TRƯỜNG CAO ĐẲNG NGHỀ CÔNG NGHIỆP HÀ NỘI

Chủ biên: Bùi Quang Ngọc

Đồng tác giả: Lê Văn Úy

GIÁO TRÌNH

LẬP TRÌNH TRỰC QUAN

(Lưu hành nội bộ)

Hà Nội năm 2012

Tuyên bố bản quyền

Giáo trình này sử dụng làm tài liệu giảng dạy nội bộ trong trường cao đẳng nghề Công nghiệp Hà Nội

Trường Cao đẳng nghề Công nghiệp Hà Nội không sử dụng và không cho phép bất kỳ cá nhân hay tổ chức nào sử dụng giáo trình này với mục đích kinh doanh.

Mọi trích dẫn, sử dụng giáo trình này với mục đích khác hay ở nơi khác đều phải được sự đồng ý bằng văn bản của trường Cao đẳng nghề Công nghiệp Hà Nội

MỤC LỤC

ĐỀ MỤC **TRANG**

MÔ ĐUN LẬP TRÌNH TRỰC QUAN

Mã số mô đun: MĐ23
Thời gian mô đun: 120 giờ; (Lý thuyết: 45 giờ; Thực hành: 75 giờ)

Vị trí, ý nghĩa, vai trò mô đun

- Vị trí: Mô đun được bố trí sau khi sinh viên học xong các môn học chung, các môn học cơ sở chuyên ngành đào tạo chuyên môn nghề.

- Tính chất: đào tạo chuyên môn nghề.

Mục tiêu của mô đun:

Về kiến thức:

- Hiểu được vai trò của công nghệ lập trình trực quan;
- Phân tích xác định nhiệm vụ chương trình (phải làm gì), xác định đối tượng điều khiển dữ liệu, dữ liệu và cấu trúc dữ liệu của hệ thống phù hợp với ngôn ngữ đã chọn để xây dựng các ứng dụng.
- Thiết kế tìm giải pháp kỹ thuật (làm thế nào) đối với những công việc đã xác định trong giai đoạn phân tích;
- Mô tả hằng và biến dùng trong chương trình, Trình bày được cấu trúc, cú pháp, quy trình và yêu cầu khi sử dụng các câu lệnh;
- Vận dụng điều kiện, trợ giúp môi trường của ngôn ngữ lập trình, chẳng hạn: trình biên tập mã lệnh;
- Vận dụng tốt các đối tượng cơ sở, cơ sở dữ liệu của ngôn ngữ lập trình : thuộc tính (properties), phương thức (Method), sự kiện (Event);

Về kỹ năng:

- Vận dụng quy tắc cú pháp và các đối tượng của ngôn ngữ lập trình.
- Thiết kế và xây dựng được bài tập, các chương trình ứng dụng có sự hỗ trợ của ngôn ngữ lập trình.
- Thực hiện được việc xây dựng các ứng dụng có cấu trúc, thuật toán hợp lý, mỹ thuật, phù hợp với yêu cầu người dùng.
- Kiểm định, hiệu chỉnh, hoàn thiện các ứng dụng.

Về thái độ:

- Nhận thức được tầm quan trọng của ngôn ngữ lập trình trực quan;
- Hình thành phong cách lập trình, vận dụng kết quả học tập vào việc xây dựng các ứng dụng.
- Bố trí vị trí làm việc khoa học, đảm bảo an toàn cho người và phương tiện học tập.

Mã bài	Tên bài	Loại bài dạy	Địa điểm	Thời lượng			
				Tổng số	Lý thuyết	Thực hành	Kiểm tra
MĐ23-	Tổng quan về C #	Lý	Lớp học	5	3	2	

01		thuyết					
MĐ23-02	Làm việc với Visual C#.Net	Tích hợp	Lớp học PTH	6	3	3	
MĐ23-03	Chương trình C#	Tích hợp	Lớp học PTH	8	3	5	
MĐ23-04	Nền tảng của C#	Tích hợp	Lớp học PTH	21	6	14	1
MĐ23-05	Các đối tượng điều khiển của C#	Tích hợp	Lớp học PTH	8	3	5	
MĐ23-06	File và registry Operation	Tích hợp	Lớp học PTH	15	5	9	1
MĐ23-07	Đồ hoạ và một số xử lý nâng cao	Tích hợp	Lớp học PTH	10	5	4	1
MĐ23-08	Truy xuất dữ liệu với ADO.NET	Tích hợp	Lớp học PTH	26	5	20	1
MĐ23-09	Xây dựng ứng dụng tổng hợp	Tích hợp	Lớp học PTH	18	2	16	1
TỔNG CỘNG				**120**	**45**	**70**	**5**

* Ghi chú: Thời gian kiểm tra được tích hợp giữa lý thuyết với thực hành được tính vào giờ thực hành.

YÊU CẦU VỀ ĐÁNH GIÁ HOÀN THÀNH MÔN HỌC/MÔ ĐUN

1. Phương pháp đánh giá

Hình thức kiểm tra hết môn có thể chọn một trong các hình thức sau:

+ Đối với lý thuyết :Viết, vấn đáp, trắc nghiệm
+ Đối với thực hành : Bài tập thực hành.

Thời gian kiểm tra:

+ Lý thuyết: Không quá 150 phút
+ Thực hành: Không quá 4 giờ

Thực hiện theo đúng qui chế thi, kiểm tra và công nhận tốt nghiệp trong dạy nghề hệ chính qui ở quyết định 14/2007/BLĐTB&XH ban hành ngày 24/05/2007 của Bộ trưởng Bộ LĐ-TB&XH.

2. Nội dung đánh giá

- Về kiến thức: Được đánh giá qua bài kiểm tra viết, trắc nghiệm đạt được các yêu cầu sau:

+ Cách thức lập trình trực quan.

+ Nắm bắt được một số khái niệm về câu lệnh, từ khoá, cú pháp, đối tượng, sự kiện để xây dựng một số ứng dụng cơ bản trong lập trình trực quan.

+ Có khả năng phân tích và xây dựng ứng dụng cho hệ thống dựa trên các ngôn ngữ có khả năng lập trình có thể .NET.

- Về kỹ năng: Đánh giá kỹ năng thực hành của sinh viên trong bài thực hành Lập trình trực quan đạt được các yêu cầu sau:
    + Sử dụng thành thạo các công cụ lập trình của Microsoft (C# để lập trình trực quan.
    + Thiết kế, lập trình được một ứng dụng thông dụng để phục vụ công việc quản trị mạng hoặc các ứng dụng thực tế.
- Về thái độ: Cẩn thận, tự giác, chính xác.

BÀI 1 : GIỚI THIỆU VỀ .NET 2008 và C#.
MÃ BÀI HỌC : MĐ23-01

Mục tiêu:

Sau khi học xong bài này, học viên có khả năng:

- Liệt kê được các thành phần chính của .NET Framework;
- Trình bày môi trường làm việc của .NET Framework;
- Liệt kê các phiên bản Visual Studio 2008;
- Kể tên các loại ứng dụng dùng C#;
- Trình bày được cấu trúc chương trình C#;
- Trình bày cấu trúc thư mục của ứng dụng dùng ngôn ngữ C# để xây dựng;
- Thực hiện các thao tác cài đặt, an toàn với máy tính.

TÓM TẮT BÀI:

- C# (C Sharp) là cuộc cách mạng của ngôn ngữ lập trình Microsoft C và Microsoft C++ với tính cách đơn giản, hiện đại, hướng đối tượng và có độ bảo mật cao.
- C# (C#.NET) là một trong bốn ngôn ngữ thuộc bộ Visual Studio.NET (C#, VB.NET, C++, J#.NET).
- Phiên bản đầu tiên Visual Studio.NET được Microsoft giới thiệu vào đầu năm 200 (.NET Framework phiên bản 1.0), tiếp theo đó là Visual Studio.NET 2003 (.NET Framework phiên bản 1.1) ra mắt giữa năm 2003 đã khẳng định được sức mạnh công nghệ chủ lực của Microsoft.
- Visual Studio 2005 chính thức công bố vào đầu tuần tháng 11 năm 2005 (.NET Framework phiên bản 2.0). Một lần nữa công nghệ Microsoft.NET đã và đang chinh phục các lập trình viên trên toàn thế giới với những đặc điểm mới về hướng đối tượng, phong phú hoá giao diện trực quan, dễ lập trình và độ bảo mật cao cho các ứng dụng qui mô lớn.

Các vấn đề chính sẽ được đề cập

- Tổng quan về .NET Framework.
- Giới thiệu về Visual Studio .NET 2008.
- Các loại ứng dụng dùng C#.
- Cấu trúc chương trình C# 2008.
- Cấu trúc thư mục của chương trình C# 2008.

NỘI DUNG :

A. LÝ THUYẾT

I. TỔNG QUAN VỀ .NET FRAMEWORK

.NET Framework là hạ tầng cơ bản được chuẩn hoá, độc lập ngôn ngữ lập trình, cho phép người lập trình xây dựng, tích hợp, biên dịch, triển khai, chạy các dịch vụ Web, XML, tiện ích hay thực thi chương trình đa cấu trúc (phát triển bằng các ngôn ngữ lập trình hỗ trợ .NET) trên hệ điều hành có cài đặt .NET Framework.

I.1. Thành phần .NET Framework

.NET Framework bao gồm 2 phần chính là Common Language Runtime (CLR) và .NET Framework Class Library (FCL).

CLR là thành phần chính của .NET Framework, quản lý mã (code) có thể thực thi của chương trình, quản lý các tiến trình, quản lý tiểu trình

(Threading), quản lý bộ nhớ, cung cấp dịch vụ để biên dịch, tích hợp và tác vụ truy cập từ xa (Remoting).

FCL bao gồm tất cả các dịch vụ như giao tiếp người sử dụng, điều khiển, truy cập dữ liệu, XML, Threading, bảo mật.

Tóm lại, CLR được xem như máy ảo .NET (.NET Virtual Machine), nó có thể kiểm soát, nạp và thực thi chương trình .NET.

Trong khi đó, FCL cung cấp các lớp, giao tiếp và các kiểu giá trị, phương thức truy cập và chức năng chính của hệ thống như: Microsoft.Csharp, Microsoft.Jscript, Microsoft.VisualBasic, Microsoft.Vsa, Microsoft.Win32, System (cùng với các không gian tên con của không gian tên System).

Microsoft.Csharp : cung cấp các lớp hỗ trợ biên dịch và phát sinh mã khi sử dụng ngôn ngữ lập trình C#.

Microsoft.Jscript : cung cấp các lớp hỗ trợ biên dịch và phát sinh mã khi sử dụng ngôn ngữ lập trình J#.

Microsoft.VisualBasic : cung cấp các lớp hỗ trợ biên dịch và phát sinh mã khi sử dụng ngôn ngữ lập trình VisualBasic.

Microsoft.Vsa : cung cấp các gia tiếp cho phép tích hợp với các kịch bản của .NET Framework vào ứng dụng khi biên dịch hay thực thi.

Microsoft.Win32: cung cấp hai lớp giao tiếp trực tiếp với tài nguyên của hệ điều hành và System Registry.

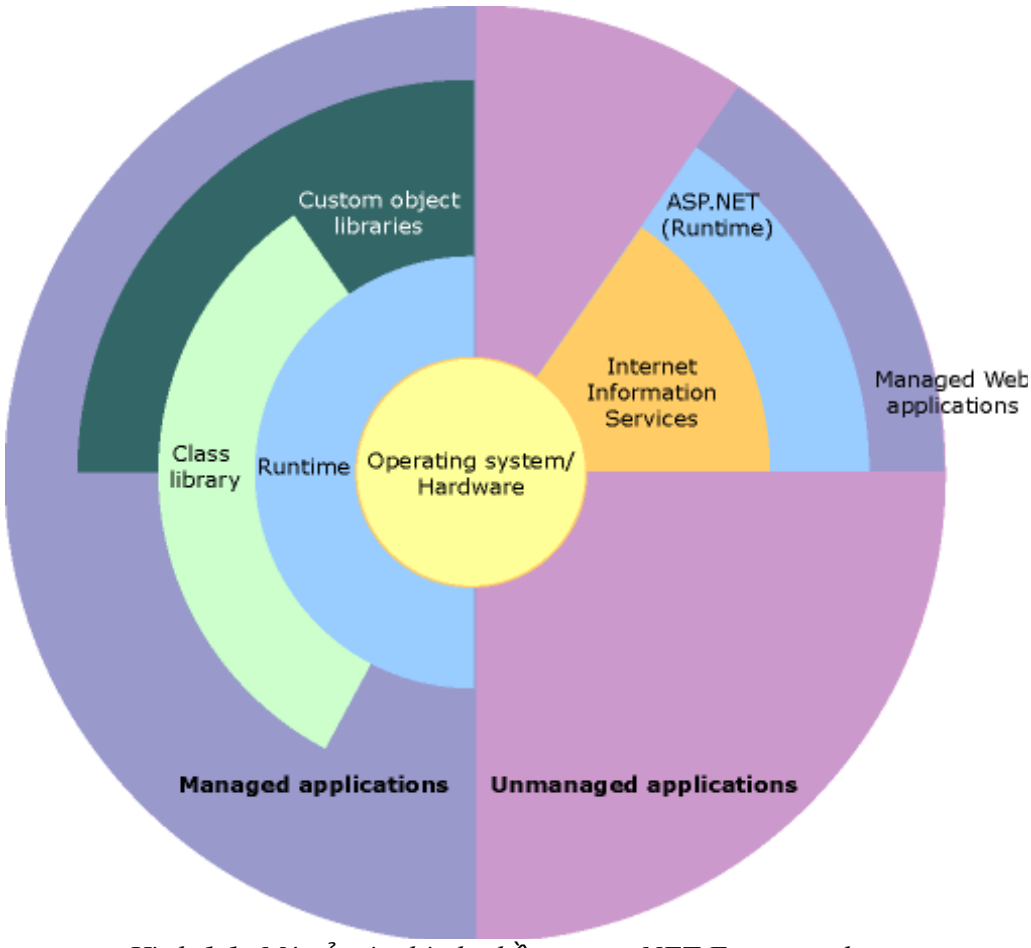
System: bao gồm các lớp cơ sở dùng để định nghĩa giá trị, tham chiếu, biên cố, giao tiếp, thuộc tính và kiểm soát ngoại lệ. Ngoài ra, một số lớp khác cung cấp các dịch vụ chuyển đổi kiểu dữ liệu, tham số, tính toán, xử lý và truy cập từ xa.

Trong đó, Code bao gồm hai loại :

Manage Code: bao gồm những chương trình được tạo ra từ các ngôn ngữ lập trình có hỗ trợ .NET, chẳng hạn, sử dụng ngôn ngữ lập trình C# để phát triển chương trình ứng dụng A, sau đó biên dịch chúng ra tập tin thi hành (.EXE), tập tin .EXE này đươc gọi là Manage Code trong môi trường .NET.

Unmanage Code : là những chương trình được tạo ra từ các ngôn ngữ lập trình ngoài .NET. Ví dụ, sử dụng ngôn ngữ lập trình Visual Basic 6.0 để khai báo lớp (Class) có tên là B, rồi biên dịch chúng ra tập tin thư viên (.DLL), tập tin .DLL được gọi là Unmanage Code khi tham chiếu chúng trong môi trường .NET

Như vậy, .NET Framework còn gọi là môi trường tương tác với hệ điều hành cho các ứng dụng và được minh hoạ như hình sau :



Hình 1.1: Mô tả các thành phần trong .NET Framework

I.2. Những đặc điểm chính của .NET Framework

.NET Framework bao gồm các đặc điểm chính như : CRL, FCL, Colmomn Type System (kiểu dư liệu thông dụng, Metadata and Selff Descring Component phần chính Siêu dữ liệu và tự đặc tả thành phần). Cross-Language Interopenrability (trao đổi và sử dụng), Assemblies (đơn vị phân phối), Application Domains (miền ứng dụng) và Runtime Host (trung tâm thi hành)

CLR : CLR là môi trường thi hành, nơi cung cấp dịch vụ để thực thi, quản lý bộ nhớ, tiểu trình cho các ứng dụng hỗ trợ bởi .NET

- Quản lý quá trình thực thi: để quản lý quá trình thực thi của trình, CLR thực hiện qua các bước sau: chọn chương trình biên dịch tương ứng với ngôn ngữ lập trình, biên dịch ứng dụng sang tập tin MSIL (trình bày chi tiết trong phần biên dịch và thực thi ứng dụng), biên dịch từ mã định dạng MSIL sang mã máy bằng trình JIT (Just-In-Time) rối sau đó CLR cung cấp cơ sở hạ tầng để thi hành chương trình.
- Quản lý bộ nhớ: tự quản lý bộ nhớ là một trong những dịch vụ mà CLR cung cấp trong quá trình thực thi chương trình. Trình thu gom (Garbage Collector) quản lý bộ nhớ đã cấp cho một tiến trình rồi sau đó tự động thu lại khi chương trình kết thúc (chúng ta sẽ tìm hiểu chi tiết về Garbage Collector trong cuốn sách " lập trình hướng đối tượng" sắp phát hành).

FCL: Bao gồm các thư viện lớp cơ sở cho phép bạn sử dụng để thực hiện mọi tác vụ liên quan đến giao diện, Internet, cơ sở dữ liệu, hệ điều hành,…

Common Type System (CTS): CTS đưa ra các quy tắc cho phép bạn khai báo, sử dụng và quản lý kiểu dữ liệu trong quá trình thi hành. Ngoài ra, CTS còn cung cấp các tiêu chuẩn cho phép phát hành tương tác giữa các ngôn ngữ lập trình với nhau. Tóm lại, CTS thực hiện các chức năng chính sau:

- Thiết lập khung cho phép tương tác giữa các ngôn ngữ, mã an toàn(safe code), tối ưu hóa xử lý.
- Cung cấp mô hình hướng đối tượng nhằm hỗ trợ quá trình cài đặt đa ngôn ngữ trong ứng dụng.
- Định nghĩa các quy tắc mà ngôn ngữ lập trình phải tuân theo và hỗ trợ tính chuyển đổi và bảo đảm đối tượng được tạo ra từ ngôn ngữ này có thể tương tác với ngôn ngữ khác.

Metadata and Self-Describing Components (MSDC): trong những phiên bản trước đây, ứng dụng được tạo ra bởi một ngôn ngữ lập trình nào đó được biên dịch ra tập tin .EXE hay .DLL và khó khăn khi sử dụng chúng với một ứng dụng được viết trong một ngôn ngữ lập trình khác. Ví dụ COM là một điển hình. Tuy nhiên, .NET framework cung cấp giải pháp chuyển đổi cho phép khai báo thông tin cho mọi module và Assembly (có thể là .EXE hay .DLL). Những thông tin này được gọi là siêu dữ liệu và sự mô tả.

Cross Language Interoperability (CLI): CLR là hỗ trợ tiến trình trao đổi và sử dụng giữa các ngôn ngữ với nhau.Tuy nhiên, hỗ trợ này không bảo đảm mã do bạn viết có thể dùng được bởi lập trình viên sử dụng ngôn ngữ lập trình khác.

Assemblies: là tập hợp các kiểu dữ liệu và tài nguyên được đóng gói dạng từng đơn vị chức năng. Ngoài ra, assemblies chính là các đơn vị chủ yếu dùng để triển khai, điều khiển phiên bản, thành phần sử dụng lại, chẳng hạn như các tập tin .EXE hay .DLL.

Applicatin Domains: miền ứng dụng cho CLR quản lý nhằm cách ly nhiều ứng dụng đang thi hành trên cùng một máy tính cụ thể:

- Mỗi ứng dụng sẽ được nạp vào tiến trình(Process) tách biệt mà không ảnh hưởng đến ứng dụng khác. Với kỹ thuật kiểu mã an toàn Application Domains bảo đảm đoạn mã đang chạy trong miền ứng dụng độc lập với tiến trình của ứng dụng khác trên cùng một máy.
- Khi tạm dừng từng thành phần thì sẽ không dừng toàn bộ tiến trình. Đối với trường hợp này Application Domains cho phép bạn loại bỏ đoạn mã đang chạy trong ứng dụng đơn.

- Application Domains cho phép bạn cấu hình, định vị, cấp quyền hay hạn chế quyền sử dụng tài nguyên đang thi hành.
- Ngoài ra, sự cách ly này cho phép CLR ngăn cấm truy cậptruwcj tiếp giữa các đối tượng của những ứng dụng khác nhau.

Runtime Hosts: là trung tâm thi hành cho phép nạp ứng dụng vào tiến trình, CLR hỗ trợ cho phép nhiều loại ứng dụng khác nhau cùng chạy trong một tiến trình.

- Mỗi loại ứng dụng thì cần đoạn mã để khởi động được gọi là Runtime hosts.
- Runtime Hosts nạp kênh thi hành vào tiến trình và tạo ra Application Domains rồi thi hành ứng dụng vào trong miền ứng dụng đó.

II. GIỚI THIỆU VISUAL STUDIO .NET 2008

Microsoft Visual Studio là tập công cụ hoàn chỉnh dùng để xây dựng ứng dụng Web (ASP.NET Web Applications), dịch vụ XML, ứng dụng để bàn (Desktop application), ứng dụng màn hình với bàn phím (Console Applications) và ứng dụng trên điện thoại di động (Mobile Applications).

Các ngôn ngữ lập trình dùng Microsoft Visual studio để phát triển ứng dụng là Visual basic, Visual C++, Visual C# và Visual J#. Cả 4 ngôn ngữ lập trình chính trên đều sử dụng chung một IDE (Integrated Development Environment), nơi cho phép chúng ta chia sẻ các tiện ích và công cụ nhằm tạo nên giải pháp tích hợp.

Nếu đã làm việc với phiên bản Visual Studio 6.0, mỗi ngôn ngữ lập trình (C++, Visual Basic, J++, Fox Pro) sẽ có riêng một IDE tương ứng. Ngoài ra, để phát triển ứng dụng ASP, ta phải sử dụng Visual Studio InterDev.

II.1. Phiên bản Visual Studio .NET 2008

Visual Studio .NET 2008 có 5 phiên bản chính thức là: Express Products (Visual Studio Express Edition), Visual Studio Standard Edition, *Visual Studio Professional Edition*, Visual Studio Tools for Office và Visual Studio Team System.

II.1.1 Visual Studio Express Edition

Đây là phiên bản đơn giản, dễ học, dễ sử dụng dùng cho những người tự học, chưa có kinh nghiệm lập trình hoặc các bạn sinh viên bước đầu làm quen với *Visual Studio .NET 2008.*

Nếu sử dụng phiên bản này, bạn cần bộ nhớ khoảng 35MB đến &70MB, miễn phí 1 năm. Hơn thế nữa, sẽ có phiên bản *Microsoft SQL Server Express* miễn phí hoàn toàn, cung cấp các chức năng chính dùng để làm việc với cơ sở dữ liệu *Microsoft SQL Server Express* 2008 từ cửa sổ *Visual Studio .NET 2008*.

Tương tự như vậy *Visual Studio Express Edition* cung cấp 4 phiên bản *Visual Basic 2008 Express Edition, Visual C# 2008 Express Edition, Visual C++ 2008 Express Edition* và *Visual J# 2008 Express Edition* ứng với 4 ngôn ngữ chính là: *Visual Basic, C#, C++* và *J#.*

Trong trường hợp phát triển ứng dụng Web, có thể sử dụng *Visual Web Developer 2008 Express* để nhanh chóng tạo ra các trang *ASP .NET* bằng các công cụ trực quan.

II.1.2 Visual Studio Standard Edition

Trong khi phiên bản *Visual Studio Express* có tính năng đơn giản, dễ sử dụng và miễn phí 1 năm thì phiên bản *Visual Studio Standard Edition* được thiết kế toàn diện hơn với chi phí vừa phải.

Được sử dụng cho các lập trình viên chuyên nghiệp làm việc đơn lẽ với các ứng dụng chạy nhanh, tối ưu, ứng dụng đa tầng trên nền *Windows,* các ứng dụng Web hay ứng dụng chạy trên thiết bị cầm tay.

Với những đặc điểm như vậy, phiên bản này là công cụ hỗ trợ cả bốn ngôn ngữ lập trình, thường dùng cho các nhà lập trình viên làm việc ngoài giờ hay công việc không thường xuyên.

Tương tự như các phiên bản khác của bộ *Visual Studio .NET 2008, Visual Studio Standard Edition* cung cấp giao diện trực quan cho phép bạn thiết lập giao diện cho các loại ứng dụng bằng việc kéo và thả (*drag and drop*), xây dựng và triển khai ứng dụng theo mô hình khách-chủ (client-server), các công cụ để thiết kế cơ sở dữ liệu.

II.1.3 Visual Studio Professional Edition

Nếu như *Visual Studio Standard Edition* dùng cho cá nhân phát triển ứng dụng thì *Visual Studio Professional Edition* bao gồm các công cụ giao diện trực quan cho phép bạn thiết lập giao diện cho các loại ứng dụng bằng việc kéo và thả *(drag and drop).*

Visual Studio Professional Edition có thể sử dụng cho cá nhân hay nhóm lập trình nhỏ khi xây dựng và triển khai ứng dụng theo mô hình khách chủ (*client-server*), thiết kế cơ sở dữ liệu, ứng dụng đa tầng trên nền Windows, ứng dụng Web hay ứng dụng chạy trên thiết bị cầm tay.

II.1.4 Visual Studio Team System

Đây là công cụ theo hướng mở rộng và tích hợp, dùng cho các công ty phát triển phần mềm hay những nhóm lập trình viên làm việc xuyên quốc gia.

Sử dụng phiên bản *Visual Studio Team System* cho phép nhóm lập trình có thể giảm độ phức tạp, tăng tính giao tiếp và hợp tác trong quá trình phát triển phần mềm.

Visual Studio Team System còn là bộ khung (Microsoft Solutions Framework) gọi là MSF. MSF cung cấp một tập được tối ưu hóa và tính uyển chuyển cùng các quy tắc đã được tích hợp áp dụng cho từng giai đoạn khi phát triển và triển khai một phần mềm.

Tùy vào từng công đoạn của quá trình xây dựng và triển khai phần mềm, có thể sử dụng 5 bộ công cụ thuộc *Visual Studio Team System như sau: Visual Studio 2008 Team Suite,Visual Studio 2008 Team Edition for Software Architects, Visual Studio 2008 Team Edition for Software Developers, Visual Studio 2008 Team Edition for Software Testers, Visual Studio 2008 Team Foundation Server và Visual Studio 2008 Team Test Load Agent.*

Bộ Visual Studio 2008 Team Suite

Visual Studio 2008 Team Suite là bộ công cụ tích hợp, hiệu suất cao cho phép nhóm lập trình viên giao tiếp và kết hợp tốt trong quá trình phát triển phần mềm. Với tổ chức của *Visual Studio 2008 Team Suite,* bạn có thể dự đoán trước được chất lượng và tổ chức trong quá trình phát triển ứng dụng.

Bộ Visual Studio 2008 Team Edition for Software Architects:

Team Edition for Software Architects cung cấp các công cụ trực quan dùng để xây dựng một giải pháp dùng cho việc thiết kế ứng dụng hay triển khai chúng nhanh và hiệu quả hơn.

Bộ Visual Studio 2008 Team Edition for Software Developers:

Team Edition for Software Developers cung cấp bộ công cụ phát triển ứng dụng cho phép nhóm lập trình tương tác, phối hợp và cùng chia sẻ trong chu trình phát triển ứng dụng.

Bộ Visual Studio 2008 Team Edition for Software Testers:

Team Edition for Software Testers giới thiệu tập các công cụ dùng để kiểm tra, đánh giá sản phẩm phần mềm được tích hợp với môi trường Visual Studio. Bộ công cụ này cho phép những người kiểm tra chất lượng sản phẩm phần mềm thông báo đến tác giả hay nhà quản lý những công việc liên quan.

Bộ Visual Studio 2008 Team Foundation Server:

Team Foundation Server là những gì mạnh nhất của quá trình hợp tác trong *Visual Studio Team System.* Khi kết hợp với *Visual Studio Team System, Team Foundation Server* cho phép bạn quản lý và theo dõi quá trình thực hiện của dự án.

Bộ Visual Studio 2008 Team Test Load Agent:

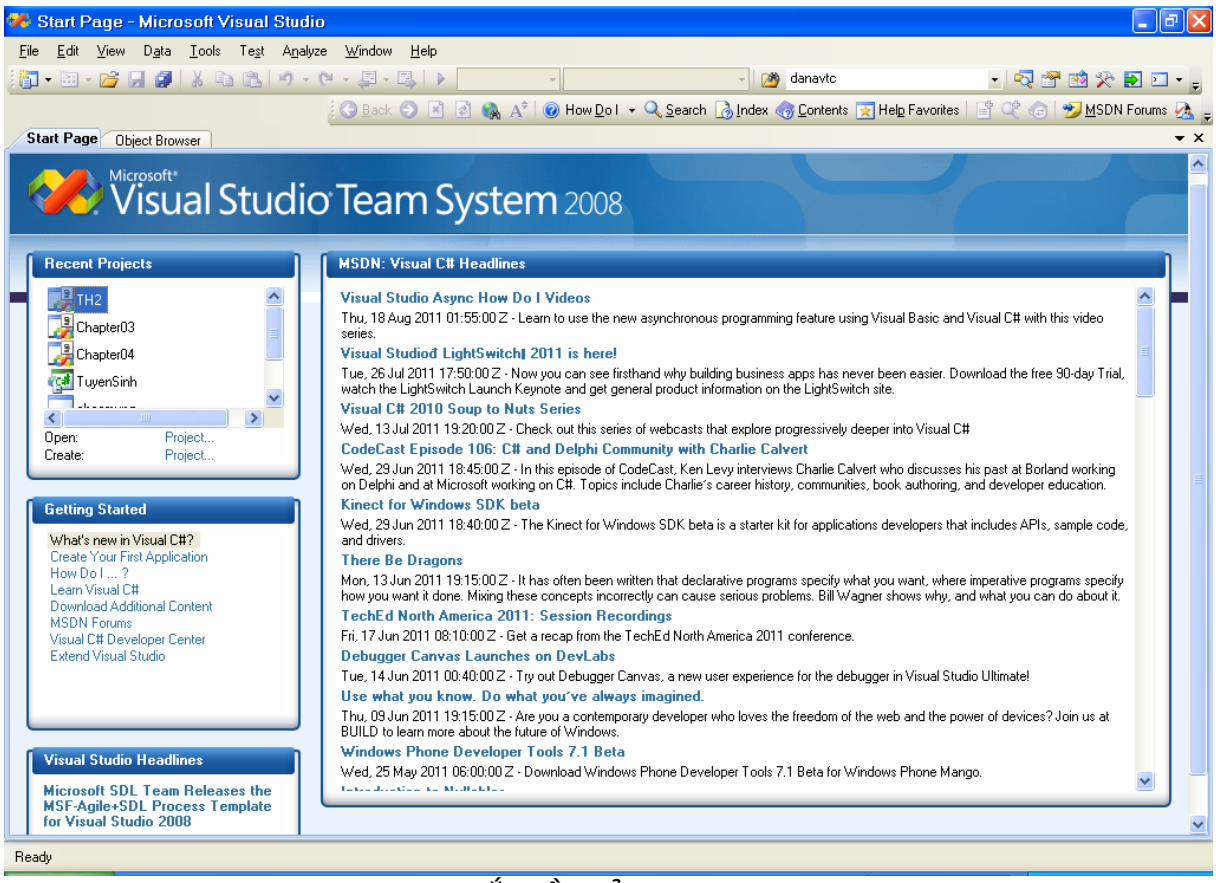
Team Test Load Agent tạo ra tiến trình kiểm tra bổ sung được sử dụng với *Visual Studio 2008 Team Edition for Software Testers,* cho phép bạn tổ chức và mô phỏng một hay nhiều người sử dụng để kiểm tra chất lượng sử dụng của ứng dụng.

II.2 Làm việc với Visual Studio .NET 2008

Từ khi Visual studio .NET ra đời, nó là một IDE dùng chung duy nhất cho mọi ngôn ngữ lập trình và các loại ứng dụng được được tích hợp. Như vậy, ứng dụng Web Forms (ASP.NET) được xem như một phần của ngôn ngữ lập trình, bạn có thể sử dụng chung IDE với ứng dụng Windows Forms.

Chẳng hạn, bạn có thể mở dự án (Project) bằng ngôn ngữ lập trình Visual Basic.NET, rồi mở tiếp một *Project* bằng ngôn ngữ lập trình C# trong cùng một Solution.

Ngoài ra, Visual Studio.NET 2008 có sự thay đổi lớn so với Visual Studio.NET 2003 là môi trường lập trình, định dạng mã, cơ chế gỡ lỗi, xây dựng, kiểm tra và triển khai ứng dụng, tự động hóa và trợ giúp người sử dụng. Ví dụ, trang bắt đầu của *Visual Studio.NET 2008 IDE* như hình 1-2

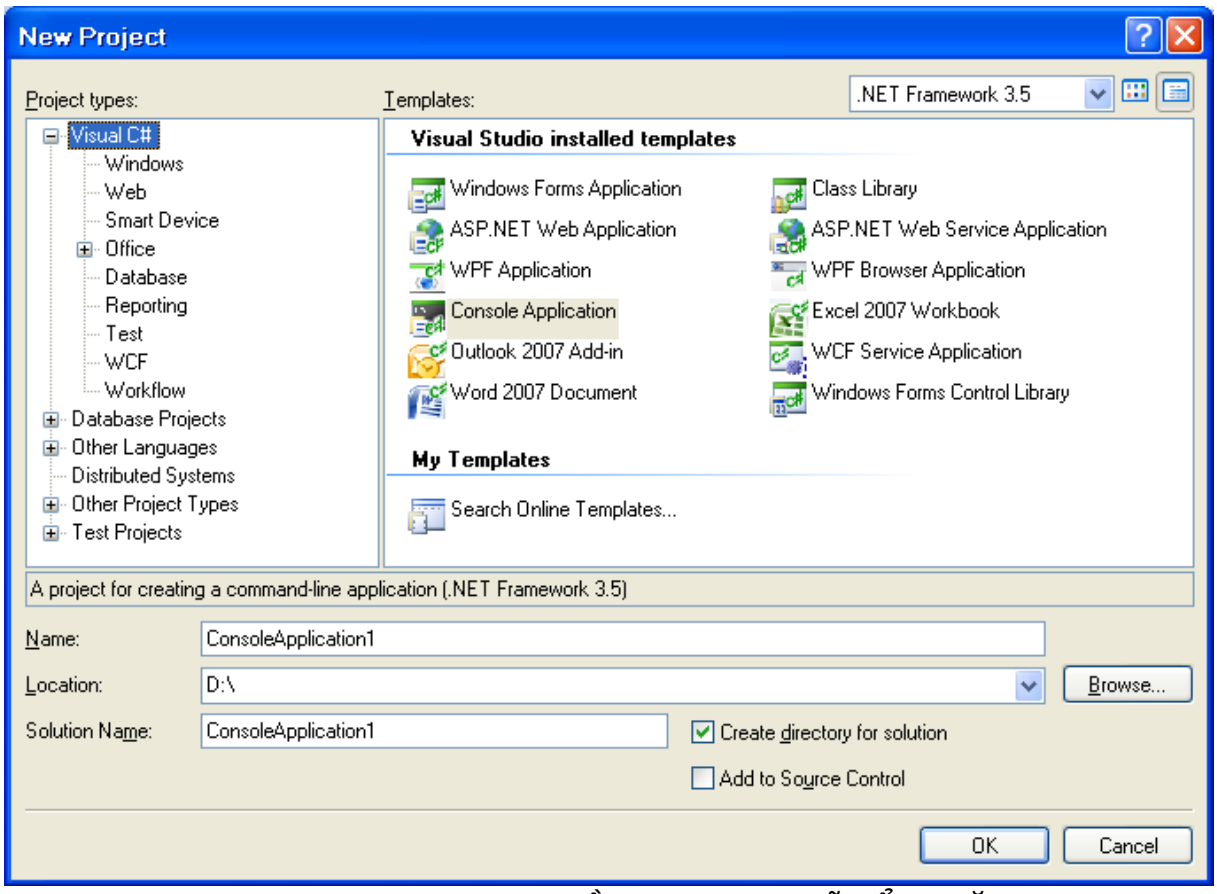


Hình 1.2 : Trang bắt đầu của *Visual Studio.NET 2008*

Sau khi cài đặt thành công Visual Studio.NET 2008, lần đầu tiên sử dụng Visual Studio.NET 2008 IDE, một cửa sổ xuất hiện yêu cầu chọn ngôn ngữ lập trình mặc định. Chẳng hạn, trong trường hợp này chúng ta chọn ngôn ngữ lập trình C# bằng cách di chuyển đến Visual C# Development Setting và nhấn mạnh Start Visual Studio.

Lưu ý, chúng ta sẽ tìm hiểu chi tiết về những công cụ, cửa sổ, cách cấu hình IDE để làm việc với ngôn ngữ lập trình C# trong những bài kế tiếp.

Sau khi chọn ngôn ngữ lập trình C# là ngôn ngữ mặc định, mỗi khi tạo mới *Project* hay *Solution, ngôn ngữ này nằm đầu tiên trong ngăn Project types như hình 1-3*, 3 ngôn ngữ lập trình còn lại là Visual basic, C++ và J# sẽ xuất hiện bên dưới phần *Other Language.*



Hình 1.3 : Màn hình yêu cầu chọn ngôn ngữ để cài đặt

Ngăn bên phải là danh sách các loại ứng dụng Windows ,bao gồm các loại như: Windows Application, Console Application, Class Library, Windows Service, Crystal Reports Application,…

Trong trường hợp muốn xây dựng ứng dụng ASP.NET, bạn có thể chọn vào trong phần tạo mới, khi đó cửa sổ sẽ xuất hiện như hình 1-5

Tương tụ như trường hợp ứng dụng vWindows, ứng dụng vWebsite bao gồm các loại như: ASP.NET vWebsite, ASP.NET vWeb Service, Srystal Reports vWebsite.

Trên thực đơn (menu) của Visual studio .NET 2008, *menu* có tên là *Community* bao gồm các *menu* con như: *Ask a question, Check Question Status, Send Feedback* nhằm hỗ trợ cho bạn tìm kiếm, gởi và kiểm tra câu hỏi hay góp ý kiến về công ty *Microsoft.*

Ngoài ra, trên *menu* này còn có các *menu* con khác, chúng cho phép bạn trỏ đến địa chỉ *internet* chứa tài nguyên hay những thông tin cập nhật về Visual studio .NET 2008 nhằm hỗ trợ tối đa cho cộng đồng lập trình .NET.

Chẳng hạn, bạn chọn vào *menu* có tên *Developer center,* cửa sổ trình duyệt xuất hiện.

III. CÁC LOẠI ỨNG DỤNG DÙNG C#

Microsoft Visual C# 2008 (C sharp) là ngôn ngữ lập trình thiết kế dùng để phát triển ứng dụng chạy trên *.NET Framework. C#* còn là ngôn ngữ lập trình đơn giản, mạnh, kiểu an toàn (type-safe) và hướng đối tượng (*object-oriented).*

Với nhiều đặc điểm mới,*C#* cho phép bạn xây dựng ứng dụng nhanh chóng nhưng vẫn giữ lại được sự diễn cảm và tao nhã của ngôn ngữ lập trình truyền thống C.

Mặc dù mọi ngôn ngữ lập trình trong bộ *.NET* đều sử dụng chung *.NET Framework,* nhưng mỗi ngôn ngữ vẫn có tính đặc thù riêng của nó. Sử dụng *C#* là một lựa chọn tối ưu khi bạn xây dựng loại ứng dụng như: quản lý, thương mại điện tử, ứng dụng tích hợp hệ thống,thư viện,ứng dụng dùng cho máy PDA hay điện thoại di động,…

III.1. Ứng dụng Windows Form

Khi xây dựng ứng dụng với giao diện người dùng chạy trên máy để bàn có cài đặt *.NET Framework,* bạn chọn *vWindows* trong phần *Project Types* rồi tiếp tục chọn vào *vWindows Application* trong phần *Templates*

III.2. Ứng dụng màn hình và bàn phím

Nếu ứng dụng với giao diện người dùng là bàn phím và màn hình chạy trên máy để bàn, bạn có thể chọn loại ứng dụng là Console Application trong phần Templates .Với ứng dụng loại này, người sử dụng thao tác bằng màn hình C onsole.

Tuy nhiên, bằng cách sử dụng các không gian tên của ứng dụng vWindows *Forms,* bạn cũng có thể tạo ra ứng dụng giao diện đồ họa bằng ứng dụng *Console Application.*

III.3. Dịch vụ hệ điều hành

Trong trường hợp ứng dụng chạy thường trú trong bộ nhớ,bạn có thể chọn loại ứng dụng là *vWindows Service* trong phần *Templates.*

Khi chọn ứng dụng này, bạn tạo ra tập tin .EXE và cài đặt chúng vàodịch vụ của hệ điều hành (Service), bạn có thể *Start, Stop hay Pause* và *Continue* như những dịch vụ của hệ điều hành đang tồn tại.Chú ý, ứng dụng dịch vụ hệ điều hành thì không cần giao diện, thay vào đó bạn sử dụng tiện ích *Service* của hệ điều hành.

III.4. Thư viện

Khi cần xây dựng thư viện dùng chung hay *COM+(*triển khai tầng *Business Logic),* bạn chọn vào *Class Library,* sau khi kết thúc khai báo,nếu biên dịch thành công thì ứng dụng này sẽ tạo ra tập tin .DLL.

Ví dụ, bạn muốn xây dựng thư viện bao gồm các lớp làm việc với cơ sở dữ liệu *SQL Server,* sau đó bạn sử dụng thư viện như những *Project* khác nhau, ứng với mục đích này bạn tạo mới *Project* loại *Class Library.*

III.5. Điều khiển do người sử dụng định nghĩa

Ngoài các điều khiển (*Control)*từ các lớp của .NET cung cấp, người sử dụng có thể kết hợp những điều khiển này thành một điều khiển tùy ý (*CustomControl)* phục vụ cho một yêu cầu cụ thể nào đó.

Đối với ứng dụng *vWindows Forms*, bạn có thể sử dụng loại *Project* là *vWindows Control Library.* Trong trường hợp làm việc với ứng dụng *ASP.NET* loại *Project* bạn dùng là *vWeb Control Library.*

Cả hai loại *Project* này đều biên dịch thành tập tin .DLL, bạn có thể thêm chúng vào công cụ (*Tool Box)* như những điều khiển của *.NET*

III.6. Ứng dụng báo cáo

Nếu có nhu cầu xây dựng ứng dụng báo cáo (*Report)* bằng *Crystal Report,* bạn chọn loại *Project* là *Crystal Report Applications.*Tuy nhiên, thông thường *Report* là một phần của ứng dụng nên bạn sử dụng *Crystal Report* như những đối tượng của *Project.*

III.7. Ứng dụng SQL Server

Để khai báo bảng dữ liệu (*Table),* bảng ảo (*View)*, thủ tục nội tại, (*Store Procedure),* hàm *(Funtion),…*bạn vào ngăn *Database* rồi chọn *Project* với loại *SQL Server Project.* Ứng dụng này cho phép bạn thiết kế cơ sở dữ liệu *SQL Server* từ *Visual Studio.NET 2008* thay vì từ trình *SQL Server Enterprise.*

Lưu ý, tương tự như trong ứng dụng *Report,* bạn có thể thêm cơ sở dữ liệu vào *Project* như một phần của ứng dụng thay vì tạo riêng*Project* về cơ sở dữ liệu.

III.8. Ứng dụng PDA và Mobile

Nếu bạn muốn xây dựng ứng dụng *.NET* cho thiết bị cầm tay như điện thoại di động (*Mobile)* hay máy kỹ thuật số hệ thống cá nhân (*PDA)* thì chọn vào *Smart Device.*

III.9. Ứng dụng đóng gói và triển khai

Sau khi kết thúc công đoạn xây dựng ứng dụng, bạn có thể đóng goi ứng dụng đó và triển khai trên máy khác. Để đóng gói ứng dụng, bạn vào ngăn *Other Project Types* rồi chọn loại *Project* là *Setup and Deployment.*

III.10. Tạo một Solution

Solution được xem như một *(Container)* dùng để quản lý nhiều *Project* trên *Visual Studio.NET 2008.*Khi tạo *Project* đầu tiên chưa tồn tại *Solution,* lập tức *Solution* được tạo ta mặc định. Trong trường hợp *Solution* đã tồn tại thì chọn *Solution* để thêm *Project* vào *Solution* đó.

Ngoài ra, bạn có thể tạo mới *Solution* trước khi thêm các *Project* khác bằng cách vào *Other Project Types* rồi chọn *Visual Studio Solutions.*

IV. CẤU TRÚC CHƯƠNG TRÌNH C#

IV.1. Cấu trúc chương trình

- ***Cấu trúc chương trình theo Windows Application Form***

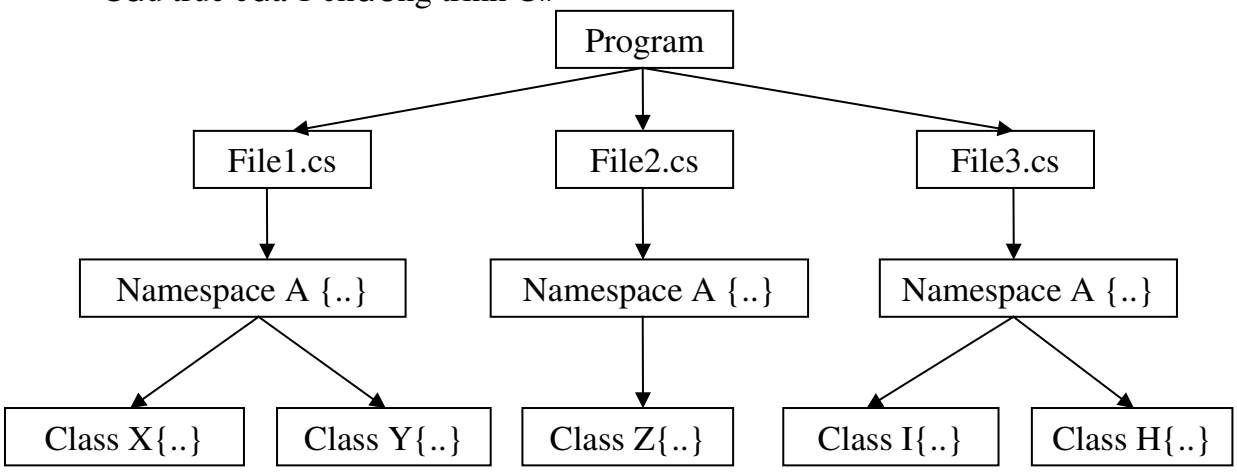
```
//Vùng bắt đầu khai báo sử dụng không gian tên
using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows.Forms;
//Vùng bắt đầu khai báo sử dụng không gian tên
//Khai báo không gian tên của ứng dụng
namespace TH2
{
        //Vùng bắt đầu khai báo tên các Class
   static class Program
   {
       //Vùng bắt đầu khai báo tên các phương thức trong lớp
      static void Main()
```

```
        {
                //Vùng khai báo lệnh
        }
    }
}
```

- ***Cấu trúc chương trình theo Console Command***

```
using System
Namespace MyNameSpace
{
 class HelloWorld
 {
  //Điểm bắt đầu của ứng dụng theo kiểu C
  static void Main(){
   Main(System.Environment.GetCommandLineArgs());
  }
  static void Main(string[] args){
   System.Console.WriteLine("Hello World")
  }
 }
}
```

- Cấu trúc của 1 chương trình C#



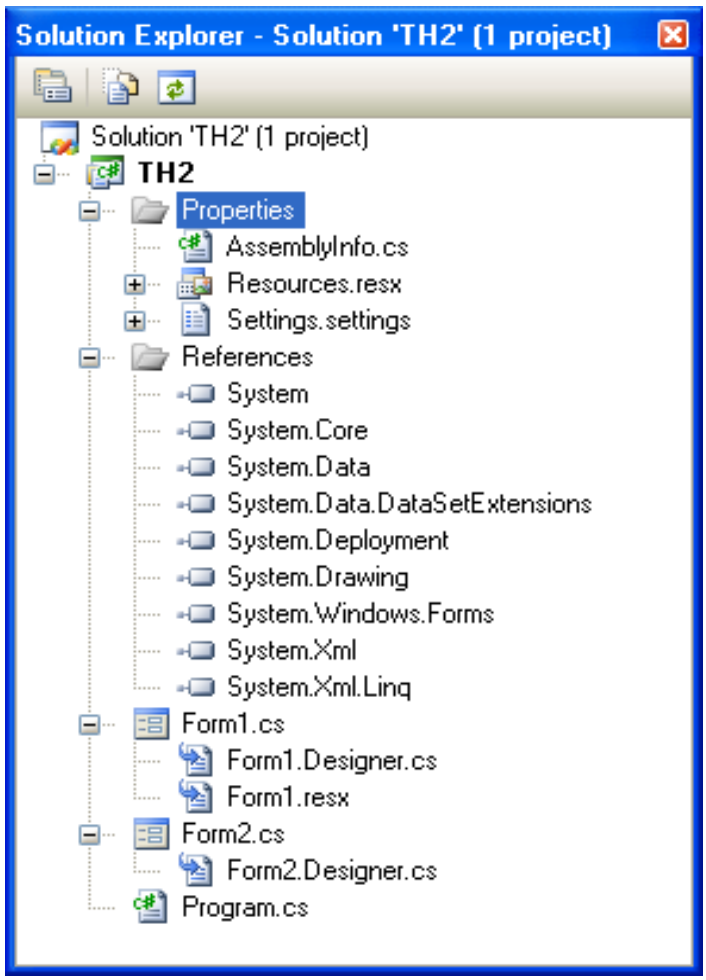
IV.2 Tổ chức cây Project

IV.2.1 Nút Properties

IV.2.2 Nút References

IV.2.3 Nút đối tượng có giao tiếp

IV.2.4 Nút đối tượng không có giao tiếp



V. CẤU TRÚC THƯ MỤC CỦA CHƯƠNG TRÌNH C# 2008

- Các File của 1 chương trình C#

G24

bin		File Folder
Debug		File Folder
obj		File Folder
Properties		File Folder
publish		File Folder
Resources		File Folder
Service References		File Folder
114.ico	1 KB	NeroPhotoSnapViewer.Files9.ico
AboutBox.cs	4 KB	Visual C# Source file
AboutBox.Designer.cs	11 KB	Visual C# Source file
AboutBox.resx	51 KB	.NET Managed Resources File
Backup	4,691 KB	File
Backup_Data	2,468 KB	File
Chung.cs	1 KB	Visual C# Source file
DM_Tinh.cs	6 KB	Visual C# Source file
DM_Tinh.Designer.cs	18 KB	Visual C# Source file
DM_Tinh.resx	7 KB	.NET Managed Resources File
FrmCapNhat.cs	20 KB	Visual C# Source file
FrmCapNhat.Designer.cs	52 KB	Visual C# Source file
FrmCapNhat.resx	13 KB	.NET Managed Resources File
FrmDM_Huyen.cs	7 KB	Visual C# Source file
FrmDM_Huyen.Designer.cs	16 KB	Visual C# Source file
FrmDM_Huyen.resx	7 KB	.NET Managed Resources File
FrmDM_Nghe.cs	6 KB	Visual C# Source file
FrmDM_Nghe.Designer.cs	14 KB	Visual C# Source file
FrmDM_Nghe.resx	7 KB	.NET Managed Resources File
FrmDMDoiTuong.cs	6 KB	Visual C# Source file
FrmDMDoiTuong.Designer.cs	17 KB	Visual C# Source file

B. CÂU HỎI VÀ BÀI TẬP

Câu 1: Nêu các thành phần chính của .NET Framework.

Câu 2: Trình bày sơ đồ môi trường .NET Framework.

Câu 3: Liệt kê những đặc điểm chính của .NET Framework.

***Câu 4**:* Liệt kê các ứng dụng dùng C#.
***Câu 5**:* Trình bày cấu trúc chương trình C#.

Câu 6 : **Trình bày các ưu điểm và nhược điểm của ngôn ngữ lập trình C#.**

Làm việc theo nhóm, tra cứu trên Internet, các phương tiện khác và trình bày báo cáo (tối đa khoảng 2 trang).

Câu 7 : **Cài đặt Visual Studio 2008.**

Cài đặt Visual Studio 2008 từ đĩa DVD

Câu 8 : **III. Kỹ năng 3 : Tìm các thông tin liên quan về C# : tính năng của phần mềm, các phiên bản**

Câu 9 : Làm việc theo nhóm.

Tìm hiểu các thông tin về C# và số lượng người dùng C# hiện nay.
Các tính năng vượt trội của C# so với các ngôn ngữ khác.
Các phiên bản C# hiện nay đã được Microsoft công bố.
Kể tên một số ứng dụng đã dùng ngôn ngữ lập trình C# mà các bạn biết.
Liệt kê và phân biệt được các thành phần trong thư mục của ứng dụng.
Cách tổ chức của cây Project, khám phá và tìm hiểu các nút.

MÃ BÀI HỌC LTTQ – 02	BÀI 2 : LÀM VIỆC VỚI VISUAL STUDIO .NET 2008	Thời gian (giờ)				
		LT 3	TH 3	BT	KT	TS 6

Mục tiêu:

Sau khi học xong bài này, học viên có khả năng:

- Phân biệt được các cửa sổ thường sử dụng trong C#;
- Sử dụng được các chức năng khi làm việc với Visual Studio.Net 2008 IDE.
- Sử dụng thành thạo các của sổ trong C#;
- Thực hiện các thao tác an toàn với máy tính.

TÓM TẮT BÀI:

- Tìm hiểu các của sổ thường sử dụng trong C#.
- Sử dụng một số chức năng khi làm việc với Visual Studio .NET 2008 IDE.

Các vấn đề chính sẽ được đề cập

- Cửa sổ Solution
- Cửa sổ thuộc tính của Project
- Cửa sổ Properties
- Của sổ Options
- Hộp công cụ
- Của sổ danh sách đối tượng
- Thực đơn Refactor

NỘI DUNG :

A. LÝ THUYẾT

I. CỬA SỔ SOLUTION

Solution (còn gọi là giải pháp) được xem như một đối tượng dùng để chứa (Container) các dự án (Project). Khi làm việc với nhiều loại Project, thay vì tạo từng Project trong Visual Studio .NET 2005 thì có thể sử dụng Solution như một Container để quản lý nhiều Project.

Bằng cách này có thể làm việc với nhiều Project, các Project này có thể thuộc những ngôn ngữ lập trình khác nhau trong cùng IDE. Tuy nhiên, nếu chỉ một project tạo ra, lập tức có một Solution quản lý Project tương ứng.

Câu 1: Trình bày sự khác nhau giữa Solution và Project ?

Câu 2 : Kể tên các của sổ thường dùng trong C# 2008.

Câu 3: Trình bày cách thức thực thi một chương trình viết bằng ngôn ngữ lập trình C#.

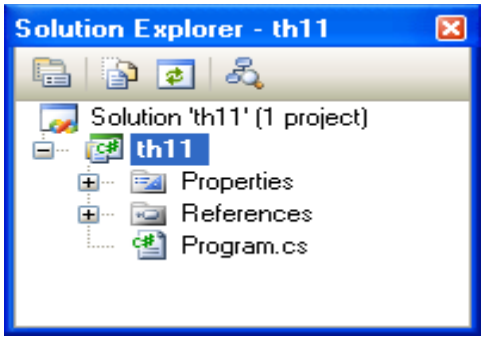
Câu 4: **Bật và tắt các của sổ trong ngôn ngữ lập trình C#.**

- Dùng menu.
- Dùng qua các vùng tương tác trên các của sổ.
- Dùng các phím tắt

Câu 5: **Khám phá các thành phần trong các của sổ của ngôn ngữ lập trình C# 2008.**

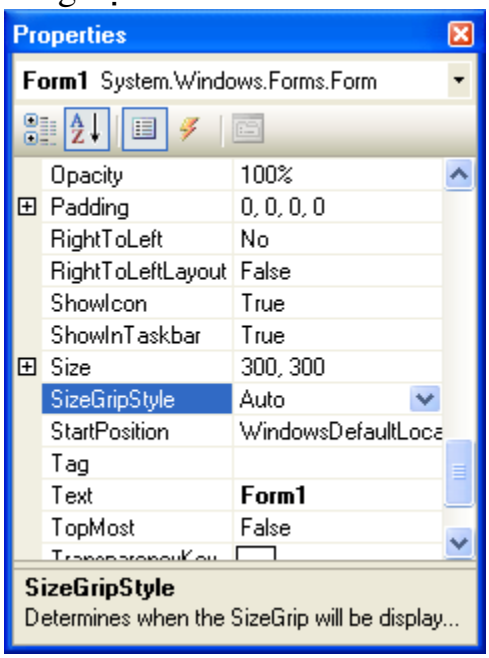
Cửa sổ Solution

- Mở cửa sổ Solution : vào View | Solution Explorer (Ctrl+W,S) hoặc dùng biểu tượng  trên thanh công cụ.



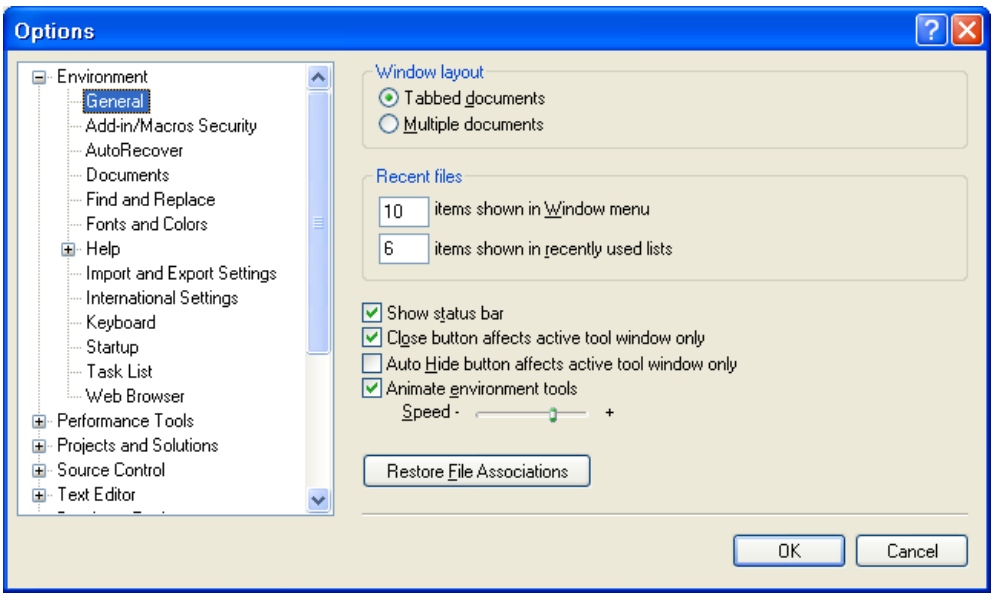
Cửa sổ thuộc tính của Project

- Mở cửa sổ Properties : vào View | Properties Windows (Ctrl+W,P) hoặc dùng biểu tượng Properties  trên thanh công cụ.



Cửa sổ Options

- Mở cửa sổ Options : vào Tool | Options…



Hộp công cụ (Toolbox)

- Mở cửa sổ Toolbox : vào View | Toolbox hoặc (Ctrl+W,X) hoặc dùng biểu tượng Toolbox  trên thanh công cụ.

Của sổ danh sách đối tượng.

Thực đơn Refactor

Câu 6: **Xây dựng và quản lý một Solution**

- Xây dựng 1 Solution đặt tên TH2
- Trong Solution TH2 có 3 Project. Đặt tên PJ1, PJ2, PJ3.
- Thiết lập Project 2 (PJ2) chạy đầu tiên.
- Xem cách tổ chức của cây Project, khám phá và tìm hiểu các nút, các thành phần trong của các sổ.

MÃ BÀI HỌC LTTQ – 03	BÀI 3 : CHƯƠNG TRÌNH C# 2008	Thời gian (giờ)				
		LT 3	TH 5	BT	KT	TS 8

Mục tiêu:

Sau khi học xong bài này, học viên có khả năng:

- Phân biệt được các không gian tên thường dùng;
- Trình bày cách biên dịch một chương trình trong C#.
- Phân biệt được các dạng phương thức Main trong mỗi chương trình C#;
- Thực hiện các thao tác an toàn với máy tính.

TÓM TẮT BÀI:

- Tìm hiểu cách biên dịch chương trình C#.
- Tìm hiểu các không gian tên thường sử dụng, các dạng của phương thức Main trong mỗi chương trình C#
- Cách định dạng kết quả trình bày trên màn hình Command Prompt.

Các vấn đề chính sẽ được đề cập

- Biên dịch và thực thi chương trình.
- Giải thích các không gian tên.
- Các dạng của phương thức Main.
- Định dạng kết quả của cửa sổ Command Prompt.
- Chú thích trong C#.
- Khai báo chỉ thị region.

NỘI DUNG :

A.LÝ THUYẾT

I. BIÊN DỊCH VÀ THỰC THI CHƯƠNG TRÌNH

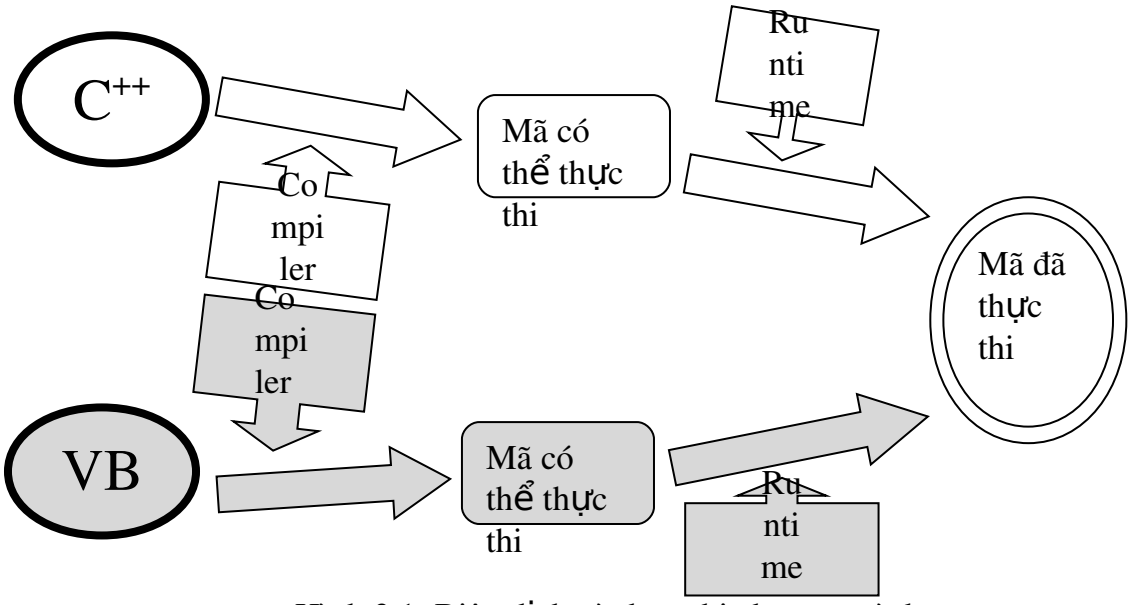
I.1 Biên dịch và thực thi của C++, Visual Basic 6.0

Nhìn lại phiên bản Visual Studio 6.0, hai ngôn ngữ lập trình thường được sử dụng là Visual C++ và Visual Basic 6.0. Mỗi ngôn ngữ sẽ có IDE, trình biên dịch (compiler), trình thi hành môi trường chạy (runtime environment) tương ứng.

