự được nhập vào không đúng định dạng số thực. Lệnh thứ hai là lệnh thực hiện phép chia lấy số dư (%), lệnh này gây ra biệt lệ khi số chia bằng 0.

```
static void Main(string[] args)
{
    Console.WriteLine("Nhập một số nguyên thứ nhất: ");
    string tempX = Console.ReadLine();
    Console.WriteLine("Nhập một số nguyên thứ hai: ");
    string tempY = Console.ReadLine();
    int x = 0;
    int y = 0;
    try
    {
        x = Convert.ToInt32(tempX);
        y = Convert.ToInt32(tempY);
        int z = 0;
        try
        {
            z = x % y;
            Console.WriteLine("x % y = " + z);
        }
        catch (DivideByZeroException e1)
        {
            Console.WriteLine("Không thể chia cho 0");
            Console.WriteLine(e1.ToString());
        }
    }
    catch (FormatException e)
    {
        Console.WriteLine("Số thực được nhập vào không đúng định dạng.");
        Console.WriteLine(e.ToString());
    }
    Console.ReadLine();
}
```

Hình 43 – Minh họa việc xử lý các biệt lệ lồng nhau

Trong ví dụ trên, khối lệnh *try/catch* bên trong thực hiện việc xử lý biệt lệ khi chia số x cho số y để lấy số dư. Nếu $y = 0$ thì biệt lệ được tạo ra, biệt lệ này thuộc lớp *DivideByZeroException*. Khối lệnh *try/catch* bên ngoài thực hiện việc xử lý biệt lệ khi chuyển kiểu từ kiểu số chuỗi ký tự sang kiểu số nguyên (Xem chi tiết ở phần III.1 trước đó).

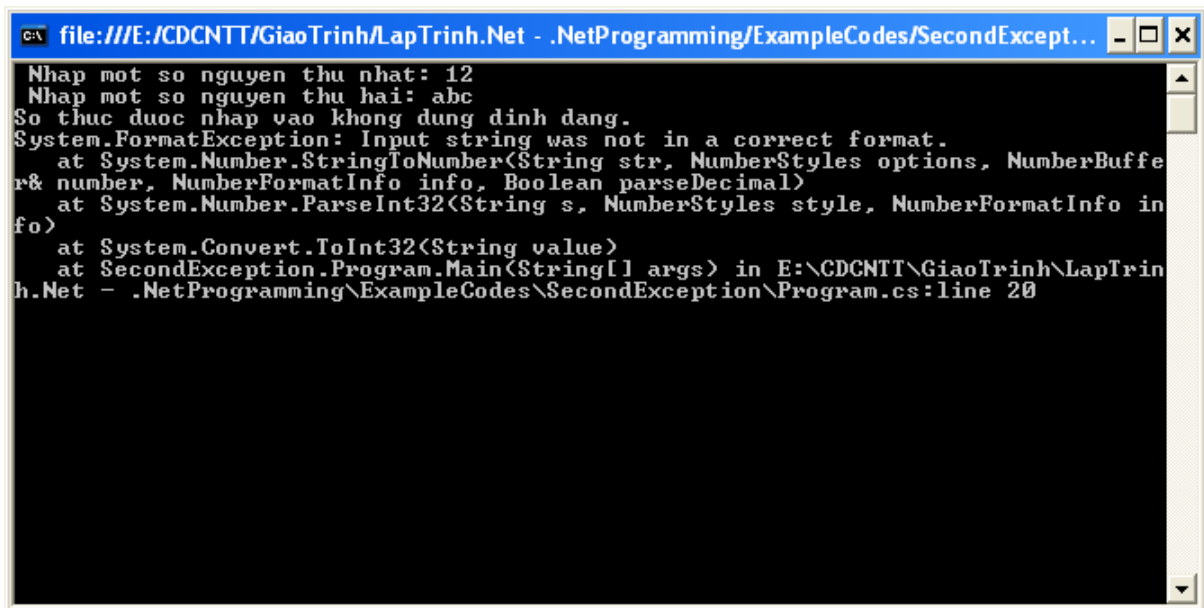
Trong trường hợp cả hai số nhập vào x, y đều có định dạng phù hợp và y khác 0 thì chương trình chạy và cho ra kết quả đúng (Như hình).



```
file:///E:/CDCNTT/GiaoTrinh/LapTrinh.Net - .NetProgramming/ExampleCodes/SecondExcept...
Nhap mot so nguyen thu nhat: 12
Nhap mot so nguyen thu hai: 5
x % y = 2
```

Hình 44 – Kết quả chạy chương trình với giá trị đầu vào hợp lệ

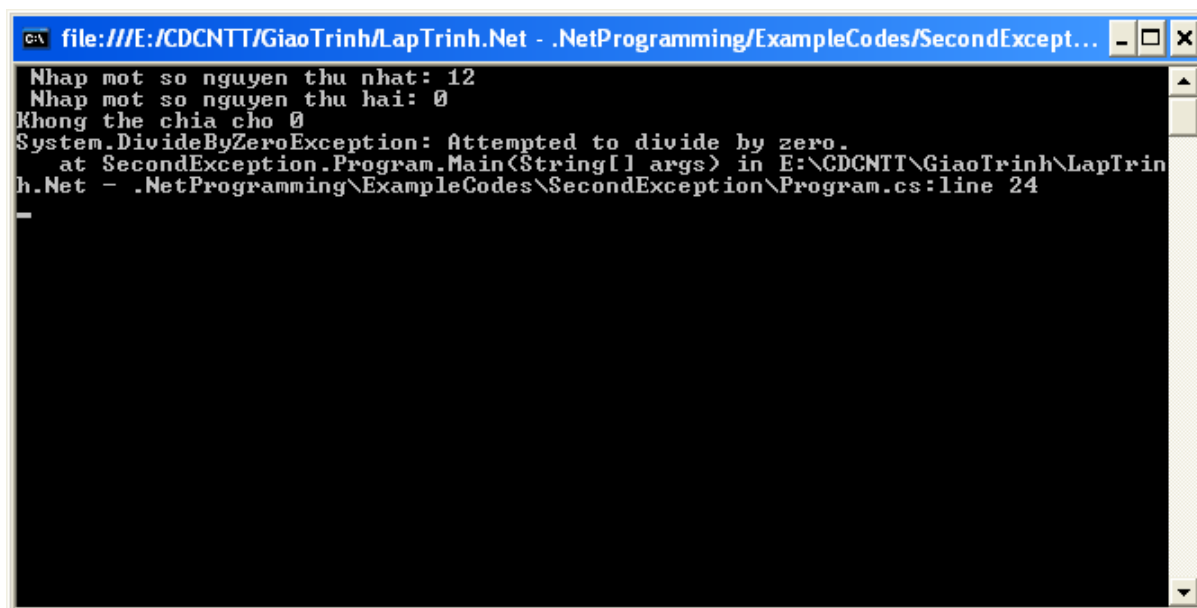
Trong trường hợp một trong hai số nhập vào có định dạng không phải là số nguyên thì biệt lệ đầu tiên được xử lý. Khi đó chương trình sẽ thông báo lỗi và chương trình không tiếp tục chạy để thực hiện phép toán chia (Như hình).



```
file:///E:/CDCNTT/GiaoTrinh/LapTrinh.Net - .NetProgramming/ExampleCodes/SecondExcept...
Nhap mot so nguyen thu nhat: 12
Nhap mot so nguyen thu hai: abc
So thuc duoc nhap vao khong dung dinh dang.
System.FormatException: Input string was not in a correct format.
   at System.Number.StringToNumber(String str, NumberStyles options, NumberBuffer& number, NumberFormatInfo info, Boolean parseDecimal)
   at System.Number.ParseInt32(String s, NumberStyles style, NumberFormatInfo info)
   at System.Convert.ToInt32(String value)
   at SecondException.Program.Main(String[] args) in E:\CDCNTT\GiaoTrinh\LapTrinh.Net - .NetProgramming\ExampleCodes\SecondException\Program.cs:line 20
```

Hình 45 – Biệt lệ xảy ra khi giá trị nhập vào không đúng định dạng số nguyên

Trong trường hợp cả hai số nhập vào đúng định dạng số nguyên nhưng y có giá trị 0 thì biệt lệ tiếp theo được xử lý. Trong trường hợp này phép chia lấy số dư sẽ không thực hiện được và chương trình sẽ thông báo lỗi (Như hình).



```
file:///E:/CDCNTT/GiaoTrinh/LapTrinh.Net - .NetProgramming/ExampleCodes/SecondExcept...
Nhập một số nguyên thu nhất: 12
Nhập một số nguyên thu hai: 0
Không thể chia cho 0
System.DivideByZeroException: Attempted to divide by zero.
   at SecondException.Program.Main(String[] args) in E:\CDCNTT\GiaoTrinh\LapTrinh.Net - .NetProgramming\ExampleCodes\SecondException\Program.cs:line 24
```

Hình 46 – Biệt lệ xảy ra khi số chia bằng 0

Với ví dụ trên, ta có thể dễ dàng nhận thấy việc xử lý biệt lệ lồng nhau một cách phân cấp sẽ làm cho chương trình chặt chẽ và lỗi được nắm bắt dễ dàng. Với mỗi biệt lệ được thông báo ra màn hình, người lập trình sẽ dễ dàng nhận thấy và chỉnh sửa ở đoạn mã phù hợp. Tuy nhiên, việc xử lý nhiều biệt lệ lồng nhau một cách phân cấp làm cho mã nguồn trở của chương trình thêm nặng nề, các khối *try/catch* xuất hiện nhiều lần và lồng nhau làm cho cấu trúc của chương trình trở nên phức tạp.

III.3. Xử lý biệt lệ song song

Như đã trình bày ở phần III.2, khi có nhiều lệnh có khả năng gây ra biệt lệ thì ta cần phải thực hiện xử lý tất cả các biệt lệ. Để xử lý các biệt lệ, ta thực hiện việc sử dụng cấu trúc *try/catch* một cách tuần tự: lệnh nào thực hiện trước thì xử lý trước, sau đó, ngay bên trong khối *try*, người lập trình sử dụng một cấu trúc *try/catch* khác để xử lý biệt lệ cho lệnh tiếp theo. Cách làm này tuy xử lý tất cả các biệt lệ nhưng cũng làm cho chương trình trở nên phức tạp vì phải sử dụng nhiều cấu trúc *try/catch* lồng nhau. Để tối ưu hóa mã lệnh và làm cho chương trình thêm tường minh, thông thường người lập trình thực hiện việc xử lý biệt lệ song song.

Xử lý biệt lệ song song là quá trình xử lý biệt lệ trong đó chỉ có một khối *try* nhưng lại có nhiều khối *catch*. Tất cả các lệnh cần thiết đều đặt trong khối *try* và mỗi khối *catch* sẽ thực hiện việc bắt một biệt lệ tương ứng. Khi chạy chương trình, ứng với một biệt lệ được tạo ra, chương trình thực thi của .Net Framework sẽ tự động tham chiếu đến khối *catch* có biệt lệ tương ứng để gọi lệnh xử lý.

Ví dụ sau trình bày việc cải tiến ví dụ ở mục III.2, trong ví dụ này, biệt lệ được xử lý bằng cách sử dụng hai khối *catch* khác nhau để bắt lấy 2 biệt lệ khác nhau. Kết quả chạy chương trình cải tiến này hoàn toàn giống với kết quả khi chạy ví dụ trong mục III.2

```
static void Main(string[] args)
{
    Console.Write(" Nhập một số nguyên thứ nhất: ");
    string tempX = Console.ReadLine();
    Console.Write(" Nhập một số nguyên thứ hai: ");
    string tempY = Console.ReadLine();
    int x = 0;
    int y = 0;
    try
    {
        x = Convert.ToInt32(tempX);
        y = Convert.ToInt32(tempY);
        int z = 0;
        z = x % y;
        Console.WriteLine(" x % y = " + z);
    }
    catch (FormatException e)
    {
        Console.WriteLine("Số thực được nhập vào không đúng định dạng.");
        Console.WriteLine(e.ToString());
    }
    catch (DivideByZeroException e1)
    {
        Console.WriteLine("Không thể chia cho 0");
        Console.WriteLine(e1.ToString());
    }
    Console.ReadLine();
}
```

Hình 47 – Xử lý nhiều biệt lệ song song

THƯ VIỆN LIÊN KẾT ĐỘNG

I. Mục tiêu

Sau khi hiểu được nguyên lý “Lập trình hướng đối tượng với C#”, người lập trình có thể tự xây dựng được các lớp theo sự đặc tả của bài toán thực tế. Tuy nhiên, để ứng dụng trở nên trong sáng và khoa học, các lớp cần được phân loại và tập hợp thành các thư viện. Các thư viện này không những giúp cho người lập trình dễ dàng tìm kiếm, cập nhật, nâng cấp chính mã nguồn của mình mà còn giúp cho việc phát triển các ứng dụng sau này một cách dễ dàng bằng cách sử dụng lại các thư viện đã có. Trong chương này giáo trình sẽ trình bày một số khái niệm liên quan đến việc xây dựng thư viện trong các ngôn ngữ thuộc họ .Net mà tiêu biểu là ngôn ngữ lập trình C#. Những chủ đề chính được trình bày trong chương này bao gồm:

- Namespace
- Thư viện liên kết động
- Cách xây dựng một thư viện
- Cách sử dụng một thư viện
- Một số thư viện và namespace có sẵn

Cuối chương này người học sẽ có đủ kiến thức để có thể phân chia ứng dụng của mình thành những phần riêng biệt một cách khoa học để tiện cho việc phát triển sau này.

II. Thư viện trong lập trình

II.1. Khái niệm

Khi lập trình, người lập trình không nhất thiết phải định nghĩa mọi thứ mà có thể sử dụng những đoạn chương trình từ những người khác hoặc những đoạn chương trình từ một ứng dụng trước đó của chính mình. Do đó, khi lập trình, người lập trình cần phải nhúng vào chương trình của mình một hoặc nhiều đoạn mã của chương trình khác.

Trong lập trình, thư viện là một khái niệm dùng để chỉ tập hợp các đoạn mã hoặc đoạn chương trình mà các đoạn mã này được nhúng vào một chương trình khác nhằm phục vụ một số chức năng xác định. Ví dụ các thư viện phục vụ lập trình đồ họa, các thư viện điều khiển phần cứng (còn được biết đến với thuật ngữ driver), ...

Việc sử dụng thư viện có một ý nghĩa vô cùng thiết thực khi người lập trình cần phải thực hiện việc điều khiển một số thiết bị đặc thù hoặc lập trình giao diện hoặc kết nối với hệ thống phần cứng, vì người lập trình không thể hiểu hết toàn bộ thông tin về các thiết bị cũng như hệ thống đồ họa và phần cứng do đó các thông tin này được nhà sản xuất cung cấp. Thông thường, các thông tin về phần cứng và thiết bị được nhà sản xuất cung cấp cho người lập trình dưới dạng thư viện.

II.2. Phân loại thư viện

Đối với việc lập trình ứng dụng, thư viện thường được phân loại dựa vào việc nhúng thư viện đó vào chương trình. Với tiêu chí này, thư viện được phân thành hai loại: thư viện tĩnh và thư viện liên kết động

II.2.1. Thư viện tĩnh

Thư viện tĩnh là khái niệm chỉ đoạn mã hoặc đoạn chương trình được nhúng trực tiếp vào một chương trình khác. Những đoạn mã của thư viện tĩnh có thể là mã chương trình hoặc mã tiền biên dịch (pre-compiler code – một dạng mã rút gọn được hiểu bởi một ngôn ngữ lập trình đặc thù). Khi biên dịch, cả mã nguồn của chương trình lẫn mã của thư viện tĩnh được biên dịch cùng một lúc thành mã thực thi hoặc nhị phân (mã máy) và chứa trong một file của chương trình.

Đối với ngôn ngữ C thì việc sử dụng thư viện tĩnh được thực hiện thông qua lệnh `#include`, lệnh này thực hiện việc nhúng toàn bộ đoạn mã của thư viện được gọi vào chương trình mà người lập trình đang sử dụng.

Việc sử dụng thư viện tĩnh tuy giúp cho người lập trình sử dụng lại những đoạn mã đã xây dựng nhưng cách làm này vẫn có hai nhược điểm cơ bản:

- Làm tăng độ lớn của chương trình: Khi nhúng nhiều nhiều thư viện vào một chương trình thì dung lượng của chương trình sẽ phải tăng thêm vì chứa thêm mã nguồn của các thư viện cho dù người lập trình chỉ sử dụng một ít chức năng của thư viện đó. Ngoài ra, khi một thư viện tĩnh được nhúng vào nhiều chương trình thì tất cả các chương trình đều tăng dung lượng.
- Tạo ra sự không đồng bộ khi cập nhật: Khi người lập trình chỉnh sửa thư viện thì kết quả vẫn không được cập nhật vào các chương trình vốn đã nhúng thư viện đó. Để chương trình có sự cập nhật đồng bộ, người lập trình bắt buộc phải biên dịch lại chương trình để toàn bộ mã nguồn của chương trình được biên dịch lại với phiên bản mới nhất của thư viện.

Để giải quyết hai nhược điểm trên, Microsoft đã đề ra giải pháp “Thư viện liên kết động” – Dynamic Linked Library gọi tắt là dll.

II.2.2. Thư viện liên kết động

Thư viện liên kết động (Dynamic Linked Library) là khái niệm được Microsoft đưa ra khi xây dựng hệ điều hành Windows. Thư viện liên kết động được dùng để chỉ những đoạn chương trình hoặc những đoạn mã nhị phân nằm trong một file (thông thường có phần mở rộng là .dll) mà những đoạn mã này có thể được các chương trình khác gọi bằng các tham chiếu thay vì phải nhúng toàn bộ đoạn mã vào chương trình.

Với thư viện liên kết động, các chương trình sẽ giảm bớt dung lượng vì rất nhiều chương trình có thể sẽ dùng chung một thư viện, đặc biệt là các thư viện của hệ thống như thư viện đồ họa, các *driver* điều khiển thiết bị. Việc sử dụng thư viện liên kết động sẽ tạo ra sự đồng bộ khi lập trình và chỉnh sửa vì chỉ cần chỉnh sửa ngay tại thư viện, tất cả các chương trình có sử dụng thư viện sẽ tự động thay đổi mà không cần phải biên dịch lại.

Khác với thư viện tĩnh, các đoạn mã chứa trong thư viện liên kết động là những đoạn mã thực thi được (executable code). Các đoạn mã này nếu được gọi đúng cách thì vẫn có thể thực thi được. Điều này cũng nảy sinh nhược điểm là các thư viện liên kết động vẫn có thể bị nhiễm virus. Những virus này tuy không làm gián đoạn chương trình nhưng có thể làm cho chương trình bị lỗi hoặc chạy sai kết quả. Các lỗi này sẽ làm cho người lập trình khó có thể chỉnh sửa đúng đắn vì người lập trình không có khả năng can thiệp vào các thư viện của các nhà cung cấp.

Trong các ngôn ngữ thuộc họ .Net, các lớp trong các thư viện được phân nhóm. Những nhóm này được gọi là *namespace*.

III. Namespace

Namespace còn được biết dưới một thuật ngữ khác là “không gian tên”. *Namespace* là một khái niệm được đưa vào họ ngôn ngữ .Net từ phiên bản Visual Studio 2003. *Namespace* được dùng để chỉ một nhóm các lớp trong một thư viện.

Khái niệm *namespace* giúp cho việc tổ chức các thư viện được trong sáng, dễ dàng. Ngoài ra, *namespace* còn giúp người lập trình tránh né sự xung đột về tên lớp, tên hàm, tên các kiểu dữ liệu khi các tên này bị trùng.

