

TRƯỜNG ĐẠI HỌC DUY TÂN
KHOA CÔNG NGHỆ THÔNG TIN



TẬP BÀI GIẢNG

Môn học: **LẬP TRÌNH ỨNG DỤNG.NET**

Mã môn học: **CS 464**

Số tín chỉ: **03**

Lý thuyết: **02**

Thực hành: **01**

Dành cho sinh viên ngành: **CÔNG NGHỆ PHẦN MỀM**

Khoa: **CÔNG NGHỆ THÔNG TIN**

Bậc đào tạo: **Đại học**

Học kỳ: 2

Năm học: 2021 – 2022

Đà Nẵng, tháng 03 năm 2022

PHÂN BỐ THỜI GIAN GIẢNG DẠY

GIỜ THỨ	NỘI DUNG	TRANG
0 – 1	Microsoft .NET và C#	01 – 11
2 – 4	Cơ bản về C#.NET	12 – 78
5 – 10	Lập trình hướng đối tượng với C#	79 – 214
11 – 20	Windows Form	215 – 264
21 – 28	Truy xuất cơ sở dữ liệu	265 – 323
29 – 30	ÔN TẬP	

MỤC LỤC

CHƯƠNG 1: MICROSOFT .NET	1
1.1. Tổng quan về Microsoft.Net	1
1.1.1. .NET là gì	1
1.1.2. Các lĩnh vực của .NET	2
1.2. Tổng quan về .NET Framework	3
1.2.1. .NET Framework là gì?	3
1.2.2. Lịch sử phát triển của .NET Framework	4
1.2.3. Thành phần .NET Framework	7
CHƯƠNG 2: LẬP TRÌNH C# CƠ BẢN	12
2.1. Tại sao phải sử dụng ngôn ngữ C#	12
2.1.1. C# là ngôn ngữ đơn giản	12
2.1.2. C# là ngôn ngữ hiện đại	12
2.1.3. C# là ngôn ngữ hướng đối tượng	12
2.1.4. C# là ngôn ngữ mạnh mẽ và mềm dẻo	13
2.1.5. C# là ngôn ngữ ít từ khóa	13
2.1.6. C# là ngôn ngữ module hóa	14
2.1.7. C# đã và đang trở nên phổ biến	14
2.1.8. Ngôn ngữ C# với những ngôn ngữ khác	14
2.2. Các bước chuẩn bị cho chương trình C#	16
2.2.1. Thực hiện ứng dụng Console C#.Net đơn giản	16
2.2.2. Phát triển chương trình minh họa C#	20
2.3. Kiểu dữ liệu	22
2.4. Biến (Variable)	41
2.5. Hằng	42
2.6. Biểu thức và toán tử	44
2.6.1. Biểu thức (Expression)	44
2.6.2. Toán tử (Operator)	44
2.7. Lệnh (Statement)	52

2.7.1.	Lệnh đơn	52
2.7.2.	Khối lệnh.....	52
2.7.3.	Cấu trúc rẽ nhánh.....	52
2.7.4.	Cấu trúc lặp	58
2.7.5.	Các kiểu lệnh khác.....	64
2.8.	Ngoại lệ và xử lý ngoại lệ.....	65
2.8.1.	Lệnh try ...catch ...finally	65
2.8.2.	Lệnh throw	68
2.8.3.	Các lớp ngoại lệ.....	69
2.9.	Lớp System.Console.....	70
2.9.1.	Định dạng kết xuất.....	70
2.9.2.	Nhập và xuất với lớp Console	71
2.9.3.	Định dạng kết xuất cho phương thức Write.....	72
CHƯƠNG 3: LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG VỚI C#		79
3.1.	Xây dựng lớp, đối tượng.....	80
3.1.1.	Định nghĩa lớp.....	80
3.1.2.	Tạo đối tượng	86
3.1.3.	Sử dụng các thành viên tĩnh	94
3.1.4.	Hủy đối tượng.....	100
3.1.5.	Truyền tham số	103
3.1.6.	Nạp chồng phương thức.....	111
3.1.7.	Đóng gói dữ liệu với thành phần thuộc tính	115
3.1.8.	Thuộc tính chỉ đọc	119
3.2.	Ké thừa – Đa hình.....	121
3.2.1.	Đặc biệt hóa và tổng quát hóa	121
3.2.2.	Sự kế thừa.....	124
3.2.3.	Đa hình.....	129
3.2.4.	Lớp trùu tượng	138
3.2.5.	Các lớp lồng nhau.....	145
3.3.	Nạp chồng toán tử	148

3.3.1.	Sử dụng toán tử Operator.....	148
3.3.2.	Hỗ trợ ngôn ngữ .Net khác.....	149
3.3.3.	Sử dụng toán tử	150
3.3.4.	Toán tử so sánh bằng	152
3.3.5.	Toán tử chuyển đổi	153
3.4.	Cấu trúc.....	161
3.4.1.	Định nghĩa một cấu trúc.....	162
3.4.2.	Tạo cấu trúc	165
3.5.	Thực thi giao diện.....	171
3.5.1.	Thực thi một giao diện.....	171
3.5.2.	Truy cập phương thức giao diện	185
3.5.3.	Thực thi phủ quyết giao diện	194
3.5.4.	Thực thi giao diện tường minh	199
3.5.5.	Lựa chọn thẻ hiện phương thức giao diện	203
3.5.6.	Ân thành viên	204
CHƯƠNG 4: WINDOWS FORM	216
4.1.	Bắt đầu với Windows form.....	216
4.1.1.	Ứng dụng Windows Form đơn giản	216
4.1.2.	Tạo đối tượng của lớp điều khiển.....	219
4.1.3.	Tập các đối tượng điều khiển con (Controls collection)	220
4.1.4.	Đặt, xóa một đối tượng điều khiển trên Form hay điều khiển khác.....	221
4.1.5.	Thừa kế lớp Form	221
4.1.6.	Truy cập các đối tượng điều khiển	221
4.1.7.	Thành phần (Components).....	222
4.2.	Lớp Control	223
4.2.1.	Các thành viên cơ sở của lớp Control.....	223
4.2.2.	Phím truy cập	224
4.2.3.	Focus và thứ tự Tab	224
4.3.	Xử lý sự kiện	225
4.3.1.	Sự kiện là gì.....	225

4.3.2.	Các sự kiện thường dùng	225
4.3.3.	Thêm một sự kiện vào phương thức.....	226
4.4.	Form.....	226
4.4.1.	Form và các thành viên cơ sở của lớp Form.....	227
4.4.2.	Thiết lập vị trí Form.....	229
4.4.3.	Cuộn Form	230
4.4.4.	Tạo cửa sổ hộp đối thoại	230
4.4.5.	Form chủ (form ownership)	232
4.4.6.	Làm việc với ứng dụng MDI.....	232
4.4.7.	Form định nghĩa trước	233
4.4.8.	Form đa ngôn ngữ.....	234
4.5.	Các điều khiển cơ bản.....	236
4.5.1.	Nhãn (Label)	236
4.5.2.	Nhãn liên kết (LinkLabel).....	236
4.5.3.	Nút lệnh (Button).....	237
4.5.4.	Hộp văn bản (Textbox)	237
4.5.5.	Hộp đánh dấu (CheckBox) và nút lựa chọn (RadioButton)	238
4.5.6.	PictureBox.....	239
4.5.7.	Các điều khiển danh sách (List Controls)	239
4.5.8.	Điều khiển font	241
4.6.	Các điều khiển chuyên biệt	241
4.6.1.	Điều khiển NotifyIcon	241
4.6.2.	Trình đơn (MenuStrip), thanh trạng thái (StatusStrip), thanh công cụ (ToolStrip)	242
4.6.3.	Các điều khiển ngày (Date Controls)	246
4.6.4.	Các điều khiển Container.....	248
4.6.5.	ListView và TreeView	250
4.6.6.	Điều khiển Timer và biểu tượng động.....	251
4.6.7.	Sự kiện kéo lê (Drag và Drop)	252
4.6.8.	Kiểm tra dữ liệu hợp lệ	253