Mỗi ngôn ngữ phải có trình biên dịch để biên dịch mã chương trình của chúng ra tập tin có thể thực thi (Executeable Code), rồi từ tập tin này bạn có thể chạy chương trình trên mỗi runtime tương ứng.

Khi triển khai chương trình được viết bằng một trong những ngôn ngữ lập trình thuộc bộ Visual Basic 6.0 trên máy PC, bạn cần cài đặt các thư viên tương ứng để cho phép chương trình có thể chạy trên máy đó. Tuy nhiên, vấn đề đính kèm các thư viện này được thực hiện khi đóng gói và triển khai chương trình.

Tóm lại, chúng ta thử hình dung cơ chế biên dịch và thực thi chương trình của hai ngôn ngữ lập trình chính trong bộ Visual Studio 6.0 như hình :



Hình 3.1: Biên dịch và thực thi chương trình.

I.2. Biên dịch và thực thi chương trình .NET

Với .NET thì trình biên dịch compiler của mỗi ngôn ngữ sẽ biên dịch mã nguồn ra một định dạng trung gian gọi là IL (Intermediate Language) hay MSIL (Microsoft Intermediate Language), định dạng trung gian thay thế Executable Code trong hình trên.

Bên cạnh đó, .NET không tồn tại trình Runtime cho mỗi ngôn ngữ lập trình của chúng, thay vào đó trình runtime được thay thế bởi một trình dùng chung do .NET Framwork cung cấp là CLR (Comman Language Runtime)

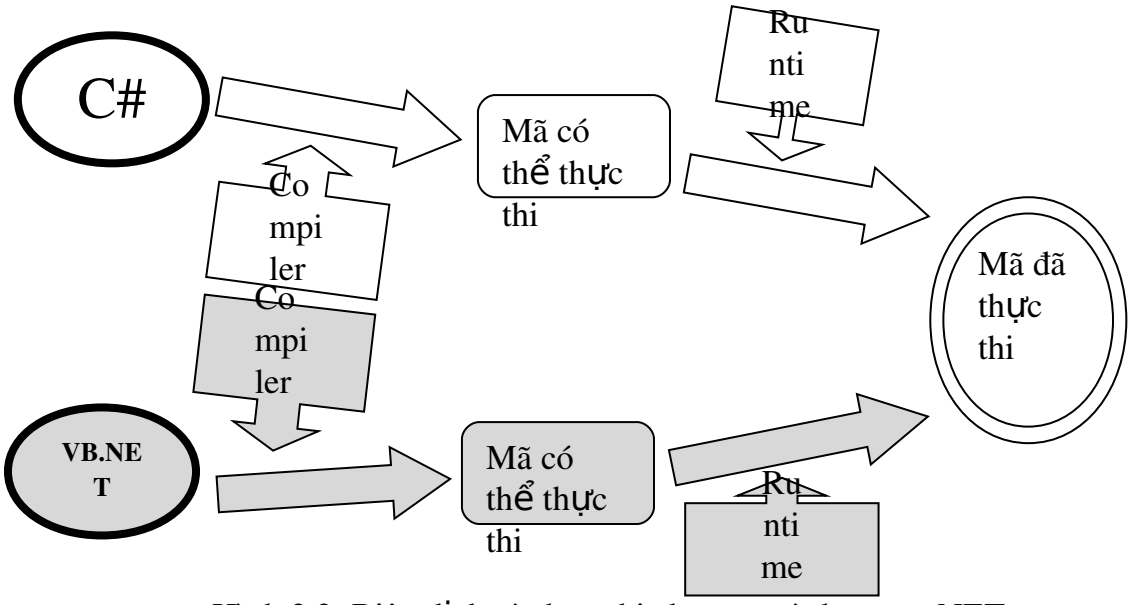
I.2.1 MSIL

MSIL hay IL là ngôn ngữ hỗ trợ thao tác giữa các thành phần , chúng gần giống với mã nhị phân và được định nghĩa như một tập lệnh và dễ dàng chuyển sang mã máy bằng CLR.

Lưu ý, chương trình .NET được biên dịch lần thứ nhất có thể chạy trên hệ điều hành bất kỳ hay CPU có hỗ trợ CLR.

1.2.2. CLR

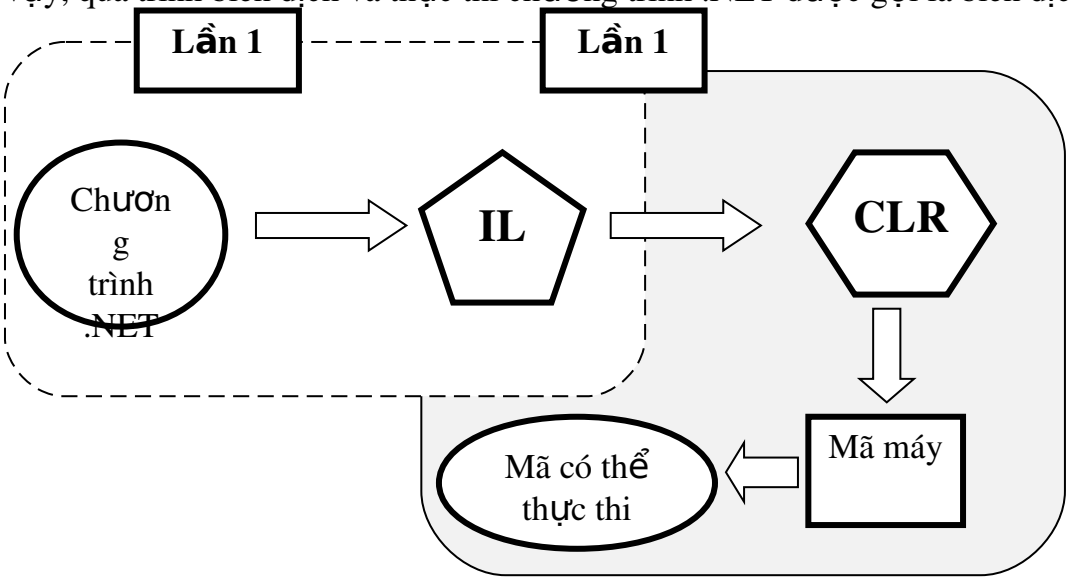
Trình CLR dùng để biên dịch tập tin định dạng MSIL ra mã máy. Tóm lại, quá trình biên dịch và thực thi chương trình trên .NET được mô tả bởi hình sau:



Hình 3.2: Biên dịch và thực thi chương trình trong .NET.

I.3. Vai trò của MSIL

Vai trò của CLR là dùng để thực thi đoạn mã với định dạng MSIL thành mã máy khi chạy chương trình bằng cách sử dụng cơ chế JIT (Just In Time). Chính vì vậy, quá trình biên dịch và thực thi chương trình .NET được gọi là biên dịch hai lần.



Hình 3.3 : Thực thi chương trình .NET

Lần thứ nhất chính là biên dịch chương trình ra tập tin với định dạng MSIL, lần thứ hai được thực hiện khi bạn thực thi những Assembly này trên môi trường .NET, chúng sẽ biên dịch thành mã máy, bạn có thể tham khảo mô phỏng này tương ứng với hình trên.

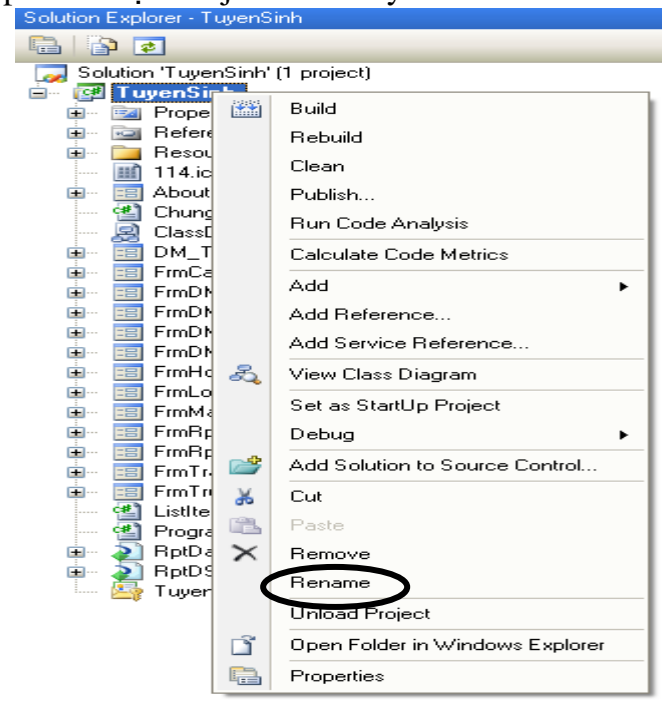
I.4. Biên dịch chương trình C#

Chúng ta có thể biên dịch chương trình C# từ Visual Studio .NET 2008 hoặc tiện ích csc.exe của .NET Framework. Nếu chương trình được xây dựng có giao diện phức tạp thì nên dùng Visual Studio .NET 2008 để biên dịch chương trình là sự lựa chọn tốt nhất.

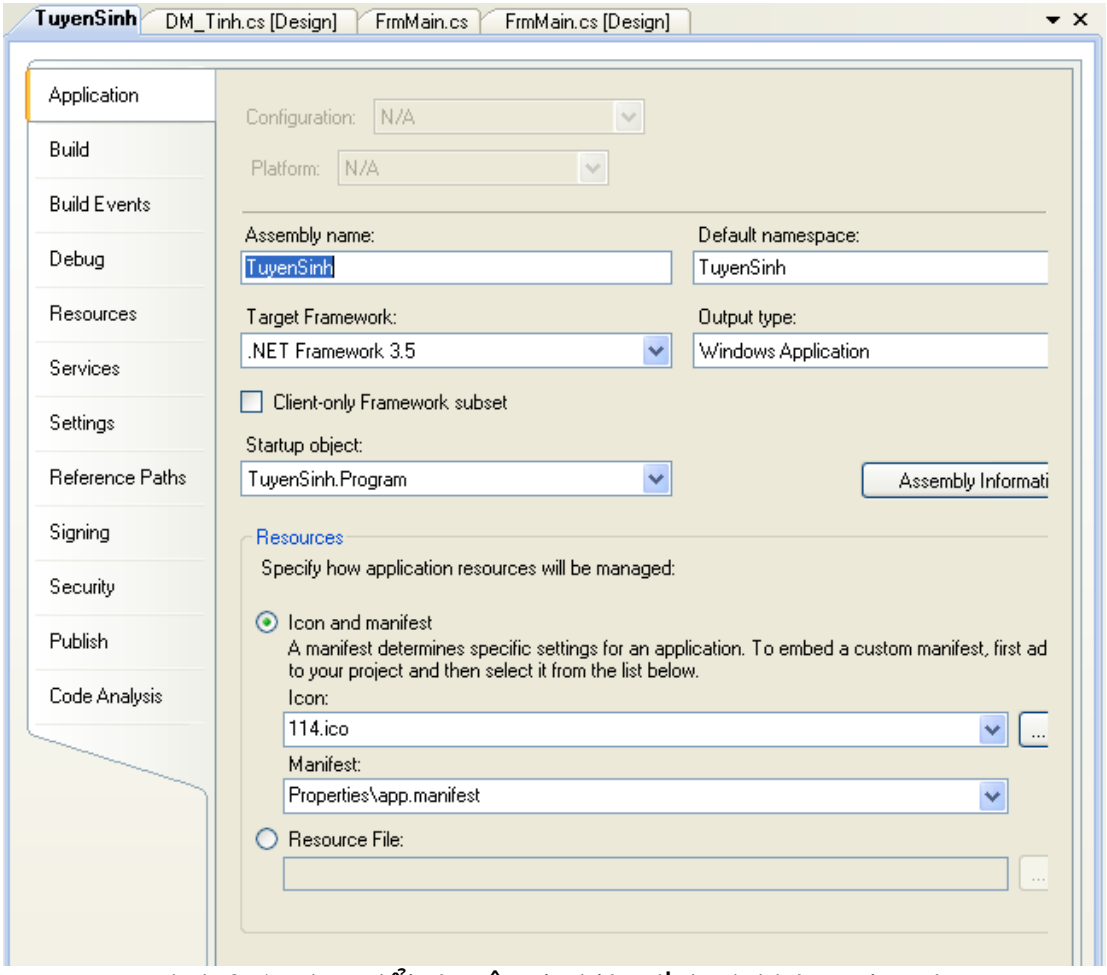
Tuy nhiên nếu chương trình đơn giản, thay vì sử dụng bản quyền cho Visual Studio .NET 2008 thì bạn có thể sử dụng tiện ích miễn phí của .NET Framework là csc.exe.

Để biên dịch chương trình C# từ Visual Studio .NET 2008, bạn chọn tên Project rồi ***R-Click*** | ***Build*** hoặc chọn vào trình đơn ***test*** | ***Run***.

Lưu ý, tên tập tin Assembly mặc định là tên của Project và tên không gian tên mặc định là tên của Project . Bạn có thể thay đổi tên tập tin Assembly và tên không gian tên của Project bằng cách chọn vào thực đơn Project | Properties hay từ cửa sổ Solution Explorer chọn Project cần thay đổi tên R-Click | Rename.



Hình 3.4: cách thay đổi các thông tin về Project



Hình 3.5: Thay đổi tên tập tin biên dịch và không gian tên

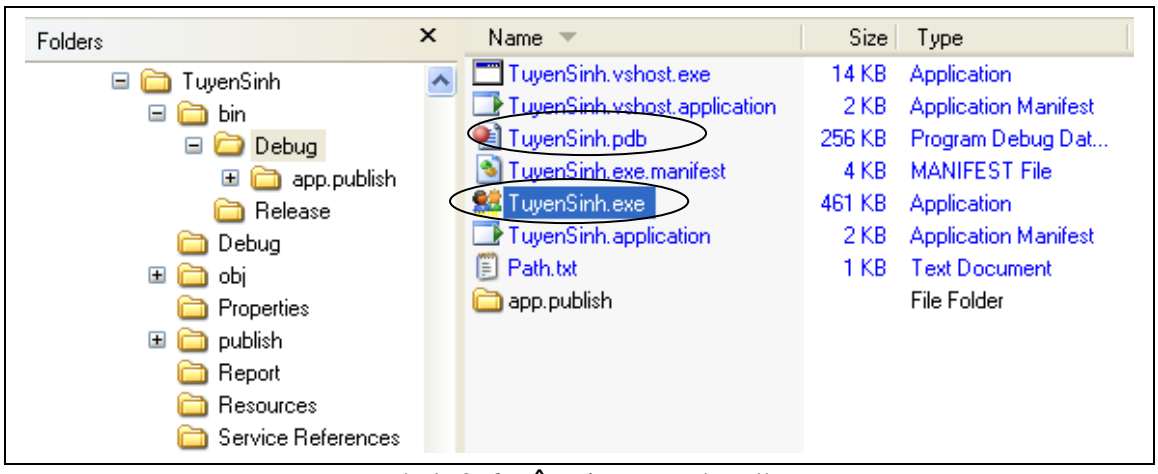
Trong trường hợp có nhiều Project trong một Solution, bạn có thể biên dịch tất cả các Project của Solution đó bằng cách chọn tên Solution rồi R-Click | Build Solution hoặc chọn vào thực đơn Build | Build Solution.

Nếu trước đó đã biên dịch, trong những lần biên dịch sau, thay vì chọn Build Project hay Build Solution thì có thể chọn Rebuild Project hay Rebuild Solution.

Khi biên dịch chương trình C# trong Visual Studio .NET 2008, có hai chế độ biên dịch là gỡ rối (Debug) và phát hành (Release).

Nếu biên dịch Project ở chế độ Debug, sẽ nhận được hai tập tin, tập tin thứ nhất với định dạng là MSIL tuỳ thuộc vào loại Project (EXE, DLL, …) và tập tin thứ hai là .PDB (tập tin lưu trữ các thông tin gỡ rối và trạng thái của Project).

Ví dụ : khi biên dịch ứng dụng tuyển sinh ở chế độ Debug thì tập tin Exe và .PDB tạo ra nằm trong thư mục bin/Debug thuộc thư mục của ứng dụng.

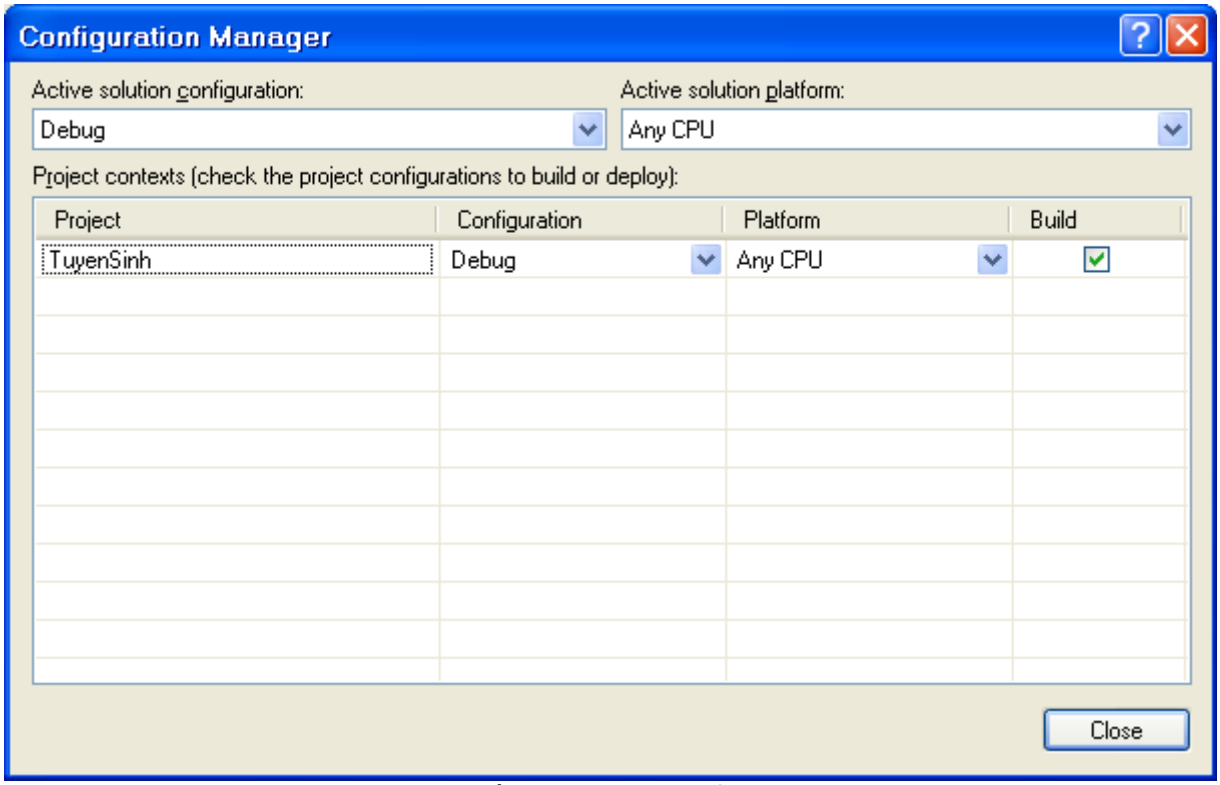


Hình 3.6: tập tin Exe và Pdb

Trong trường hợp biên dịch ở chế độ Release thì tập tin .Exe tạo ra nằm trong thư mục bin/Release thuộc thư mục của ứng dụng.

Lưu ý, khi biên dịch chương trình C# ở chế Release thì Assembly được tạo ra đã được tối ưu hoá về mã thực thi và kích thước của tập tin.

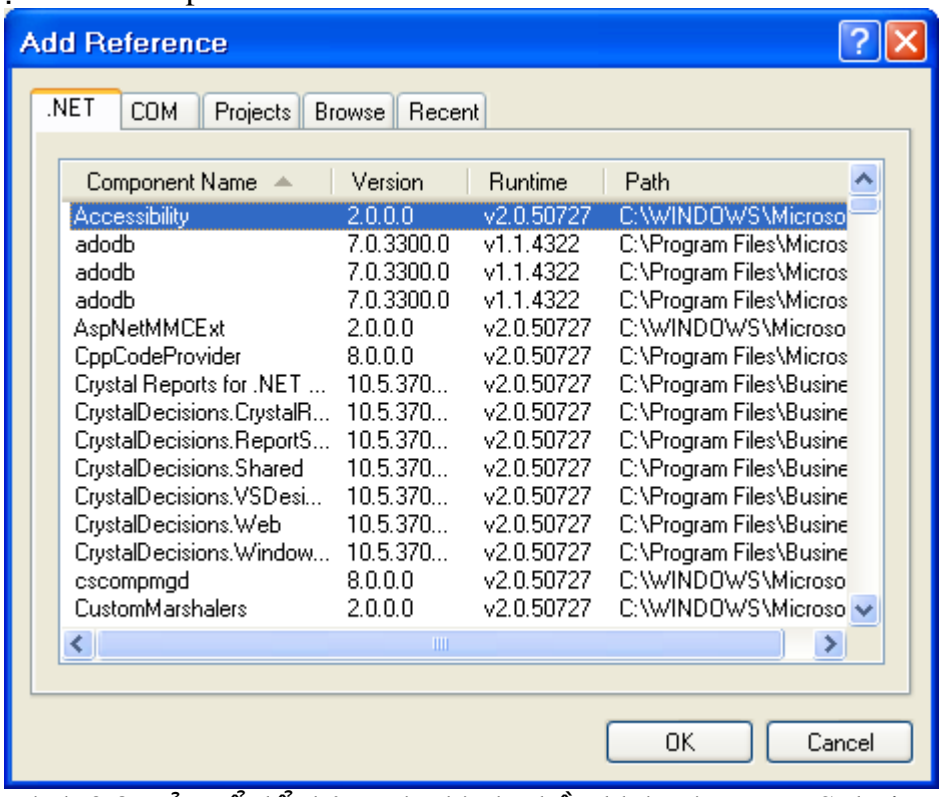
Chúng ta cũng có thể chọn chế độ biên dịch khác nhau cho từng Project trong Solution bằng cách vào Configuration Manager trong phần Solution Configuration, lập tức của sổ Configuration Manager xuất hiện với từng chế độ biên dịch.



Hình 3.7 : Chế độ biên dịch của từng Project

Đối với trường hợp sử dụng Assembly hay Component từ bên ngoài, có thể tham chiếu đến chúng từ ngăn References. Chẳng hạn, có thể tham chiếu đến tập tin DLL nào đó, bạn có thể chọn trong cửa sổ Solution Explorer như sau : R-Click

vào Solution | Add Reference sẽ xuất hiện của sổ Add Reference như hình sau và sau đó chọn các thành phần khác nhau để thêm vào cho Solution.



Hình 3.8: cửa sổ để thêm các thành phần khác vào trong Solution

I.4.2. Từ tiện ích CSC.

Khi biên dịch Class bằng cửa sổ Command Prompt sử dụng tiện ích CSC.EXE, mặc định chế độ biên dịch là Release. Điều này có nghĩa khi biên dịch Class theo chế độ Release thì tập tin Assembly kết xuất ra ở dạng nhị phân .EXE, .DLL, … và tập tin này sẽ nằm cùng nằm thư mục chứa tập tin .cs (có thể nằm ngoài nếu chỉ định nơi chứa tập tin .exe khi dùng tiện ích CSC.EXE để biên dịch).

Cú pháp biên dịch Class bằng Command Prompt như sau :

csc /t : loại /r:dllname file.cs

csc /t : loại /r:dllname file.cs /out:assemblyname

Trong đó, tuỳ chọn /t (có thể khai báo đầy đủ /target) chỉ định loại tập tin được tạo khi biên dịch, nếu ứng dụng là Console Application, Windows Application hay Class thì tập tin tạo ra là .EXE. Trong những trường hợp loại Project là ClassLibrary hay Class thì có thể là .DLL.

Tuỳ chọn /r:tên thư viện cần tham chiếu có sử dụng hay kế thừa trong Class hay Project.

Cách biên dịch chương trình C# đối với ứng dụng Console

Cách 1 : Sử dụng Notepad soạn thảo

Bước 1: Soạn thảo tập tin và lưu với tên **C:\ChaoMung.cs** có nội dung như sau

```
class ChaoMung
  {
      static void Main()
        {
```

```
            // Xuat ra man hinh chuoi thong bao 'Chao mung ban den voi C# 2008 '
            System.Console.WriteLine("Chao mung ban den voi C# 2008 ") ;
            System.Console.ReadLine() ;
        }
    }
```

Bước 2: Vào menu Start | All Programs | Microsoft Visual Studio 2008 | Visual Studio Tools | Visual Studio 2008 Command Prompt.

Bước 3: Gõ lệnh biên dịch tập tin **ChaoMung.cs** sang tập tin **ChaoMung.exe**

C:\> csc /t:exe /out:chaomung.exe chaomung.cs

C:\> chaomung.exe

Chao mung ban den voi C# 2008

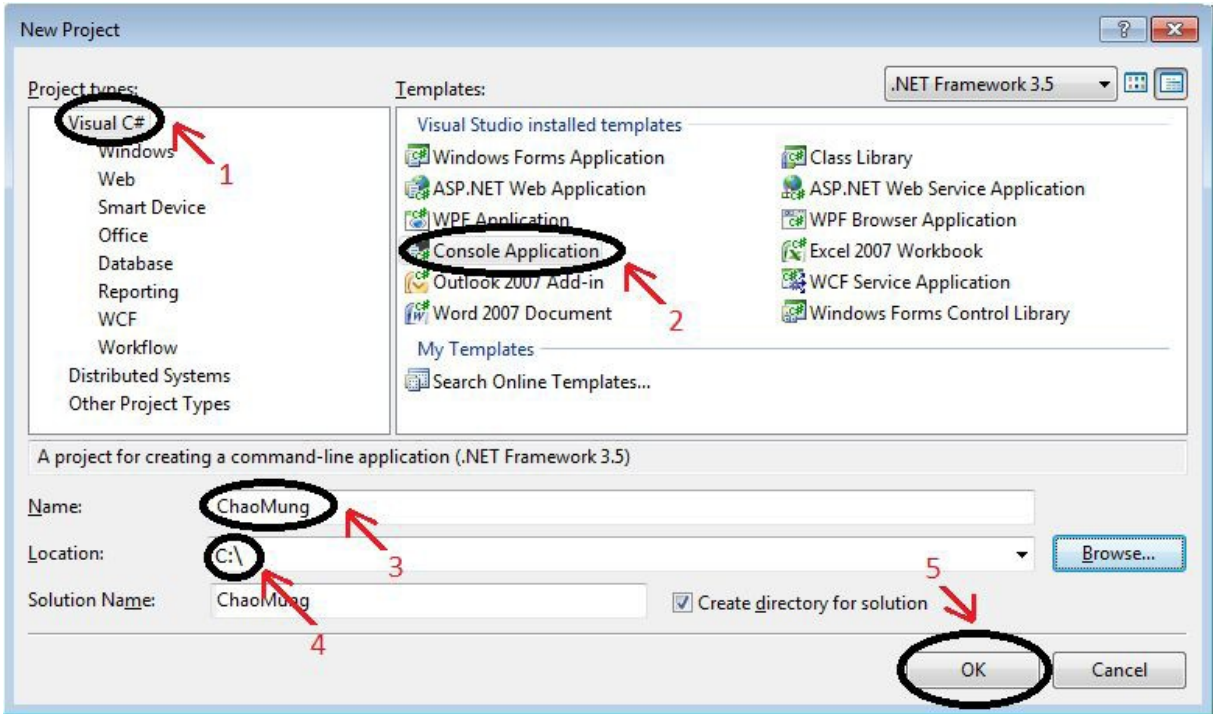
Cách 2. Sử dụng Micosoft Visual Studio 2008 để tạo chương trình

Bước 1: Khởi động Visual Studio 2008

Start | All Programs | Microsoft Visual Studio 2008 | Microsoft Visual Studio 2008

Bước 2: Vào menu File | New | Project

Bước 3: Khai báo



* Mặc định: Visual Studio 2008 (Visual Studio .NET) sẽ tạo ra tập tin Program.cs chứa một **namespace** tên ChaoMung và trong namespace này chứa một **class** tên Program.

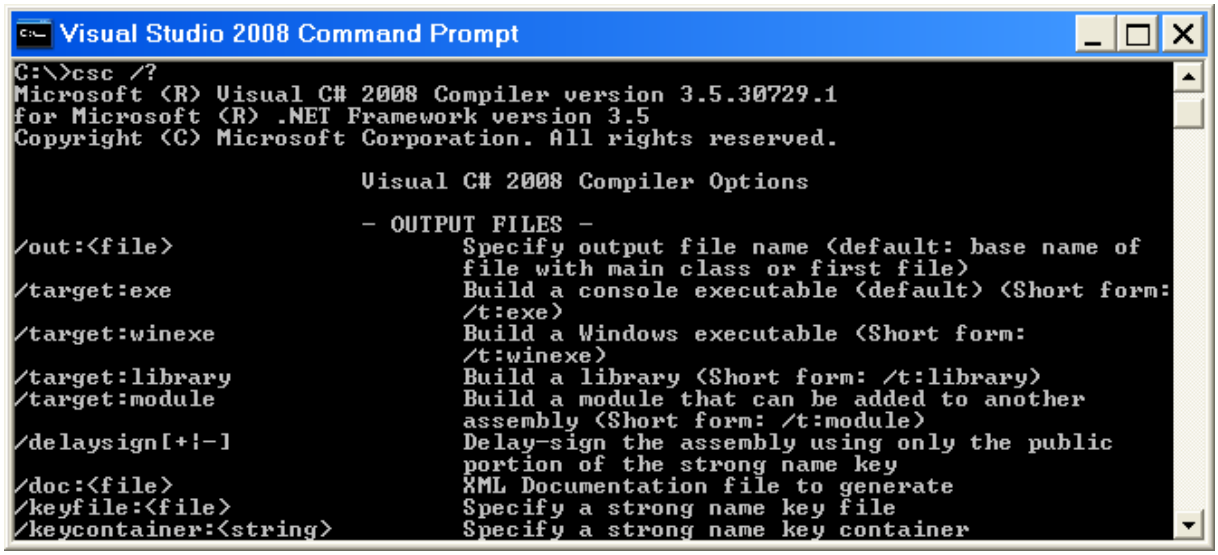
Bước 4: trong phương thức Main, gõ đoạn mã lệnh sau

* Ví dụ:

```
            // Xuat ra man hinh chuoi thong bao 'Chao mung ban den voi C# 2008 '
            System.Console.WriteLine("Chao mung ban den voi C# 2008 ") ;
            System.Console.ReadLine() ;
```

Bước 5: Để chạy chương trình, nhấn F5 hoặc nhắp vào nút 

Muốn biết các tham số của tiện ích csc dùng câu lệnh sau trong Dos : c:\>csc /?

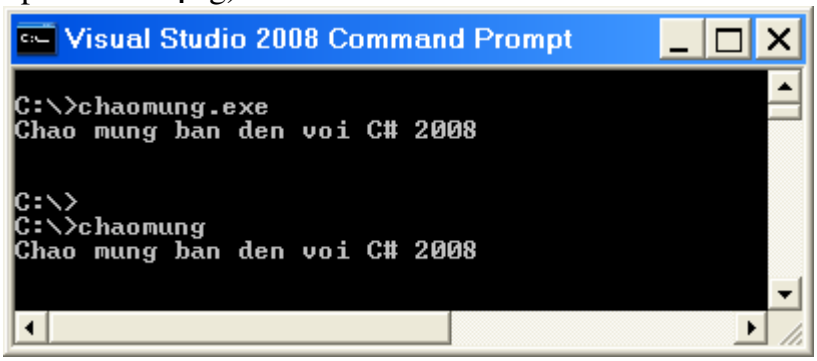


Lưu ý, sử dụng tiện ích miễm phí csc.exe của .NET Framework cho phép biên dịch mọi loại ứng dụng trên .NET. Tuy nhiên, trong trường hợp đã sở hữu Visual .NET 2008 thì không nên sử dụng tiện ích này do nó có thể tốn nhiều thời gian cho soạn thảo chương trình lẫn quá trình biên dịch.

I. 5. Thực thi chương trình C#

Sau khi đã biên dịch chương trình C# ra tập tin .exe có thể thực thi chương trình này bằng cách Double Click vào tập tin .exe trong giao diện trực quan của màn hình Windows.

Đối với trường hợp sử dụng Command Prompt, gọi tên của tập tin EXE (có hoặc không có phần mở rộng) như hình sau :



II. GIẢI THÍCH CÁC KHÔNG GIAN TÊN

II.1 Không gian tên

.NET Framework từ version 2.0 trở lên cung cấp nhiều không gian tên chứa đựng Class, giao tiếp (Interface), kiểu dữ liệu dùng chung cho mọi ngôn ngữ lập trình hỗ trợ .NET.

Như chúng ta đa biết .NET cung cấp một thư viện các lớp đồ sộ và thư viện này có tên là FCL (Framework Class Library). Trong đó Console chỉ là một lớp nhỏ trong hàng ngàn lớp trong thư viện. Mỗi lớp có một tên riêng, vì vậy FCL có hàng ngàn tên như ArrayList, Dictionary, FileSelector,…Điều này làm nảy sinh vấn đề,

người lập trình không thể nào nhớ hết được tên của các lớp trong .NET Framework. Tệ hơn nữa là sau này có thể ta tạo lại một lớp trùng với lớp đã có chẳng hạn.

Namespace cung cấp cho ta cách mà chúng ta tổ chức quan hệ giữa các lớp và các kiểu khác.Namespace(địa bàn hoạt động của các tên) là cách mà .NET tránh né việc các tên lớp, tên biến, tên hàm. đụng độ vì trùng tên giữa các lớp.

Ví dụ trong quá trình phát triển một ứng dụng ta cần xây dựng một lớp từ điển và lấy tên là Dictionary, và điều này dẫn đến sự tranh chấp khi biên dịch vì C# chỉ cho phép một tên duy nhất. Chắc chắn rằng khi đó chúng ta phải đổi tên của lớp từ điển mà ta vừa tạo thành một cái tên khác chẳng hạn như myDictionary. Khi đó sẽ làm cho việc phát triển các ứng dụng trở nên phức tạp, cồng kềnh. Đến một sự phát triển nhất định nào đó thì chính là cơn ác mộng cho nhà phát triển.

Giải pháp để giải quyết vấn đề này là việc tạo ra một namespace, namsespace sẽ hạn chế phạm vi của một tên, làm cho tên này chỉ có ý nghĩa trong vùng đa định nghĩa.

Giả sử có một người nói Tùng là một kỹ sư, từ kỹ sư phải đi kèm với một lĩnh vực nhất định nào đó, vì nếu không thì chúng ta sẽ không biết được là anh ta là kỹ sư cầu đường, cơ khí hay phần mềm. Khi đó một lập trình viên C# sẽ bảo rằng Tùng là CauDuong.KySu phân biệt với CoKhi.KySu hay PhanMem.KySu. Namespace trong trường hợp này là CauDuong, CoKhi, PhanMem sẽ hạn chế phạm vi của những từ theo sau. Nó tạo ra một vùng không gian để tên sau đó có nghĩa.

Tương tự như vậy ta cứ tạo các namespace để phân thành các vùng cho các lớp trùng tên không tranh chấp với nhau. Tương tự như vậy, .NET Framework có xây dựng một lớp Dictionary bên trong namespace System.Collections, và tương ứng ta có thể tạo một lớp Dictionary khác nằm trong namespace ProgramCSharp.DataStructures, điều này hoàn toàn không dẫn đến sự tranh chấp với nhau.

Trong ví dụ minh họa đối tượng Console bị hạn chế bởi namespace bằng việc sử dụng mã lệnh:

System.Console.WriteLine();

Toán tử '.'

Trong vídụ này trên dấu '.' được sử dụng để truy cập đến phương thức hay dữ liệu trong một lớp (trong trường hợp này phương thức là WriteLine()), và ngăn cách giữa tên lớp đến một namespace xác nhận (namspace System và lớp là Console). Việc thực hiện này theo hướng từ trên xuống, trong đó mức đầu tiên namespace là System, tiếp theo là lớp Console, và cuối cùng là truy cập đến các phương thức hay thuộc tính của lớp.

Trong nhiều trường hợp namespace có thể được chia thành các namespace con gọi là subnamespace. Ví dụ trong namespace System có chứa một số các subnamesapce như Configuration, Collections, Data, và còn rất nhiều nữa, hơn nữa trong namespace Collection còn chia thành nhiều namesapce con nữa.

Namespace giúp chúng ta tổ chức và ngăn cách những kiểu. Khi chúng ta viết một chương trình C# phức tạp, chúng ta có thể phải tạo một kiến trúc namespace riêng cho mình, và không giới hạn chiều sâu của cây phân cấp namespace. Mục đích của namespace là giúp chúng ta chia để quản lý những kiến trúc đối tượng phức tạp.

Từ khóa using

Để làm cho chương trình gọn hơn, và không cần phải viết từng namespace cho từng đối tượng, C# cung cấp từ khóa là using, sau từ khóa này là một namespace hay subnamespace với mô tả đầy đủ trong cấu trúc phân cấp của nó. Ta có thể dùng dòng lệnh :

using System;

ở đầu chương trình và khi đó trong chương trình nếu chúng ta có dùng đối tượng Console thì không cần phải viết đầy đủ : System.Console. mà chỉ cần viết Console. thôi.

Ví dụ : Dùng khóa using

```
using System;
class ChaoMung
        {
                static void Main()
                {
                        //Xuat ra man hinh chuoi thong bao
                        Console.WriteLine("Chao Mung");
                }
        }
```

Kết quả:

Chao Mung

Lưu ý rằng phải đặt câu using System trước định nghĩa lớp ChaoMung. Mặc dù chúng ta chỉ định rằng chúng ta sử dụng namespace System, và không giống như các ngôn ngữ khác, không thể chỉ định rằng chúng ta sử dụng đối tượng System.Console.

Ví dụ 2.4: Không hợp lệ trong C#.

```
using System.Console;
class ChaoMung
        {
                static void Main()
                {
                        //Xuat ra man hinh chuoi thong bao
                        WriteLine("Chao Mung");
                }
        }
```

Đoạn chương trình trên khi biên dịch sẽ được thông báo một lỗi như sau:

error CS0138: A using namespace directive can only be applied to namespace; 'System.Console' is a class not a namespace.

Cách biểu diễn namespace có thể làm giảm nhiều thao tác gõ bàn phím, nhưng nó có thể sẽ không đem lại lợi ích nào bởi vì nó có thể làm xáo trộn những namespace có tên không khác nhau. Giải pháp chung là chúng ta sử dụng từ khóa using với các namespace đa được xây dựng sẵn, các namespace do chúng ta tạo ra, những namespace này chúng ta đa nắm chắc sưu liệu về nó. Còn đối với namespace do các hãng thứ ba cung cấp thì chúng ta không nên dùng từ khóa using.

Tuỳ thuộc vào từng loại ứng dụng mà các lập trình viên phát triển để có thể tham chiếu đến các không gian tên chứa các lớp bao gồm đối tượng mà lập trình

viên dùng. Sau đây chúng ta sẽ tìm hiểu một số không gian tên thường được sử dụng khi xây dựng bằng C#.

Bí danh Namespace

Một cách sử dụng khác từ khoá using là gán những bí danh cho các lớp và namespace. Nếu bạn có 1 namespace dài lê thê mà bạn muốn quy chiếu nhiều chỗ trên đoạn mã . bạn có thể gán một alias cho namespace.

Cú pháp : using alias = NamespaceName;

Chúng ta sử dụng đối tượng này để trả về tên của lớp namespace thí dụ sau Introduction được đặt cho Wrox.ProCSharp.Basics namespace đại diện cho đối tượng NamespaceExample Đối tượng này có một phương thức GetNamespace(), sử dụng phương thức GetType() mà mọi lớp đều có thể truy xuất một đối tượng Type tượng trương cho kiểu dữ liệu của lớp. Chúng ta dùng đối tượng này để trả về tên namespace cho lớp.

```
using System;
using Introduction = Wrox.ProCSharp.Basics;
class Test
{
  public static int Main()
  {
    Introduction.NamespaceExample NSEx =
      new Introduction.NamespaceExample();
    Console.WriteLine(NSEx.GetNamespace());
    return 0;
  }
}

namespace Wrox.ProCSharp.Basics
{
  class NamespaceExample
  {
    public string GetNamespace()
    {
      return this.GetType().Namespace;
    }
  }
}
```

Một số loại namespace thông dụng :

II.2. Một số không gian tên thường sử dụng

System; System.Collections.Generic; System.Linq; System.Text;

III. CÁC DẠNG CỦA PHƯƠNG THỨC MAIN

Như đã giới thiệu ở chương 1, phương thức Main là phương thức đặc biệt của mỗi chương trình C#, nó dùng để làm điểm khởi động chương trình nếu đó là chương trình thực thi.

Nếu ứng dụng chương trình C# thuộc dạng chương trình thực thi thì phải có 1 và chỉ một phương thức Main trong Project và không nên sử dụng từ khoá Public.

Phương thức Main dạng ứng dụng Console

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace th111
{
   class Program
   {
      static void Main(string[] args)
      {
         //các câu lệnh được đưa vào
                  // bởi người lập trình
      }
   }
}
```

Phương thức Main dạng Windows Application

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows.Forms;
namespace WindowsFormsApplication1
{
   static class Program
   {
      /// <summary>
      /// The main entry point for the application.
      /// </summary>
      [STAThread]
      static void Main()
      {
                  Application.EnableVisualStyles();
                  Application.SetCompatibleTextRenderingDefault(false);
                  Application.Run(new Form1());
      }
   }
}
```

Phương thức main có truyền tham số

```
class ChaoMung
{
  static void Main(string[] args)
   {
      // Xuất ra màn hình chuỗi thông báo truyền vào cho hàm Main
      System.Console.WriteLine(args.Length) ;
```

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

Phương thức Main không có truyền tham số

```
class ChaoMung
{
  static void Main()
    {
        // Xuat ra man hinh chuoi thong bao
       //'Chao   mung ban den voi C# 2008 '
      System.Console.WriteLine("Chao mung ban den voi C# 2008 ") ;
      System.Console.ReadLine() ;
    }
}
```

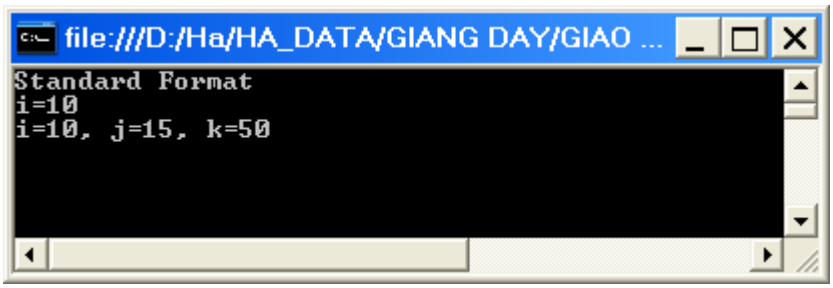
IV. ĐỊNH DẠNG KẾT QUẢ CỦA CỬA SỔ COMMAND PROMPT

Nếu đã chạy ứng dụng Console Application, kết quả trình bày trên cửa sổ Command Prompt theo định dạng do người lập trình định nghĩa.

Ví dụ : để trình bày giá trị của 3 biến ra cửa sổ Command Prompt, sử dụng cặp dấu { } và số thứ tự vào bên trong (i) để chỉ định giá trị của biến in ra.

```
static void Standard()
{
        int i=10,j=15,k=50;
       Console.WriteLine("Standard Format");
        Console.WriteLine("i={0}", i);
       Console.WriteLine("i={0}, j={1}, k={2}", i,j,k);
    }
```

Kết quả :



Một số dạng thức khi thực hiện khi in ra trong của sổ Command Prompt

Ký tự	Ý nghĩa
\’	Dấu nháy đơn
\”	Dấu nháy kép
\\	Dấu chéo
\0	Ký tự null
\a	Đỗ chuông
\b	Backspace

\f	Sang trang form feed
\n	Dòng mới
\r	Đầu dòng
\t	Tab ngang
\v	Tab dọc

Ví dụ : về định dạng ngày, giờ

```
static void Date()
    {
       DateTime thisDate = DateTime.Now;
       Console.WriteLine("DateTime Format");
       Console.WriteLine(
          "(d) Short date: . . . . . {0:d}\n" +
          "(D) Long date:. . . . . {0:D}\n" +
          "(t) Short time: . . . . . {0:t}\n" +
          "(T) Long time:. . . . . {0:T}\n" +
          "(f) Full date/short time: . . {0:f}\n" +
          "(F) Full date/long time:. . . {0:F}\n" +
          "(g) General date/short time:. {0:g}\n" +
          "(G) General date/long time: . {0:G}\n" +
          " (default):. . . . . {0} "+
          "(default = 'G')\n" +
          "(M) Month:. . . . . {0:M}\n" +
          "(R) RFC1123:. . . . . {0:R}\n" +
          "(s) Sortable: . . . . . {0:s}\n" +
          "(u) Universal sortable: . . . {0:u}"+
          "(invariant)\n" +
          "(U) Universal sortable: . . . {0:U}\n" +
          "(Y) Year: . . . . . {0:Y}\n",
          thisDate);
    }
```

Ví dụ : về định dạng số

```
static void Number()
    {
       Console.WriteLine("Numeric Format");
       Console.WriteLine(
       "(C) Currency: . . . . . {0:C}\n"  +
       "(D) Decimal:. . . . . {0:D}\n"  +
       "(E) Scientific: . . . . . {1:E}\n"  +
       "(F) Fixed point:. . . . . {1:F}\n"  +
       "(G) General:. . . . . {0:G}\n"  +
       "    (default):. . . . . {0} "+
       "(default = 'G')\n" +
       "(N) Number: . . . . . {0:N}\n"  +
       "(P) Percent:. . . . . {1:P}\n"  +
```

```
            "(R) Round-trip: . . . . . {1:R}\n" +
            "(X) Hexadecimal:. . . . . {0:X}\n" , 453,  453.45F);
        }
```

Ví dụ : về định dạng màu sắc

```
    enum Color { Yellow = 1, Blue, Green };
        static void Enumeration()
        {

            Console.WriteLine("Enumeration Format");
            Console.WriteLine(
                "(G) General:. . . . . {0:G}\n" +
                "(default):. . . . . {0}"+
                "(default = 'G')\n" +
                "(F) Flags:. . . . . {0:F} "+
                "(flags or integer)\n" +
                "(D) Decimal number: . . . . . {0:D}\n" +
                "(X) Hexadecimal:. . . . . {0:X}\n", Color.Green);
        }
```

V. CHÚ THÍCH TRONG CHƯƠNG TRÌNH C#

Chú thích (Comment) trong chương trình C# là những phần text làm rõ hơn cho phần code của lập trình viên, hiểu được mục đích và chi tiết kỹ thuật của câu lệnh, đoạn chương trình, phương thức, thuộc tính hay class.

Nếu không có chú thích rõ ràng mục đích và chi tiết kỹ thuật của đoạn chương trình tạo ra, khi lỗi phát sinh hay có nhu cầu thay đổi, thêm chức năng, tương tác với chức năng khác… thì ngay chính người lập trình phải tiêu tốn nhiều thời gian để tìm hiểu những gì đã làm trước đây trên mỗi dòng code. Chính vì vậy Comment là những câu lệnh quan trọng, đoạn chương trình, phương thức, thuộc tính hay class là điều bắt buộc khi làm việc trong nhóm lập trình.

Chú thích không được đọc bởi trình biên dịch, nó không liên quan gì đến chương trình.

Có 2 cách viết chú thích trong C#:

Nếu chú thích trên một dòng bạn đặt phần chú thích sau 2 dấu sổ chéo

```
// chú thích
```

Nếu chú thích trên nhiều dòng bạn đặt p hần chú thích trong cặp /* */ cụ thể

```
/* chú thích*/
```

VI. KHAI BÁO CHỈ THỊ REGION

Khi có nhiều phương thức muốn nhóm theo từng nhóm, có thể sử dụng khai báo mở #region và khai báo đóng là #endregion

```
    namespace myRegion
    {
       static class Program
       {
          /// <summary>
          /// The main entry point for the application.
          /// </summary>
```

```
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Form1());
        }
        #region Format of Number
            static void Standard()
            {
                int i = 10, j = 15, k = 50;
                Console.WriteLine("Standard Format");
                Console.WriteLine("i={0}", i);
                Console.WriteLine("i={0}, j={1}, k={2}", i, j, k);
            }
            enum Color { Yellow = 1, Blue, Green };
            static void Enumeration()
            {
                Console.WriteLine("Enumeration Format");
                Console.WriteLine(
                    "(G) General:. . . . . {0:G}\n" +
                    "(default):. . . . . {0}" +
                    "(default = 'G')\n" +
                    "(F) Flags:. . . . . {0:F} " +
                    "(flags or integer)\n" +
                    "(D) Decimal number: . . . . . {0:D}\n" +
                    "(X) Hexadecimal:. . . . . {0:X}\n",
                    Color.Green);
            }
        #endregion
    }
}
```

B. CÂU HỎI VÀ BÀI TẬP

I. Kỹ năng 1 : Xây dựng và thực thi một ứng dụng

1.1. Sử dụng Notepad soạn thảo

Bước 1: Soạn thảo tập tin và lưu với tên **C:\ChaoMung.cs** có nội dung như sau

```
class ChaoMung
  {
      static void Main()
        {
            // Xuat ra man hinh chuoi thong bao 'Chao mung ban den voi C# 2008 '
            System.Console.WriteLine("Chao mung ban den voi C# 2008 ") ;
            System.Console.ReadLine() ;
        }
```

}

Bước 2: Vào menu Start | All Programs | Microsoft Visual Studio 2008 | Visual Studio Tools | Visual Studio 2008 Command Prompt.

Bước 3: Gõ lệnh biên dịch tập tin **ChaoMung.cs** sang tập tin **ChaoMung.exe**

C:\> csc /t:exe /out:chaomung.exe chaomung.cs

C:\> chaomung.exe

Chao mung ban den voi C# 2008

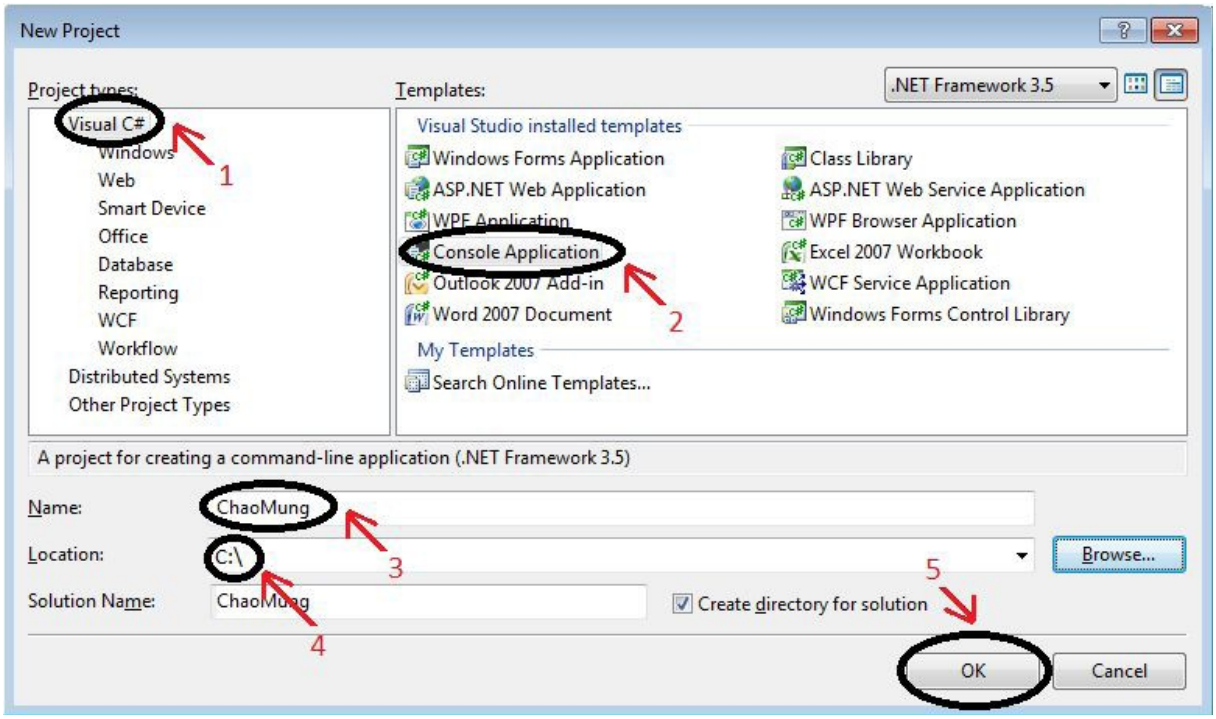
1.2. Sử dụng Micosoft Visual Studio 2008 để tạo chương trình

Bước 1: Khởi động Visual Studio 2008

Start | All Programs | Microsoft Visual Studio 2008 | Microsoft Visual Studio 2008

Bước 2: Vào menu File | New | Project

Bước 3: Khai báo



* Mặc định: Visual Studio 2008 (Visual Studio .NET) sẽ tạo ra tập tin Program.cs chứa một **namespace** tên ChaoMung và trong namespace này chứa một **class** tên Program.

Bước 4: trong phương thức Main, gõ đoạn mã lệnh sau

* Ví dụ:

```
// Xuat ra man hinh chuoi thong bao 'Chao mung ban den voi C# 2008 '
System.Console.WriteLine("Chao mung ban den voi C# 2008 ") ;
System.Console.ReadLine() ;
```

Bước 5: Để chạy chương trình, nhấn F5 hoặc nhắp vào nút 

Chú ý : Khi tạo một chương trình trong C#, chúng ta nên thực hiện theo các bước sau:

Bước 1: Xác định mục tiêu của chương trình.

Bước 2: Xác định những phương pháp giải quyết vấn đề.

Bước 3: Tạo một chương trình để giải quyết vấn đề.

Bước 4: Thực thi chương trình để xem kết quả.

II. Kỹ năng 2 : viết các chương trình hiển thị trong cửa sổ Command Promt.

Viết các chương trình hiển thị trong cửa sổ Command Promt. Với các định dạng : ngày giờ, tháng năm, màu của ký tự hiển thị tương ứng với các ví dụ trên.

III. Kỹ năng 3 : viết chương trình theo định dạng cho sẵn ở cửa sổ Command Promt

Chương trình 1 :

```
-----------------------------------
|                                 |
|                                 |
|                                 |
-----------------------------------
```

Chương trình 2 :

```
        *
      *   *
    *       *
  *           *
* * * * * * * *
```

MÃ BÀI HỌC LTTQ – 03	BÀI 4 : NỀN TẢNG CỦA NGÔN NGỮ C#	Thời gian (giờ)				
		LT 5	TH 10	BT 5	KT 1	TS 21

Mục tiêu:

Sau khi học xong bài này, học viên có khả năng:

- Phân biệt được các kiểu dữ liệu, trình bày phạm vi, các phép toán trên các kiểu dữ liệu;
- Khai báo được biến và đối tượng có kiểu dữ liệu trước khi sử dụng;
- Sử dụng đúng cú pháp các câu lệnh điều kiện, vòng lặp, xử lý lỗi;
- Kiểm soát được các lỗi phát sinh trong chương trình;
- Sử dụng thành thạo debugger;
- Thực hiện các thao tác an toàn với máy tính.

TÓM TẮT BÀI:

- Kiểu dữ liệu trong C# bao gồm : kiểu dữ liệu được cài đặt sẵn như int, char…và kiểu dữ liệu do người sử dụng định nghĩa. Khi kiểu dữ liệu được khai báo đều thuộc một trong hai laọi chính là kiểu giá trị (Value Types) dùng để lưu trữ giá trị hay kiểu tham chiếu (Reference Types) dùng để lưu trữ tham chiếu đến giá trị thực.
- Tìm hiểu về cách khai báo biến, các kiểu dữ liệu, hằng và enum được sử dụng trong C#;
- Các phép toán sử dụng trong chương trình C# và cách khai chuyển đổi kiểu dữ liệu;
- Danh sách phát biểu điều khiển sắp xếp nhóm như : Phát biểu rẽ nhánh (Selection), vòng lặp (Iteration), nhảy (Jump), khối lệnh (using), kiểm tra (Checked và Unchecked), khoá (lock) và bố trí (Fixed);
- Kiểm soát được các lỗi có thể phát sinh trong quá trình thi hành, sử dụng cấu trúc try… catch… finally.

Các vấn đề chính sẽ được đề cập

- Kiểu dữ liệu
- Kiểu dữ liệu xây dựng sẵn
- Chọn kiểu dữ liệu
- Chuyển đổi các kiểu dữ liệu
- Biến và hằng
- Gán giá trị xác định cho biến
- Hằng
- Kiểu liệt kê
- Kiểu chuỗi kí tự
- Định danh
- Biểu thức
- Khoảng trắng
- Câu lệnh
- Phân nhánh không có điều kiện
- Phân nhánh có điều kiện
- Câu lệnh lặp

- Toán tử
- Namespace
- Các chỉ dẫn biên dịch
- Lớp ngoại lệ
- Cấu trúc kiểm soát ngoại lệ try…catch…finally
- Ném lỗi
- Sử dụng debugger

NỘI DUNG :

A. LÝ THUYẾT

Trong bài trước chúng ta đa tìm hiểu một chương trình C# đơn giản nhất. Chương trình đó chưa đủ để diễn tả một chương trình viết bằng ngôn ngữ C#, có quá nhiều phần và chi tiết đã bỏ qua. Do vậy trong bài này chúng ta sẽ đi sâu vào tìm hiểu cấu trúc và cú pháp của ngôn ngữ C#.

Bài này sẽ thảo luận về hệ thống kiểu dữ liệu, phân biệt giữa kiểu dữ liệu xây dựng sẵn (như int, bool, string…) với kiểu dữ liệu do người dùng định nghĩa (lớp hay cấu trúc do người lập trình tạo ra...). Một số cơ bản khác về lập trình như tạo và sử dụng biến dữ liệu hay hằng cũng được đề cập cùng với cấu trúc liệt kê, chuỗi, định danh, biểu thức và cậu lệnh.

Trong phần hai của bài hướng dẫn và minh họa việc sử dụng lệnh phân nhánh **if, switch**, **while**, **do**...**while**, **for**, và **foreach**. Và các toán tử như phép gán, phép toán logic, phép toán quan hệ, và toán học...

Như chúng ta đa biết C# là một ngôn ngữ hướng đối tượng rất mạnh, và công việc của người lập trình là kế thừa để tạo và khai thác các đối tượng. Do vậy để nắm vững và phát triển tốt người lập trình cần phải đi từ những bước đi dầu tiên tức là đi vào tìm hiểu những phần cơ bản và cốt lõi nhất của ngôn ngữ.

I. KIỂU DỮ LIỆU TRONG C#

C# là ngôn ngữ lập trình mạnh về kiểu dữ liệu, một ngôn ngữ mạnh về kiểu dữ liệu là phải khai báo kiểu của mỗi đối tượng khi tạo (kiểu số nguyên, số thực, kiểu chuỗi, kiểu điều khiển...). Kiểu dữ liệu của một đối tượng là cách thức để trình biên dịch nhận biết được kích thước của đối tượng (kiểu int có kích thước là 4 byte) và các khả năng của nó (chẳng hạn, một đối tượng Button có thể vẽ, phản ứng khi nhấn,...).

Tương tự như C++ hay Java, C# có hai kiểu dữ liệu chính: kiểu xây dựng sẵn (ngôn ngữ cung cấp cho người lập trình) và kiểu do người dùng định nghĩa (người dùng tự xây dựng các kiểu dữ liệu mới dựa trên các kiểu có sẵn).

Ứng với kiểu dữ liệu xây dựng sẵn hoặc kiểu dữ liệu do người dùng định nghĩa, C# lại phân các loại kiểu dữ liệu này thành hai nhóm: Kiểu dữ liệu giá trị (value) và kiểu dữ liệu tham chiếu (reference). Có được sự phân biệt giữa kiểu dữ liệu giá trị và kiểu dữ liệu tham chiếu là do cơ chế lưu trữ của chúng trong bộ nhớ. Kiểu dữ liệu giá trị lưu giá trị thật của nó trong bộ nhớ stack, trong khi đó, kiểu dữ liệu tham chiếu lại lưu trữ địa chỉ của nó

trong bộ nhớ heap.

Nếu ta có đối tượng với kích thước rất lớn thì việc lưu giữ chúng trên bộ nhớ heap rất có ích, trong chương 4 sẽ trình bày những lợi ích và bất lợi khi làm việc với kiểu dữ liệu tham chiếu, còn trong chương này chỉ tập trung kiểu dữ kiểu cơ bản hay kiểu xây dựng sẵn.

✍ *Ghi chú*: Tất cả các kiểu dữ liệu xây dựng sẵn là kiểu dữ liệu giá trị ngoại trừ các đối tượng và chuỗi. Và tất cả các kiểu do người dùng định nghĩa ngoại trừ kiểu cấu trúc đều là kiểu dữ liệu tham chiếu.

C# là ngôn ngữ lập trình có cấu trúc trong bộ Visual Studio . NET 2008, chính vì vậy mọi biến và đối tượng đều phải khai báo kiểu dữ liệu trước khi sử dụng.

Kiểu dữ liệu trong C# bao gồm: Kiểu dữ liệu được cài sẵn như int, char,… và kiểu dữ liệu do người sử dụng định nghĩa. Khi kiểu dữ liệu được khai báo đều thuộc một trong hai loại chính là kiểu giá trị (Value Types) dùng để lưu trữ giá trị hay kiểu tham chiếu (Reference Types) dùng để lưu trữ tham chiếu đến giá trị thực.

Trong phần này chúng ta tập trung tìm hiểu cách khai báo biến, các kiểu dữ liệu, hằng và enum được sử dụng trong C#.

I.1. KIỂU VALUE

Kiểu dữ liệu Value chia thành hai loại chính: Structs, Enumerations. Trong đó, Structs bao gồm kiểu số (Numeric) tương ứng với ba loại chính:

Kiểu số.

Số nguyên tổng quát (Integral).

Chấm động (Floating-point).

Kiểu số thập phân (Decimal).

Kiểu luận lý (bool).

Kiểu do người dùng định nghĩa (User defined).

I.1.1 Kiểu số

I.1.1.1. Kiểu số nguyên tổng quát

Kiểu số nguyên tổng quát được phân ra thành nhiều loại khác nhau tùy thuộc vào kích thước và khoảng giá trị của kiểu số nguyên tổng quát như bảng 4-1.