Nhiều *namespace* có thể được đặt lồng nhau và người lập trình được tùy ý đặt tên các *namespace* như cách đặt tên biến. Tuy nhiên, thông thường *namespace* được đặt tên theo một quy ước chung của một chuẩn phần mềm. Quy ước về việc đặt tên *namespace* thông thường bao gồm tên doanh nghiệp, tên dự án, tên module và được phân cách với nhau bởi dấu chấm (dot) như: <Tên doanh nghiệp>.<Tên dự án>.<Tên module>

Ví dụ: namespace Wrox.ProCSharp.Basics;

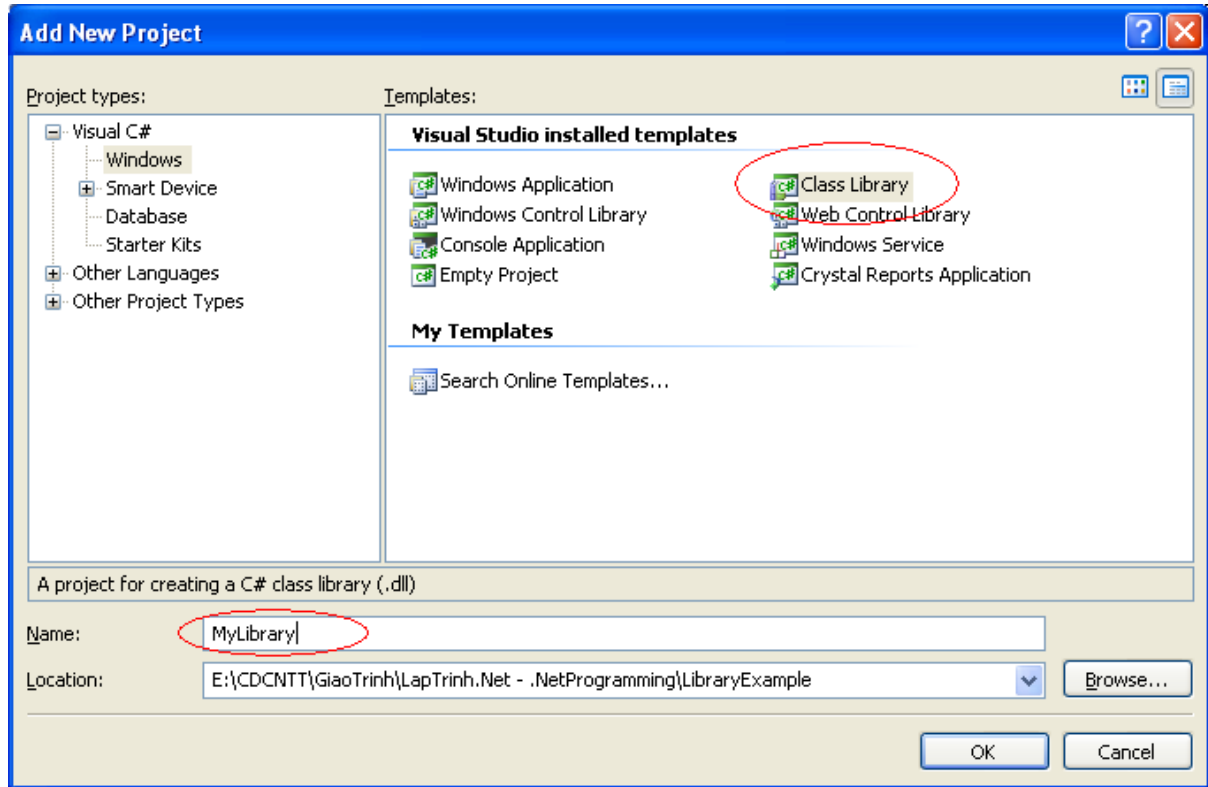
IV. Thư viện liên kết động

IV.1. Cách xây dựng thư viện với Visual Studio 2005

Trong phần này, giáo trình sẽ hướng dẫn xây dựng một thư viện có tên là MyLibrary. Thư viện này chứa *namespace* mà *namespace* này tập hợp các lớp có các hàm liên quan đến các công thức toán học. Để xây dựng thư viện này, người lập trình có thể thực hiện theo các bước sau.

IV.1.1. Tạo một project cho thư viện

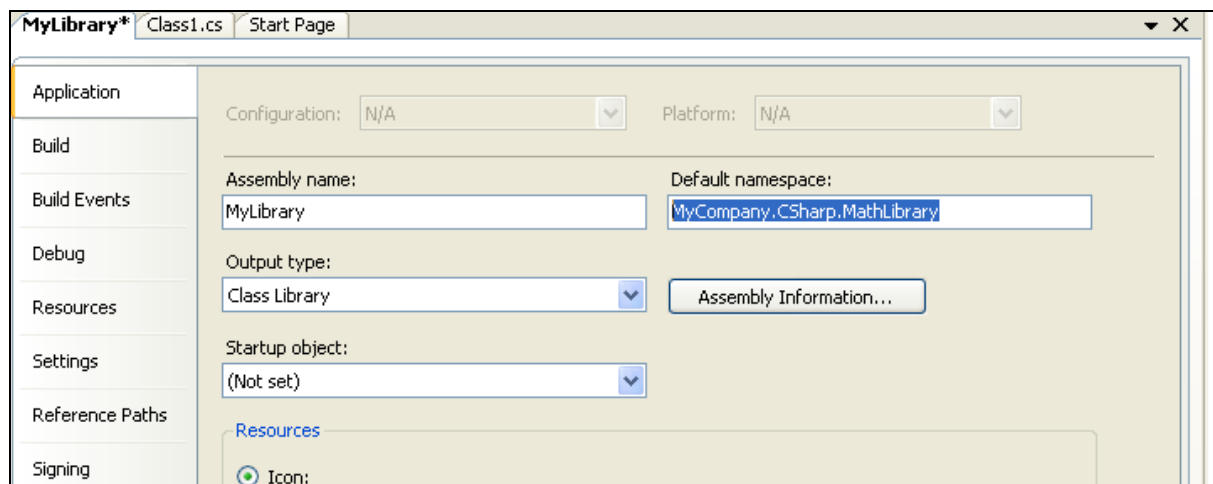
Khi tạo *project* cho thư viện, *project* này có template là Class Library. Việc khai báo tên và thư mục được thực hiện tương tự như việc tạo Console Application project đã trình bày ở phần trên. Ví dụ sau minh họa việc tạo một thư viện có tên là MyLibrary.



Hình 48 – Tạo mới một project Library

IV.1.2. Cấu hình cho project

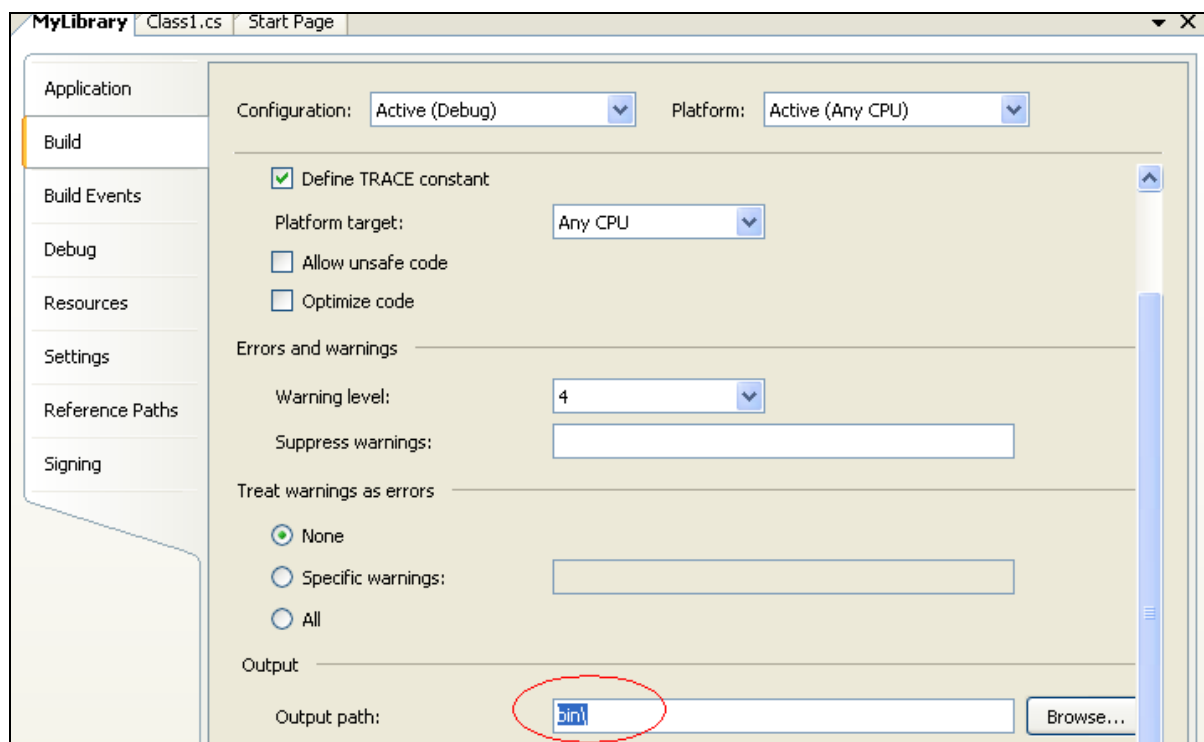
Trong bước này chúng ta sẽ thiết lập *namespace* mặc định cho thư viện và thiết lập đường dẫn cho thư viện. Khi tạo các lớp cho thư viện thì các *namespace* mặc định này sẽ tự động được gán cho các lớp. Đường dẫn của thư viện chính là vị trí của file .dll sau khi Visual Studio biên dịch thành công.



Hình 49 – Minh họa việc cấu hình namespace mặc định

Với minh họa ở hình trên, namespace mặc định được đặt tên theo chuẩn gồm tên doanh nghiệp, tên dự án, tên module là `MyCompany.CSharp.MathLibrary`. Namespace này được

dùng để chỉ tập hợp các lớp có các hàm liên quan đến các công thức toán học. Sau khi thiết lập *namespace* mặc định, người lập trình sẽ thiết lập thêm đường dẫn của thư viện.



Hình 50 – Minh họa việc thiết lập đường dẫn cho thư viện

Với minh họa ở hình trên thì đường dẫn của thư viện là thư mục *bin*, thư mục này nằm ngay trong thư mục chứa *project*. Sau khi biên dịch thành công, file *MyLibrary.dll* sẽ được tạo ra ngay trong thư mục *bin* này.

Trong phần cấu hình cho project còn có nhiều tham số khác. Tùy vào yêu cầu cụ thể, người lập trình có thể cấu hình theo ý của chính mình. Các tham số khác về cấu hình, người học tự nghiên cứu và tìm hiểu thêm.

IV.1.3. Xây dựng lớp và phương thức cần thiết

Trong bước này, người lập trình có thể xây dựng các lớp. Trong các lớp, các thuộc tính và phương thức được xây dựng theo nguyên lý lập trình hướng đối tượng. Khi xây dựng các lớp trong thư viện, vì các lớp này cần được truy xuất từ một ứng dụng khác (truy xuất từ bên ngoài), do đó chỉ dẫn truy cập của các lớp thông thường là *public*. Ví dụ sau trình bày việc xây dựng một lớp trong thư viện, lớp này có tên là *MyMath*. Trong lớp *MyMath*, ví dụ trình bày việc xây dựng một phương thức *IsPrime(int x)*, phương thức này là một hàm thực hiện việc kiểm tra một số có phải là số nguyên tố hay không. Phương thức *IsPrime* trả về *true* nếu giá trị đầu vào là số nguyên tố và trả về *false* nếu ngược lại.

Visual Studio từ phiên bản 2003 trở đi cung cấp một giải pháp viết chú thích (write comment) cho cả một phương thức. Với giải pháp này các ghi chú của một phương thức được ghi thành từng dòng và bắt đầu bằng ba dấu chia (triple slashes). Người lập trình sẽ viết ghi chú và ghi chú này được sử dụng như một hướng dẫn cho những người lập trình sau này

khi sử dụng. Chú ý, người lập trình có thể soạn thảo ghi chú bằng Tiếng Việt có dấu theo chuẩn Unicode.

```
namespace MyCompany.CSharp.MathLibrary
{
    public class MyMath
    {
        /// <summary>
        /// Hàm IsPrime thực hiện việc kiểm tra một số nguyên có phải
        /// là số nguyên tố hay không.
        /// </summary>
        /// <param name="x">Giá trị đầu vào là một số nguyên</param>
        /// <returns>
        /// Kết quả trả về là true nếu giá trị đầu vào là số nguyên tố
        /// </returns>
        public static bool IsPrime(int x)
        {
            if (x < 2) return false;
            for (int i = 2; i < Math.Sqrt(x); i++)
                if (x % i == 0) return false;
            return true;
        }
    }
}
```

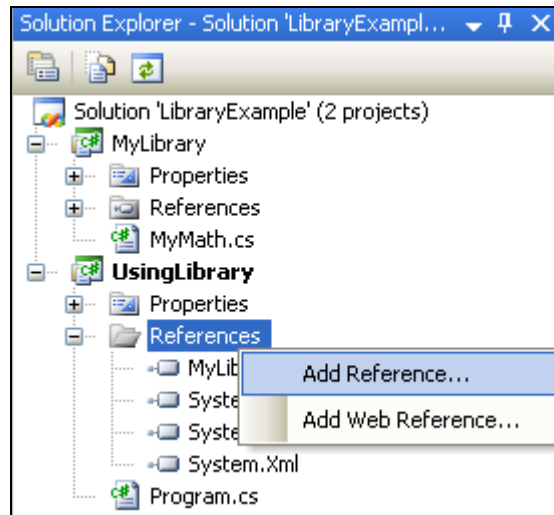
Hình 51 – Minh họa việc xây dựng lớp trong thư viện

IV.2. Cách sử dụng thư viện

Lập trình viên Visual Studio .Net có thể sử dụng một số thư viện có sẵn ở dạng file .dll mà file này cũng được xây dựng bằng Visual Studio .Net. Đối với Visual Studio .Net, các thư viện bằng những ngôn ngữ khác nhau (trong họ .Net) có thể tham chiếu lẫn nhau và cũng có thể được sử dụng bởi những chương trình khác trong họ .Net. Trong phần này giáo trình sẽ trình bày một ví dụ thực hiện việc sử dụng phương thức IsPrime(x) của thư viện trên. Ví dụ này thực hiện việc hiển thị tất cả các số nguyên tố nhỏ hơn 100. Để sử dụng thư viện vừa được xây dựng, ta thực hiện các bước sau.

IV.2.1. Tạo thêm tham chiếu (add reference).

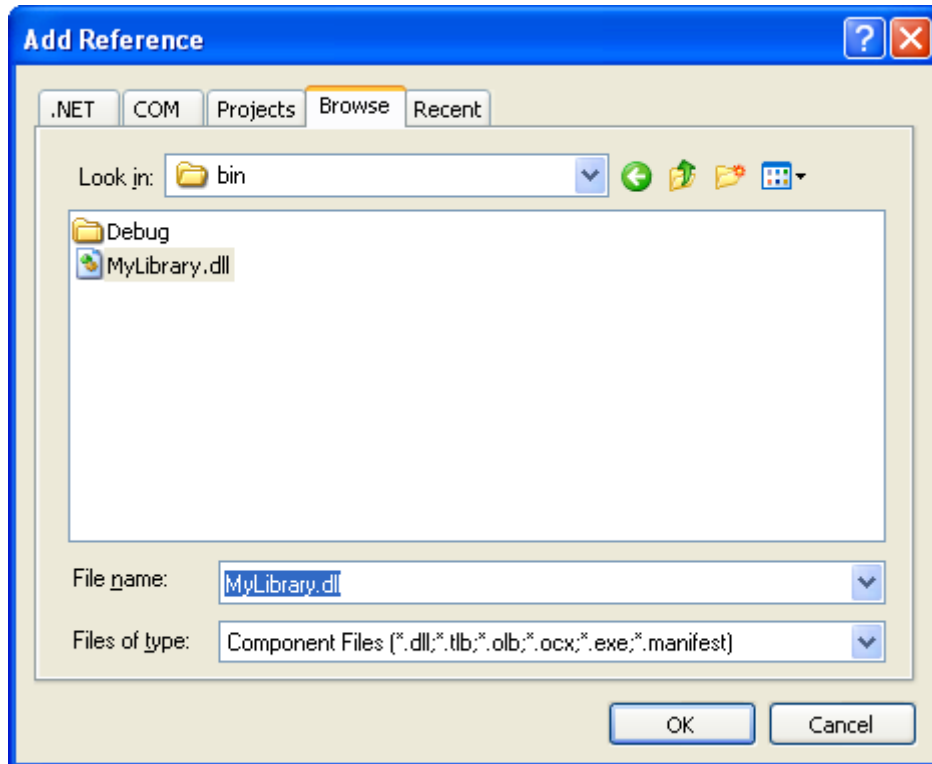
Để tạo tham chiếu đến thư viện, ta thực hiện việc kích chuột phải (right click) vào mục *References* trong cửa sổ *Solution Explorer* rồi chọn *Add Reference*. Việc tạo thêm tham chiếu bản chất là sự khai báo đường dẫn đến thư viện cần tham chiếu. Với khai báo này, trình biên dịch sẽ thực hiện việc truy xuất đến các file .dll và sử dụng các lớp, phương thức trong các file đó.



Hình 52 – Tạo thêm tham chiếu cho ứng dụng

IV.2.2. Khai báo tham chiếu

Trong phần này, Visual Studio sẽ cung cấp cho người lập trình một giao diện tùy chọn để người lập trình lựa chọn một thư viện tương ứng. Trong giao diện này có các thẻ (tab) tương ứng với từng loại thư viện bao gồm: thẻ .NET chứa các thư viện có sẵn của .NET Framework, thẻ COM chứa các thư viện của ứng dụng COM, thẻ Project chứa các Project có dạng thư viện trong cùng Solution, thẻ Browse chứa tùy chọn đến file .dll độc lập.



Hình 53 – Khai báo tham chiếu

Trong phần này, giáo trình trình bày việc tham chiếu đến một thư viện độc lập bằng cách lựa chọn đến file .dll trong thẻ Browse. Cụ thể trong ví dụ ở hình trên là file MyLibrary.dll trong thư mục *bin* của project MyLibrary vốn được trình bày ở phần trước.

IV.2.3. Sử dụng thư viện

Sau khi khai báo tham chiếu, người lập trình sẽ phải khai báo việc sử dụng namespace trong thư viện. Namespace này được khai báo bằng lệnh *using*. Lệnh này có cú pháp và chức năng tương tự với lệnh *import* của ngôn ngữ Java (của học phần “lập trình Java” trước đó).

Khi sử dụng, người lập trình thực hiện việc khai báo đối tượng hoặc việc triệu gọi phương thức giống như việc sử dụng các lớp có sẵn. Nếu trong quá trình xây dựng thư viện, người lập trình có thực hiện việc viết chú thích cho các phương thức thì khi triệu gọi, các ghi chú này sẽ được hiển thị như các hướng dẫn của các phương thức có sẵn của .Net Framework.

```
using System;
using MyCompany.CSharp.MathLibrary;

namespace UsingLibrary
{
    class Program
    {
        static void Main(string[] args)
        {
            for (int i = 1; i < 100; i++)
                if (MyMath.IsPrime(i)) Console.WriteLine(i + " ");
            Console.ReadLine();
        }
    }
}
```

Hình 54 – Minh họa việc sử dụng thư viện

Hình trên trình bày việc sử dụng hàm *IsPrime(x)* của thư viện *MyLibrary*. Hàm này thuộc lớp *MyMath* và được nhóm vào namespace *MyCompany.CSharp.MathLibrary*.

V. Các namespace có sẵn của .Net Framework 2.0

V.1. Namespace *System.Windows.Forms*

Namespace *System.Windows.Forms* chứa các lớp dùng để tạo ứng dụng Windows với giao diện người sử dụng mang các đặc điểm ưu việt của hệ điều hành Windows.

Đây là namespace chính cung cấp các lớp dùng để xây dựng ứng dụng Windows được trình bày trong chương tiếp theo. Các lớp trong namespace này được chia thành các nhóm sau:

- *Control*, *UserControl* và *Form*: Hầu hết các lớp trong namespace *System.Windows.Forms* kế thừa từ lớp *Control*. Lớp này cung cấp các chức năng cơ

bản dùng để hiển thị trên *Form* với nhiều dạng: dạng hộp thoại (*DialogBox*), dạng cửa sổ (*Window*), dạng MDI (*Multiple Document Interface*).

- *Menus* và *Toolbars*: Nhóm này chứa các lớp dùng để tạo thanh công cụ hoặc menu. Trong nhóm này có các lớp: *ToolStrip*, *MenuStrip*, *ContextMenuStrip* và *StatusStrip*.
- *Controls*: Nhóm này chứa các lớp dùng để thiết kế giao diện người dùng. Một số lớp dùng để nhập liệu gồm: *TextBox* và *ComboBox*, một số lớp dùng để trình bày dữ liệu: *Label* và *ListView*, một số lớp dùng để làm việc với Internet như: *WebBrowser*, *HtmlDocument*.
- *Layout*: Nhóm này chứa các lớp phục vụ việc định dạng và tổ chức các đối tượng trên *Form*. Trong nhóm này có lớp *FlowLayoutPanel* cho phép sắp xếp các đối tượng theo thứ tự; lớp *TableLayoutPanel* cho phép sắp xếp đối tượng theo hàng và cột để định vị lưới; lớp *SplitContainer* cho phép phân chia vùng làm việc thành nhiều phần khác nhau.
- *Data* và *DataBinding*: Nhóm này chứa các lớp định nghĩa các kiến trúc phục vụ việc liên kết dữ liệu nguồn hoặc tập tin XML với các đối tượng hiển thị trên *Form* ví dụ như lớp *DataGridView*.

V.2. Namespace System.Data

Namespace này cung cấp các lớp truy xuất dữ liệu có cấu trúc. Namespace này thường được sử dụng khi thực hiện tương tác cơ sở dữ liệu. Trong namespace này có các namespace thành viên như sau:

- *System.Data.Common*: Cung cấp các lớp dùng chung khi kết nối với các hệ quản trị cơ sở dữ liệu khác nhau. Ví dụ: *DataAdapter*, *DataTableMapping*.
- *System.Data.SqlClient*: Cung cấp các lớp phục vụ việc kết nối với Ms SQL Server. Ví dụ: *SqlCommand*, *SqlConnection*, *SqlDataAdapter*, *SqlDataReader*.
- *System.Data.SqlTypes*: Cung cấp kiểu dữ liệu dùng trong SQL Server. Ví dụ: *SqlBinary*, *SqlDecimal*, *SqlDateTime*.