CHƯƠNG 5: TRUY XUẤT CƠ SỞ DỮ LIỆU	266
5.1. Giới thiệu ADO.NET.....	267
5.1.1. Giới thiệu chung	267
5.1.2. So sánh với phiên bản ADO.....	268
5.1.3. Kiến trúc ADO.NET	270
5.2. Kết nối và truy xuất cơ sở dữ liệu.....	273
5.2.1. Đôi tượng Connection.....	275
5.2.2. Đôi tượng command	279
5.2.3. Đôi tượng data reader	285
5.2.4. Đôi tượng data adapter.....	288
5.2.5. Đôi tượng DataSet	293
5.2.6. Đôi tượng DataTable	296
5.2.7. Đôi tượng DataView.....	301
5.2.8. Đôi tượng DataRelation	305
5.2.9. Ràng buộc dữ liệu vào các điều khiển	308
5.2.10. Điều khiển tạo lập báo cáo sử dụng Crystal Report.....	313
5.3. LINQ (Language Integrated Query).....	316
5.3.1. Giới thiệu LINQ	316
5.3.2. Truy vấn dữ liệu LINQ	318

LỜI MỞ ĐẦU

Bài giảng môn học Lập trình WinForm C# được phát hành nội bộ để phục vụ giảng dạy cho đối tượng sinh viên

Mục tiêu của bài giảng là cung cấp cho sinh viên những kiến thức nền tảng về kỹ thuật lập trình trực quan tạo ứng dụng WinForm với C#

Bên cạnh đó, sinh viên có thể củng cố các kiến thức về lập trình cơ sở, lập trình hướng đối tượng. Sinh viên có thể sử dụng thành thạo ngôn ngữ lập trình WinForm C# trong nhiều ứng dụng thực tế

Ngoài ra, sinh viên có thể vận dụng kiến thức nền tảng về lập trình hướng đối tượng trong việc tiếp cận ngôn ngữ lập trình C# và nhiều ngôn ngữ lập trình hướng đối tượng hiện nay

Bài giảng bao gồm 5 chương cung cấp cho sinh viên tiếp cận từ lập trình cơ bản, lập trình hướng đối tượng, đến lập trình Windows Form truy xuất cơ sở dữ liệu trong C#

Chương 1. Microsoft .Net và C#

Chương 2. Cơ bản về C#

Chương 3. Lập trình hướng đối tượng với C#

Chương 4. Windows Form

Chương 5. Truy xuất cơ sở dữ liệu

GIẢNG VIÊN

Ths. Phạm Phú Khương

CHƯƠNG 1: MICROSOFT .NET

A. MỤC TIÊU CHƯƠNG

1. VỀ KIẾN THỨC

Cung cấp cho sinh viên những kiến thức về:

- Nguồn gốc của Microsoft
- Lịch sử phát triển của .NET Framework
- Kiến trúc .NET Framework
- Common Language Runtime
- Thư viện .NET Framework

2. VỀ KỸ NĂNG

Sau khi học xong chương này sinh viên có thể vận dụng những kiến thức cơ bản nhất của NET Framework để triển khai các ứng dụng.

B. NỘI DUNG

1.1. Tổng quan về Microsoft.Net

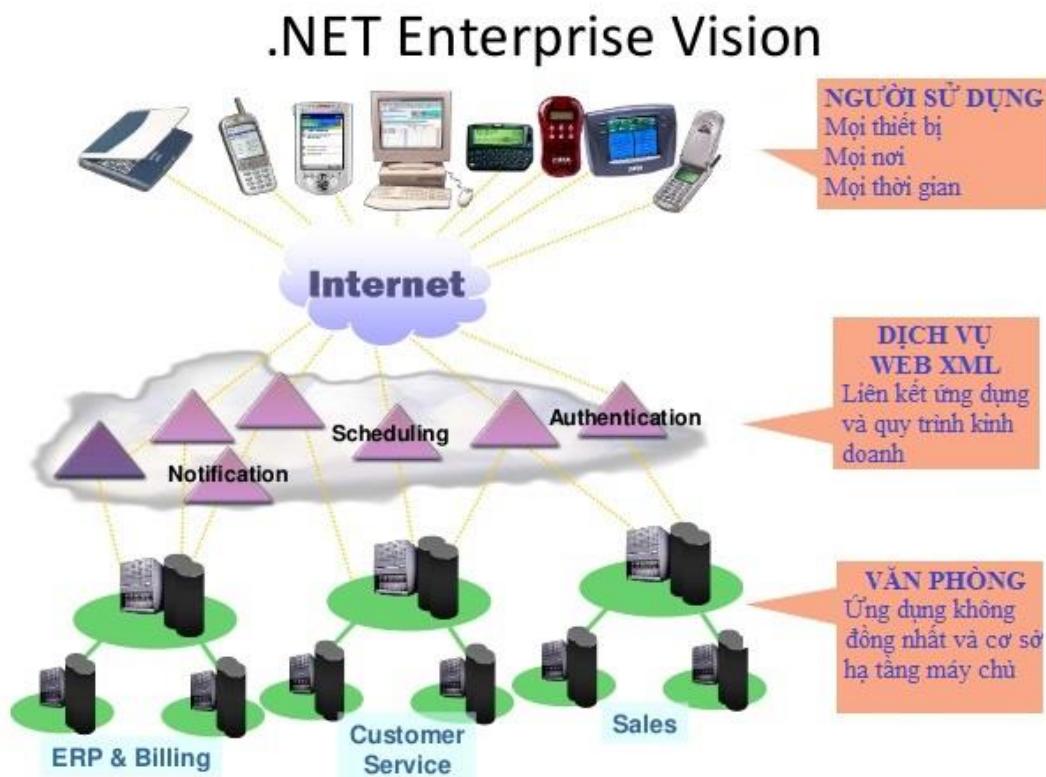
1.1.1. .NET là gì

Năm 1991, một nhóm các kỹ sư của Sun Microsystems muốn lập trình để điều khiển các thiết bị điện tử dân dụng như tivi, đầu video,... Ban đầu, họ định sử dụng C/C++ nhưng trình biên dịch C/C++ lại phụ thuộc vào từng loại CPU. Do đó, họ đã bắt tay vào xây dựng một ngôn ngữ chạy nhanh, hiệu quả và độc lập với thiết bị. Java khởi đầu tên là “Oak” – là cây sồi mọc phía sau văn phòng của nhà thiết kế chính ông Jame Gosling. Sau này ông nhận thấy đã có ngôn ngữ lập trình Oak rồi nên nhóm thiết kế quyết định đổi tên thành Java. Java là tên của quán cà phê mà nhóm thiết kế Java hay đến đó uống.

.NET là nền tảng của Microsoft cho các dịch vụ Web XML, là thế hệ phần mềm kế tiếp kết nối thế giới thông tin, các thiết bị và tất cả mọi người trong một thế thống nhất. Nền tảng .NET cho phép tạo ra và sử dụng các ứng dụng, các quá trình và các Website dựa trên XML như những dịch vụ chia sẻ, kết nối thông tin và hoạt động cùng nhau, trên bất cứ nền tảng hay thiết bị thông minh nào, nhằm mục đích cung cấp những

giải pháp theo yêu cầu cho các tổ chức và các cá nhân riêng biệt. Nền tảng .Net của Microsoft đưa ra các công nghệ, các công cụ và sản phẩm để tạo ra và chạy các dịch vụ Web.

Microsoft NET framework đã được thiết kế với mục tiêu hoàn thành nhiều nhiệm vụ một cách đơn giản, nhanh chóng và hiệu quả. Có một số công cụ nhằm mục đích cùnghỗn gắn liền với khuôn khổ mang tính cách mạng này. Ngoài ra nó có thể làm những việc như truy cập dữ liệu, cửa sổ, kết nối Internet, và nhiều chức năng của Win32 API có thể truy cập thông qua một mô hình đối tượng rất đơn giản.