Kiểu	**Khoảng giá trị**	**Kích thước**
Sbyte	-128 -> 127	Số nguyên có dấu 8-bit.
byte	0 -> 255	Số nguyên không dấu 8-bit.
char	U+0000 -> U+ffff	Ký tự Unicode 16-bit.
short	-32,768 -> 32,767	Số nguyên có dấu 16-bit
ushort	0 -> 65,535	Số nguyên không dấu 16-bit
int	-2,147,483,647 -> 2,147,483,647	Số nguyên có dấu 32-bit
uint	0 -> 4, 294,967,295	Số nguyên không dấu 32-bit
long	-9,223,372,036,854,775,808 -> 9,223,372,036,854,775,808	Số nguyên có dấu 64-bit
ulong	0 ->18,446,744,073,709,551,615	Số nguyên không dấu 64-bit.

Bảng 4-1: Kiểu số nguyên tổng quát

Lưu ý, nếu giá trị có kiểu số nguyên vượt quá số ulong thì lỗi sẽ phát sinh khi biên dịch.

Các kiểu dữ liệu số nguyên trình bày trong bảng 4-1 thuộc các không gian tên tương ứng trong .NET Framework như bảng 4-2.

Kiểu trong C#	.NET Framework
byte	System.byte
sbyte	System.sbyte
char	System.char
int	System.int32
uint	System.uint32
long	System.int64
Ulong	System.uint64
short	System.int16
ushort	System.uint16

Bảng 4-2: Kiểu số nguyên.

I.1.1.2. Kiểu số chấm động

Kiểu số chấm động bao gồm hai loại chính là float và double với kích thước vào khoảng giá trị xấp xỉ trình bày trong bảng 4-3.

Kiểu	Giá trị xấp xỉ	Số lẻ
Float Double	1.5e-45-> 3.4e38 5.0e-324-> 1.7e308	7 số 15->16 số

Bảng 4-3: Kiểu số chấm động.

Tương tự như kiểu dữ liệu số nguyên, kiểu số chấm động trình bày trong bảng 4-3 thuộc các không gian tên tương ứng trong .NET Framework như bảng 4-4.

Kiểu trong C#	.NET Framework
Double float	System.double System.single

Bảng 4-4: Kiểu số chấm động.

I.1.1.3. Kiểu số thập phân

Kiểu số thập phân có chiều dài 128-bit. So với kiểu số chấm động thì số lẻ nhiều hơn và khoảng giá trị thì nhỏ hơn. Như vậy, kiểu dữ liệu này thường được sử dụng để trình bày các con số về tài chính và tiền tệ.

Khoảng giá trị xấp xỉ và số lẻ của kiểu số thập phân trình bày trong bảng 4-5

Kiểu	Giá trị xấp xỉ	Số lẻ
decimal	1.0 x 10e-28 -> 7.9 x 10e28	28->29 số có dấu

Bảng 4-5: Kiểu số thập phân.

Không gian tên tương ứng của kiểu số thập phân trong .NET Framework như bảng 4-6.

Kiểu trong C#	.NET Framework

decimal	System.Decimal

Bảng 4-6: Kiểu số thập phân.

I.1.2. Kiểu luận lý

Kiểu luận lý dùng để khai báo biến có thể lưu trữ giá trị là true và false, không gian tên tương ứng trong .NET Framework là System.Boolean.

I. 1.3. Kiểu do người sử dụng định nghĩa

Kiểu struct là kiểu giá trị dùng để nhóm một số biến có quan hệ với nhau nhằm để lưu trữ tập gồm nhiều giá trị của một tập thực thể nào đó.

Lưu ý, struct có thể chứa đựng constructors, hằng, thuộc tính, phương thức, chỉ mục, phép toán, biến cố và cho phépbanj khai báo các kiểu lồng.

Ngoài ra, struct có thể cài đặt interface nhưng chúng không thể kế thừa từ struct khác, đó là lý do tại sao khi khai báo struct bạn không thể sử dụng từ khóa protected để chỉ định tầm vực của struct.

I.2. KIỂU REFERENCE

Kiểu Reference dùng để lưu trữ tham chiếu đến giá trị thực, trong C# kiểu Reference bao gồm: class, interface, delegate, object, string. Trong phần này chúng ta tham kháo hai kiểu còn lại là object và string.

I.2.1 Kiểu object

Kiểu object tương ứng System.object trong .NET Framework. Trong C# mọi kiểu dữ liệu có sẵn hay do người sử dụng định nghĩa thuộc loại reference hay value đều kế thừa trực tiếp hay gián tiếp từ object.

Bạn có thể gán giá trị bất kỳ cho một biến có kiểu object. Khi một biến có kiểu value chuyển đổi sang kiểu object được gọi là boxing. Ngược lại, một biến kiểu object được chuyển đổi sang kiểu value thì được gọi là unboxing (bạn có thể tham khảo hai khái niệm này trong phần chuyển đổi kiểu dữ liệu).

I. 2.2. Kiểu string

Kiểu string trình bày một chuỗi ký tự dạng Unicode. Không gian tên tương ứng trong .NET Framework là System.String.

Mặc dù string là kiểu Reference nhưng hai phép toán so sánh bằng và không bằng (= = và !=) dùng để so sánh giá trị của đối tượng string không phải là tham chiếu.

Khai báo chuỗi hằng:

string <Tên_chuỗi_hằng> = <"Noi dung chuoi hang"> ;

Ví dụ: string HoTen = "Ho Viet Ha" ;

Khai báo biến kiểu chuỗi:

string <Biến_chuỗi> [= "Noi dung chuoi hang"] ;

Ví dụ: string hoten = "Nguyen Van Teo" ;

Nhập chuỗi:

<Biến_chuỗi> = System.Console.ReadLine() ;

Ví dụ: hoten = System.Console.ReadLine() ;

Xuất chuỗi:

System.Console.WriteLine("Chuoi") ;

Ví dụ: System.Console.WriteLine("Do dai cua chuoi la:") ;

Một số thao tác trên chuỗi:

Phương thức	Ý nghĩa
Length	Chiều dài của chuỗi
Substring()	Lấy chuỗi con
ToLower()	Trả về bản sao của chuỗi ở kiểu chữ thường
ToUpper()	Trả về bản sao của chuỗi ở kiểu chữ IN HOA

Ví dụ : Nhập vào họ và tên, in ra màn hình họ tên bằng chữ IN HOA, chữ thường, độ dài của họ và tên.

```
using System ;
class HoTen
  {
      static void Main()
        { // Khai bao bien
            string hoten ;
            // Nhap gia tri cho bien chuoi
            Console.Write("Nhap Ho va Ten: ") ;
            hoten = Console.ReadLine() ;
            // Thao tac tren chuoi
            string HT = hoten.ToUpper() ;
            string ht = hoten.ToLower() ;
            int dodai = hoten.Length ;
            // Xuat ra man hinh
            Console.WriteLine("Ho va Ten (chu IN HOA): {0}", HT) ;
            Console.WriteLine("Ho va Ten (chu thuong): {0}", ht) ;
            Console.WriteLine("Do dai Ho va Ten la: {0}",dodai) ;
         }
  }
```

Bảng : *Các kiểu ký tự đặc biệt.*

Ký tự	Ý nghĩa
\’	Dấu nháy đơn
\”	Dấu nháy kép
\\	Dấu chéo
\0	Ký tự null
\a	Đỗ chuông
\b	Backspace
\f	Sang trang form feed
\n	Dòng mới
\r	Đầu dòng
\t	Tab ngang
\v	Tab dọc

I.3. KIỂU NULLABLE

Kiểu Nullable là đối tượng thuộc cấu trúc System.Nullable có thể trình bày khoảng giá trị dạng Value và có thể thêm giá trị null.

Chẳng hạn, Nullable<Int32>, đánh vần là " Nullable of int32 ", có thể gán giá trị bất kỳ từ -2147483648 đến 2147483647 hay giá trị đó có thể là null.

Tương tự như vậy, Nullable<bool> có thể gán giá trị là true, false hay null. Với cách khai báo này cho phép chúng ta gán giá trị null vào biến có kiểu là số hay Boolean khi làm việc với cơ sở dữ liệu mà gán giá trị tong những cột của hàng nào đó chưa tồn tại giá trị.

Cú pháp T? là viết tắt của System.Nullable<T>, với T là kiểu Value và cú pháp khai báo kiểu dữ kiệu ví dụ như int?.

Lưu ý, không thể tạo một kiểu Nullable với kiểu Reference. Ngoài ra, kiểu Nullable không cho phép khai báo lồng như: Nullable<Nullable<int>>.

II. KHAI BÁO BIẾN

- Biến là một vùng lưu trữ ứng với một kiểu dữ liệu.
- Biến có thể được gán giá trị và cũng có thể thay đổi giá trị trong khi thực hiện các lệnh của chương trình.

II.1. Khai báo biến

Khai báo biến với kiểu dữ liệu trong C# là bắt buộc trước khi sử dụng biến trong chương trình. Để khai báo biến bạn sử dụng cú pháp như sau:

{Kiểu dữ liệu} tên biến [= giá trị khởi tạo];

Ví dụ : int i;

II.2. Gán giá trị cho biến

Nếu trước đó bạn không gán giá trị khởi tạo, trong quá trình tính toán bạn có thể sử dụng các phép toán để gán giá trị cho chúng.

Ví dụ : Khởi tạo và gán giá trị một biến

```
class Bien
{
        static void Main()
         {
                // Khai bao va khoi tao bien
                int bien = 9 ;
                System.Console.WriteLine("Sau khi khoi tao: bien = {0}", bien) ;
                // Gan gia tri cho bien
                bien = 5 ;
                // Xuat ra man hinh
                System.Console.WriteLine("Sau khi gan: bien = {0}", bien) ;
         }
}
```

II.3. Giá trị mặc định của biến

Nếu một biến khai báo và chưa có giá trị thì khi biên dịch sẽ phát sinh lỗi nếu bạn có tuy cập đến biến đó. Chẳng hạn, bạn khai báo biến k có kiểu là int rồi sau đó sử dụng phương thức Writeline của đối tượng Console để in giá trị của biến này trong dòng lệnh kế tiếp.

Ví dụ : Khởi tạo và gán giá trị một biến

```
class Bien
{
        static void Main()
         {
                // Khai bao va khoi tao bien
                int k ;
                System.Console.WriteLine("Sau khi khoi tao: bien = {0}", bien) ;
          }
}
```

II.4. Giá trị mặc định của biến kiểu Nullable

Trong trường hợp làm việc với kiểu dữ liệu Nullable, có thể khai báo biến với ký tự dấu hỏi (?)

Để gán một giá trị vào biến kiểu Nullable, sử dụng cú pháp tương tự như gán giá trị cho biến kiểu value.

II.5. Ký tự đặc biệt trong giá trị dạng chuỗi

Khi làm việc với biến có kiểu string, bạn có thể sử dụng ký tự @ để chỉ định các ký tự đặc biệt trong chuỗi.

II.6. Tầm vực của biến

Tầm vực của biến phụ thộc vào một trong từ khóa public, protected, internal, private bạn chỉ định trước tên biến. Mặc định không khai báo từ khóa trước biến nằm ngoài phương thức là private.

Lưu ý, các từ khóa public, protected, internal, private có thể sử dụng các phương thức hay hằng, enum.

II.6.1. Từ khóa public

Nếu biến khai báo có từ khóa là public thì không giới hạn quyền truy cập từ bên ngoài đến chúng.

II.6.2 từ khóa protected

Nếu sử dụng từ khóa protected trước biến thì giới hạn quyền truy cập trong class và những Class kế thừa từ Class chứa đựng biến đó.

II.6.3. Từ khóa internal

Giới hạn truy cập trong Assembly hiện hành.

Để truy cập vào biến là internal thì bản thân Class khi báo sử dụng Class chứa biến đó cùng chung một Assembly.

II.6.4. Từ khóa private

Từ khóa private ấn định biến cho phép truy cập chỉ bên trong Class khai báo nó. Ngoài ra, nếu biến khai báo bên trong Class không ấn định 1 trong 4 từ khóa public, private, protected hay internal thì mặc định là private.

II.6.5. Biến cục bộ

Biến cục bộ là biến có tầm vực ngay bên trong phương thức mà nó được khai báo

II.6.5.1 Từ khóa static

Từ khóa static sử dụng cho Class, phương thức(method), Constructor, thuộc tính (property), biến cố(event) và biến. Có thể tìm hiểu chi tiết về từ khóa static

dùng cho Class trong chương trình lớp, phương thức và thuộc tính. Trong phần này chúng ta tìm hiểu từ khóa static dùng khi khai báo biến.

III. HẰNG VÀ ENUM

- Hằng cũng là một biến nhưng giá trị của hằng không thay đổi trong khi thực hiện các lệnh của chương trình.
- Hằng được phân làm 3 loại:
  + Giá trị hằng (literal)
  + Biểu tượng hằng (symbolic constants)
  + Kiểu liệt kê (enumerations)

III.1. Hằng

Khai báo:

<const> <Kiểu_Dữ_Liệu> <tên_hằng> = <giá_trị> ;

Từ khóa const dùng để chỉ định khai báo trường hay một biến cục bộ với giá trị cho trước và không thể thay đổi trong quá trình thực thi.

Thông thường khai báo hằng khi giá trị nào đó không thay đổi

Ví dụ: x = 100; // 100 được gọi là giá trị hằng

Ví dụ : Nhập vào bán kính, in ra chu vi và diện tích hình tròn.

```
class HinhTron
{
        static void Main()
          {
                // Khai bao bieu tuong hang
                const double PI = 3.14159 ;
                // Khai bao bien
                int bankinh ;
                double chuvi , dientich ;
                string chuoi ;
                // Nhap gia tri cho bien chuoi
                System.Console.Write("Nhap ban kinh hinh tron: ") ;
                chuoi = System.Console.ReadLine() ;
                // Doi chuoi thanh so va gan vao bien so
                bankinh = System.Convert.ToInt32(chuoi) ;
                // Gan gia tri cho bien
                chuvi = 2 * bankinh * PI ;
                dientich = bankinh * bankinh * PI ;
                // Xuat ra man hinh
                System.Console.WriteLine("Chu vi hinh tron = {0:0.00}", chuvi) ;
                System.Console.WriteLine("Dien tich hinh tron = {0:0.00}", dientich) ;
         }
}
```

III.2. Kiểu liệt kê (Enum)

Kiểu liệt kê đơn giản là tập hợp các tên hằng có giá trị không thay đổi (thường được gọi là danh sách liệt kê), và được định nghĩa sau từ khóa enum.

Từ khóa enum sử dụng để định nghĩa một bảng dạng liệt kê(enumeration)

các hằng có kiểu cữ liệu khác nhau, bảng liệt kê này còn được gọi là enumerator list.

Mọi phần tử trong enumeration đều có kiểu dữ liệu, bạn có thể gán giá trị bất kỳ là kiểu số tổng quát (integral) ngoài trừ kiểu char. Nếu phần tử trong enumeration không khai báo kiểu thì kiểu dữ liệu mặc định là int.

Nếu không khai báo giá trị thì giá trị mặc định của phần tử thứ nhất là 0 các phần tử kế tiếp giá trị tăng lên 1.

Ví dụ : Sử dụng kiểu liệt kê.

```
---------------------------------------------------------------------------
using System;
class Vd24
{
    // Khai báo kiểu liệt kê
    enum NhietDoNuoc
    {DoDong=0, DoNguoi=20, DoAm=40, DoNong=60, DoSoi=100}
    static void Main()
    {
      Console.WriteLine("Nhiet do dong: {0}",NhietDoNuoc.DoDong);
      Console.WriteLine("Nhiet do nguoi:{0}", NhietDoNuoc.DoNguoi);
      Console.WriteLine("Nhiet do am: {0}", NhietDoNuoc.DoAm);
      Console.WriteLine("Nhiet do nong: {0}", NhietDoNuoc.DoNong);
      Console.WriteLine("Nhiet do soi: {0}", NhietDoNuoc.DoSoi);
    }
}
---------------------------------------------------------------------------
```

Kết quả:

```
Nhiet do dong: 0
Nhiet do nguoi: 20
Nhiet do am: 40
Nhiet do nong: 60
Nhiet do soi: 100
---------------------------------------------------------------------------
```

Chúng ta đã tìm hiểu về các kiểu dữ liệu, cách khai báo biến, tầm vực của biến và khai báo hằng, enum trong chương trình C#.

Trong phần kế tiếp, chúng ta tiếp tục tìm hiểu các phép toán sử dụng trong C# và các phương thức chuyển đổi kiểu dữ liệu.

IV. PHÉP TOÁN VÀ CHUYỂN ĐỔI KIỂU DỮ LIỆU

Chúng ta đã tìm hiểu các kiểu dữ liệu, cách khai báo biến, hằng, enum trong phần trước.

Trong phần này chúng ta tập trung tìm hiểu các phép toán sử dụng trong chương trình C# và cách khai chuyển đổi kiểu dữ liệu.

Các vấn đề chính sẽ được đề cập:

Phép toán.

Chuyển đổi kiểu dữ liệu.

Boxing và Unboxing.

Nhóm toán tử	Toán tử	Ý nghĩa
Toán học	+ - * / %	cộng , trừ, nhân chia, lấy phần dư
Logic	& | ^ ! ~ && || true false	phép toán logic và thao tác trên bit
Ghép chuỗi	+	ghép nối 2 chuỗi

Tăng, giảm	++, --	tăng / giảm toán hạng lên / xuống 1. Đứng trước hoặc sau toán hạng.
Dịch bit	<< >>	dịch trái, dịch phải
Quan hệ	== != < > <= >=	bằng, khác, nhỏ/lớn hơn, nhỏ/lớn hơn hoặc bằng
Gán	= += -= *= /= %= &= |= ^= <<= >>=	phép gán
Chỉ số	[]	cách truy xuất phần tử của mảng
Ép kiểu	()	
Indirection và Address	* -> [] &	dùng cho con trỏ

Bảng 2.3. *Các nhóm toán tử trong C#*

IV.1. PHÉP TOÁN

C# cung cấp tập các phép toán bằng các ký tự hiếm tượng trưng cho phép bạn thực hiện các biểu thức tính toán. Phép toán trong C# chia thành các nhóm như: phép toán số học (arithmetic), logical, gán, so sánh,…Chẳng hạn như: ==, ! =,<,>,<=,>=, binary +, binary -, ^, &, ~, + +,- - và sizeof().

Ngoài ra, một số phép toán có thể cài đặt chồng (overloading).

IV.1.1. Phép toán Arithmetic (số học)

Phép toán số học bao gồm 5 phép toán chính: + , - , * , / , % .

IV.1.2. Phép toán logic

C# cung cấp các phép toán logic ứng với các ký hiệu tượng trưng như: &, l, ^, !, ~, ll, true, false. Thường các phép toán này được sử dụng trong các phát biểu điều khiển.

! : phép phủ định, && : phép và, ll : phép hoặc.

IV.1.3. Phép toán tăng giảm

Phép toán tăng và giảm cho phép bạn tăng hoặc giảm giá trị hiện tại đi giá trị là 1.

Các phép toán tăng giảm : += , -= , *= , /= , %=

IV.1.4. Phép toán quan hệ

Phép toán quan hệ (Relational) bao gồm các phép toán như: == (bằng) , != (khác) , <, >, <=, >= thường sử dụng trong phát biểu điều khiển.

IV.1.5. Phép toán gán

Phép toán gán (Assignment) trong C# bao gồm: =, +=, -=, *=, /=, %=, &=, l=, ^=, <<=, ??. Trong đó, phép toán = dùng để gán giá trị cho biến hay biểu thức, +=, -=, *=, /=, %=, &=, l= dùng để gán tiếp giá trị sau một phép tính tương ứng cho biến. Chẳng hạn, += cho phép gán thêm giá trị sau phép cộng vào giá trị đang có.

Lưu ý: Trong chương trình trước chúng ta đã tìm hiểu kiểu dữ liệu Nullable, khi làm việc với biến có kiểu này bạn có thể sử dụng phép toán gán ?? để gán giá trị mặc định cho biến khi giá trị của biến đó là null.

IV.1.6. Phép toán chuyển kiểu dữ liệu

Phép toán chuyển kiểu dữ liệu () được sử dụng để ép kiểu dữ liệu hiện hành của đối tượng sang kiểu khác. Chẳng hạn, bạn khai báo biến có kiểu dữ liệu là object, sau khi gán giá trị cho biến này là 10, bạn có thể ép kiểu của biến này sang kiểu số nguyên.

IV.1.7. Phép toán điều kiện

Phép toán điều kiện ?: sử dụng trong trường hợp phát biểu điều khiển if .. else chỉ có một câu lệnh.

Toán tử 3 ngôi: (Điều_Kiện) ? (Biểu_Thức_1) : (Biểu_Thức_2) ;

IV.1.8. Phép toán định danh

Phép toán :: dùng để định danh ứng với nhận dạng cho không gian tên. Ví dụ, tạo Project rồi đặt tên AliasOperator, bên trong Project này bạn thêm Class và đặt tên System.

IV.1.9. Phép toán khởi tạo

Phép toán new dùng để khởi tạo đối tượng.

IV.1.10. Phép toán kiểm tra tràn số

Phép toán kiểm tra tràn số bao gồm hai phép toán chính là checked và unchecked. Trong đó, phép toán checked cho phép kiểm tra tràn số khi thực hiện các phép toán số học ứng với toán hạng là số hoặc chuyển đổi giữa chúng.

Phép toán unchecked dùng để chỉ thị phép toán hay biểu thức không cần kiểm tra tràn số trong quá trình thực hiện tính toán hay chuyển đổi.

IV.1.11. Phép toán khác

Ngoài các phép toán trên, bạn có thể sử dụng một số phép toán khác như: as, is, sizeof, typeof để kiểm tra loại đối tượng.

IV.1.11.1. Phép toán as

Phép toán as dùng để thực hiện việc chuyển đổi giữa hai kiểu dữ liệu dạng Reference.

IV.1.11.2. Phép toán is

Phép toán is kiểm tra đối tượng có giống như kiểu dữ liệu mà bạn muốn so sánh.

IV.1.11.3. Phép toán sizeof

Phép toán sizeof dùng để lấy kích thước bằng byte của kiểu Value (không cho phép sử dụng phép toán này cho kiểu Reference)

Ngoài ra, phép toán sizeof trả về giá trị là kiểu int chính là kích thước của kiểu Value truyền vào như tham số. Trong đó, các kiểu Value thường sử dụng có kích thước tương ứng như bảng 5-1.

Bảng 5-1: ***Phép toán sizeof***

Biểu thức	Kích thước
Sizeof(sbyte)	1
Sizeof (byte)	1
Sizeof(short)	2
Sizeof(ushort)	2
Sizeof(int)	4
Sizeof(uint)	4
Sizeof(long)	8

Sizeof(ulong)	8
Sizeof(char)	2 (Unicode)
Sizeof(float)	4
Sizeof(double)	8
Sizeof(bool)	1

IV.1.11.4. Phép toán typeof

Phép toán typeof dùng để lấy kiểu của đối tượng. Giá trị trả về là System.Type cho phép bạn liệt kê danh sách thuộc tính, phương thức của đối tượng đó.

Bảng : *Thứ tự ưu tiên của các nhóm toán tử (chiều ưu tiên từ trên xuống)*

Nhóm toán tử	Toán tử	Ý nghĩa
Primary (chính)	{x}x.y f(x) a[x] x++ ; x--	
Unary	+ - ! ~ ++x –x (T)x	
Nhân	* ; / ; %	Nhân, chia, lấy phần dư
Cộng	+ ; -	cộng, trù
Dịch bít	<< ; >>	Dịch trái, dịch phải
Quan hệ	< ; > ; <= ; >=; is	nhỏ hơn, lớn hơn, nhỏ hơn hay bằng, lớn hơn hay bằng và *là*
Bằng	==; !=	bằng, khác
Logic trên bit AND	&	*Và* trên bit.
XOR	^	Xor trên bit
OR	|	*hoặc* trên bit
Điều kiện AND	&&	*Và* trên biểu thức điều kiện
Điều kiện OR	||	*Hoặc* trên biểu thức điều
Điều kiện	?:	*điều kiện* tương tự *if*
Khởi gán	= ; *=; /=; %=; +=; -= ; <<=; =>>; &= ;^= ; |=	

IV.2. CHUYỂN ĐỔI KIỂU DỮ LIỆU

Chuyển đổi kiểu dữ liệu là việc thường làm trong bất kỳ ứng dụng nào khi làm việc với C#, bạn có thể chuyển đổi kiểu dữ liệu số với nhau với giới hạn cho phép. Chẳng hạn, chuyển đổi kiểu trong giới hạn cho phép giữa các kiểu gọi là Implicit Conversion.

Một đối tượng có thể chuyển từ kiểu này sang kiểu kia theo hai hình thức: ngầm định hoặc tường minh. Hình thức ngầm định được chuyển tự động còn hình thức tường minh cần sự can thiệp trực tiếp của người lập trình (giống với C++ và Java).

```
short x = 5;
int y ;
y = x; // chuyển kiểu ngầm định - tự động
```

```
x = y; // lỗi, không biên dịch được
x = (short) y; // OK
```

Bạn có thể tham khảo giới hạn cho phép chuyển đổi kiểu dữ liệu trong bảng 5-2

Bảng 5-2: *Chuyển đổi kiểu dữ liệu*

Từ kiểu dữ liệu	Sang kiểu dữ liệu
Sbyte	Short, int, long, float, double, decimal
Byte	Short, ushort, int, uint, long, ulong, float, double,
Short	decimal
Ushort	Int, long, float, double, decimal
Int	Int, uint, long, ulong, float, double, decimal
Uint	Long, float, double, decimal
Long	Long, ulong, float, double, decimal
Char	Float, double, decimal
Float	Ushort, int, uint, long, ulong, float, double, decimal
ulong	Double
	Float, double, decimal

Ngoài ra, bạn cũng có thể sử dụng chuyển đổi kiểu dữ liệu tương thích với nhau (Explicit Conversion), nếu không phù hợp thì lỗi sẽ phát sinh.Bảng 5-3 là danh sách của các kiểu dữ liệu có thể chuyển đổi.

Bảng 5-3: *Chuyển đổi kiểu dữ liệu*.

Từ kiểu dữ liệu	Sang kiểu dữ liệu
Sbyte	Byte, ushort, uint, ulong, char
Byte	Sbyte or char
Short	Sbyte, byte, ushort, uint, ulong, char
ushort	Sbyte, byte, short, char
int	Sbyte, byte, short, ushort, uint, ulong, char
uint	Sbyte, byte, short, ushort, int, char
long	Sbyte, byte, short, ushort, uint, int, ulong, char
ulong	Sbyte, byte, short, ushort, uint, int, long, char
char	Sbyte, byte, short
float	Sbyte, byte, short, ushort, uint, int, long, ulong, char, float, decimal
double	Sbyte, byte, short, ushort, uint, int, long, ulong, char, float, decimal
decimal	Sbyte, byte, short, ushort, uint, int, long, ulong, char, float, double

Chú ý, khi chuyển đổi kiểu dữ liệu có thể bị mất do làm tròn số hoặc tự cắt bỏ để thích hợp kiểu dữ liệu tương ứng trong quá trình chuyển đổi.

Để chuyển đổi kiểu dữ liệu này sang kiểu dữ liệu khác, bạn có thể sử dụng một số phương thức của đối tượng Convert.

IV.3. BOXING VÀ UNBOXING

Khái niệm Boxing và Unboxing cho phép kiểu Value được đối xử như một object. Boxing là đóng gói một kiểu giá trị thành đối tượng kiểu Reference, điều này cho phép bạn lưu giá trị trên vùng bộ nhớ quan trọng(heap).Trong khi đó, Unboxing trích dữ liệu từ kiểu object thành kiểu Value.

Chúng ta vừa tìm hiểu về các phép toán thường sử dụng trong chương trình C# và cách chuyển đổi kiểu dữ liệu bằng các phương thức của đối tượng Convert cùng với phương thức ToString của chính đối tượng đó.

V. TÊN, TỪ KHOÁ VÀ CHÚ THÍCH

V.1. Tên

Tên là một khái niệm rất quan trọng, nó dùng để xác định các đại lượng khác nhau trong một chương trình. Chúng ta có tên hằng, tên biến, tên mảng, tên hàm, tên phương thức, tên đối tượng, ...

Tên được đặt theo qui tắc sau: Tên bắt đầu bằng một chữ cái hoặc dấu gạch dưới, các kí tự còn lại phải là chữ cái, chữ số hoặc dấu gạch dưới.

✍*Ghi chú:* Không được đặt tên trùng với từ khóa của C#.

V.2. Từ khóa

Từ khóa là những từ có sẵn trong C#, bảng 2.5 giới thiệu một số từ khóa của C#

Bảng 2.5. *Từ khóa của ngôn ngữ C#*

abstract	default	foreach	object	sizeof	unsafe
as	delegate	goto	operator	stackalloc	Ushort
base	do	if	out	static	Using
bool	double	implicit	override	String	virtual
break	else	in	params	struct	volatile
byte	enum	int	private	switch	void
case	event	interface	protected	this	while
catch	explicit	internal	public	throw	
char	extern	is	readonly	True	
checked	false	lock	ref	Try	
class	finally	long	return	Typeof	
const	fixed	namespace	sbyte	Uint	
continue	float	new	sealed	Ulong	
decimal	For	null	short	unchecked	

V.3. Chú thích

- Lời giải thích trên một dòng: Đặt sau cặp kí hiệu //
- Lời giải thích trên nhiều dòng: Đặt lồng giữa cặp kí hiệu /* và */

VI. CÂU LỆNH TRONG C#

Trong C# một chỉ dẫn lập trình đầy đủ được gọi là câu lệnh. Chương trình bao gồm nhiều câu lệnh tuần tự với nhau. Mỗi câu lệnh phải kết thúc với một dấu chấm phẩy, ví dụ như:

```
int x; // một câu lệnh
x = 32; // câu lệnh khác
int y =x; // đây cũng là một câu lệnh
```

Những câu lệnh này sẽ được xử lý theo thứ tự. Đầu tiên trình biên dịch bắt đầu ở vị trí đầu của danh sách các câu lệnh và lần lượt đi từng câu lệnh

cho đến lệnh cuối cùng.

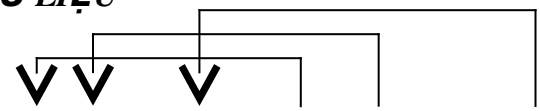
Có hai loại câu lệnh:

+ Câu lệnh đơn giản: lệnh khai báo, lệnh gán, lệnh gọi hàm.

+ Câu lệnh có cấu trúc: câu lệnh điều kiện, lệnh lặp, các lệnh đơn giản đặt trong cặp dấu {…}

VI.1 NHẬP VÀ XUẤT DỮ LIỆU

VI.1.1. Xuất dữ liệu



Cú pháp:

System.Console.Writeline("{0} {1} … {n}", bt_0, bt_1, …, bt_n);

Trong đó:

bt_0 truyền giá trị cho **{0}**

bt_1 truyền giá trị cho **{1}**

….

bt_n truyền giá trị cho **{n}**

VI.1.2. Nhập dữ liệu

Để nhập liệu cho các biến khác kiểu string, ta sử dụng cú pháp:

<Tên biến> = <Kiểu dữ liệu>.Parse(System.Console.ReadLine());

Để nhập liệu cho các biến kiểu string, ta sử dụng cú pháp003A

<Tên biến> = System.Console.ReadLine();

Ví dụ 2.5. Nhập xuất dữ liệu

```
using Sysstem;
class Vd25
{
 static void Main(string[] args)
 {
     Console.Clear();
     string ten;
     int tuoi;
     float dtb;
     Console.Write("Nhap ten: ");
     ten = Console.ReadLine();
     Console.Write("Nhap tuoi: ");
     tuoi = int.Parse(Console.ReadLine());
     Console.Write("Nhap dtb: ");
     dtb = float.Parse(Console.ReadLine());
     Console.WriteLine("----------------------------------------");
     Console.WriteLine("Du lieu sau khi nhap: ");
     Console.Write("Ten:{0}, Tuoi:{1}, DTB:{2:00.0}",ten,tuoi,dtb);
     Console.WriteLine("----------------------------------------");
     Console.ReadLine();
 }
}
```

VI.2. CẤU TRÚC LỰA CHỌN

VI.2.1. Cấu trúc if .. else …

Cú pháp:

```
if (<biểu_thức_logic>)
  <khối_lệnh>;
```

hoặc

```
if (<biểu_thức_logic>)
```

```
  <khối_lệnh_1>;
else
  <khối_lệnh 2>;
```

Trong đó:

- <biểu_thức_logic> là biểu thức cho giá trị true hoặc false.
- <khối_lệnh> là một lệnh đơn giản hoặc có thể là một lệnh có cấu trúc.
- Nếu <biểu_thức_logic> cho giá trị true thì <khối_lệnh> hay <khối_lệnh_1> sẽ được thực thi, ngược lại <khối_lệnh_2> sẽ thực thi.

✍ **Ghi chú**:

- Một điểm khác biệt với C++ là biểu thức trong câu lệnh if phải là biểu thức logic, không thể là biểu thức số.
- Ta có thể sử dụng cấu trúc if lồng nhau

VI.2.2. Cấu trúc switch

Cú pháp:

```
switch (<biểu_thức_lựa_chọn>)
{
  case <biểu_thức_hằng_1>:
       [<khối_lệnh_1>]; [break];
  case <biểu_thức_hằng_2>:
       [<khối_lệnh_2>]; [break];
  …
  case <biểu_thức_hằng_n>:
       [<khối_lệnh_n>];[break];
  [default: <khối_lệnh_n+1>;]
}
```

Trong đó: <biểu_thức_lựa_chọn> là biểu thức sinh ra trị nguyên hay chuỗi, switch sẽ so sánh <biểu_thức_lựa_chọn> với các <biểu_thức_hằng_?> để biết phải thực hiện với <khối_lệnh_?> tương ứng, lệnh break để thoát khỏi cấu trúc switch và bắt buộc phải có.

VI.3. CẤU TRÚC LẶP

VI.3.1. Cấu trúc lặp while

Cú pháp:

```
while (<biểu_thức_logic>)
   <khối_lệnh>;
```

Mô tả quá trình thực hiện trong cấu trúc lặp while:

Nếu <biểu_thức_logic> vẫn còn đúng thì <khối_lệnh> sẽ được thực hiện, ngược lại sẽ thoát khỏi vòng lặp.

VI.3.2. Cấu trúc lặp do … while

Cú pháp:

```
do
  <khối_lệnh>
while (<biểu_thức_logic>)
```

Mô tả quá trình thực hiện trong cấu trúc lặp do … while:

Trong cấu trúc lặp do … while, <khối_lệnh> sẽ được thực hiện trước, sau đó <biểu_thức_logic> được kiểm tra, nếu <biểu_thức_logic> đúng <khối_lệnh> lại được thực hiện, ngược lại sẽ thoát khỏi vòng lặp.

VI.3.3. Cấu trúc lặp for

Cú pháp:

```
for ([<bt1>]; [<bt2>]; [<bt3>])
    <khối_lệnh>;
```

Trong đó:

- <bt1> là biểu thức khởi tạo biến điều khiển
- <bt2> là biểu thức logic
- <bt3> là biểu thức thay đổi biến điều khiển

Mô tả quá trình thực hiện trong cấu trúc lặp for

Bước 1: <bt1> được thực hiện.

Bước 2: <bt2> được thực hiện. Nếu <bt2> có giá trị đúng thì qua Bước 3, ngược lại qua Bước 5.

Bước 3: <khối_lệnh> được thực hiện

Bước 4: <bt3> được thực hiện

Bước 5: Thoát

VI.3.4. Cấu trúc lặp foreach

Cú pháp:

```
foreach (<kiểu_dữ_liệu> <tên_biến> in <tập_hợp>)
    <khối_lệnh>;
```

Trong đó: <tập_hợp> chứa các phần tử cùng kiểu dữ liệu với <tên_biến>

Mô tả quá trình thực hiện trong cấu trúc lặp for

Lần lượt các phần tử trong <tập_hợp> sẽ được gán cho <tên_biến>, sau mỗi lần gán, <khối_lệnh> sẽ được thực thi. Quá trình kết thúc khi tất cả các phần tử trong <tập_hợp> đã được duyệt.

VI.3. KIỂM SOÁT NGOẠI LỆ

Ngôn ngữ C# cũng giống như bất cứ ngôn ngữ hướng đối tượng khác, cho phép xử lý những lỗi và các điều kiện không bình thường với những ngoại lệ. Ngoại lệ là một đối tượng đóng gói những thông tin về sự cố của một chương trình không bình thường.

Một điều quan trọng để phân chia giữa bug, lỗi, và ngoại lệ. Một bug là một lỗi lập trình có thể được sửa chữa trước khi mã nguồn được chuyển giao. Những ngoại lệ thì không được bảo vệ và tương phản với những bug. Mặc dù một bug có thể là nguyên nhân sinh ra ngoại lệ, chúng ta cũng không dựa vào những ngoại lệ để xử lý những bug trong chương trình, tốt hơn là chúng ta nên sửa chữa những bug này.

Một lỗi có nguyên nhân là do phía hành động của người sử dụng. Ví dụ, người sử dụng nhập vào một số nhưng họ lại nhập vào ký tự chữ cái. Một lần nữa, lỗi có thể làm xuất hiện ngoại lệ, nhưng chúng ta có thể ngăn ngừa điều này bằng cách bắt giữ lỗi với mã hợp lệ. Những lỗi có thể được đoán trước và được ngăn ngừa.

Thậm chí nếu chúng ta xóa tất cả những bug và dự đoán tất cả các lỗi của người dùng, chúng ta cũng có thể gặp phải những vấn đề không mong đợi, như là xuất hiện trạng thái thiếu bộ nhớ (out of memory), thiếu tài nguyên hệ thống,... những nguyên nhân này có thể do các chương trình khác cùng hoạt động ảnh hưởng đến. Chúng ta không thể ngăn ngừa các ngoại lệ này, nhưng chúng ta có thể xử lý chúng để chúng không thể làm tổn hại đến chương trình.

Khi một chương trình gặp một tình huống ngoại lệ, như là thiếu bộ nhớ thì nó sẽ tạo một ngoại lệ. Khi một ngoại lệ được tạo ra, việc thực thi của các chức năng hiện hành sẽ bị treo cho đến khi nào việc xử lý ngoại lệ tương ứng được tìm thấy.

Điều này có nghĩa rằng nếu chức năng hoạt động hiện hành không thực hiện việc xử lý ngoại lệ, thì chức năng này sẽ bị chấm dứt và hàm gọi sẽ nhận sự thay đổi đến việc xử lý ngoại lệ. Nếu hàm gọi này không thực hiện việc xử lý ngoại lệ, ngoại lệ sẽ được xử lý sớm bởi CLR, điều này dẫn đến chương trình của chúng ta sẽ kết thúc.

Một trình xử lý ngoại lệ là một khối lệnh chương trình được thiết kế xử lý các ngoại lệ mà chương trình phát sinh. Xử lý ngoại lệ được thực thi trong trong câu lệnh catch. Một cách lý tưởng đó là nếu một ngoại lệ được bắt và được xử lý, thì chương trình có thể sửa chữa được vấn đề và tiếp tục hoạt động bình thường. Thậm chí nếu chương trình không tiếp tục, bằng việc bắt giữ ngoại lệ chúng ta có cơ hội để in ra những thông điệp có ý nghĩa và kết thúc chương trình một cách rõ ràng.

Nếu đoạn chương trình thực hiện mà không cần chú ý đến bất cứ ngoại lệ nào (chẳng hạn như khi giải phóng tài nguyên mà chương trình được cấp phát), thì ta có thể đặt đoạn mã này trong khối finally, khi đó, đảm bảo chương trình sẽ luôn luôn thực hiện, thậm chí ngay cả khi có một ngoại lệ xuất hiện.

VI.3.1. Phát sinh và bắt giữ ngoại lệ

Trong ngôn ngữ C#, chúng ta thể tạo ra những đối tượng có kiểu dữ liệu để xử lý ngoại lệ, đó là System.Exception, hay những đối tượng được dẫn xuất từ kiểu dữ liệu này. Namespace System của CLR chứa một số các kiểu dữ liệu xử lý ngoại lệ mà chúng ta có thể sử dụng trong chương trình. Những kiểu dữ liệu ngoại lệ này bao gồm ArgumentNull-Exception, InValidCastException, và OverflowException, cũng như nhiều lớp khác nữa.

VI.3.1.1. Câu lệnh throw

Trong C#, để phát tín hiệu báo hiệu một sự việc không bình thường trong một lớp, ta cần phải phát sinh một ngoại lệ. Để làm được điều này, ta sử dụng từ khóa throw. Dòng lệnh sau tạo ra một thể hiện mới của ngoại lệ và sau đó ném nó:

```
throw new System.Exception();
```

Khi phát sinh ngoại lệ, ngay tức khắc chương trình sẽ ngừng thực thi, trong khi CLR sẽ tìm kiếm một trình xử lý ngoại lệ. Nếu trình xử lý ngoại lệ không được tìm thấy trong phương thức hiện thời, thì CLR tiếp tục tìm trong phương thức gọi cho đến khi nào tìm thấy. Nếu CLR trả về lớp Main() mà

không tìm thấy bất cứ trình xử lý ngoại lệ nào, thì nó sẽ kết thúc chương trình.
Ví dụ 3.1. phát sinh ngoại lệ

```
---------------------------------------------------------------------------
using System;
namespace Programing_CSharp
{
 class Vd31
 {
 static void Main()
 {
      Console.WriteLine("Enter Main...");
      Vd31 t = new Vd31();
      t.Func1();
      Console.WriteLine("Exit Main...");
      Console.ReadLine();
 }
 public void Func1()
 {
      Console.WriteLine("Enter Func1...");
      Func2();
      Console.WriteLine("Exit Func1...");
 }
 public void Func2()
 {
      Console.WriteLine("Enter Func2...");
      throw new System.Exception();
      Console.WriteLine("Exit Func2...");
 }
 }
}
---------------------------------------------------------------------------
```

Kết quả:

```
Enter Main…
Enter Func1…
Enter Func2…

Unhandled Exception: System.Exception: Exception of type
‘System.Exception’ was thrown.
      at Programming_CSharp.Test.Func2() in ... exception01.cs:line
23
      at Programt4t4ming_CSharp.Test.Func1() in ...
exception01.cs:line 17
      at Programming_CSharp.Test.Main() in ... exception01.cs:line 10
---------------------------------------------------------------------------
```

Hàm Fun2() phát sinh ngoại lệ tại dòng 23, do trong Func2() không có *trình xử lý ngoại lệ* cho nên Func2() bị lỗi; dẫn đến Func1() lỗi tại dòng 17; và Main() cũng bị lỗi tại dòng 10.

VI.3.1.2. Câu lệnh catch

Trong C#, một trình xử lý ngoại lệ hay một đoạn chương trình xử lý các ngoại lệ được gọi là một khối catch và được tạo ra với từ khóa catch.
Ví dụ 3.2. Xử lý ngoại lệ

```
---------------------------------------------------------------------------
using System;
```

```
namespace Programing_CSharp
{
 class Vd32
 {
 static void Main()
 {
      Console.WriteLine("Enter Main...");
      Vd32 t = new Vd32();
      t.Func1();
      Console.WriteLine("Exit Main...");
      Console.ReadLine();
 }
 public void Func1()
 {
      Console.WriteLine("Enter Func1...");
      Func2();
      Console.WriteLine("Exit Func1...");
 }
 public void Func2()
 {
      try
      {
           Console.WriteLine("Enter Func2...");
           throw new System.Exception();
           Console.WriteLine("Exit Func2...");
      }
      catch
      {
           Console.WriteLine("Exception caught and handled.");
      }
 }
 }
}
```

Kết quả:

```
Enter Main…
Enter Func1…
Enter Func2…
Exception caught and handled.
Exit Func1…
Exit Main…
```

Ví dụ này giống ví dụ 3.1, ngoại trừ chương trình thêm vào trong hàm Func3() một khối try/catch. Thông thường chúng ta cũng có thể đặt khối try bao quanh những đoạn chương trình tiềm ẩn gây ra sự nguy hiểm, như là việc truy cập file, cấp phát bộ nhớ... Theo sau khối try là câu lệnh catch tổng quát. Câu lệnh catch trong ví dụ này là tổng quát bởi vì chúng ta không xác định loại ngoại lệ nào mà chúng ta bắt giữ. Trong trường hợp tổng quát này thì khối catch này sẽ bắt giữ bất cứ ngoại lệ nào được phát sinh.

Ví dụ 3.3. Sử dụng câu lệnh catch để bắt giữ ngoại lệ không xác định

---

```
using System;
namespace Programming_CSharp
{
```

```
 class Vd33
 {
      public static void Main()
      {
            double x;
            Console.Write(“Nhap so: “);
            try
            {
                  x=double.Parse(Console.ReadLine());
                  Console.Write(“So vua nhap: {0}”,x);
            }
            catch
            {
                  Console.Write(“Ban vua nhap gia tri khong phai
so.”);
            }
            Console.ReadLine();
      }
}
```

Cách khác:

Ví dụ 3.4. Sử dụng phương thức TryParse để xác định lỗi

```
---------------------------------------------------------------------------
using System;
namespace Programming_CSharp
{
 class Vd34
 {
      public static void Main()
      {
            double x;
            string y;
            Console.Write(“Nhap so: “); y=Console.ReadLine();
            if(double.TryParse(y, out x))
                  Console.Write(“So vua nhap: {0}”,x);
            else
                  Console.Write(“Ban vua nhap gia tri khong phai
so.”);
            Console.ReadLine();
      }
}}
```

Ví dụ 3.5. Sử dụng câu lệnh catch để bắt giữ ngoại lệ xác định

```
---------------------------------------------------------------------------
using System;
namespace Programming_CSharp
{
 class Vd35
 {
      public static void Main()
      {
            Vd35 t = new Vd35();
            t.TestFunc();
      }
 // ta thử chia hai phần xử lý ngoại lệ riêng
      public void TestFunc()
      {
            try
            {
```

```
                double a = 5;
                double b = 0;
                Console.WriteLine("{0} / {1} = {2}", a, b,
                     DoDivide(a,b));
           }
           catch (System.DivideByZeroException)
           {
                Console.WriteLine("DivideByZeroException caught!");
           }
           catch (System.ArithmeticException)
           {
                Console.WriteLine("ArithmeticException caught!");
           }
           catch
           {
                Console.WriteLine("Unknown exception caught");
           }
     }
     // thực hiện phép chia hợp lệ
     public double DoDivide(double a, double b)
     {
           if(b == 0) throw new System.DivideByZeroException();
           if(a == 0) throw new System.ArithmeticException();
           return a/b;
     }
 }
}
```

: *Kết quả:*

```
DivideByZeroException caught!
```

Trong ví dụ này, phương thức DoDivide() sẽ không cho phép chúng ta chia cho zero bởi một số khác, và cũng không cho phép chia số zero. Nó sẽ phát sinh một ngoại lệ **DivideByzeroException** nếu chúng ta thực hiện chia với zero. Trong toán học việc lấy zero chia cho một số khác là được phép, nhưng trong ví dụ minh họa của chúng ta không cho phép thực hiện việc này, nếu thực hiện sẽ phát sinh ra một ngoại lệ **ArithmeticException**.

Khi một ngoại lệ được phát sinh, CLR sẽ kiểm tra mỗi khối xử lý ngoại lệ theo thứ tự và sẽ lấy khối đầu tiên thích hợp. Khi chúng ta thực hiện với a=5 và b=7 thì kết quả như sau:

5 / 7 = 0.7142857142857143

Như chúng ta mong muốn, không có ngoại lệ được phát sinh. Tuy nhiên, khi chúng ta thay đổi giá trị của a là 0, thì kết quả là:

```
ArithmeticException caught!
```

Ngoại lệ được phát sinh, và CLR sẽ kiểm tra ngoại lệ đầu tiên: **DivideByZeroException**. Bởi vì không phù hợp, nên nó sẽ tiếp tục đi tìm và khối xử lý ArithmeticException được chọn. Cuối cùng, giả sử chúng ta thay đổi giá trị của b là 0. Khi thực hiện điều này sẽ dẫn đến ngoại lệ **DivideByZeroException**.

@Ghi chú: Chúng ta phải cẩn thận thứ tự của câu lệnh catch, bởi vì

DivideByZero-Exception được dẫn xuất từ **ArithmeticException**. Nếu chúng ta đảo thứ tự của câu lệnh catch, thì ngoại lệ **DivideByZeroException** sẽ phù hợp với khối xử lý ngoại lệ **ArithmeticException**. Và việc xử lý ngoại lệ sẽ không bao giờ được giao cho khối xử lý **DivideByZeroException**. Thật vậy, nếu thứ tự này được đảo, nó sẽ không cho phép bất cứ ngoại lệ nào được xử lý bởi khối xử lý ngoại lệ **DivideByZeroException**. Trình biên dịch sẽ nhận ra rằng **DivideByZeroException** không được thực hiện bất cứ khi nào và nó sẽ thông báo một lỗi biên dịch.

Chúng ta có thể phân phối câu lệnh try/catch, bằng cách bắt giữ những ngoại lệ xác định trong một hàm và nhiều ngoại lệ tổng quát trong nhiều hàm. Mục đích của thực hiện này là đưa ra các thiết kế đúng. Giả sử chúng ta có phương thức A, phương thức này gọi một phương thức khác tên là phương thức B, đến lượt mình phương thức B gọi phương thức C. Và phương thức C tiếp tục gọi phương thức D, cuối cùng phương thức D gọi phương thức E. Phương thức E ở mức độ sâu nhất trong chương trình của chúng ta, phương thức A, B ở mức độ cao hơn. Nếu chúng ta đoán trước phương thức E có thể phát sinh ra ngoại lệ, chúng ta có thể tạo ra khối try/catch để bắt giữ những ngoại lệ này ở chỗ gần nơi phát sinh ra ngoại lệ nhất. Chúng ta cũng có thể tạo ra nhiều khối xử lý ngoại lệ chung ở trong đoạn chương trình ở mức cao trong trường hợp những ngoại lệ không đoán trước được.

VI.3.1.3. Câu lệnh finally

Khi mở một tập tin để làm công việc A, có 2 trường hợp xảy ra:

- Trường hợp gặp ngoại lệ phát sinh trước khi làm công việc A: ta giải quyết công việc A trong lệnh catch.
- Trường hợp không phát sinh ngoại lệ: ta giải quyết công việc A trong lệnh try.

Do vậy, người lập trình cần viết công việc A cả trong try lẫn trong catch, và điều này làm chương trình mất vẻ thẩm mĩ. Câu lệnh finally giải quyết công việc này.

Ví dụ 3.6. Sử dụng câu lệnh finally

```
---------------------------------------------------------------------------
using System;
namespace Programming_CSharp
{
 class Vd36
 {
      static void Main(string[] args)
      {
            Vd36 t = new Vd36();
            t.TestFunc();
            Console.ReadLine();
      }
      // ta thử chia hai phần xử lý ngoại lệ riêng
      public void TestFunc()
      {
            try
            {
```

```
                Console.WriteLine("Open file here");
                double a = 5;
                double b = 0;
                Console.WriteLine("{0}/{1} = {2}",a,b,DoDivide(a,
b));
                Console.WriteLine("This line may or not print");
           }
           catch (System.DivideByZeroException)
           {
                Console.WriteLine("DivideByZeroException caught!");
           }
           catch
           {
                Console.WriteLine("Unknown exception caught");
           }
           finally
           {
                Console.WriteLine("Close file here");
           }
      }
      // thực hiện phép chia hợp lệ
      public double DoDivide(double a, double b)
      {
           if (b == 0) throw new System.DivideByZeroException();
           if (a == 0) throw new System.ArithmeticException();
           return a / b;
      }
 }
}
```

: Kết quả:

```
Open file here
DivideByZeroException caught!
Close file here
```

Ví dụ 3.7.

```
using System;
namespace Programming_CSharp
{
 class Vd37
 {
      static void Main(string[] args)
      {
            float n,m;
            Console.Write("Nhap n: ");
            n = int.Parse(Console.ReadLine());
            Console.Write("Nhap m: ");
            m = int.Parse(Console.ReadLine());
            try
            {
                if (m == 0) throw new
System.DivideByZeroException();
                Console.Write("Thuong: {0}", n / m);
            }
            catch//(System.DivideByZeroException)
            {
```

```
                Console.Write("so chia la zero");
            }
            Console.ReadLine();
        }
```

VI.3.2. Những đối tượng ngoại lệ

Trong phần này chúng ta sẽ tiến hành việc tìm hiểu các đối tượng được xây dựng cho việc xử lý ngoại lệ. Đối tượng **System.Exception** cung cấp một số các phương thức và thuộc tính hữu dụng.

- Thuộc tính **Message** cung cấp thông tin về ngoại lệ, như là lý do tại sao ngoại lệ được phát sinh. Thuộc tính **Message** là thuộc tính chỉ đọc, đoạn chương trình phát sinh ngoại lệ có thể thiết lập thuộc tính **Message** như là một đối mục cho bộ khởi dựng của ngoại lệ.

- Thuộc tính **HelpLink** cung cấp một liên kết để trợ giúp cho các tập tin liên quan đến các ngoại lệ. Đây là thuộc tính chỉ đọc.

- Thuộc tính **Source** đưa ra tên của ứng dụng hoặc đối tượng gây lỗi.

Ví dụ 3.8. Sử dụng thuộc tính Message

```
using System;
class Vd38
{
  static void Main(string[] args)
  {
    try
    {
      float x;
      Console.Write("Nhap du lieu cho x: ");
      x = float.Parse(Console.ReadLine());
    }
    catch(System.Exception teo)
    {
      Console.WriteLine(teo.Message);
    }
    Console.ReadLine();
  }
}
```

Khi nhập dữ liệu cho biến x, ta cố tình nhập vào một chuỗi, lúc đó sẽ hiện thông báo:

Input string was not in a correct format

Bảng 3.1. Các ngoại lệ thường gặp

CÁC LỚP NGOẠI LỆ	
Tên ngoại lệ	**Mô tả**
MethodAccessException	Lỗi truy cập, do truy cập đến thành viên hay phương thức không được truy cập
ArgumentException	Lỗi tham số đối mục
ArgumentNullException	Đối mục Null, phương thức được truyền đối mục null không được chấp nhận
ArithmeticException	Lỗi liên quan đến các phép toán
ArrayTypeMismatchException	Kiểu mảng không hợp, khi cố lưu trữ kiểu không thích hợp vào mảng
DivideByZeroException	Lỗi chia zero

FormatException	Định dạng không chính xác một đối mục nào đó
IndexOutOfRangeException	Chỉ số truy cập mảng không hợp lệ, dùng nhỏ hơn chỉ số nhỏ nhất hay lớn hơn chỉ số lớn nhất của mảng
InvalidCastException	Phép gán không hợp lệ
MulticastNotSupportedException	Multicast không được hỗ trợ, do việc kết hợp hai delegate không đúng
NotFiniteNumberException	Không phải số hữu hạn, số không hợp lệ
NotSupportedException	Phương thức không hỗ trợ, khi gọi một phương thức không tồn tại bên trong lớp.
NullReferenceException	Tham chiếu null không hợp lệ.
OutOfMemoryException	Out of memory
OverflowException	Lỗi tràn phép toán
StackOverflowException	Tràn stack
TypeInitializationException	Kiểu khởi tạo sai, khi bộ khởi dựng tĩnh có lỗi.

VI.4. MẢNG

VI.4.1. Định nghĩa mảng

Mảng là một tập hợp các phần tử có cùng kiểu dữ liệu, các phần tử của mảng được đặt chung bằng một tên và mỗi phần tử được xác định theo chỉ mục.

VI.4.2. Mảng một chiều

VI.4.2.1. Khai báo

<kiểu dữ liệu>[] <tên mảng>;

Ví dụ: int[] myIntArray;

Button[] myButtonArray;

Sau khi khai báo mảng, ta cần cấp phát số phần tử cho mảng theo mẫu:

<tên mảng> = new <kiểu dữ liệu>[số phần tử];

Khi cấp phát số phần tử cho mảng xong, thì giá trị mặc định cho mỗi phần tử trong mảng là null.

Ví dụ: int[] myIntArray;

myIntArray=new int[5];//cấp 5 ptử cho mảng myIntArray

Button[] myButtonArray = new Button[3];

VI.4.1.2. Truy cập đến các phần tử trong mảng

Để truy cập đến những phần tử trong mảng, ta sử dụng toán tử lấy chỉ mục []. Cũng giống như C/C++, chỉ mục mảng được tính bắt đầu từ phần tử 0.

Thuộc tính Length của lớp Array cho biết được kích thước một mảng. Như vậy chỉ mục của mảng đi từ 0 đến Length - 1.

Ví dụ 3.6. Tính tổng các phần tử trên mảng một chiều

```
using System;
namespace Programming_CSharp
{
 class Vd39
 {
        static void Main()
        {
```

```
            int[] A = new int[5] {4,2,6,9,1};
            int S = 0;
            for (int i = 0; i < A.Length; i++)S += A[i];
            Console.WriteLine("Tong: {0}",S);
            Console.ReadLine();
        }
 }
}
```

Kết quả

Tong: 22

Ta có thể thay câu lệnh for trong ví dụ 3.9

```
            for (int i = 0; i < A.Length; i++)S += A[i];
```

bằng câu lệnh foreach, kết quả không đổi

```
foreach (int x in A)S += x;
```

VI.4.3. Mảng hai chiều

VI.4.3.1. Khai báo

```
            <kiểu dữ liệu>[,] <tên mảng>;
```

Ví dụ: `int[,] A;`

Sau khi khai báo mảng, ta cần cấp phát số phần tử cho mảng theo mẫu:

```
            <tên mảng> = new <kiểu dữ liệu>[số hàng, số cột];
```

Khi cấp phát số phần tử cho mảng xong, thì giá trị mặc định cho mỗi phần tử trong mảng là null.

Ví dụ:

```
int[,] A;
 A = new int[2,3];//Mảng A có 2x3 ptử có gtrị null
```

Ta có thể nhập dữ liệu cho mảng từ bàn phím, hoặc có thể gán sẵn dữ liệu lúc khai báo, ví dụ:

```
            int[,] A = {{1, 4, 2},{3, 6, 9} };
                    //A[0,0]=1, A[0,1]=4, …
```

Hoặc

```
            int[,] A;
            A = new int[2,3] {{1, 4, 2},{3, 6, 9} };
```

VI.4.3.2. Truy cập đến các phần tử trong mảng

Để truy cập đến những phần tử trong mảng, ta sử dụng toán tử lấy chỉ mục [,]. Cũng giống như C/C++, chỉ mục mảng được tính bắt đầu từ phần tử [0,0].

B. CÂU HỎI VÀ BÀI TẬP

Viết chương trình khai báo một mảng một chiều chứa các số nguyên, nhập dữ liệu cho mảng và in kết quả vừa nhập.

1. Nhập vào vào 2 số nguyên, sau đó tính tổng, hiệu, tích, thương của nó?
2. Tìm lỗi của chương trình sau. Sửa lỗi và biên dịch lại chương trình.

```
using System;
class BaiTap3_3
{
public static void Main()
```

```
{
double myDouble; decimal myDecimal;
myDouble = 3.14; myDecimal = 3.14;
Console.WriteLine("My Double: {0}", myDouble);
Console.WriteLine("My Decimal: {0}", myDecimal);
}
}
```

3. Viết chương trình hiển thị ra màn hình 3 kiểu sau:

*	$ $ $ $ $	*
* *	$ $ $ $	* * *
* * *	$ $ $	* * * * *
* * * *	$ $	* * * * * * *
* * * * *	$	* * * * * * * * *
a.	b.	c.

4. Viết chương trình giải phương trình bậc nhất.
5. Viết chương trình giải phương trình bậc hai.
6. Viết chương trình nhập vào 3 cạnh tam giác, sau đó kiểm tra tam giác này thuộc loại gì? (đều, cân, vuông cân, vuông, thường).
7. Viết chương trình in ra bảng cửu chương.
8. Hãy viết đoạn lệnh để thực hiện việc bắt giữa ngoại lệ liên quan đến câu lệnh sau đây: Ketqua = Sothu1 / Sothu2;
9. Chương trình sau đây có vấn đề. Hãy xác định vấn đề có thể phát sinh ngoại lệ khi chạy chương trình. Và viết lại chương trình hoàn chỉnh gồm các lệnh xử lý ngoại lệ:

```
using System;
class Tester
{
        public static void Main()
        {
                uint so1=10;
                int so2, so3;
                so2 = -10; so3 = 0;
                //tính giá trị lại
                so1 -= 5;
                so2 = 5/so3;
                //xuất kết quả
                Console.Writeline("So 1: {0}, So 2: {1}", so1, so2);
        }
}
```

10. Viết chương trình tạo mảng số nguyên một chiều có n phần tử chứa giá trị ngẫu nhiên. Sắp xếp các thành phần trong mảng theo thứ tự tăng dần và hiển thị kết quả (Làm tương tự với trường hợp sắp xếp mảng theo thứ tự giảm dần).

Hướng dẫn:

```
int[] A;
int n;
Console.Write("Nhap n: ");
```

```
n = int.Parse(Console.ReadLine());
A = new int[n];
Random x = new Random();
for(int i=0; i<n; i++) A[i] = x.Next(10);
….
```

11. Viết một chương trình tạo mảng số nguyên một chiều có n phần tử chứa giá trị ngẫu nhiên. Sắp xếp chúng theo thứ tự số âm thì tăng còn số dương thì giảm dần. Hiển thị kết quả ra màn hình
12. Viết chương trình tìm số lớn nhất và nhỏ nhất trong mảng hai chiều có kích thước cố định. Các thành phần của mảng được phát sinh ngẫu nhiên.
13. Viết chương trình cộng hai ma trận nxm, tức là mảng hai chiều có kích thước n dòng, m cột. Các giá trị của hai mảng phát sinh ngẫu nhiên, cho biết kết quả cộng hai ma trận?
14. Viết chương trình cho phép người dùng nhập vào một ma trận nxm, sao đó tìm kiếm một giá trị nào đó theo yêu cầu người dùng, kết quả của việc tìm kiếm là giá trị và thứ tự của giá trị tìm được trong ma trận.
15. Viết chương trình tạo một mảng hai chiều không cùng kích thước. Cố định số dòng của mảng là 5, còn từng dòng có kích thước bằng giá trị của dòng, tức là dòng thứ nhất có kích thước 1 (tức là có 1 cột), dòng thứ hai có kích thước là 2 (tức là 2 cột).... Các giá trị phát sinh ngẫu nhiên. Hãy xuất kết quả của ma trận theo kiểu sau:
a[i][j] = <giá trị aij>
...
Việc xuất kết quả của ma trận trên có thể thực hiện bằng vòng lặp foreach được không? Nếu được thì hãy viết đoạn chương trình xuất ra kết quả?

MÃ BÀI HỌC LTTQ – 05	**BÀI 5 : CÁC ĐỐI TƯỢNG ĐIỀU KHIỂN CỦA C#.**	Thời gian (giờ)				
		LT	TH	BT	KT	TS
		3	2			5

Mục tiêu:

Sau khi học xong bài này, học viên có khả năng:

- Mô tả được các thành phần chính của giao diện Visual C#.net
- Có khả năng quản lý được các đối tượng và lập trình trên các đối tượng.
- Thực hiện các thao tác an toàn với máy tính.