LẬP TRÌNH ỨNG DỤNG WINDOWS

I. Mục tiêu

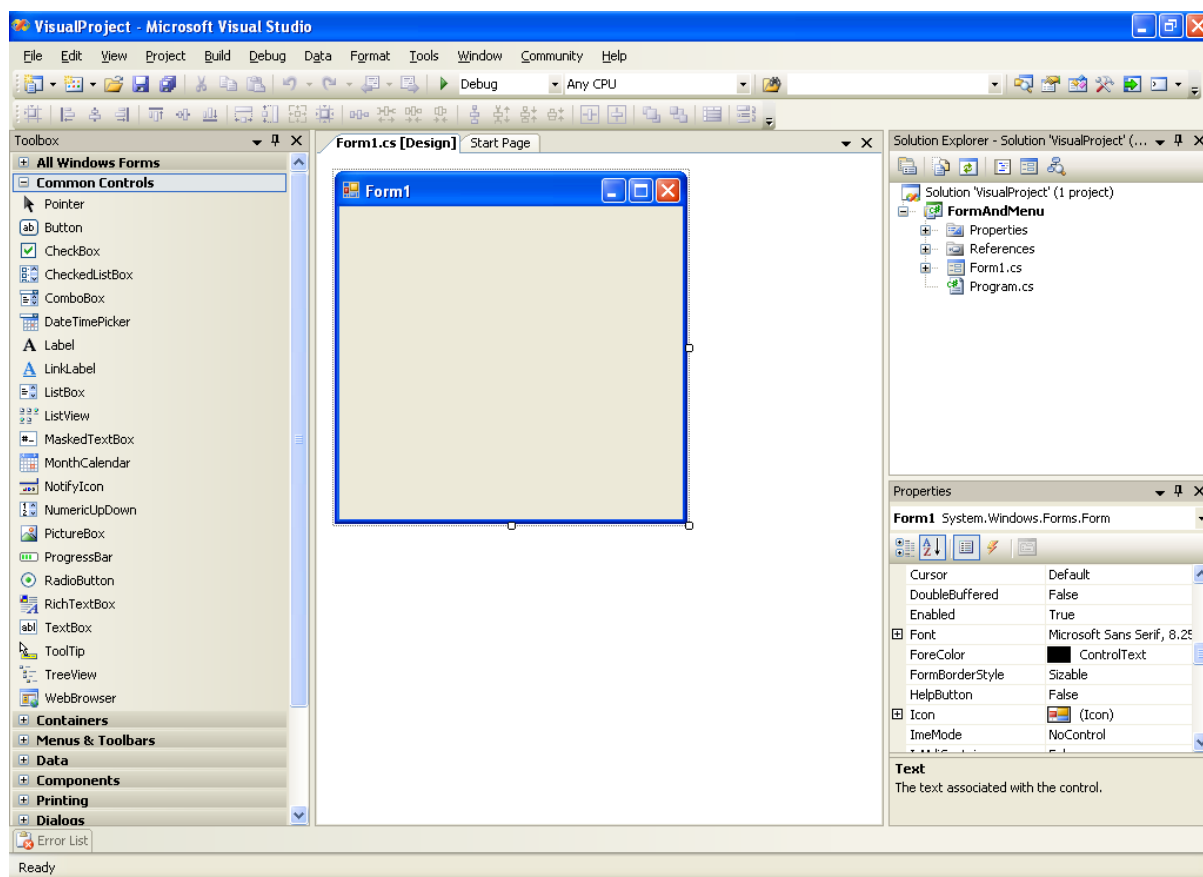
Trong chương này, giáo trình sẽ trình bày cách xây dựng một ứng dụng Windows dựa trên .Net Framework. Những chủ đề chính được trình bày trong chương này bao gồm:

- Form
- Label
- TextBox
- ComboBox, ListBox
- CheckBox, RadioButton
- Button

Kết thúc chương này, người học sẽ có đủ kiến thức để xây dựng một ứng dụng nhỏ với nội dung đơn giản.

II. Các bảng điều khiển

Khi tạo project với định dạng *Windows Application*, *Visual Studio* sẽ tự động tạo ra một form có tên là *Form1* và đây là form chính của ứng dụng. Màn hình giao diện sử dụng của *Visual Studio* lúc này có một số thay đổi và thay đổi này là cho giao diện khác với khi tạo project *Console Application* (trình bày ở chương 2). Giao diện làm việc của *Visual Studio* như sau:



Hình 55 – Giao diện làm việc của Visual Studio với Windows Application project

Visual Studio chia màn hình làm việc thành nhiều phần, trong đó quan trọng nhất được gọi là vùng lập trình nằm ở giữa. Trong phần này người lập trình có thể chỉnh sửa giao diện của chương trình, có thể lập trình và có thể cấu hình dự án. Vùng lập trình có thể nở rộng hoặc thu nhỏ tùy thuộc vào các bảng điều khiển phụ xung quanh. Các bản điều khiển này thông thường gồm có: Toolbox panel, Solution Explorer panel, Properties Panel.

II.1. Toolbox panel

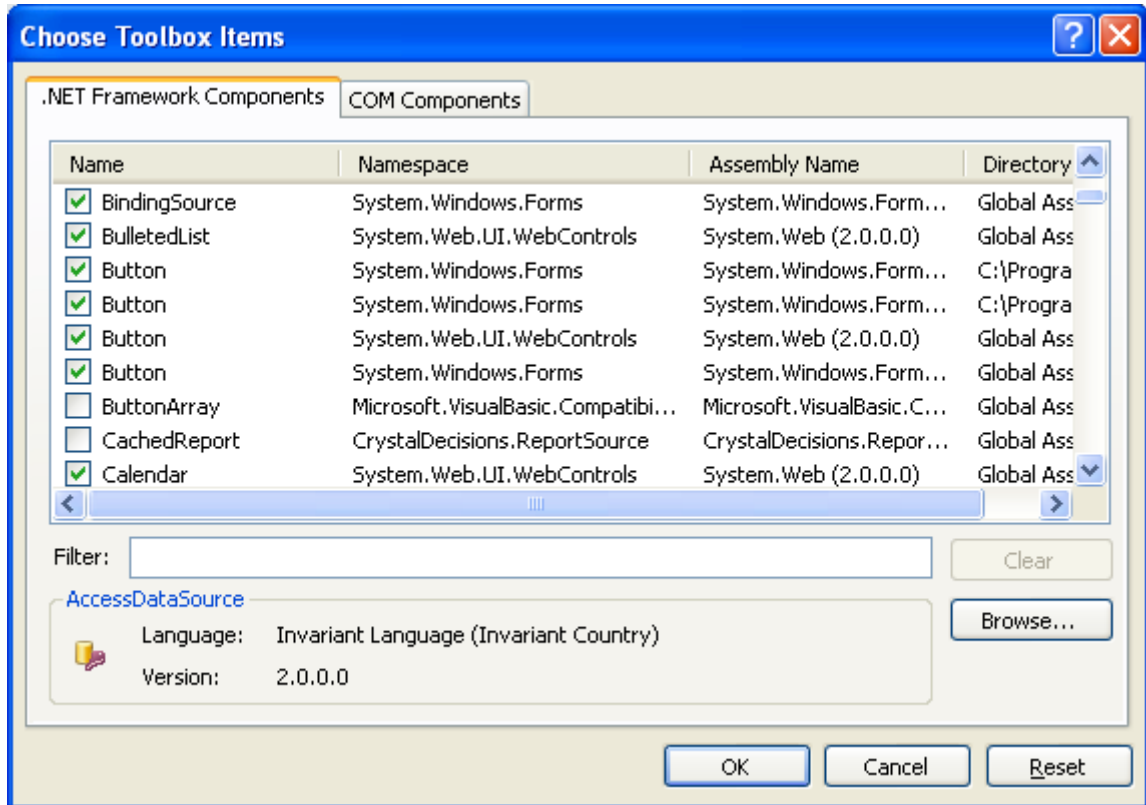
Toolbox panel là bảng điều khiển trong đó có chứa các đối tượng trình bày giao diện cho ứng dụng. Trong Toolbox panel, các đối tượng được chia thành nhiều nhóm gồm: Common Controls, Containers, Menus & Toolbars, Data, Component, Printing, Dialogs.

Toolbox panel thường nằm bên trái màn hình làm việc và Toolbox panel có chức năng tự ẩn (Auto Hide). Khi chức năng này được kích hoạt, Toolbox panel sẽ ẩn đi và khi người lập trình di chuyển con trỏ chuột (mouse cursor) đến vị trí “Toolbox” thì Toolbox panel sẽ được bung ra. Việc thiết lập chức năng tự ẩn cho Toolbox panel sẽ làm cho vùng lập trình rộng thêm và điều này có ý nghĩa quan trọng khi người lập trình muốn hiệu chỉnh giao diện của ứng dụng.

Để sử dụng các công cụ, người lập trình chỉ cần thực hiện động tác kéo – thả (drag & drop) công cụ đó vào form cần thiết. Visual Studio cung cấp một bộ sinh mã tự động (Auto Generating Code), bộ sinh mã này sẽ tự tạo các đoạn mã khởi tạo đối tượng cho chương trình,

các đoạn mã cấu hình cho từng đối tượng và các đoạn mã điều khiển việc trình bày đối tượng trên form mỗi khi người lập trình di chuyển hoặc chỉnh sửa kích thước các đối tượng.

Visual Studio cung cấp cho người lập trình hầu hết tất cả các công cụ phục vụ việc xây dựng giao diện cho ứng dụng. Tuy nhiên, do giới hạn của màn hình làm việc nên Toolbox panel chỉ chứa một số công cụ hữu ích cho các mục đích thông thường. Trong trường hợp người lập trình muốn sử dụng các công cụ chưa xuất hiện trên Toolbox panel, người lập trình có thể click chuột phải (right – click) vào Toolbox rồi chọn “Choose Item”, cửa sổ liệt kê danh sách các *.Net Framework Component* sẽ xuất hiện như hình sau.



Hình 56 – Danh sách các đối tượng công cụ

II.2. Solution Explorer panel

Solution Explorer panel là bảng điều khiển hiển thị cấu trúc chung của project. Các thành phần của project được hiển thị trong bảng này theo cấu trúc cây (giống như cấu trúc cây thư mục). Các thành phần của project chủ yếu là các file và các file này được nhóm thành các thư mục.

Thông thường, Solution Explorer panel thường ở bên phải của màn hình làm việc. Người lập trình có thể thay đổi chức năng Auto Hide giống như Toolbox panel cho phù hợp với mục đích sử dụng. Trong trường hợp Solution Explorer panel không hiển thị, người lập trình có thể kích hoạt bằng cách chọn View trên thanh menu, sau đó chọn Solution Explorer.

II.3. Properties panel

Properties panel là bảng điều khiển chứa các thuộc tính của đối tượng đang được kích hoạt. Trong Properties panel, các thuộc tính của đối tượng được nhóm thành từng nhóm hoặc được sắp xếp theo thứ tự chữ cái tùy vào lựa chọn của người lập trình. Nếu người lập trình kích hoạt nút “Categorized” thì tất cả các thuộc tính được nhóm thành từng nhóm. Nếu người lập trình chọn nút “Alphabetical” thì tất cả các thuộc tính được sắp xếp theo thứ tự chữ cái.

Properties panel còn chứa các sự kiện của đối tượng và các sự kiện này cũng được sắp xếp theo thứ tự chữ cái hoặc được gom theo nhóm giống như các thuộc tính. Khi người lập trình nhấn đúp chuột trái (double click) vào sự kiện nào, Visual Studio sẽ tự động phát sinh mã lệnh cho sự kiện đó và con trỏ soạn thảo sẽ được chỉ định ngay tại vị trí phương thức của sự kiện.

Trong trường hợp Properties panel không xuất hiện, người lập trình có thể kích hoạt bằng hai cách. Cách thứ nhất giống như việc kích hoạt Solution panel, người lập trình chỉ cần chọn View trên thanh menu sau đó chọn “Properties Window” thì Properties panel sẽ xuất hiện. Cách thứ hai, người lập trình chỉ cần kích chuột phải (right click) vào đối tượng cần thiết, popup menu sẽ xuất hiện và sau đó chọn “Properties”. Thông thường, Properties panel xuất hiện ở góc phía dưới bên phải của màn hình soạn thảo.

III. Form và Label

III.1. Form

III.1.1. Khái niệm Form

Form là khái niệm dùng để chỉ thành phần được dùng làm giao tiếp giữa người sử dụng với máy tính thông qua ứng dụng Windows. Form là thành phần chính của ứng dụng desktop, thông qua form, người sử dụng có thể làm việc với một giao diện thân thiện với các chức năng nhập liệu, trình bày dữ liệu, trình bày các bảng biểu...

Có hai loại form: Normal Form và MDI Form. MDI Form (Multiple Documents Interface Form) là dạng form trong đó cho phép chứa các form khác. Thông thường, trên MDI Form, người lập trình thường xây dựng menu để khi gọi từng chức năng, các form tương ứng sẽ xuất hiện. Normal Form là form bình thường, trên Normal Form chúng ta có thể trình bày các đối tượng đồ họa phục vụ việc giao tiếp giữa người với ứng dụng thông qua giao diện; Normal Form không thể chứa Form. Người lập trình có thể thay đổi một Form từ Normal sang MDI và ngược lại bằng cách thay đổi giá trị của thuộc tính IsMdiContainer, IsMdiContainer có giá trị *false* thì Form sẽ là Normal và IsMdiContainer có giá trị *true* khi form là MDI.

III.1.2. Các thuộc tính của Form

- Thuộc tính Name: Thuộc tính Name là thuộc tính nhận dạng duy nhất của đối tượng Form trong project. Mỗi Form phải có giá trị hợp lệ trong thuộc tính Name. Visual Studio không cho phép có nhiều hơn một Form có cùng tên trong một Project.

- Thuộc tính Text: Thuộc tính Text dùng để trình bày chuỗi ký tự trên thanh tiêu đề của Form. Người lập trình có thể khai báo thuộc tính này trong chế độ thiết lập hoặc gán giá trị cho thuộc tính này lúc thực thi.
- Thuộc tính ShowIcon: Thuộc tính ShowIcon chỉ định rằng Form có hiển thị Icon hay không. Icon này tùy thuộc vào tập tin ảnh mà người lập trình khai báo cho thuộc tính *Icon*, và sẽ xuất hiện ở góc phải trên bên trái của form khi chạy chương trình. Trong trường hợp không có nhu cầu trình bày Icon, bạn có thể khai báo thuộc tính ShowIcon bằng *false*.
- Thuộc tính Opacity: Thuộc tính Opacity cho phép người lập trình làm trong suốt bề mặt của form tùy thuộc vào giá trị phần trăm khai báo trong thuộc tính này.
- Thuộc tính Icon: Thuộc tính này cho phép người lập trình chỉ định tập tin hình ảnh để thể hiện biểu tượng của Form. Biểu tượng của form được hiển thị ở góc phía trên, bên trái khi sửa dụng ứng dụng.
- Thuộc tính WindowState: Khi nạp lên màn hình, form thường được định vị theo một trong ba trạng thái quy định trong thuộc tính WindowState: Normal, Minimized hoặc Maximized. Nếu WindowState là Normal thì form được hiển thị trên màn hình đúng bằng kích thước mà người lập trình đã thiết kế. Nếu WindowState là Minimized thì form sẽ được thu nhỏ lại thành biểu tượng và hiển thị ở TaskBar của Windows. Nếu WindowState là Maximized thì form sẽ phủ đầy màn hình hoặc sẽ phủ đầy MDI Form chứa form đó.
- Thuộc tính AcceptButton: Khi làm việc với form, đôi khi trên form có một nút mà khi bấm nút đó thì chương trình thực hiện một chức năng xác định. Nếu người lập trình có mong muốn chỉ cần bấm phím “Enter” mà chức năng của nút được thực hiện thì AcceptButton sẽ được gán giá trị là nút ấy.
- Thuộc tính CancelButton: Thuộc tính này có chức năng tương tự thuộc tính AcceptButton và khi người lập trình bấm phím “Esc” thì nút được khai báo cho thuộc tính này sẽ tự động được kích hoạt.
- Thuộc tính MdiParent: Thuộc tính này xác định Form MDI chứa Form hiện tại. Chi tiết về cách sử dụng thuộc tính này được trình bày trong phần tiếp theo.

III.1.3. Các sự kiện của Form

- Sự kiện Load: sự kiện này được tự động kích hoạt khi mở form. Thông thường trong sự kiện này, người lập trình thực hiện các lệnh khai báo hoặc các lệnh đầu tiên của chương trình.
- Sự kiện FormClosing: sự kiện này được tự động kích hoạt khi form đang đóng. Người lập trình có thể sử dụng sự kiện này để xác nhận với người sử dụng có thực sự muốn đóng form hay không.
- Sự kiện FormClosed: sự kiện này được tự động kích hoạt khi form đã đóng. Trong sự kiện này, Visual Studio cung cấp thuộc tính CloseReason cho phép người lập trình hiển thị lý do đóng form đối với người sử dụng.

III.2. Label

III.2.1. Khái niệm Label

Label là khái niệm dùng để chỉ đối tượng được sử dụng để trình bày một dòng văn bản trên form. Label cho phép người lập trình trình bày tiêu đề và chú giải cho các đối tượng nhập liệu khác trên form.

III.2.2. Các thuộc tính của Label

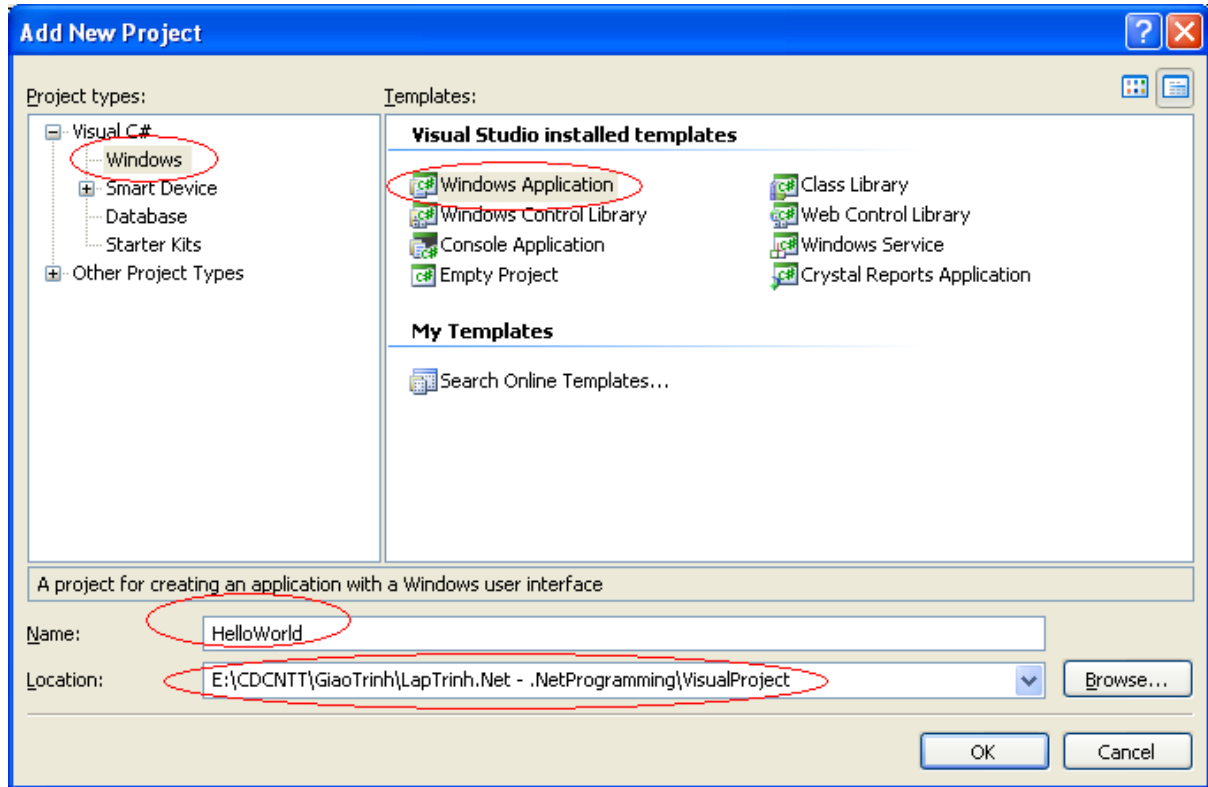
- Thuộc tính `BorderStyle`: thuộc tính `BorderStyle` được sử dụng để thiết lập kiểu đường viền của đối tượng Label.
- Thuộc tính `Font`: thuộc tính này cho phép người lập trình thay đổi kích thước và kiểu chữ trình bày trên Label.
- Thuộc tính `TextAlign`: thuộc tính này phục vụ việc canh lề của đoạn văn bản trong Label.

III.3. Ứng dụng Form và Label

Ví dụ sau minh họa việc sử dụng Form và Label để xây dựng một ứng dụng mà trên đó hiển thị dòng chữ “Hello World”. Ứng dụng được trình bày theo các bước sau:

III.3.1. Tạo mới project

Người lập trình tạo mới project bằng cách chọn File trên thanh menu, sau đó chọn New rồi chọn Project. Trong cửa sổ “New Project”, người lập trình chọn *Windows* trong mục “Project types”, chọn *Windows Application* trong mục “Templates”, đặt tên cho project trong mục “Name” và cuối cùng là xác định đường dẫn cho project trong mục “Location”.