Hình 1.1. Tổng quan về .NET

1.1.2. Các lĩnh vực của .NET

- NET Framework: khuôn khổ này là một môi trường mà trong đó người ta có thể xây dựng, tạo và triển khai ứng dụng của mình và các thành phần thê hệ tiếp theo, được gọi là dịch vụ Web.

- Sản phẩm .NET : Tất cả các sản phẩm của Microsoft đã được tích hợp vào tầm nhìn lớn và cốt lõi của nó, XML là ngôn ngữ để đại diện cho / mô tả dữ liệu, và SOAP là giao thức chính để truyền dữ liệu.
- Dịch vụ .NET : Đây là các dịch vụ có thể được phát triển và triển khai trong nền tảng .NET.

Tóm lại, chiến lược của NET. Triển khai hệ thống phân tán sử dụng XML cho tất cả các hệ thống tính toán khác nhau như máy tính, máy tính xách tay, thiết bị không dây (di động...), và các thiết bị thông tin liên lạc khác để trao đổi thông tin với nhau trong một ứng dụng duy nhất.

1.2. Tổng quan về .NET Framework

1.2.1. .NET Framework là gì?

.NET Framework là một nền tảng lập trình và cũng là một nền tảng thực thi ứng dụng chủ yếu trên hệ điều hành Microsoft Windows được phát triển bởi Microsoft. Các chương trình được viết trên nền .NET Framework sẽ được triển khai trong môi trường phần mềm (ngược lại với môi trường phần cứng) được biết đến với tên Common Language Runtime (CLR). Môi trường phần mềm này là một máy ảo trong đó cung cấp các dịch vụ như an ninh phần mềm (security), quản lý bộ nhớ (memory management), và các xử lý lỗi ngoại lệ (exception handling).

.NET Framework bao gồm tập các thư viện lập trình lớn, và những thư viện này hỗ trợ việc xây dựng các chương trình phần mềm như lập trình giao diện; truy cập, kết nối cơ sở dữ liệu; ứng dụng web; các giải thuật, cấu trúc dữ liệu; giao tiếp mạng... CLR cùng với bộ thư viện này là 2 thành phần chính của .NET framework.

.NET Framework đơn giản hóa việc viết ứng dụng bằng cách cung cấp nhiều thành phần được thiết kế sẵn, người lập trình chỉ cần học cách sử dụng và tùy theo sự sáng tạo mà gắn kết các thành phần đó lại với nhau. Nhiều công cụ được tạo ra để hỗ trợ xây dựng ứng dụng .NET, và IDE (Integrated Development Environment) được phát triển và hỗ trợ bởi chính Microsoft là Visual Studio.

1.2.2. Lịch sử phát triển của .NET Framework

Bảng 1.1: Lịch sử phát triển của .NET Framework

Phiên bản	Số hiệu phiên bản	Ngày phát hành	Visual Studio	Được phát hành kèm theo
1.0	1.0.3705.0	12/02/2002	Visual Studio.NET	Windows XP Tablet and Media Center Editions
1.1	1.1.4322.573	24/04/2003	Visual Studio.NET 2003	Windows Server 2003
2.0	2.0.50727.42	07/11/2005	Visual Studio 2005	Windows Server 2003 R2
3.0	3.0.4506.30	06/11/2006		Windows Vista, Windows Server 2008
3.5	3.5.21022.8	19/11/2007	Visual Studio 2008	Windows 7, Windows Server 2008 R2
4.0	4.0.30319.1	12/04/2010	Visual Studio 2010	
4.5	4.5.50709	15/08/2012	Visual Studio 2012	Windows 8, Windows Server 2012

.NET Framework 1.0

Đây là phiên bản đầu tiên của .NET framework, nó được phát hành vào năm 2002 cho các hệ điều hành Windows 98, NT 4.0, 2000 và XP. Việc hỗ trợ chính thức từ Microsoft cho phiên bản này kết thúc vào 10/7/2007, tuy nhiên thời gian hỗ trợ mở rộng được kéo dài đến 14/7/2009.

.NET Framework 1.1

Phiên bản nâng cấp đầu tiên được phát hành vào 4/2003. Sự hỗ trợ của Microsoft kết thúc vào 14/10/2008, và hỗ trợ mở rộng được định đến 8/10/2013.

Những thay đổi so với phiên bản 1.0:

- Tích hợp hỗ trợ mobile ASP.NET (trước đây chỉ là phần mở rộng tùy chọn)
- Thay đổi về kiến trúc an ninh - sử dụng sandbox khi thực thi các ứng dụng từ Internet.
- Tích hợp hỗ trợ ODBC và cơ sở dữ liệu Oracle
- .NET Compact Framework
- Hỗ trợ IPv6 (Internet Protocol version 6)
- Vài thay đổi khác trong API

.NET Framework 2.0

Kể từ phiên bản này,.NET framework hỗ trợ đầy đủ nền tảng 64-bit. Ngoài ra, cũng có một số thay đổi trong API; hỗ trợ các kiểu "generic"; bổ sung sự hỗ trợ cho ASP.NET; .NET Micro Framework - một phiên bản.NET framwork có quan hệ với Smart Personal Objects Technology.

.NET Framework 3.0

Đây không phải là một phiên bản mới hoàn toàn, thực tế đây chỉ là một bản nâng cấp của.NET 2.0. Phiên bản 3.0 này còn có tên gọi khác là WinFX, nó bao gồm nhiều sự thay đổi nhằm hỗ trợ việc phát triển và chuyển đổi (porting) các ứng dụng trên Windows Vista. Tuy nhiên, không có sự xuất hiện của.NET Compact Framework 3.0 trong lần phát hành này.

Bốn thành phần chính trong phiên bản 3.0:

- Windows Presentation Foundation (WPF): Đây là một công nghệ mới, và là một nỗ lực của Microsoft nhằm thay đổi phương pháp hay cách tiếp cận việc lập trình một ứng dụng sử dụng giao diện đồ họa trên Windows với sự hỗ trợ của ngôn ngữ XAML.
- Windows Communication Foundation (WCF - tên mã là Indigo): Một nền tảng mới cho phép xây dựng các ứng dụng hướng dịch vụ (service-oriented).

- Windows Workflow Foundation (WF): Một kiến trúc hỗ trợ xây dựng các ứng dụng workflow (luồng công việc) một cách dễ dàng hơn. WF cho phép định nghĩa, thực thi và quản lý các workflow từ cả cách nhìn theo hướng kĩ thuật và hướng thương mại.
- Windows CardSpace (tên mã là InfoCard): một kiến trúc để quản lý định danh (identity management) cho các ứng dụng được phân phối.
- Ngoài ra Silverlight (hay WPF / E), một phiên bản nhánh.NET Framework hỗ trợ các ứng dụng trên nền web, được Microsoft tạo ra để cạnh tranh với Flash.

Có thể minh họa.NET 3.0 bằng một công thức đơn giản:

$$\text{.NET 3.0} = \text{.NET 2.0} + \text{WPF} + \text{WCF} + \text{WF} + \text{WCS}$$

.NET Framework 3.5

Được phát hành vào 11/2007, phiên bản này sử dụng CLR 2.0. Đây có thể được xem là tương đương với phiên bản .NET Framework 2.0 SP1 và phiên bản .NET Framework 3.0 SP1 cộng lại. .NET Compact Framework 3.5 được ra đời cùng với phiên bản.NET framework này.