TÓM TẮT BÀI:

ứng dụng thường có hai phần chính, phần giao diện và mã chương trình, phần giao diện là màn hình Form chứa các Control cho phép người sử dụng thao tác, chỉ thị, nhập liệu và trình bày dữ liệu phục vụ cho nhu cầu kinh doanh. Trong khi đó, mã chương trình là tập lệnh bao gồm hai phần chính là khai báo và chương trình con, khi làm việc với C# 2008, chúng ta sẽ tìm hiểu về các loại Form, Control, không gian tên cho phép tạo nên ứng dụng chạy trên máy để bàn (còn gọi là Desktop Application).

- Các vấn đề chính sẽ được đề cập
- Ứng dụng Windows Forms.
- Không gian tên, thực đơn Project, Thanh công cụ, Định dạng mã C#.
- Các loại Forms, thuộc tính, biến cố và phương thức của Forms.
- Các điều khiển thông dụng và đặc biệt trong C# 2008.
- Điều khiển dùng để xây dựng Menu.
- Điều khiển chứa đựng điều khiển khác.
- Điều khiển DIALOG và phương thức MESSAGEBOX.
- Điều khiển Dialog và phương thức Messagebox.
- Làm việc với điều khiển in ấn.
- Điều khiển do người dùng tạo ra.

NỘI DUNG :

A. LÝ THUYẾT

I. GIỚI THIỆU

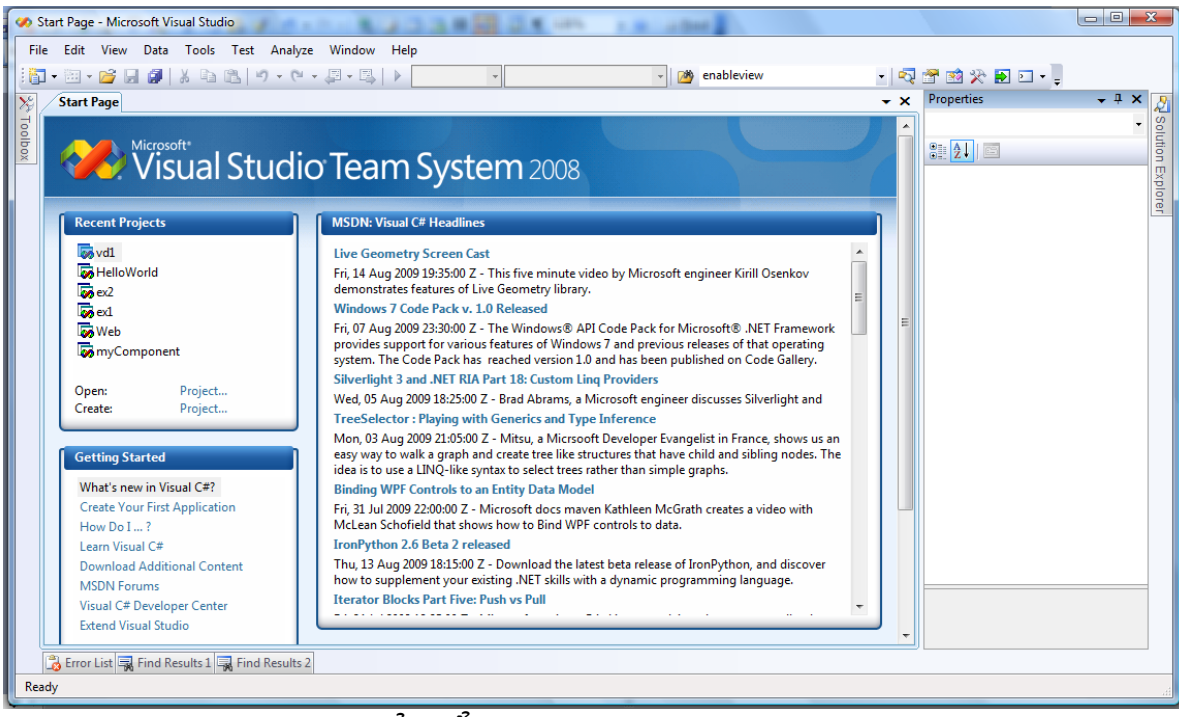
Bộ Visual Studio .Net gồm nhiều ngôn ngữ lập trình khác nhau như Visual Studio C#, Visual Studio C++.Net, Visual Basic .Net, bên cạnh đó, bộ Visual Studio .Net còn là môi trường phát triển tích hợp IDE gồm nhiều cửa sổ thiết kế và cửa sổ công cụ; ngoài ra, bộ Visual Studio .Net còn cung cấp hệ thống giúp đỡ tích hợp.

Chương này, ta chỉ nghiên cứu Visual Studio C#. Trong Visual Studio C# có rất nhiều loại dự án, ở đây ta chỉ tập trung nghiên cứu các dự án triển khai trên Window, cụ thể là Windows Forms Application.

Để tạo mới một dự án trên Windows Forms Application, ta thực hiện theo trình tự sau:

Bước 1. Vào Start menu → Programs → Microsoft Visual Studio 2008 → Microsoft Visual Studio 2008.

Xuất hiện cửa sổ Start Page – Microsoft Visual Studio như hình 5.1

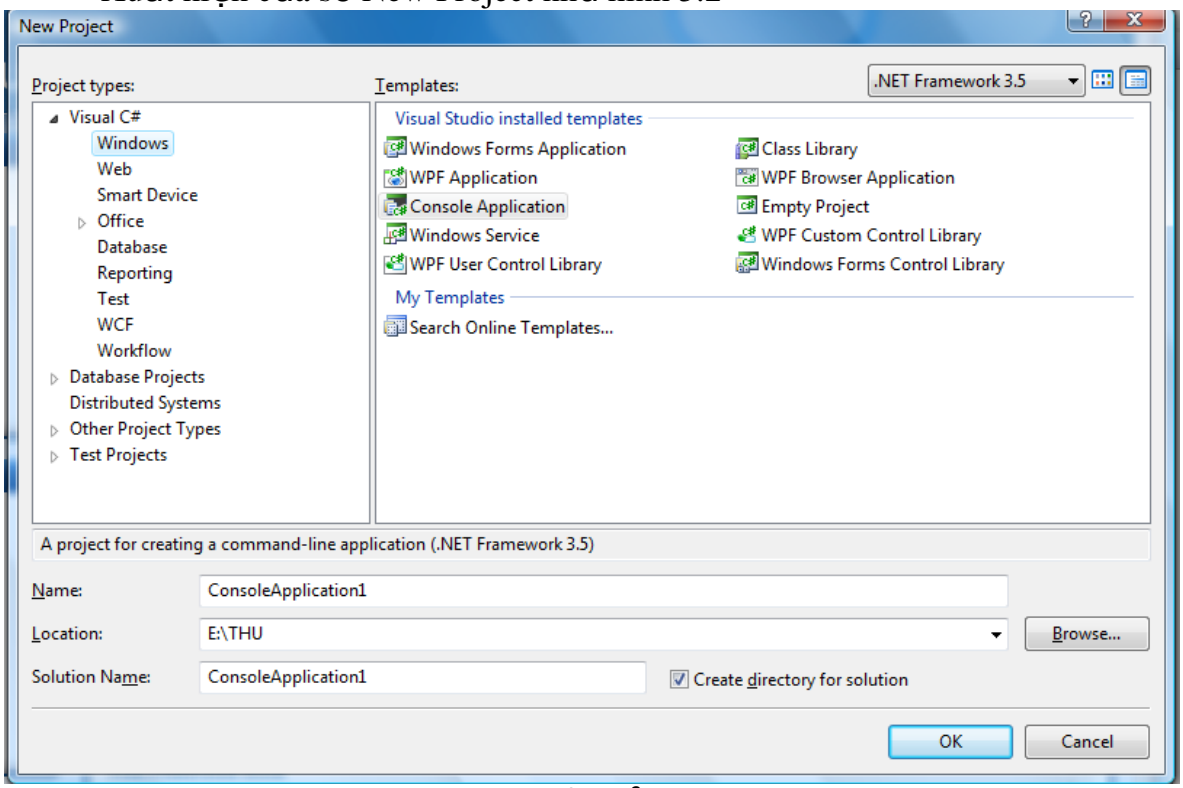


Hình 5.1. Cửa sổ Start Page – Microsoft Visual Studio

Hộp Recent Projects chứa các dự án mới làm gần đây, ta có thể click vào đây để mở dự án đã có. Hoặc bạn có thể click chuột tại Project… (thuộc mục Open trong hộp Recent Projects) để mở một dự án đã có. Trường hợp tạo mới một dự án, ta click chuột tại Project… (thuộc mục Create trong hộp Recent Projects)

Bước 2. Tại cửa sổ của hình 5.1, vào menu File → New → Project (hoặc bấm tổ hợp phím nóng Ctrl_Shift_N)

Xuất hiện cửa sổ New Project như hình 5.2



Hình 5.2. Cửa sổ New Project

Trong hộp Templates của hình 5.2 có nhiều khuôn dạng được cài đặt sẵn trong Visual Studio

- Windows Forms Application: Tạo ứng dụng với giao diện sử dụng Windows.

- Class Library: Dự án tạo các lớp sử dụng cho các ứng dụng khác.

- WPF Application (Windows Presentation Foudation Application): Dự án tạo ứng dụng với giao diện sử dụng Windows (loại dự án này tương tự với Windows Forms Application, nó chỉ xuất hiện ở bộ Visual Studio .Net 2008 với bộ .Net FrameWork 3.0 với nhiều tính năng mạnh bao trùm cả Windows Form Application, và việc sử dụng nó hiện nay đang là sự lựa chọn tối ưu so với Windows Form Application).

- WPF Browser Application: Dự án tạo ứng dụng với giao diện sử dụng trình duyệt web.

- Console Application: Dự án tạo ứng dụng trên các dòng lệnh chạy trên nền DOS.

- Empty Project: Dự án rỗng tạo ứng dụng cục bộ.

- Windows Service: Dự án tạo các dịch vụ cho Windows.

- WPF Custom Control Library: Dự án tạo thư viện liên kết động phục vụ cho WPF.

- WPF User Control Library: Dự án tạo thư viện liên kết động phục vụ cho WPF.

- Windows Forms Control Library: Dự án tạo thư viện liên kết động phục vụ cho Windows Forms Application.

Trong mục Template, ta chọn Windows Forms Application.

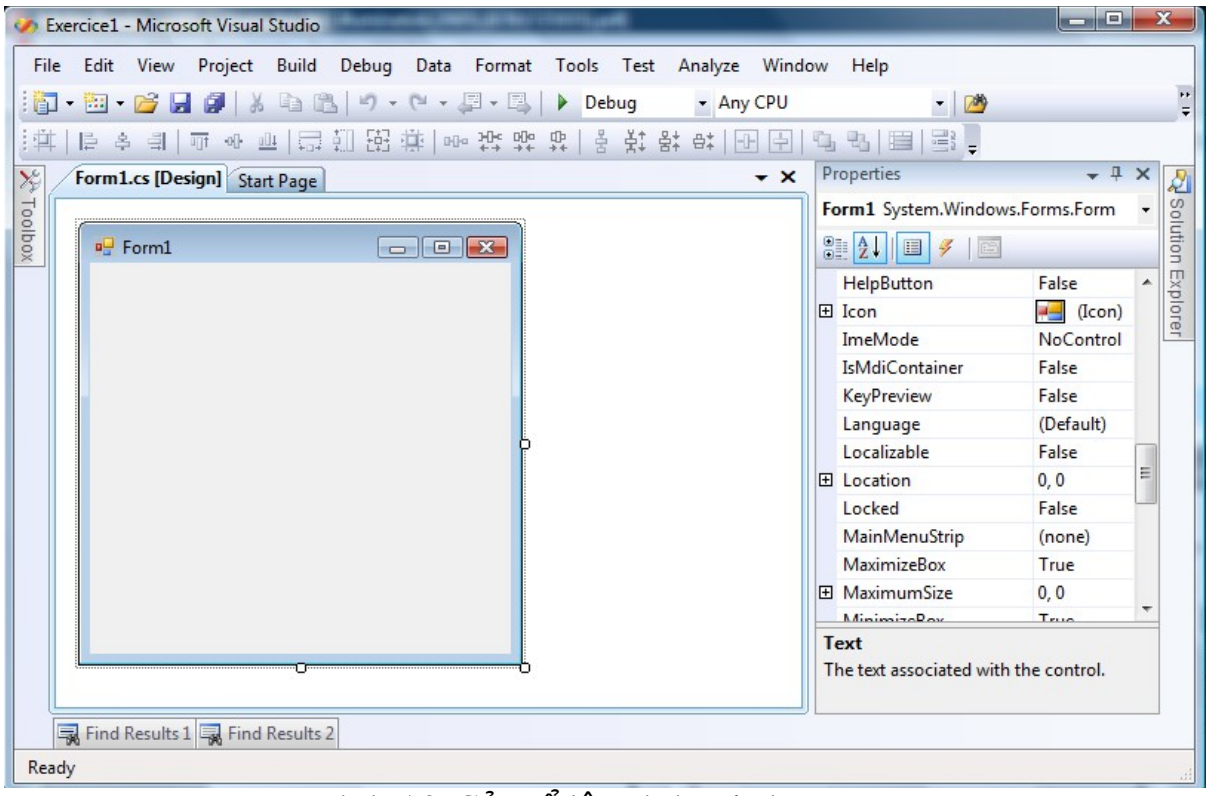
Bước 3. Tại hình 5.2

- Trong hộp Project types chọn Visual C# → Windows

- Trong hộp Templates chọn Windows Forms Application

- Textbox nhãn Name gõ: Nhập tên dự án

- Textbox nhãn Location chọn ổ đĩa cần lưu trữ, xuất hiện màn hình như hình 5.3

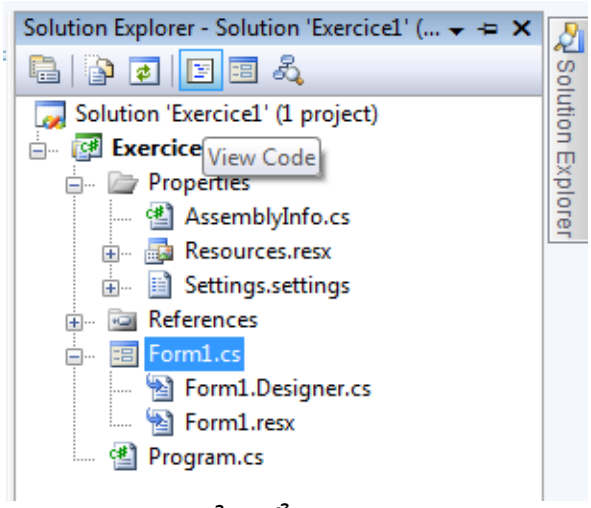


Hình 5.3. Cửa sổ lập trình Window Form

II. CÁC THÀNH PHẦN CƠ BẢN TRÊN CỬA SỔ WINDOWS FORM

II.1. Cửa sổ Form

Cửa sổ Form là mặc định với tên (Name) ban đầu là Form1 và đang ở chế độ thiết kế (Design), ta có thể chuyển đổi cửa sổ này sang cửa sổ mã lệnh bằng cách vào cửa sổ Solution Explorer, chọn Form1.cs, chọn View Code.



Hình 5.4. Cửa sổ Solution Explorer

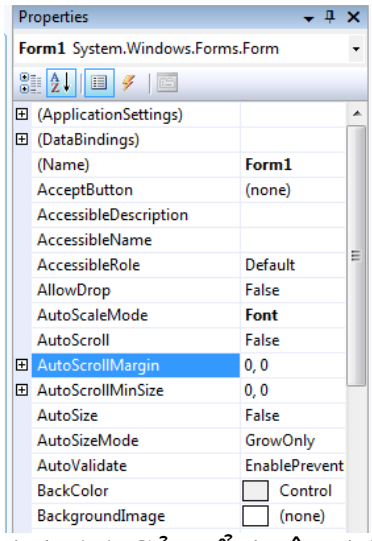
Trên Form ta có thể thêm vào các điều khiển khác nhau (các điều khiển này nằm trên hộp công cụ Toolbox.

II.2. Cửa sổ thuộc tính (Properties)

Trong hình 5.3, cửa sổ thuộc tính nằm bên trái cửa sổ Form, nếu cửa sổ thuộc tính không xuất hiện, ta có thể kích hoạt nó bằng cách vào menu View, chọn mục Properties Window (hoặc tổ hợp phím nóng Ctrl+W, P).

Sử dụng thanh trượt, ta có thể thấy nhiều thuộc tính khác nhau trên cửa sổ thuộc tính, và các thuộc tính này ta có thể sử dụng để cấu hình cửa sổ Form, hoặc các điều khiển (với điều kiện là Form hoặc nút điều khiển tương ứng đã được chọn).

Các thuộc tính thường dùng cho Form gồm: Name, BackColor, Enable, Font, Size, Text, WindowState



Hình 5.5. Cửa sổ thuộc tính

II.3. Cửa sổ Solution Explorer

Cửa sổ Solution Explorer hiển thị cây giải pháp hiện hành của Visual Microsoft .Net. Sử dụng cửa sổ này ta có thể duyệt qua tất cả các dự án hiện có cũng như các file liên quan trong mỗi dự án. Nếu khi kích đôi chuột lên một file dự án thì file này sẽ mở ra và ta có thể hiệu chỉnh nó.

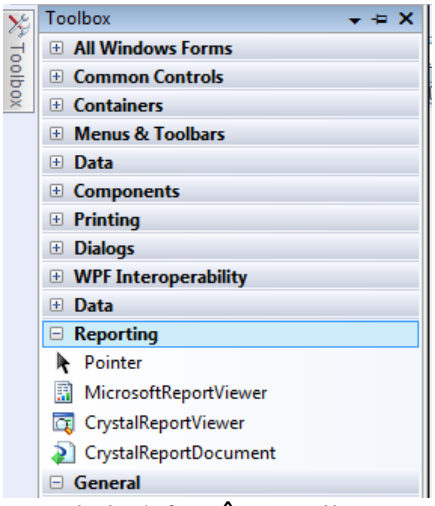
Để hiển thị cửa sổ Solution Explorer, vào menu View, chọn mục Solution Explorer (hoặc tổ hợp phím nóng Ctrl+W, S)

II.4. Hộp công cụ Toolbox

Hộp công cụ Toolbox chứa nhiều điều khiển mà ta có thể thêm vào Form. Để hiển thị hộp Toolbox, ta vào menu View, chọn mục Toolbox (hoặc tổ hợp phím nóng Ctrl+W, X). Hộp Toolbox sẽ gom các điều khiển theo loại nhóm:

- Common Controls: Các điều khiển cơ bản
- Containers: Các điều khiển liên quan khung chứa
- Menus & Toolbars: Các điều khiển tạo thực đơn, thanh công cụ
- Data: Các điều khiển liên quan về dữ liệu, khung chứa dữ liệu

.....



Hình 5.6. Hộp Toolbox

III. MỘT SỐ CÔNG CỤ CƠ BẢN TRÊN HỘP TOOLBOX

Có rất nhiều loại điều khiển khác nhau được sử dụng nhằm tạo các ứng dụng trên Windows Form. Một số loại điều khiển trong hộp Toolbox thuộc dạng đơn giản, số khác thuộc loại phức tạp. Tất cả các điều khiển cùng chia sẻ tập thuộc tính nhằm thiết lập cách thức hoạt động và sự trình bày cho điều khiển. Ngoài thuộc tính, mỗi một điều khiển sẽ có các sự kiện (event) nhằm chỉ rõ phản ứng của điều khiển trước các tác động bên ngoài.

Tất cả các điều khiển trên Windows Form được dẫn xuất từ lớp Control. Lớp này chứa tất cả các thành phần của Windows Form để xây dựng giao diện người dùng, kể các lớp Form. Lớp Control còn bao hàm cách thức thực hiện cơ bản của các điều khiển về thuộc tính, sự kiện, và phương thức. Và cuối cùng, Control là lớp trừu tượng, cho nên ta không thể trực tiếp tạo một thể hiện cho nó.

III.1. TextBox

Trong một ứng dụng, để nhập được văn bản ta sử dụng điều khiển TextBox. Để vẽ được TextBox trên Form, ta chỉ cần kích đôi chuột vào biểu tượng TextBox trong hộp ToolBox (hoặc gắp biểu tượng TextBox trong hộp ToolBox và thả lên Form).

Sau khi thêm TextBox vào Form (hoặc là sau khi kích chọn TextBox trên Form) cửa sổ thuộc tính sẽ đưa ra tất cả các thuộc tính của TextBox (cửa sổ thuộc tính sẽ đưa ra giá trị các thuộc tính của điều khiển được chọn). Các thuộc tính thường dùng của TextBox là:

Bảng 5.1. Bảng thuộc tính của điều khiển TextBox

Thuộc tính	**Ý nghĩa**
Name	Tên của TextBox thường bắt đầu bởi tiếp đầu ngữ **txt**
BackColor	Chọn màu cho TextBox
Enabled	Khóa cứng hoặc không một TextBox
Font	Định dạng Font chữ cho chuỗi dữ liệu của TextBox
Multiline	Cho phép nhập dữ liệu vào TextBox trên nhiều dòng
PasswordChar	Kí tự đại diện khi nhập mật khẩu
Text	Chuỗi dữ liệu xuất hiện trong TextBox
Visible	Hiển thị hoặc che dấu TextBox

Các sự kiện (Event) thường gặp: Leave, TextChanged, Enter

Ví dụ 5.1. Thiết lập các thuộc tính và sử dụng sự kiện TextChanged

Hãy tạo TextBox với Name: txtA, Text: trống; Event: TextChanged nhằm đưa ra dòng thông báo chuỗi dữ liệu của txtA.

Đoạn mã lệnh xử lý sự kiện trong Form1.cs có dạng:

```
public Form1()
{
    InitializeComponent();
}
private void txtA_TextChanged(object sender, EventArgs e)
{
    MessageBox.Show("Vua bam: " + txtA.Text);
}
```

Ví dụ 5.2. Thiết lập các thuộc tính và sử dụng sự kiện Leave

Hãy tạo 2 TextBox:

- TextBox thứ nhất với Name: txtA, BackColor: màu hồng, Text: trống, Event: Leave nhằm thay đổi màu sắc của txtA sang trắng, chuyển con trỏ màn sang txtB, xóa hết dữ liệu trên txtB, sau đó đổi màu sắc txtB sang hồng.
- TextBox thứ nhất với Name: txtA, BackColor: màu trắng, Text: trống

Đoạn mã lệnh xử lý sự kiện trong Form1.cs có dạng:

```
public Form1()
{
    InitializeComponent();
}
private void txtA_Leave(object sender, EventArgs e)
{
   txtA.BackColor = Color.FromArgb(((int)(((byte)(255)))),
((int)(((byte)(255)))), ((int)(((byte)(255)))));
   txtB.Focus();
   txtB.Clear();
    txtB.BackColor = Color.FromArgb(((int)(((byte)(255)))),
((int)(((byte)(224)))), ((int)(((byte)(192)))));
}
```

Ví dụ 5.3. Tạo TextBox tự động và sử dụng sự kiện Leave

Hãy tạo 5 TextBox có tên lần lượt là txt0, txt1, ..., txt4 với kích thước mỗi TextBox là 200x40; và sự kiện Leave tương ứng với mỗi TextBox nhằm đưa ra dòng thông báo: “Bạn vừa rời khỏi TextBox txt?”.

Đoạn mã lệnh trong Form1.cs có dạng:

```
public Form1()
{
   InitializeComponent();
   TextBox[] A; A = new TextBox[5];
   for (int i = 0; i < 5; i++)
   {
      A[i] = new TextBox();
      A[i].Name = "txt" + i.ToString();
```

```
        A[i].Size = new Size(200,40);
        A[i].Location = new Point(47, (i+1)*40 + 27);
        A[i].Parent = this;
        A[i].Leave += new System.EventHandler(this.textBox_Leave);
    }
}
private void textBox_Leave(object sender, EventArgs e)
{
    TextBox a;
    a = (TextBox)sender;
    MessageBox.Show("Ban vua roi khoi TextBox " + a.Name);
}
```

III.2. Label

Label thường được sử dụng để làm nhãn diễn tả và được đặt trên Form. Các thuộc tính thường dùng của Label là: Name, BackColor, Font, ForeColor, và Text.

III.3. Các nút lệnh

Có 3 loại nút lệnh: Button, RadioButton, CheckBox. Mỗi nút lệnh được sử dụng để biểu diễn kiểu nút lệnh chuẩn của Windows. Cả 3 loại nút lệnh này được thừa kế từ lớp cơ sở: ButtonBase. Hẳn là những ai lần đầu lập trình trên Windows cũng đều ngạc nhiên khi RadioButton và CheckBox lại thuộc vào nhóm các nút lệnh, bởi do hai loại nút này giống Button ở điểm: Chúng có thể được kích hoạt, và chúng đều có nhãn.

III.3.1. Button

Button là nút nhấn trên Windows chứa các thuộc tính, phương thức, sự kiện nhằm làm đơn giản hóa công việc khi thao tác trên Button. Để đưa một Button vào Form, ta chỉ cần kích đôi chuột vào biểu tượng Button trong hộp ToolBox (hoặc gắp biểu tượng Button trong hộp ToolBox và thả lên Form).

Sau khi thêm Button vào Form (hoặc là sau khi kích chọn Button trên Form) cửa sổ thuộc tính sẽ đưa ra tất cả các thuộc tính của Button. Các thuộc tính thường dùng của Button là:

Bảng 5.2. Bảng thuộc tính của Button

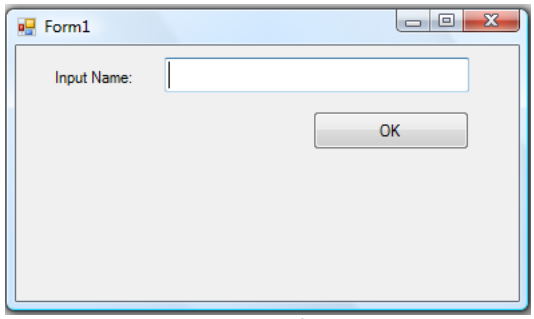
Thuộc tính	Ý nghĩa
Name	Tên của Button thường bắt đầu bởi tiếp đầu ngữ btn
BackColor	Chọn màu cho Button
Enabled	Khóa cứng hoặc không một Button
Font	Định dạng Font chữ cho chuỗi dữ liệu của Button
ForeColor	Màu sắc cho chuỗi Text
Text	Chuỗi dữ liệu xuất hiện trong Button
Visible	Hiển thị hoặc che dấu Button

Các sự kiện (Event) thường gặp: Click

Ví dụ 5.4. Tạo Label, TextBox, Button và sử dụng sự kiện Click

Hãy tạo 1 Label (Text: Input Name), 1 TextBox (thuộc tính Name: txtName, Text: trống), 1 Button (Name: btnOk, Text: OK) trong hình 5.7.

Yêu cầu: Khi kích vào Button OK thì xuất hiện dòng thông báo: “Bạn vừa nhập tên ” + txtName.Text.



Hình 5.7. Mô tả ví dụ 5.4

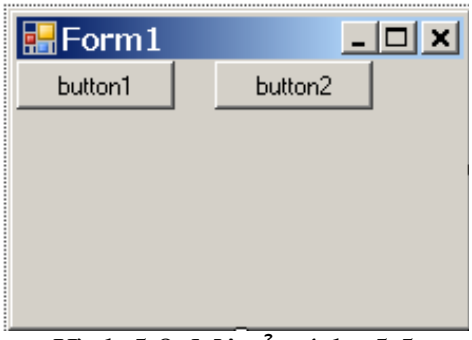
Đoạn mã lệnh trong Form1.cs có dạng:

```
public Form1()
{
    InitializeComponent();
}
private void btnOK_Click(object sender, EventArgs e)
{
    MessageBox.Show("Ban vua nhap ten " + txtName.Text);
    txtName.Clear();
    txtName.Focus();
}
```

Ví dụ 5.5: Tạo Form có 2 nút lệnh như hình vẽ 5.8.

Yêu cầu:

- Khi click chuột vào button1 → xuất hiện "Vừa bấm button1"
- Khi click chuột vào button2 → xuất hiện "Vừa bấm button2"



Hình 5.8. Mô tả ví dụ 5.5

Hướng dẫn: Ta chỉ cài đặt chung một sự kiện Click cho cả hai Button1 và Button2 như sau:

```
private void button_Click(object sender, EventArgs e)
{
   Button a;
   a = (Button)sender;
   MessageBox.Show("Vua bam " + a.Text);
}
```

Trong phần InitializeComponent() của Form1.Designer.cs, ta sửa thêm lần lượt 2 sự kiện vào sau phần xây dựng của Button1 và Button2 như sau:

```
//Button1
this.button1.Click += new System.EventHandler(this.button_Click);
```

```
//Button2
this.button2.Click += new System.EventHandler(this.button_Click);
```

III.3.2. RadioButton

RadioButton được sử dụng để tạo các nút chọn radio. Khi chọn lựa một trong nhiều công việc, người ta thường sử dụng điều khiển RadioButton để thực hiện, và mặc định trong nhóm các radio chỉ có một radio được chọn mà thôi.

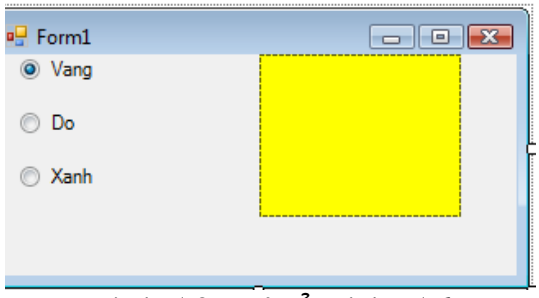
Ngoài các thuộc tính tương tự như các mục nêu trên, nút chọn radio còn có một thuộc tính quan trọng đó là thuộc tính Checked nhằm xác định nút radio có được chọn hay không?

Sự kiện thường sử dụng trong nút chọn radio là Click.

Ví dụ 5.6: Tạo Form có 3 RadioButton, 1 PictureBox như hình vẽ 5.9.

Yêu cầu:

- Khi chọn Vang → PictureBox chuyển thành màu vàng.
- Khi chọn Do → PictureBox chuyển thành màu đỏ.
- Khi chọn Xanh → PictureBox chuyển thành màu xanh.



Hình 5.9. Mô tả ví dụ 5.6

Đoạn mã lệnh trong Form1.cs có dạng:

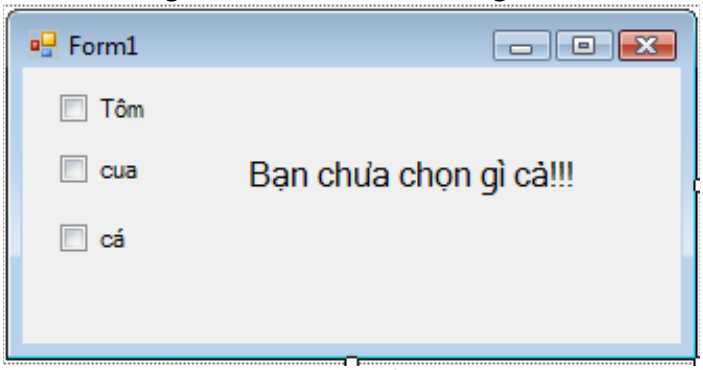
```
public Form1()
{
  InitializeComponent();
}
private void radioButton1_CheckedChanged(object sender, EventArgs e)
{
  if (radioButton1.Checked) pictureBox1.BackColor = Color.Yellow;
}
private void radioButton2_CheckedChanged(object sender, EventArgs e)
{
  if (radioButton2.Checked) pictureBox1.BackColor = Color.Red;
}
private void radioButton3_CheckedChanged(object sender, EventArgs e)
{
  if (radioButton3.Checked) pictureBox1.BackColor = Color.Green;
}
```

III.3.3. CheckBox

CheckBox tương tự RadioButton nhưng chỉ khác ở điểm, ta có thể chọn nhiều CheckBox cùng lúc.

Ví dụ 5.7: Tạo Form có 3 CheckBox, 1 Label (thuộc tính Text nhập "Bạn chưa chọn gì cả") như hình vẽ 5.10.

Yêu cầu: Khi chọn CheckBox nào thì dữ liệu Text tương ứng sẽ hiện ra trên Label, ngược lại sẽ hiện dòng chữ: Bạn chưa chọn gì cả!!!



Hình 5.10. Mô tả ví dụ 5.7

Đoạn mã lệnh trong Form1.cs có dạng:

```
public Form1()
{
   InitializeComponent();
}
private void checkBox_CheckedChanged(object sender, EventArgs e)
{
   string s = "Ban da chon: ";
   if (checkBox1.Checked) s += checkBox1.Text + ", ";
   else s.Replace(checkBox1.Text, "");
   if (checkBox2.Checked) s += checkBox2.Text + ", ";
   else s.Replace(checkBox2.Text, "");
   if (checkBox3.Checked) s += checkBox3.Text;
   else s.Replace(checkBox3.Text, "");
   if (checkBox1.Checked || checkBox2.Checked || checkBox3.Checked)
       label1.Text = s;
   else label1.Text = "Bạn chưa chọn gì cả!!!";
}
```

Hoặc

```
private void checkBox_CheckedChanged(object sender, EventArgs e)
{
   string s = "Ban da chon: ";
   if (checkBox1.Checked) s += checkBox1.Text + ", ";
   if (checkBox2.Checked) s += checkBox2.Text + ", ";
   if (checkBox3.Checked) s += checkBox3.Text;
   if (checkBox1.Checked || checkBox2.Checked || checkBox3.Checked)
   { s=s.Substring(0,S.Length-2); label1.Text = s; }
   else label1.Text = "Bạn chưa chọn gì cả!!!";
}
```

Trong phần InitializeComponent() của Form1.Designer.cs, ta sửa thêm lần lượt 3 sự kiện vào sau phần xây dựng của checkBox1, checkBox2 và checkBox3 như sau:

```
//checkBox1
this.checkBox1.CheckedChanged += new
System.EventHandler(this.checkBox_CheckedChanged);
//checkBox2
this.checkBox2.CheckedChanged += new
System.EventHandler(this.checkBox_CheckedChanged);
//checkBox3
this.checkBox3.CheckedChanged += new
System.EventHandler(this.checkBox_CheckedChanged);
```

III.4. ListBox

ListBox được sử dụng để tạo hộp các danh mục và người sử dụng có thể chọn lựa bằng cách kích chuột. Trong ListBox ta có thể chọn lựa một hoặc nhiều danh mục.

Ngoài các thuộc tính tương tự như các mục nêu trên, ListBox còn có một thuộc tính cần quan tâm:

- Thuộc tính Items nhằm nhập vào ListBox các mục chọn.
- Thuộc tính SelectionMode, có 4 giá trị:

+ None : Không có mục nào được chọn
+ One : Chỉ chọn được một mục chon
+ MultiSimple : Có thể chọn được nhiều mục chọn
+ MultiExtended : Có thể chọn nhiều mục chọn, và người sử dụng có thể dùng phím Shift, Ctrl, mũi tên để chọn.

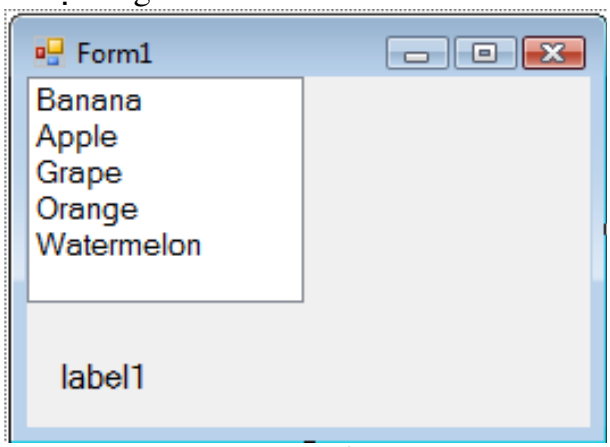
- Thuộc tính Sorted nhằm sắp xếp các mục chọn (nếu có giá trị true)

Sự kiện mà ListBox thường dùng là SelectedIndexChanged.

- Để lấy được một mục trong danh mục ta sử dụng thuộc tính SelectedItem.
- Muốn lấy tất cả các mục đã được chọn ta sử dụng thuộc tính SelectedItems, các mục được chọn lựa được đánh theo chỉ mục, bắt đầu từ 0, kế tiếp 1, ...

Ví dụ 5.8: Tạo Form có 1 ListBox (thuộc tính SelectionMode: One), 1 Label như hình vẽ 5.11.

Yêu cầu: Khi chọn một mục trên ListBox thì tên được chọn cùng với độ dài của mục chọn sẽ hiện thị trong Label



Hình 5.11. Mô tả ví dụ 5.8

Đoạn mã lệnh trong Form1.cs có dạng:

```
public Form1()
{
  InitializeComponent();
}
private void lstFruit_SelectedIndexChanged(object sender, EventArgs e)
{
  label1.Text = lstFruit.SelectedItem + " has " +
    lstFruit.SelectedItem.ToString().Length.ToString() + " letters.";
}
```

Ví dụ 5.9: Tạo Form có 1 ListBox (thuộc tính SelectionMode: MultiSimple), 1 Label như hình vẽ 5.11.

Yêu cầu: Các mục chọn trên ListBox sẽ hiện thị trong Label

Đoạn mã lệnh trong Form1.cs có dạng:

```
public Form1()
{
  InitializeComponent();
}
private void lstFruit_SelectedIndexChanged(object sender, EventArgs e)
{
  string s = "";
  for(int i=0;i<lstFruit.SelectedItems.Count;i++)
     s += lstFruit.SelectedItems[i]+ ", ";
  s=s.Remove(s.Length-2);
  label1.Text = s;
}
```

Lưu ý: trong vòng for, i sẽ tương ứng với các mục chọn màu xanh và bắt đầu từ vị trí 0. Do vậy lstFruit.SelectedItems[i] sẽ tương ứng với mục chọn.

III.5. CheckedListBox

CheckedListBox được sử dụng để tạo hộp chứa các nút kiểm để người sử dụng có thể chọn lựa bằng cách kích chuột (đi liền với nút kiểm là thông tin để ta làm cơ sở chọn lựa).

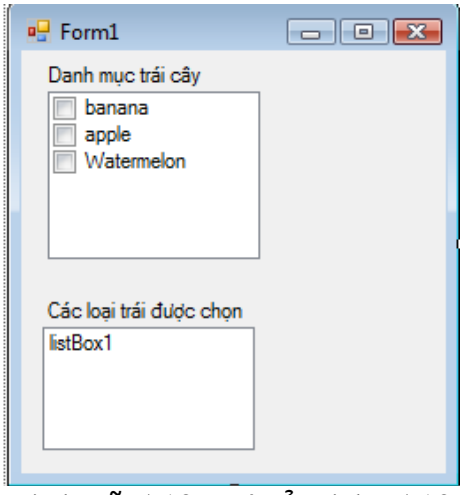
CheckedListBox được thừa kế từ lớp ListBox nhưng thêm vào chức năng các dấu kiểm trước các mục chọn. Do vậy ngoài các thuộc tính mà CheckedListBox thừa kế từ lớp ListBox, CheckedListBox còn có một thuộc tính cần quan tâm, đó là thuộc tính CheckOnClick, nếu thuộc tính này để giá trị false thì khi kích chọn vào dấu kiểm ta phải kích hoạt 2 lần, để kích hoạt 1 lần ta chọn giá trị true.

Sự kiện mà CheckedListBox thường sử dụng là sự kiện ItemCheck, trình kiểm soát sự kiện ItemCheck sẽ nhận một tham số đối tượng ItemCheckEventArgs, đối tượng này được dẫn xuất từ lớp EventArgs và có 3 thuộc tính liên quan đến sự kiện ItemCheck, đó là:

- Index: Cho biết số thứ tự của mục chọn của hộp kiểm trong hộp CheckedListBox
- CurrentValue: Hiển thị tình trạng của hộp kiểm trong hộp CheckedListBox
- NewValue: Trả về giá trị của hộp kiểm trong hộp CheckedListBox

Ví dụ 5.10: Tạo Form có 1 CheckedListBox (thuộc tính CheckOnClick: true), 1 ListBox, 2 Label như hình vẽ 5.12.

Yêu cầu: Khi kích chọn (hoặc hủy) loại trái cây nào trong bảng Danh mục trái cây thì loại trái được chọn được thêm vào (hoặc loại ra) bảng Các loại trái cây được chọn.



Hình vẽ 5.12. Mô tả ví dụ 5.10

Đoạn mã lệnh trong Form1.cs có dạng:

```
public Form1()
{
  InitializeComponent();
}
private void checkedListBox1_ItemCheck(object sender, ItemCheckEventArgs e)
{
   if (e.NewValue == CheckState.Checked)
                listBox1.Items.Add(checkedListBox1.SelectedItem.ToString());
   else
               listBox1.Items.Remove(checkedListBox1.SelectedItem.ToString());
}
```

III.6. ComboBox

Giống như ListBox, ComboBox cũng chứa rất nhiều mục chọn. ComboBox chiếm ít không gian hơn ListBox, do vậy ComboBox chỉ hiển thị một mục chọn, còn các mục chọn khác ComboBox ẩn đi và khi cần chọn những mục ẩn này ta chỉ cần kích hoạt vào mũi tên xuống bên phải.

Các thuộc tính của ComboBox cũng tương tự như TextBox, ListBox. Thuộc tính thường dùng là SelectedItem (giá trị mục chọn), Items, SelectedIndex (số thứ tự của mục chọn).

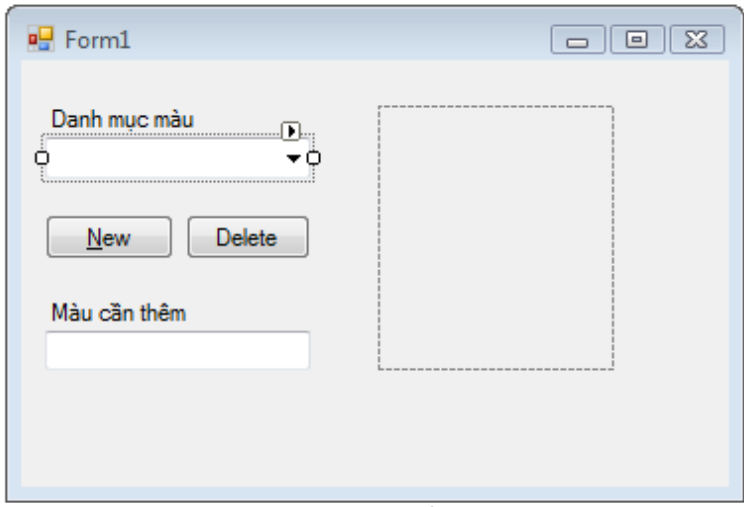
Sự kiện thường dùng là SelectionChangeCommitted, sự kiện này nhằm khẳng định sự thay đổi của một mục chọn trên ComboBox.

Ví dụ 5.11: Tạo Form có 1 ComboBox, 1 TextBox, 2 Label, 2 Button (thuộc tính lần lượt là Name: btnNew, Text: &New, và Name: btnDelete, Text: &Delete), 1 PictureBox như hình vẽ 5.13.

Yêu cầu:

- Khởi tạo 3 màu Green, Red, Yellow cho ComboBox
- Khi kích nút thêm thì dữ liệu màu từ TextBox sẽ thêm vào ComboBox.
- Khi kích nút xóa thì màu được chọn từ ComboBox sẽ bị xóa.

- Khi chọn màu trong bảng danh mục màu, thì PictureBox sẽ tô màu theo màu đã chọn trong bảng danh mục màu.



Hình vẽ 5.13. Mô tả ví dụ 5.11

Đoạn mã lệnh trong Form1.cs có dạng:

```
public Form1()
{
    InitializeComponent();
    comboBox1.Items.Add("Green");
    comboBox1.Items.Add("Red");
    comboBox1.Items.Add("Yellow");
    Text_Init();
}
private void Text_Init()
{
    if (comboBox1.Items[0].ToString() != "")
    {
        comboBox1.Text = comboBox1.Items[0].ToString();
        pictureBox1.BackColor =
Color.FromName(comboBox1.Items[0].ToString());
    }
}
private void comboBox1_SelectionChangeCommitted(object sender, EventArgs e)
{
    pictureBox1.BackColor =
Color.FromName(comboBox1.SelectedItem.ToString());
}
private void btnNew_Click(object sender, EventArgs e)
{
    if (textBox1.Text != "")
    {
       comboBox1.Items.Add(textBox1.Text);
       textBox1.Clear();
    }
```

```
    else MessageBox.Show("Bạn cần nhập màu vào hộp màu cần thêm!!!");
}
private void btnDelete_Click(object sender, EventArgs e)
{
    comboBox1.Items.RemoveAt(comboBox1.SelectedIndex);
    Text_Init();
}
```

III.7. ScrollBar

ScrollBar là thanh cuộn màn hình, khi hiển thị một điều khiển hoặc một Form có kích thước lớn hơn màn hình thì ta cần phải sử dụng ScrollBar. Có hai loại thanh cuộn màn hình, thanh cuộn dọc – VScrollBar, và thanh cuộn ngang – HscrollBar, cả hai thanh này được kế thừa từ lớp ScrollBar.

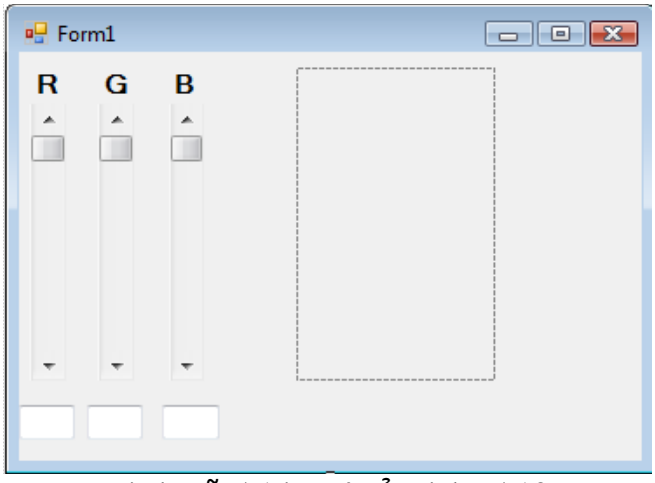
Đôi khi ta không cần tạo ScrollBar, vì nhiều lớp của Windows Form đã hỗ trợ đặc tính này, những lớp này được dẫn xuất từ lớp ScrollableControl. Lớp này cung cấp kiểu xây dựng sẵn hỗ trợ cho các thanh cuộn, cho phép tạo thanh cuộn bằng cách thiết lập thuộc tính, chẳng hạn trong Form, ta có thể sử dụng thuộc tính AutoScroll = true.

Thuộc tính VScrollBar hoặc HscrollBar thường hay sử dụng là: Maximum (giá trị cực đại của thanh cuộn), Minimum (giá trị cực tiểu của thanh cuộn), Value (giá trị hiện tại của thanh cuộn), LargeChange (giá trị thay đổi lớn nhất).

Sự kiện thường dùng là ValueChanged.

Ví dụ 5.12: Tạo Form có 3 VscrollBar (thuộc tính Maximum: 255, LargeChange: 1), 3 TextBox, 3 Label (Thuộc tính Text lần lượt là R, G, B) 1 PictureBox như hình vẽ 5.14.

Yêu cầu: Khi di chuyển thanh trượt nào thì giá trị thanh trượt hiện ra trong hộp TextBox tương ứng, và màu sắc của PictureBox sẽ phụ thuộc vào giá trị của 3 thanh trượt.



Hình vẽ 5.14. Mô tả ví dụ 5.12

Đoạn mã lệnh trong Form1.cs có dạng:

```
public Form1()
{
   InitializeComponent();
}
```

```
private void vScrollBar_ValueChanged(object sender, EventArgs e)
{
   pictureBox1.BackColor = Color.FromArgb(vScrollBar1.Value,
vScrollBar2.Value, vScrollBar3.Value);
   textBox1.Text = vScrollBar1.Value.ToString();
   textBox2.Text = vScrollBar2.Value.ToString();
   textBox3.Text = vScrollBar3.Value.ToString();
}
```

Trong phần InitializeComponent() của Form1.Designer.cs, ta sửa lần lượt 3 sự kiện vào sau phần xây dựng của vScrollBar1, vScrollBar2 và vScrollBar3 như sau:

```
//vScrollBar1
this.vScrollBar1.ValueChanged += new
System.EventHandler(this.vScrollBar_ValueChanged);
//vScrollBar2
this.vScrollBar2.ValueChanged += new
System.EventHandler(this.vScrollBar_ValueChanged);
//vScrollBar3
this.vScrollBar3.ValueChanged += new
System.EventHandler(this.vScrollBar_ValueChanged);
```

III.8. Trackbar

TrackBar cung cấp giao diện đơn giản cho phép người dùng lấy giá trị trong khoảng giá trị cố định sẵn trên TrackBar bằng cách sử dụng chuột hoặc bàn phím tác động lên thanh trượt.

TrackBar có 2 phần, đó là thanh trượt và các dấu ghi khoảng cách. Thanh trượt liên quan đến thuộc tính Value. Thanh TrackBar có thể nằm ngang hoặc dọc, điều này được quy định bởi thuộc tính Orientation (Horizotal: chiều ngang, Vertical: chiều dọc). Các thuộc tính còn lại giống với ScrollBar.

Sự kiện thường dùng là ValueChanged.

Ví dụ 5.13: Giống ví dụ 5.12, nhưng thay 3 thanh VscrollBar thành 3 thanh TrackBar.

Đoạn mã lệnh trong Form1.cs có dạng:

```
public Form1()
{
   InitializeComponent();
}
private void trackBar_ValueChanged(object sender, EventArgs e)
{
   TrackBar a = (TrackBar)sender;
   pictureBox1.BackColor = Color.FromArgb(trackBar1.Value,
trackBar2.Value, trackBar3.Value);
   textBox1.Text = trackBar1.Value.ToString();
   textBox2.Text = trackBar2.Value.ToString();
   textBox3.Text = trackBar3.Value.ToString();
}
```

Trong phần InitializeComponent() của Form1.Designer.cs, ta sửa lần lượt 3 sự kiện vào sau phần xây dựng của trackBar1, trackBar2 và trackBar3 như sau:

```
//trackBar1
this.trackBar1.ValueChanged += new
System.EventHandler(this.trackBar_ValueChanged);
//trackBar2
this.trackBar2.ValueChanged += new
System.EventHandler(this.trackBar_ValueChanged);
//trackBar3
this.trackBar3.ValueChanged += new
System.EventHandler(this.trackBar_ValueChanged);
```

III.9. GroupBox

GroupBox là một khung bao quanh một nhóm có điều khiển, mỗi GroupBox có tiêu đề hoặc không. GroupBox thường được sử dụng để chia nhỏ một Form theo từng nhóm chức năng, và cho người dùng cách nhìn tổng quan theo nhóm các điều khiển.

Tại thời điểm thiết kế, ta có thể bổ sung các điều khiển vào GroupBox. Nếu di chuyển GroupBox thì các điều khiển trên nó sẽ di chuyển theo.

III.10. Timer

Timer cho phép ta đặt thời gian và thực hiện một công việc theo sự kiện Tick sau 1 khoảng thời gian đã được đặt.

Thuộc tính thường sử dụng trên Timer là Interval (nếu đặt Interval = 1000 thì tương ứng với khoảng thời gian 1 giây), Enable (đặt là true).

Sự kiện thường được sử dụng là Tick.

Ví dụ 5.14: Tạo Form có 1 Button (thuộc tính Text: Stop), 1 Label, 1 Timer (thuộc tính Interval: 1000, Enable: true) như hình vẽ 5.15.

Yêu cầu: Cứ mỗi giây, Label sẽ hiển thị thời gian hiện hành 1 lần, nếu kích vào nút lệnh Stop, quá trình hiển thị này sẽ dừng lại và nút lệnh chuyển thành Start, nếu kích vào Start, quá trình hiển thị thời gian trong Label lại giống như ban đầu.

Hình vẽ 5.15. Mô tả ví dụ 5.10

Đoạn mã lệnh trong Form1.cs có dạng:

```
public Form1()
```

```
{
    InitializeComponent();
}
private void button1_Click(object sender, EventArgs e)
{
    if (button1.Text == "Stop")
    {
        button1.Text = "Start";
        timer1.Enabled = false;
    }
    else
    {
        button1.Text = "Stop";
        timer1.Enabled = true;
    }
}
private void timer1_Tick(object sender, EventArgs e)
{
    label1.Text = DateTime.Now.ToString();
}
```

III.11. MonthCalendar

MonthCalendar trình bày giao diện đồ họa trực quan cho người dùng xem, chọn và đặt thông tin về ngày tháng.

Các thuộc tính thường dùng:

- AnnuallyBoldedDate: Định nghĩa dãy các ngày in đậm trên lịch hằng năm.
- BolderDate: Định nghĩa dãy các ngày in đậm trên lịch
- CalendarDimensions: Xác định số cột và hàng của tháng hiển thị
- FirstDayOfWeek: Xác định ngày đầu tiên của tuần hiển thị trên lịch tháng. Mặc định, chủ nhật được trình bày là ngày đầu tiên.
- MaxDate: Xác định ngày lớn nhất được phép

....

Ví dụ 5.15: Tạo Form có 1 MonthCalendar

Yêu cầu: Khi chọn một dãy ngày trên MonthCalendar, sẽ xuất hiện dòng thông báo: bạn đã chọn từ ngày ... đến ngày ...

Đoạn mã lệnh trong Form1.cs có dạng:

```
private void monthCalendar1_DateSelected(object sender, DateRangeEventArgs e)
{
   MessageBox.Show("Bạn đã chọn từ ngày " +
monthCalendar1.SelectionStart.ToLongDateString() + " đến ngày "
+ monthCalendar1.SelectionEnd.ToLongDateString());
}
```

IV. FORM VÀ MDI FORM

IV.1. Giới thiệu

Tất cả các dự án trước đây mà ta đã nghiên cứu đều ở dạng giao diện đơn tài liệu SDI (Single-Document Interface). Trong các chương trình SDI, mọi Form trong

ứng dụng đều ngang hàng nhau, không có sự phân cấp giữa các Form. Bên cạnh SDI, C# còn cho phép ta tạo các chương trình có các giao diện đa tài liệu MDI (Multiple-Document Interface). Một chương trình MDI có một cửa sổ cha (hay còn được gọi là cửa sổ chứa các cửa sổ khác) và các cửa sổ con. Chẳng hạn, Microsoft Word 97 là một cửa sổ MDI, vì tại cửa sổ này ta có thể mở bất kỳ cửa sổ tài liệu con nào. Trong chương trình MDI, tất cả các cửa sổ con chia sẻ cùng thanh công cụ và thanh thực đơn xuất hiện trên cửa sổ cha. Đặc điểm của các cửa sổ con đó là chúng luôn được giới hạn trong cửa sổ cha.

IV.2. Tạo MDI Form

Để tạo một MDI Form, từ Form thông thường, ta thay đổi thuộc tính IsMdiContainer bằng true.

Ví dụ 5.16: Tạo Windows Appliaction tên MDI Example. Tại Form xuất hiện, đặt các thuộc tính Name: fclsMDIParent, Text: MDIParent, IsMdiContainer: true.

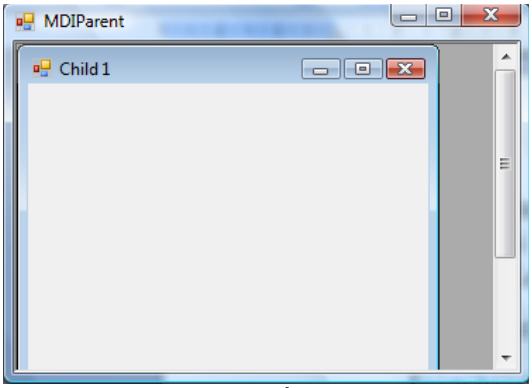
Kết quả: Sau khi đặt thuộc tính cho Form, Form chuyển sang màu xám đen.

Từ ví dụ 5.16, tạo thêm một Form mới bằng cách vào menu Project, chọn Add Windows Form. Đặt tên Form là fclsChild1.cs và đặt thuộc tính Text: Child 1. Thêm Form thứ 3 vào dự án theo cách tương tự, đặt tên Form là fclsChild2.cs và đặt thuộc tính Text: Child 2. Ta có thể tạo thêm nhiều Form như vậy, nhưng tuyệt nhiên chỉ có một MDI Form mà thôi.

Đảm bảo rằng Form cha fclsMDIParent đang được hiển thị, trường hợp không thấy, ta có thể vào cửa sổ Solution Explorer và kích đôi chuột vào fclsMDIParent. Tiếp đến, ta kích đôi chuột vào Form này nhằm kích hoạt sự kiện mặc định của Form (sự kiện Load). Nhập vào đoạn mã sau:

```
private void fclsMDIParent_Load(object sender, EventArgs e)
{
    fclsChild1 objChild = new fclsChild1();
    objChild.MdiParent = this;
    objChild.Show();
}
```

Câu lệnh đầu tiên nhằm khai báo biến đối tượng kiểu fclsChild1, sau đó khởi nó thành thể hiện của Form fclsChild1. Câu lệnh thứ 3 nhằm trình bày 1 Form. Câu lệnh ta cần quan tâm ở đây là câu lệnh thứ 2. Nó đặt thuộc tính MdiParent của Form fclsChild1 tới Form fclsMDIParent (this chính là Form fclsMDIParent), có được điều này do ta đã đặt thuộc tính của IsMdiContainer của Form fclsMDIParent là true. Lưu lại các công việc vừa thực hiện và bấm F5 để chạy chương trình, ta được kết quả như hình 5.16.



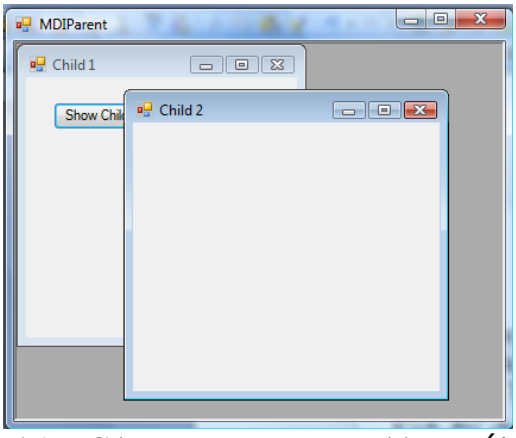
Hình 5.16. Form con xuất hiện trong Form cha

Trở về chương trình thiết kế ban đầu, kích đôi chuột vào Form fclsChild1 trong cửa sổ Solution Explorer. Thêm vào Form này 1 Button (thuộc tính Name: btnShowChild2, Text: Show Child 2).

Kích đôi chuột vào Button này để kích hoạt sự kiện Click của nó, sau đó nhập vào đoạn mã sau:

```
private void btnShowChild2_Click(object sender, EventArgs e)
{
    fclsChild2 objChild = new fclsChild2();
    objChild.MdiParent = this.MdiParent;
    objChild.Show();
}
```

Kết quả thu được như hình 5.17.



Hình 5.17. Các Form con ngang hàng với nhau

V. MENU

Thanh thực đơn (menu) là một đặc trưng chuẩn mực dành cho các phần mềm hữu dụng. Trong C#, điều khiển MenuStrip cho phép người lập trình dễ dàng thêm thanh thực đơn vào ứng dụng. Hộp thoại (Dialog) là cách tiếp cận khác cho phép người lập trình dễ dàng chọn lựa bằng cách tích vào các mục trong hộp có sẵn hoặc kích chọn một mục trong bảng danh mục cho trước. Trong phần này, ta tìm hiểu OpenFileDialog, SaveFileDialog, PrintDialog, ColorDialog, FontDialog, RichTextBox, và PrintDocument.

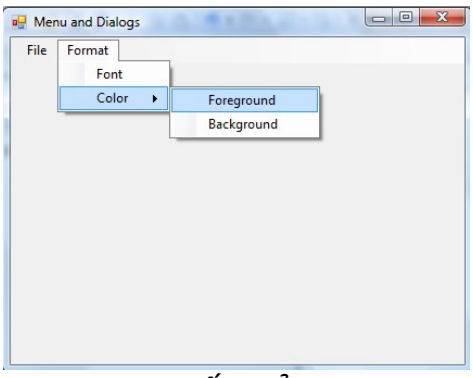
V.1. MenuStrip (MainMenu)

MenuStrip chứa cấu trúc thanh thực đơn dành cho Form. Sử dụng hộp công cụ Toolbox, ta sẽ thêm được MenuStrip cho Form. MenuStrip tự nó không xuất hiện trên Form, nó chỉ xuất hiện bên dưới. Một hộp xuất hiện ở góc trên bên trái của Form với tiêu đề Type here, chỉ dẫn nơi chúng ta nhập các chọn lựa của thực đơn.

Ví dụ 5.16: Hãy tạo thực đơn sau:

File
Open Ctrl+O
Save Ctrl+S
Print Ctrl+P

Format
Font
Color Foreground
Background



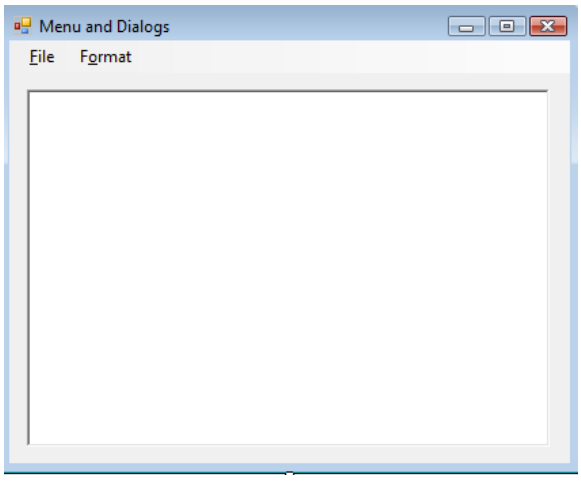
Hình 5.18. Kết quả chèn menu

Trong ví dụ 5.16, để tạo được kí tự gạch chân ta sử dụng dấu & trước kí tự cần gạch chân, chẳng hạn, để tạo được File ta chỉ cần gõ &File. Để tạo được nhóm phím nóng cho menu, ta sử dụng bảng thuộc tính với thuộc tính ShortCutKeys. Chẳng hạn, mục chọn Open (thuộc tính Name: openMenu, ShortCutKeys: Ctrl+O), mục chọn Save (thuộc tính Name: saveMenu, ShortCutKeys: Ctrl+S), mục chọn Print (thuộc tính Name: printMenu, ShortCutKeys: Ctrl+P), mục chọn Font (thuộc tính Name: fontMenu), mục chọn Foreground (thuộc tính Name: foreGroundMenu), mục chọn Background (thuộc tính Name: backGroundMenu).
Trong ví dụ 5.16, ta sẽ thực hiện các sự kiện tương ứng với mỗi mục trong menu, trong đó:

- Open: Chọn một tập tin và sao chép đến RichTextBox.
- Save: Chọn đường dẫn của tập tin và lưu nội dung của RichTextBox vào vị trí đường dẫn đã chọn.
- Print: Chọn lựa các thông số in và đưa thông tin máy in bận.
- Font: Chọn Font cho văn bản trong RichTextBox.
- Foreground: Chọn và đặt màu cận cảnh cho RichTextBox.
- Background: Chọn và đặt màu nền cho RichTextBox.