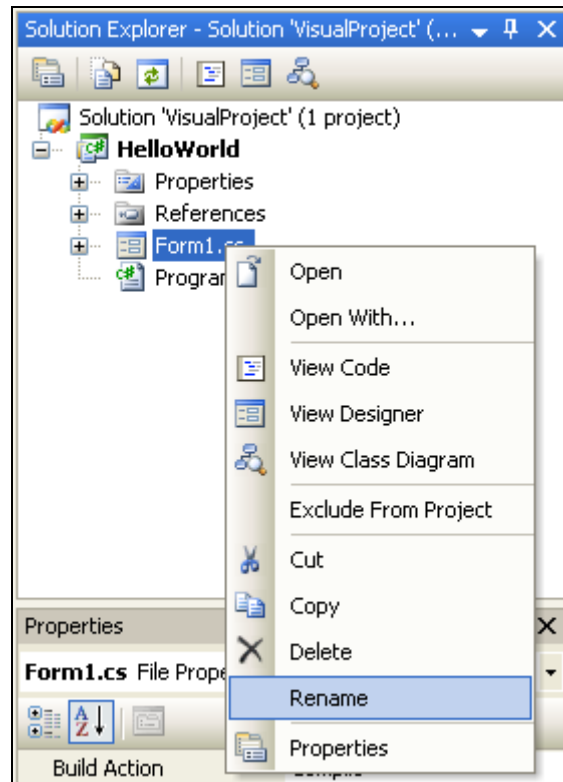


Hình 57 – Tạo mới Windows Application project

Khi khởi tạo, Visual Studio sẽ tự động tạo ra cấu trúc cho project trong đó có một Form chính có tên là Form1 (Xem hình 55). Form này sẽ là form chính khi chạy chương trình.

III.3.2. Đổi tên Form chính

Người lập trình có thể thay đổi tên của Form chính bằng cách kích chuột phải (right click) vào Form trong Solution Explorer panel sau đó chọn Rename.

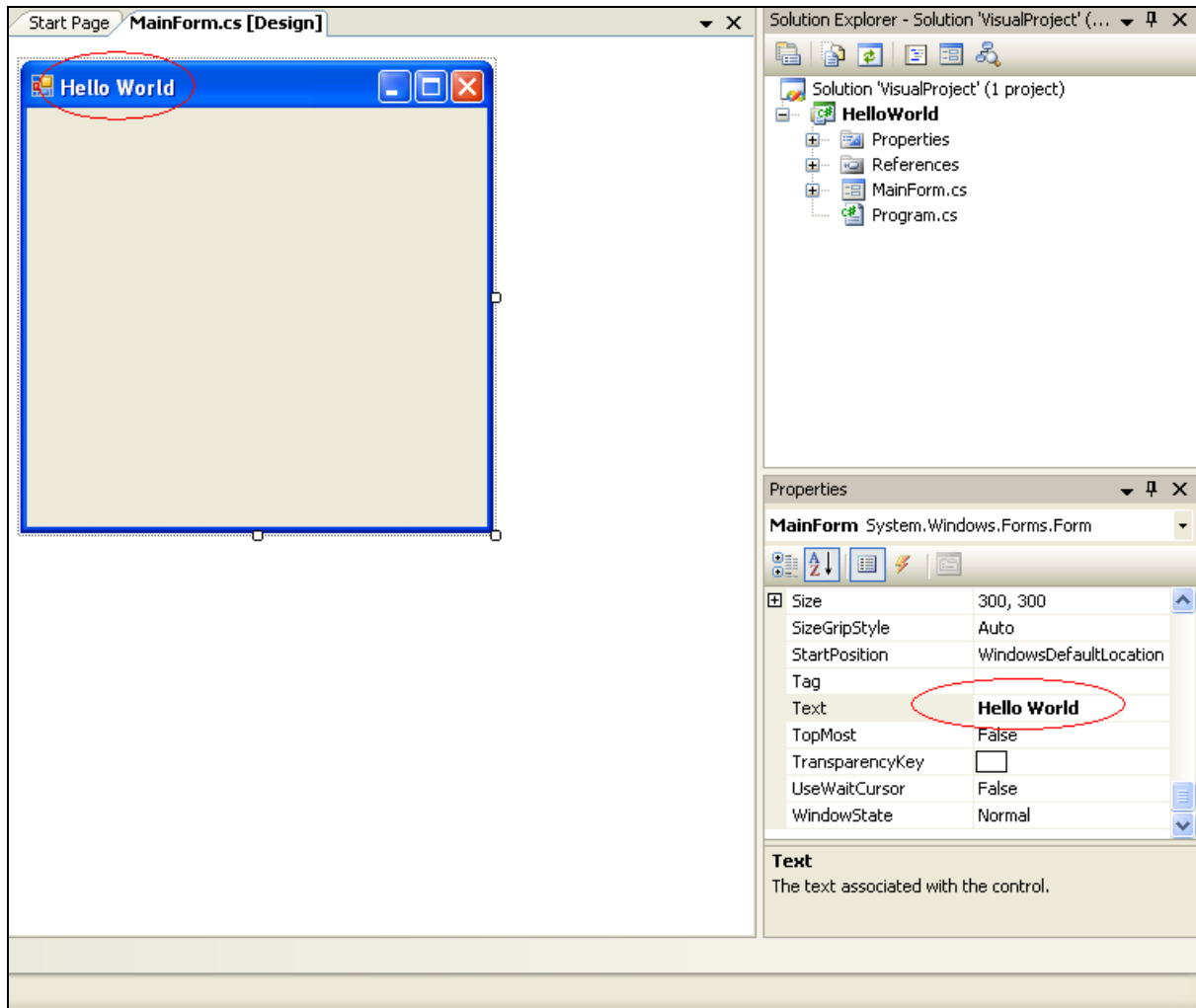


Hình 58 – Thay đổi tên Form

Sau khi đổi tên, Visual Studio sẽ yêu cầu người lập trình xác nhận để sau đó tên Form mới sẽ được áp dụng trên toàn bộ project.

III.3.3. Đặt tiêu đề cho Form

Khi người lập trình kích chuột vào Form thì trong bảng Properties panel sẽ xuất hiện các thuộc tính của Form. Trong bảng này, ta chỉnh sửa thuộc tính Text để Form có tiêu đề như ý muốn. Với ví dụ này, thuộc tính Text được gán giá trị “Hello World”.

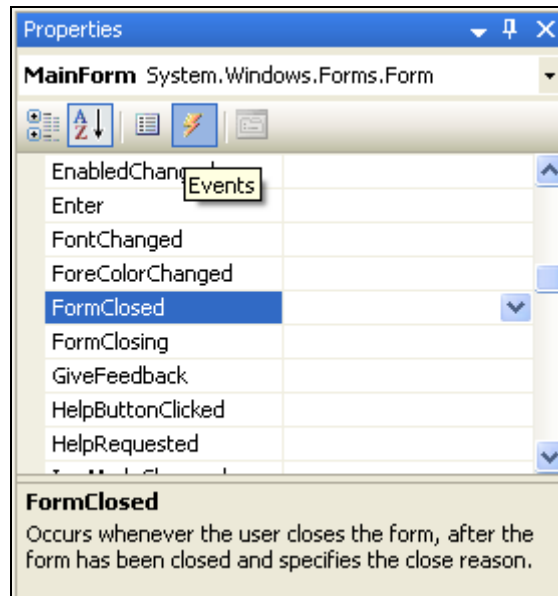


Hình 59 – Thay đổi thuộc tính Text của Form

III.3.4. Cài đặt sự kiện *FormClosed*

Bước này không bắt buộc phải được cài đặt cho form, tuy nhiên để minh họa cho quá trình .Net Framework, quá trình xử lý sự kiện *FormClosed* được cài đặt cho Form.

Trong cửa sổ Properties panel, chọn nút Events để chuyển sang bảng danh sách các sự kiện. Sau đó chọn sự kiện *FormClosed* và kích đúp chuột vào TextBox bên cạnh, Visual Studio sẽ chuyển con trỏ soạn thảo đến phương thức cấu hình cho sự kiện và người lập trình sẽ soạn thảo đoạn mã ở đây.



Hình 60 – Khai báo sự kiện FormClosed

Trong vùng lập trình, người lập trình có thể sử dụng viết các đoạn mã và các đoạn mã này sẽ được thực thi khi Form được đóng lại. Trong đoạn mã này, người lập trình có thể sử dụng lệnh CloseReason thực hiện việc truy xuất lý do đóng Form.

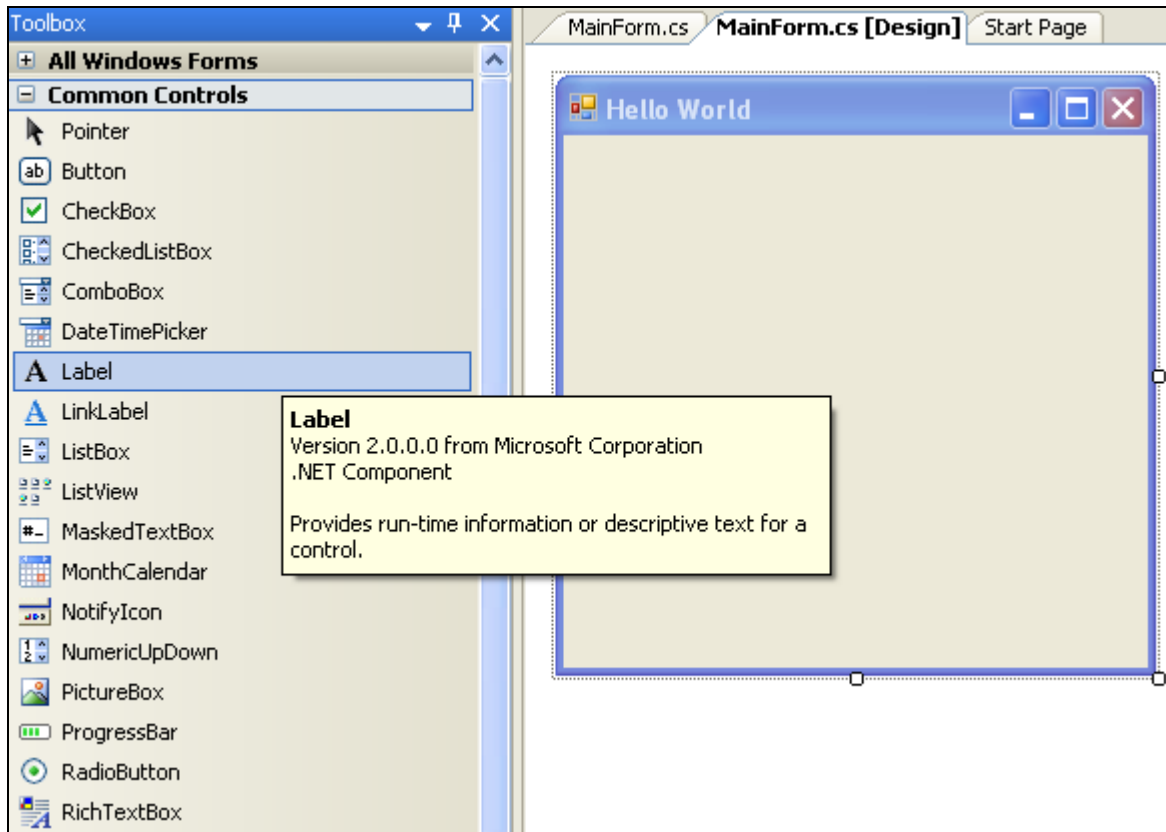
```
public partial class MainForm : Form
{
    public MainForm()
    {
        InitializeComponent();
    }

    private void MainForm_FormClosed(object sender, FormClosedEventArgs e)
    {
        MessageBox.Show(e.CloseReason.ToString());
    }
}
```

Hình 61 – Mã lệnh cho sự kiện FormClosed

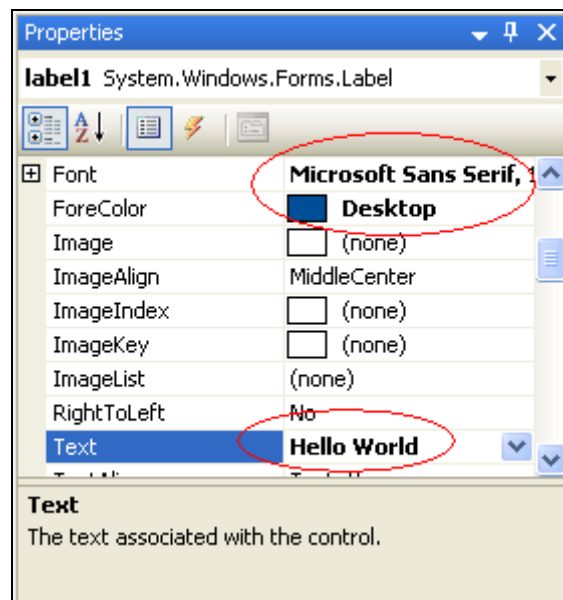
III.3.5. Thêm Label vào Form

Trong bảng Toolbox panel, thực hiện động tác kéo thả đối tượng Label vào Form. Trong trường hợp Toolbox panel chưa hiển thị trên màn hình, người lập trình có thể kích hoạt để xuất hiện Toolbox panel bằng cách chọn vào View trên thanh menu rồi chọn Toolbox.



Hình 62 – Thực hiện việc thêm Label vào Form

Sau khi Label được kéo và thả vào Form, người lập trình có khai báo các giá trị cho các thuộc tính của Label trong Properties panel.



Hình 63 – Thay đổi các thuộc tính của Label

Trong ví dụ ở hình trên, Label được thay đổi các thuộc tính như Font: Microsoft Sans Serif, ForeColor: Desktop, Text: Hello World.

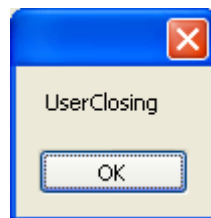
III.3.6. Biên dịch và chạy ứng dụng

Thực hiện việc biên dịch và chạy ứng dụng giống như Console Application. Sau khi chạy ứng dụng ta được kết quả như sau.



Hình 64 – Kết quả chạy ứng dụng HelloWorld

Khi đóng ứng dụng, sự kiện FormClosed được gọi và chương trình sẽ hiển thị thông báo lý do đóng ứng dụng như sau.



Hình 65 – Kết quả khi đóng ứng dụng

IV. TextBox và Button

IV.1. TextBox

IV.1.1. Khái niệm TextBox

TextBox là khái niệm dùng để chỉ đối tượng trình bày trên Form, đối tượng này cho phép người sử dụng nhập dữ liệu đầu vào từ bàn phím.

IV.1.2. Các thuộc tính của TextBox

- BorderStyle: Thuộc tính này quy định kiểu đường viền của TextBox.

- Enable: Thuộc tính này có giá trị *true* thì người sử dụng có thể thao tác với TextBox, ngược lại thì người sử dụng không thể thao tác với TextBox. Trong quá trình lập trình, người lập trình có thể thiết lập thuộc tính này trong trường hợp muốn phân quyền, với những người dùng có tài khoản hợp lệ thì Enable có giá trị *true* để người dùng có thể thao tác và ngược lại, Enable có giá trị *false* đối với những tài khoản không hợp lệ.
- MaxLength: Thuộc tính này quy định số ký tự tối đa nhập vào TextBox.
- Multiline: Thuộc tính này có giá trị *true* thì TextBox sẽ có nhiều dòng và ngược lại.
- ScrollBars: Thuộc tính này được sử dụng khi thuộc tính Multiline có giá trị *true*. Nếu ScrollBars có giá trị *true* thì TextBox sẽ xuất hiện thanh cuộn và ngược lại.
- ReadOnly: Thuộc tính này có giá trị *true* thì người sử dụng không được phép nhập dữ liệu vào TextBox. Thuộc tính này cũng rất hữu ích trong trường hợp phân quyền.
- PasswordChar: Thuộc tính này được sử dụng cho những TextBox dùng để nhập mật khẩu. Với một ký tự được gán cho thuộc tính này, khi người sử dụng nhập vào TextBox, ký tự trong TextBox sẽ được tự động thay bởi giá trị của thuộc tính này.
- TextAlign: Thuộc tính này quy định việc canh lề cho đoạn văn bản trong TextBox.

IV.1.3. Các sự kiện của TextBox

- MouseClick: Sự kiện này xảy ra khi người sử dụng kích chuột vào TextBox.
- MouseDoubleClick: Sự kiện này xảy ra khi người sử dụng kích đúp chuột (double click) vào TextBox.
- TextChanged: Sự kiện này xảy ra khi người sử dụng thay đổi nội dung trong TextBox.

IV.2. Button

IV.2.1. Khái niệm Button

Button là khái niệm được dùng để chỉ đối tượng được trình bày trên Form, đối tượng này cho phép người sử dụng kích chuột vào để thực hiện một tác vụ nào đó.

Người lập trình có thể gán thuộc tính AcceptButton hoặc CancelButton của Form bằng các đối tượng Button để khi bấm phím *Enter* hoặc phím *Esc* thì sự kiện của Button tương ứng được triệu gọi.

IV.2.2. Các thuộc tính của Button

- FlatStyle: Thuộc tính này quy định kiểu đường viền của Button.
- Image: Thuộc tính này quy định hình ảnh hiển thị trên Button. Đường dẫn của hình ảnh được khai báo trong phần *Resource*.
- Text: Thuộc tính này quy định chuỗi ký tự trên Button.
- Enable: Thuộc tính này quy định Button có bị vô hiệu hóa hay không. Cũng giống như TextBox và một số đối tượng khác, thuộc tính này thường được sử dụng trong trường hợp phân quyền.

IV.2.3. Các sự kiện của Button

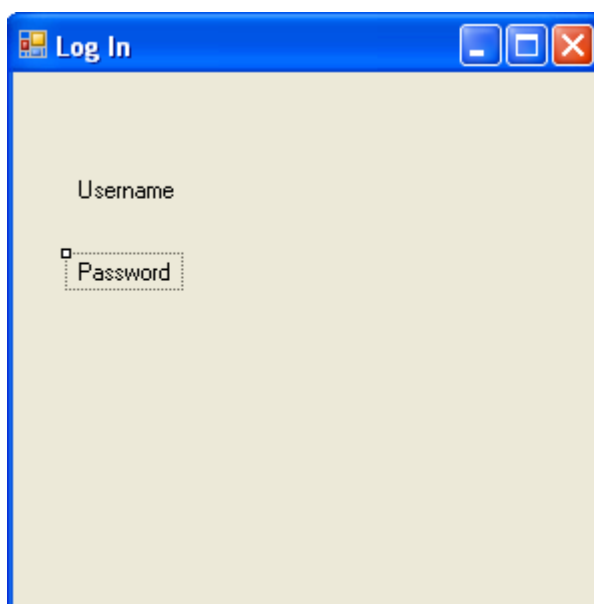
- `MouseClick`: Sự kiện này xảy ra khi người sử dụng kích chuột vào Button.
- `EnabledChanged`: Sự kiện này xảy ra khi thuộc tính `Enable` của Button thay đổi.

IV.3. Ứng dụng TextBox và Button

Ví dụ sau trình bày việc ứng dụng Form, Label, TextBox và Button trong việc xây dựng một Form đăng nhập. Form này gồm có hai TextBox để người sử dụng nhập tên tài khoản, mật khẩu và nút “Log In” để thực hiện hành động đăng nhập vào ứng dụng. Trong ví dụ này, hành động đăng nhập vào ứng dụng chỉ đơn thuần là hiển thị tài khoản và mật khẩu ra màn hình. Ứng dụng được xây dựng theo từng bước như sau:

IV.3.1. Tạo project, tạo Form và các Label

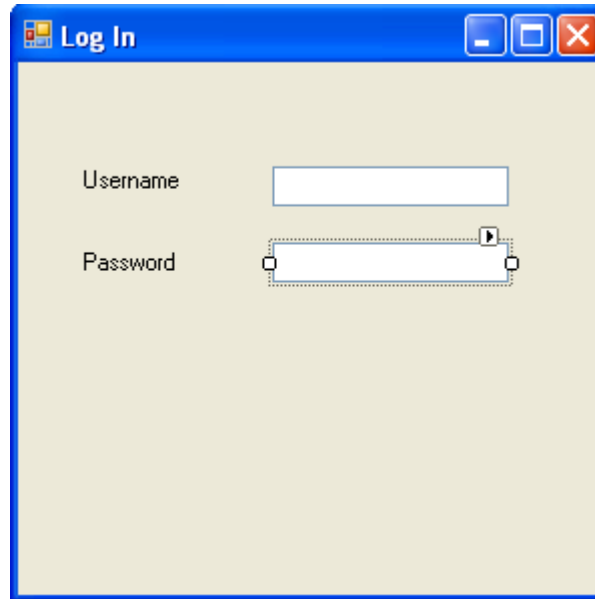
Thực hiện việc tạo project Windows Application, sau đó kéo thả hai Label vào Form (Xem ví dụ ở phần III.3 của chương này). Trong ví dụ này, hai Label lần lượt có thuộc tính Text là “Username” và “Password”.