Các thay đổi kể từ phiên bản 3.0:

- Các tính năng mới cho ngôn ngữ C# 3.0 và VB.NET 9.0
- Hỗ trợ Expression Tree và Lambda
- Các phương thức mở rộng (Extension methods)
- Các kiểu ẩn danh (Anonymous types)
- LINQ
- Phân trang (paging) cho ADO.NET
- API cho nhập xuất mạng không đồng bộ (asynchronous network I/O)
- Peer Name Resolution Protocol resolver.
- Cải thiện WCF và WF
- Tích hợp ASP.NET AJAX
- Namespace mới System.CodeDom
- Microsoft ADO.NET Entity Framework 1.0

Cũng như phiên bản 3.0, có thể minh họa sự thay đổi của.NET 3.5 bằng công thức:

.NET 3.5 =.NET 3.0 + LINQ + ASP.NET 3.5 + REST

.NET Framework 4.0

Phiên bản beta đầu tiên của.NET 4 xuất hiện vào 5/2009 và phiên bản RC (Release Candidate) được ra mắt vào 2/2010. Bản chính thức của.NET 4 được công bố và phát hành cùng với Visual Studio 2010 vào 12/4/2010.

Các tính năng mới được Microsoft bổ sung trong.NET 4:

- Dynamic Language Runtime
- Code Contracts
- Managed Extensibility Framework
- Hỗ trợ các tập tin ánh xạ bộ nhớ (memory-mapped files)
- Mô hình lập trình mới cho các ứng dụng đa luồng (multithreaded) và bất đồng bộ (asynchronous)
- Cải thiện hiệu năng, các mô hình workflow.

.NET Framework 4.5

Những thông tin đầu tiên của.NET 4.5 được Microsoft công bố vào 14/9/2011 tại BUILD Windows Conference, và nó chính thức được ra mắt vào 15/8/2012.

Kể từ phiên bản này, Microsoft bắt đầu cung cấp 2 gói cài đặt riêng biệt, gói đầy đủ và gói giản chức năng client profiles.

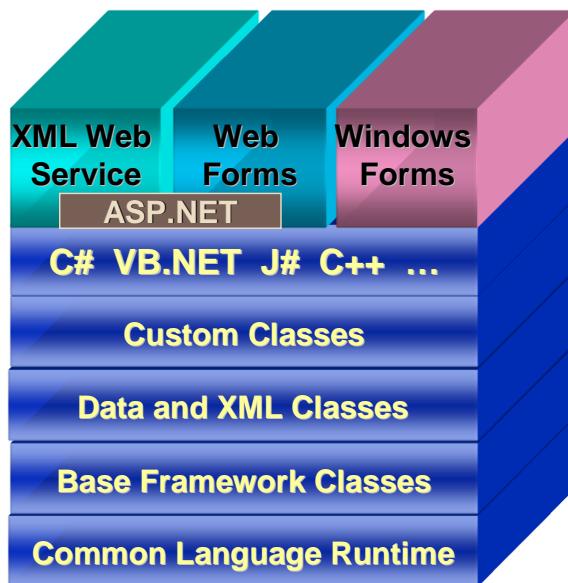
1.2.3. Thành phần .NET Framework

.NET Framework là một nền tảng mới làm đơn giản việc phát triển ứng dụng trong môi trường phân tán của Internet. .NET Framework được thiết kế đầy đủ để đáp ứng theo quan điểm sau:

- Cung cấp một môi trường lập trình hướng đối tượng vững chắc
- Cung cấp một môi trường thực thi mã nguồn giảm thiểu đóng gói phần mềm và tranh chấp về phiên bản, đảm bảo việc thực thi an toàn mã nguồn, bao gồm cả

việc mã nguồn được tạo bởi hãng thứ ba hay bất cứ hãng nào mà tuân thủ theo kiến trúc .NET, loại bỏ được những lỗi thực hiện các mã script hay môi trường thông dịch.

- Làm cho những người phát triển có thể nắm vững nhiều kiểu ứng dụng khác nhau, từ những ứng dụng trên nền Windows đến những ứng dụng Web. Để xây dựng tất cả các thông tin dựa trên tiêu chuẩn công nghiệp để đảm bảo rằng mã nguồn .NET có thể tích hợp với bất kỳ mã nguồn khác.



Hình 1.2. Mô hình tổng quan .Net Framework

Kiến trúc .NET Framework có hai thành phần chính như sau đó là CLR (Common Language Runtime) và .NET Framework classes:

CLR (Common Language Runtime): là nền tảng của .NET Framework, môi trường quản lý thi hành các mã nguồn, cung cấp các dịch vụ cốt lõi như: quản lý bộ nhớ, quản lý tiểu trình, và quản lý từ xa. Ngoài ra nó còn thúc đẩy việc sử dụng kiểu an toàn và các hình thức khác của việc chính xác mã nguồn, đảm bảo cho việc thực hiện được bảo mật và mạnh mẽ. Thật vậy, khái niệm quản lý mã nguồn là nguyên lý nền tảng của runtime. Mã nguồn mà đích tới runtime thì được biết như là mã nguồn được quản lý (managed code). Trong khi đó mã nguồn mà không có đích tới runtime thì được biết như mã nguồn không được quản lý (unmanaged code).

Khi biên dịch mã nguồn, mã nguồn được biên dịch ra thành một ngôn ngữ trung gian gọi là MSIL (Microsoft Intermediate Language) và lưu trữ trong một tập tin gọi là assembly. Chính ngôn ngữ trung gian MSIL này là ngôn ngữ chung cho tất cả các ngôn ngữ .NET hiện có. Trong khi biên dịch như vậy, các ứng dụng cũng tạo ra những thông tin cần thiết để giới thiệu ứng dụng, ta gọi những thông tin này là metadata.

Khi chúng ta chạy chương trình, thì MSIL được biên dịch một lần nữa, sử dụng trình biên dịch Just-In-Time (JIT). Kết quả là mã máy được thực thi bởi bộ xử lý của máy. Trình biên dịch JIT tiêu chuẩn thì thực hiện theo yêu cầu. Khi một phương thức được gọi, trình biên dịch JIT phân tích MSIL và tạo ra sản phẩm mã máy có hiệu quả cao, mã này có thể chạy rất nhanh. Trình biên dịch JIT đủ thông minh để nhận ra khi một mã đã được biên dịch, do vậy khi ứng dụng chạy thì việc biên dịch chỉ xảy ra khi cần thiết, tức là chỉ biên dịch mã MSIL chưa biên dịch ra mã máy. Khi đó một ứng dụng .NET thực hiện, chúng có xu hướng là chạy nhanh và nhanh hơn nữa, cũng như là những mã nguồn được biên dịch rồi thì được dùng lại.

Do tất cả các ngôn ngữ .NET Framework cùng tạo ra sản phẩm MSIL giống nhau, nên kết quả là một đối tượng được tạo ra từ ngôn ngữ này có thể được truy cập hay được dẫn xuất từ một đối tượng của ngôn ngữ khác trong .NET. Ví dụ, người phát triển có thể tạo một lớp cơ sở trong VB.NET và sau đó dẫn xuất nó trong C# một cách dễ dàng.

.NET Framework classes: Thư viện lớp .NET Framework là một tập hợp những kiểu dữ liệu được dùng lại và được kết hợp chặt chẽ với Common Language Runtime. Thư viện lớp là hướng đối tượng cung cấp những kiểu dữ liệu mà mã nguồn được quản lý của chúng ta có thể dẫn xuất. Điều này không chỉ làm cho những kiểu dữ liệu của .NET Framework dễ sử dụng mà còn làm giảm thời gian liên quan đến việc học đặc tính mới của .NET Framework.Thêm vào đó, các thành phần của các hằng thứ ba có thể tích hợp với những lớp trong .NET Framework.

Cũng như mong đợi của người phát triển với thư viện lớp hướng đối tượng, kiểu dữ liệu .NET Framework cho phép người phát triển thiết lập nhiều mức độ thông dụng của việc lập trình, bao gồm các nhiệm vụ như: quản lý chuỗi, thu thập hay chọn lọc dữ liệu, kết nối với cơ sở dữ liệu, và truy cập tập tin. Ngoài những nhiệm vụ thông dụng

trên. Thư viện lớp còn đưa vào những kiểu dữ liệu để hỗ trợ cho những kịch bản phát triển chuyên biệt khác. Ví dụ người phát triển có thể sử dụng .NET Framework để phát triển những ứng dụng và dịch vụ như sau:

- Ứng dụng Console
- Ứng dụng giao diện GUI trên Windows (Windows Forms)
- Ứng dụng ASP.NET
- Dịch vụ XML Web
- Dịch vụ Windows

Trong đó những lớp Windows Forms cung cấp một tập hợp lớn các kiểu dữ liệu nhằm làm đơn giản việc phát triển các ứng dụng GUI chạy trên Windows. Còn nếu như viết các ứng dụng ASP.NET thì có thể sử dụng các lớp Web Forms trong thư viện .NET Framework.