V.2. RichTextBox

RichTextBox cho phép người sử dụng nhập và sửa đổi dữ liệu văn bản và cung cấp một số công cụ định dạng cao cấp. Chọn RichTextBox từ hộp công cụ ToolBox và thêm vào Form, nới rộng RichTextBox gần bằng kích thước của Form. Ta không thêm bất kỳ điều khiển nào khác ngoài RichTextBox vào Form, do vậy ta sẽ không chừa bất kỳ khoảng trống nào trên Form. Các mục chọn của menu File và menu Format nằm phía trên, bên trái của Form như hình 5.19.



Hình 5.19. Form sau khi chèn RichTextBox

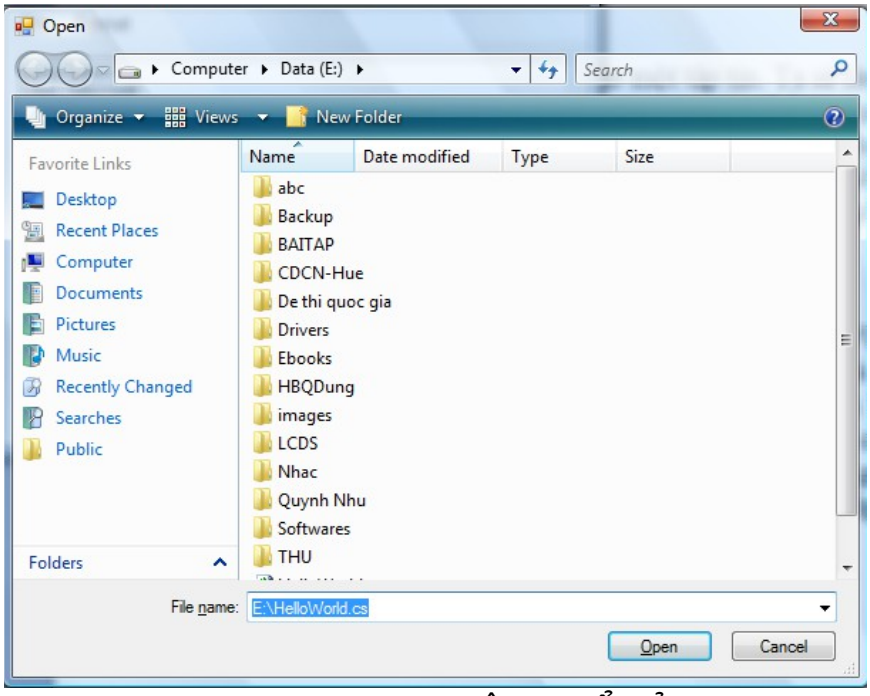
Để richTextBox1 thay đổi kích thước cùng Form khi Form có sự thay đổi kích cỡ, ta đặt thuộc tính Anchor của RichTextBox lần lượt là Top, Bottom, Left, Right; thuộc tính WordWrap là false, nhằm tránh trường hợp ngắt dòng không được gọn.

Thanh cuộn ngang, dọc sẽ tự động thêm vào richTextBox1 khi cần thiết. Thuộc tính ScrollBar sẽ đảm trách công việc này với giá trị mặc định là Both.

RichTextBox có phương thức LoadFile để tải văn bản từ một tập tin, phương thức SaveFile để lưu văn bản vào một tập tin. Ta sẽ làm việc với các phương thức này theo hộp thoại menu trong các mục sau. Để sử dụng các phương thức này, ta cần biết làm thế nào để tên tập tin được tải về hoặc lưu lại. Các hộp thoại sẽ cho phép ta chọn các tên tập tin. Ta cũng cần chú ý đến kiểu tập tin. Riêng đối với ứng dụng đang xây dựng, ta sẽ sử dụng kiểu PlainText gồm các tập tin văn bản đơn giản không có định dạng đặc biệt.

V.3. File Dialogs

Chọn các điều khiển OpenFileDialog và SaveFileDialog từ hộp công cụ Toolbox để thêm vào Form, nhưng openFileDialog và saveFileDialog1 không hiển thị trên Form mà xuất hiện cạnh menuStrip1 (nằm bên dưới Form). Các điều khiển này có phương thức ShowDialog nhằm đưa ra cửa sổ cho phép người dùng lựa chọn tập tin để mở hay lưu. Hình 5.20 cho thấy người sử dụng đã chọn tập tin HelloWorld.cs để mở. Sau khi kích nút Open, tên tập tin được chọn sẽ lưu vào thuộc tính FileName của richTextBox1.



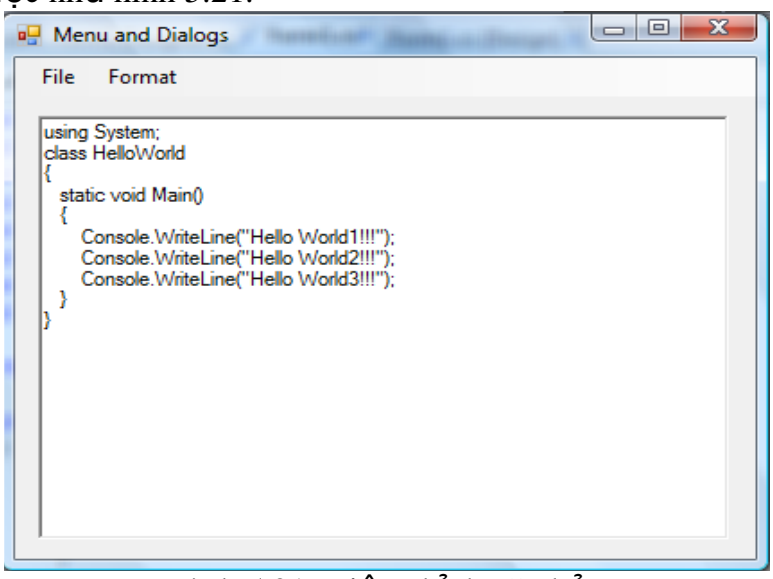
Hình 5.20. Chọn 1 tập tin để mở

Ta muốn khi kích chuột vào mục chọn Open trong menu File, hộp thoại mở tập tin sẽ xuất hiện và rồi tải nội dung tập tin vừa chọn về richTextBox1. Để làm được điều này, ta viết mã cho điều khiển sự kiện của mục chọn Open. Trên cửa sổ thiết kế, kích đôi chuột vào mục chọn Open của menu File, điền vào đoạn mã sau:

```
private void openMenu_Click(object sender, EventArgs e)
```

```
{
   openFileDialog1.ShowDialog();
   richTextBox1.LoadFile(openFileDialog1.FileName,
RichTextBoxStreamType.PlainText);
}
```

Kết quả thu được như hình 5.21.



Hình 5.21. Hiệu chỉnh văn bản

Tương tự cho mục chọn Save của menu File, ta điền vào đoạn mã sau:

```
private void saveMenu_Click(object sender, EventArgs e)
{
   saveFileDialog1.ShowDialog();
   richTextBox1.SaveFile(saveFileDialog1.FileName,
RichTextBoxStreamType.PlainText);
}
```

V.4. PrintDialog

Điều khiển PrintDialog cho phép người sử dụng cấu hình các nhiệm vụ in ấn như chọn lựa máy in, số bảng in, số trang in. Kích trên mục chọn Print của menu File để khởi tạo sự kiện Click. Để thực hiện in ấn văn bản, ta cần viết mã C# trên từng dòng của văn bản, và ta sẽ nghiên cứu việc thực hiện in ấn này sau.

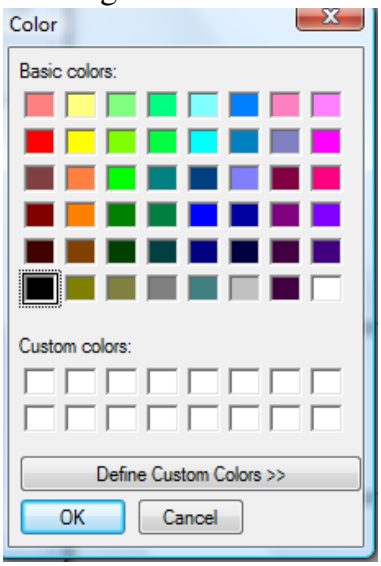
Chọn điều khiển PrintDialog từ hộp công cụ ToolBox, và printDialog1 xuất hiện cạnh saveFileDialog1. Tại sự kiện Click mục chọn Print, ta đưa vào đoạn mã sau:

```
private void printMenu_Click(object sender, EventArgs e)
{
    printDialog1.ShowDialog();
    richTextBox1.Text = "Máy in bận\n Thử lại sau.";
}
```

V.5. ColorDialog

ColorDialog cho phép người sử dụng chọn màu trên bảng màu. Hình 5.21 trình bày bảng màu khi người sử dụng kích chuột vào Foreground trong mục chọn

Color của menu Format. Ta sẽ viết sự kiện để khi người sử dụng kích hoạt vào một màu trên bảng màu, thì văn bản trong richTextBox1 sẽ có màu tương ứng.



Hình 5.21. Hộp thoại màu

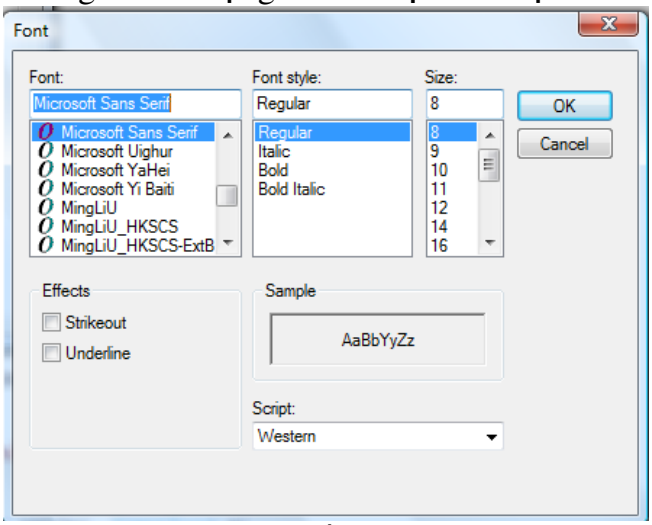
Tại màn hình thiết kế, thêm điều khiển ColorDialog từ hộp công cụ Toolbox vào Form, nó sẽ xuất hiện bên dưới cửa sổ Form. Kích đôi chuột tại Foreground trong mục chọn Color của menu Format, sau đó điền đoạn mã sau vào sự kiện Click.

```
private void foreGroundMenu_Click(object sender, EventArgs e)
{
   colorDialog1.ShowDialog();
   richTextBox1.ForeColor = colorDialog1.Color;
}
```

Thực hiện tương tự cho mục chọn BackColor.

V.6. FontDialog

Khi người sử dụng kích chọn vào mục chọn menu Font, một hộp thoại Font hiện ra để người sử dụng chọn lựa và thay đổi Font của richTextBox1. Hình 5.22 trình bày bảng Font khi người sử dụng kích chuột vào mục Font của menu Format.



Hình 5.22. Hộp thoại Font

Tại màn hình thiết kế, thêm điều khiển FontDialog từ hộp công cụ Toolbox vào Form, nó sẽ xuất hiện bên dưới cửa sổ Form. Kích đôi chuột tại mục Font của menu Format, sau đó điền đoạn mã sau vào sự kiện Click.

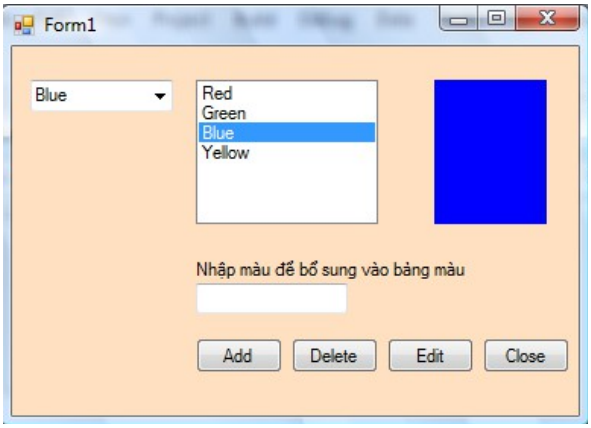
```
private void fontMenu_Click(object sender, EventArgs e)
{
   fontDialog1.ShowDialog();
   richTextBox1.Font = fontDialog1.Font;
//richTextBox1.SelectionFont = fontDialog1.Font;
}
```

B. CÂU HỎI VÀ BÀI TẬP

Bài 5.1. Tạo Form có 1 ComboBox, 1 ListBox, 1 TextBox, 1 Label, 4 Button (thuộc tính lần lượt là Name: btnAdd, Text: &Add; Name: btnDelete, Text: &Delete, Enabled: False; Name: btnEdit, Text: &Edit, Enabled: False; Name: btnClose, Text: &Close), 1 PictureBox như hình vẽ 5.23.

Yêu cầu:

- Khi kích nút Add thì dữ liệu màu từ TextBox sẽ thêm vào ComboBox, ListBox.
- Khi kích nút Delete thì màu được chọn từ ComboBox và ListBox sẽ bị xóa.
- Khi chọn màu trong bảng danh mục màu của ComboBox, thì màu được chọn tương ứng sẽ hiển thị trong ListBox, đồng thời PictureBox sẽ tô màu theo màu đã chọn trong bảng danh mục màu. Ngược lại, Khi chọn màu trong bảng danh mục màu của ListBox, thì màu được chọn tương ứng sẽ hiển thị trong ComboBox, đồng thời PictureBox sẽ tô màu theo màu đã chọn trong bảng danh mục màu.
- Khi kích chọn màu trong ComboBox (hoặc ListBox), và kích chọn vào nút Edit, dữ liệu được chọn trong ComboBox (hoặc ListBox) sẽ được chọn xuống hộp TextBox để ta chỉnh sửa (kích chọn nút Add để bổ sung lên ComboBox hoặc ListBox).

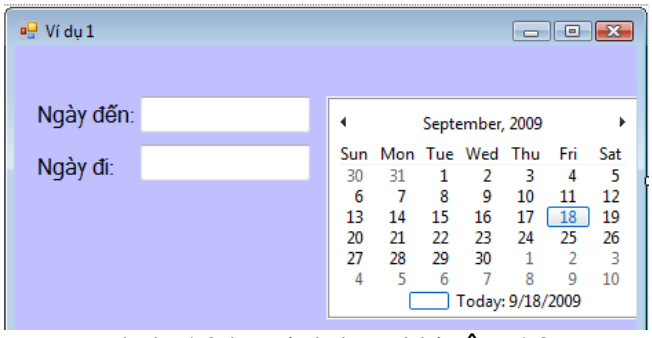


Hình 5.23. Minh họa bài tập 5.1

Bài 5.2. Tạo Form có 2 Label, 2 TextBox, 1 MonthCalendar như hình vẽ 5.24

Yêu cầu:

- Khi kích chọn lần 1 để chọn ngày trên MonthCalendar, ngày được chọn sẽ hiện thị trong TextBox1 "ngày đến".
- Khi kích chọn lần 2 để chọn ngày trên MonthCalendar, ngày được chọn sẽ hiện thị trong TextBox1 "ngày đi".

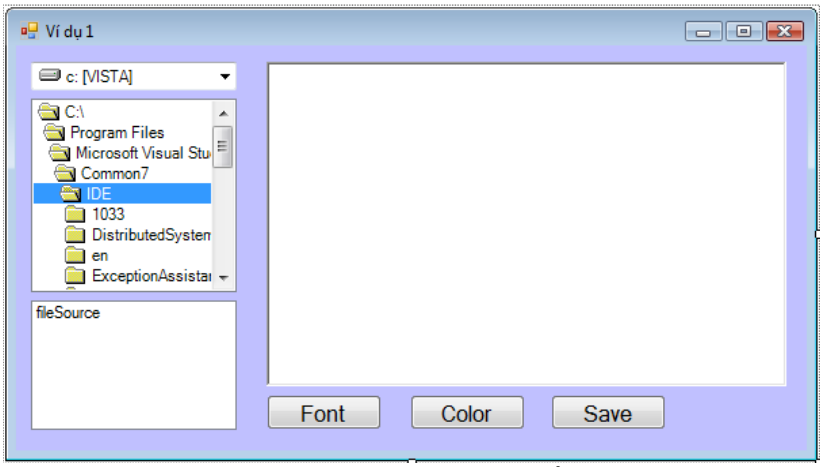


Hình 5.24. Minh họa bài tập 5.2

Bài 5.3. Tạo Form gồm 1 DriveBox (Name: drvSource), 1 DirListBox (Name:dirSource), 1 FileListBox (Name:fileSource, Partern: *.txt), 1 RichTextBox (Name: rtxtFile), 3 Button (thuộc tính Name lần lượt là: btnFont, btnColor, btnSave; thuộc tính Text lần lượt là: Font, Color, Save), 1 FontDialog, 1 ColorDialog như hình 5.25.

Yêu cầu:

- Khi kích chọn tập tin trong FileListBox thì nội dung của tập tin sẽ hiển thị trong RichTextBox.
- Khi kích chọn vào nút lệnh Font, Hộp Font sẽ hiển thị để chọn Font cho vùng dữ liệu được bôi đen trên RichTextBox.
- Khi kích chọn vào nút lệnh Color, Hộp màu sẽ hiển thị để chọn màu cho vùng dữ liệu được bôi đen trên RichTextBox.
- Khi kích chọn vào nút lệnh Save, dữ liệu trên RichTextBox sẽ lưu vào bộ nhớ.



Hình 5.25. Minh họa bài tập 5.3

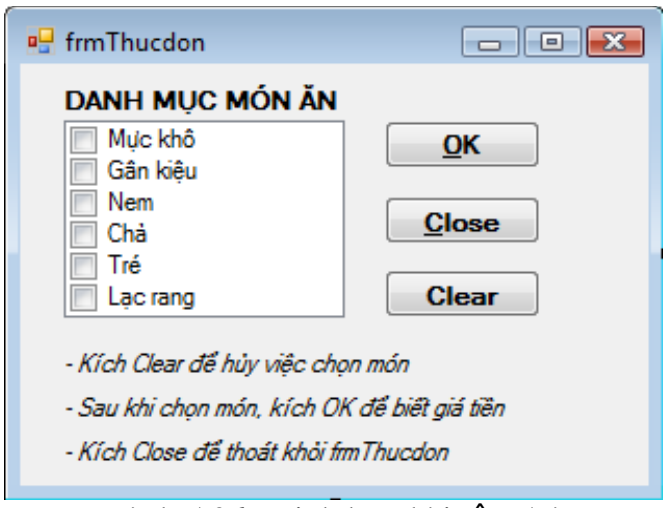
Bài 5.4.

- Tạo Form Thực đơn (Hình 5.26)chứa các công cụ:

+ Nút lệnh OK để hiển thị thông tin giá tiền đối với các món ăn được chọn trong bảng DANH MỤC MÓN ĂN. Biết giá tiền mỗi món ăn như sau: Mực khô – 100, Gân kiệu – 200, Nem – 300, Chả – 400, Tré – 500, Lạc rang – 600.

+ Nút lệnh Clear để hủy việc chọn lựa các món ăn.

+ Nút lệnh Close để đóng Form thực đơn.



Hình 5.26. Minh họa bài tập 5.4

Hướng dẫn giải bài tập

Bài 5.1.

```
private void btnAdd_Click(object sender, EventArgs e)
{
  if (textBox1.Text != "")
  {
            listBox1.Items.Add(textBox1.Text);
            comboBox1.Items.Add(textBox1.Text);
            listBox1.SelectedItem = listBox1.Items[0];
            textBox1.Clear();
            textBox1.Focus();
            btnDelete.Enabled = true;
            btnEdit.Enabled = true;
  }
  else MessageBox.Show("Chọn màu thêm ở hộp màu", "Chú ý",
MessageBoxButtons.YesNoCancel);
}
private void listBox1_SelectedIndexChanged(object sender, EventArgs e)
{
   try
   {
      pictureBox1.BackColor =
Color.FromName(listBox1.SelectedItem.ToString());
      comboBox1.Text = listBox1.SelectedItem.ToString();
   }
   catch
   {
      try
      {
         pictureBox1.BackColor =
Color.FromName(listBox1.Items[0].ToString());
         comboBox1.Text = listBox1.Items[0].ToString();
      }
      catch { }
   }
}
private void btnClose_Click(object sender, EventArgs e)
{
   this.Close();
}
private void btnDelete_Click(object sender, EventArgs e)
{
   try
   {
      comboBox1.Items.Remove(comboBox1.SelectedItem);
      listBox1.Items.Remove(listBox1.SelectedItem);
```

```
        listBox1.SelectedItem = listBox1.Items[0];
    }
    catch
    {
        listBox1.Items.Clear();
        comboBox1.Items.Clear();
        comboBox1.Text = "";
        pictureBox1.BackColor = Color.Transparent;
        btnDelete.Enabled = false;
        btnEdit.Enabled = false;
    }
}
private void btnEdit_Click(object sender, EventArgs e)
{
    try
    {
        textBox1.Text = listBox1.SelectedItem.ToString();
        comboBox1.Items.Remove(comboBox1.SelectedItem);
        listBox1.Items.Remove(listBox1.SelectedItem);
        listBox1.SelectedItem = listBox1.Items[0];
    }
    catch
    {
        comboBox1.Text = "";
    }
    btnEdit.Enabled = false;
}
```

Bài 5.2.

Trong MonthCalendar, kích chọn sự kiện DateSelected và đặt tên ChonNgay.

Đoạn mã được thực hiện như sau:

```
public partial class frmChinh : Form
{
  bool kt;
  public frmChinh()
  {
    InitializeComponent();
    kt = true;
  }
  private void ChonNgay(object sender, DateRangeEventArgs e)
  {
    if (kt == true)
    {
      textBox1.Text = monthCalendar1.SelectionStart.ToShortDateString();
      kt = false;
    }
    else
```

```
    {
        textBox2.Text = monthCalendar1.SelectionEnd.ToShortDateString();
        kt = true;
      }
    }
  }
```

Bài 5.3.

Tại thanh ToolBox, kích chuột phải tại All Windows Form, chọn Choose items…, lần lượt chọn DirListBox, DriveListBox, FileListBox.

Đoạn mã được thực hiện như sau:

```
private void drvSource_SelectedIndexChanged(object sender, EventArgs e)
{
  dirSource.Path = drvSource.Drive;
}
private void dirSource_SelectedIndexChanged(object sender, EventArgs e)
{
  fileSource.Path = dirSource.Path;
}
private void fileSource_SelectedIndexChanged(object sender, EventArgs e)
{
  rtxtFile.LoadFile(dirSource.Path + "\\" +
fileSource.FileName,RichTextBoxStreamType.RichText);
}
private void btnFont_Click(object sender, EventArgs e)
{
  fontDialog1.ShowDialog();
  rtxtFile.SelectionFont = fontDialog1.Font;
}
private void btnColor_Click(object sender, EventArgs e)
{
  colorDialog1.ShowDialog();
  rtxtFile.SelectionColor = colorDialog1.Color;
}
private void btnSave_Click(object sender, EventArgs e)
{
  rtxtFile.SaveFile(dirSource.Path + "\\" + fileSource.FileName);
}
```

Bài 5.4.

Đoạn mã được thực hiện như sau:

```
  public partial class frmThucdon : Form
  {
    int[] A = { 100, 200, 300, 400, 500 };
    public frmThucdon()
    {
      InitializeComponent();
    }
```

```
    private void BtnOK_Click(object sender, EventArgs e)
    {
       int i,s=0;
       for (i = 0; i < checkedListBox1.Items.Count; i++)
          if(checkedListBox1.GetItemChecked(i))
             s += A[i];
       MessageBox.Show(s.ToString());
    }

    private void BtnClose_Click(object sender, EventArgs e)
    {
       checkedListBox1.ClearSelected();
    }

    private void BtnClear_Click(object sender, EventArgs e)
    {
       for (int i = 0; i < checkedListBox1.Items.Count; i++)
          checkedListBox1.SetItemChecked(i, false);
    }
}
```

MÃ BÀI HỌC LTTQ – 06	BÀI 6 : FILE VÀ REGISTRY	Thời gian (giờ)				
		LT 3	TH 2	BT	KT	TS 5

Mục tiêu:

Sau khi học xong bài này, học viên có khả năng:

- Liệt kê được các thành phần chính của không gian tên IO;
- Trình bày cách lập trình hệ thống;
- Lập trình trên lớp File và Directory;
- Quản lý đối tượng File và Directory;
- Vận dụng lớp File và Directory để quản lý hệ thống File và thư mục;
- Thực hiện các thao tác cài đặt, an toàn với máy tính.

TÓM TẮT BÀI:

- Microsoft cung cấp rất nhiều mô hình đối tượng trực giác mà chúng ta sẽ được khảo sát trong chương này, chúng ta cũng biết cách sử dụng các lớp cơ bản của .NET để thực hiện các nhiệm vụ liên quan đến quản lý các thư mục, File trong hệ thống lưu trữ.
- Để thực hiện các nhiệm vụ như đọc từ file và viết ra file và hệ thống đăng ký (Registry) trong C# dùng các lớp File, Directory.

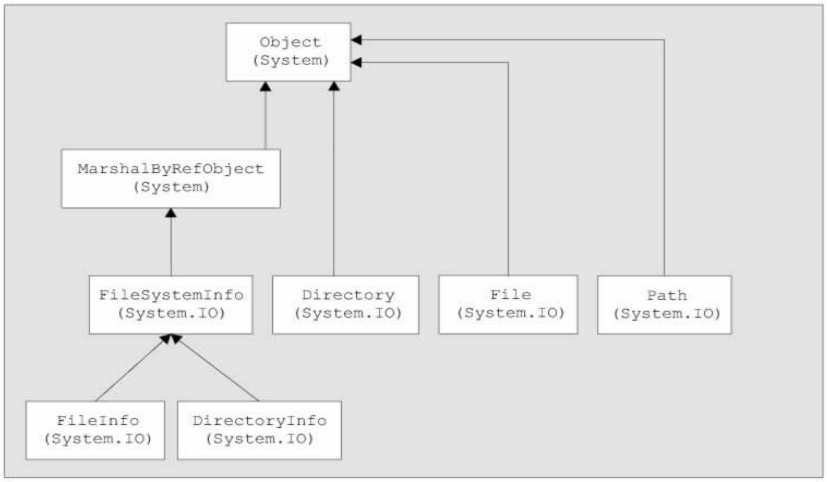
Các vấn đề chính sẽ được đề cập

- Cấu trúc thư mục, tìm kiếm file và folder hiện diện và kiểm tra thuộc tính của chúng.
- Di chuyển, sao chép, huỷ các file và folder
- Đọc và ghi văn bản trong các file
- Đọc và ghi các khoá trong Registry.

A. LÝ THUYẾT

I. QUẢN LÝ TẬP TIN HỆ THỐNG

Các lớp dưới đây được sử dụng để duyệt qua các file hệ thống và cách thức thực hiện như di chuyển, sao chép, huỷ các file được hiển thị trên sơ đồ. Namespace của mỗi lớp đặt trong ngoặc trong sơ đồ.



Mục đích của các lớp trình bày dưới đây:

System.MarshalByRefObject – Lớp đối tượng cơ sở cho các lớp của .NET nó điều khiển từ xa; cho phép điều hành dữ liệu giữa các vùng ứng dụng.
FileSystemInfo – Lớp đối tượng cơ sở biểu diễn các file đối tượng hệ thống
FileInfo and File – Các lớp này thể hiện một file trên file hệ thống
DirectoryInfo and Directory – Các lớp này thể hiện một folder trên tập tin hệ thống
Path – Lớp này chứa các bộ phận tĩnh dùng chế tác các pathnames

I.1. Các lớp .NET thể hiện các File và Folder

Trước khi xem làm thế nào bạn có thể lấy dữ liệu từ các tập tin hoặc viết dữ liệu lên tập tin, .NET hỗ trợ thế nào những thao tác liên quan đến tập tin và thư mục. Trên .NET Framework, namespace System.IO là vùng của các thư viện lớp dành cho những dịch vụ liên quan đến xuất nhập dữ liệu dựa trên tập tin.

Ở hình trên ta thấy System.IO cung cấp cho bạn 4 lớp (Directory, File, DirectoryInfo, FileInfo) cho phép bạn thao tác với tập tin riêng lẽ cũng như tương tác với cấu trúc thư mục của máy. Hai lớp đầu Directory và File cho phép những thao tác tạo, gỡ bỏ và thao tác khác nhau trên các tập tin và thư mục. Hai lớp có liên hệ mật thiết DirectoryInfo và FileInfo cũng có những chức năng tương tự nhưng các thành viên không phải là static, nên muốn triệu gọi các thành viên trước tiên bạn phải tạo một thể hiện đối tượng lớp.Hai lớp đầu Directory và File dẫn xuất từ lớp System.Object trong khi DirectoryInfo và FileInfo dẫn xuất từ lớp FileSystemInfo. Lớp FileSystemInfo là lớp cơ bản abstract có một số thuộc tính và phương thức cung cấp những thông tin liên quan đến một tập tin và thư mục.

I.1.1. Các thuộc tính của lớp cơ bản FileSystemInfo:

Name	Description
CreationTime	Thời gian file, folder được tạo
DirectoryName (FileInfo), Parent (DirectoryInfo)	Tên đường dẫn của folder chức đựng
Exists	Xác định file ,folder hiện hữu
Extension	Tên mở rộng của file, trả về khoảng trắng nếu là folder
FullName	Tên đường dẫn của file ,folder
LastAccessTime	Thời gian file, folder truy xuất làn cuối
LastWriteTime	Time file or folder was last modified
Name	Name of the file or folder
Root	Đường dẫn gốc
Length	Kích thước file tính bằng bytes (chỉ FileInfo)

Các phương thức bạn có thể thực hiện như sau:

Name	Purpose
Create()	Tạo một folder hoặc một file rỗng
Delete()	Huỷ file, folder
MoveTo()	Di chuyển hoặc sửa tên file, folder.

Name	Purpose
CopyTo()	(FileInfo only) Sao chép file, không sao chép fương thức cho folders.
GetDirectories()	(DirectoryInfo only) Trả về một mảng các đối tượng của DirectoryInfo đại diện tát cả folders được chứa trong folder này.
GetFiles()	(DirectoryInfo only) Trả về một mảng các đối tượng của FileInfo đại diện tát cả folders được chứa trong folder này
GetFileSystemObjects()	(DirectoryInfo only) Trả về đối trương FileInfo và DirectoryInfo như mảng của tham khảoFileSystemInfo .

I.1.2.Tạo một đối tượng DirectoryInfo

Bạn bắt đầu làm việc với lớp DirectoryInfo bằng cách khai báo một đường dẫn cụ thể ví dụ : "C:\", "D:\WINNT", . . .nếu bạn muốn truy cập thư mục của ứng dụng đang thi hành bạn dùng ký hiệu "." thí dụ :

```
//Tạo một thư mục mới từ thư mục hiện hành trở đi
DirectoryInfo dirl = new DirectoryInfo(".");

//Tạo mộtthư mục mới từ thư mục C:\Foo\Bar trở đi
DirectoryInfo dir2 = new DirectoryInfo(@"C:\Foo\Bar");
```

I.2. Lớp Path

Lớp Path không là một lớp mà bạn khai báo cụ thể, Đúng ra, nó trình bày các phương thức tỉnh để thực hiện các phép toán trên tên đường dẫn dễ dàng hơn. ví dụ bạn muốn hiển thị tên đường dẫn cho một file ReadMe.txt trong folder C:\My Documents. Bạn có thể tìm đường dẫn đến file như sau:

```
Console.WriteLine(Path.Combine(@"C:\My Documents", "ReadMe.txt"));
```

Sử dụng lớp Path dễ dàng hơn nhiều so với khi bạn thực hiện các ký hiệu bằng tay nhất là bởi vì lớp Path nhận biết được các định dạng khác nhau của đường dẫn trên các hệ điều dành khác nhau. Tại lúc viết, Windows chỉ hỗ trợ hệ điều hành được hỗ trợ bởi .NET nhưng nếu .NET sử dụng trên Unix, Path sẽ dùng / thay cho \ làm dấu ngăn cách tên đường dẫn. Path.Combine() là phương thức mà lớp này thường hay sử dụng nhưng Path cũng thực hiện những phương thức khác để cung cấp thông tin về đường dẫn hoặc yêu cầu định dạng của nó.

Thí dụ : A File Browser

Phần này trình bày ví dụ hướng dẫn làm thế nào để trình duyệt thư mục và hiển thị thuộc tính của file:

Một thí dụ ứng dụng C# gọi FileProperties, nó trình bày một giao diện người dùng đơn giản cho phép bạn trình duyệt các file hệ thống và hiển thị thời gian tạo thành, thời gian truy xuất làn cuối, kích thước file.

Ứng dụng FileProperties nhìn như sau. Bạn gõ tên của folder hoặc tên file trong textbox chính ở phía trên của cửa sổ và nhấn vào nút Display. Nội dung của folder

được tạo trên listbox. Màn hình sẽ hiển thị thuộc tính file đang được dùng xem xét một folder:

Nếu bạn chỉ muốn hiển thị creation time, last access time, and last modification time cho folders – DirectoryInfo thực thi các thuộc tính thích hợp, chúng ta chỉ chọn vào thuộc tính của chúng.

Ta tạo một dự án như sau standard C# Windows application trong Visual Studio.NET, và thêm vào các textboxes và listbox từ Windows Forms area của toolbox. Chúng ta cũng đổi tên các điều khiển một cách trực giác như textBoxInput, textBoxFolder, buttonDisplay, buttonUp, listBoxFiles, listBoxFolders, textBoxFileName, txtBoxCreationTime, textBoxLastAccessTime, textBoxLastWriteTime, và txtBoxFileSize.

Sau đó chúng ta thêm vào đoạn code. Đầu tiên ta cần khai báo sử dụng System.IO namespace:

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
using System.IO;
```

Chúng ta thêm trường thành viên vào main form:

```
public class Form1 : System.Windows.Forms.Form
{
   private string currentFolderPath;
```

currentFolderPath sẽ giữ đường dẫn của thư mục và nội dung sẽ hiển thị trên listboxes.

Chúng ta cần thêm một số sự kiện ngõ ra như sau:

User nhấn nút Display: Trong trường hợp này ta cần có nơi ngõ ra folder. Nếu là một folder ta đưa ra danh sách file và thư mục con, Nếu nó là file, chúng ta cho hiển thị thuộc tính ra textboxes.

User nhấn lên file trên listbox: chúng ta chỉ cho hiển thị thuộc tính của file trên textboxes.

User nhấn trên một folder trên Folders listbox: chúng ta xoá tất cả các điều khiển và hiển thị nội dung của thư mục con trên listboxes.

User nhấn trên nút Up : chúng ta xoá tất cả điều khiển và hiển thị thư mục cha trên listboxes.

Trước tiên ta lên danh sách các sự kiện, và làm rỗng nội dung của các điều khiển:

```
protected void ClearAllFields()
{
  listBoxFolders.Items.Clear();
  listBoxFiles.Items.Clear();
  textBoxFolder.Text = "";
  textBoxFileName.Text = "";
  textBoxCreationTime.Text = "";
  textBoxLastAccessTime.Text = "";
  textBoxLastWriteTime.Text = "";
  textBoxFileSize.Text = "";
}
```

thứ hai ta định nghĩa một fương thức, DisplayFileInfo(), nó thực hiện tiến trình hiển thị thông tin lên textboxes. Phương thức này lấy một thông số, tên đường dẫn file, và làm việc bởi một đối tượng FileInfo dựa trên path này:

```
protected void DisplayFileInfo(string fileFullName)
{
  FileInfo theFile = new FileInfo(fileFullName);
  if (!theFile.Exists)
    throw new FileNotFoundException("File not found: " + fileFullName);
  textBoxFileName.Text = theFile.Name;
  textBoxCreationTime.Text = theFile.CreationTime.ToLongTimeString();
  textBoxLastAccessTime.Text = theFile.LastAccessTime.ToLongDateString();
  textBoxLastWriteTime.Text = theFile.LastWriteTime.ToLongDateString();
  textBoxFileSize.Text = theFile.Length.ToString() + " bytes";
}
```

Ta tạo phương thức DisplayFolderList(), Nó hiển thị nội dung được cho bởi folder trong hai listboxes.

```
protected void DisplayFolderList(string folderFullName)
{
  DirectoryInfo theFolder = new DirectoryInfo(folderFullName);
  if (!theFolder.Exists)
    throw new DirectoryNotFoundException("Folder not found: "
                                + folderFullName);
  ClearAllFields();
  textBoxFolder.Text = theFolder.FullName;
  currentFolderPath = theFolder.FullName;

  // list all subfolders in folder
  foreach(DirectoryInfo nextFolder in theFolder.GetDirectories())
```

```
    listBoxFolders.Items.Add(nextFolder.Name);

  // list all files in folder
  foreach(FileInfo nextFile in theFolder.GetFiles())
    listBoxFiles.Items.Add(nextFile.Name);
}
protected void OnDisplayButtonClick(object sender, EventArgs e)
{
  try
  {
    string folderPath = textBoxInput.Text;
    DirectoryInfo theFolder = new DirectoryInfo(folderPath);
    if (theFolder.Exists)
    {
      DisplayFolderList(theFolder.FullName);
      return;
    }
    FileInfo theFile = new FileInfo(folderPath);
    if (theFile.Exists)
    {
      DisplayFolderList(theFile.Directory.FullName);
      int index = listBoxFiles.Items.IndexOf(theFile.Name);
      listBoxFiles.SetSelected(index, true);
      return;
    }
    throw new FileNotFoundException("There is no file or folder with "
                                    + "this name: " + textBoxInput.Text);
  }
  catch(Exception ex)
  {
    MessageBox.Show(ex.Message);
  }
}
```

Trong đoạn code trên chúng ta thành lập nếu văn bản cung cấp trình bày một folder hoặc một file bởi thể hiện DirectoryInfo và FileInfo instances and khảo sát thuộc tính có sẵn của mỗi đối tượng Nếu không có sẵn chúng ta đưa ra ngoại lệ.

```
protected void OnListBoxFilesSelected(object sender, EventArgs e)
{
  try
  {
    string selectedString = listBoxFiles.SelectedItem.ToString();
    string fullFileName = Path.Combine(currentFolderPath, selectedString);
    DisplayFileInfo(fullFileName);
  }
  catch(Exception ex)
  {
```

```
    MessageBox.Show(ex.Message);
  }
}
```

Bộ quản lý sự kiện cho chọn lựa của folder trong Folders listbox được thi hành trong cách tương tự ngoại trừ trường hợp chúng ta gọi phương thức DisplayFolderList() để cập nhật nội dung của listboxes:

```
protected void OnListBoxFoldersSelected(object sender, EventArgs e)
{
  try
  {
    string selectedString = listBoxFolders.SelectedItem.ToString();
    string fullPathName = Path.Combine(currentFolderPath, selectedString);
    DisplayFolderList(fullPathName);
  }
  catch(Exception ex)
  {
    MessageBox.Show(ex.Message);
  }
}
```

Cuối cùng khi nút Up được nhấn , DisplayFolderList() phải cũng được gọi ngoại trừ lúc này chúng ta cần nhận được đường dẫn của cha thư mục hiện hành được hiển thị.

```
protected void OnUpButtonClick(object sender, EventArgs e)
{
  try
  {
    string folderPath = new FileInfo(currentFolderPath).DirectoryName;
    DisplayFolderList(folderPath);
  }
  catch(Exception ex)
  {
    MessageBox.Show(ex.Message);
  }
}
```

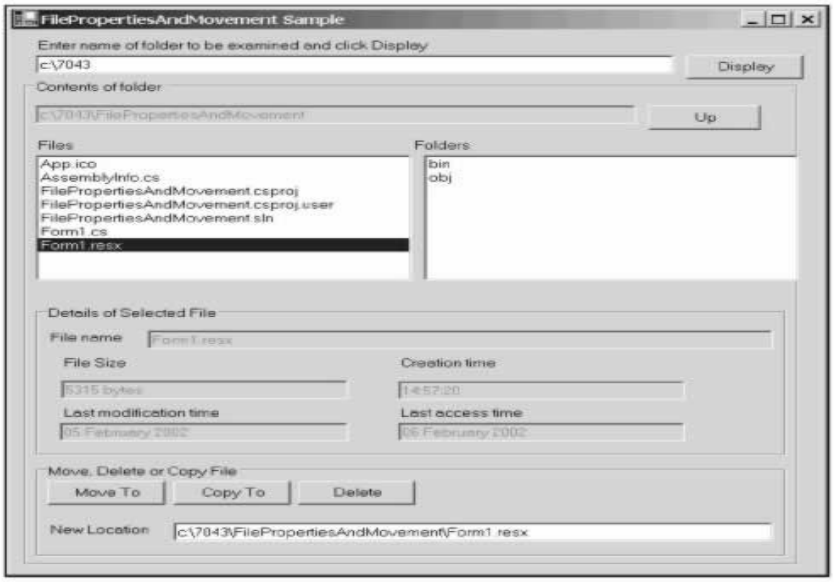
II . DI CHUYỂN, SAO CHÉP, HUỶ FILE

Chúng ta vừa mới đế cập di chuyển và huỷ files hoặc folders bằng phương thức MoveTo() và Delete() của lớp FileInfo và DirectoryInfo. Các phương thức tương đương nhau trên các lớp File và Directory là Move() và Delete(). Lớp FileInfo và File cũng có cách thức thực hiện tương tự, CopyTo() and Copy(). Không có fương thức nào copy các folder, tuy nhiên bạn có thể copy từng file trong folder.

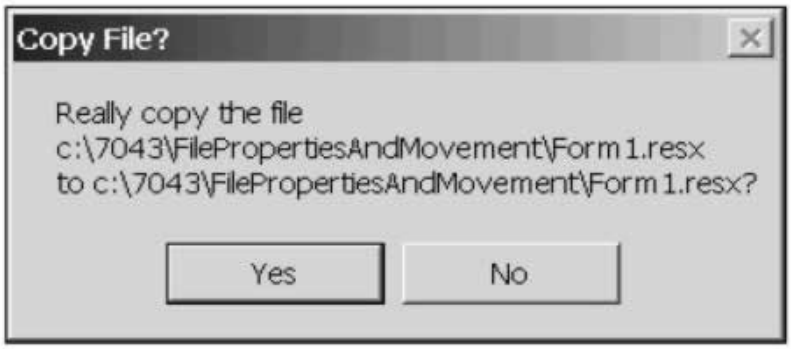
Tất cả các phương thức này đều hoàn toàn trực giác, bạn có thể tìm kiếm chi tiết trong help MSDN. Trong phần này chúng ta sẽ học cách gọi các phương thức tỉnh Move(), Copy(), và Delete() trong lớp File.Để thực hiện chúng ta dùng bài học phần trước FileProperties làm ví dụ, FilePropertiesAndMovement. Ví dụ này ta sẽ thêm tính năng mỗi lần thuộc tính của một file được hiển thị, ứng dụng này sẽ cho ta chọn lựa thêm xoá file hoặc di chuyển hoặc sao chép nó đến vị trí khác

Ví dụ: FilePropertiesAndMovement

Ví dụ mới như sau:



Từ hình trên chúng ta có thể thấy nó rất giống nhau trong lần xuất hiện ở ví dụ FileProperties, Ngoại trừ chúng có thêm một nhóm gồm ba nút button và một textbox tại phía dưới cửa xổ. Các điều khiển này chỉ hiện khi ví dụ hiển thị thuộc tính của một file- Lần khác chúng bị ẩn, Khi thuộc tính của file được hiển thị ,FilePropertiesAndMovement tự động đặt tên đầy đủ đường dẫn của file đó ở cuối của xổ trong textbox. User có thể nhấn bất kỳ buttons để thực hiện phép toán thích hợp. Khi chương trình chạy một message box tương ứng được hiển thị xác nhận hành động.



Để mã hoá chúng ta cần thêm các điều khiển thích hợp, giống như thêm các sự kiện điều khiển cho ví dụ FileProperties . Chúng ta tạo các controls mới với tên buttonDelete, buttonCopyTo, buttonMoveTo, và textBoxNewPath.
Chúng ta sẽ thấy sự kiện điều kiện nhận được khi user nhấn vào Delete button;

```
protected void OnDeleteButtonClick(object sender, EventArgs e)
{
  try
  {
    string filePath = Path.Combine(currentFolderPath,
                          textBoxFileName.Text);
    string query = "Really delete the file\n" + filePath + "?";
    if (MessageBox.Show(query,
```

```
        "Delete File?", MessageBoxButtons.YesNo) == DialogResult.Yes)
      {
        File.Delete(filePath);
        DisplayFolderList(currentFolderPath);
      }
    }
    catch(Exception ex)
    {
      MessageBox.Show("Unable to delete file. The following exception"
                + " occurred:\n" + ex.Message, "Failed");
    }
}
```

Đoạn code thực hiện phương thức này sẽ có phần nắm bắt lỗi, thông báo sẽ lỗi không thể xoá được nếu file xoá đang thực hiện trên một tiến trình khác. Chúng ta xây dựng đường dẫn của file của file bị xoá từ trường CurrentParentPath , Nơi nó chứa dường dẫn thư mục cha, và tên file trong textBoxFileName textbox:
Phương thức di chuyển và sao chép file được cấu trúc :

```
protected void OnMoveButtonClick(object sender, EventArgs e)
{
  try
  {
    string filePath = Path.Combine(currentFolderPath,
                        textBoxFileName.Text);
    string query = "Really move the file\n" + filePath + "\nto "
              + textBoxNewPath.Text + "?";
    if (MessageBox.Show(query,
      "Move File?", MessageBoxButtons.YesNo) == DialogResult.Yes)
    {
      File.Move(filePath, textBoxNewPath.Text);
      DisplayFolderList(currentFolderPath);
    }
  }
  catch(Exception ex)
  {
    MessageBox.Show("Unable to move file. The following exception"
              + " occurred:\n" + ex.Message, "Failed");
  }
}

protected void OnCopyButtonClick(object sender, EventArgs e)
{
  try
  {
    string filePath = Path.Combine(currentFolderPath,
                        textBoxFileName.Text);
    string query = "Really copy the file\n" + filePath + "\nto "
```

```
                + textBoxNewPath.Text + "?";
    if (MessageBox.Show(query,
      "Copy File?", MessageBoxButtons.YesNo) == DialogResult.Yes)
    {
      File.Copy(filePath, textBoxNewPath.Text);
      DisplayFolderList(currentFolderPath);
    }
  }
  catch(Exception ex)
  {
    MessageBox.Show("Unable to copy file. The following exception"
              + " occurred:\n" + ex.Message, "Failed");
  }
}
```

Chúng ta cũng tạo buttons và textbox mới được đánh dấu enabled và disabled ở thời điểm thích hợp để enable chúng khi chúng ta hiển thị nội dung của file, chúng ta cần thêm đoạn code sau:

```
protected void DisplayFileInfo(string fileFullName)
{
  FileInfo theFile = new FileInfo(fileFullName);
  if (!theFile.Exists)
    throw new FileNotFoundException("File not found: " + fileFullName);

  textBoxFileName.Text = theFile.Name;
  textBoxCreationTime.Text = theFile.CreationTime.ToLongTimeString();
  textBoxLastAccessTime.Text = theFile.LastAccessTime.ToLongDateString();
  textBoxLastWriteTime.Text = theFile.LastWriteTime.ToLongDateString();
  textBoxFileSize.Text = theFile.Length.ToString() + " bytes";

  // enable move, copy, delete buttons
  textBoxNewPath.Text = theFile.FullName;
  textBoxNewPath.Enabled = true;
  buttonCopyTo.Enabled = true;
  buttonDelete.Enabled = true;
  buttonMoveTo.Enabled = true;
}
```

Chúng ta cũng cần thay đổi DisplayFolderList:

```
protected void DisplayFolderList(string folderFullName)
{
  DirectoryInfo theFolder = new DirectoryInfo(folderFullName);
  if (!theFolder.Exists)
    throw new DirectoryNotFoundException("Folder not found: " + folderFullName);

  ClearAllFields();
  DisableMoveFeatures();
  textBoxFolder.Text = theFolder.FullName;
```

```
  currentFolderPath = theFolder.FullName;

  // list all subfolders in folder
  foreach(DirectoryInfo nextFolder in theFolder.GetDirectories())
    listBoxFolders.Items.Add(NextFolder.Name);

  // list all files in folder
  foreach(FileInfo nextFile in theFolder.GetFiles())
    listBoxFiles.Items.Add(NextFile.Name);
}
```

DisableMoveFeatures là một hàm tiện ích nhỏ nó disables các controls mới:

```
    void DisableMoveFeatures()
    {
      textBoxNewPath.Text = "";
      textBoxNewPath.Enabled = false;
      buttonCopyTo.Enabled = false;
      buttonDelete.Enabled = false;
      buttonMoveTo.Enabled = false;
    }
```

Chúng ta cần thêm phương thức ClearAllFields() để xoá các textbox thêm vào:

```
    protected void ClearAllFields()
    {
      listBoxFolders.Items.Clear();
      listBoxFiles.Items.Clear();
      textBoxFolder.Text = "";
      textBoxFileName.Text = "";
      textBoxCreationTime.Text = "";
      textBoxLastAccessTime.Text = "";
      textBoxLastWriteTime.Text = "";
      textBoxFileSize.Text = "";
      textBoxNewPath.Text = "";
    }
```

III. ĐỌC VÀ VIẾT VÀO FILE

Đọc và viết vào files nói chung rất đơn giản; tuy nhiên , Điều này không phải bắt buộc biết các đối tượng DirectoryInfo hoặc FileInfo mà chúng ta vừa khảo sát.Thay vào đó chúng ta phải biết một số lớp trình bày nội dung chung gọi là stream, Điều này chúng ta sẽ khảo sát sau đây.

III.1.Streams

Đọc và viết dữ liệu sẽ được thực hiện thông qua lớp stream. Stream là dòng dữ liệu chảy đi. Đây là một thực thể (entity) có khả năng nhận được hoặc tạo ra một "nhúm" dữ liệu. System.IO.Stream là một lớp abstract định nghĩa một số thành viên chịu hỗ trợ việc đọc/viết đồng bộ (synchronus) hoặc không đồng bộ (asynchronous) đối với khối trữ tin (nghĩa là một tập tin trên đĩa hoặc tập tin trên ký ức).

Vì Stream là một lớp abstract, nên bạn chỉ có thể làm việc với những lớp được dẫn xuất từ Stream. Các hậu duệ của Stream tượng trưng dữ liệu như là một dòng dữ liệu thô dạng bytes (thay vì dữ liệu dạng văn bản). Ngoài ra, các lớp được dẫn xuất từ Stream hỗ trợ việc truy tìm (seek) nghĩa là một tiến trình nhận lấy và điều chỉnh vị trí trên một dòng chảy. Trước khi tìm hiểu những chức năng mà lớp Stream cung cấp, bạn nên xem qua các thành viên của lớp Stream.

Ý tưởng của stream đã có từ lâu. Một stream là một đối tượng dùng để chuyển dữ liệu. Dữ liệu có thể được truyền theo hai hướng:

Nếu dữ liệu được truyền từ nguồn bên ngoài vào trong chương trình của bạn, ta gọi là đọc dữ liệu.

Nếu dữ liệu được truyền từ chương trình của bạn ra nguồn bên ngoài , ta gọi là viết dữ liệu

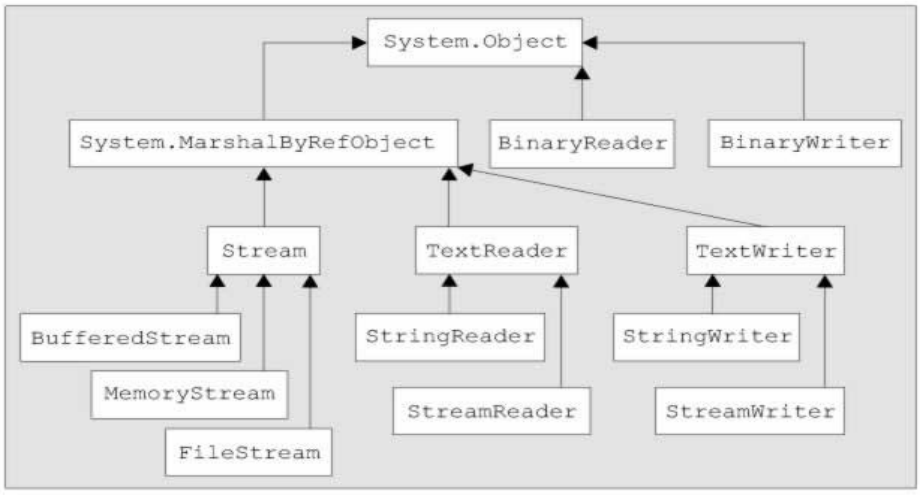
Thường thì nguồn bên ngoài sẽ là một file, ngoài ra nó còn bao gồm cả trường hợp sau:

Đọc hoặc ghi dữ liệu trên mạng dùng giao thức mạng

Đọc hoặc ghi đến một đường ống chỉ định

Đọc hoặc ghi đến một vùng của bộ nhớ

Các lớp có mối liên hệ trong namespace System.IO như hình sau:



III.2.Làm việc với Binary Files

Reading and writing to binary files thường được làm việc với lớp FileStream . Làm việc với FileStream

Lớp FileStream đem lại việc thi công cho những thành viên của lớp abstract Stream theo một thể thức thích hợp đối với các file-base streaming giống như các lớp DirectoryInfo và FileInfo, lớp FileStream cho phép mở những tập tin hiện hữu cũng như tạo mới file. Khi tạo tập tin , lớp FileStream thường dùng những enum FileMode, FileAccess và FileShare

```
// tạo một tập tin mới trên thư mục làm việc
FileStream myFStream = new FileStream("test.dat",FileMode.OpenOrCreate, FileAccess.ReadWrite);
```

The FileStream Class

FileStream được sử dụng đọc và viết dữ liệu vào hoặc từ một file. Để khởi tạo một FileStream, bạn cần 4 phần sau:

file bạn muốn truy xuất .

mode, cho biết bạn muốn mở file như thế nào.

access, cho biết bạn muốn truy xuất file như thế nào – bạn định đọc hoặc viết file hoặc cả hai.

share access – khả năng truy xuất file.

Enumeration	Values
FileMode	Append, Create, CreateNew, Open, OpenOrCreate, or Truncate
FileAccess	Read, ReadWrite, or Write
FileShare	Inheritable, None, Read, ReadWrite, or Write

III.3. Làm việc với BufferedStream

Khi bạn triệu gọi hàm Read() thì một công tác đọc dữ liệu cho đầy buffer từ đĩa được tiến hành. Tuy nhiên, để cho có hiệu năng, hệ điều hành thường phải đọc trong một lúc một khối lượng lớn dữ liệu tạm thời trữ trên bufer. Buffer hoạt động như mọt kho hàng.

Một đối tượng Bufered stream cho phép hệ điều hành tạo buffer riêng cho mình dùng, rồi đọc dữ liệu vào hoặc viết dữ liệu lên ổ đĩa theo một khối lượng dữ liệu nào đó mà hệ điều hành thấy là có hiệu năng. Tuy nhiên, bạn xũng có thể ấn định chiều dài khối dữ liệu. Nhưng bạn nhớ cho là buffer sẽ chiêmd chỗ trong ký ức chứ không phải trên đĩa từ. Hiệu quả sử dụng đến buffer là ciệc xuất nhập dữ liệu chạy nhanh hơn.

Một đối tượng BufferedStream được hình thành xung quanh một đối tượng Stream mà bạn đã tạo ra trước đó. Muốn sử dụng đến một BufferedStream bạn bắt đầu tạo một đối tượng Stream thông thường như trong thí dụ :

```
stream inputstream = File.OpenRead(@"C;\test\source\folder3.cs ");
stream outputstream = File.Openwrite(@"C:test\source\folder3.bak");
```

Một khi bạn đã có stream bình thường, bạn trao đối tượng này cho hàm constructor của buffere stream:

```
BufferedStrream bufInput = new BufferedStream(inputstream);
BufferedStream bufOutput =new BufferedStream(outputstream);
```

Sau đó, bạn sử dụng BufferedStream như là một stream bình thường, bạn triệu gọi hàm Read() hoặc Write() như bạn đã làm trước kia. Hệ điều hành lo việc quản lý vùng đêm:

```
while ((bytesRead = bufInput.Read(buffer, 0, SIZE_BUFF))>0)
{  bufOutput.Write(buffer, 0, bytesRead);
}
```

Chỉ có một khác biệt mà bạn phải nhớ cho là phải tuôn ghi (flush) nội dung của buffer khi bạn muốn bảo đảm là dữ liệu được ghi lên đĩa.

```
bufOutput.Flush();
```

Lệnh trên bảo hệ điều hành lấy toàn bộ dữ liệu trên buffer cho tuôn ra ghi lên tập tin trên đĩa.

III.4 Làm việc với những tập tin văn bản

Nếu bạn biết file bạn đang làm việc (đọc/viết) thuộc loại văn bản nghĩa là dữ liệu kiểu string, thì bạn nên nghĩ đến việc sử dụng đến các lớp StreamReader và StreamWriter. Cả hai lớp theo mặc nhiên làm việc với ký tự Unicode. Tuy nhiên bạn

có thể thay đổi điều này bằng cách cung cấp một đối tượng quy chiếu được cấu hình một cách thích hợp theo System.Text.Reference. Nói tóm lại hai lớp này được thiết kế để thao tác dễ dàng các tập tin loại văn bản.

Lớp StreamReader được dẫn xuất từ một lớp abstract mang tên TextReader cũng giống như String Reader. Lớp cơ bản TextReader cung cấp một số chức năng hạn chế cho mỗi hậu duệ, đặc biệt khả năng đọc và "liếc nhìn" (peek) lên một dòng ký tự (character stream).

Lớp StreamWriter và StringWriter cũng được dẫn xuất từ một lớp abstract mang tên TextWriter; lớp này định nghĩa những thành viên cho phép các lớp dẫn xuất viết những dữ liệu văn bản lên một dòng văn bản nào đó

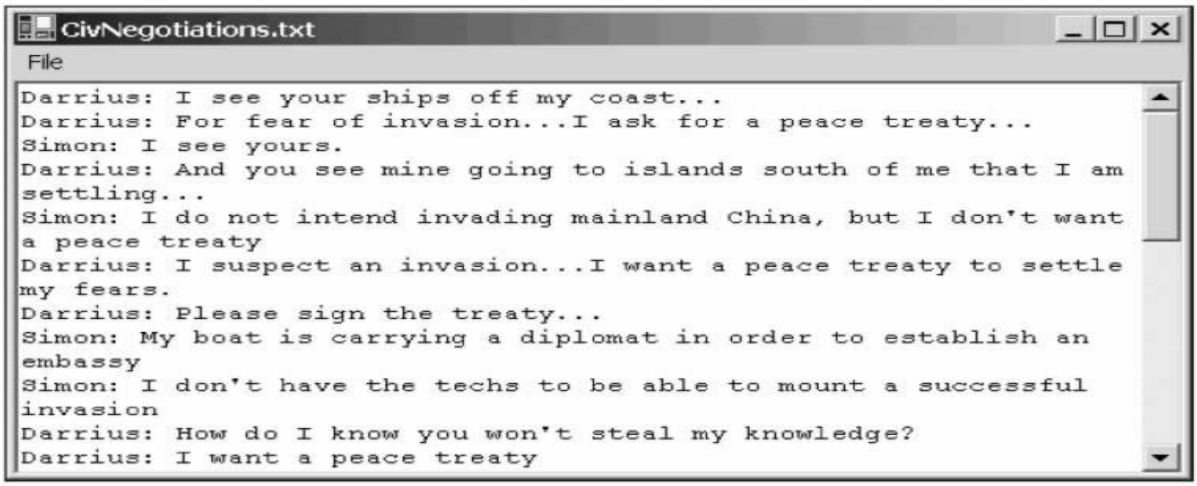
Các thành viên của lớp TextWriter

Tên thành viên	Ý nghĩa
Close()	Cho đóng lại các writer và giải phóng mọi nguồn lực chiếm dụng
Flush()	Cho xoá sạch tất cả các buffer đối với writer hiện hành
NewLine	Thuộc tính này dùng làm hằng sang hằng
Write()	Viết một hằng lên text stream không có newline constant
WriteLine()	Viết một hằng lên text stream có newline constant

Ví dụ đọc, viết một tập tin văn bản:

Ví dụ ReadWriteText trình bày cách sử dụng của lớp StreamReader và StreamWriter. Nó trình bày file được đọc vào và hiển thị Nó cũng có thể lưu file. Nó sẽ lưu bất kỳ file ở định dạng Unicode .

Màn hình trình bày ReadWriteText được dùng hiển thị file CivNegotiations. Chúng ta có thể đọc được ở nhiều định dạng file khác.



Chúng ta nhìn vào đoạn mã sau. Trước tiên ta thêm câu lệnh using , Từ đây bên cạnh System.IO, chúng ta sử dụng lớp StringBuilder từ System.Text namespace để xây dựng chuỗi trong textbox:

```
using System.IO;
using System.Text;
```

Tiếp theo chúng ta thêm các trường cho lớp main form

```
public class Form1 : System.Windows.Forms.Form
{
  private OpenFileDialog chooseOpenFileDialog = new OpenFileDialog();
  private string chosenFile;
```

Chúng ta cũng cần thêm vài chuẩn mã Windows Forms để thực hiện điều khiển cho menu và hộp thoại:

```
  public Form1()
  {
    InitializeComponent();
    menuFileOpen.Click += new EventHandler(OnFileOpen);
    chooseOpenFileDialog.FileOk += new
      CancelEventHandler(OnOpenFileDialogOK);
  }
  void OnFileOpen(object Sender, EventArgs e)
  {
    chooseOpenFileDialog.ShowDialog();
  }
  void OnOpenFileDialogOK(object Sender, CancelEventArgs e)
  {
    chosenFile = chooseOpenFileDialog.FileName;
    this.Text = Path.GetFileName(chosenFile);
    DisplayFile();
  }
```

Từ đây chúng ta thấy mỗi khi người sử dụng nhấn OK để chọn một file trong hộp thoại, chúng ta gọi phương thức DisplayFile(), dùng để đọc file.