Hình 66 – Tạo Form và Label cho ứng dụng

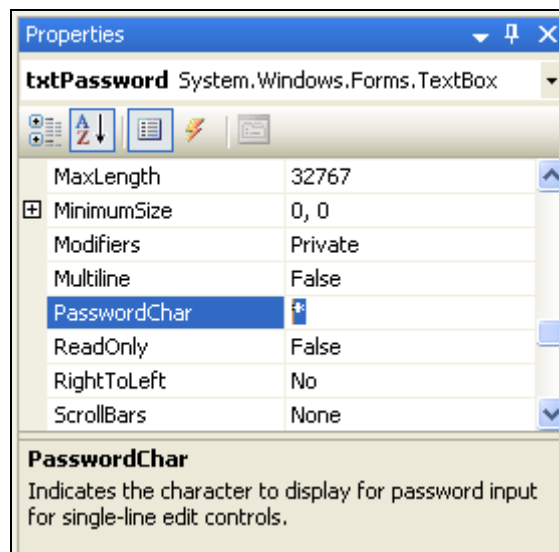
IV.3.2. Tạo các TextBox

Thực hiện việc kéo thả hai TextBox vào Form. Các TextBox được đặt tên lần lượt là “txtUsername”, “txtPassword” và được đặt ở vị trí tương ứng với hai Label đã được tạo ở bước trước đó.



Hình 67 – Tạo các TextBox tương ứng với các Label

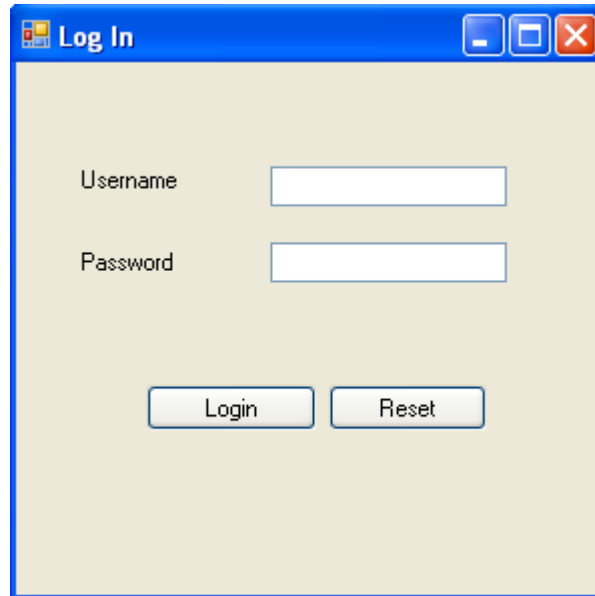
Đối với TextBox txtPassword, thực hiện việc thay đổi thuộc tính PasswordChar bằng cách gán ký tự “*”. Với ký tự này, khi người sử dụng nhập vào txtPassword thì trên đối tượng đó chỉ hiển thị các ký tự *, điều này là cần thiết khi nhập mật khẩu.



Hình 68 – Thay đổi thuộc tính PasswordChar

IV.3.3. Thêm các Button

Kéo thả thêm hai Button vào Form và đặt tên lần lượt là btnLogIn và btnReset. Ứng với mỗi Button, thay đổi thuộc tính Text tương ứng để dòng văn bản hiển thị trên mỗi Button lần lượt là “Login” và “Reset”.



Hình 69 – Thêm các Button vào Form

IV.3.4. Cài đặt sự kiện cho từng Button

Sự kiện cơ bản của Button là Click, sự kiện này được triệu gọi khi người sử dụng kích chuột vào Button.

Để lập trình sự kiện cho nút Reset, thực hiện kích đúp chuột vào nút Reset. Visual Studio sẽ chuyển con trỏ soạn thảo đến phương thức của sự kiện Click. Trong phương thức này, lập trình thực hiện việc gán giá trị của các TextBox thành chuỗi rỗng.

```
private void btnReset_Click(object sender, EventArgs e)
{
    txtUsername.Text = "";
    txtPassword.Text = "";
}
```

Hình 70 – Xử lý sự kiện cho nút Reset

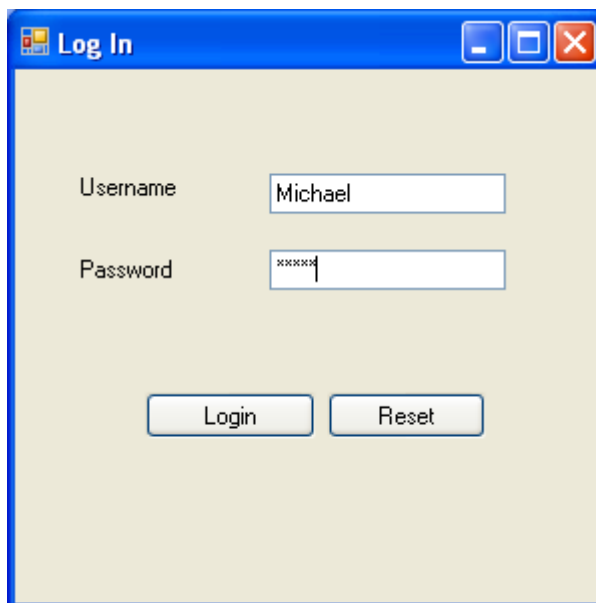
Để lập trình sự kiện cho nút Login, thực hiện kích đúp chuột vào nút Login và lập trình thực hiện việc hiển thị thông báo ra màn hình.

```
private void btnLogin_Click(object sender, EventArgs e)
{
    if (txtUsername.Text == "")
    {
        MessageBox.Show("You must type username");
    }
    else
    {
        MessageBox.Show("You have logged with username "+txtUsername.Text);
    }
}
```

Hình 71 – Xử lý sự kiện nút Login

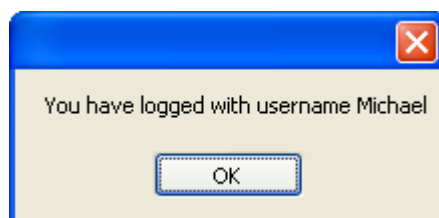
IV.3.5. Biên dịch và chạy chương trình

Sau khi biên dịch và chạy thì chương trình sẽ trình bày như sau.



Hình 72 – Kết quả chạy chương trình

Nếu người sử dụng bấm nút Reset thì các TextBox sẽ tự động được xóa. Nếu người sử dụng bấm nút Login thì chương trình sẽ hiển thị thông báo rằng người sử dụng đã thực hiện việc Login.



Hình 73 – Kết quả hiển thị khi người sử dụng bấm nút Login

V. ComboBox, CheckBox, RadioButton

V.1. ComboBox

V.1.1. Khái niệm ComboBox

ComboBox là khái niệm dùng để chỉ đối tượng hiển thị trên Form, đối tượng này cho phép người dùng lựa chọn một hoặc nhiều phần tử trong danh sách “đổ xuống” (dropdown list).

V.1.2. Các thuộc tính của ComboBox

- FlatStyle: Thuộc tính này quy định kiểu đường viền của ComboBox.

- DataSource: Tập dữ liệu đưa vào ComboBox. DataSource có thể được gán gằng một mảng của các chuỗi hoặc một ArrayList.
- DisplayMember: Tên của trường tương ứng trong danh sách được hiển thị trên ComboBox.
- Items: Thuộc tính này quy định tập các phần tử của ComboBox. Người lập trình có thể thêm phần tử vào ComboBox bằng phương thức Add hoặc AddRange.
- MaxDropDownItems: Thuộc tính này quy định số phần tử lớn nhất có thể liệt kê trong ComboBox, mặc định là 8 phần tử.
- MaxLength: Thuộc tính này quy định độ dài lớn nhất của chuỗi ký tự trong ComboBox.
- ValueMember: Thuộc tính này trả về giá trị ứng với khóa nếu phần tử có khóa và giá trị.
- Text: Thuộc tính này trả về chuỗi ký tự của nhãn trên ComboBox ứng với phần tử được chọn.
- SelectedText: Thuộc tính này có chức năng tương tự thuộc tính Text.
- SelectedItem: Thuộc tính này gán hoặc lấy đối tượng ứng với phần tử đang được chọn.
- SelectedValue: Lấy hoặc gán giá trị ứng với phần tử kiểu object đang chọn.
- SelectedIndex: Gán hoặc lấy giá trị chỉ mục ứng với phần tử đang chọn.

V.1.3. Các phương thức của ComboBox

- Add: Phương thức này thực hiện thêm một đối tượng vào ComboBox.
- AddRange: Phương thức này thực hiện thêm tập hợp đối tượng vào ComboBox.

V.1.4. Các sự kiện của ComboBox

- MouseClick: Sự kiện này xảy ra khi người sử dụng kích chuột vào ComboBox.
- SelectedIndexChanged: Sự kiện này xảy ra khi chỉ mục của phần tử được thay đổi.
- SelectedValueChanged: Sự kiện này xảy ra khi giá trị của phần tử được thay đổi.
- SelectionchangeCommitted: Sự kiện này xảy ra khi người sử dụng kết thúc việc chọn phần tử.

V.2. CheckBox

V.2.1. Khái niệm CheckBox

CheckBox là khái niệm dùng để chỉ đối tượng trình bày trên Form, đối tượng này cho phép người sử dụng lựa chọn hoặc không lựa chọn bằng cách đánh dấu vào đối tượng. CheckBox cho phép hiển thị cả hình ảnh và chuỗi văn bản trong trường hợp cần thiết.

V.2.2. Các thuộc tính của CheckBox

- FlatStyle: Thuộc tính này quy định đường viền cho CheckBox.
- Appearance: Thuộc tính này quy định hình dạng của CheckBox.
- Checked: Thuộc tính này có giá trị *true* nếu CheckBox được chọn và ngược lại.
- Text: Thuộc tính này trình bày chuỗi văn bản trên CheckBox.
- TextAlign: Thuộc tính này quy định việc canh lề cho dòng văn bản trên CheckBox.

V.2.3. Các sự kiện của CheckBox

- MouseClick: Sự kiện này xảy ra khi người sử dụng kích chuột vào CheckBox.
- CheckedChanged: Sự kiện này xảy ra khi người sử dụng thực hiện việc chọn hoặc bỏ chọn trên CheckBox.
- CheckedStateChanged: Sự kiện này xảy ra khi trạng thái Checked của CheckBox thay đổi. Trong một vài ngữ cảnh thì sự kiện này giống với sự kiện CheckedChanged.

V.3. RadioButton

V.3.1. Khái niệm RadioButton

RadioButton là khái niệm dùng để chỉ đối tượng được trình bày trên Form, đối tượng này cho phép người sử dụng lựa chọn một trong số các RadioButton khác trong danh sách tùy chọn.

V.3.2. Các thuộc tính của RadioButton

- FlatStyle: Thuộc tính này quy định kiểu đường viền của RadioButton.
- Appearance: Thuộc tính này quy định hình dạng của RadioButton.
- Checked: Thuộc tính này quy định trạng thái của RadioButton là được chọn hay chưa được chọn. Nếu Checked có giá trị *true* nghĩa là RadioButton đó đã được chọn.
- Text: Thuộc tính này trình bày chuỗi văn bản hiển thị ra màn hình.
- TextAlign: Thuộc tính này giúp cho việc canh lề của dòng văn bản trong RadioButton.

V.3.3. Các sự kiện của RadioButton

- MouseClick: Sự kiện này xảy ra khi người sử dụng kích chuột vào CheckBox.
- CheckedChanged: Sự kiện này xảy ra khi người sử dụng thực hiện việc chọn hoặc bỏ chọn trên CheckBox.

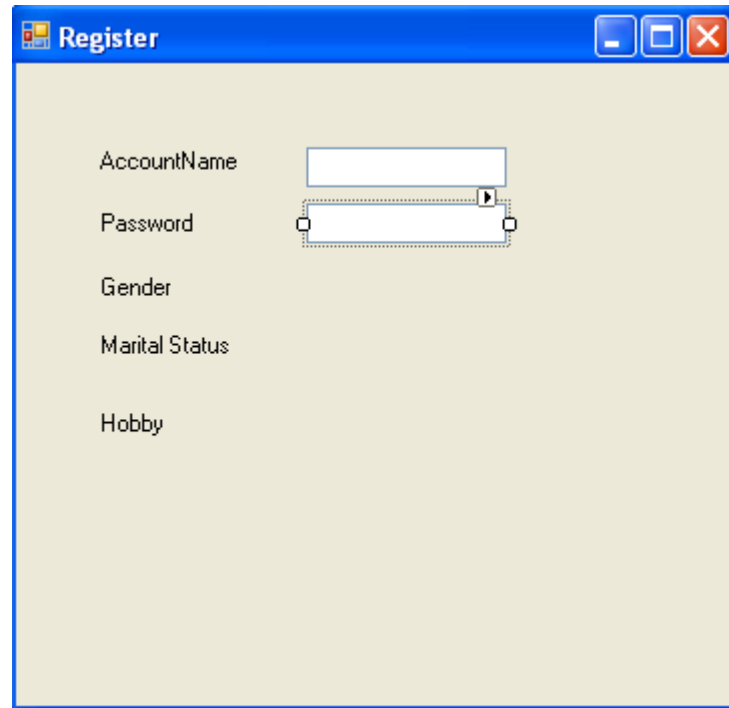
V.4. Ứng dụng ComboBox, CheckBox, RadioButton

Ví dụ sau đây trình bày việc ứng dụng Form, Label, TextBox, Button, ComboBox, CheckBox và RadioButton trong việc xây dựng một Form đăng ký. Form này yêu cầu người sử dụng nhập thông tin cá nhân và sau đó bấm nút đăng ký, chương trình lúc này sẽ hiển thị

thông báo là người dùng đã thực hiện việc đăng ký. Ứng dụng được xây dựng theo từng bước như sau:

V.4.1. Tạo project, tạo Form, tạo các Label và TextBox

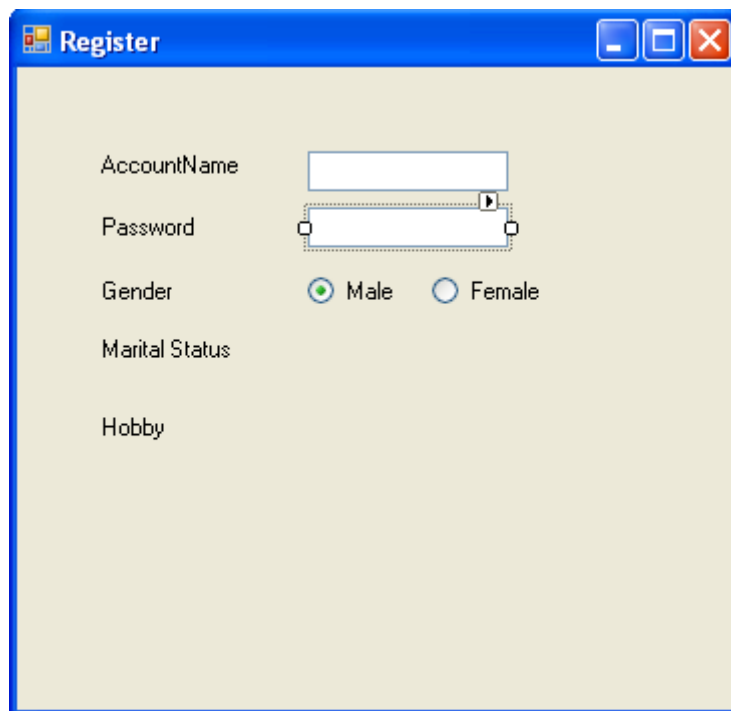
Tạo các Form, tạo Label và TextBox dựa theo hướng dẫn của ví dụ ở phân III.3 và IV.3 của chương này. Các Label lần lượt có tên là AccountName, Password, Gender, Marital Status và Hobby. Các TextBox cũng lần lượt có tên là txtAccountName và txtPassword, trong đó thuộc tính PasswordChar của txtPassword được thiết lập với ký tự “*”.



Hình 74 – Tạo Form, Label và TextBox liên quan

V.4.2. Tạo các RadioButton

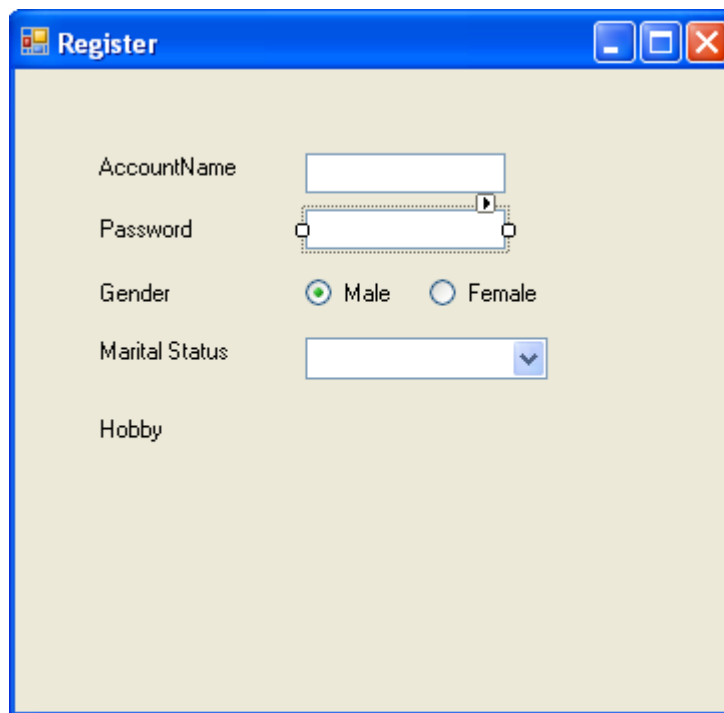
Ứng với Label Gender, thực hiện việc kéo thả hai RadioButon từ Toolbox panel, hai đối tượng này lần lượt được đặt tên là rdiMale và rdiFemale tương ứng với giới tính nam và giới tính nữ để người sử dụng lựa chọn. Ứng với từng RadioButton, thay đổi thuộc tính Text để các đối tượng này hiển thị các dòng văn bản lần lượt là “Male” và “Female”. Chú ý, một trong hai đối tượng này được gán thuộc tính Checked là *true*.



Hình 75 – Tạo các RadioButton

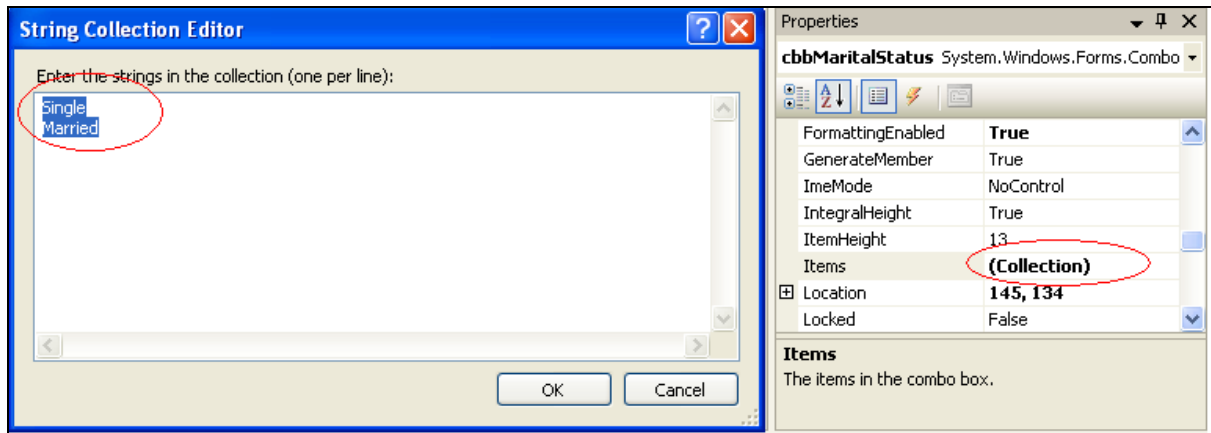
V.4.3. Tạo ComboBox

Ứng với Label Marital Status, tạo thêm ComboBox và đặt tên là cbbMaritalStatus.



Hình 76 – Tạo thêm ComboBox

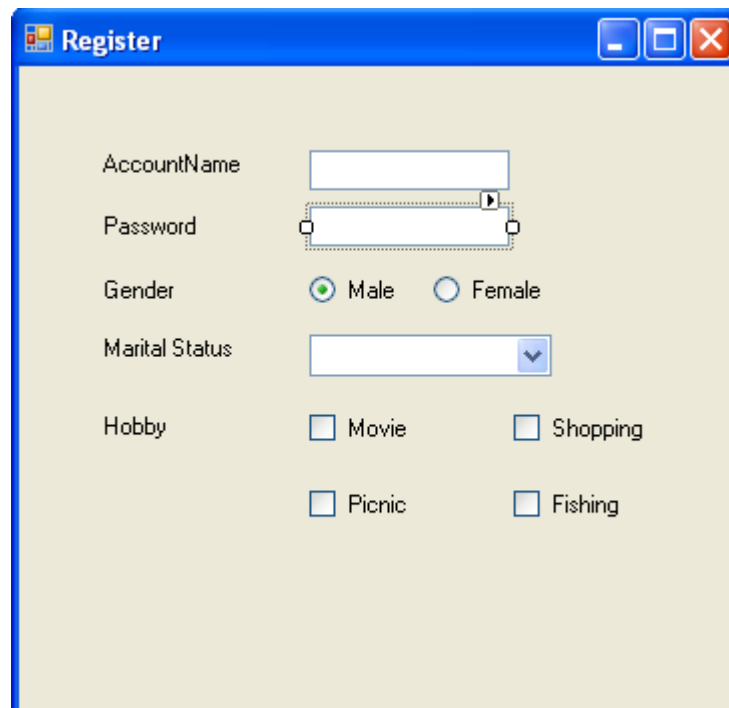
Đối với ComboBox, thực hiện việc khai báo thêm Item có tên là Single và Married. Trong Properties panel, bấm vào thuộc tính Item. Visual Studio sẽ hiển thị một Form để người dùng nhập các Item vào ComboBox, mỗi Item ứng với một dòng.