C. HÌNH THÚC VÀ PHƯƠNG PHÁP GIẢNG DẠY

- Trình chiếu powerpoint
- Đặt vấn đề, trao đổi
- Thực nghiệm kết hợp với máy tính

D. TÀI LIỆU THAM KHẢO

- [1] Professional C#, 2nd Edition, Wrox Press Ltd.
- [2] A programmer's Introduction to C#, Eric Gunnerson, Apress, 2000
- [3] Programming C#, Jesse Liberty, O'Reilly, First Edition, 2001
- [4] C# bible, Jeff Ferguson et al, Wiley Publishing, 2002
- [5] Thinking in C#, Larry O'Brien, Bruce Eckel, Prentice Hall.
- [6] Presenting C#, Sams Publishing, 2002
- [7] C# Language Reference, Anders Hejlsberg and Scott Wiltamuth, Microsoft Corp.

E. CÂU HỎI

Câu hỏi 1: NET Framework có bao nhiêu thành?

Trả lời 1: Net.Framework có ba thành phần (Common Language Runtime, NET Framework Base Classes, Giao diện người dùng)

Câu hỏi 2: Common Language Runtime cung cấp những tính năng nào?

Trả lời 2: Common Language Runtime cung cấp những tính năng (Quản lý bộ nhớ tự động, tương thích ngôn ngữ, nền tảng độc lập, quản lý an ninh, cung cấp kiểu an toàn)

Câu hỏi 3: Assemblies lưu trữ những thông tin gì?

Trả lời 3: Assemblies lưu trữ các thông tin (thông tin về tên, phiên bản, các thông tin bảo mật ...)

CHƯƠNG 2: LẬP TRÌNH C# CƠ BẢN

A. MỤC TIÊU CHƯƠNG

1. VỀ KIẾN THỨC

Cung cấp cho sinh viên những kiến thức về:

- Ngôn ngữ C# và đặc trưng của ngôn ngữ C#
- Cách biên dịch và thực hiện một chương trình C# đơn giản

2. VỀ KỸ NĂNG

Sau khi học xong chương này sinh viên có thể vận dụng những kiến thức cơ bản nhất của.NET Framework để triển khai các ứng dụng.

B. NỘI DUNG

2.1. Tại sao phải sử dụng ngôn ngữ C#

2.1.1. C# là ngôn ngữ đơn giản

- C# loại bỏ được một vài sự phức tạp và rối rắm của các ngôn ngữ C++ và Java.
- C# khá giống C / C++ về diện mạo, cú pháp, biểu thức, toán tử.
- Các chức năng của C# được lấy trực tiếp từ ngôn ngữ C / C++ nhưng được cải tiến để làm cho ngôn ngữ đơn giản hơn.

2.1.2. C# là ngôn ngữ hiện đại

- C# có được những đặc tính của ngôn ngữ hiện đại như:
- Xử lý ngoại lệ.
- Thu gom bộ nhớ tự động.
- Có những kiểu dữ liệu mở rộng.
- Bảo mật mã nguồn.

2.1.3. C# là ngôn ngữ hướng đối tượng

- C# hỗ trợ tất cả những đặc tính của ngôn ngữ hướng đối tượng là:
- Sự đóng gói (encapsulation).

- Sự kế thừa (inheritance).
- Đa hình (polymorphism).

2.1.4. C# là ngôn ngữ mạnh mẽ và mềm dẻo

- Với ngôn ngữ C#, chúng ta chỉ bị giới hạn ở chính bản thân của chúng ta. Ngôn ngữ này không đặt ra những ràng buộc lên những việc có thể làm.
- C# được sử dụng cho nhiều dự án khác nhau như: tạo ra những ứng dụng xử lý văn bản, ứng dụng đồ họa, xử lý bảng tính; thậm chí tạo ra những trình biên dịch cho các ngôn ngữ khác.
- C# là ngôn ngữ sử dụng giới hạn những từ khóa. Phần lớn các từ khóa dùng để mô tả thông tin, tuy nhiên không vì thế mà C# kém phần mạnh mẽ. Chúng ta dễ dàng có thể tìm thấy rằng ngôn ngữ này có thể được sử dụng để làm bất cứ nhiệm vụ nào.

2.1.5. C# là ngôn ngữ ít từ khóa

C# là ngôn ngữ sử dụng giới hạn những từ khóa. Phần lớn các từ khóa được sử dụng để mô tả thông tin. Chúng ta có thể nghĩ rằng một ngôn ngữ có nhiều từ khóa thì sẽ mạnh hơn. Điều này không phải sự thật, ít nhất là trong trường hợp ngôn ngữ C#, chúng ta có thể tìm thấy rằng ngôn ngữ này có thể được sử dụng để làm bất cứ nhiệm vụ nào. Bảng sau liệt kê các từ khóa của ngôn ngữ C#.

Bảng 1.2: Bảng từ khóa của C#

<u>tim</u>	<u>default</u>	<u>foreach</u>	<u>object</u>	<u>sizeof</u>	<u>unsafe</u>
<u>as</u>	<u>delegate</u>	<u>goto</u>	<u>operator</u>	<u>stackalloc</u>	<u>ushort</u>
<u>base</u>	<u>do</u>	<u>if</u>	<u>out</u>	<u>static</u>	<u>using</u>
<u>bool</u>	<u>double</u>	<u>implicit</u>	<u>override</u>	<u>string</u>	<u>virtual</u>
<u>break</u>	<u>else</u>	<u>in</u>	<u>params</u>	<u>struct</u>	<u>volatile</u>
<u>byte</u>	<u>enum</u>	<u>int</u>	<u>private</u>	<u>switch</u>	<u>void</u>
<u>case</u>	<u>event</u>	<u>interface</u>	<u>protected</u>	<u>this</u>	<u>while</u>
<u>catch</u>	<u>explicit</u>	<u>internal</u>	<u>public</u>	<u>throw</u>	
<u>char</u>	<u>extern</u>	<u>is</u>	<u>readonly</u>	<u>true</u>	
<u>checked</u>	<u>false</u>	<u>lock</u>	<u>ref</u>	<u>try</u>	
<u>class</u>	<u>finally</u>	<u>long</u>	<u>return</u>	<u>typeof</u>	
<u>const</u>	<u>fixed</u>	<u>namespace</u>	<u>sbyte</u>	<u>uint</u>	

continue	<u>float</u>	<u>new</u>	<u>sealed</u>	<u>ulong</u>	
decimal	<u>for</u>	<u>null</u>	<u>short</u>	<u>unchecked</u>	

2.1.6. C# là ngôn ngữ module hóa

Mã nguồn C# có thể được viết trong những phần được gọi là những lớp, những lớp này chứa các phương thức thành viên của nó. Những lớp và những phương thức có thể được sử dụng lại trong ứng dụng hay các chương trình khác. Bằng cách truyền các mẫu thông tin đến những lớp hay phương thức chúng ta có thể tạo ra những mã nguồn dùng lại có hiệu quả.