```
  void DisplayFile()
  {
    int nCols = 16;
    FileStream inStream = new FileStream(chosenFile, FileMode.Open,
                                         FileAccess.Read);
    long nBytesToRead = inStream.Length;
    if (nBytesToRead > 65536/4)
      nBytesToRead = 65536/4;
    int nLines = (int)(nBytesToRead/nCols) + 1;
    string [] lines = new string[nLines];
    int nBytesRead = 0;
    for (int i=0 ; i<nLines ; i++)
    {
      StringBuilder nextLine = new StringBuilder();
      nextLine.Capacity = 4*nCols;
      for (int j = 0 ; j<nCols ; j++)
      {
        int nextByte = inStream.ReadByte();
        nBytesRead++;
        if (nextByte < 0 || nBytesRead > 65536)
          break;
```

```
            char nextChar = (char)nextByte;
            if (nextChar < 16)
              nextLine.Append(" x0" + string.Format("{0,1:X}",
                                    (int)nextChar));
            else if
              (char.IsLetterOrDigit(nextChar) ||
                            char.IsPunctuation(nextChar))
              nextLine.Append("  " + nextChar + " ");
            else
              nextLine.Append(" x" + string.Format("{0,2:X}",
                             (int)nextChar));
          }
          lines[i] = nextLine.ToString();
        }
        inStream.Close();
        this.textBoxContents.Lines = lines;
    }
```

Như vậy chúng ta đã mở được file nhờ phương thức DisplayFile(). bây giờ chúng ta xử lý cách để lưu file chúng ta thêm đoạn mã SaveFile(). Bạn nhìn vào phương thức SaveFile() chúng ta viết mỗi dòng ra textbox, bằng stream StreamWriter

```
    void SaveFile()
    {
      StreamWriter sw = new StreamWriter(chosenFile, false,
                        Encoding.Unicode);
      foreach (string line in textBoxContents.Lines)
        sw.WriteLine(line);
      sw.Close();
    }
```

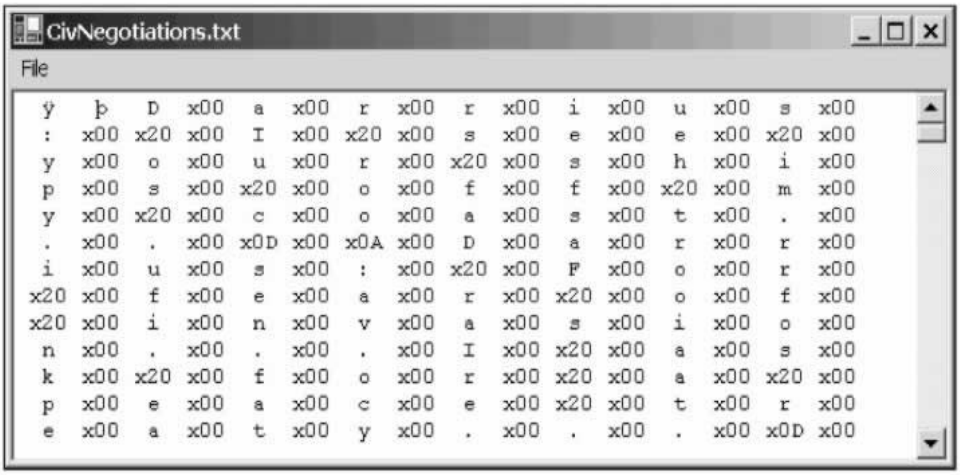
Bây giờ ta xem xét làm thế nào file được đọc vào. Trong quá trình xử lý thực sự chúng ta không biết có bao nhiêu dòng sẽ được chứa (cũng có nghĩa là có bao nhiêu ký tự (char)13(char)10 tuần tự trong file đến khi nào kết thúc file) Chúng ta giải quyết vấn đề này bằng cách ban đầu đọc file vào trong lớp đại diện StringCollection, được nằm trong System.Collections.Specialized namespace. Lớp này được thiết kế để giữ một bộ của chuỗi có thể được mở rộng một cách linh hoạt. Nó thực thi hai phương thức : Add(), nó thêm một chuỗi vào bộ chọn lựa (collection) , và CopyTo(), nó sao chép string collection vào trong một mảng. Mỗi thành phần của đối tượng StringCollection object sẽ giữ 1 hàng của file.

Bây giờ chúng ta sẽ xem xét phương thức ReadFileIntoStringCollection() . Chúng ta sử dụng StreamReader để đọc trong mỗi hàng. Khó khăn chính là cần đếm ký tự đọc để chắc chúng ta không vượt quá khả năng chứa đựng của textbox:

```
    StringCollection ReadFileIntoStringCollection()
    {
      const int MaxBytes = 65536;
      StreamReader sr = new StreamReader(chosenFile);
      StringCollection result = new StringCollection();
```

```
        int nBytesRead = 0;
        string nextLine;
        while ( (nextLine = sr.ReadLine()) != null)
        {
          nBytesRead += nextLine.Length;
          if (nBytesRead > MaxBytes)
            break;
          result.Add(nextLine);
        }
        sr.Close();
        return result;
    }
```

Đến đây đoạn mã được hoàn thành.



IV. ĐỌC VÀ VIẾT VÀO REGISTRY

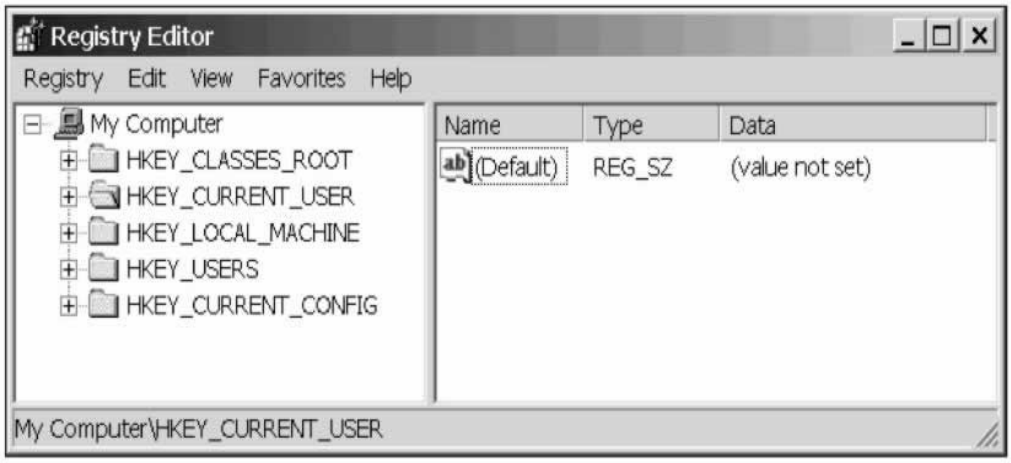
Trong các của Windows từ Windows 95 trở đi Registry là trung tâm lưu trữ tất cả các thông tin cấu hình liên quan đến cài đặt Windows, sở thích người dùng, phần mền cài đặt, thiết bị. Hầu hết tất cả các phần mềm thương mại sử dụng Registry để chứa thông tin của chính nó, và các thành phần COM phải được đặt thông tin của chúng trong Registry để mà được gọi bởi các ứng dụng khách..NET Framework đã giảm sự quan trọng của Registry đối với ứng dụng, vì assembly đã trở thành "tự cung tự cấp" do đó không cần thông tin đặc điết để trữ trên Registry. Registry giờ đây chỉ là nơi tiện lợi để bạn trữ thông tin về sở thích của người sử dụng (user preference). Namespace Microsoft.Win32 định nghĩa một vài lớp cho phép đọc hoặc viết system registry một cách dễ dàng.

Trước tiên chúng ta cùng xem lại cấu trúc của Registry

IV.1. The Registry

Registry có một cấu trúc đẳng cấp giống như hệ thống các tập tin (file system). Cách thông thường để nhìn xem hoặc thay đổi nội dung của Registry là với một trong hai tiên ích: regedit.exe hoặc regedt32.exe hiện diện trong tất cả các phiên bản Windows, từ khi Window 95 trở thành chuẩn. Còn Regedt32.exe thì chỉ hiện diện trong Windows NT và Windows 2000, ít thân thiện so với regedit.exe nhưng cho phép truy cập vào thông tin an ninh mà regedit không có khả năng nhìn xem. Trong phần này chúng ta sử dụng regedit.exe tại khung đối thoại Run hoặc command promt

Khi bạn khởi chạy regedit đầu tiên bạn sẽ thấy hình sau đây:



Regedit có giao diện mang dáng dấp treeview/listview giống như Windows Explorer, khớp với cấu trúc đẳng cấp của bản thân Registry . Tuy nhiên chúng ta sẽ thấy có vài sự khác biệt.

Trong một file system, các mắt cấp chóp có thể được xem là những partitions trên ổ đĩa , C:\, D:\, ... Trong Registry,tương đương với partition là registry hive. Các khuôn này có định và không thể thay đổi và có cả thảy là bảy.

HKEY_CLASSES_ROOT (HKCR) chứa những chi tiết và các loại tập tin(.txt, .doc, and so on), và những ứng dụng nào có khả năng mở các tập tin loại nào. Ngoài ra nó còn chứa thông tin đăng ký đối với tất cả các cấu kiện COM (chiếm phần lớn Registry, vì Windows mang theo vô số thành phần COM).

HKEY_CURRENT_USER (HKCU) chứa chi tiết liên quan đến sở thích của người sử dụng hiện đang đang nhập trên máy tính

HKEY_LOCAL_MACHINE (HKLM) là hive đồ sộ chứa chi tiết tất cả phần mềm và phấn cứng được cài đặt trên máy tính

HKEY_USERS (HKUSR) chứa chi tiết liên quan đến sở thích của tất cả người sử dụng Như bạn có thể chờ đợi, nó cũng chứa hive HKCU đơn giản là một ánh xạ lên một trong những key trên HKEY_USERS.

HKEY_CURRENT_CONFIG (HKCF) chứa đựng chi tiết liên quan đến phần cứng máy tính.

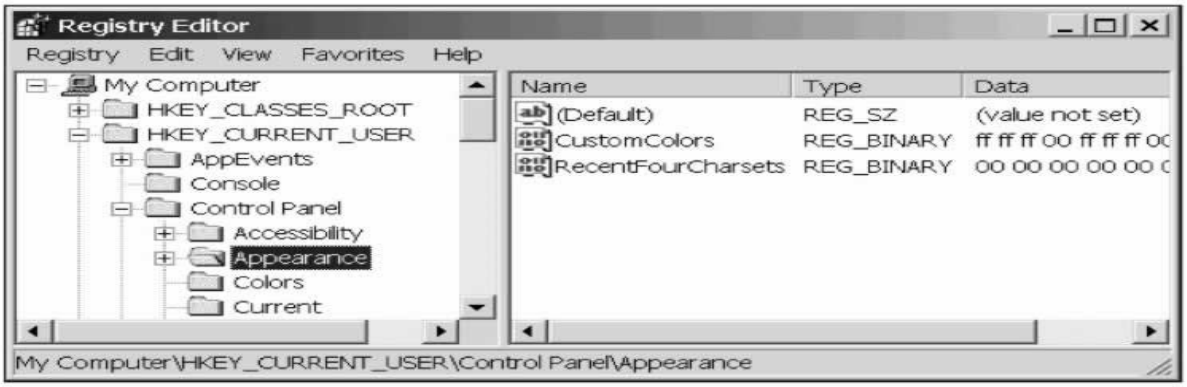
Phần còn lại là hai key chứa thông tin mang tình trạng tạm thời và thay đổi thường xuyên:

HKEY_DYN_DATA là một container tổng quát đối với bất cứ dữ liệu volatile nào cần lưu trữ đâu đó trên Registry

HKEY_PERFORMANCE_DATA chứa thông tin liên quan đến thành tích ứng dụng đang chạy.

Trong lòng các hive là một cấu trúc cây gồm các Registry key. Mỗi key (mục khoá) cũnggiống như một folder hoặc file trong một file system. Tuy nhiên có một khác biệt rất quan trọng . File system phân biệt giữa các files và folders nhưng Registry hiện diện chỉ toàn là key. Một key có thể chứa cả dữ liệu và các key khác.

Nếu một key chứa dữ liệu thì lúc này nó sẽ hiện diện như là một loạt các trị Mỗi trị sẽ có một cái tên một kiểu dữ liệu và một trị Một key có thể có một trị mặc nhiên không được đặt tên



Key HKCU\Control Panel\Appearance có 3 bộ trị có mang tên mặc dù trị mặc nhiên không chứa bất cứ dữ liệu nào. Cột Type chi tiết hoá kiểu dữ liệu của mỗi trị. Các mục vào vào Registry có thể được định dạng theo một trong 3 kiểu dữ liệu:

REG_SZ gần như tương đương với .NET string

REG_DWORD gần như tương đương với .NET unit

REG_BINARY bản dãy các byte

IV.2. The .NET Registry Classes

Việc truy cập vào Registry trên .NET sẽ thông qua hai lớp Registry và RegistryKey thuộc amespace Microsoft.Win32. Một thể hiện của lớp RegistryKey tượng trưng cho một registry key. Lớp RegistryKey cung cấp những thành viên cốt lõi cho phép bạn làm việc với registry key.

Lớp RegistryKey sẽ là lớp mà bạn sẽ dùng để làm việc với registry key. Ngược lại lớp Registry là lớp mà bạn chả bao giờ thể hiện. Vai trò của nó là cung cấp cho bạn những thể hiện RegistryKey tượng trưng cho key top-level những hive khác nhau để qua các thuộc tính static và có cả thảy 7 thuộc tính bao gồm ClassesRoot, CurrentConfig, CurrentUser, DynData, LocalMachine, PerformanceData, and Users. chắc chắn bạn đã biết thuộc tính nào chỉ Registry hive nào Do đó muốn có một thể hiện của một RegistryKey tượng trưng cho key HKLM, bạn viết

```
RegistryKey hklm = Registry.LocalMachine;
```

Nếu bạn muốn đọc một vài dữ liệu trên key HKLM\Software\Microsoft, bạn phải đi lấy qui chiếu về key như sau :

```
RegistryKey hklm = Registry.LocalMachine;
RegistryKey hkSoftware = hklm.OpenSubKey("Software");
RegistryKey hkMicrosoft = hkSoftware.OpenSubKey("Microsoft");
```

Một registry key được truy cập theo kiểu này chi cho phép bạn đọc mà thôi, Nếu bạn muốn có khả năng viết lên key (bao gồm viết lên trị của key, tạo hoặc gỡ bỏ cây con cái thuộc quyền), bạn phải sử dụng một OpenSubkey nhận thêm một thông số thứ hai thuộc kiểu bool cho biết quyền read-write đối với key. Ví dụ bạn muốn có khả năng thay đổi key Microsoft

```
RegistryKey hklm = Registry.LocalMachine;
RegistryKey hkSoftware = hklm.OpenSubKey("Software");
RegistryKey hkMicrosoft = hkSoftware.OpenSubKey("Microsoft", true);
```

Phương thức OpenSubKey() là một trong những hàm mà bạn triệu gọi nếu bạn chờ đợi key hiện hữu. Nếu nó không có thì nó sẽ trở về null preference. Còn nếu bạn muốn tạo một key mới bạn sẽ dùng CreateSubKey() (hàm này tự hoạt động cho quyền read write):

```
RegistryKey hklm = Registry.LocalMachine;
RegistryKey hkSoftware = hklm.OpenSubKey("Software");
RegistryKey hkMine = hkSoftware.CreateSubKey("MyOwnSoftware");
```

Một khi bạn đã có registry key bạn muốn đọc hoặc thay đổi, bạn có thể sử dụng các phương thức SetValue() hoặc GetValue() để đặt hoặc để lấy dữ liệu trên key. Thí dụ:

```
RegistryKey hkMine = HkSoftware.CreateSubKey("MyOwnSoftware");
hkMine.SetValue("MyStringValue", "Hello World");
hkMine.SetValue("MyIntValue", 20);
```

Đoạn mã trên sẽ đặt key về hai trị : MyStringValue sẽ mang kiểu dữ liệu REG_SZ, trong khi MyIntValue sẽ mang kiểu dữ liệu REG_DWORD.

RegistryKey.GetValue() cũng như vậy Nó được định nghĩa trả về một quy chiếu đối tượng, nghĩa là trả về một quy chiếu string nếu nó thấy có kiểu dữ liệu REG_SZ, và int nếu phát hiện kiểu dữ liệu REG_DWORD:

```
string stringValue = (string)hkMine.GetValue("MyStringValue");
int intValue = (int)hkMine.GetValue("MyIntValue");
```

Cuối cùng khi xong việc bạn phải cho đóng lại

```
hkMine.Close();
```

Các thành phần của RegistryKey bao gồm thuộc tính và phương thức sau:

Properties

Property Name	Description
Name	Tên của key (read-only)
SubKeyCount	số lượng sub key
ValueCount	Các trị trên key

Methods

Method Name	Purpose
Close()	Đóng lại key
CreateSubKey()	Tạo một subkey của một tên được cho
DeleteSubKey()	Bỏ một key được chỉ định
DeleteSubKeyTree()	Gỡ bỏ một cách đệ quy một subkey
DeleteValue()	Tháo bỏ một tên trị từ một key
GetSubKeyNames()	Trả về một dãy chuỗi chứa tên của subkeys
GetValue()	Trả về một tên trị
GetValueNames()	Trả về một dãy chuỗi chứa tên của tất cả các trị của key
OpenSubKey()	Trả về một tham khảo đến một RegistryKey,hàm này cho tìm lại subkey
SetValue()	Hàm này cho đặt một trị được chỉ định.

MÃ BÀI HỌC LTTQ – 07	BÀI 7 : ĐỒ HOẠ VÀ MỘT SỐ XỬ LÝ NÂNG CAO	Thời gian (giờ)				
		LT 4	TH 5	BT	KT 1	TS 10

Mục tiêu:

Sau khi học xong bài này, học viên có khả năng:

- Liệt kê được các thành phần chính của không tên đồ hoạ
- Trình bày môi trường làm việc đồ hoạ của .NET Framework;
- Trình bày được cấu pháp để thực hiện các đối tượng đồ hoạ trong C#;
- Thiết lặp được toạ độ, các thuộc tính của các đối tượng đồ hoạ;
- Vẽ được các đối tượng đồ hoạ;
- Dùng ngôn ngữ C# để xây dựng để xây dựng các ứng dụng đồ hoạ;
- Thực hiện các thao tác cài đặt, an toàn với máy tính.

TÓM TẮT BÀI:

- Giới thiệu GDI+:
- GDI (Graphics Development Interface) là các API của Windows cung cấp các hàm và các CTDL cần thiết để tạo ra những hình ảnh đồ họa 1 cách nhanh chóng và hiệu quả ra các thiết bị phần cứng (như màn hình, máy in).
- GDI+ cải thiện GDI với các đặc điểm mới và tối tưu các đặc điểm đã có ở GDI..

Các vấn đề chính sẽ được đề cập

- Không gian tên System.Drawing, System.Drawing3D
- Các định nghĩa về toạ độ, đơn vị trong C#
- Vẽ các đối tượng cơ bản : Line, Arc, Rectange, Cricle...
- Các đối tượng đồ hoạ nâng cao.

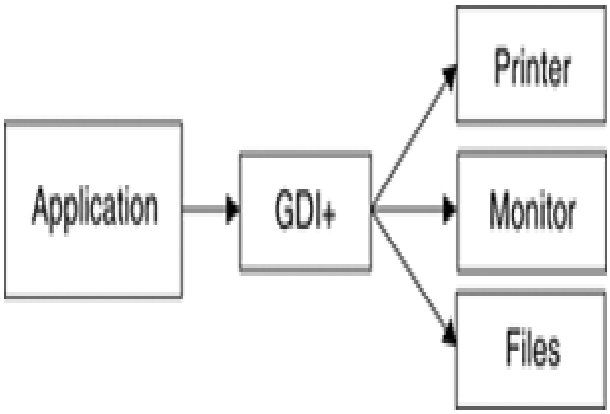
NỘI DUNG

A. LÝ THUYẾT

GDI (Graphics Development Interface) là các API của Windows cung cấp các hàm và các CTDL cần thiết để tạo ra những hình ảnh đồ họa 1 cách nhanh chóng và hiệu quả ra các thiết bị phần cứng (như màn hình, máy in).

GDI+ cải thiện GDI với các đặc điểm mới và tối tưu các đặc điểm đã có ở GDI.

- **Sơ đồ:**



Trong Microsoft Visual Studio, GDI+ có trong khối hợp ngữ System.Drawing.dll, 1 vài namespace trong đó là:

- System.Drawing.Design;
- System.Drawing;
- System.Drawing.Drawing2D;
- System.Drawing.Printing;
- System.Drawing.Text;

Các kiểu dữ liệu GDI+ trên .NET FRAMEWORK là những lớp vỏ bọc bao quanh các hàm API cấp thấp, và không có thêm gì mới. Tuy nhiên, các kiểu dữ liệu GDI+. .NET cung cấp 1 mức độ trừu tượng hóa cao cấp hơn với những hỗ trợ khá thuận lợi về biến đổi hình học, kỹ thuật vuốt mịn các đường cong (antialiasing), và pha màu (pallete blending).

I. GIỚI THIỆU VỀ LỚP GRAPHICS

Trước khi chúng ta vẽ các hình ảnh đồ họa như các đường, hình, ảnh, text.. Với GDI+, chúng ta phải tạo ra 1 đối tượng Graphics đê thực hiện các thao tác vẽ đó.

Một số hàm vẽ và tô màu với Graphisc:

FillEllipse	**Fills the interior of an ellipse defined by a bounding rectangle.**

FillPath	Fills the interior of a path.
FillPie	Fills the interior of a pie section.
FillPolygon	Fills the interior of a polygon defined by an array of points.
FillRectangle	Fills the interior of a rectangle with a Brush.
FillRectangles	Fills the interiors of a series of rectangles with a Brush.
FillRegion	Fills the interior of a Region.

DrawArc	Draws an arc from the specified ellipse.
DrawBezier	Draws a cubic bezier curve.
DrawBeziers	Draws a series of cubic Bezier curves.
DrawClosedCurve	Draws a closed curve defined by an array of points.
DrawCurve	Draws a curve defined by an array of points.
DrawEllipse	Draws an ellipse.
DrawImage	Draws an image.
DrawLine	Draws a line.
DrawPath	Draws the lines and curves defined by a GraphicsPath.
DrawPie	Draws the outline of a pie section.
DrawPolygon	Draws the outline of a polygon.
DrawRectangle	Draws the outline of a rectangle.
DrawString	Draws a string.

II. KHỞI TẠO CÁC ĐỐI TƯỢNG GRAPHIC

Chúng ta có thể tạo ra các đối tượng đồ họa bằng các cách sau:

- Sử dụng Paint Evnet
- Override phương thức Onpaint.
- Sử dụng phương thức CreateGraphisc()
- Sử dụng Object được dẫn xuất từ lớp Image

Sử dụng Paint Event:

- private void Form1_Paint(object sender, PaintEventArgs e)

```
{
    Graphics g = e.Graphics;
    Pen p = new Pen(Color.Blue);
    g.DrawEllipse(p, new Rectangle(10, 10, 50, 50));
}
```

Override phương thức Onpaint

- protected override void OnPaint(PaintEventArgs e)

```
{
    Graphics g = e.Graphics;
    Pen p = new Pen(Color.Blue);
    g.DrawEllipse(p, new Rectangle(60, 60, 50, 50));
}
```

Sử dụng phương thức CreateGraphics()

- private void _CreateGraphics()
  {
  Graphics g = CreateGraphics();
  Pen p = new Pen(Color.Blue);
  g.DrawEllipse(p, new Rectangle(60, 60, 100, 100));
  }

Sử dụng Object được dẫn xuất từ lớp Image

- private void _CreateGraphicsFromImage()
  {
  Size _fullSize = SystemInformation.PrimaryMonitorMaximizedWindowSize;
  Bitmap _bitmap = new Bitmap(_fullSize.Width, _fullSize.Height);
  _g = Graphics.FromImage(_bitmap);
  Rectangle rect = new Rectangle (20, 20, 100, 100);
  _g.DrawRectangle(Pens.Red, rect);
  }

III. CÁC ĐỐI TƯỢNG GRAPHIC.

- Pen
- Brush
- Color
- Font
- Bitmap
- Image

III.1. Pen

- Dùng để vẽ các đường thẳng, đường cong hay các hình với màu sắc, độ dày nét vẽ hay 1 style tùy chọn.
- **Graphics** cung cấp các phương thức để vẽ như DrawRectangle, DrawlEllipse…Còn **Pen** cho phép bạn xác định màu vẽ , kiểu kiểu vẽ và độ dày nét vẽ…
- **Constructor:**

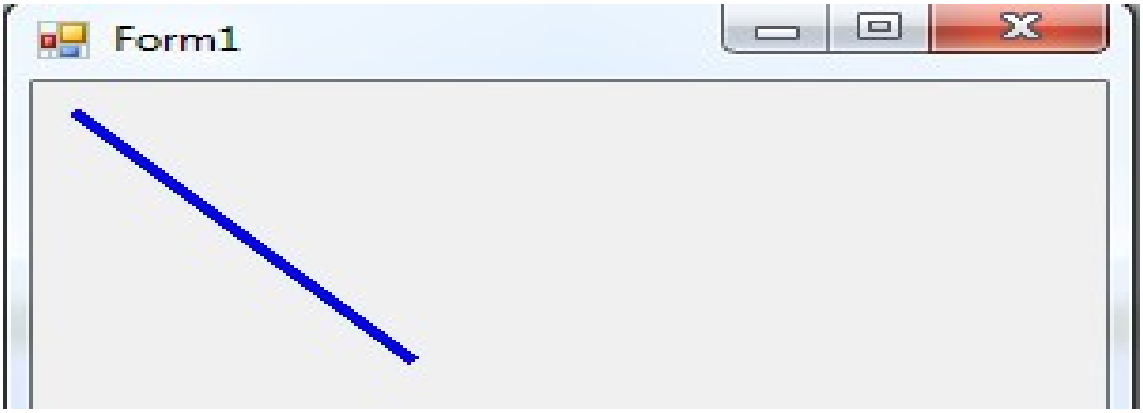
	Name	Description
	Pen(Brush)	Initializes a new instance of the Pen class with the specified Brush.
	Pen(Color)	Initializes a new instance of the Pen class with the specified color.
	Pen(Brush, Single)	Initializes a new instance of the Pen class with the specified Brush and Width.
	Pen(Color, Single)	Initializes a new instance of the Pen class with the specified Color and Width properties.

- **Properties:**

	Name	Description
	Alignment	Gets or sets the alignment for this Pen.
	Brush	Gets or sets the Brush that determines attributes of this Pen.
	Color	Gets or sets the color of this Pen.
	PenType	Gets the style of lines drawn with this Pen.
	Width	Gets or sets the width of this Pen, in units of the Graphics object used for drawing.

- **Ví dụ:**

```
private void Form1_Paint(object sender, PaintEventArgs e)
{
    Graphics g = e.Graphics;
    Pen p = new Pen(Color.Blue, (float)4);
    g.DrawLine(p, new Point(10, 10), new Point(100, 100) );
}
```



III.2. Brush

- Để tô các đối tượng đồ họa như hình chữ nhật, ellipse...
- **Graphics** cung cấp các phương thức để tô như FillRectangle , FillEllipse… Còn **Brush** cho phép bạn xác định màu tô , kiểu tô…
- **Constructor:**

	Name	Description
	Brush	Initializes a new instance of the Brush class.

- Brush là một abstract class vì thế nó không được tạo ra như một đối tượng mà cần phải sử dụng các class con của nó.
- Trong GDI+ có tất cả 5 loại chổi vẽ (Brush) :

```
System.Drawing.Brush
  System.Drawing.Drawing2D.HatchBrush
  System.Drawing.Drawing2D.LinearGradientBrush
  System.Drawing.Drawing2D.PathGradientBrush
  System.Drawing.SolidBrush
  System.Drawing.TextureBrush
```

- **SolidBrush: tô với 1 màu duy nhất.**

```
private void Form1_Paint(object sender, PaintEventArgs e)
{
    Graphics g = e.Graphics;
    Brush brSolid = new SolidBrush(Color.Blue);
    g.FillRectangle(brSolid, new Rectangle(10, 10, 50, 50));
}
```

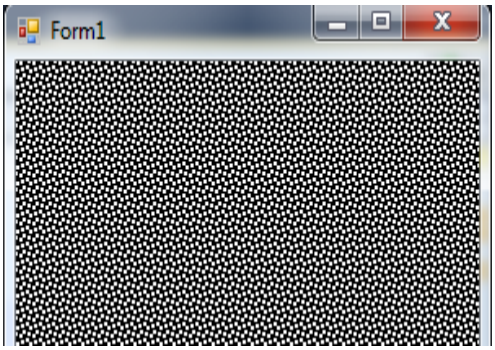
- **TextureBrush**: -Tô với chất liệu lấy từ một tập tin ảnh bất kỳ.

```
private void Form1_Paint(object sender, PaintEventArgs e)
{
    Graphics g = e.Graphics;
    Image img = new Bitmap(@"D:\mypic.bmp");
    Brush brush = new TextureBrush(img);
    Rectangle r = ClientRectangle;

    g.DrawString("Bitmaps as brushes!",
      new Font("Arial", 30,
      FontStyle.Bold | FontStyle.Italic),
      brush,
      r);
}
```

- **HatchBrush:**
  - Tô với mẫu có sẵn
  - Các mẫu tô lấy từ enum **HatchStyle**

```
private void Form1_Paint(object sender, PaintEventArgs e)
{
    Graphics g = e.Graphics;
    HatchBrush hb = new HatchBrush(
      HatchStyle.LargeConfetti,
      Color.White,
      Color.Black);
    g.FillRectangle(hb, this.ClientRectangle);
}
```



- **LinearGradientBrush**: Tô theo kiểu màu tô được chuyển dần từ màu này sang màu khác.Sự chuyển tiếp màu bắt đầu từ một cạnh của hình rồi lan dần qua cạnh khác.

```
private void Form1_Paint(object sender, PaintEventArgs e)
{
    Graphics g = e.Graphics;
    Brush brGradient = new LinearGradientBrush(new Rectangle(0, 0, 30, 100),
                                               Color.Red, Color.Blue, 45, false);
    g.FillRectangle(brGradient, 10, 10, 200, 200);
}
```

- **PathGradientBrush**: -Giống như LinearGradientBrush nhưng sự chuyển tiếp màu bắt đầu từ giữa hình rồi lan dần ra bên ngoài.

```
private static void Form1Paint(object sender, PaintEventArgs e)
{
    Graphics g = e.Graphics;
    Point[] mPoints = new Point[] { new Point(10, 10), new Point(160, 10), new Point(160, 110), new Point(10, 110) };
    Color[] mSurroundColors = new Color[] { Color.Black, Color.Orange, Color.Blue, Color.Violet };
    PathGradientBrush brush = new PathGradientBrush(mPoints);
    brush.CenterColor = Color.Red;
    brush.SurroundColors = mSurroundColors;
    g.FillRectangle(brush, new Rectangle(0, 0, 160, 160));

}
```

III.3. Font

Font là tập hợp hoàn chỉnh các chữ cái, các dấu câu, các con số, và các ký tự đặc biệt, theo một kiểu loại, định dạng (thường hoặc đậm nét), hình dáng (thẳng hoặc nghiêng) và kích cỡ phù hợp và có thể **phân biệt khác nhau**.

Các loại Font trên HĐH Windows:

- Bitmap font
- Vector font
- True type font
- Open type font
- **Class Font:** Dùng để xác định cách định dạng văn bản.

Font(string, float)
Font(string, float, FontStyle)
Font(FontFamily, float)
Font(FontFamily, float, FontStyle)
Font(string, float, GraphicsUnit)
Font(string, float, FontStyle, GraphicsUnit)
Font(FontFamily, float, GraphicsUnit)
Font(string, float, FontStyle, GraphicsUnit, byte, bool)
Font(string, float, FontStyle, GraphicsUnit, byte)
Font(Font, FontStyle)
Font(FontFamily, float, FontStyle, GraphicsUnit)
....

```
private void Form1_Paint(object sender, PaintEventArgs e)
{
    Graphics g = e.Graphics;
    Font f = new Font("Times New Roman", (float)15);
    g.DrawString("Hello World", f, Brushes.Blue, new PointF(10, 10));
}
```

- **FontStyle:**

Member Name	Value	Description
Bold	0	Bold text
Italic	1	Italic text
Regular	2	Normal text
Strikeout	4	Text with a line through the middle
Underline	8	Underlined text

```
private void Form1_Paint(object sender, PaintEventArgs e)
{
    Graphics g = e.Graphics;
    FontStyle style = FontStyle.Bold | FontStyle.Italic;
    Font f = new Font("Times New Roman", (float)15, style);
    g.DrawString("Hello World", f, Brushes.Blue, new PointF(10, 10));
}
```

- **FontFamily:**

```
private void Form1_Paint(object sender, PaintEventArgs e)
{
    Graphics g = e.Graphics;
    FontFamily family = new FontFamily("Times New Roman");
    FontStyle style = FontStyle.Bold | FontStyle.Italic;
    Font f1 = new Font(family, (float) 20, style);
    g.DrawString("Hello World", f1, Brushes.Blue, new PointF(10, 10));
    Font f2 = new Font(family, (float)10, FontStyle.Strikeout);
    g.DrawString("Hello World", f2, Brushes.Blue, new PointF(50, 50));
}
```

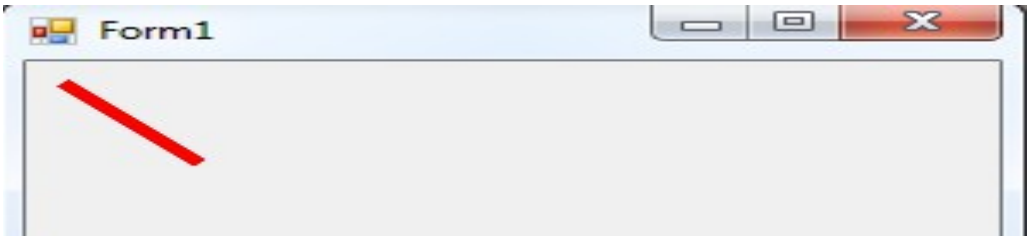
III.4. Color

Trong GDI+,color được biểu diễn bởi một cấu trúc gồm bốn thành phần :

Thành phần	**Giải thích**
A	Transparency
R	Red
G	Green
B	Blue

- **Lớp Color :**
  - Chứa nhiều property đại diện cho các màu.

```
private void Form1_Paint(object sender, PaintEventArgs e)
{
    Graphics g = e.Graphics;
    Color color = Color.Red;
    Pen p = new Pen(color, (float) 5);
    g.DrawLine(p, new PointF(10, 10), new PointF(50, 50));
}
```



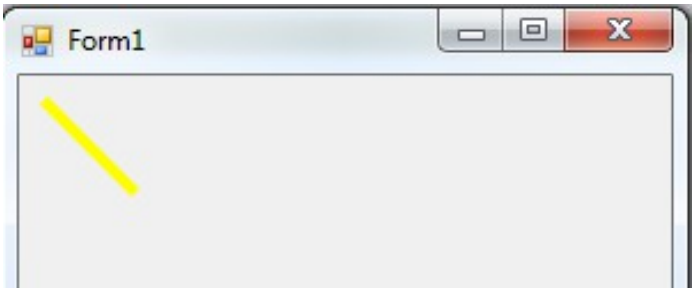
Hổ trợ các phương thức tĩnh để tạo đối tượng Color :

- Color.FromArgb : Trả về đối tượng Color với các giá trị tương ứng

```
private void Form1_Paint(object sender, PaintEventArgs e)
{
    Graphics g = e.Graphics;
    Color color = Color.FromArgb(100, 20, 40, 20);
    Pen p = new Pen(color, (float) 5);
    g.DrawLine(p, new PointF(10, 10), new PointF(50, 50));
}
```

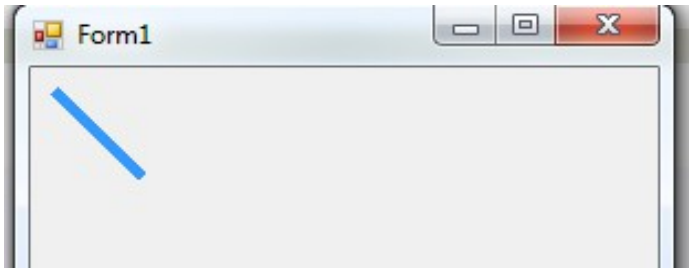
- Color.FromName: Trả về đối tượng Color được định nghĩa sẵn có tên tương ứng.

```
private void Form1_Paint(object sender, PaintEventArgs e)
{
    Graphics g = e.Graphics;
    Color color = Color.FromName("Yellow");
    Pen p = new Pen(color, (float) 5);
    g.DrawLine(p, new PointF(10, 10), new PointF(50, 50));
}
```



Lớp SystemColors :

```
private void Form1_Paint(object sender, PaintEventArgs e)
{
    Graphics g = e.Graphics;
    Color color = SystemColors.MenuHighlight;
    Pen p = new Pen(color, (float) 5);
    g.DrawLine(p, new PointF(10, 10), new PointF(50, 50));
}
```



III.5. Giới thiệu về Bitmap

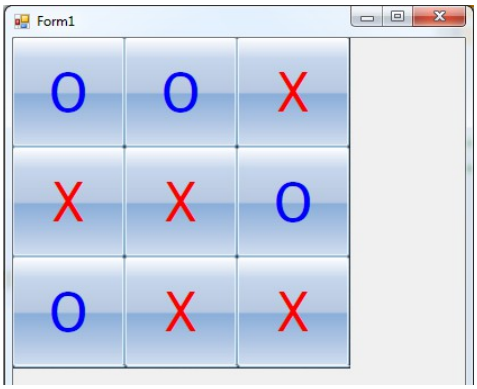
Trong đồ họa máy vi tính, BMP, còn được biết đến với tên tiếng Anh khác là *Windows bitmap*, là một định dạng tập tin hình ảnh khá phổ biến. Các tập tin đồ họa lưu dưới dạng BMP thường có đuôi là .BMP hoặc .DIB (*Device Independent Bitmap*).

- **Tạo 1 ảnh Bitmap:**

	Name	Description
	Bitmap(Image)	Initializes a new instance of the Bitmap class from the specified existing image.
	Bitmap(Stream)	Initializes a new instance of the Bitmap class from the specified data stream.
	Bitmap(String)	Initializes a new instance of the Bitmap class from the specified file.
	Bitmap(Image, Size)	Initializes a new instance of the Bitmap class from the specified existing image, scaled to the specified size.
	Bitmap(Int32, Int32)	Initializes a new instance of the Bitmap class with the specified size.
	Bitmap(Stream, Boolean)	Initializes a new instance of the Bitmap class from the specified data stream.
	Bitmap(String, Boolean)	Initializes a new instance of the Bitmap class from the specified file.
	Bitmap(Type, String)	Initializes a new instance of the Bitmap class from a specified resource.
	Bitmap(Image, Int32, Int32)	Initializes a new instance of the Bitmap class from the specified existing image, scaled to the specified size.
	Bitmap(Int32, Int32, Graphics)	Initializes a new instance of the Bitmap class with the specified size and with the resolution of the specified Graphics object.
	Bitmap(Int32, Int32, PixelFormat)	Initializes a new instance of the Bitmap class with the specified size and format.
	Bitmap(Int32, Int32, Int32, PixelFormat, IntPtr)	Initializes a new instance of the Bitmap class with the specified size, pixel format, and pixel data.

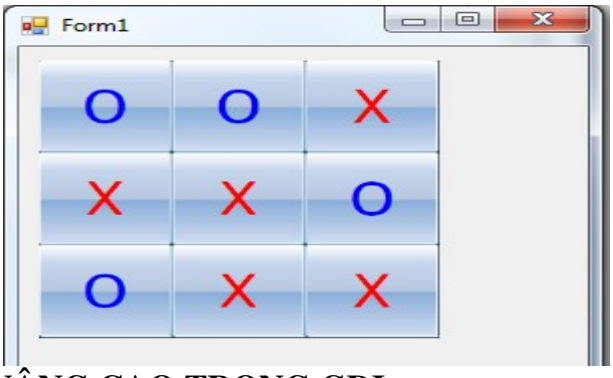
```
Bitmap bmp = new Bitmap(@"D:\MyPicture.bmp");
```

```
private void Form1_Paint(object sender, PaintEventArgs e)
{
    Graphics g = e.Graphics;
    Bitmap bmp = new Bitmap(@"D:\MyPicture.bmp");
    g.DrawImage(bmp, new PointF(0, 0));
}
```



- **Image:**

```
private void Form1_Paint(object sender, PaintEventArgs e)
{
    Graphics g = e.Graphics;
    Image img = Image.FromFile(@"D:\MyPicture.bmp");
    g.DrawImage(img, new Rectangle(new Point(10, 10), new Size(200, 200)));
    img.Save(@"D:\MyPicture1.bmp", ImageFormat.Bmp);
}
```



IV. CÁC CHỦ ĐỀ NÂNG CAO TRONG GDI+

- Chế độ tô vẽ và vẽ mượt mà (Antialiasing).
- Double buffering.

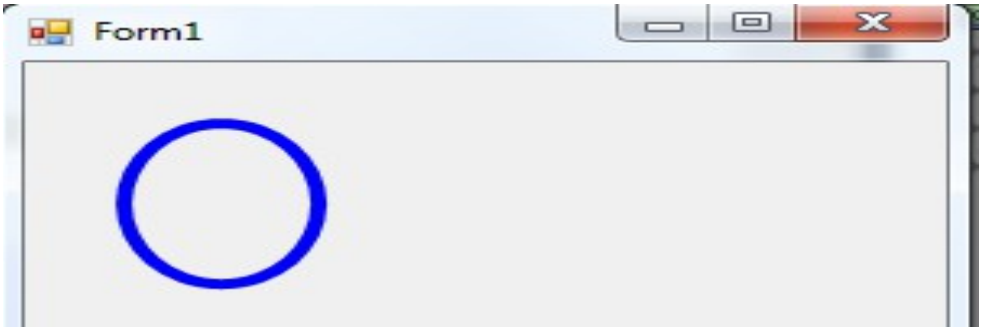
IV.1. Antialiasing

- Là 1 kỹ thuật dùng làm cho mượt mà những góc cạnh trên các hình dáng và văn bản. Nó hoạt động bằng cách thêm những bóng tại đường viền ở góc cạnh. Thí dụ, bóng màu xám (grey shading) có thể được thêm vào 1 đường cong màu đen làm cho góc cạnh mượt hơn. Về mặt kỹ thuật, antialiasing trộn lẫn (blend) một đường cong với nền của nó.

- **Ví dụ:**

```
private void Form1_Paint(object sender, PaintEventArgs e)
{
    Graphics g = e.Graphics;
    Color color = Color.Blue;
    Pen p = new Pen(color, (float) 5);
    g.DrawEllipse(p, new RectangleF(new PointF(30, 30), new SizeF(60, 80) ));
}
```

```
private void Form1_Paint(object sender, PaintEventArgs e)
{
    Graphics g = e.Graphics;
    g.SmoothingMode = SmoothingMode.AntiAlias;
    Color color = Color.Blue;
    Pen p = new Pen(color, (float) 5);
    g.DrawEllipse(p, new RectangleF(new PointF(30, 30), new SizeF(60, 80) ));
}
```



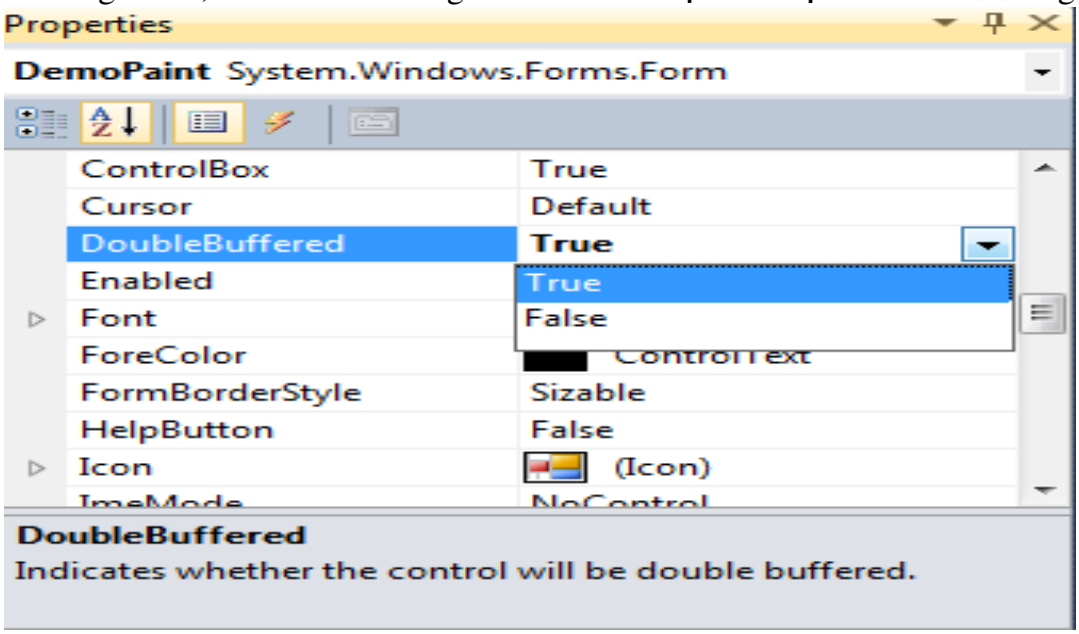
IV.2. Kỹ thuật Double Buffering

Trong một số trường hợp khi xử lý đồ họa, chúng ta có thể gặp trường hợp phải vẽ đi vẽ lại nhiều lần các bức hình với cường độ cao. Điều đó sẽ dẫn đến tình trạng "nháng hình".

Double Buffering sẽ làm giảm sự "nháng hình" đó.

Với "double buffering", logic vẽ sẽ ghi một hình bitmap trong bộ nhớ, và hình này được chép lên form vào cuối quá trình vẽ bằng một thao tác vẽ lại đơn lẻ trong suốt, nhờ đó mà hiện tượng rung hình giảm một cách đáng kể.

Trong .NET, có vài cách đơn giản để kích hoạt chế độ Double Buffering:



```
private void EnableDoubleBuffering()
{
    SetStyle(ControlStyles.UserPaint, true);
    SetStyle(ControlStyles.AllPaintingInWmPaint, true);
    SetStyle(ControlStyles.DoubleBuffer, true);
    UpdateStyles();
}
```

B. CÂU HỎI VÀ BÀI TẬP

Viết chương trình vẽ các hình cơ bản (hình tròn, hình chữ nhật, hình elip,…) lên form và thực hiên tô màu các hình vẽ đó với các màu tùy chọn.

MÃ BÀI HỌC MĐ23– 08	BÀI 8 : TRUY XUẤT DỮ LIỆU VỚI ADO.NET	Thời gian (giờ)				
		LT 5	TH 15	BT 5	KT 1	TS 26

Mục tiêu:

Sau khi học xong bài này, học viên có khả năng:

- Mô tả được cách thức lập trình với Database
- Sử dụng công cụ DataConnection để nối kết dữ liệu.
- Xây dựng được các phần mềm ứng dụng dựa trên hệ quản trị cơ sở dữ liệu có sẵn.
- Thực hiện các thao tác an toàn với máy tính;

Các nội dung chính sẽ được đề cập

- Giới thiệu lập trình cơ sở dữ liệu.
- Đối tượng SQLConnection.
- Đối tượng Oledbconnection.
- Đối tượng SQLCommand và Oledbcommand.
- Đối tượng SQLParameter và Parameters Collection.
- Đối tượng SQLDatareader

Trong chương này, chúng ta sẽ bàn về cách làm sao để một chương trình C# sử dụng ADO.NET. Kết thúc chương này, chúng ta sẽ có được các kiến thức sau:

Các kết nối cơ sở dữ liệu - làm sao để có thể sử dụng các lớp mới SqlConnection và OleDbConnection để kết nối và huỷ kết nối với cơ sở dữ liệu. Các kết nối dùng các kiểu giống như chuỗi kết nối của các trình cung cấp OLEDB. Sau đó chúng ta sẽ làm thử một vài kết nối cơ sở dữ liệu, và phải bảo đảm rằng kết nối sẽ được đóng lại sau khi dùng, thông qua một vài ứng dụng đơn giản.

Các lệnh thực thi - ADO.NET chứa một đối tượng command, thực thi SQL, hoặc có thể phát ra một stored procedure để trả về các giá trị. Các tùy chọn khác của đối tượng command sẽ được bàn kĩ, với các ví dụ cho từng tuỳ chọn được đưa ra trong các lớp Sql và OleDB.

Stored Procedures - Làm sao để gọi các stored procedure bằng các đối tượng command, và làm sao kết hợp các giá trị trả về với dữ liệu trên trình khách.

The ADO.NET object model - đây là một cách truyền đạt khác đến những đối tượng có sẵn với ADO, và các lớp DataSet, DataTable, DataRow, và DataColumn sẽ được bàn kĩ. Một DataSet có thể bao gồm các quan hệ giữa các table, cũng như các ràng buộc. Chúng sẽ được bàn kĩ.

Sử dụng XML và XML Schemas - ADO.NET được xây dựng trên một XML framework, vì thế chúng ta sẽ xem xét làm sao thêm vào các lớp dữ liệu một vài hỗ trợ của XML.

I. GIỚI THIỆU VỀ LẬP TRÌNH CƠ SỞ DỮ LIỆU

I.1 Giới thiệu ADO.NET

Giống như hầu hết các thành phần của .NET Framework, ADO.NET không chỉ là vỏ bọc của một vài API sẵn có. Nó chỉ giống ADO ở cái tên - các lớp và phương thức truy xuất dữ liệu đều khác hoàn toàn.

ADO (Microsoft's ActiveX Data Objects) là một thư viên của các thành phần COM đã từng được ca ngợi trong một vài năm trở lại đây. Phiên bản hiện tại là 2.7, các thành phần chủ yếu của ADO là Connection, Command, Recordset, và các Field object. Một connection có thể mở cơ sở dữ liệu, một vài dữ liệu được chọn vào một recordset, bao gồm các trường, dữ liệu này sau đó có thể thao tác, cập nhập lên server, và connection cần phải được đóng lại. ADO cũng giới thiệu một disconnected recordset, cái được dùng khi không muốn giữ kếp nối trong một thời gian dài.

Có một vài vấn đề với ADO đó là sự không hài lòng về địa chỉ, sự cồng kềnh của một disconnected recordset. Hỗ trợ này không cần thiết với sự tiến hoá của tin học "web-centric", vì vậy nó cần được loại bỏ. Có một số giống nhau giữa lập trình ADO.NET và ADO (không phải ở cái tên), vì thế việc chuyển từ ADO không qua khó khăn. Hơn thế nữa, nếu dùng SQL Server, có một bộ các quản mới rất tuyệt cho viêc thao tác bên ngoài cơ sở dữ liệu. Chừng đó lí do cũng đủ để các bạn quan tâm đến ADO.NET.

ADO.NET chứa hai không gian tên cơ sơ dữ liệu - một cho SQL Server, và một cái khác cho các cơ sở dữ liệu được trình bày thông qua một giao diện OLE DB. Nếu cơ sở dữ liệu của bạn chọn là một bộ phận của OLE DB, bạn có thể dễ dàng kết nối với nó từ .NET - chỉ cần dùng các lớp OLE DB và kết nối thông qua các driver cơ sở dữ liêu hiện hành.

I.1.1 Kiến trúc của ADO.NET

Có hai thành phần chính của ADO.NET là truy cập và thao tác dữ liệu :

* NET Framework data providers : được thiết kế đọc nhanh, chỉ đọc, một chiều các loại dữ liệu. Chúng là tập gồm các trình điều khiển dữ liệu của kiến trúc kết nối (Connnectivity Layer).

* Connection

II.1. Các Namespace

Tất cả các ví dụ trong chương này truy xuất dữ liệu trong một vài cách. Các không gian tên sau chỉ ra các lớp và các giao diện được dùng cho việc truy xuất dữ liệu trong .NET:

System.Data - Các lớp truy xuất dữ liệu chung
System.Data.Common - Các lớp dùng chung bởi các data provider khác nhau
System.Data.OleDb - Các lớp của OLE DB provider
System.Data.SqlClient - Các lớp của SQL Server provider
System.Data.SqlTypes - Cac kiểu của SQL Server

Các lớp chính trong ADO.NET được liệt kê dưới đây:

II.2. Các lớp dùng chung

ADO.NET chứa một số lớp được dùng không quan tâm là bạn đang dùng các lớp của SQL Server hay là các lớp của OLE DB.

Các lớp trong không gian tên System.Data được liệt kê sau đây:

DataSet - Đối tượng này chứa một bộ các DataTable, có thể bao gồm quan hệ giữa các bảng, và nó được thiết kế cho truy xuất dữ liệu không kết nối.

DataTable - Một kho chứa dữ liệu. Một DataTable bao gồm một hoặc nhiều DataColumns, và khi được tạo ra nó sẽ có một hoặc nhiều DataRows chứa dữ liệu.

DataRow - Một bộ giá trị, có bà con với một dòng trong bảng cơ sở dữ liệu, hoặc một dòng của bảng tính.

DataColumn - Chứa cá định nghĩa của một cột, chẳng hạn như tên và kiểu dữ liệu.

DataRelation - Một liên kết giữa hai DataTable trong một DataSet. Sử dụng cho khóa ngoại và các mối quan hệ chủ tớ.

Constraint - Định nghĩa một qui tắt cho một DataColumn (hoặc môt bộ các cột dữ liệu), như các giá trị là độc nhất.

Sau đây là hai lớp được tìm thấy trong không gian tên System.Data.Common:

DataColumnMapping - Ánh xạ tên của một cột từ cơ sở dữ liệu vào tên của một cột trong một DataTable.

DataTableMapping - Ánh xạ tên của một bảng từ cơ sở dữ liệu vào một bảng trong một DataSet.

II.3. Các lớp cơ sở dữ liệu chuyên biệt

Bổ sung cho các lớp dùng chung ở trên, ADO.NET có một số các lớp dữ liệu chuyên biệt được đưa ra dưới đây. Các lớp này thực thi một bộ các giao diện chuẩn được định nghĩa trong không gian tên System.Data, cho phép sử dụng các lớp có cùng kiểu giao diện. Ví dụ cả hai lớp SqlConnection và OleDbConnection thực thi giao diện IDbConnection.

SqlCommand, OleDbCommand - Một vỏ bọc của các câu lệnh SQL hoặc các lời gọi stored procedure.

SqlCommandBuilder, OleDbCommandBuilder - Một lớp sử dụng các câu lệnh SQL (chẳng hạn như các câu lệnh INSERT, UPDATE, vàDELETE) từ một câu lệnh SELECT.

SqlConnection, OleDbConnection - Kết nối với cơ sở dữ liệu. Giống như một ADO Connection.

SqlDataAdapter, OleDbDataAdapter - Một lớp giữ các câu lệnh select, insert, update, và delete, chúng được sử dụng để tạo một DataSet và cập nhật Database.

SqlDataReader, OleDbDataReader - Chỉ đọc, kết nối với data reader.

SqlParameter, OleDbParameter - Định nghĩa một tham số cho một stored procedure.

SqlTransaction, OleDbTransaction - Một giao tiếp cơ sở dữ liện, được bọc trong một đối tượng.

Một đặc tính quan trọng của các lớp ADO.NET là chúng được thiết kế để làm việc trong môi trường không kết nối, đóng một vai trò quan trọng trong thế giới

"web-centric". Nó hiện được dùng để kiến trúc một server (chẳng hạn như mua sách qua mạng) để kết nối một server, lấy một vài dữ liệu, và làm việc trên những dữ liệu này trên PC khách trước khi kết nối lại và truyền dữ liệu trở lại để xử lí.

ADO 2.1 giới thiệu recordset không kết nối, nó cho phép dữ liệu có thể được lấy từ một cơ sở dữ liệu, được truyền cho trình khách để xử lí. Nó thường khó xử dụng do cách ứng xử không kết không được thiết kế từ đâu. Các lớp ADO.NET thì khác - Sql/OleDb DataReader được thiết kết cho để dùng cho các cơ sở dữ liệu offline.

Sử dụng các lớp và các giao diện để truy cập cơ sở dữ liệu trong được trình bày trong chương sau. Tôi chủ yếu tập trung vào các lớp Sql khi kết nối cơ sở dữ liệu, bởi vì các ví dụ được cài đặt một cơ sở dữ liệu MSDE (SQL Server). Trong hầu hết các trường hợp các lớp OleDb ứng sử tương tự như mã Sql.

I.2 Không gian tên Microsoft.SqlServer

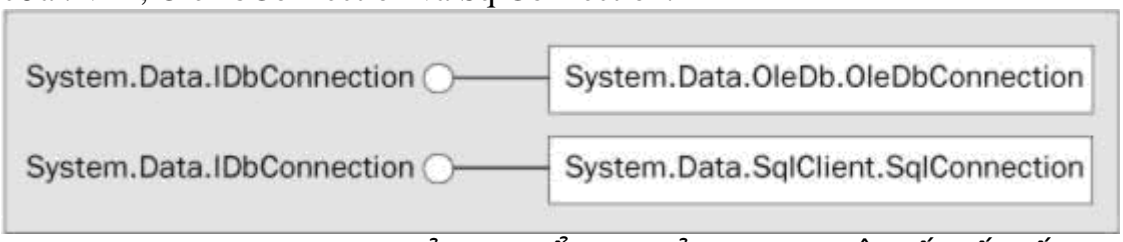
I.3 Làm việc với Microsoft.SqlServer.Management

I.4 Thư viện SqlServer.Management

I.5 Thao tác dữ liệu

III. SỬ DỤNG CÁC DATABASE CONNECTION

Trong trìn tự truy xuất cơ sở dữ liệu, bạn cần cung cấp các thông số kết nối, chẳng hạn như thiết bị mà cơ sở dữ liệu đang chạy, và khả năng đăng nhập của bạn. Bất kì ai đã từng làm việc với ADO sẽ dễ dàng quen với các lớp kết nối của .NET, OleDbConnection và SqlConnection:



Đoạn mã sau đây mô tả cách để tạo, mở và đóng một kết nối đến cơ sở dữ liệu Northwind. Các ví dụ trong chương này chúng ta dùng cơ sở dữ liệu Northwind, được cài đặt chung với các ví dụ của .NET Framework SDK:

```
using System.Data.SqlClient;
string source = "server=(local)\\NetSDK;" +
            "uid=QSUser;pwd=QSPassword;" +
            "database=Northwind";
SqlConnection conn = new SqlConnection(source);
conn.Open();
// Do something useful
conn.Close();
```

Chuỗi kết nối sẽ trở nên thân thiện nếu bạn đã từng dùng ADO hay OLE DB trước đây - thật vậy, bạn có thể cắt và dán từ mã cũ của bạn, nếu bạn dùng OleDb provider. Trong ví dụ chuỗi kết nối này, các tham số được dùng như sau (các tham số cách nhau bởi dấu chấm phẩy trong chuỗi kết nối).

server=(local)\\NetSDK - Nó biểu diễn database server được kết nối. SQL Server cho phép một số các tiến trình database server processes khác nhau chạy trên cùng một máy, vì vậy ở đây chúng ta thực hiện kết nối với tiến trình NetSDK trên máy cụ bộ.

uid=QSUser - Tham số này mô tả người dùng cơ sở dữ liệu. Bạn cũng có thể sử dụng User ID.

pwd=QSPassword - và đây là password cho người dùng đó. .NET SDK là một bộ các cơ sở dữ liệu giống nhau, và user/password này được liên kết và được thêm vào trong quá trình cài đặt các ví dụ .NET. Bạn cũng có thể dùng Password.

database=Northwind - Cái này mô tả loại dữ liệu để kết nối - mỗi tiến trình SQL Server có thể đưa ra một vài loại dữ liệu khác nhau.

Ví dụ trên mở một kết nối cơ sở dữ liệu ùng chuỗi kết nối đã được định nghĩa, sau đó đóng kết nối lại. Khi kết nối đã được mở, bạn có thể phát các lệnh để thao tác trên cơ sở dữ liệu, và khi hoàn tất, kết nối có thể được đóng lại.

SQL Server có một chế độ bảo mật khác - nó có thể dùng chế độ bảo mật của Windows, vì thế các khả năng truy cập của Windows có thể truyền cho SQL Server. Với lựa chọn này bạn có thể bỏ đi các vị trí uid và pwd trong chuỗi kết nối, và thêm vào Integrated Security=SSPI.

Trong lúc download mã nguồn sẵn có cho chương này, bạn cần tìm file Login.cs nó đơn giãn hóa các ví dụ trong chương này. Nó được kết nối với tất cả các mã ví dụ, và bao gồm thông tin kết nối cơ sở dữ liệu dùng cho các ví dụ; sau đó bạn có thể cung cấp tên server, user, and password một cách thích hợp. Nếu mặc định dùng Windows integrated security; bạn cần thay đổi username và password cho phù hợp.

Bây giờ chúng ta đã biết cách mở các kết nối, trước khi chuyển qua vấn đề khác chúng ta cần xem xét một vài thực hành tốt có liên quan đến các kết nối.

III.1. Sử dụng hiệu quả các Connection

Một cách tổng quát, khi sử dụng các tài nguyên "hiếm" trong .NET, chẳng hạn như các kết nối cơ sở dữ liệu, các cửa sổ,hoặc các đối tượng đồ họa, tốt hơn hết bạn nên đảm bảo rằng các tài nguyên này luôn phải được đóng lại sau khi đã sử dụng xong. Dù vậy các nhà thiết kết của .NET có thể làm điều này nhờ trình thu gom rác, nó luôn làm sau bộ nhớ sau một khoảng thời gian nào đó, tuy nhiên nó nên được giải phóng càng sớm càng tốt.

Rõ ràng là khi viết mã truy xuất một cơ sở dữ liệu, việc giữ một kết nối càng ít thời gian càng tốt để không làm ảnh hưởng đến các phần khác. Trong nhiều tình huống tiêu cực, nếu không đóng một kết nối có thể khoá không cho các người dùng khác truy nhập vào các bảng dữ liệu đó, một tác hại to lớn đối với khả năng thực thi của ứng dụng. Việc đóng một kết nối cơ sở dữ liệu có thể coi là bắt buộc, vì thế ứng dụng này chỉ ra cách cấu trúc mã của bạn để giảm thiểu các rủi ro cho một mã nguồn mở.

Có hai cách để đảm bảo rằng các kết nối cơ sở dữ liệu được giải phóng sau khi dùng.

III.2. Tùy chọn một - try/catch/finally

Tùy chọn thứ nhất để đảm bảo rằng các tài nguyên được dọn sạch là sử dụng các khối lệnh try…catch…finally, và đảm bảo rằng bạn đã đóng các kết nối trong khối lệnh finally. Đây là một ví dụ nhỏ:

```
try
{
  // Open the connection
  conn.Open();
  // Do something useful
}
catch ( Exception ex )
{
  // Do something about the exception
}
finally
{
  // Ensure that the connection is freed
  conn.Close ( ) ;
}
```

Với khối kết nối bạn có thể giải phóng bất kì tài nguyên nào mà bạn đã dùng. Vấn đề duy nhất trong phương thức này là bạn phải bảo đảm rằng bạn có đóng các kết nối - rất là dễ quên việc thêm vào khối finally, vì vậy một phong cách lập trình tốt rất quan trọng.

Ngoài ra, bạn có thể mở một số tài nguyên (chẳng hạn hai kết nối cơ sở dữ liệu và một file) trong một phương thức, vì vậy đôi khi các khối try…catch…finally trở nên khó đọc. Có một cách khác để đảm bảo rằng các tài nguyên được dọn dẹp - sử dụng câu lệnh.

III.3. Tùy chọn hai - Sử dụng khối câu lệnh

Trong lúc phát triển C#, phương thức .NET's dọn dẹp các đối tượng khi chúng không còn được tham chiếu nữa sử dụng các huỷ bất định trở thành một vấn đề nóng hổi. Trong C++, ngay khi một đối tượng rời khỏi tầm vực, khối huỷ tử của nó sẽ tự động được gọi. Nó là một điều rất mới cho các nhà thiết cớ các lớp sử dụng tài nguyên, khi một huỷ tử được sử dụng để đóng các tài nguyên nếu các người dùng quên làm điều đó. Một huỷe tử C++ được gọi bất kì khi nào một đối tượng vượt quá tầm vực của nó - vì vậy khi một ngoại lệ được phát ra mà không được chặn, tât cả các hủy tử cần phải được gọi.

Với C# và các ngôn ngữ có quản khác, tất cả đều tự động, các khối huỷ tử định trước được thay thế bởi trình thu gom rác, cái được dùng để tháo các tài nguyên tại một thời điểm trong tương lai. Chúng mang tính bất định, nghĩa là bạn sẽ không biết trước được khi nào thì việc đó sẽ xảy ra. Nếu quên không đóng một kết nối cơ sở dữ liệu có thể là nguyên nhân gây ra lỗi khi chạy trong .NET. Mã sau đây sẽ giải thích cách để sử dụng giao diện IDisposable để giải phóng tài nguyên khi thoát khỏi khối using .