Hình 77 – Thêm Item cho ComboBox

V.4.4. Tạo các CheckBox

Ứng với Label Hobby, tạo các CheckBox bằng cách kéo thả từ Toolbox Panel vào Form. Các CheckBox lần lượt là chkMovie, chkShopping, chkPicnic, chkFishing tương ứng với các sở thích như xem film, mua sắm, đi du lịch, câu cá để người dùng lựa chọn.



Hình 78 – Tạo thêm các CheckBox

V.4.5. Tạo Button

Thực hiện tạo Button có tên là btnRegister. Button này có sự kiện thực hiện lệnh thông báo là người sử dụng đã thực hiện việc đăng nhập. Thực hiện việc viết mã lệnh để xử lý sự kiện khi người sử dụng kích vào btnRegister như sau.

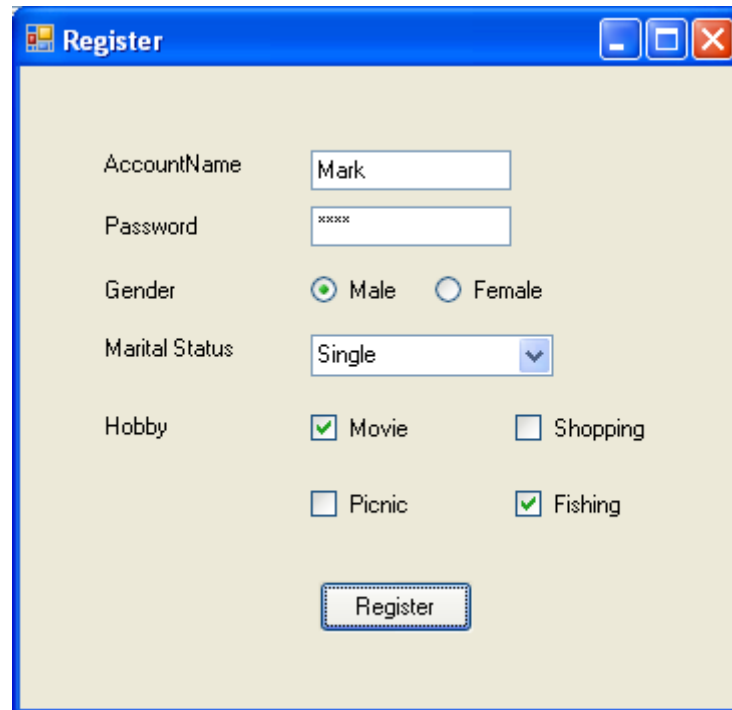
```
private void btnRegister_Click(object sender, EventArgs e)
{
    string message = "You have registered with ";
    message = message + "AccountName: " + txtAccountName.Text + "; ";
    message = message + "Password: " + txtPassword.Text + "; ";
    message = message + "Gender: ";
    if (rdiMale.Checked) message = message + "Male; ";
    else message = message + "Female; ";
    message = message + "Marital Status: "
        + cbbMaritalStatus.SelectedItem.ToString() + "; ";
    message = message + "Hobby: ";
    foreach (Control chk in this.Controls)
    {
        if (chk is CheckBox)
        {
            if (((CheckBox)chk).Checked)
                message = message + chk.Text + " ";
        }
    }
    MessageBox.Show(message);
}
```

Hình 79 – Xử lý sự kiện khi người sử dụng kích chuột vào nút btnRegister

Đoạn mã của sự kiện của Button thực hiện việc lấy các thông tin từ Form để nối vào chuỗi message rồi hiển thị ra màn hình. Để lấy thông tin từ ComboBox, ta thực hiện truy xuất thuộc tính SelectedItem. Để lấy thông tin từ CheckBox, ta thực hiện việc truy xuất tất cả các đối tượng của Form, ứng với đối tượng là CheckBox, ta kiểm tra xem thuộc tính Checked có giá trị *true* hay *false*, nếu có giá trị *true* nghĩa là người sử dụng đã chọn và ngược lại.

V.4.6. Biên dịch và chạy chương trình

Thực hiện việc biên dịch và chạy ứng dụng (Xem các ví dụ ở mục III.3, IV.3 ở chương này) ta được kết quả như sau.



Hình 80 – Kết quả chạy ứng dụng

Khi người sử dụng điền đầy đủ thông tin và bấm nút Register thì chương trình sẽ hiển thị thông báo như sau.



Hình 81 – Thông báo khi người sử dụng nhấn nút Register

VI. MDI Form và MenuStrip

VI.1. MDI Form

VI.1.1. *Khái niệm MDI Form*

MDI Form (Multiple Document Interface Form) là khái niệm dùng để chỉ một loại Form mà trong đó có chứa các Form khác. Các Form chứa trong MDI Form được gọi là Normal Form. Với phiên bản Visual Studio 2005 trở về trước thì MDI Form không thể chứa được MDI Form khác.

Người lập trình có thể chuyển một Normal Form thành một MDI Form bằng cách thay đổi thuộc tính *IsMdiContainer* từ *False* thành *True*.

VI.1.2. Các thuộc tính của MDI Form

- Thuộc tính `IsMdiContainer`: Thuộc tính này có giá trị *false* nếu form là Normal và có giá trị *true* nếu form là MDI. Người lập trình có thể thiết lập giá trị này trong khi thiết kế hoặc thiết lập trong lúc chạy chương trình.
- Thuộc tính `MdiChildren`: Thuộc tính này trả về danh sách các Form con (Normal Form) đang chứa trong MDI Form hiện tại.

VI.2. MenuStrip

VI.2.1. Khái niệm MenuStrip

MenuStrip là khái niệm dùng để chỉ đối tượng được trình bày trên Form, đối tượng này tổ chức các phần tử (`ToolStripMenuItem`) theo cấu trúc cây và trình bày trên màn hình thành thanh menu để người sử dụng có thể lựa chọn.

VI.2.2. Các thuộc tính của MenuStrip

- `AllowItemReorder`: Thuộc tính này cho phép người sử dụng tùy chọn cá phần tử bằng bàn phím khi bấm phím Alt.
- `TextDirection`: Thuộc tính này cho phép người sử dụng thay đổi hình thức trình bày trên menu là ngang (`Horizontal`) hoặc dọc (`Vertical90` hoặc `Vertical270`).
- `Items`: Thuộc tính này trả về danh sách các phần tử của MenuStrip.

VI.3. ToolStripMenuItem

VI.3.1. Khái niệm ToolStripMenuItem

`ToolStripMenuItem` là khái niệm dùng để chỉ một phần tử của MenuStrip, phần tử này có chức năng tương tự như Button, nghĩa là cho phép người sử dụng kích chuột hoặc dùng phím để lựa chọn để từ đó ứng dụng sẽ thực hiện những chức năng tương ứng.

VI.3.2. Các thuộc tính của ToolStripMenuItem

- `Enable`: Thuộc tính này chỉ định đối tượng có bị vô hiệu hóa hay không. Thuộc tính `Enable` thường được sử dụng khi người lập trình muốn phân quyền người sử dụng.
- `Image`: Hình ảnh xuất hiện bên cạnh chuỗi văn bản của đối tượng.
- `ImageScaling`: Thuộc tính này thường được dùng với thuộc tính `Image`, thuộc tính này quy định hình ảnh đi kèm hiển thị với kích thước thật hay kích thước thu nhỏ.
- `ShortcutKeyDisplayString`: Thuộc tính này trình bày dòng ký tự ứng với phím tắt.
- `ShortcutKeys`: Thuộc tính này quy định phím tắt để kích hoạt `ToolStripMenuItem`.
- `ShowShortcutKeys`: Thuộc tính này cho phép hiển thị tổ hợp phím tắt.
- `Text`: Thuộc tính này trình bày dòng văn bản.

- TooltipText: Thuộc tính này quy định dòng văn bản sẽ xuất hiện khi người lập trình đưa chuột vào ToolStripMenuItem.

VI.3.3. Các sự kiện của ToolStripMenuItem

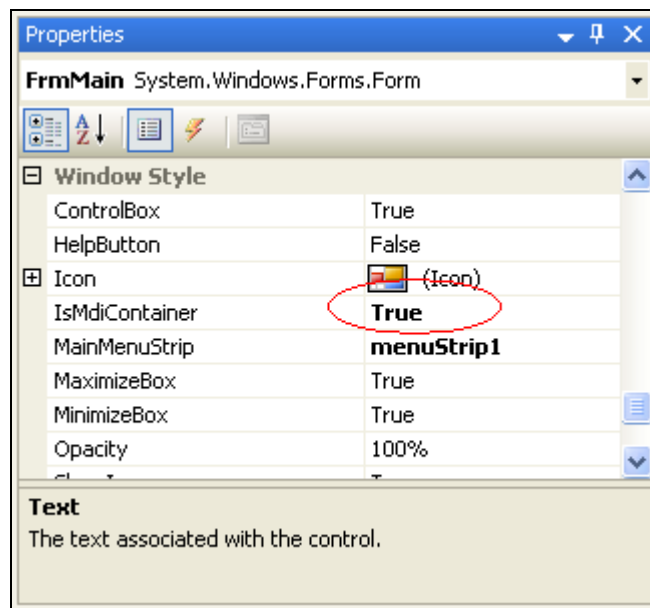
- Click: Sự kiện này xảy ra khi người sử dụng kích chuột vào đối tượng.
- DoubleClick: Sự kiện này xảy ra khi người sử dụng kích đúp chuột vào đối tượng.

VI.4. Ứng dụng MDI Form, MenuStrip

Ví dụ sau đây trình bày việc xây dựng một ứng dụng trong đó các chức năng của ứng dụng được thể hiện bằng menu. Khi người sử dụng chọn chức năng trên menu, Form tương ứng sẽ được triệu gọi. Trong ví dụ, ứng dụng được xây dựng bằng cách sử dụng hai Form là LogIn và Register của mục IV.3 và V.4 trong chương này. Ứng dụng được thực hiện theo các bước sau.

VI.4.1. Tạo project và cấu hình MDI Form

Khi tạo mới một project, Visual Studio sẽ tạo ra một Form mặc định có tên là Form1. Đổi tên Form này thành MainForm sau đó chỉnh thuộc tính IsMdiContainer thành giá trị *true*.



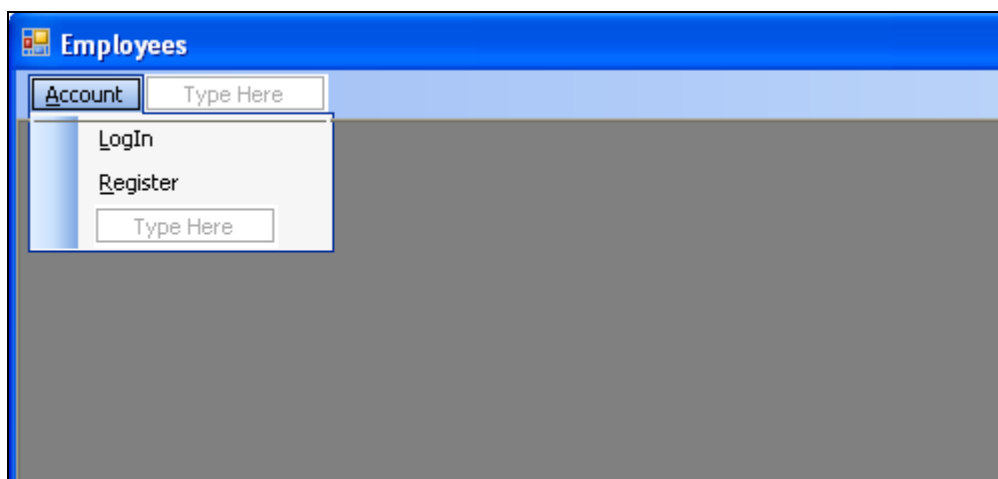
Hình 82 – Thay đổi thuộc tính để được MDI Form

VI.4.2. Tạo Form LogIn và Form Register

Tạo Form LogIn và Register theo hướng dẫn ở mục IV.3 và V.4 của chương này. Người lập trình có thể copy các Form này rồi dán vào project, sau đó thay đổi namespace trùng với tên project thì có thể sử dụng lại được các Form đã có.

VI.4.3. Tạo MenuStrip

Thực hiện kéo thả MenuStrip từ Toolbox panel vào Form ta được một thanh Menu. Trên thanh Menu, tạo Item có tên là Account rồi sau đó tạo tiếp trong mục Account hai ToolStripMenuItem khác có tên là LogIn và Register. Để tạo các Item cho menu, chỉ cần nhập tên vào các TextBox có sẵn (có tên là “Type Here”), Visual Studio sẽ tự động tạo ra các đoạn mã tạo Menu.



Hình 83 – Tạo thanh Menu và ToolStripMenuItem

VI.4.4. Viết sự kiện cho từng ToolStripMenuItem

Thực hiện kích đúp chuột vào ToolStripMenuItem, Visual Studio sẽ chuyển con trỏ soạn thảo đến phương thức của sự kiện Click của đối tượng. Trong phương thức này, lập trình thực hiện việc tạo Form và gán vào Form chính.

```
private void logInToolStripMenuItem_Click(object sender, EventArgs e)
{
    foreach (Form frm in this.MdiChildren)
    {
        frm.Close();
    }
    FrmLogIn frmLogIn = new FrmLogIn();
    frmLogIn.MdiParent = this;
    frmLogIn.Show();
}
```

Hình 84 – Viết sự kiện cho ToolStripMenuItem

Trong sự kiện này, thông thường chương trình thực hiện các bước sau. Đầu tiên là đóng tất cả các Form con hiện có trong Form chính, lệnh *foreach* được minh họa trong hình trước đó thể hiện công việc này. Bước thứ hai là tạo mới Form cần triệu gọi, toán tử *new* được sử dụng trong trường hợp này để khởi tạo đối tượng. Bước thứ ba là gán Form con vào Form chính, trong bước này ta gán thuộc tính *MdiParent* của Form con thành giá trị *this* (*this* trong trường hợp này là con trỏ chỉ đến Form chính). Bước cuối cùng là gọi lệnh hiển thị Form con, trong trường hợp này, phương thức *Show()* được sử dụng.

TƯƠNG TÁC CƠ SỞ DỮ LIỆU

I. Mục tiêu

Trong chương này, giáo trình sẽ trình bày phương pháp tương tác với cơ sở dữ liệu bằng C# dựa trên mô hình ADO.NET với hệ quản trị cơ sở dữ liệu Ms SQL Server 2000 hoặc 2005. Người học có thể dựa vào mô hình ADO.NET để tự nghiên cứu việc tương tác với các hệ quản trị cơ sở dữ liệu khác thông qua các tài liệu được đề cập trong phần tài liệu tham khảo ở cuối giáo trình. Những nội dung được đề cập trong chương này gồm

- ADO.NET
- Ms SQL Server 2000 hoặc 2005
- Stored Procedure
- SqlConnection
- SqlCommand
- SqlDataReader
- SqlDataAdapter
- DataSet, DataTable
- DataGridView

Kết thúc chương này, người học có thể xây dựng được ứng dụng tác nghiệp với các xử lý liên quan đến cơ sở dữ liệu.

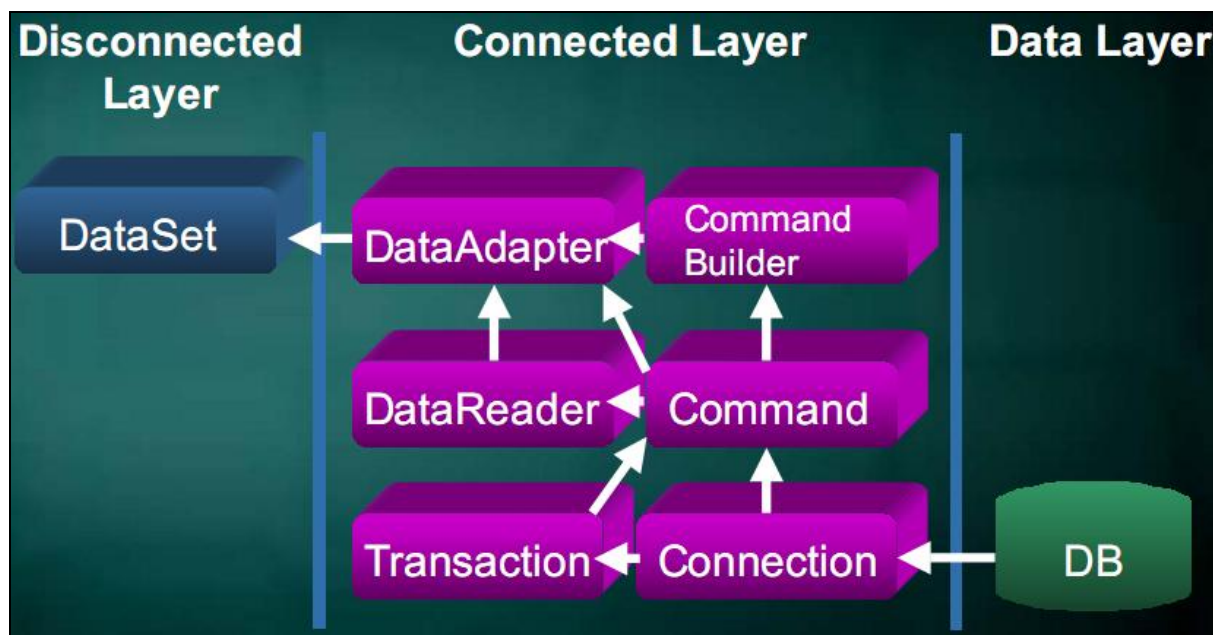
II. ADO.NET

II.1. Khái niệm

ADO.NET (ActiveX Data Objects .Net) là một phần của .NET Framework, nó được xem là “bộ thư viện lớp” chịu trách nhiệm xử lý dữ liệu trong ngôn ngữ MS.NET. ADO.NET được thiết kế với dạng dữ liệu “ngắt kết nối”, nghĩa là chúng ta có thể lấy cả một cấu trúc phức tạp của dữ liệu từ cơ sở dữ liệu, sau đó ngắt kết nối với cơ sở dữ liệu rồi mới thực hiện các thao tác cần thiết. Đây là một sự tiến bộ về mặt thiết kế bởi vì thiết kế ADO trước đây luôn cần duy trì một kết nối trong quá trình thao tác dữ liệu.

II.2. Kiến trúc

Kiến trúc của ADO.NET gồm hai tầng cơ bản: tầng kết nối (Connected Layer) và tầng không kết nối (Disconnected Layer).



Hình 85 – Kiến trúc của ADO.NET

Tầng kết nối: phần này được sử dụng khi ta kết nối với cơ sở dữ liệu và thao tác dữ liệu, yêu cầu phải thực hiện kết nối với cơ sở dữ liệu khi đang thao tác. Các đối tượng của phần này bao gồm:

- **Connection:** Đối tượng này thực hiện việc quản lý đóng /mở kết nối tới Cơ sở dữ liệu. Vì có nhiều nguồn dữ liệu (DataSource) khác nhau do đó .NET Framework cũng cung cấp nhiều dạng Connection khác nhau: SqlConnection tương ứng với Ms SQL Server, OleDbConnection tương ứng với Access, OracleConnection tương ứng với cơ sở dữ liệu Oracle.
- **Command :** Đối tượng này thực hiện các câu lệnh tương tác truy vấn, rút trích dữ liệu từ cơ sở dữ liệu khi đã thiết lập kết nối tới dữ liệu và trả về kết quả. Tương tự như Connection, Command cũng có nhiều dạng để người lập trình sử dụng tương ứng với các hệ quản trị cơ sở dữ liệu liên quan.
- **DataReader :** Đối tượng phục vụ việc xử lý đọc dữ liệu. Đối tượng này thường được dùng cho các ứng dụng website vì đối tượng này chỉ cho phép xử lý một luồng dữ liệu trong một thời điểm. DataReader cũng có nhiều dạng tương ứng với nhiều loại cơ sở dữ liệu khác nhau. Dữ liệu của đối tượng được tạo ra khi đối tượng Command thực hiện phương thức ExecuteReader().
- **DataAdapter :** Đây là đối tượng rất quan trọng của ADO.NET, đối tượng này là cầu nối giữa cơ sở dữ liệu và DataSet. DataSet là đối tượng ngắt kết nối nên không thể tương tác trực tiếp với cơ sở dữ liệu, vì thế nó cần một đối tượng trung gian lấy dữ liệu từ cơ sở dữ liệu cho nó. Vì DataAdapter khi thao tác với cơ sở dữ liệu vẫn phải duy trì kết nối nên DataAdapter thường được sử dụng trong các ứng dụng trên máy đơn.