2.1.7. C# đã và đang trở nên phổ biến

- C# mang đến sức mạnh của C++ cùng với sự dễ dàng của ngôn ngữ Visual Basic. Bởi vì chúng được viết lại từ một nền tảng. Chúng ta có thể viết nhiều chương trình với ít mã nguồn hơn nếu dùng C#.
- Mặc dù C# loại bỏ một vài đặc tính của C++, nhưng bù lại nó tránh được những lỗi thường gặp trong ngôn ngữ C++. Điều này có thể tiết kiệm được hàng giờ hay thậm chí hàng ngày để hoàn tất một chương trình.
- Điểm giống nhau C# và Java là cả hai cùng biên dịch ra mã trung gian: C# biên dịch ra MSIL còn Java biên dịch ra bytecode. Sau đó chúng được thực hiện bằng cách thông dịch hoặc biên dịch trong từng máy áo tương ứng. Tuy nhiên, trong ngôn ngữ C# nhiều hỗ trợ được đưa ra để biên dịch mã ngôn ngữ trung gian sang mã máy. C# chứa nhiều kiểu dữ liệu cơ sở hơn Java và cũng cho phép nhiều mở rộng với kiểu dữ liệu giá trị.
- Tương tự như Java, C# cũng từ bỏ tính đa kế thừa, tuy nhiên mô hình kế thừa đơn này được mở rộng bởi tính hiện thực nhiều giao tiếp

2.1.8. Ngôn ngữ C# với những ngôn ngữ khác

Chúng ta đã từng nghe đến những ngôn ngữ khác như Visual Basic, C++ và Java. Có lẽ chúng ta cũng tự hỏi sự khác nhau giữa ngôn ngữ C# và những ngôn ngữ đó. Và cũng tự hỏi tại sao lại chọn ngôn ngữ này để học mà không chọn một trong những ngôn

ngữ kia. Có rất nhiều lý do và chúng ta hãy xem một số sự so sánh giữa ngôn ngữ C# với những ngôn ngữ khác giúp chúng ta phần nào trả lời được những thắc mắc.

Microsoft nói rằng C# mang đến sức mạnh của ngôn ngữ C++ với sự dễ dàng của ngôn ngữ Visual Basic. Có thể nó không dễ như Visual Basic, nhưng với phiên bản Visual Basic.NET (Version 7) thì ngang nhau. Bởi vì chúng được viết lại từ một nền tảng. Chúng ta có thể viết nhiều chương trình với ít mã nguồn hơn nếu dùng C#.

Mặc dù C# loại bỏ một vài các đặc tính của C++, nhưng bù lại nó tránh được những lỗi mà thường gặp trong ngôn ngữ C++. Điều này có thể tiết kiệm được hàng giờ hay thậm chí hàng ngày trong việc hoàn tất một chương trình. Chúng ta sẽ hiểu nhiều về điều này trong các chương của giáo trình.

Một điều quan trọng khác với C++ là mã nguồn C# không đòi hỏi phải có tập tin header. Tất cả mã nguồn được viết trong khai báo một lớp.

Như đã nói ở bên trên. .NET runtime trong C# thực hiện việc thu gom bộ nhớ tự động. Do điều này nên việc sử dụng con trỏ trong C# ít quan trọng hơn trong C++. Những con trỏ cũng có thể được sử dụng trong C#, khi đó những đoạn mã nguồn này sẽ được đánh dấu là không an toàn (unsafe code).

C# cũng từ bỏ ý tưởng đa kế thừa như trong C++. Và sự khác nhau khác là C# đưa thêm thuộc tính vào trong một lớp giống như trong Visual Basic. Và những thành viên của lớp được gọi duy nhất bằng toán tử “.” và khác với C++ có nhiều cách gọi trong các tình huống khác nhau.

Một ngôn ngữ khác rất mạnh và phổ biến là Java, giống như C++ và C# được phát triển dựa trên C. Nếu chúng ta quyết định sẽ học Java sau này, chúng ta sẽ tìm được nhiều cái mà học từ C# có thể được áp dụng.

Điểm giống nhau C# và Java là cả hai cùng biên dịch ra mã trung gian: C# biên dịch ra MSIL còn Java biên dịch ra bytecode. Sau đó chúng được thực hiện bằng cách thông dịch hoặc biên dịch just-in-time trong từng máy ảo tương ứng. Tuy nhiên, trong ngôn ngữ C# nhiều hỗ trợ được đưa ra để biên dịch mã ngôn ngữ trung gian sang mã máy. C# chứa nhiều kiểu dữ liệu cơ bản hơn Java và cũng cho phép nhiều sự mở rộng

với kiểu dữ liệu giá trị. Ví dụ, ngôn ngữ C# hỗ trợ kiểu liệt kê (enumerator), kiểu này được giới hạn đến một tập hằng được định nghĩa trước, và kiểu dữ liệu cấu trúc đây là kiểu dữ liệu giá trị do người dùng định nghĩa. Chúng ta sẽ được tìm hiểu kỹ hơn về kiểu dữ liệu tham chiếu và kiểu dữ liệu giá trị sẽ được trình bày trong phần sau. Tương tự như Java, C# cũng từ bỏ tính đa kế thừa trong một lớp, tuy nhiên mô hình kế thừa đơn này được mở rộng bởi tính đa kế thừa nhiều giao diện.

2.2. Các bước chuẩn bị cho chương trình C#

2.2.1. Thực hiện ứng dụng Console C#.Net đơn giản

Để bắt đầu cho việc tìm hiểu ngôn ngữ C#, chương đầu tiên trình bày cách soạn thảo, biên dịch và thực thi một chương trình C# đơn giản nhất.

Ví dụ: *Viết chương trình hiển thị ra màn hình dòng chữ “Welcome to C#.NET”*

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace Hello
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Wellcome to C#.Net");
            Console.ReadLine();
        }
    }
}
```

Cấu trúc của một chương trình C# đơn giản như sau:

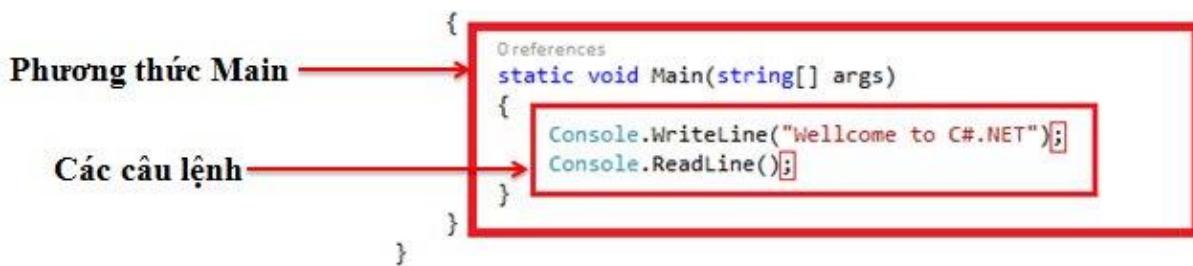
Phần khai báo dùng NameSpace (Option)

```
-----  
using System;  
  
using System.Collections.Generic;  
  
using System.Linq;  
  