```
string source = "server=(local)\\NetSDK;" +
                "uid=QSUser;pwd=QSPassword;" +
```

```
                "database=Northwind";
using ( SqlConnection conn = new SqlConnection ( source ) )
{
  // Open the connection
  conn.Open ( ) ;

  // Do something useful
}
```

Mệnh đề using đã được giới thiệu trong chương 2. Đối tượng trong mệnh đề using phải thực thi giao diện IDisposable, nếu không một se tạo ra một lỗi biên dịch. Phương thức Dispose() sẽ tự động được gọi trong khi thoát khỏi khối using.

Khi xem mã IL của phương thức Dispose() của SqlConnection (và OleDbConnection), cả hai đều kiểm tra trạng thái của đối tượng kết nối, và nếu nó đang mở phương thức Close() sẽ được gọi.

Khi lập trình bạn nên dùng cả hai tùy chọn trên.Ở nhưng chỗ bạn cần các tài nguyên tốt nhất là sử dụng mệnh đề using(), dù vậy bạn cũng có thể sử dụng câu lệnh Close(), nếu quên không sử dụng thì khối lệnh using sẽ đóng lại giúp bạn. Không gì có thể thay thế được mọt bẫy ngoại lệ tốt, vì thế tốt nhất bạn dùng trộn lẫn hai phương thức như ví dụ sau:

```
try
{
  using (SqlConnection conn = new SqlConnection ( source ))
  {
    // Open the connection
    conn.Open ( ) ;
    // Do something useful
    // Close it myself
    conn.Close ( ) ;
  }
}
catch (Exception e)
{
  // Do something with the exception here...
}
```

Ở đây đã gọi tường minh phương thức Close() mặc dù điều đó là không bắt buộc vì khối lệnh using đã làm điều đó thay cho bạn; tuy nhiên, bạn luôn chắc rằng bất kì tài nguyên nào cũng được giải phóng sớm nhất có thể - bạn có thể có nhiều mã trong khối lệnh mã không khoá tài nguyên.

Thêm vào đó, nếu một ngoại lệ xảy ra bên trong khối using, thì phương thức IDisposable.Dispose sẽ được gọi để bảo đảm rằng tài nguyên được giải phóng, điều này đảm bảo rằng kết nối cơ sở dữ liệu luôn luôn được đóng lại. Điều này làm cho mã dễ đọc và luôn đảm bảo rằng kết nối luôn được đóng khi một ngoại lệ xảy ra.

Cuối cùng, nếu bạn viết các lớp bao bọc một tài nguyên có lẽ luôn thực hiện giao diện IDisposable để đóng tài nguyên. Bằng cách dùng câu lệnh using() nó luôn đảm bảo rằng tài nguyên đó sẽ được dọn dẹp.

III.4. Các Transaction (giao dịch)

Thường khi có nhiều hơn một cập nhật dữ cơ sở dữ liệu thì các thực thi này được thực hiện bên trong tầm vực của một transaction. Một transaction trong ADO.NET được khởi tạo bằng một lời gọi đến các phương thức BeginTransaction() trên đối tượng kết nối cơ sở dữ liệu. Những phương thức này trả về một đối tượng có thể thực thi giao diện IDbTransaction, được định nghĩa trong System.Data.

Chuỗi mã lệnh dưới đây khởi tạo một transaction trên một kết nối SQL Server:

```
string source = "server=(local)\\NetSDK;" +
          "uid=QSUser;pwd=QSPassword;" +
          "database=Northwind";
SqlConnection conn = new SqlConnection(source);
conn.Open();
SqlTransaction tx = conn.BeginTransaction();
// Execute some commands, then commit the transaction
tx.Commit();
conn.Close();
```

Khi bạn khởi tạo một transaction, bạn có thể chọn bậc tự do cho các lệnh thực thi trong transaction đó. Bậc này chỉ rõ sự tự do của transaction này với các transaction khác xảy ra trên cơ sở dữ liệu. Các hệ cơ sở dữ liệu có thể hỗ trợ bốn tùy chọn sau đây:

Isolation Level	**Description**
ReadCommitted	Mặc định cho. Bậc này đảm bảo rằng dữ liệu đã ghi bởi transaction sẽ chỉ có thể truy cập được bởi một transaction khác sau khi nó hoàn tất công việc của mình.
ReadUncommitted	Tùy chọn này cho phép transaction của bạn có thể đọc dữ liệu trong cơ sở dữ liệu, dù cho dữ liệu đó đang được một transaction khác sử dụng. Ví dụ như, nếu hai người dùng truy cập cùng lúc vào một cơ sở dữ liệu, và người thứ nhất chiên một vài dữ liệu trong transaction của họ (đó là một Commit hoặc Rollback), và người thứ hai với tuỳ chọn bậc tự do là ReadUncommitted có thể đọc dữ liệu.
RepeatableRead	Bậc này là một mở rộng của ReadCommitted, nó bảo đảm rằng nếu một lệnh tương tự được phát ra trong transaction, ensures that if the same statement is issued within the transaction, regardless of other potential updates made to the database, the same data will always be returned. This level does require extra locks to be held on the data, which could adversely affect performance. This level guarantees that, for each row in the initial query, no changes can be made to that data. It does however permit "phantom" rows to show up - these are completely new rows that another transaction may have inserted while your transaction is running.

Serializable	This is the most "exclusive" transaction level, which in effect serializes access to data within the database. With this isolation level, phantom rows can never show up, so a SQL statement issued within a serializable transaction will always retrieve the same data. The negative performance impact of a Serializable transaction should not be underestimated - if you don't absolutely need to use this level of isolation, it is advisable to stay away from it.

Bậc tự do mặc định của The SQL Server, ReadCommitted, là một kết hợp tốt giữa tính chặc chẽ và sẵn dùng của dữ liệu. Tuy nhiêu, có nhiều trường hợp bậc tự do có thể tăng lên, trong .NET bạn đơn giản khởi tạo một transaction khác với bậc khác bậc mặc định. Điều đó còn dựa vào kinh nghiệm của mỗi người.

Một điếu cuối cùng về transactions - nếu bạn đang dùng một cơ sở dữ liệu không hỗ trợ các transaction, thì tốt nhất bạn nên đổi sang một cơ sở dữ liệu có dùng chúng!

IV.COMMANDS

Chúng ta lại nói lại về commands. Một command là một một kiểu đơn giản, một chuỗi lệnh SQL được dùng để truy xuất dữ liệu. Một command có thể là một stored procedure, hoặc là tên của một bảng sẽ trả về:

```
string source = "server=(local)\\NetSDK;" +
        "uid=QSUser;pwd=QSPassword;" +  "database=Northwind";
string select = "SELECT ContactName,CompanyName FROM Customers";
SqlConnection conn = new SqlConnection(source);
conn.Open();
SqlCommand cmd = new SqlCommand(select, conn);
```

Các mệnh đề SqlCommand và OleDbCommand thường được gọi là CommandType, chúng được dùng để định nghĩa các mệnh đề SQL, một stored procedure, hoặc một câu lệnh SQL. Sau đây là một bảng liệt kê đơn giản về CommandType:

CommandType	**Example**
Text (default)	String select = "SELECT ContactName FROM Customers"; SqlCommand cmd = new SqlCommand(select , conn);

StoredProcedure	SqlCommand cmd = new SqlCommand("CustOrderHist", conn);
	cmd.CommandType = CommandType.StoredProcedure;
	cmd.Parameters.Add("@CustomerID", "QUICK");

TableDirect	OleDbCommand cmd = new OleDbCommand("Categories", conn); cmd.CommandType = CommandType.TableDirect;

Khi thực thi một stored procedure, cần truyền các tham số cho procedure. Ví dụ trên cài đặt trực tiếp tham số @CustomerID, dù vậy có nhiều cách để cài giá trị tham số, chúng ta sẽbàn kĩ trong phần sau của chương này.

chú ý Kiểu TableDirect command không chỉ đúng cho OleDb provider – có một ngoại lệ xảy ra khi bạn cố dùng command này trong Sql provider.

IV.1. Executing Commands

Bạn đã định nghĩa các command, và bạn muốn thực thi chúng. Có một số cách để phát ra các statement, dựa vào kết quả mà bạn muốn command đó muốn trả về. Các mệnh đề SqlCommand và OleDbCommand cung cấp các phương thức thực thi sau:

ExecuteNonQuery() – Thực thi các command không trả về kết quả gì cả

ExecuteReader() – Thực thi các command và trả về kiểu IDataReader

ExecuteScalar() – Thực thi các command và trả về một giá trị đơn

Lớp SqlCommand cung cấp thêm một số phương thức sau

ExecuteXmlReader() – Thực thi các command trả về một đối tượng XmlReader, các đối tượng được dùng để xem xét các XML được trả về từ cơ sở dữ liệu.

IV.1.1. ExecuteNonQuery()

Phương thức này thường được dùng cho các câu lệnh UPDATE, INSERT, hoặc DELETE, để trả về số các mẫu tin bị tác động. Phương thức này có thể trả về các kết quả thông qua các tham số được truyền vào stored procedure.

```
using System;
using System.Data.SqlClient;
public class ExecuteNonQueryExample
{
  public static void Main(string[] args)
  {
                        string source = "server=(local)\\NetSDK;" +
"uid=QSUser;pwd=QSPassword;" +
                "database=Northwind";
    string select = "UPDATE Customers " +  "SET ContactName = 'Bob' " +
                "WHERE ContactName = 'Bill'";
    SqlConnection  conn = new SqlConnection(source);
    conn.Open();
    SqlCommand cmd = new SqlCommand(select, conn);
    int rowsReturned = cmd.ExecuteNonQuery();
    Console.WriteLine("{0} rows returned.", rowsReturned);
    conn.Close();
  }
}
```

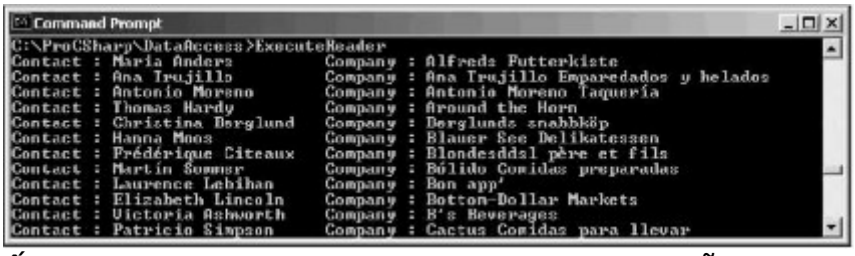
ExecuteNonQuery() trả về một số kiểu int cho biết số dòng bị tác động command.

IV.1.2. ExecuteReader()

Phương thức này thực hiện các lệnh trả về một đối tượng SqlDataReader hoặc OleDbDataReader. Đối tượng này có thể dùng để tạo ra các mẫu tin như mã sau đây:

```
using System;
using System.Data.SqlClient;
public class ExecuteReaderExample
{
  public static void Main(string[] args)
```

```
  {
    string source = "server=(local)\\NetSDK;" +
                    "uid=QSUser;pwd=QSPassword;" +
                    "database=Northwind";
    string select = "SELECT ContactName,CompanyName FROM Customers";
    SqlConnection conn = new SqlConnection(source);
    conn.Open();
    SqlCommand cmd = new SqlCommand(select, conn);
    SqlDataReader reader = cmd.ExecuteReader();
    while(reader.Read())
    {
      Console.WriteLine("Contact : {0,-20} Company : {1}" ,
                        reader[0] , reader[1]);
    }
  }
}
```



Các đối tượng SqlDataReader và OleDbDataReader sẽ được trình bày trong chương sau.

IV.1.3 ExecuteScalar()

Trong nhiều trường hợp một câu lệnh SQL cần phải trả về một kết quả đơn, chẳng hạn như số các record của một bảng, hoặc ngày giờ hiện tại của server. Phương thức ExecuteScalar có thể dùng cho những trường hợp này:

```
using System;
using System.Data.SqlClient;
public class ExecuteScalarExample
{
  public static void Main(string[] args)
  {
    string source = "server=(local)\\NetSDK;" +
                    "uid=QSUser;pwd=QSPassword;" +
                    "database=Northwind";
    string select = "SELECT COUNT(*) FROM Customers";
    SqlConnection conn = new SqlConnection(source);
    conn.Open();
    SqlCommand cmd = new SqlCommand(select, conn);
    object o = cmd.ExecuteScalar();
    Console.WriteLine ( o ) ;
  }
}
```

Phương thức trả về một đối tượng, Bạn có thể chuyển sang kiểu thích hợp.

IV.1.3. ExecuteXmlReader() (SqlClient Provider Only)

Giống như tên đã gọi, nó có thể thực thí command và trả về một đôi tượng XmlReader. SQL Server cho phép câu lệnh SQL SELECT dùng cho kiểu FOR XML. Mệnh đề này có thể có ba kiểu tùy chọn sau:

FOR XML AUTO – tạo một cây cơ sở cho các bảng trong mệnh đề FROM

FOR XML RAW – trả về một bộ các mẫu tin ánh xạ định các nhân tố, với các cột được ánh xạ đến các thuộc tính

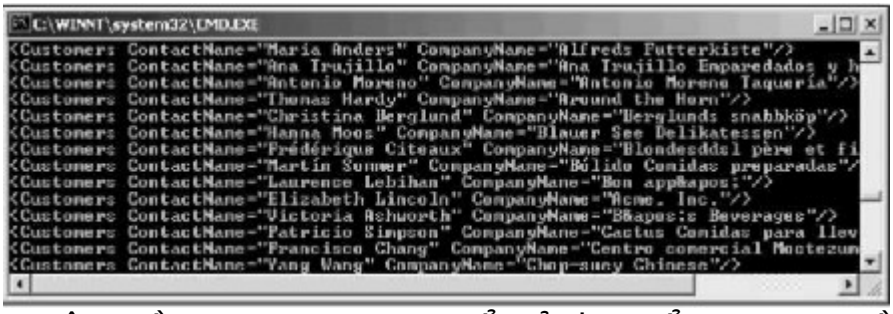
FOR XML EXPLICIT – bạn cần phải chỉ định hình dạng của cây XML trả về

Professional SQL Server 2000 XML (Wrox Press, ISBN 1-861005-46-6) diễn tả đầy đủ các thuộc tính này:

```
using System;
    using System.Data.SqlClient;
    using System.Xml;
    public class ExecuteXmlReaderExample
    {
      public static void Main(string[] args)
      {
        string source = "server=(local)\\NetSDK;" +
                    "uid=QSUser;pwd=QSPassword;" +
                    "database=Northwind";
        string select = "SELECT ContactName,CompanyName " +
                    "FROM Customers FOR XML AUTO";
        SqlConnection conn = new SqlConnection(source);
        conn.Open();
        SqlCommand cmd = new SqlCommand(select, conn);
        XmlReader xr = cmd.ExecuteXmlReader();
        xr.Read();
        string s;
        do
        {
          s = xr.ReadOuterXml();
          if (s!="")
            Console.WriteLine(s);
        } while (s!= "");
        conn.Close();
      }
    }
```

Chú ý rằng chúng ta có thể nhập không gian tên System.Xml namespace cho các kiểu trả về XML. Không gian này được dùng cho những khả năng của XML trong .NET Framework trong tương lại được trình bày kĩ trong chương 11.

Ở đây chúng bao gồm các mệnh đề FOR XML AUTO trong mệnh đề SQL, sau đó gọi phương thức ExecuteXmlReader(). Sau đây là kết quả của các mã lệnh trên:



Trong mệnh đề SQL, chúng ta có thể chỉ định, để các thành phần của kiểu Customers được hiển thị trong phần kết xuất. Để làm điều đó ta phải thêm các thuộc tính cho mỗi một cột trong cơ sở dữ liệu. Điều này sẽ tạo ra một sự phân mảnh trong việc chọn các mẫu tin từ cơ sở dữ liệu.

IV.2.Gọi các Stored Procedure

Việc gọi một stored procedure với một đối tượng command đơn giản là định nghĩa tên của stored procedure cần dùng, thêm vào các tham số của procedure đó, thực thi command với một trong các phương thức đã giới thiệu ở phần trên.

Để dễ dàng cho việc lấy ví dụ trong phần này, Tôi đã định nghĩa một bộ các stored procedures dùng để chèn, cập nhât, và xoá các mẫu tin từ bảng Region trong cơ sở dữ liệu Northwind. Tôi đã chọn bảng này vì nó đủ nhỏ để có thể áp dụng các ví vị cho mỗi kiểu của storeprocedure.

IV.2.1. Gọi một Stored Procedure không trả lại kết quả

Ví dụ này sẽ mô tả cách gọi một stored procedure không trả lại kết quả. Có hai procedure được định nghĩa dưới đây, một dùng cho việc cập nhật các mẫu Region sẵn có, và một dùng để xóa các mẫu tin trong Region.

Record Update

Cập nhật một mẫu Region là một công việc khá đơn giản, chỉ thay đổi một trường duy nhất (vì không thể cập nhật khóa chính). Bạn có thể gõ trực tiếp các ví dụ này trong SQL Server Query Analyzer, để cài đặt các stored procedure dùng cho phần này:

```
CREATE PROCEDURE RegionUpdate (@RegionID INTEGER,
                    @RegionDescription NCHAR(50)) AS
  SET NOCOUNT OFF
  UPDATE Region
    SET RegionDescription = @RegionDescription
    WHERE RegionID = @RegionID
GO
```

Một lệnh cập nhật trong một bảng thực tế hơn cần phải chon lại và trả về trạng thái của các mẫu được cập nhật. Stored procedure này cần nhập vào hai tham số (@RegionID và @RegionDescription, và phát ra một câu lệnh UPDATE để thao tác trên cơ sở dữ liệu.

Để chạy stored procedure này trong mã .NET, bạn cần phải định nghĩa một lệnh SQL và thực thi nó:

```
SqlCommand aCommand = new SqlCommand("RegionUpdate", conn);
aCommand.CommandType = CommandType.StoredProcedure;
aCommand.Parameters.Add(new SqlParameter ("@RegionID",
                        SqlDbType.Int, 0, "RegionID"));
```

```
        aCommand.Parameters.Add(new SqlParameter("@RegionDescription",
                                    SqlDbType.NChar, 50, "RegionDescription"));
        aCommand.UpdatedRowSource = UpdateRowSource.None;
```

Đoạn mã này tạo một đối tượng SqlCommand mới tên là aCommand, và định nghĩa nó là một stored procedure. Sau đó chúng ta thêm vào các tham số nhập, cũng như các tham số chứa giá trị mong muốn trả về từ stored procedure để biết được các giá trị trong các dòng UpdateRowSource được liệt kê, chúng ta sẽ bàn kĩ vấn đề ở các phần sau của chương này.

Một command được tạo ra, có thể được thực thi bởi việc phát ra các lệnh sau:

```
        aCommand.Parameters[0].Value = 999;
        aCommand.Parameters[1].Value = "South Western England";
        aCommand.ExecuteNonQuery();
```

Ở đây chúng ta đang cài đặt giá trị cho các tham số, sau đó thực thi stored procedure.

Các Command parameters có thể được cài đặt bằng chỉ số như đã trình bày ở trên, hoặc dùng tên.

Record Deletion

Stored procedure tiếp theo dùng để xóa một mẫu tin trong bảng Region:

```
CREATE PROCEDURE RegionDelete (@RegionID INTEGER) AS
  SET NOCOUNT OFF
  DELETE FROM Region
  WHERE      RegionID = @RegionID
GO
```

Procedure này chỉ yêu cầu khóa chính của mẫu tin. Mã sử dụng một đối tượng SqlCommand để gọi stored procedure này như sau:

```
SqlCommand aCommand = new SqlCommand("RegionDelete" , conn);
aCommand.CommandType = CommandType.StoredProcedure;
aCommand.Parameters.Add(new SqlParameter("@RegionID" , SqlDbType.Int , 0 ,
                          "RegionID"));
aCommand.UpdatedRowSource = UpdateRowSource.None;
```

Lệnh này chỉ chấp nhận một tham số đơn để thực thi RegionDelete stored procedure; đây là ví dụ cho việc cài đặt tham số theo tên:

```
aCommand.Parameters["@RegionID"].Value= 999;
aCommand.ExecuteNonQuery();
```

IV.2.2. Gọi một Stored Procedure trả về các tham số trả về

Cả hai ví dụ về stored procedures ở trên đều không có giá trị trả về. Nếu một stored procedure bao gồm các tham số trả về, sau đó những phương thức này cần được định nghĩa trong .NET client rằng chúng có thể lấy giá trị trả về từ procedure.

Ví dụ sau chỉ ra cách chền một mẫu tin vào cơ sở dữ liệu, và trả về khoá chính của mẫu tin đó.

Record Insertion

Bảng Region chỉ chứa khóa chính (RegionID) và một trường diễn giải (RegionDescription). Để chèn một mẫu tin, cần phải cung cấp khóa chính, và sau đó một mẫu tinh mới sẽ được chèn vào cơ sở dữ liệu. Tôi đã chọn cách tạo khóa chính đơn giản nhất trong ví dụ này bằng cách tạo ra một số mới trong stored procedure.

Phương thức được dùng hết sức thô sơ, tôi sẽ bàn kĩ về cách tạo khóa chính ở phần sau của chương. Và đây là ví dụ thô sơ của chúng ta:

```
CREATE PROCEDURE RegionInsert(@RegionDescription NCHAR(50),
                    @RegionID INTEGER OUTPUT)AS
  SET NOCOUNT OFF
  SELECT @RegionID = MAX(RegionID)+ 1
  FROM Region
  INSERT INTO Region(RegionID, RegionDescription)
  VALUES(@RegionID, @RegionDescription)
GO
```

Insert procedure này tạo ra một mẫu tin Region mới. Khóa chính được phát ra bởi chính cơ sở dữ liệu, giá trị này được tra về như một tham số của procedure (@RegionID). Đây là một ví dụ đơn giản, nhưng đối với các bảng phức tạp hơn, nó thường không sử dụng các tham số trả về mà thay vào đó nó chọn các dòng được cập nhật và trả nó về cho trình gọi.

```
SqlCommand  aCommand = new SqlCommand("RegionInsert" , conn);
aCommand.CommandType = CommandType.StoredProcedure;
aCommand.Parameters.Add(new SqlParameter("@RegionDescription" ,
                        SqlDbType.NChar , 50 , "RegionDescription"));
aCommand.Parameters.Add(new SqlParameter("@RegionID" ,  SqlDbType.Int,  0 ,
                        ParameterDirection.Output , false , 0 ,  0 ,  "RegionID" ,
                        DataRowVersion.Default ,  null));
aCommand.UpdatedRowSource = UpdateRowSource.OutputParameters;
```

Đây là phần định nghĩa phức tạp hơn cho các tham số. Tham số thứ hai, @RegionID, được định nghĩa để bao gồm các tham số trực tiếp của nó, trong ví dụ này nó là Output. Chúng ta sử dụng tập hợp UpdateRowSource để thêm cờ OutputParameters trên dòng cuối của mã, cờ này cho phép chúng ta trả dữ liệu từ stored procedure này vào các tham số. Cờ này được dùng chủ yếu cho việc gọi các stored procedure từ một DataTable (được giải thích trong chương sau).

Việc gọi stored procedure này giống như các ví dụ trước, ngoại trừ ở đây chúng ta cần đọc tham số xuất sau khi thực thi procedure:

```
aCommand.Parameters["@RegionDescription"].Value = "South West";
aCommand.ExecuteNonQuery();
int newRegionID = (int) aCommand.Parameters["@RegionID"].Value;
```

Sau khi thực thi lệnh, chúng ta đọc giá trị tham số @RegionID và ép nó vào một integer.

Có thể bạn sẽ hỏi phải làm gì nếu stored procedure mà bạn gọi trả về các tham số xuất và một tập các dòng. Trong trường hợp này, định nghĩa các tham số tương ứng, sau đó gọi phương thức ExecuteNonQuery(), cũng có thể gọi một trong những phương thức khác (chẳng hạn như ExecuteReader()) nó cho phép bạn lấy các mẫu tin trả về.

V. TRUY CẬP NHANH CƠ SỞ DỮ LIỆU VỚI DATA READER

Một data reader là cách đơn giản nhất và nhanh nhất để chọn một vài dữ liệu từ một nguồn cơ sơ dữ liệu, nhưng cũng ít tính năng nhất. Bạn có thể truy xuất trực tiếp một đối tượng data reader – Một minh dụ được trả về từ một đối tượng SqlCommand hoặc OleDbCommand từ việc gọi một phương thức ExecuteReader()

– có thể là một đối tượng SqlCommand, một đối tượng SqlDataReader, từ một đối tượng OleDbCommand là một OleDbDataReader.

Mã lệnh sau đây sẽ chứng minh cách chọn dữ liệu từ bản Customers của cơ sở dữ liệu Northwind. Ví dụ kết nối với cơ sở dữ liệu chọn một số các mẫu tin, duyệt qua các mẫu tin được chọn và xuất chúng ra màn hình console.

Ví dụ này có thể dùng cho OLE DB provider. Trong hầu hết các trường hợp các phương thức của SqlClient đều được ánh xạ một một vào các phương thức của đối OleDBClient.

Để thực thi lại các lệnh đối với một OLE DB data source, lớp OleDbCommand được sử dụng. Mã lệnh dưới đây là một ví dụ một câu lệnh SQL đơn giảnvà đọc các mẫu tin được trả về bởi đối tượng OleDbDataReader.

Mã của ví dụ có thể được tìm thấy trong thư mục Chapter 09\03_DataReader.

Chú ý hai câu lệnh using dưới đây được dùng trong lớp OleDb:

```
using System;
using System.Data.OleDb;
```

Tất cả các trình cung cấp dữ liệu đều sẵn chứa bên trong các data DLL, vì vậy chỉ cần tham chiếu đến System.Data.dll assembly để dùng cho các lớp trong phần này:

```
public class DataReaderExample
{
  public static void Main(string[] args)
  {
    string source = "Provider=SQLOLEDB;" +
                    "server=(local)\\NetSDK;" +
                    "uid=QSUser;pwd=QSPassword;" +
                    "database=northwind";
    string select = "SELECT ContactName,CompanyName FROM Customers";
    OleDbConnection conn = new OleDbConnection(source);
    conn.Open();
    OleDbCommand cmd = new OleDbCommand(select , conn);
    OleDbDataReader aReader = cmd.ExecuteReader();
    while(aReader.Read())
      Console.WriteLine("'{0}' from {1}" ,
                        aReader.GetString(0) , aReader.GetString(1));
    aReader.Close();
    conn.Close();
  }
}
```

Mã nguồn trên đây bao gồm các đoạn lệnh quen thuộc đã được trình bày trong các chương trước. Để biên dịch ví dụ này, ta dùng các dòng lệnh sau:

csc /t:exe /debug+ DataReaderExample.cs /r:System.Data.dll

Mã sau đây từ ví dụ trên cho phép tạo một kết nối OLE DB .NET, dựa trên chuỗi kết nối:

```
OleDbConnection conn = new OleDbConnection(source);
conn.Open();
OleDbCommand cmd = new OleDbCommand(select, conn);
```

Dòng thứ ba tạo một đối tượng OleDbCommand mới, dựa vào câu lệnh SELECT, kết nối sẽ thực thi câu lệnh lệnh này. Nếu bạn tạo một command hợp lệ, bạn có thể thực thi chúng để trả về một minh dụ OleDbDataReader:

```
OleDbDataReader aReader = cmd.ExecuteReader();
```

Mội OleDbDataReader chỉ là một con trỏ "connected" định trước. Mặt khác, bạn có thể chỉ duyệt qua các mẫu tin được trả về, kết nối hiện tạo sẽ lưu giữ các mẫu tin đó cho đến khi data reader bị đóng lại.

Lớp OleDbDataReader không thể tạo minh dụ một cách trực tiếp – nó luôn được trả về thông qua việc gọi phương thức ExecuteReader() của lớp OleDbCommand. Nhưng bạn có thể mở một data reader, có một số cách khác nhau để truy cập dữ liệu trong reader.

Khi một đối tượng OleDbDataReader bị đóng lại (thông qua ciệc gọi phương thức Close(), hoặc một đợt thu dọn rác), kết nối bên dưới có thể bị đóng lại thông qua một lời gọi phương thức ExecuteReader(). Nếu bạn gọi ExecuteReader() và truyền CommandBehavior.CloseConnection, bạn có thể ép kết nối đóng lại khi đóng reader.

Lớp OleDbDataReader có một bộ các quyền truy xuất thông qua các mảng quen thuộc:

```
object o = aReader[0];
object o = aReader["CategoryID"];
```

Ở đây CategoryID là trường đầu tiên trong câu lệnh SELECT của reader, cả hai dòng trên đều thực hiện công việc giống nhau tuy nhiên cách hai hơi chậm hơn cách một – Tôi đã viết một ứng dụng đơn giản để thực thi việc lập lại quá trình truy cập cho hàng triệu lần một cột trong một mẫu tin reader, chỉ để lấy một vài mẫu. Tôi biết bạn hầu như không bao giờ đọc một cột giống nhau hàng triệu lần, nhưng có thể là một số lần, bạn nên viết mã để tối ưu quá trình đó.

Bạn có biết kết quả là thế nào không, việc truy cập môt triệu lần bằng số thứ tự chỉ tốn có 0.09 giây, còn dùng chuỗi kí tự phải mất 0.63 giây. Lí do của sự chậm trễ này là vì khi dùng chuỗi kí tự ta phải dò trong schema để lấy ra số thứ tự của cột từ đó mới truy xuất được cơ sở dữ liệu. Nếu bạn biết được các thông tin này bạn có thể viết mã truy xuất dữ liệu tốt hơn.

Vì vậy việc dùng chỉ số cột là cách dùng tốt nhất.

Hơn thế nữa, OleDbDataReader có một bộ các phương thức type-safe có thể dùng để đọc các cột. Những phương thức này có thể đọc hầu hết các loại dữ liệu như GetInt32, GetFloat, GetGuid, vân vân.

Thí nghiệm của tôi khi dùng GetInt32 là 0.06 giây. Nhanh hơn việc dùng chỉ số cột, vì khi đó bạn phải thực hiện thao tác ép kiểu để đưa kiểu trả về kiểu integer. Vì vậy nếu biết trước schema bạn nên dùng các chỉ số thay vì tên.

Chắc bạn cũng biết nên giữ sự cân bằng giữa tính dễ bảo trì và tốc độ. Nếu bạn muốn dùng các chỉ mục, bạn nên định nghĩa các hằng số cho mỗi cột mà bạn sẽ truy cập.

Ví dụ dưới đây giống như ví dụ ở trên nhưng thay vì sử dụng OLE DB provider thì ở đây sử dụng SQL provider. Nhưng phần thay đổi của mã so với ví dụ trên được tô đậm. Ví dụ này nằm trong thư mục 04_DataReaderSql:

```
using System;
using System.Data.SqlClient;
```

```
public class DataReaderSql
{
  public static int Main(string[] args)
  {
    string source = "server=(local)\\NetSDK;" +
                "uid=QSUser;pwd=QSPassword;" +
                "database=northwind";
    string select = "SELECT ContactName,CompanyName FROM Customers";
    SqlConnection conn = new SqlConnection(source);
    conn.Open();
    SqlCommand cmd = new SqlCommand(select , conn);
    SqlDataReader aReader = cmd.ExecuteReader();
    while(aReader.Read())
      Console.WriteLine("'{0}' from {1}" , aReader.GetString(0) ,
                    aReader.GetString(1));
    aReader.Close();
    conn.Close();
    return 0;
  }
}
```

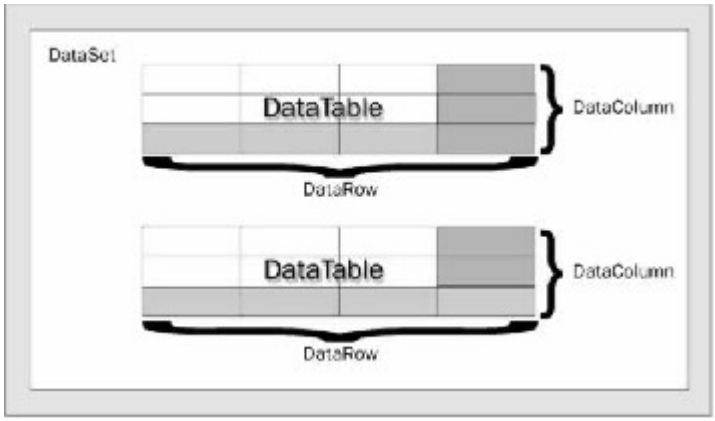
Tôi đã chạy thử nghiệm của mình trên SQL provider, và kết quả là 0.13 giây cho một triệu lần truy cập bằng chỉ mục, và 0.65 giây nếu dùng chuỗi. Bạn có mong rằng SQL Server provider nhanh hơn so với OleDb, tôi đã test thử nghiệm của mình trong phiên bản .NET.

Nếu bạn có hứng thú chạy mã này trên máy tính của bạn thì nó nằm trong các ví dụ 05_IndexerTestingOleDb và 06_IndexerTestingSql trong mã bạn đã down về.

VI. MANAGING DATA VÀ RELATIONSHIPS: THE DATASET

Lớp DataSet được thiết kế như là một thùng chứa các dữ liệu không kết nối. Nó không có khái niệm về các kết nối dữ liệu. Thật vậy, dữ liệu được giữ trong một DataSet không quan tâm đến nguồn cơ sở dữ liệu – nó có thể chỉ là những mẫu tin chứa trong một file CSV, hoặc là những đầu đọc từ một thiết bị đo lường.

Một DataSet bao gồm một tập các bảng dữ liệu, mỗi bảng là một tập các cột dữ liệu và dòng dữ liệu. Thêm vào đó là các định nghĩa dữ liệu, bạn có thể định nghĩa các *link* giữa các DataSet. Mối quan hệ phổ biến giữa các DataSet là parent-child relationship. Một mẫu tin trong một bảng (gọi là Order) có thể liên kết với nhiều mẫu tin trong bảng khác (Bảng Order_Details). Quan hệ này có thể được định nghĩa và đánh dấu trong DataSet.



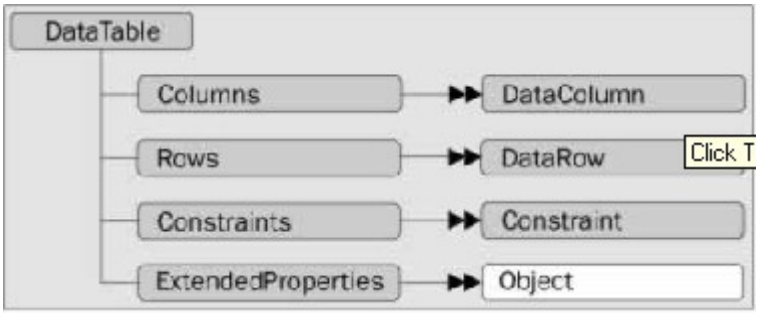
Phần dưới đây giải thích các lớp được dùng trong một DataSet.

VI.1. Data Tables

Một data table rất giống một bảng cơ sở dữ liệu vật lí – nó bao gồm một bộ các cột với các thuộc tính riêng, và có thể không chứa hoặc chứa nhiều dòng dữ liệu. Một data table có thể định nghĩa một khóa chínhm, bao gồm một hoặc nhiều cột, và cũng có thể chứa các ràng buộc của các cột. Tất cả các thông tin đó được thể hiện trong **schema**.

Có nhiều các để định nghĩa một schema cho một bảng dữ liệu riêng. Chúng sẽ được thảo luận ngay sau phần giới thiệu về cột dữ liệu và dòng dữ liệu.

Sơ đồ dưới đây chỉ ra một vài đối tượng có thể truy cập thông qua một bảng dữ liệu:



Một đối tượng DataTable (cũng như một DataColumn) có thể có một số các mở rộng riêng liên quan đến thuộc tính của nó. Tập hợp này có thể nằm trong thông tin user-defined gắng liền với đối tượng. Ví dụ, một cột có thể đưa ra một mặt nạ nhập liệu dùng để giới hạn các giá trị hợp lệ cho cột đó – một ví dụ về số phúc lợi xã hội Mĩ. Các thuộc tính mở rộng đặc biệt quan trọng khi dữ liệu được cấu trúc ở một tầng giữa và trả về cho client trong một số tiến trình. Bạn có thể lưu một chuẩn hợp lệ (như min và max) cho các số của các cột.

Khi một bảng dữ liệu được tạo ra, có thể do việc chọn dữ liệu từ một cơ sở dữ liệu, đọc dữ liệu từ một file, hoặc truy xuất thủ công trong mã, tập hợp Rows được dùng để chứa giá trị trả về.

Tập hợp Columns chứa các thể hiện DataColumn có thể được thêm vào bảng này. Những định nghĩa schema của dữ liệu, ví dụ như kiểu dữ liệu, tính khả rỗng, giá trị mặc định, vân vân... Tập Constraints có thể được tạo ra bởi các ràng buộc khóa chính hoặc tính độc nhất.

Thông tin về sơ đồ của một bảng dữ liệu có thể được sử dụng trong việc biểu diễn của một bảng dữ liệu bằng DataGrid (chúng ta sẽ bàn về vấn đề này trong chương sau). Điều khiển DataGrid sử dụng các thuộc tính như kiểu dữ liệu của cột để quyết định điều khiển gì dùng cho cột đó. Một trường bit trong cơ sở dữ liệu có thể được biểu diễn như một checkbox trong DataGrid. Nếu một cột được định nghĩa trong cơ sở sơ đồ dữ liệu như là một NOT NULL, lựa chọn này được lưu trữ trong DataColumn vì vậy nó sẽ được kiểm tra khi người dùng cố gằng di chuyển khỏi một dòng.

VI.1.1. Data Columns

Một đối tượng DataColumn định nghĩa các thuộc tính của một cột trong DataTable, chẳng hạn như kiểu dữ liệu của cột đó, chẳng hạn cột là chỉ đọc, và các sự kiện khác. Một cột có thể được tạo bằng mã, hoặc có thể được tạo tự động trong thời gian chạy.

Khi tạo một cột, tốt hơn hết là nên đặt cho nó một cái tên; nếu không thời gian chạy sẽ tự động sinh cho bạn một cái tên theo định dạng Columnn, n là mố sô tự động tăng.

Kiểu dữ liệu của một cột có thể cài đặt bằng cách cung cấp trong cấu trúc của nó, hoặc bằng cách cài đặt thuộc tính DataType. Một khi bạn đã load dữ liệu vào một bảng dữ liệu bạn không thể sửa lại kiểu dữ liệu của một cột – nếu không bạn sẽ nhận một ngoại lệ.

Các cột dữ liệu có thể được tạo để giữ các kiểu dữ liệu của .NET Framework sau:

Boolean	Decimal	Int64	TimeSpan
Byte	Double	Sbyte	UInt16
Char	Int16	Single	UInt32
DateTime	Int32	String	UInt64

Một khi đã được tạo, bước tiếp theo là cài các thuộc tính khác cho đối tượng DataColumn, chẳng hạn như tính khả rỗng nullability, giá trị mặc định. Đoạn mã sau chỉ ra một số các tùy chọn được cài đặt trong một DataColumn:

```
DataColumn customerID = new DataColumn("CustomerID" , typeof(int));
customerID.AllowDBNull = false;
customerID.ReadOnly = false;
customerID.AutoIncrement = true;
customerID.AutoIncrementSeed = 1000;
DataColumn name = new DataColumn("Name" , typeof(string));
name.AllowDBNull = false;
name.Unique = true;
```

Các thuộc tính sau có thể được cài đặt trong một DataColumn:

Property	**Description**
AllowDBNull	Nếu là true, cho phép cột có thể chấp nhận DBNull.
AutoIncrement	Cho biết rằng dữ liệu của cột này là một số tự động tăng.
AutoIncrementSeed	Giá trị khởi đầu cho một cột AutoIncrement.
AutoIncrementStep	Cho biết bước tăng giữa các giá trị tự động, mặc định là 1.
Caption	Có thể dùng cho việc biểu diễn tên của cột trên màn hình.

ColumnMapping	Cho biết cách một cột ánh xạ sang XML khi một DataSet được lưu bằng cách gọi phương thức DataSet.WriteXml.
ColumnName	Tên của cột. Nó tự động tạo ra trong thời gian chạy nếu không được cài đặt trong cấu trúc.
DataType	Kiểu giá trị của cột.
DefaultValue	Dùng để định nghĩa giá trị mặc định cho một cột
Expression	Thuộc tính này định nghĩa một biểu thức dùng cho việct tính toán trên cột này

VI.1.2 Data Rows

Lớp này cấu thành các phần khác của lớp DataTable. Các cột trong một data table được định nghĩa trong các thuộc tính của lớp DataColumn. Dữ liệu của bảng thật sự có thể truy xuất được nhờ vào đối tượng DataRow. Ví dụ sau trình bày cách truy cập các dòng trong một bảng dữ liệu. Trước tiên là các thông tin về kết nối:

```
string source = "server=(local)\\NetSDK;" + "uid=QSUser;pwd=QSPassword;" +
                "database=northwind";string select = "SELECT ContactName,
CompanyName FROM Customers";
SqlConnection  conn = new SqlConnection(source);
```

Mã sau đây giới thiệu lớp SqlDataAdapter, được dùng để điền dữ liệu cho một DataSet. SqlDataAdapter sẽ phát ra các SQL, và điền vào một bảng Customers trong DataSet. Chúng ta sẽ bàn về lớp data adapter trong phần *Populating a DataSet* dưới đây.

```
SqlDataAdapter da = new SqlDataAdapter(select, conn);
DataSet ds = new DataSet();
da.Fill(ds , "Customers");
```

Trong mã dưới đây, bạn chú ý cách dùng chỉ mục của DataRow để truy xuất giá trị trong dòng đó. Giá trị của một cột có thể trả về bằng cách dụng một trong những chỉ mục được cài đè. Chúng cho phép bạn trả về một giá trị cho biết số, tên, hoặc DataColumn:

```
foreach(DataRow row in ds.Tables["Customers"].Rows)
  Console.WriteLine("'{0}' from {1}" , row[0] ,row[1]);
```

Một trong những điều quan trọng nhất của một DataRow là phiên bản của nó. Điều đó cho phép bạn nhận được những giá trị khác nhau cho một dòng cụ thể. Các phiên bản được mô tả trong bảng sau:

DataRowVersion Value	**Description**
Current	Giá trị sẵn có của cột. Nếu không xảy một hiệu chỉnh nào, nó sẽ mang giá trị gốc. Nếu có một hiệu chỉnh xảy ra, giá trị sẽ là giá trị hợp lệ cuối cùng được cập nhật.
Default	Giá trị mặc định (nói một cách khác, giá trị mặc định được cài đặt cho cột).
Original	Giá trị của cột trong cơ sở dữ liệu vào lúc chọn. Nếu phương thức AcceptChanges DataRow được gọi, thì giá trị này sẽ được cập nhật thành giá trị hiện tại.
Proposed	Khi các thay đổi diễn ra trên một dòng nó có thể truy lục giá trị thay đổi này. Nếu bạn gọi BeginEdit() trên mộg dòng và tạo các thay đổi, mỗi một cột giữ một giá trị cho

DataRowVersion Value	Description
	đến khi phương thức EndEdit() hoặc CancelEdit() được gọi.

Phiên bản của một cột có thể dùng theo nhiều cách. Một ví dụ cho việc cập nhật các dòng trong cơ sở dữ liệu, đó là một câu lệnh SQL phổ biến như sau:

```
UPDATE Products
SET    Name = Column.Current
WHERE  ProductID = xxx
AND    Name = Column.Original;
```

Rõ ràng mã này không bao giờ được biên dịch, nhưng nó chỉ ra một cách dùng cho các giá trị hiện tại và gốc của một cột trong một dòng.

Để trả về một giá trị từ DataRow, dùng các phương thức chỉ mục thừa nhận một giá trị DataRowVersion như là một tham số. Đoạn mã sau đây chỉ ra cách đạt được tất cả các giá trị cho mỗi cột của một DataTable:

```
foreach (DataRow row in ds.Tables["Customers"].Rows )
{
  foreach ( DataColumn dc in ds.Tables["Customers"].Columns )
  {
    Console.WriteLine ("{0} Current  = {1}" , dc.ColumnName ,
                                  row[dc,DataRowVersion.Current]);
    Console.WriteLine ("    Default  = {0}" , row[dc,DataRowVersion.Default]);
                Console.WriteLine    ("    Original    =    {0}"    ,
row[dc,DataRowVersion.Original]);
  }
}
```

Mỗi dòng có một cờ trạng thái gọi là RowState, nó có thể dùng để xác định thực thi nào là cần thiết cho dòng đó khi nó cập nhật cơ sở dữ liệu. Thuộc tính RowState có thể được cài đặt để theo dõi tất cả các trạng thái thay đổi trên DataTable, như thêm vào các dòng mới, xóa các dòng hiện tại, và thay đổi các cột bên trong bảng. Khi dữ liệu được cập nhật vào cơ sở dữ liệu, cờ trạng thái được dùng để nhận biết thực thi SQL nào sẽ xảy ra. Những cờ này được định nghĩa bởi bảng liệt kê DataRowState:

DataRowState Value	Description
Added	Dòng được vừa mới được thêm vào tập hợp DataTable's Rows. Tất cả các dòng đươc tạo trên máy khách đều được cài đặt giá trị này, và cuối cùng là phát ra câu lệnh SQL INSERT khi cập nhật cho cơ sở dữ liệu.
Deleted	Giá trị này cho biết dòng đó có thể được đánh dấu xoá trong DataTable bởi phương thức DataRow.Delete(). Dòng này vẫn tồn tại trong DataTable, nhưng không thể trông thấy từ màn hình (trừ khi một DataView được cài đặt rõ ràng). Các DataView sẽ được trình bày trong chương tiếp theo. Các dòng được đánh dấu trong DataTable sẽ bị xoá khỏi cơ sở dữ liệu khi nó được cập nhật.

DataRowState Value	Description
Detached	Một dòng sẽ có trạng thái này ngay sau khi nó được tạo ra , và có thể cũng trả về trạng thái này bởi việc gọi phương thức DataRow.Remove(). Một dòng detached không được coi là một thành phần của bảng dữ liệu.
Modified	Một dòng sẽ được Modified nếu giá trị trong cột bất kì bị thay đổi.
Unchanged	Một dòng sẽ không thay đổi kể từ lần cuối cùng gọi AcceptChanges().

Trạng thái của một dòng phụ thuộc vào phương thức mà dòng đó đã gọi. Phương thức AcceptChanges() thường được gọi sau một cập nhật dữ liệu thành công (có nghĩa là sau khi thực hiện cập nhật cơ sở dữ liệu).

Cách phổ biến nhất để thay đổi dữ liệu trong một DataRow là sử dụng chỉ số, tuy vậy nếu bạn có một số thay đổi bạn ccũgn cần gọi các phương thức BeginEdit() và EndEdit() methods.

Khi một cập nhật được tạo ra trên một cột trong một DataRow, sự kiện ColumnChanging sẽ được phát ra trên các dòng của DataTable. Nó cho phép bạn ghi đè lên thuộc tính ProposedValue của các lớp DataColumnChangeEventArgs, và thay đổi nó nếu muốn. Cách này cho phép các giá tri trên cột có hiệu lực . Nếu bạn gọi BeginEdit() trước khi tạo thay đổi, sự kiện ColumnChanging vẫn xảy ra. Chúng cho phép bạn tạo một sự thay đổi kép khi cố gọi EndEdit(). Nếu bạn muốn phục hồi lại giá trị gốc, hãy gọi CancelEdit().

Một DataRow có thể liên kết với một vài dòng khác của dữ liệu. Điều này cho phép tạo các liên kết có thể điều khiển được giữa các dòng, đó là kiểu master/detail. DataRow chứa một phương thức GetChildRows() dùng để thay đổi một mảng các dòng liên quan đến các cột từ một bản khác trong cùng DataSet như là dòng hiện tại. Chúng sẽ được trình bày trong phần *Data Relationships* nằm ở phần sau của chương này.

VI1.3. Schema Generation

Có ba cách để tạo một schema cho một DataTable. Đó là:

Hãy để thời gian chạy làm điều đó giúp bạn

Viết mã tạo các bảng

Dùng trình tạo sơ đồ XML

Runtime Schema Generation

Ví dụ về DataRow ở trên đã chỉ ra mã để chọn dữ liệu từ một cơ sở dữ liệu và tạo ra môt DataSet:

```
SqlDataAdapter da = new SqlDataAdapter(select , conn);
DataSet ds = new DataSet();
da.Fill(ds , "Customers");
```

Nó rõ ràng dễ sử dụng, nhưng nó cũng có môt vài trở ngại. Một ví dụ là bạnc phải làm việc với tên cột được chọn từ cơ sở dữ liệu, điều đó cũng tốt thôi, nhưng chăc rằng muốn đổi tên vật lí thành tên thân thiện hơn.

Bạn có thể thực hiện việc đổi tên một cách thủ công trong mệnh đề SQL, chẳng hạn như trong SELECT PID AS PersonID FROM PersonTable; bạn luôn

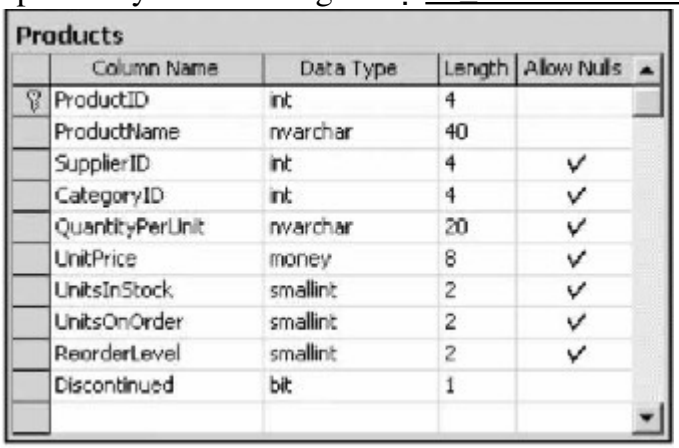
được cảnh báo không nên đổi tên các cột trong SQL, chỉ thay thế một cột khi thật sự cần để tên xuất hiện trên màn hình được thân thiện hơn.

Một vấn đề tiềm ẩn khác không các trình phát DataTable/DataColumn tự động là bạn không thể điều khiển vượt quá kiểu của cột, các kiểu này được thời gian chạy lựa chọn cho bạn. Nó rất có ích trong việc chọn kiểu dữ liệu đúng cho bạn, nhưng trong nhiều trường hợp bạn muốn có nhiều khả năng hơn . Ví dụ bạn cần định nghĩa một tập các kiểu giá trị dùng cho một cột, vì vậy mã cần phải được viết lại. Nếu bạn chấp nhận kiểu giá trị măc định cho các cột đươc tạo ra trong thời gian chạy, có thể là một số nguyên 32-bit.

Cuối cùng một điều rất quan trọng, đó là sử dụng các trình tạo bảng tự động, bạn không thể truy xuất dữ liệu access to the data within the DataTable – you are at the mercy of indexers, which return instances of object rather than derived data types. If you like sprinkling your code with typecast expressions then skip the following sections.

Hand-Coded Schema

Việc phát ra mã để tạo một DataTable, với đầy đủ các cột là một việc tương đối đơn giản. Các ví dụ trong phần này sẽ truy cập bảng Products từ cơ sỏ dữ liệu Northwind. Mã của phần này sẵn có trong ví dụ 08_ManufacturedDataSet.



Products

Column Name	Data Type	Length	Allow Nulls
ProductID	int	4	
ProductName	nvarchar	40	
SupplierID	int	4	✓
CategoryID	int	4	✓
QuantityPerUnit	nvarchar	20	✓
UnitPrice	money	8	✓
UnitsInStock	smallint	2	✓
UnitsOnOrder	smallint	2	✓
ReorderLevel	smallint	2	✓
Discontinued	bit	1	

Dưới đây là mã để tạo thủ công một DataTable, có sơ đồ như trên.

```
public static void ManufactureProductDataTable(DataSet ds)
{
  DataTable   products = new DataTable("Products");
  products.Columns.Add(new DataColumn("ProductID", typeof(int)));
  products.Columns.Add(new DataColumn("ProductName", typeof(string)));
  products.Columns.Add(new DataColumn("SupplierID", typeof(int)));
  products.Columns.Add(new DataColumn("CategoryID", typeof(int)));
  products.Columns.Add(new DataColumn("QuantityPerUnit", typeof(string)));
  products.Columns.Add(new DataColumn("UnitPrice", typeof(decimal)));
  products.Columns.Add(new DataColumn("UnitsInStock", typeof(short)));
  products.Columns.Add(new DataColumn("UnitsOnOrder", typeof(short)));
  products.Columns.Add(new DataColumn("ReorderLevel", typeof(short)));
  products.Columns.Add(new DataColumn("Discontinued", typeof(bool)));
  ds.Tables.Add(products);
}
```

Bạn có thể sửa đổi mã trong ví dụ DataRow và sử dụng các định nghĩa sau:

```
string source = "server=localhost;" +
            "integrated security=sspi;" +
            "database=Northwind";
string select = "SELECT * FROM Products";
SqlConnection conn = new SqlConnection(source);
SqlDataAdapter cmd = new SqlDataAdapter(select, conn);
DataSet ds = new DataSet();
ManufactureProductDataTable(ds);
cmd.Fill(ds, "Products");
foreach(DataRow row in ds.Tables["Products"].Rows)
  Console.WriteLine("'{0}' from {1}", row[0], row[1]);
```

Phương thức ManufactureProductDataTable() tạo một DataTable mới, thay đổi cho từng cột, và sau đó thêm nó vào danh sách các bảng trong DataSet. DataSet có một bộ chỉ mục nắm giữ tên của bảng và trả về DataTable được gọi.

Ví dụ trên không thật sự là bảo toàn kiểu, Tôi đã dùng bộ chỉ mục cột để lấy dữ liệu. Tốt hơn hết là dùng một lớp (hoặc một bộ các lớp) để điều khiển các DataSet, DataTable, và DataRow, dùng để định nghĩa các bộ truy xuất bảo vệ kiểu cho các bảng, các dòng, các cột. Bạn có thể viết mã của mình – đó quả là một công việc chán nản, bạn có thể sử dụng các lớp bảo vệ kiểu sẵn có.

.NET Framework có các hỗ trợ cho việc dùng các sơ đồ XML để định nghĩa một DataSet, DataTable, và các lớp khác mà chúng ta có thể làm trong phần này. Phần *XML Schemas* nằm trong chương này sẽ bàn về các phương thức này, nhưng trước tiên, chúng ta sẽ xem xét về các quan hệ và ràng buộc trong một DataSet.

VI.1.4. Các quan hệ dữ liệu

Khi viết một ứng dụng, thường cần phải có sẵn nhiều bảng để lưu trữ thông tin. Lớp DataSet là một nơi để chứa các thông tin này.

Lớp DataSet là một thiết kế để tạo nên các mối quan hệ giữa các các bảng. Mã trong phần này được tôi thiết kế để tạo bằng tay mối quan hệ cho hai bảng dữ liệu. Vì vậy, nếu bạn không có SQL Server hoặc cơ sở dữ liệu NorthWind, bạn cũng có thể chạy ví dụ này.

```
DataSet ds = new DataSet("Relationships");
ds.Tables.Add(CreateBuildingTable());
ds.Tables.Add(CreateRoomTable());
ds.Relations.Add("Rooms",
            ds.Tables["Building"].Columns["BuildingID"],
            ds.Tables["Room"].Columns["BuildingID"]);
```

Các bảng đơn giản chứa một khóa chính và một trường tên, trong đó bảng Room có một khóa ngoại BuildingID.

Sau đó thêm một số dữ liệu cho mỗi bảng.

```
foreach(DataRow theBuilding in ds.Tables["Building"].Rows)
{
```

```
    DataRow[] children = theBuilding.GetChildRows("Rooms");
    int roomCount = children.Length;
    Console.WriteLine("Building {0} contains {1} room{2}",
                theBuilding["Name"],
                roomCount,
                roomCount > 1 ? "s" : "");
    // Loop through the rooms
    foreach(DataRow theRoom in children)
      Console.WriteLine("Room: {0}", theRoom["Name"]);
  }
```

Sự khác biệt lớn nhất giữa DataSet và kiểu đối tượng Recordset cổ điển là sự biểu hiện của quan hệ. Trong một Recordset cổ điển, một quan hệ được biểu diễn là một cột giả trong dòng. Cột này bản thân nó là một Recordset có thể lập lại. Trong ADO.NET, một quan hệ đơn giản là một lời gọi phương thức GetChildRows():

```
DataRow[] children = theBuilding.GetChildRows("Rooms");
```

Phương thức này có một số kiểu, ví dụ trên chỉ ra cách dùng tên của quan hệ. Nó trả về một mảng các dòng có thể cập nhật bằng bộ chỉ mục như đã đề cập ở các ví dụ trước đây.

Thích thú hơn là quan hệ dữ liệu có thể xem xét theo hai cách. Không chỉ có thể đi từ cha đến con, mà có thể tìm được các dòng cha của một mẫu tin con bằng cách sử dụng thuộc tính ParentRelations trên lớp DataTable. Thuộc tính này trả về một DataRelationCollection, có thể truy cập bằng kí hiệu mảng [] (ví dụ, ParentRelations["Rooms"]), hoặc dùng luân phiên phương thức GetParentRows() như mã dưới đây:

```
foreach(DataRow theRoom in ds.Tables["Room"].Rows)
{
  DataRow[] parents = theRoom.GetParentRows("Rooms");
  foreach(DataRow theBuilding in parents)
    Console.WriteLine("Room {0} is contained in building {1}",
                theRoom["Name"],
                theBuilding["Name"]);
}
```

Có hai phương thức với rất nhiều các cài đặt đè khác nhau để trả về các dòng cha – GetParentRows() (trả về một mảng các dòng), hoặc GetParentRow() (trả về một dòng cha duy nhất của một quan hệ).

VI.1.5. Ràng buộc dữ liệu

Thay đổi kiểu dữ liệu của một cột đã được tạo trên một máy đơn không chỉ là một khả năng tuyệt vời của một DataTable. ADO.NET cho phép bạn tạo một tập các ràng buộc trên một cột (hoặc nhiều cột), dùng cho các nguyên tắc chuẩn hóa dữ liệu.

Thời gian chạy hỗ trợ các kiểu ràng buộc sau, như là các lớp trong không gian System.Data.

Constraint	**Description**
ForeignKeyConstraint	Thực một liên kết giữa hai DataTables trong một DataSet

Constraint	Description
UniqueConstraint	Bảo đảm tính độc nhất của cột

Cài đặt khóa chính

Một điều phổ biến của một bảng trong một cơ sở dữ liệu quan hệ, bạn có thể cung cấp một khóa chính, dựa vào một hoặc nhiều cột trong một DataTable.

Mã sau tạo một khóa chính cho bảng Products, mà sơ đồ của nó đã được tạo bằng thủ công trong các ví dụ trên, chúng ta có thể tìm thấy trong thư mục 08_ManufactureDataSet.

Chú ý rằng một khóa chính cảu một bảng chỉ là một kiểu của ràng buộc. Khi một khóa chính được thêm vào một DataTable, thời gian chạy cũng phát ra một ràng buộc độc nhất trên khóa chính. Bởi vì thực tế không tồn tại kiều ràng buộc PrimaryKey – một khóa chính đơn giản là một ràng buộc duy nhất trên một hoặc nhiều cột.