Tầng không kết nối: chỉ có một đối tượng chịu trách nhiệm ngắt kết nối đó chính là DataSet. Trong kiến trúc của ADO.NET, DataSet được thiết kế tách biệt với cơ sở dữ liệu và có nhiệm vụ nhận dữ liệu về từ DataAdapter và xử lý.

II.3. Các namespace phục vụ cho ADO.NET

Visual Studio cung cấp nhiều namespace phục vụ cho việc tương tác với hầu hết các hệ quản trị cơ sở dữ liệu. Giáo trình này chỉ trình bày việc tương tác với hệ quản trị cơ sở dữ liệu Ms SQL Server 2000 hoặc 2005, nên chỉ đề cập đến các namespace cơ bản sau:

- System.Data: Cung cấp các lớp truy xuất dữ liệu chung.
- System.Data.Common: Cung cấp các lớp dùng chung khi kết nối với các hệ quản trị cơ sở dữ liệu khác nhau.
- System.Data.SqlClient: Cung cấp các lớp phục vụ việc kết nối với Ms SQL Server.
- System.Data.SqlTypes: Cung cấp kiểu dữ liệu dùng trong SQL Server.

III. SqlConnection, SqlCommand

III.1. SqlConnection

III.1.1. Khái niệm SqlConnection

SqlConnection là đối tượng chịu trách nhiệm quản lý kết nối tới nguồn dữ liệu (Data Source). Đối tượng SqlConnection của ADO.NET chỉ nhận một tham số đầu vào là chuỗi kết nối (connection string). Trong chuỗi kết nối, các thông số được cách nhau bằng dấu “;”, chuỗi kết nối này có các thông số sau:

- Provider: tên hệ quản trị cơ sở dữ liệu, đối với cơ sở dữ liệu Access cần khai báo là SQLOLEDB. Đối với SQL Server thì chuỗi kết nối không cần thuộc tính này.
- DataSource (hoặc Server): tên / địa chỉ Database Server cần kết nối tới.
- Initial catalog (hoặc Database): tên của cơ sở dữ liệu cần tương tác.
- Uid: tài khoản để đăng nhập vào Database Server.
- Pwd: mật khẩu để đăng nhập vào Database Server.

Đối với việc kết nối với cơ sở dữ liệu SQL Server thì tham số Provider có thể được bỏ đi và chuỗi kết nối như sau:

```
Server=(local);Database=CSharpEx;uid=csharp;pwd=cpass;
```

III.1.2. Các thuộc tính của SqlConnection

- ConnectionString: thuộc tính thiết lập hoặc lấy chuỗi kết nối.
- ConnectionTimeout: thuộc tính thiết lập hoặc lấy thời gian chờ trong khi truy xuất vào cơ sở dữ liệu. Khi truy xuất, chương trình sẽ chờ đúng khoảng thời gian này nếu chờ qua khoảng thời gian này mà vẫn không kết nối được vào cơ sở dữ liệu thì chương trình sẽ báo lỗi.
- Database: thuộc tính thiết lập hoặc lấy tên của cơ sở dữ liệu của đối tượng SqlConnection hiện thời.

- DataSource: thuộc tính thiết lập hoặc lấy tên của Database Server của đối tượng SqlConnection hiện thời.
- State: thuộc tính này lấy trạng thái hiện thời của SqlConnection bao gồm: Connecting, Broken, Open, Closed, Executing, Fetching.

III.1.3. Các phương thức của SqlConnection

- BeginTransaction: phương thức này được sử dụng cho trường hợp xử lý giao tác của ứng dụng. Việc xử lý giao tác rất có lợi trong khi xử lý dữ liệu từ database vì có lúc việc xử lý dữ liệu gặp lỗi và lúc ấy cần thực hiện câu lệnh như Rollback để khôi phục lại trạng thái của dữ liệu trước khi xử lý.
- Open: phương thức này thực hiện việc mở một kết nối. Phương thức này cần được thực hiện trong khối try/catch để xử lý các biệt lệ.
- Close: phương thức này thực hiện việc đóng kết nối và giải phóng tài nguyên. Phương thức này thường được đặt trong khối finally để thực hiện giải phóng tài nguyên khi kết thúc sử lý với cơ sở dữ liệu.

Ví dụ sau trình bày việc mở và đóng một kết nối tới cơ sở dữ liệu.

```
string cnnStr = "Server=(local);Database=CSharpEx;uid=csharp;pwd=cpass;";
SqlConnection myConnection = null;
try
{
    myConnection = new SqlConnection(cnnStr);
    myConnection.Open();
}
catch(SqlException exception1)
{
    MessageBox.Show(exception1.ToString());
}
finally
{
    if (myConnection != null) myConnection.Close();
}
```

Hình 86 – Minh họa việc mở kết nối và đóng kết nối đến cơ sở dữ liệu

III.2. SqlCommand

III.2.1. Khái niệm SqlCommand

SqlCommand là đối tượng thực hiện các lệnh tương tác với cơ sở dữ liệu. Đối tượng này được xây dựng để vừa có thể xử lý các stored procedure của SQL Server vừa có thể thực hiện các câu lệnh truy vấn trực tiếp đến cơ sở dữ liệu.

III.2.2. Các thuộc tính của SqlCommand

- CommandType: Thiết lập hoặc lấy kiểu lệnh tương tác, lệnh tương tác này có thể là stored procedure hoặc là câu lệnh truy vấn.

- **CommandText**: Thiết lập hoặc lấy lệnh thao tác với dữ liệu. Lệnh này có thể là tên của stored procedure đã có sẵn trong cơ sở dữ liệu hoặc là câu lệnh truy vấn tùy thuộc vào thuộc tính **CommandType**.
- **CommandTimeout**: Thiết lập hoặc lấy thời gian chờ thực hiện lệnh. Sau khoảng thời gian này nếu tương tác cơ sở dữ liệu vẫn chưa xong thì chương trình sẽ báo lỗi.
- **Parameters**: Các tham số truyền vào cho đối tượng command. Thuộc tính này được sử dụng hiệu quả khi **CommandType** là stored procedure.
- **Connection**: Thiết lập hoặc lấy kết nối đang được đối tượng **SqlCommand** sử dụng.

III.2.3. Các phương thức của SqlCommand

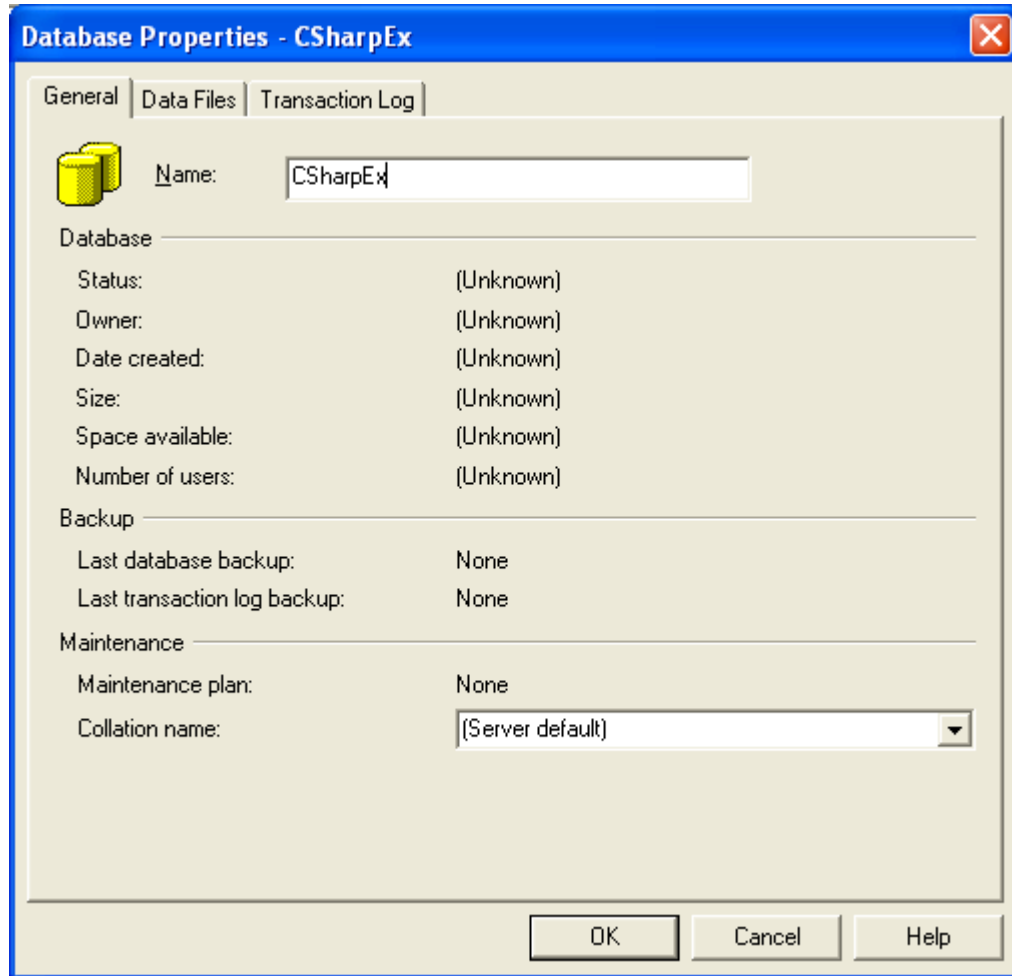
- **ExecuteReader**: Thực thi câu lệnh **CommandText** của đối tượng **SqlCommand** và trả về kiểu **SqlDataReader**. Phương thức này thường được sử dụng khi nội dung câu lệnh tương tác cơ sở dữ liệu là lệnh *select*.
- **ExecuteNonQuery**: Thực thi câu lệnh **CommandText** của đối tượng **SqlCommand**, đây là dạng câu lệnh cập nhật cơ sở dữ liệu (thêm hoặc xoá hoặc sửa) nên chỉ trả về số dòng bị ảnh hưởng mà không trả về dòng dữ liệu nào.
- **ExecuteScalar**: Thực thi câu truy vấn của đối tượng **Command** và chỉ trả về cột đầu tiên của dòng đầu tiên của kết quả. Các kết quả còn lại bị bỏ qua.

III.3. Ứng dụng SqlConnection, SqlCommand, ExcuteNonQuery

Ví dụ sau trình bày việc ứng dụng **SqlConnection** và **SqlCommand** trong việc thêm một người dùng vào cơ sở dữ liệu khi người sử dụng làm việc với Form Register đã được trình bày ở mục V.4 của chương 6. Việc nâng cấp ứng dụng này được thực hiện từng bước như sau:

III.3.1. Tạo cơ sở dữ liệu

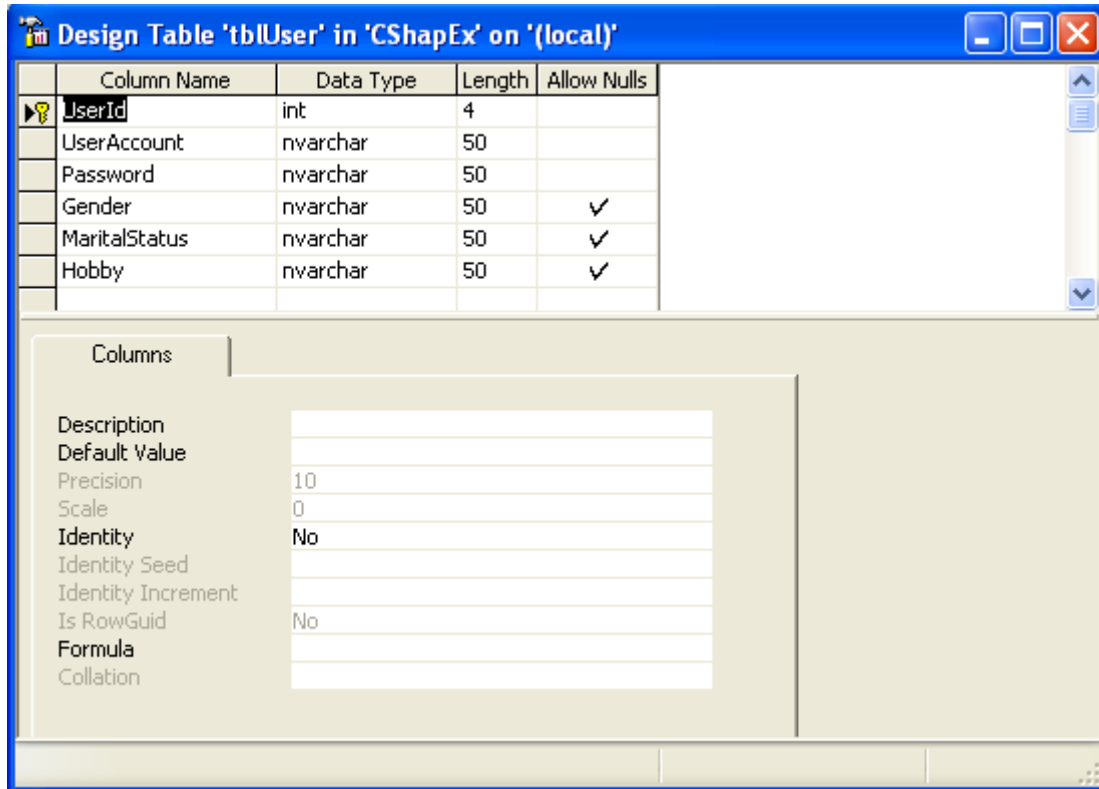
Khởi động Enterprise Manager của SQL Server 2000 và tạo cơ sở dữ liệu. Trong trường hợp của ví dụ này, cơ sở dữ liệu được tạo ra và được đặt tên là **CSharpEx**.



Hình 87 – Tạo cơ sở dữ liệu bằng SQL Server 2000

III.3.2. Tạo bảng tblUser

Người lập trình có thể tạo bảng bằng cách sử dụng chức năng Design Table của Enterprise Manager hoặc có thể tạo bằng câu lệnh *create table* thông qua Query Analyzer. (Các thao tác này người học có thể xem lại các học phần liên quan đến cơ sở dữ liệu đã học trước học phần này). Ví dụ này trình bày việc tạo cơ sở dữ liệu bằng chức năng Design Table của Enterprise Manager.

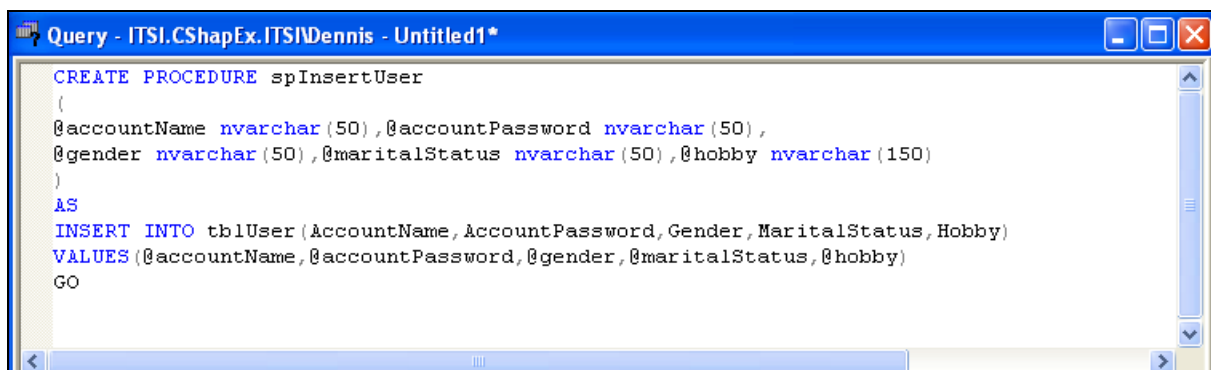


Hình 88 – Tạo bảng tblUser bằng chức năng Design Table

III.3.3. Tạo stored procedure

Stored procedure là khái niệm được dùng để chỉ tập hợp lệnh liên quan đến một chức năng nào đó mà các lệnh này được lưu trữ và thực thi ngay bên trong hệ quản trị cơ sở dữ liệu. Việc sử dụng stored procedure tạo nên sự tách biệt về mặt công việc, điều này làm cho việc phát triển phần mềm trở nên tối ưu vì hệ quản trị cơ sở dữ liệu và công nghệ lập trình ở tầng ứng dụng thường độc lập với nhau và có thể được xây dựng bởi những nhà cung cấp khác nhau.

Ví dụ này trình bày việc xây dựng stored procedure spInsertUser thực hiện việc thêm mới một tài khoản người dùng vào cơ sở dữ liệu. Stored procedure này được tạo ra bằng câu lệnh thông qua Query Analyzer.



Hình 89 – Tạo stored procedure

Stored procedure thông thường gồm hai phần: phần tham số và phần câu lệnh truy vấn. Trong phần tham số, các tham số bắt đầu bằng ký tự @. Phần câu lệnh truy vấn chứa câu lệnh truy vấn cơ sở dữ liệu và trong câu lệnh này có thể sử dụng các tham số ở phần tham số. Một stored procedure có thể chứa nhiều hơn một câu lệnh truy vấn.

III.3.4. Lập trình tương tác cơ sở dữ liệu

Quay lại ứng dụng Register ở mục V.4 của chương 5, khi người sử dụng bấm nút Register thì chương trình chỉ thông báo rằng người sử dụng đã đăng ký mà không hề thêm thông tin người dùng vào cơ sở dữ liệu. Để thông tin người dùng được thêm vào cơ sở dữ liệu (đã được tạo ở 3 bước trên), người lập trình thực hiện việc thay đổi sự kiện bên dưới của nút Register như sau:

Ban đầu, khai báo và khởi tạo đối tượng SqlConnection, trong trường hợp này đối tượng có tên là myConnection. Xem ví dụ ở hình 86.

Bước thứ hai là xây dựng đối tượng SqlCommand, trong trường hợp này đối tượng có tên là myCommand. Quá trình xây dựng đối tượng SqlCommand bao gồm các việc: khởi tạo đối tượng, gán thuộc tính Connection, gán thuộc tính CommandType là StoredProcedure, khai báo tên stored procedure cần sử dụng và thêm các tham số với các giá trị lấy từ các đối tượng trên Form.

```
SqlCommand myCommand = new SqlCommand();
myCommand.Connection = myConnection;
myCommand.CommandType = CommandType.StoredProcedure;
myCommand.CommandText = "spInsertUser";
myCommand.Parameters.Add("@accountName", SqlDbType.NVarChar, 50).Value
    = txtAccountName.Text;
myCommand.Parameters.Add("@accountPassword", SqlDbType.NVarChar, 50).Value
    = txtPassword.Text;
if (rdiMale.Checked) tempStr = "Male";
else tempStr = "Female";
myCommand.Parameters.Add("@gender", SqlDbType.NVarChar, 50).Value
    = tempStr;
tempStr = cbbMaritalStatus.SelectedItem.ToString();
myCommand.Parameters.Add("@maritalStatus", SqlDbType.NVarChar, 50).Value
    = tempStr;
tempStr = "";
foreach (Control chk in this.Controls)
{
    if (chk is CheckBox)
    {
        if (((CheckBox)chk).Checked)
            tempStr = tempStr + chk.Text + " ";
    }
}
myCommand.Parameters.Add("@hobby", SqlDbType.NVarChar, 50).Value
    = tempStr;
```

Hình 90 – Xây dựng đối tượng SqlCommand

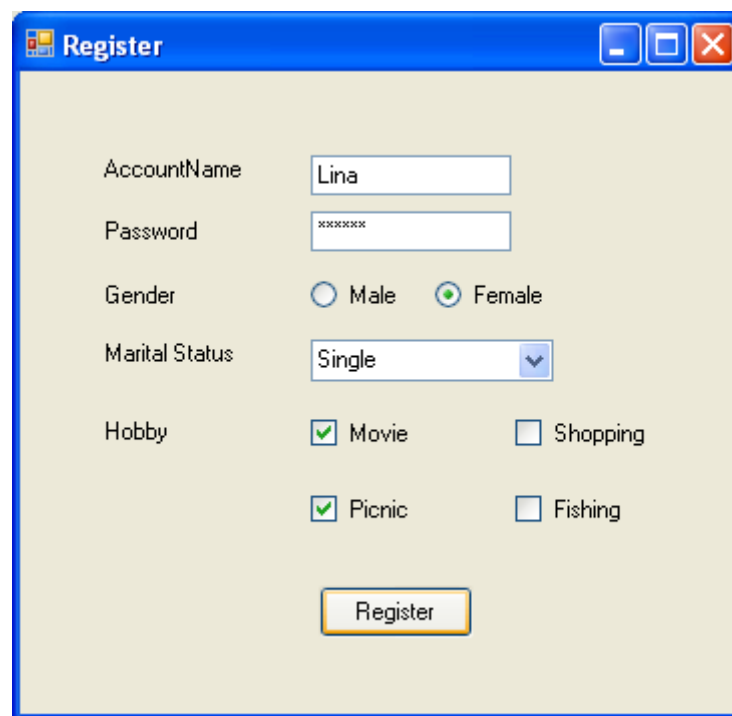
Bước cuối cùng là mở kết nối và gọi phương thức thực hiện truy vấn. Trong bước này, phương thức `Open()` của `SqlConnection` và phương thức `ExecuteNonQuery()` của `SqlCommand` được sử dụng.

```
myConnection.Open();  
myCommand.ExecuteNonQuery();  
MessageBox.Show("Create user successfully");
```

Hình 91 – Triệu gọi các phương thức thực hiện tương tác cơ sở dữ liệu

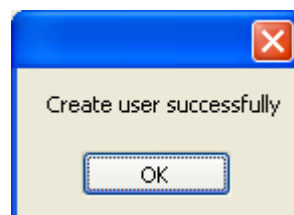
III.3.5. Biên dịch và chạy ứng dụng

Thực hiện biên dịch và chạy ứng dụng ta được giao diện giống như khi chạy ứng dụng Register. Tuy nhiên, khi bấm vào nút Register thì ứng dụng sẽ lưu thông tin người dùng vào cơ sở dữ liệu.