```
public static void ManufacturePrimaryKey(DataTable dt)
{
  DataColumn[] pk = new DataColumn[1];
  pk[0] = dt.Columns["ProductID"];
  dt.PrimaryKey = pk;
}
```

Một khóa chính có thể bao gồm một vài cột, nó được xem như là một mảng các DataColumns. Một khóa chính của một bảng được cài đặt trên những cột này đơn giản được còi là một mảng của các cột làm nên thuộc tính.

Để kiểm tra các ràng buộc của một bảng, bạn có thể lập lại ConstraintCollection. Đối với ràng buộc tự sinh như ví dụ trên, tên của ràng buộc sẽ là Constraint1. Nó không phải là một tên tốt, vì vậy tốt nhất là nên tạo ràng buộc trước sau đó định nghĩa các cột tạo nên khóa chính, như chúng ta sẽ làm dưới đây.

Là một lập trình viên cơ sở dữ liệu lâu năm, tôi nhận thấy ràng tên của một ràng buộc cần phải thật rõ nghĩa. Mã dưới đây định danh ràng buộc trước khi tạo khóa chính:

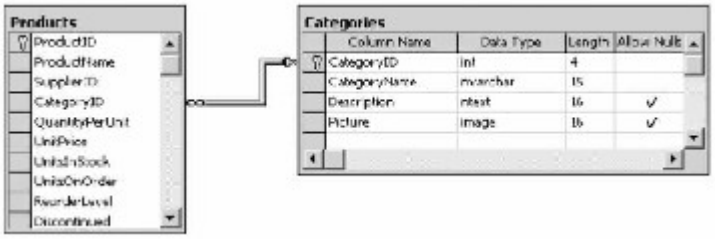
```
DataColumn[] pk = new DataColumn[1];
pk[0] = dt.Columns["ProductID"];
dt.Constraints.Add(new UniqueConstraint("PK_Products", pk[0]));
dt.PrimaryKey = pk;
```

Ràng buộc duy nhất có thể áp dụng cho bao nhiêu cột tùy thích.

Tạo một khóa ngoại

Ngoài các ràng buộc duy nhất, một DataTable có thể chứa các ràng buộc khóa ngoại. Nó thường được áp dụng cho các mối quan hệ chủ tớ, nhưng cũng có thể dùng để tạo bảng sao các cột giữa các bảng nếu bạng tạo một ràng buộc chính xác. Một quan hệ chủ tớ là một mẫu tin cha có thể có nhiều mẫu tin con, liên kết với khóa chính của mẫu tin cha.

Một ràng buộc khóa ngoại có thể chỉ thực thi trên các bảng bên trong một DataSet, ví dụ dưới đây sử dụng bảng Categories trong cơ sở dữ liệu Northwind, và tạo một ràng buộc giữa nó với bảng Products table.



Bước đầu tiên là tạo một bảng dữ liệu mới cho bảng Categories. Ví dụ :

```
DataTable categories = new DataTable("Categories");
categories.Columns.Add(new DataColumn("CategoryID", typeof(int)));
categories.Columns.Add(new DataColumn("CategoryName", typeof(string)));
categories.Columns.Add(new DataColumn("Description", typeof(string)));
categories.Constraints.Add(new UniqueConstraint("PK_Categories",
                    categories.Columns["CategoryID"]));
categories.PrimaryKey = new DataColumn[1]
                    {categories.Columns["CategoryID"]};
```

Dòng cuối cùng của mã trên tạo một khóa chính cho bảng Categories. Khóa chính là một cột đơn, tất nhiên nó cũng thể tạo một khóa chính trên nhiều cột bằng các dùng kí tự mảng.

Sau đó tôi tạo một ràng buộc giữa hai bảng:

```
DataColumn parent = ds.Tables["Categories"].Columns["CategoryID"];
DataColumn child = ds.Tables["Products"].Columns["CategoryID"];
ForeignKeyConstraint fk =
  new ForeignKeyConstraint("FK_Product_CategoryID", parent, child);
fk.UpdateRule = Rule.Cascade;
fk.DeleteRule = Rule.SetNull;
ds.Tables["Products"].Constraints.Add(fk);
```

Ràng buộc này dùng để liên kết giữa Categories.CategoryID và Products.CategoryID. Có bốn cấu trúc khác nhau cho ForeignKeyConstraint, nhưng tôi khuyên bạn nên dùng tên của ràng buộc.

Tạo các ràng buộc Update và Delete

Bổ sung cho phần định nghĩa tất nhiên là một vài kiểu của ràng buộc giữa các bảng cha và con, bạn có thể định nghĩa phải làm gì trong một ràng buộc cập nhật.

Ví dụ trên tạo một qui tắc cập nhật và một qui tắc xóa. Những qui tắc này được dùng khi môt sự kiện được phát ra trên cột (hoặc dòng) trong bảng cha, và qui tắc được dùng để quyết định chuyện gì sẽ xảy ra trong bảng con. Có bốn qui tắc khác nhau có thể áp dụng được liệt kê trong Rule enumeration:

Cascade – Nếu khóa cha được cập nhật sau đó copy giá trị mới này cho tất cả các mã của khóa con. Nếu mẫu cha bị xoá, thì xóa luôn các mẫu con. Nó là tùy chọn mặc định.

None – Không làm gì hết. Tùy chọn này sẽ bỏ các dòng mô côi khỏi bảng dữ liệu con.

SetDefault – Mỗi thay đổi trên dòng con được mang giá trị mặc định của nó, nếu nó được định nghĩa trước.

SetNull – Tất cả các dòng được chọn là DBNull.

Chú ý: Các ràng buộc chỉ có hiệu lực trong một DataSet nếu thuộc tính

EnforceConstraints của DataSet là true.

Tôi đã đổi các lớp chính dùng để tạo nên DataSet, chỉ ra cách tạo thủ công các lớp bằng mã. Có một cách khác để định nghĩa một DataTable, DataRow, DataColumn, DataRelation, và Constraint – bằng cáchc dùng các file sơ đồ XML và công cụ XSD sẵn có trong .NET. Đoạn dưới đây trình bày cách để tạo một sơ đồ đơn giản và tạo các lớp bảo vệ kiểu để truy cập dữ liệu của bạn.

VII. CÁC SƠ ĐỒ XML

XML là một đường hào vững chắc bao bọc ADO.NET - thật vậy, các định dạng điều khiển cho việc truyền dữ liệu hiện tại là XML. Với thời gian chạy .NET, nó có thể mô tả một DataTable trong một file sơ đồ XML. Hơn thế nữa, bạn có thể định nghĩa một DataSet, với một số DataTables, một bộ các quan hệ giữa các bảng, bao gồm các chi tiết và mô tả đây đủ của dữ liệu.

Khi bạn có một file XSD, có một công cụ trong thời gian chạy để chuyển sơ đồ này thành các lớp dữ liệu tương ứng, chẳng hạn như một lớp DataTable ở trên. Trong phần này chúng ta sẽ bắt đầu với một file XSD đơn giản dùng để mô tả các thông tin tương tự ví dụ Products ở trên, và sau đó tạo ra một vài tính năng mở rộng.

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema
  id="Products"
  targetNamespace="http://tempuri.org/XMLSchema1.xsd"
  elementFormDefault="qualified"
  xmlns="http://tempuri.org/XMLSchema1.xsd"
  xmlns:mstns="http://tempuri.org/XMLSchema1.xsd"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
 <xs:element name="Product">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="ProductID" type="xs:int" />
    <xs:element name="ProductName" type="xs:string" />
    <xs:element name="SupplierID" type="xs:int" minOccurs="0" />
    <xs:element name="CategoryID" type="xs:int" minOccurs="0" />
    <xs:element name="QuantityPerUnit" type="xs:string" minOccurs="0" />
    <xs:element name="UnitPrice" type="xs:decimal" minOccurs="0" />
    <xs:element name="UnitsInStock" type="xs:short" minOccurs="0" />
    <xs:element name="UnitsOnOrder" type="xs:short" minOccurs="0" />
    <xs:element name="ReorderLevel" type="xs:short" minOccurs="0" />
    <xs:element name="Discontinued" type="xs:boolean" />
   </xs:sequence>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

Chúng ta sẽ xem xét kĩ trong chương 11; còn bây giờ, file này cơ bản định nghĩa mọt sơ đồ với các thuộc tính id tạo thành Products. Một kiểu phức tạp Product đã được định nghĩa, để chứa một số các yếu tố, cho mỗi trường trong bảng Products.

Cảm ơn .NET Framework đã công cụ XSD.EXE để tạo ra tất cả các mã cho các lớp này chỉ cần một file nhập XSD.

Tạo mã với XSD

Bạn có thể lưu file trên với tên Product.xsd, và chuyển nó thành mã với lệnh sau:

xsd Product.xsd /d

Nó sẽ tạo ra file Product.cs.

Có một vài cách có thể dùng XSD để thay đổi output generated. Một vài cách phổ biến được đưa ra trong bảng sau.

Switch	**Description**
/dataset (/d)	Các lớp được thừa kế từ DataSet, DataTable, và DataRow.
/language:<language>	Cho phép bạn chon ngôn ngữ để chuyển. C# là giá trị mặc định, nhưng có thể chọn VB cho một file Visual Basic .NET.
/namespace:<namespace>	Định nghĩa không gian tên của code được phát ra. Giá trị mặc định là no namespace.

Một phiên bản ngắn gọn của XSD cho sơ đồ Products được trình bày dưới đây. Tôi đã bỏ đi môt vài mã không cần thiết chỉ giữ lại những gì quan trọng nhất, và làm một vài thao tác định dạng lại để vừa với khổ giấy. Để xem kết quả cuối cùng, chạy XSD.EXE trên sơ đồ Products và xem xet file .cs được tạo ra. Mã ví dụ trong thư mục 10_XSD_DataSet:

```
//------------------------------------------------------------------------------
// <autogenerated>
//     This code was generated by a tool.
//     Runtime Version: 1.0.3512.0
//
//     Changes to this file may cause incorrect behavior and will be lost if
//     the code is regenerated.
// </autogenerated>
//------------------------------------------------------------------------------
// This source code was auto-generated by xsd, Version=1.0.3512.0.
//
using System;
using System.Data;
using System.Xml;
using System.Runtime.Serialization;
[Serializable()]
[System.ComponentModel.DesignerCategoryAttribute("code")]
[System.Diagnostics.DebuggerStepThrough()]
[System.ComponentModel.ToolboxItem(true)]
public class Products : DataSet
{
  private ProductDataTable tableProduct;
  public Products()
```

```
  public ProductDataTable Product
  public override DataSet Clone()
  public delegate void ProductRowChangeEventHandler ( object sender,
                                        ProductRowChangeEvent e);
  [System.Diagnostics.DebuggerStepThrough()]
  public class ProductDataTable : DataTable, System.Collections.IEnumerable
  [System.Diagnostics.DebuggerStepThrough()]
  public class ProductRow : DataRow
}
```

Tôi đã hơi tùy tiện trong mã này, tôi đã tách nó thành 3 phần và bỏ đi các thành phần protected và private vì vậy chúng ta chỉ có thể tập trung vào các giao diện chính. Chúng ta sẽ xem xét mã sau trong phần nó về DataSet.

Cấu trúc của Products() gọi một phương thức tĩnh, InitClass(), nó xây dựng một thể hiện của lớp ProductDataTable xuất phát từ lớp DataTable, và thêm nó vào tập hợp các Tables của DataSet. Bảng dữ liệu Products có thể được truy cập như mã dưới đây:

```
DataSet ds = new Products();
DataTable products = ds.Tables["Products"];
```

Hoặc, đơn giản hơn bằng cách sử dụng thuộc tính Product, sẵn có trong các đối tượng xuất phát từ DataSet:

```
DataTable products = ds.Product;
```

Lớp ProductDataTable bao gồm các mã khác như sau:

```
[System.Diagnostics.DebuggerStepThrough()]
public class ProductDataTable :DataTable,
System.Collections.IEnumerable
{
  private DataColumn columnProductID;
  private DataColumn columnProductName;
  private DataColumn columnSupplierID;
  private DataColumn columnCategoryID;
  private DataColumn columnQuantityPerUnit;
  private DataColumn columnUnitPrice;
  private DataColumn columnUnitsInStock;
  private DataColumn columnUnitsOnOrder;
  private DataColumn columnReorderLevel;
  private DataColumn columnDiscontinued;
  internal ProductDataTable() : base("Product")
  {
    this.InitClass();
  }
```

Lớp ProductDataTable, được xuất phát từ DataTable và thực thi giao diện IEnumerable, định nghĩa một thể hiện DataColumn tĩnh cho mỗi cột trong bảng. Chúng được khởi động một lần nữa từ cấu tử bởi việc gọi phương thức tĩnh InitClass(). Mỗi cột cung cấp một con trỏ nội, lớp DataRow được mô tả sau.

```
[System.ComponentModel.Browsable(false)]
public int Count
```

```
        {
          get { return this.Rows.Count; }
        }
        internal DataColumn ProductIDColumn
        {
          get { return this.columnProductID; }
        }
// Other row accessors removed for clarity - there is one for each of the columns
```

Việc thêm các dòng vào bảng được cung cấp bởi hai quá tải trong phương thức AddProductRow(). Quá tải thứ nhất tạo DataRow và không trả về giá trị nào cả. Quá tải thứ hai là một tập các giá trị, mỗi giá trị dành cho các cột trong DataTable, tạo các giá trị cho một dòng mới, thêm dòng vào DataTable và trả dòng về cho trình gọi.

```
        public void AddProductRow(ProductRow row)
        {
          this.Rows.Add(row);
        }
        public ProductRow AddProductRow ( string ProductName , int
        SupplierID,  int CategoryID , string QuantityPerUnit, System.Decimal
        UnitPrice , short UnitsInStock , short UnitsOnOrder , short ReorderLevel
        , bool Discontinued )
        {
          ProductRow rowProductRow = ((ProductRow)(this.NewRow()));
          rowProductRow.ItemArray = new object[]
          {
            null, ProductName, SupplierID, CategoryID, QuantityPerUnit,
            UnitPrice, UnitsInStock, UnitsOnOrder, ReorderLevel, Discontinued
          };
          this.Rows.Add(rowProductRow);
          return rowProductRow;
        }
```

Giống như thành phần InitClass() trong lớp xuất phát từ DataSet, dùng để thêm bảng vào trong DataSet, thành phần InitClass() trong ProductDataTable thêm các cột vào DataTable. Thuộc tính của mỗi cột được điền thích hợp, và cột sau đó được thêm vào tập hợp columns.

```
        private void InitClass()
        {
           this.columnProductID = new DataColumn ( "ProductID", typeof(int), null,
        System.Data.MappingType.Element);
          this.Columns.Add(this.columnProductID);
          // Other columns removed for clarity
          this.columnProductID.AutoIncrement = true;
          this.columnProductID.AllowDBNull = false;
          this.columnProductID.ReadOnly = true;
          this.columnProductName.AllowDBNull = false;
          this.columnDiscontinued.AllowDBNull = false;
```

```
        }

        public ProductRow NewProductRow()
        {
           return ((ProductRow)(this.NewRow()));
        }
```

Phương thức cuối cùng mà tôi muốn nói tới, NewRowFromBuilder(), được gọi trực tiếp từ phương thức NewRow() của DataTable. Ở đây nó tạo một dòng định kiểu mạnh. Thể hiện của DataRowBuilder được tạo bởi DataTable, các thành phần của nó chỉ có thể truy sử dụng trong nhị phân System.Data.

```
        protected override DataRow NewRowFromBuilder(DataRowBuilder builder)
        {
          return new ProductRow(builder);
        }
```

Và lớp cuối cùng được nói tới là ProductRow xuất phát từ DataRow. Lớp này được dùng để cung cấp cách truy cập bảo vệ kiểu cho tất cả các trường của dữ liệu trong bảng dữ liệu. Nó là một bao một dòng riêng và cung các các thành phần cót thể đọc và viết cho các cột trong bảng.

Các vùng khả rỗng sẽ được kiểm tra kĩ lưỡng. Ví dụ dưới đây chỉ ra các khả năng của cột SupplierID:

```
[System.Diagnostics.DebuggerStepThrough()]
public class ProductRow : DataRow
{
  private ProductDataTable tableProduct;

  internal ProductRow(DataRowBuilder rb) : base(rb)
  {
    this.tableProduct = ((ProductDataTable)(this.Table));
  }

  public int ProductID
  {
    get { return ((int)(this[this.tableProduct.ProductIDColumn])); }
    set { this[this.tableProduct.ProductIDColumn] = value; }
  }
 // Other column accessors/mutators removed for clarity

  public bool IsSupplierIDNull()
  {
    return this.IsNull(this.tableProduct.SupplierIDColumn);
  }
  public void SetSupplierIDNull()
  {
    this[this.tableProduct.SupplierIDColumn] = System.Convert.DBNull;
  }
}
```

Giờ đây chúng ta có thể kết hợp các lớp được sinh ra bởi XSD.EXE vào mã nguồn. Mã dưới đây sử dụng các lớp để nhận dữ liệu từ bảng Products và thể hiện ra màn hình console:

```
using System;
using System.Data;
using System.Data.SqlClient;
public class XSD_DataSet
{
  public static void Main()
  {
    string source = "server=(local)\\NetSDK;" +
                "uid=QSUser;pwd=QSPassword;" +
                "database=northwind";
    string select = "SELECT * FROM Products";
    SqlConnection conn = new SqlConnection(source);
    SqlDataAdapter da = new SqlDataAdapter(select , conn);
    Products ds = new Products();
    da.Fill(ds , "Product");
    foreach(Products.ProductRow row in ds.Product )
    Console.WriteLine("'{0}' from {1}" , row.ProductID ,  row.ProductName);
  }
}
```

Các phần chính được bôi đậm. Mã trên chứa một lớp Products xuất phát từ DataSet, được tạo và điền dữ liệu bởi trình cung cấp dữ liệu.

Để biên dịch ví dụ này bạn dùng lệnh sau:

xsd product.xsd /d

và

csc /recurse:*.cs

Dòng đầu tiên tạo ra file Products.cs từ sơ đồ Products.XSD, và dòng thứ hại sử dụng tham số /recurse:*.cs để duyệt tất cả các file trong extension .cs và thêm vào nhị phân cuối.

VIII.TẠO MỘT DATASET

Trước tiên bạn đã định nghĩa sơ đồ của bộ dữ liệu của bạn, với đầy đủ các DataTable, DataColumn, Constraint, và những gì cần thiết, bạn nên tạo DataSet vi một vài thông tin bổ sung. Có hai cách chính để đọc dữ liệu từ một nguồn bên ngoài và chèn nó vào DataSet:

Dùng trình cung cấp dữ liệu

Đọc XML vào trong DataSet

VIII.1.Tạo một DataSet dùng một DataAdapter

Đoạn mã về dòng dữ liệu được giới thiệu trong lớp SqlDataAdapter, được trình bày như sau:

```
string select = "SELECT ContactName,CompanyName FROM Customers";
SqlConnection conn = new SqlConnection(source);
SqlDataAdapter da = new SqlDataAdapter(select , conn);
DataSet ds = new DataSet();
```

da.Fill(ds , "Customers");

Hai dòng in đậm chỉ ra cách dùng của SqlDataAdapter – OleDbDataAdapter cũng có nhưng tính năng ảo giống như Sql equivalent.

SqlDataAdapter và OleDbDataAdapter là hai lớp xuất phát từ một lớp cơ bản chứ không phải là một bộ các giao diện, và nhất là các lớp SqlClient- hoặc OleDb. Cây kế thưa được biểu diễn như sau:

System.Data.Common.DataAdapter
System.Data.Common.DbDataAdapter
System.Data.OleDb.OleDbDataAdapter
System.Data.SqlClient.SqlDataAdapter

Trong quá trình lấy dữ liệu từ một DataSet, cần phải có một vài lệnh được dùng để chọn dữ liệu. Nó có thể là một câu lệnh SELECT, một stored procedure, hoặc OLE DB provider, một TableDirect command. Ví dụ trên sử dụng một trong những cấu trúc sẵn có trong SqlDataAdapter để truyền câu lệnh SELECT vào một SqlCommand, và phát nó khi gọi phương thức Fill() trên adapter.

Trở lại với các ví dụ về stored procedures trong chương trước, Tôi đã định nghĩa các stored procedure INSERT, UPDATE, và DELETE, nhưng chưa đưa ra một procedure để SELECT dữ liệu. Chúng ta sẽ lấp lỗ hổng này trong phần sau, và chỉ ra cách làm sao để gọi một stored procedure từ một SqlDataAdapter để tạo dữ liệu cho một DataSet.

VIII.1.1.Sử dụng một Stored Procedure trong một DataAdapter

Trước tiên chúng ta cần định nghĩa một stored procedure và cài nó vào cơ sở dữ liệu database. Stored procedure đẻ SELECT dữ liệu như sau:

```
CREATE PROCEDURE RegionSelect AS
  SET NOCOUNT OFF
  SELECT * FROM Region
GO
```

Ví dụ này tương đối đơn giản nó thật không xứng tầm với một stored procedure, chỉ là một câu lệnh SQL đơn giản. Stored procedure này có thể đánh vào SQL Server Query Analyzer, hoặc bạn có thể chạy file StoredProc.sql để sử dụng ví dụ này.

Tiếp theo, chúng ta cần định nghĩa một SqlCommand để thực thi stored procedure này. Một lần nữa mã rất đơn giản, và hầu hết đã được đưa ra trong các phần trên:

```
private static SqlCommand GenerateSelectCommand(SqlConnection conn )
{
  SqlCommand  aCommand = new SqlCommand("RegionSelect" , conn);
  aCommand.CommandType = CommandType.StoredProcedure;
  aCommand.UpdatedRowSource = UpdateRowSource.None;
  return aCommand;
}
```

Phương thức này phát ra SqlCommand để gọi thủ tục RegionSelect khi thực thi. Và cuối cùng là móc nói nó với một SqlDataAdapter thông qua lời gọi phương thức

Fill():

```
DataSet ds = new DataSet();
```

```
// Create a data adapter to fill the DataSet
SqlDataAdapter da = new SqlDataAdapter();
// Set the data adapter's select command
da.SelectCommand = GenerateSelectCommand (conn);
da.Fill(ds , "Region");
```

Ở đây tôi tạo một SqlDataAdapter mới, xem SqlCommand được phát ra thông qua thuộc tính SelectCommand của data adapter, và gọi Fill(), để thực thi stored procedure và chèn tất cả các dòng vào the Region DataTable.

VIII.1.2. Tạo một DataSet từ XML

Ngoài việc tạo sơ đồ cho một DataSet và các bảng tương ứng, một DataSet có thể đọc và ghi các dữ liệu bẩm sinh XML, giống như một file trên đĩa, một stream, hoặc một text reader.

Để load XML vào một DataSet, đơn giản gọi một trong những phương thức ReadXML(), chẳng hạn như mã sau, dùng để đọc từ một file trên đĩa:

```
DataSet ds = new DataSet();
ds.ReadXml(".\\MyData.xml");
```

IX. CÁC CỐ GẮNG THAY ĐỔI DATASET

Sau khi soạn thạo dữ liệu trong một DataSet, cũng có những lúc cần phải thay đổi nó. Một ví dụ khá phổ biến đó là chọn dữ liệu từ một cơ sở dữ liệu, biểu diễ nó cho người dùng, và cập nhật cho cơ sở dữ liệu.

IX.1.Cập nhật với các Data Adapter

Một SqlDataAdapter có thể bao gồm SelectCommand, một InsertCommand, UpdateCommand, và DeleteCommand. Giống như tên gọi, những đối tượng này là những thể hiện của SqlCommand (hoặc OleDbCommand dùng cho OleDbDataAdapter), vì vậy những câu lệnh này có thể chuyển thành SQL hoặc một stored procedure.

Trong ví dụ này, tôi đã khôi phục lại các mã stored procedure từ phần *Calling Stored Procedures* để chèn, cập nhật, và xóa các mẫu tin Region.

IX.1.1. Chèn một dòng mới

Có hai cách để thêm một dòng mới vào một DataTable. Cách thứ nhất là gọi phương thức NewRow, để trả về một dòng trống sau đó định vị và thêm vào tập Rows như sau:

```
DataRow r = ds.Tables["Region"].NewRow();
r["RegionID"]=999;
r["RegionDescription"]="North West";
ds.Tables["Region"].Rows.Add(r);
```

Cách thứ hai để thêm một dòng mới là truyền một mảng dữ liệu vào phương thức Rows.Add() giống như sau:

```
DataRow r = ds.Tables["Region"].Rows.Add
  (new object [] { 999 , "North West" });
```

Mỗi dòng trong DataTable sẽ cài RowState là Added. Ví dụ sẽ xổ ra các mẫu tin trước khi nó thay đổi được cập nhật cho dữ liệu, vì vậy sau khi thêm các dòng sau vào DataTable, các dòng sẽ giống như sau. Chú ý rằng cột bên phải là trạng thái dòng.

New row pending inserting into database

```
  1 Eastern                    Unchanged
  2 Western                    Unchanged
  3 Northern                   Unchanged
  4 Southern                   Unchanged
999 North West                 Added
```

Để cập nhật cơ sở dữ liệu từ một DataAdapter, gọi phương thức Update() như sau đây:

```
da.Update(ds , "Region");
```

Đối với một dòng mới trong DataTable, sẽ thực thi stored procedure và xuất ra các mẫu tìn trong DataTable một lần nữa.

New row updated and new RegionID assigned by database

```
  1 Eastern                    Unchanged
  2 Western                    Unchanged
  3 Northern                   Unchanged
  4 Southern                   Unchanged
  5 North West                 Unchanged
```

Hãy nhìn dòng cuối của DataTable. Tôi đã nhập RegionID trong mã là 999, nhưng sau khi sẽ đổi RegionInsert stored procedure giá trị được đổi thành 5. Có sở dữ liệu thường tạo khoá chính cho bạn.

```
SqlCommand aCommand = new SqlCommand("RegionInsert" , conn);
aCommand.CommandType = CommandType.StoredProcedure;
aCommand.Parameters.Add(new SqlParameter("@RegionDescription" ,
                SqlDbType.NChar , 50, "RegionDescription"));
aCommand.Parameters.Add(new SqlParameter("@RegionID" ,
                SqlDbType.Int, 0, ParameterDirection.Output , false, 0, 0,
                "RegionID" ,   // Defines the SOURCE column
                DataRowVersion.Default,  null));
aCommand.UpdatedRowSource = UpdateRowSource.OutputParameters;
```

Chuyện gì sẽ xảy ra khi một data adapter phát các lệnh này, các tham số xuất sẽ được ánh xạ trở lại mã nguồn của dòng. Stored procedure có một tham số xuất ánh xạ trở lại DataRow.

Giá trị của UpdateRowSource được cho trong bảng sau:

UpdateRowSource Value	**Description**
Both	Một stored procedure có thể trả về nhiều tham số xuất và cũng có thể là một cơ sở dữ liệu gồm các mẫu tin đã được cập nhật.
FirstReturnedRecord	Nó trả về một mẫu dữ liệu đơn, nội dung của mẫu tin đó có thể được trộn vào DataRow nguồn. Nó có có ích đối với một bảng có các cột mang giá trị mặc định hoặc tính toán, sau khi một câu lệnh INSERT chúng cần phải được đồng bộ với các DataRow trên máy trạm. Một ví dụ có thể là be 'INSERT (columns) INTO (table) WITH (primarykey)', sau đó là 'SELECT (columns) FROM (table) WHERE (primarykey)'. Các mẫu tin trả về có thể trộn vào tập các dòng.

None	Tất cả dữ liệu trả về từ câu lệnh đều bị vứt bỏ.
OutputParameters	Bất kì tham số xuất nào của câu lệnh đều được ánh xạ vào một cột thích hợp trong DataRow.

IX.1.2. Cập nhật một dòng đã có

Cập nhật một dòng có sẵn trong DataTable chỉ là một trường hợp việc sử dụng bộ chỉ mục của lớp DataRow với tên của một cột hoặc số thứ tự của cột, giống như ví dụ sau đây:

```
r["RegionDescription"]="North West England";
r[1] = "North East England";
```

Các hai câu lệnh đều cho cùng kết quả:

```
Changed RegionID 5 description
  1  Eastern                    Unchanged
  2  Western                    Unchanged
  3  Northern                   Unchanged
  4  Southern                   Unchanged
  5  North West England         Modified
```

Trong quá trình cập nhật cơ sở dữ liệu, trạng của dòng được cập nhật sẽ được gán là Modified.

IX.1.3. Xóa một dòng

Xóa một dòng là kết quả của việc gọi phương thức Delete():

```
r.Delete();
```

Một dòng được xóa có trạng thái là Deleted, nhưng bạn không thể đọc các cột từ một dòng đã xoá, nó không còn giá trị nữa. Khi gọi phương thức Update(), tất cả các dòng bị xoá sẽ sử dụng DeleteCommand, trong trường hợp này sẽ chạy stored procedure RegionDelete.

IX.2.Viết XML xuất

Như bạn đã thấy, DataSet hỗ trợ mạnh việc định nghĩa sơ đồ của nó trong một XML, và bạn có thể đọc dữ liệu từ một tài liệu XML, bạn cũng có thể viết một tài liệu XML.

Phương thức DataSet.WriteXml() cho phép bạn xuất các thành phần khác nhau của dữ liệu được lưu trong DataSet. Bạn có thể chọn chỉ xuất dữ liệu, hoặc dữ liệu và sơ đồ. Mã sau đây đưa ra một ví dụ của cho cả hai loại trên:

```
ds.WriteXml(".\\WithoutSchema.xml");
ds.WriteXml(".\\WithSchema.xml" , XmlWriteMode.WriteSchema);
```

File đầu tiên, WithoutSchema.xml được đưa ra dưới đây:

```
<?xml version="1.0" standalone="yes"?>
<NewDataSet>
  <Region>
    <RegionID>1</RegionID>
    <RegionDescription>Eastern                </RegionDescription>
  </Region>
  <Region>
    <RegionID>2</RegionID>
    <RegionDescription>Western                </RegionDescription>
  </Region>
```

```
  <Region>
    <RegionID>3</RegionID>
    <RegionDescription>Northern                </RegionDescription>
  </Region>
  <Region>
    <RegionID>4</RegionID>
    <RegionDescription>Southern                </RegionDescription>
  </Region>
</NewDataSet>
```

File WithSchema.xml bao gồm cả sơ đồ và dữ liệu của DataSet:

```
<?xml version="1.0" standalone="yes"?>
<NewDataSet>
  <xs:schema id="NewDataSet" xmlns=""
          xmlns:xs="http://www.w3.org/2001/XMLSchema"
          xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
    <xs:element name="NewDataSet" msdata:IsDataSet="true">
      <xs:complexType>
        <xs:choice maxOccurs="unbounded">
          <xs:element name="Region">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="RegionID"
                        msdata:AutoIncrement="true"
                        msdata:AutoIncrementSeed="1"
                        type="xs:int" />
                <xs:element name="RegionDescription"
                        type="xs:string" />
              </xs:sequence>
            </xs:complexType>
          </xs:element>
        </xs:choice>
      </xs:complexType>
    </xs:element>
  </xs:schema>
  <Region>
    <RegionID>1</RegionID>
    <RegionDescription>Eastern                </RegionDescription>
  </Region>
  <Region>
    <RegionID>2</RegionID>
    <RegionDescription>Western                </RegionDescription>
  </Region>
  <Region>
    <RegionID>3</RegionID>
    <RegionDescription>Northern                </RegionDescription>
  </Region>
```

```
  <Region>
    <RegionID>4</RegionID>
    <RegionDescription>Southern                </RegionDescription>
  </Region>
</NewDataSet>
```

Chú ý rằng, các thuộc tính mở rộng cho các cột trong một DataSet, như AutoIncrement và AutoIncrementSeed - những thuộc tính này tương ứgn với các thuộc tính có thể định nghĩa trong một DataColumn.

X. LÀM VIỆC VỚI ADO.NETàm việc với ADO.NET

Phần cuối cùng này sẽ cố gắng đưa ra nhưng kịch bản phổ biến khi phát triển các ứng dụng truy cập cơ sở dữ liệu với ADO.NET.

X.1.Phân tầng các ứng dụng

Việc sản xuất các phần mềm tương tác với dữ liệu thường chia ứng dụng thành nhiều tầng. Một mô hình phổ biến của một ứng dụng phân tầng là các dịch vụ dữ liệu phân tầng, và một cơ sở dữ liệu phân tầng.

Một trong những cái khó của mô hình này là việc phân tách dữ liệu giữa các tầng, và định dạng truyền giữa các tầng. ADO.NET đã giải quyết các vấn đề này và đã sớm hỗ trợ cho kiểu cấu trúc này.

Sao chép và trộn dữ liệu

Thật khó để copy một DB recordset? Trong In .NET thậy dễ dàng để sao chép một DataSet:

```
DataSet source = {some dataset};
DataSet dest = source.Copy();
```

Nó tạo một bản copy của DataSet nguồn – từng DataTable, DataColumn, DataRow, và Relation sẽ được sao chép y chan, và tất cả dữ liệu với các trạng thái trong file nguồn đều đươc sao chép. Nếu như bạn chỉ muốn sao chép sơ đồ của DataSet, bạn có thể làm như sau:

```
DataSet source = {some dataset};
DataSet dest = source.Clone();
```

Nó chỉ sao chép tất cả các table, relation, vân vân. Tất nhiên, DataTable sẽ rỗng.

Một thực tế phổ biến khi viết các hệ thống phân tầng, dựa trên Win32 hoặc web, là có truyền dữ liệu giữa các lớp càng ít càng tốt.

DataSet có phương thưc GetChanges() để giải quyết các yêu cầu này. Phương thức đơn giản này thực thi một loạt các công việc và trả về một DataSet với những dòng được cập nhật trong dataset nguồn. Đây là ý tưởng truyền dữ liệu giữa các tầng, chỉ một tập nhở dữ liệu được truyền.

Ví dụ sau chỉ ra cách tạo một "changes" DataSet:

```
DataSet source = {some dataset};
DataSet dest = source.GetChanges();
```

Bên dưới lớp vỏ bọc là rất nhiều thứ hấp dẫn. Có hai quá tải của phương thức GetChanges(). Một quá tải lấy giá trị của một DataRowState, và chỉ trả về các trạng thái tương ứng. GetChanges() đơn giản gọi GetChanges(Deleted | Modified | Added), và kiểm tra nếu để bảo đảm rằng có một vài thay đổi bằng cách gọi HasChanges(). Nếu không có thay đổi nào, một giá trị được trả về ngay lập tức.

Tiếp theo là sao chép DataSet. Trước tiên, một DataSet mới bỏ qua các ràng buộc (EnforceConstraints = false), sau đó mỗi dòng đã thay đổi được sao chép vào một DataSet mới.

Như vậy bạn có một DataSet chỉ chứa các thay đổi, sau đó bạn có thể truyền dữ liệu này qua các tầng để sử lí. Khi dữ liệu được cập nhật vào cơ sở dữ liệu, "changes" DataSet có thể trả về cho trình gọi (trong ví dụ này, một vài tham số xuất từ các stored procedure đã cập nhật trong các cột). Những thay đổi này có thể trộn vào bộ DataSet bằng cách dùng phương thức Merge(). Tiến trình này được mô tả như sau:

X.2. Tạo khoá với SQL Server

Stored procedure RegionInsert trong ví dụ ở phần trước đã từng tạo ra một giá trí khóa chính để chèn vào cơ sở dữ liệu. Phương thức tạo khoá đó còn thô sơ và không linh động, vì vậy một ứng dụng thực tế cần dùng đến các kĩ thật tạo khóa cao cấp hơn.

Đầu tiên có thể là định nghĩa một định dạng cột đơn giản, và trả về giá trị @@IDENTITY từ một stored procedure. Stored procedure dưới đây sử dụng bảng Categories trong cơ sở dữ liệu Northwind. Gõ stored procedure này vào SQL Query Analyzer, hoặc chạy the file StoredProcs.sql trong thư mục 13_SQLServerKeys:

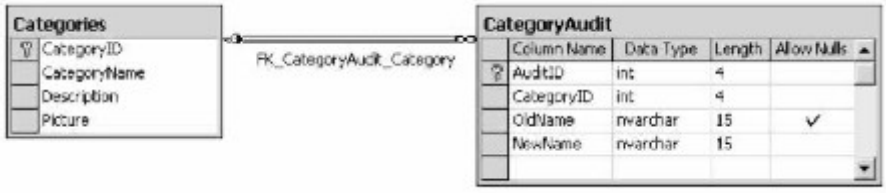
```
CREATE PROCEDURE CategoryInsert(@CategoryName NVARCHAR(15),
                    @Description NTEXT, @CategoryID INTEGER OUTPUT) AS
  SET NOCOUNT OFF
  INSERT INTO Categories (CategoryName, Description)
    VALUES(@CategoryName, @Description)
  SELECT @CategoryID = @@IDENTITY
GO
```

Nó chèn một dòng mới vào bảng Category, và trả về khóa chính cho trình gọi. Bạn có thể kiểm tra procedure này bằng cách gõ dòng SQL sau vào Query Analyzer:

```
DECLARE @CatID int;
EXECUTE CategoryInsert 'Pasties' , 'Heaven Sent Food' , @CatID OUTPUT;
PRINT @CatID;
```

Khi thực thi một bó lệnh, nó sẽ chèn mọt dòng mới vào bảng Categories, và trả về nhận dạng của dòng mới này, sau đó biểu diễn cho người dùng.

Giả sử rằng sau một vài tháng sử dụng, một ai đó muốn có một sổ theo dõi đơn giản, để báo cáo những cập nhật và sửa đổi trên category name. Bạn sẽ định nghĩa một bảng như sau, để chỉ ra các giá trị mới và cũ của category:



Mã sẵn có trong StoredProcs.sql. Cột AuditID được định nghĩa như một cột IDENTITY. Sau đó bạn cấu trúc một cặp trigger để báo cáo các thay đổi trên trường CategoryName:

```
CREATE TRIGGER CategoryInsertTrigger
  ON Categories
  AFTER UPDATE
AS
  INSERT INTO CategoryAudit(CategoryID , OldName , NewName )
    SELECT old.CategoryID, old.CategoryName, new.CategoryName
    FROM Deleted AS old,
       Categories AS new
    WHERE old.CategoryID = new.CategoryID;
GO
```

Bạn phải dùng Oracle stored procedure, SQL Server không hỗ trợ nội dung OLD và NEW của các dòng, thay vì chèn một trigger nó có một bộ bảng trong bộ nhớ gọi là Inserted, để xóa và cập nhật, các dòng cũ tồn tại trong bảng Deleted.

Trigger này nhận CategoryID cho các cột giả và lưu các giá trị cũ và mới của cột CategoryName.

Giờ đây, khi bạn gọi một stored procedure để chèn một CategoryID mới, bạn nhận một giá trị nhận dạng; Dĩ nhiên, nó không còn là giá trị nhận của dòng được chèn vào bảng Categories, nó là một giá trị mới được tạo trong bảng CategoryAudit. Ouch!

Để xem vấn đề, mở SQL Server Enterprise manager, xem nội dung của bảng Categories table.

	CategoryID	CategoryName	Description
	1	Beverages	Soft drinks, coffees, teas, beers, and ales
	2	Condiments	Sweet and savory sauces, relishes, spreads, and seasonings
	3	Confections	Desserts, candies, and sweet breads
	4	Dairy Products	Cheeses
	5	Grains/Cereals	Breads, crackers, pasta, and cereal
	6	Meat/Poultry	Prepared meats
	7	Produce	Dried fruit and bean curd
	8	Seafood	Seaweed and fish
▶	20	Pasties	Heaven Sent Grub
*			

Bảng này liệt kê tất cả categories tôi có trong thể hiện của cơ sở dữ liệu.

Giá trị nhận dạng tiếp theo cho bảng Categories có thể là 21, vì vậy chúng ta sẽ chèn một dòng mới bằng cách thực thi mã sau đây, và xem nó trả về ID nào:

```
DECLARE @CatID int;
EXECUTE CategoryInsert 'Pasties' , 'Heaven Sent Food' , @CatID OUTPUT;
PRINT @CatID;
```

Giá trị trả về trên máy của tôi là 17. Khi xem bảng CategoryAudit, tôi nhận ra rằng đó là nhận dạng của dòng mới chèn trong bảng audit, không phải của category.

	AuditID	CategoryID	OldName	NewName
▶	17	30	<NULL>	Vegetables
*				

Đó là vì @@IDENTITY trả về giá trị nhận dạng cuối.

Có hai nhận dạng cơ bản bạn có thể sử dụng thay cho @@IDENTITY, chúng cũng không thể giải quyết vấn đề trên. Đầu tiên là SCOPE_IDENTITY(), sẽ trả về giá trị nhận dạng cuối cùng trong tầm vực hiện tại. SQL Server định nghĩa tầm vực như như một stored procedure, trigger, hoặc hàm. Nếu vì một ai đó thêm

một câu lệnh INSERT khác vào stored procedure, thì bạn sẽ nhận một giá trị không mong chờ.

IDENT_CURRENT() sẽ trả về giá trị nhận dạng cuối cùng được phát ra trên một bảng trong bất cứ tầm vực nào, trong trường hợp này, nếu hai người dùng đang truy cập SQL Server cùng một lúc, nó có thể nhận giá trị của người khác.

Chỉ có cách quản lí thủ công, bằng cách dùng cột IDENTITY trong SQL Server.

X.3.Qui tắt đặt tên

Trong nhiều năm làm việc với các ứng dụng cơ sở dữ liệu, Tôi nhận được một vài giới thiệu cho cách đặt tên để tiện cho việc dùng chung. Tôi biết nó không liên quan đến .NET, nhưng những qui tắt này rất hữu ích khi đặt tên. Bỏ qua phần này nếu bạn có cách đặt tên riêng của mình.

Database Tables

Luôn là số ít – Product tốt hơn là Products. Nó sẽ tốt hơn về mặt ngữ pháp khi nói "The Product table contains products" hơn là "The Products table contains products". Hãy xem cơ sở dữ liệu Northwind để thấy được nhận xét này.

Chấp nhận một vài qui tắt đặt tên cho các cột trong một bảng – chẳng hạn <Table>_ID cho khóa chính của bảng (Ở đây khóa chính là một cột), tên của cột phải là một tên thân thiện, và giải thích chứa đựng thông tin về cột đó. Một qui tắc đặt tên tốt sẽ cho bạn một cái nhìn bao quát vè khả năng của các trường trong cơ sở dữ liệu.

Database Columns

Dùng danh từ số ít tốt hơn là danh từ số nhiều.

Bất kì cột nào liên kết với bảng khác nên được đặt cùng tên với khóa chính của bảng kia. Chẳng hạn, một liên kết với bảng Product là Product_ID, và đến bảng Sample là Sample_ID. Không phải lúc nào cũng vậy, chẳng hạn một bảng có nhiều liên kết với bảng khác. Trong trường hợp này tùy bạn sử dụng.

Các trường Date nên kết thúc bằng _On, chẳng hạn Modified_On, Created_On. Như vậy sẽ dễ hiểu hơn.

Các trường báo cao nêu kết thúc bằng _By, chẳng hạn Modified_By hay Created_By.

Constraints

Nếu có thể, nên bao gồm tên của bảng và cột của ràng buộc, chẳng hạn CK_<Table>_<Field>. Ví dụ CK_PERSON_SEX để kiểm tra ràng buộc trên cột SEX của bảng PERSON. Một khóa ngoại có thể là FK_Product_Supplier_ID, đây là khóa ngoại giữa product và supplier.

Chỉ ra kiểu của ràng buộc như một tiếp đầu ngữ, chẳng hạn CK cho một kiểm tra ràng buộc và FK cho một khóa ngoại. Chẳng hạn CK_PERSON_AGE_GT0 cho một ràng buộc trên cột age khai báo rằng age phải lớn hơn zero.

Nếu bạn muốn rút gọn tên của ràng buộc, nên làm điều đó trên tên của bảng hơn là tên của cột. Khi bạn có một ràng buộc vi phạm, nó sẽ dễ dàng nhận ra lỗi xảy ra trên bảng nào, nhưng không dễ kiểm tra xem trường nào đã sinh lỗi. Oracle giới hạn tên là 30-kí tự.

Stored Procedures

Nhiều nhà phát triển SQL Server nhận thấy dùng tiếp đầu ngữ 'sp_' là qui tắc tốt nhất để đặt tên cho các stored procedure.

SQL Server dùng tiếp đầu ngữ 'sp_' cho tất cả các stored procedure hệ thống. Vì vậy, có thể sẽ xảy ra tình trạng xung đột tên khi các stored procedure của bạn cũng bắt đầu là 'sp_' như 'sp_widget' giống với stored procedure chuẩn của SQL Server. Khi xem xét một stored procedure, SQL Server sẽ sửa các procedure với tiếp đầu ngữ 'sp_'.

Nêu bạn dùng tiếp đầu ngữ này, khi thực thi, SQL Server sẽ xem tầm vực hiện tại, và nhảy đến cơ sở dữ liệu chủ để tìm stored procedure đó. Nếu có tồn tại thì sẽ có một lỗi phát sinh sớm.

X.4. Performance

Bộ managed provider hiện tại của .NET có một vài giới hạn – bạn có thể chọn OleDb hoặc SqlClient; OleDb cho phép kết nối với bất kì nguồn dữ liệu nào nếu nó là một OLE DB driver (chẳng hạn như Oracle), còn SqlClient là một trình cung câp dùng riêng cho SqlServer.

Trình cung cấp SqlClient đã được viết hoàn toàn bằng mã có quản, và sử dụng một vài lớp để kết nối cơ sở dữ liệu. Trình cung cấp viết các gói **TDS** (**Tabular Data Stream**) trực tiếp từ SQL Server, về bản chất nó nhanh hơn OleDb provider, nó có thể duyệt qua các lớp trước khi tác động vào cơ sở dữ liệu.

Để kiểm tra điều đó, hãy chạy mã sau trên cùng cơ sở dữ liệu, khác biệt ở chỗ sử dụng SqlClient managed provider trên ADO provider:

```
SqlConnection conn = new SqlConnection(Login.Connection);
conn.Open();
SqlCommand cmd = new SqlCommand ( "update tempdata set AValue=1 Where ID=1" , conn);
DateTime   initial, elapsed ;
initial = DateTime.Now ;
for(int i = 0; i < iterations; i++)
  cmd.ExecuteNonQuery();
elapsed = DateTime.Now ;
conn.Close();
```

OLE DB thường sử dụng OleDbCommand hơn là SqlCommand. Tôi đã tạo một bảng cơ sở dữ liệu nhỏ với hai côtj như dưới đây, và điền vào đó một dòng đơn:

Câu lệnh SQL được sử dụng là một câu lệnh UPDATE đơn giản:

UPDATE TempData SET AValue = 1 WHERE ID = 1.

SQL là đơn giản nhất trong các provider. Kết quả tính bằng giây cho được liệt kê trong bảng sau:

Provider	**100**	**1000**	**10000**	**50000**
OleDb	0.109	0.798	7.95	39.11
Sql	0.078	0.626	6.23	29.27

Nếu bạn chỉ hướng vào SQL Server thì dĩ nhiên bạn sẽ chọn Sql provider. Trong thực tế, nếu bạn sử dụng các cơ sở dữ liệu khác bạn sẽ sử dụng OleDb provider.

Microsoft đã tạo ra một giao thức tri cập chung cho các cơ sở dữ liệu khác nhau trong System.Data.Các lớp chung này sẽ dùng cơ sở dữ liệu thích hợp vào thời gian chạy. Nó là một lớp vở bọc giữa OleDb và Sql, nếu các nhà cung cấp cơ sở dữ liệu khác viết các trình quản lí cho các sản phẩm của họ, bạn có thể dùng ADO cho provider đó với một ít thay đổi ở mã. Ví dụ trong truy cập cơ sở dữ liệu .NET, "Scientific Data Center" trong "*Data-Centric .NET Programming with C#*" (Wrox Press, ISBN 1-861005-92-x) giải thích việc sử dụng C# để truy cập cơ sở dữ liệu MySQL.

B. CÂU HỎI VÀ BÀI TẬP

Câu 1: Thực hiện kết nối đến một cơ sở dữ liệu có sẵn.

Câu 2: Thực hiện đọc từng dòng dữ liệu trong table trên cơ sở dữ liệu đưa lên đối tượng hiển thị (chẳn hạn combobox).

Câu 3: Kết nối và truy xuất dữ liệu với đối tượng datagridview.

MÃ BÀI HỌC LTTQ – 09	BÀI 9 : XÂY DỰNG ỨNG DỤNG TỔNG HỢP	Thời gian (giờ)				
		LT 2	TH 10	BT 5	KT 1	TS 18

Mục tiêu:

Sau khi học xong bài này, học viên có khả năng:

- Xác định yêu cầu của các ứng dụng;
- Phân tích bài toán thực tế;
- Xây dựng được các phần mềm ứng dụng dựa trên ngôn ngữ lập trình C#.
- Tổ chức làm việc nhóm;
- Tự phát triển công nghệ lập trình;
- Thực hiện các thao tác an toàn với máy tính;

TÓM TẮT BÀI:

- Phân tích và tổng hợp các bài toán thực tế.
- Vận dụng ngôn ngữ lập trình C# để xây dựng một ứng dụng vào thực tế.
- Gỡ rối các chương trình xây dựng bằng ngôn ngữ lập trình C#.

Các vấn đề chính sẽ được đề cập

- Xây dựng các chương trình liên quan đến Cấu trúc dữ liệu
- Xây dựng các chương trình liên quan đến cơ sở dữ liệu về việc quản lý thông tin, CSDL.
- Xây dựng các chương trình liên quan đến quản trị hệ thống mạng.
- Xây dựng các chương trình liên quan đến đồ hoạ, xử lý ảnh, xử lý đa phương tiện.
- Xây dựng các chương trình liên quan đến quản lý File.

NỘI DUNG

Bài 1: Xây dựng chương trình ứng dụng "Xổ số điện toán"

Yêu cầu: - Giao diện đẹp, dễ sử dụng.

- Đảm bảo tính khách quan khi quay số.

- Có các lựa chọn, như quay từng giải (với thời gian quy định xuất hiện từng số), quay một lần, giá vé, cơ cấu giải thưởng, trị giá giải thưởng…)

Bài 2: Xây dựng chương trình ứng dụng "máy tính tay"

Yêu cầu: - Giao diện đẹp, dễ sử dụng.

- Thực hiện các chức năng tính toán tối thiểu cơ bản, như: cộng, trừ, nhân, chia, phần trăm, lũy thừa, nhớ. Chú ý tam khảo giao diện của ứng dụng calc của windows)

Bài 3: Xây dựng chương trình ứng dụng "MiniWord" (phần mềm soạn thảo văn bản)

Yêu cầu: - Giao diện đẹp, dễ sử dụng.

- Đảm bảo có thể sử dụng để soạn văn bản, gồm:

+ Hệ thống thực đơn.

+ Hệ thống thanh công cụ.

+ Hỗ trợ phím nóng.

Bài 4: Xây dựng chương trình ứng dụng "Quản lý điểm"

Yêu cầu: - Giao diện đẹp, dễ sử dụng.

- Cơ sở dữ liệu MS Access hoặc SQL server.

- Đảm bảo các chức năng, gồm;

+ Đăng ký, đăng nhập.

+ Nhập điểm, xem điểm.

+ Lọc theo các điều kiện: sinh viên chưa thi, thi thiếu điểm…

+ Tìm kiếm theo mã sinh viên, tên sinh viên hoặc theo khoa, lớp.

CÁC THUẬT NGỮ CHUYÊN MÔN

Số TT	Thuật ngữ	Giải thích
1.	Absolute	Tuyệt đối
2.	Abstract	Trừu tượng
3.	Access	Truy xuất
4.	Access modifier	Từ khóa thay đổi tầm truy xuất
5.	Accessibility	Khả năng truy xuất
6.	Account	Tài khoản
7.	ACL [Access Control List]	Danh sách điều khiển truy xuất
8.	Administrator	Người quản trị
9.	Aggregate function	Hàm tập hợp
10.	Algorithm	Giải thuật
11.	API [Application Programming Interface]	Giao diện lập trình ứng dụng
12.	Application	Ứng dụng
13.	Application domain	Miền ứng dụng
14.	Argument	Đối số
15.	Arithmetic	Số học
16.	Array	Mảng
17.	Assembly	Gói kết hợp
18.	Asynchronous	Bất đồng bộ
19.	Attribute	Đặc tính
20.	Authentication	Sự xác thực
21.	Authorization	Sự phân quyền
22.	Availability	Tính khả dụng
23.	Binary	Nhị phân
24.	Block	Khối
25.	Bound	Cận
26.	Boundary	Đường biên / Ranh giới
27.	Breakpoint	Điểm dừng
28.	Browser	Trình duyệt
29.	Buffer	Bộ đệm
30.	Built-in	Nội tại
31.	Cache	Kho chứa (truy xuất nhanh)
32.	Caching	Cơ chế lưu giữ
33.	CAS [Code Access Security]	Bảo mật truy xuất mã lệnh
34.	Certificate	Chứng chỉ
35.	Channel	Kênh
36.	Character	Ký tự
37.	Class	Lớp
38.	Client	Trình khách
39.	Clone [v]	Sao chép

40.	Cloneable [adj]	Khả sao chép
41.	CLR [Common Language Runtime]	Bộ thực thi ngôn ngữ chung
42.	Code	Mã lệnh
43.	Collection	Tập hợp
44.	Column	Cột
45.	Command	Lệnh
46.	Communication	Sự giao tiếp
47.	Comparable	Khả so sánh
48.	Compare	So sánh
49.	Compatibility	Tính tương thích
50.	Compile	Biên dịch
51.	Compiler	Trình biên dịch
52.	Component	Thành phần
53.	Component tray	Khay thành phần
54.	Configuration	Cấu hình
55.	Connection	Kết nối
56.	Constant	Hằng
57.	Constructor	Phương thức khởi dựng
58.	Context	Ngữ cảnh
59.	Context-sensitive help	Trợ giúp cảm-ngữ-cảnh
60.	Control	Điều kiểm
61.	Convert	Chuyển đổi
62.	Convertible	Khả chuyển đổi
63.	Cryptography	Mật mã
64.	Culture	Miền văn hóa
65.	Custom	Tùy biến
66.	Data	Dữ liệu
67.	Data binding Kỹ thuật kết dữ liệu	Database Cơ sở dữ liệu
68.	De-compile Dịch ngược	De-serialize Giải tuần tự hóa
69.	Decrypt Giải mật hóa	Decryption Sự giải mật hóa
70.	Debug Gỡ rối	DebuggerTrình gỡ rối
71.	Default Mặc định	Delegate Ủy nhiệm hàm
72.	Deploy Triển khai	Destructor Phương thức hủy

73.	Device Thiết bị	Derive Dẫn xuất
74.	Data binding Kỹ thuật kết dữ liệu	Database Cơ sở dữ liệu
75.	Dictionary	Từ điển
76.	Digital signature	Chữ ký số
77.	Directive	Chỉ thị
78.	Directory	Thư mục
79.	Disposable	Khả hủy
80.	Dispose	Hủy
81.	Distributed	Có tính phân tán
82.	Document	Tài liệu
83.	Domain	Miền
84.	Edit	Hiệu chỉnh
85.	Editor	Trình soạn thảo
86.	Encapsulation	Sự đóng gói
87.	Encode	Mã hóa
88.	Encoding	Phép mã hóa
89.	Encrypt	Mật hóa
90.	Encryption	Sự mật hóa
91.	Entry	Khoản mục
92.	Enumeration	Kiểu liệt kê
93.	Environment	Môi trường
94.	Error	Lỗi
95.	Event	Sự kiện
96.	Event hander	Phương thức thụ lý sự kiện
97.	Event log	Nhật ký sự kiện
98.	Evidence	Chứng cứ
99.	Exception	Biệt lệ
100.	Exception hander	Phương thức thụ lý biệt lệ
101.	Expiration	Sự hết hiệu lực
102.	Export	Xuất
103.	Expression	Biểu thức
104.	Feature	Tính năng
105.	Field	Trường
106.	File	Tập tin

107.	Filter	Bộ lọc
108.	Flag	Cờ
109.	Flexibility	Tính linh hoạt
110.	Form	Biểu mẫu
111.	Format	Định dạng
112.	FTP [File Transfer Protocol]	Giao thức truyền file
113.	Function	Hàm
114.	Functionality	Chức năng
115.	GAC [Global Assembly Cache]	Kho chứa gói kết hợp toàn cục
116.	GC [Garbage Collector]	Bộ thu gom rác
117.	Generalization	Tính tổng quát hóa
118.	Global	Toàn cục
119.	Globalization	Sự toàn cầu hóa
120.	Graphics	Đồ họa
121.	Group	Nhóm
122.	GUI [Graphical User Interface]	Giao diện người dùng đồ họa
123.	GUID [Globally Unique Identifier]	Định danh duy nhất toàn cục
124.	Handle	Mục quản
125.	Hash	Băm
126.	Hash code	Mã băm
127.	Hashtable	Bảng băm
128.	Help	Trợ giúp
129	HTML [HyperText Markup Language]	Ngôn ngữ đánh dấu siêu văn bản

130.	Hyperlink	Siêu kiên kết
131.	IDE [Integrated Development Environment]	Môi trường phát triển tích hợp
132.	Identifier	Điện từ
133.	Imperson	Giả nhận
134.	Impersonation	Sự giả nhận
135.	Implement	Hiện thực
136.	Implementation	Bản hiện thực
137.	Import	Nhập
138.	Index	Chỉ mục
139.	Indexer	Bộ chỉ mục
140.	Inheritance	Sự thừa kế
141.	Initialize	Khởi tạo
142.	Input	Đầu vào / Dữ liệu nhập
143.	Insert	Chèn
144.	Install	Cài đặt
145.	Instance	Thể hiện
146.	Integration	Sự tích hợp
147.	Interface	Giao diện
148.	Interoperability	Khả năng liên tác
149.	IP [Internet Protocol]	Giao thức Internet
150.	Item	Mục chọn
151.	JIT [just-in-time]	Tức thời / Vừa đúng lúc

152.	Key	Khóa
153.	Keyword	Từ khóa
154.	Language	Ngôn ngữ
155.	Length	Chiều dài
156.	Library	Thư viện
157.	Lifetime	Thời gian sống
158.	Link	Liên kết
159.	List	Danh sách
160.	Literal	Trực kiện
161.	Load	Nạp
162.	Local	Cục bộ
163.	Locale	Bản địa
164.	Localization	Sự bản địa hóa
165.	Lock	Chốt
166.	Logic	Mã thi hành chức năng
167.	Loop	Vòng lặp
168.	Managed	Được quản lý
169.	Management	Sự quản lý
170.	Mapping	Phép ánh xạ
171.	Member	Thành viên
172.	Membership	Tư cách thành viên
173.	Memory	Bộ nhớ
174	Menu	Trình đơn

175.	Message	Thông điệp
176.	Metacharacter	Siêu ký tự
177.	Metadata	Siêu dữ liệu
178.	Method	Phương thức
179.	Model	Mô hình
180.	Module	Đơn thể
181.	MSIL [Microsoft Intermediate Language]	Ngôn ngữ trung gian
182.	Multilingual	Đa ngôn ngữ
183.	Multithreading	Lập trình đa tiểu trình
184.	Native	Nguyên sinh
185.	Namespace	Không gian tên
186.	Network	Mạng
187.	Node	Nút
188.	Object	Đối tượng
189.	Object-oriented programming	Lập trình hướng đối tượng
190.	Operating system	Hệ điều hành
191.	Operator	Toán tử
192.	Output	Xuất ra / Kết xuất
193.	Overload	Bản nạp chồng
194.	Override	Chép đè
195.	Parameter	Thông số
196.	Password	Mật khẩu

197.	Path	Đường dẫn
198.	Pattern	Kiểu mẫu
199.	Performance	Hiệu năng
200.	Permission	Quyền
201.	Pixel	Điểm ảnh
202.	Platform	Nền
203.	Pointer	Con trỏ
204.	Policy	Chính sách
205.	Polymorphisme	Tính đa hình
206.	Pool	Kho dự trữ
207.	Pooling	Cơ chế dự trữ
208.	POP3 [Post Office Protocol 3]	Giao thức nhận mail 3
209.	Port	Cổng
210.	Postfix	Hậu tố
211.	Prefix	Tiền tố
212.	Private	Riêng
213.	Privilege	Đặc quyền
214.	Procedure	Thủ tục
215.	Process	Tiến trình
216.	Processor	Bộ xử lý
217.	Project	Dự án
218.	Property	Thuộc tính
219	Protected	Được bảo vệ

220.	Protocol	Giao thức
221.	Public	Công khai
222.	Query	Truy vấn
223.	Queue	Hàng đợi
224.	Random	Ngẫu nhiên
225.	RBS [Role-Based Security]	Bảo mật dựa-trên-vai-trò
226.	Record	Bản ghi / Mẩu tin
227.	Recursion	Sự đệ quy
228.	Reference	Tham chiếu
229.	Reflection	Cơ chế phản chiếu
230.	Register	Đăng ký
231.	Regular expression	Biểu thức chính quy
232.	Relationship	Mối quan hệ
233.	Relative	Tương đối
234.	Remotable	Khả truy xuất từ xa
235.	Resource	Tài nguyên
236.	Reusability	Khả năng tái sử dụng
237.	Role	Vai trò
238.	Routine	Thường trình
239.	Row	Hàng / Dòng
240.	Runtime	Bộ thực thi
241.	Schema	Lược đồ / Khuôn

242.	Script	Kịch bản
243.	Security	Sự bảo mật
244.	Serialize	Tuần tự hóa
245.	Serializable	Khả tuần tự hóa
246.	Serialization	Sự tuần tự hóa
247.	Server	Trình chủ
248.	Service	Dịch vụ
249.	Session	Phiên làm việc
250.	Setting	Thiết lập
251.	Shared	Được chia sẻ / Dùng chung
252.	Signature	Chữ ký
253.	SMTP [Simple Mail Transfer Protocol]	Giao thức truyền mail đơn giản
254.	SOAP [Simple Object Access Protocol]	Giao thức truy xuất đối tượng đơn giản
255.	Solution	Giải pháp
256.	Specialization	Tính chuyên hóa
257.	SQL [Structured Query Language]	Ngôn ngữ truy vấn có cấu trúc
258.	Stack	Ngăn chồng
259.	State	Trạng thái
260.	State | Stateless	Có trạng thái | Phi trạng thái
261.	Statement	Câu lệnh / Khai báo
262.	Static	Tĩnh
263.	Stored procedure	Thủ tục tồn trữ
264	Stream	Dòng chảy

265.	String	Chuỗi
266.	Strong name	Tên mạnh
267.	Strong type	Kiểu mạnh
268.	Strongly-named	Được định tên mạnh
269.	Strongly-typed	Được định kiểu mạnh
270.	Structure	Cấu trúc
271.	Symmetric	Đối xứng
272.	Synchronization	Sự đồng bộ hóa
273.	Synchronous	Đồng bộ
274.	System	Hệ thống
275.	System tray	Khay hệ thống
276.	Table	Bảng
277.	Tag	Thẻ
278.	Task	Tác vụ
279.	Template	Khuôn mẫu
280.	Thread	Tiểu trình / Mạch trình / Tuyến đoạn
281.	Thread-safe	An toàn về tiểu trình
282.	Throw	Ném
283.	Timestamp	Tem thời gian
284.	Tool	Công cụ
285.	Toolbox	Hộp công cụ
286.	Transaction	Phiên giao dịch

287.	Type	Kiểu
288.	Type-safe	Antoàn về kiểu dữ liệu
289.	Unmanaged	Không được quản lý
290.	Update	Cập nhật
291.	URI [Uniform Resource Identifier]	Bộ nhận dạng tài nguyên đồng dạng
292.	URL [Uniform Resource Locator]	Bộ định vị tài nguyên đồng dạng
293.	User	Người dùng
294.	Utility	Tiện ích
295.	Validation	Sự xác nhận tính hợp lệ
296.	Value	Giá trị
297.	Variable	Biến
298.	Version	Phiên bản
299.	Virtual	Áo
300.	Visible	Khả kiến
301.	Visual	Trực quan
302.	Wildcard	Ký tự đại diện
303.	Window	Cửa sổ
304.	Worker	Thợ
305.	Wrapper	Vỏ bọc
306.	WSDL [Web Services Description Language]	Ngôn ngữ mô tả dịch vụ Web
307.	XML [Extensible Markup Language]	Ngôn ngữ đánh dấu mở rộng

TÀI LIỆU THAM KHẢO

[1]. Nguyễn Văn Lân, *Lập Trình Cơ Sở Dữ Liệu Với C# - Mô Hình Nhiều Tầng* , Nhà xuất bản Lao động Xã hội, 2009

[2]. Phạm Hữu Khang, *Lập trình ứng dụng chuyên nghiệp SQL Server 2000*, Nhà xuất bản Giáo dục, 2002.

[3]. Phạm Hữu Khang, *C# 2008* (tập 1->6), Nhà xuất bản Lao động Xã hội, 2009.

[4]. Richard Blum, *C# Network Programming*, Joel Fugazzotto, 2003.

[5]. Fiach Reid, *Network Programming in .NET*, Donegal-Ireland, 2004

[6]. Website : http: //codeproject.com

DANH SÁCH BAN BIÊN SOẠN GIÁO TRÌNH DẠY NGHỀ
TRÌNH ĐỘ TRUNG CẤP, CAO ĐẲNG

(font chữ Times New Roman, in hoa, cỡ chữ 14 Bold)

Tên giáo trình: *(font chữ Times New Roman, cỡ chữ 14, Bold)*

Tên nghề: *(font chữ Times New Roman, cỡ chữ 14, Bold)*

1. Ông (bà).....	Chủ nhiệm
2. Ông (bà).....	Phó chủ nhiệm
3. Ông (bà).....	Thư ký
4. Ông (bà).....	Thành viên
5. Ông(bà).....	Thành viên
6. Ông(bà).....	Thành viên
7. Ông(bà).....	Thành viên
8. Ông(bà).....	Thành viên
9. Ông(bà).....	Thành viên

DANH SÁCH HỘI ĐỒNG NGHIỆM THU
GIÁO TRÌNH DẠY NGHỀ TRÌNH ĐỘ TRUNG CẤP, CAO ĐẲNG

(font chữ Times New Roman, in hoa, cỡ chữ 14 Bold)

1. Ông (bà).....	Chủ tịch
2. Ông (bà).....	Phó chủ tịch
3. Ông (bà).....	Thư ký
4. Ông (bà).....	Thành viên
5. Ông(bà).....	Thành viên
6. Ông(bà).....	Thành viên
7. Ông(bà).....	Thành viên
8. Ông(bà).....	Thành viên
9. Ông(bà).....	Thành viên