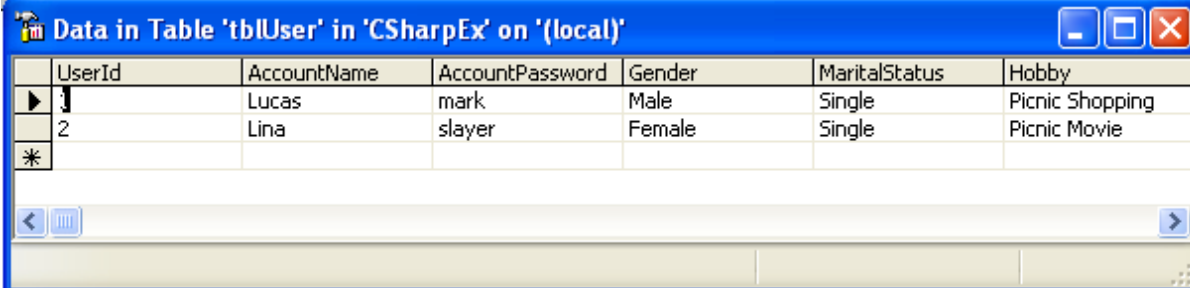
Hình 92 – Kết quả sau khi biên dịch và chạy ứng dụng

Sau khi bấm nút Register, dữ liệu người dùng sẽ được lưu vào cơ sở dữ liệu và chương trình sẽ hiển thị thông báo là tạo người dùng thành công.



Hình 93 – Thông báo sau khi đã tương các cơ sở dữ liệu thành công

Cơ sở dữ liệu lúc này sẽ có thêm một tài khoản người dùng mới trong bảng tblUser, tài khoản này được tạo với các thông tin từ Form vừa mới nhập.



UserId	AccountName	AccountPassword	Gender	MaritalStatus	Hobby
1	Lucas	mark	Male	Single	Picnic Shopping
2	Lina	slayer	Female	Single	Picnic Movie

Hình 94 – Thông tin người dùng trong cơ sở dữ liệu

IV. SqlDataReader và phương thức ExecuteReader

IV.1. SqlDataReader

IV.1.1. Khái niệm SqlDataReader

SqlDataReader là đối tượng được .Net Framework cung cấp nhằm phục vụ việc truy cập vào cơ sở dữ liệu. Dữ liệu sau khi được truy vấn từ cơ sở dữ liệu sẽ được lưu trữ trong SqlDataReader dưới dạng bảng gồm nhiều dòng và nhiều cột giống như trong cơ sở dữ liệu.

Dữ liệu trong SqlDataReader được truy xuất một cách tuần tự và một chiều. Điều này nghĩa là người lập trình chỉ được đọc từ SqlDataReader một cách tuần tự, muốn đọc được dòng thứ i thì phải đọc $i-1$ dòng trước đó (giống như việc đọc file trên đĩa). SqlDataReader không cung cấp cơ chế sắp xếp cũng như cơ chế truy xuất ngẫu nhiên, do đó người lập trình thường sử dụng vòng lặp khi thực hiện đọc dữ liệu từ SqlDataReader để hiển thị lên giao diện.

IV.1.2. Các thuộc tính của SqlDataReader

- FieldCount: thuộc tính này trả về số trường trong *record* hiện hành.
- IsClosed: thuộc tính này trả về trạng thái của SqlDataReader là đóng hay mở.
- HasRows: thuộc tính này chỉ định SqlDataReader có *record* dữ liệu nào hay không.

IV.1.3. Các phương thức của SqlDataReader

- Close: phương thức này thực hiện việc đóng SqlDataReader và giải phóng tài nguyên.
- GetBoolean, GetByte, GetChar, GetDateTime, GetDecimal: lấy các giá trị tại cột đang xét tùy vào kiểu dữ liệu.
- GetValue, GetValues: lấy về giá trị hoặc tập giá trị ở dạng “nguyên thủy” (kiểu dữ liệu gốc của Database).
- Read: Đọc *record* tiếp theo của DataReader.

IV.2. Phương thức ExecuteReader

ExecuteReader là phương thức của đối tượng SqlCommand, phương thức này thực hiện việc truy vấn cơ sở dữ liệu trong trường hợp nội dung câu truy vấn (CommandText) là câu lệnh *select*. Kết quả trả về của phương thức ExecuteReader là SqlDataReader, đối tượng này tổ chức dữ liệu thành bảng giống như kết quả trả về của câu lệnh truy vấn trong hệ quản trị cơ sở dữ liệu.

IV.3. Ứng dụng SqlDataReader và phương thức ExecuteReader

Ví dụ sau trình bày việc ứng dụng DataReader và phương thức ExecuteReader để xử lý sự kiện LogIn trong ứng dụng đã trình bày ở mục IV.3 của chương 6. Trong sự kiện LogIn này, chương trình sẽ xử lý việc truy vấn cơ sở dữ liệu để kiểm tra xem tài khoản có AccountName và Password mà người sử dụng nhập vào từ các TextBox đã có sẵn trong cơ sở dữ liệu hay không. Đoạn mã của sự kiện LogIn như sau.

```
string cnnStr = "Server=(local);Database=CSharpEx;uid=csharp;pwd=cpass;";
string sqlStr = "select AccountName from tblUser where (AccountName = '"
    + txtUsername.Text + "' and AccountPassword = '"
    + txtPassword.Text + "')";
SqlConnection myConnection = null;
try
{
    myConnection = new SqlConnection(cnnStr);
    myConnection.Open();
    SqlCommand myCommand = new SqlCommand(sqlStr, myConnection);
    SqlDataReader dataReader = myCommand.ExecuteReader();
    if (dataReader.HasRows)
    {
        MessageBox.Show("Login successfully.");
    }
    else
    {
        MessageBox.Show("Incorrect username or password.");
    }
}
catch (Exception exception1)
{
    MessageBox.Show(exception1.ToString());
}
finally
{
    if (myConnection != null) myConnection.Close();
}
```

Hình 95 – Ứng dụng DataReader và ExecuteQuery

Trong đoạn mã trên thực hiện việc kiểm tra đăng nhập bằng cách thực hiện lệnh truy vấn *select*. Lệnh này thực hiện việc tìm trong bảng tblUser các *record* có AccountName và AccountPassword trùng với dữ liệu nhập vào từ các TextBox. Kết quả thu được từ câu lệnh truy vấn này được lưu trữ trong đối tượng SqlDataReader. Nếu đối tượng này có record

(HasRows = true) thì tài khoản người dùng nhập vào là hợp lệ và chương trình sẽ thông báo đăng nhập thành công, ngược lại thì chương trình sẽ báo lỗi.

V. SqlDataAdapter, DataSet và DataGridView

V.1. SqlDataAdapter

SqlDataAdapter là một khái niệm .Net Framework, khái niệm này được dùng để chỉ đối tượng làm cầu nối giữa cơ sở dữ liệu và DataSet. SqlDataAdapter chứa một phần dữ liệu của cơ sở dữ liệu và hoạt động theo cơ chế “kết nối”. Với cơ chế “kết nối”, SqlDataAdapter được trang bị một số phương thức để lấy dữ liệu từ cơ sở dữ liệu hoặc điền ngược dữ liệu vào cơ sở dữ liệu khi cần thiết.

Các thuộc tính thông dụng của SqlDataAdapter bao gồm:

- SelectCommand: Thuộc tính này quy định câu lệnh select của SqlDataAdapter. Câu lệnh select của thuộc tính này thường được triệu gọi khi SqlDataAdapter thực hiện phương thức Fill().
- InsertCommand, UpdateCommand, DeleteCommand: Các thuộc tính này lần lượt quy định câu lệnh insert, update, delete của SqlDataAdapter. Các câu lệnh này được triệu gọi khi SqlDataAdapter thực hiện phương thức Update() để cập nhật dữ liệu từ SqlDataAdapter vào cơ sở dữ liệu.

Các phương thức thông dụng của SqlDataAdapter bao gồm:

- Fill: Đối số của phương thức này là DataSet hoặc DataTable. Phương thức này thực hiện việc điền dữ liệu tương ứng với câu lệnh select từ cơ sở dữ liệu vào một DataSet hoặc DataTable.
- Update: Thực hiện việc cập nhật dữ liệu từ SqlDataAdapter vào cơ sở dữ liệu.

V.2. DataSet

DataSet là khái niệm của .Net Framework, khái niệm này được dùng để chỉ đối tượng ở tầng “không kết nối” trong mô hình ADO.NET. DataSet được thiết kế tách biệt với cơ sở dữ liệu và khi vận hành không cần biết đến việc cơ sở dữ liệu thuộc kiểu gì, kết nối ra sao. Nhiệm vụ của DataSet là nhận dữ liệu về từ DataAdapter và xử lý nó.

DataSet có thể được xem như một cơ sở dữ liệu trong bộ nhớ gồm tất cả các bảng, dữ liệu, quan hệ và ràng buộc dữ liệu. DataSet có nhiều đối tượng cấp thấp hơn đi kèm với nó như : DataTable (tương đương với một bảng), cấp thấp hơn của DataTable có các đối tượng DataRow (tương đương với một dòng), DataColumn (tương đương với một cột), DataRelation (tương đương với các quan hệ).

DataSet nhận dữ liệu từ DataAdapter thông qua phương thức Fill() và dữ liệu này được hiển thị lên giao diện thông qua thuộc tính DataSource của đối tượng trình bày. Việc sử dụng DataSet là một tiến bộ lớn của kiến trúc ADO.NET tuy nhiên với các ứng dụng Website, việc

sử dụng DataSet không được khuyến khích vì đối tượng DataSet được xem là quá lớn, nặng nề khó thích hợp cho đường truyền vốn rất hạn chế.

V.3. DataGridView

V.3.1. *Khái niệm DataGridView*

DataGridView là khái niệm dùng để chỉ đối tượng trình bày giao diện trên Form, đối tượng này bố trí dữ liệu thành nhiều cột và nhiều hàng. Đối tượng này còn được gọi là lưới dữ liệu.

V.3.2. *Các thuộc tính của DataGridView*

- DataSource: Thuộc tính này quy định bảng dữ liệu được hiển thị lên giao diện. Đối tượng được gán vào thuộc tính này thông thường là DataTable.
- CurrentRow: Thuộc tính này trả về dòng đang được chọn, thông tin trong dòng này được chứa đựng trong đối tượng DataGridViewRow.
- CurrentCell: Thuộc tính này trả về ô đang được chọn, thông tin trong ô này được chứa đựng trong đối tượng DataGridViewCell.

V.3.3. *Các sự kiện của DataGridView*

- SelectionChanged: Sự kiện này xảy ra khi người sử dụng di chuyển con trỏ soạn thảo trên các dòng của DataGridView.
- DoubleClick: Sự kiện này xảy ra khi người sử dụng kích đúp chuột vào DataGridView.

V.4. Ứng dụng SqlDataAdapter, DataSet, DataGridView

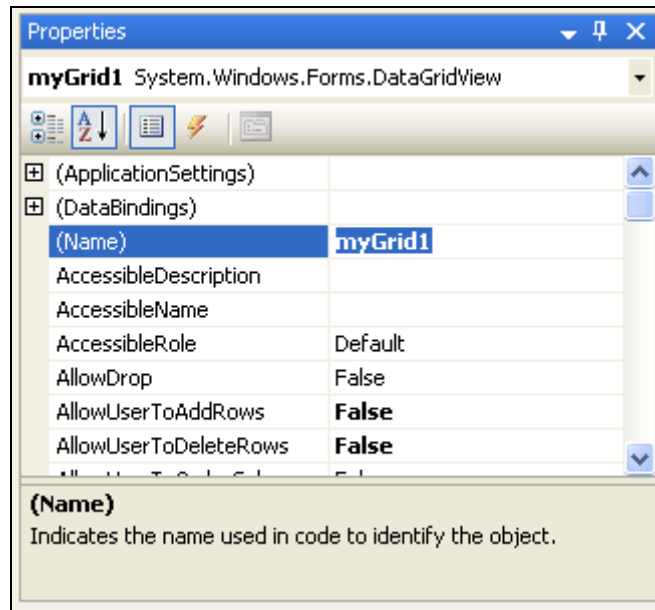
Ví dụ sau trình bày việc ứng dụng SqlDataAdapter, DataSet và DataGridView trong việc hiển thị dữ liệu từ cơ sở dữ liệu lên giao diện. Ứng dụng được thực hiện từng bước như sau:

V.4.1. *Tạo project*

Thực hiện tạo project Windows Application như đã trình bày ở mục IV.3 chương 6. Sau khi project được tạo thì một Form chính có tên là Form1 được tạo ra.

V.4.2. *Thêm đối tượng DataGridView*

Thực hiện việc kéo thả đối tượng DataGridView từ Toolbox panel vào Form chính. Sau đó đặt tên cho DataGridView. Trong trường hợp này, DataGridView được đặt tên là myGrid1.



Hình 96 – Cấu hình DataGridView

V.4.3. Cài đặt sự kiện Load của Form

Như đã trình bày ở chương trước, sự kiện Load của Form xảy ra khi Form được khởi tạo. Trong ứng dụng này, sự kiện Load thực hiện việc kết nối cơ sở dữ liệu và lấy thông tin, sau đó điền vào DataGridView để hiển thị trên Form. Đoạn mã của sự kiện này được cài đặt như sau:

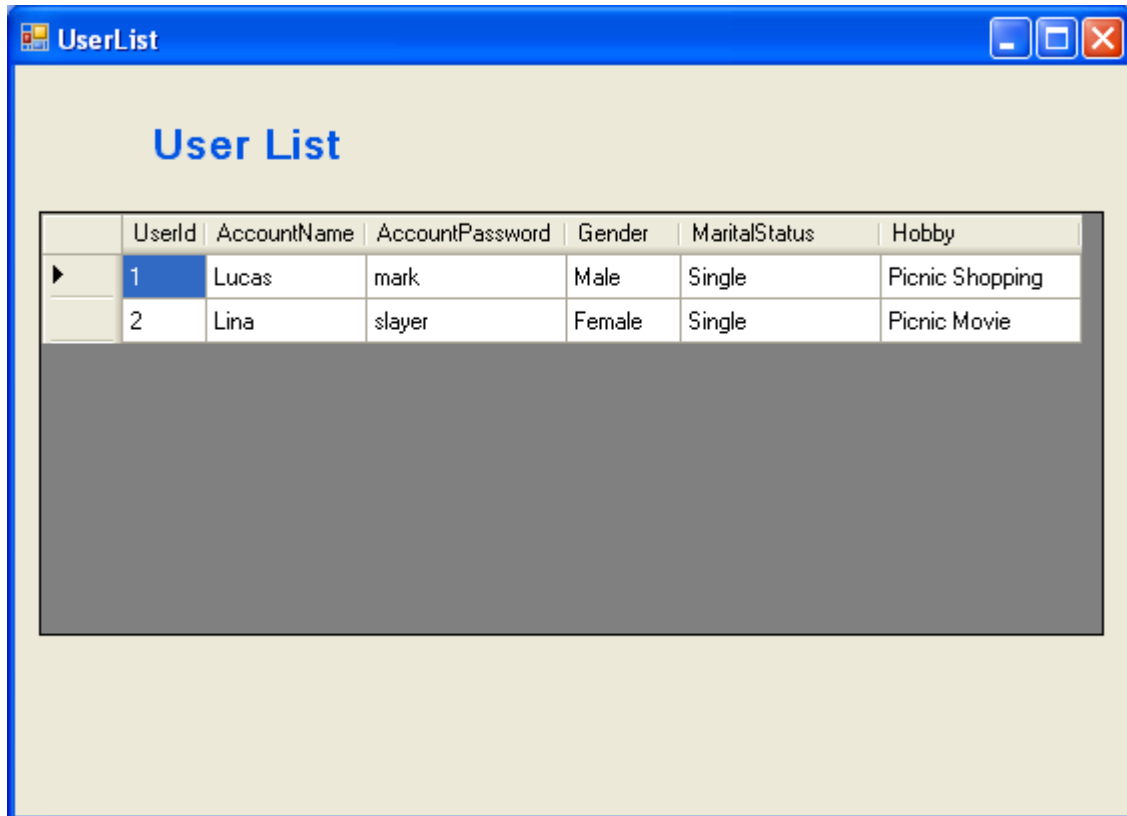
```
string cnnStr = "Server=(local);Database=CSharpEx;uid=csharp;pwd=cpass;";
string sqlStr = "select * from tblUser";
SqlConnection myConnection = null;
try
{
    myConnection = new SqlConnection(cnnStr);
    myConnection.Open();
    SqlDataAdapter myAdapter = new SqlDataAdapter(sqlStr, myConnection);
    DataSet myDataSet = new DataSet();
    myAdapter.Fill(myDataSet, "tblUser");
    myGrid1.DataSource = myDataSet.Tables["tblUser"];
}
catch (Exception exception1)
{
    MessageBox.Show(exception1.ToString());
}
finally
{
    if (myConnection != null) myConnection.Close();
}
```

Hình 97 – Sử dụng DataAdapter, DataSet và DataGridView

Trong đoạn mã minh họa trên, đối tượng SqlDataAdapter được khởi tạo bằng cách cung cấp hai tham số: câu lệnh truy vấn và đối tượng SqlConnection. Sau khi khởi tạo, đối tượng SqlDataAdapter thực hiện phương thức Fill() để điền dữ liệu từ cơ sở dữ liệu vào một bảng có tên là “tblUser” trong DataSet. Bảng này được hiển thị lên giao diện bằng cách gán nó cho thuộc tính DataSource của đối tượng DataGridView.

V.4.4. Biên dịch và chạy chương trình

Thực hiện biên dịch và chạy chương trình được kết quả như sau:



Hình 98 – Kết quả chạy chương trình

Kết quả chạy chương trình này hoàn toàn giống với kết quả khi thực hiện hiển thị dữ liệu của Sql Server bằng Enterprise Manager ở hình 94.

KẾT LUẬN

Giáo trình này trình bày phương pháp lập trình .NET với ngôn ngữ C# của phiên bản Visual Studio 2005. Tuy giáo trình chỉ trình bày những nội dung cơ bản nhất của công nghệ .NET, nhưng với những kiến thức này, người lập trình có thể tự nghiên cứu các chuyên đề cao hơn. Từ những nội dung trong giáo trình và những kiến thức từ các học phần tiên quyết của học phần này, người học có đủ kiến thức để xây dựng một ứng dụng Windows.

Tuy nhiên, giáo trình được viết cho sinh viên cao đẳng nên về nội dung còn có nhiều hạn chế, cách tiếp cận vấn đề chưa thật sự chính xác và cách giải quyết vấn đề chưa hoàn toàn tối ưu và tổng quát. Giáo trình chỉ đi sâu vào kỹ năng thực hành, các ví dụ minh họa được trình bày theo phương pháp “hướng dẫn từng bước” (step – by – step) nên giáo trình chưa thể hướng người học đến tư duy trừu tượng của lập trình bậc cao. Do thời lượng giảng dạy không nhiều nên giáo trình cũng chưa thể trình bày một ứng dụng hoàn chỉnh của một dự án phần mềm thực tế.

Cuối cùng, tác giả mong người học có những góp để kịp thời cải tiến và hoàn thiện.

TÀI LIỆU THAM KHẢO

- [1] Phương Lan. *Lập trình Windows với C#.Net*. Nhà xuất bản lao động – xã hội
- [2] Phạm Hữu Khang, Đoàn Thiện Ngân. *C# 2005*. Tập 1, 2, 3, 4, 5. Nhà xuất bản Lao động và Xã hội.
- [3] Jesse Liberty. *Programming C# for Visual Studio .NET*. O'Reilly.
- [4] Jason Price. *Mastering C Sharp Database Programming*. Sybex.
- [5] Jesse Liberty. *Programming C#*. O'Reilly.
- [6] Rebecca M. Riordan. *Microsoft ADO.NET Step by Step*. MS Press.

TÓM TẮT LUẬN VĂN TỐT NGHIỆP

Phần này viết tên đề tài (chữ nhỏ 13pt) và :

tóm tắt nội dung LUẬN VĂN TỐT NGHIỆP, viết ngắn gọn và rõ ràng (15 đến 20 dòng) cho biết :

- Đề tài đã được đặt ra như thế nào (bối cảnh), mục đích (giải thích rõ hơn tên đề tài) và nhiệm vụ phải thực hiện (các mục tiêu cụ thể và kết quả cần có)
- SV đã giải quyết vấn đề gì (đã nghiên cứu lý thuyết, thực tiễn như thế nào, đã đề xuất được những giải pháp (biện pháp) hay sáng kiến gì ?).
- SV đã giải quyết đến đâu (nêu một số kết quả tiêu biểu).