using System.Text;
```

Phần định nghĩa NameSpace và lớp

```
-----  
namespace Hello  
{  
    class Program  
    {  
        static void Main(string[] args)  
        {  
            Console.WriteLine("Wellcom to C#.Net");  
            Console.ReadLine();  
        }  
    }  
}
```

Các câu lệnh được viết trong thân của phương thức (ở đây là phương thức Main) để thực hiện một công việc nào đó và kết thúc bằng dấu chấm phẩy (;)



Hình 2.1. Cấu trúc chương trình C# đơn giản

Đây là một định nghĩa lớp Hello gồm có một phương thức tên là Main(). Tất cả các chương trình C# phải chứa một lớp có phương thức là Main() là phương thức được gọi thực hiện đầu tiên mỗi khi thực hiện chương trình. Phương thức Console.WriteLine() sử dụng để xuất một chuỗi trên màn hình console và kết thúc dòng.

C# là một ngôn ngữ phân biệt hoa, thường. Bạn cần phải chú ý là các từ khóa trong C# đều được viết thường ví dụ public, class, static... trong khi các tên namespace, tên lớp, phương thức... sẽ được viết hoa đầu từ, ví dụ System, Console.WriteLine, Main...

Có hai cách để soạn thảo, biên dịch và thực thi chương trình đó là:

❖ **Sử dụng trình biên dịch dòng lệnh C#**

Sử dụng chương trình soạn thảo văn bản bất kỳ như Notepad soạn thảo mã nguồn chương trình, rồi lưu vào tập tin có phần mở rộng là *.cs, trong ví dụ này là Hello.cs.

Bước tiếp theo là biên dịch tập tin nguồn vừa tạo ra, sử dụng trình biên dịch dòng lệnh C# (C# command-line compiler) csc.exe : Chọn Start/ All Programs/ Microsoft Visual Studio/ Visual Studio Tools/ Visual Studio Command Prompt.

Chuyển đến thư mục chứa tập tin nguồn, rồi gõ lệnh sau:

csc.exe [/out: TậpTinThựcThi] TậpTinNguồn

Ví dụ: csc /out:Hello.exe Hello.cs

hay csc Hello.cs

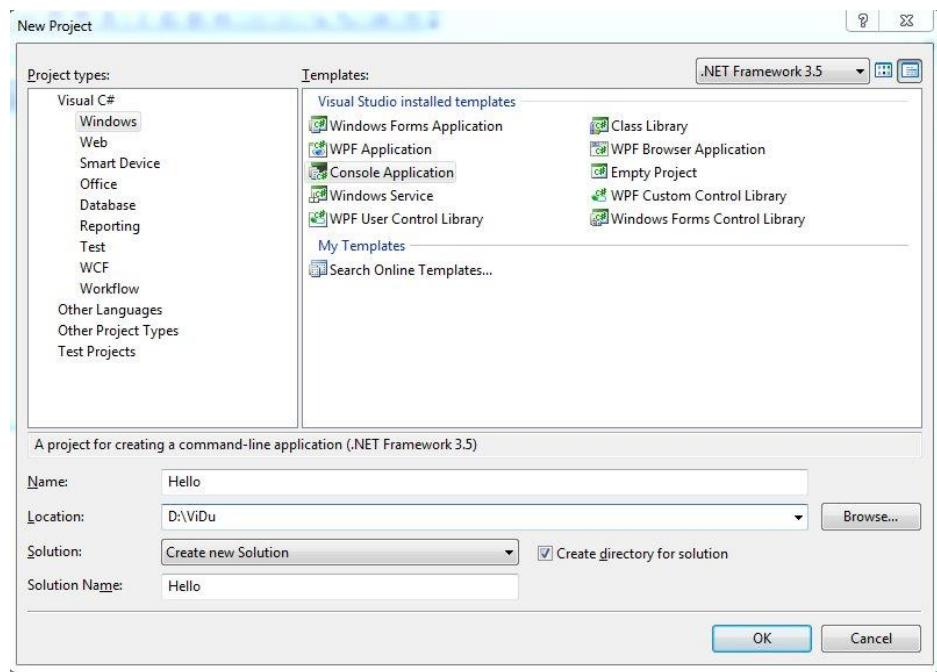
Kết quả tập tin Hello.exe sẽ xuất hiện trong cùng thư mục chứa tập tin nguồn. Có thể sử dụng tùy chọn /out, theo sau là tên của tập tin chương trình thực thi hay chính là kết quả biên dịch tập tin nguồn. Các tham số tùy chọn có rất nhiều nếu muốn tìm hiểu chúng ta có thể dùng lệnh: csc.exe /? Lệnh này hiển thị toàn bộ các tùy chọn biên dịch và các hướng dẫn sử dụng. Cuối cùng, thực hiện tập tin Hello.exe bằng cách gõ: Hello

❖ **Sử dụng môi trường phát triển tích hợp IDE (Integrated Development Environment) Visual Studio .NET**

Để tạo chương trình Hello.cs trong IDE, khởi động Visual Studio .NET, chọn menu File/ New/ Project. Chức năng này sẽ mở cửa sổ New Project. Để tạo ứng dụng, ta chọn mục Visual C# trong vùng Project types bên trái, ở vùng Templates bên phải, chọn Console Application. Lúc này chúng ta có thể nhập tên cho ứng dụng ở mục Name và lựa chọn thư mục nơi lưu trữ các tập tin này ở mục Location. Lưu ý rằng Visual Studio .NET tạo ra một namespace dựa trên tên của project mà ta vừa cung cấp (Hello), và thêm vào chỉ dẫn sử dụng namespace System bằng lệnh using System, bởi hầu như mọi chương trình mà chúng ta viết đều cần sử dụng các kiểu dữ liệu chứa trong namespace System. Visual Studio .NET tạo một lớp tên là Program.cs, có thể tùy ý đổi tên của chúng trong cửa sổ Solution Explorer. Khi đổi tên tập tin chứa lớp là Hello.cs, tên lớp cũng thay đổi thành Hello.

- Để biên dịch chương trình, chọn menu Build/ Build Solution.
- Để chạy chương trình không có hay không sử dụng chế độ debug, chọn Debug/ Start Debugging hay Start Without Debugging

Sau khi biên dịch và chạy chương trình, kết quả dòng chữ “Welcome to C#.NET” hiển thị ra màn hình



Hình 2.2. Hộp thoại NewProject

Chúng ta cần tìm hiểu khái niệm assembly. Một assembly là một khối xây dựng cơ bản của bất kỳ ứng dụng .NET Framework. Ví dụ, khi bạn tạo một ứng dụng C# đơn giản, Visual Studio sẽ tạo ra một assembly ở dạng một hay nhiều tập tin khả thi (executable file) .exe hay thư viện liên kết động (Dynamic Link Library) .dll. Assembly chứa các thông tin phát triển, chi tiết tất cả các dữ liệu và kiểu đối tượng trong dự án, quản lý bảo mật, chia sẻ, phiên bản và mã MSIL khả thi. Tập tin assembly được tự động phát sinh bởi trình biên dịch khi biên dịch thành công một ứng dụng .NET.

2.2.2. Phát triển chương trình minh họa C#

❖ Chương trình Console (CLI – Command Line Interface)

- Giao tiếp với người dùng bằng bàn phím
- Không có giao diện đồ họa (GUI – Graphical User Interface)



Hình 2.3. Giao Diện Console

❖ Chương trình Windows Form

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



Hình 2.4. Giao Diện Winform

❖ Chương trình Web Form

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



Hình 2.5. Giao Diện Web Form

2.3. Kiểu dữ liệu

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...) và trình biên dịch sẽ giúp cho người lập trình không bị lỗi khi chỉ cho phép một loại kiểu dữ liệu có thể được gán cho các kiểu dữ liệu khác. Kiểu dữ liệu của một đối tượng là một tín hiệu để trình biên dịch nhận biết kích thước của một đối tượng (kiểu int có kích thước là 4 byte) và khả năng của nó (như một đối tượng button có thể vẽ, phản ứng khi nhấn,...).

Tương tự như C++ hay Java, C# chia thành hai tập hợp kiểu dữ liệu chính: Kiểu xây dựng sẵn (built-in) mà ngôn ngữ cung cấp cho người lập trình và kiểu được người dùng định nghĩa (user-defined) do người lập trình tạo ra.

C# phân tập hợp kiểu dữ liệu này thành hai loại: Kiểu dữ liệu giá trị (value) và kiểu dữ liệu tham chiếu (reference). Việc phân chia này do sự khác nhau khi lưu kiểu dữ liệu giá trị và kiểu dữ liệu tham chiếu trong bộ nhớ. Đối với một kiểu dữ liệu giá trị thì sẽ được lưu giữ kích thước thật trong bộ nhớ đã cấp phát là stack. Trong khi đó thì địa chỉ của kiểu dữ liệu tham chiếu thì được lưu trong stack nhưng đối tượng thật sự thì lưu trong bộ nhớ heap.

Nếu chúng ta có một đối tượng có 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 liệu 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.

Ngoài ra C# cũng hỗ trợ một kiểu con trỏ C++, nhưng hiếm khi được sử dụng, và chỉ khi nào làm việc với những đoạn mã lệnh không được quản lý (unmanaged code). Mã lệnh không được quản lý là các lệnh được viết bên ngoài nền .MS.NET, như là các đối tượng COM.

2.3.1.1. Kiểu dữ liệu xây dựng sẵn

Ngôn ngữ C# đưa ra các kiểu dữ liệu xây dựng sẵn rất hữu dụng, phù hợp với một ngôn ngữ lập trình hiện đại, mỗi kiểu dữ liệu được ánh xạ đến một kiểu dữ liệu được hỗ trợ bởi hệ thống xác nhận ngôn ngữ chung (Common Language Specification: CLS) trong MS.NET. Việc ánh xạ các kiểu dữ liệu nguyên thủy của C# đến các kiểu dữ liệu của .NET sẽ đảm bảo các đối tượng được tạo ra trong C# có thể được sử dụng đồng thời với các đối tượng được tạo bởi bất cứ ngôn ngữ khác được biên dịch bởi .NET, như VB.NET.

Mỗi kiểu dữ liệu có một sự xác nhận và kích thước không thay đổi, không giống như C++, int trong C# luôn có kích thước là 4 byte bởi vì nó được ánh xạ từ kiểu Int32 trong .NET.

Bảng 2.1: Một số các kiểu dữ liệu được xây dựng sẵn

Kiểu C#	Số byte	Kiểu .NET	Mô tả
byte	1	Byte	Số nguyên dương không dấu từ 0-255
char	2	Char	Ký tự Unicode
bool	1	Boolean	Giá trị logic true/ false
sbyte	1	Sbyte	Số nguyên có dấu (từ -128 đến 127)
short	2	Int16	Số nguyên có dấu giá trị từ -32768 đến 32767.
ushort	2	UInt16	Số nguyên không dấu 0 – 65.535
int	4	Int32	Số nguyên có dấu -2.147.483.647 và 2.147.483.647
uint	4	UInt32	Số nguyên không dấu 0 – 4.294.967.295
float	4	Single	Kiểu dấu chấm động, giá trị xấp xỉ từ 3,4E-38 đến 3,4E+38, với 7 chữ số có nghĩa..
double	8	Double	Kiểu dấu chấm động có độ chính xác gấp đôi, giá trị xấp xỉ từ 1,7E-308 đến 1,7E+308, với 15,16 chữ số có nghĩa
decimal	8	Decimal	Có độ chính xác đến 28 con số và giá trị thập phân, được dùng trong tính toán tài chính, kiểu này đòi hỏi phải có hậu tố “m” hay “M” theo sau giá trị.
long	8	Int64	Kiểu số nguyên có dấu có giá trị trong khoảng : -9.223.370.036.854.775.808 đến 9.223.372.036.854.775.807
ulong	8	UInt64	Số nguyên không dấu từ 0 đến 0xffffffffffffffff

*Ghi chú: Kiểu giá trị logic chỉ có thể nhận được giá trị là true hay false mà thôi. Một giá trị nguyên không thể gán vào một biến kiểu logic trong C# và không có bất cứ chuyển đổi ngầm định nào. Điều này khác với C/C++, cho phép biến logic được gán giá trị nguyên, khi đó giá trị nguyên 0 là false và các giá trị còn lại là true.

2.3.1.2. Kiểu dữ liệu giá trị

Kiểu dữ liệu giá trị bao gồm các kiểu dữ liệu cơ sở định nghĩa sẵn, kiểu liệt kê và kiểu cấu trúc. Kiểu dữ liệu cấu trúc sẽ được tìm hiểu trong chương kế tiếp cùng với kiểu tham chiếu lớp và kiểu delegate

a. Kiểu cơ sở

Ngôn ngữ C# đưa ra các kiểu dữ liệu cơ sở xây dựng sẵn rất hữu dụng, phù hợp với một ngôn ngữ lập trình hiện đại, mỗi kiểu dữ liệu được ánh xạ đến một kiểu dữ liệu chung được hỗ trợ bởi Microsoft.NET. Việc ánh xạ các kiểu dữ liệu cơ sở của C# đến các kiểu dữ liệu của .NET sẽ đảm bảo các đối tượng được tạo ra trong C# có thể được sử dụng đồng thời với các đối tượng được tạo bởi bất kỳ ngôn ngữ khác được biên dịch bởi .NET.

Các kiểu dữ liệu cơ sở là kiểu dữ liệu struct xây dựng sẵn chứa nhiều thuộc tính và phương thức cần thiết, trong đó phương thức thường dùng là: Parse(string s): chuyển chuỗi s thành kiểu struct tương ứng

Ví dụ: float a = float.Parse(s);

Bảng 2.2: Một số các kiểu cơ sở

Kiểu	Mô tả	Độ lớn	Kiểu .Net	Phạm vi
bool	Kiểu logic	1 bit	Boolean	true, false
char	Ký tự Unicode	16 bits	Char	\u0000 ÷ \uFFFF
byte	Số nguyên	8 bits	Byte	0 ÷ 255
sbyte	Số nguyên	8 bits	Sbyte	-128 ÷ 127
short	Số nguyên	16 bits	Int16	-32768 ÷ 32767
ushort	Số nguyên	16 bits	Uint16	0 ÷ 65535
int	Số nguyên	32 bits	Int32	-2147483648 ÷ 2147483647
uint	Số nguyên	32 bits	Uint32	0 ÷ 4.294.967.295
long	Số nguyên	64 bits	Int64	-9223372036854775808 ÷ 9223372036854775807

ulong	Số nguyên	64 bits	UInt64	0 ÷ 0xffffffffffff ffffff
float	Số thực	32 bits	Single	3.4E-38 ÷ 3.4E+38
double	Số thực	64 bits	Double	1.7E-308 ÷ 1.7E+308
decimal	Số thực	64 bits	Decimal	độ chính xác đến 28 chữ số và giá trị thập phân, yêu cầu có hậu tố “m” hay “M” theo sau giá trị

b. Kiểu liệt kê

Kiểu liệt kê đơn giản là tập hợp các tên hằng có giá trị không thay đổi, giúp bạn tổ chức dữ liệu khoa học hơn, mã được trong sáng dễ hiểu hơn.

Để khai báo một kiểu liệt kê ta thực hiện theo cú pháp sau:

```
enum Tên kiểu liệt kê [Kiểu cơ sở]
```

```
{
```

```
//danh sách các thành phần liệt kê
```

```
}
```

Một kiểu liệt kê bắt đầu với từ khóa **enum**, tiếp sau là một định danh cho kiểu liệt kê đó.

Danh sách liệt kê là các hằng được gán giá trị, và mỗi thành phần phải phân cách nhau dấu phẩy. Nếu chúng ta không khởi tạo giá trị cho các thành phần này thì chúng sẽ nhận các giá trị liên tiếp bắt đầu từ 0. Mỗi tên hằng trong kiểu liệt kê thuộc bất kỳ một kiểu dữ liệu cơ sở nguyên nào như int, short, long..., ngoại trừ kiểu ký tự.

Kiểu cơ sở chính là kiểu khai báo cho các thành phần liệt kê. Nếu bỏ qua thành phần này thì trình biên dịch sẽ gán giá trị mặc định là kiểu nguyên int. Khi đó, mỗi thành phần trong kiểu liệt kê tương ứng với một giá trị số nguyên.

Truy xuất giá trị của danh sách liệt kê, cần phải chuyển kiểu một cách tường minh

Ví dụ:

```
enum VisibleType
{
    None = 0,
    Hidden = 2,
    Visible = 4
};

// In ra tên và giá trị thành phần của enum

static void Main(string[] args)
{
    VisibleType visEnum = VisibleType.Hidden;
    Console.WriteLine(visEnum.ToString());
    Console.WriteLine((int)VisibleType.Hidden);
    Console.ReadKey();
}
```

2.3.1.3. Kiểu dữ liệu tham chiếu

a. Kiểu chuỗi

Kiểu dữ liệu chuỗi là kiểu tham chiếu xây dựng sẵn, lưu giữ một dãy ký tự. Chuỗi trong C# là không thay đổi, mỗi khi chuỗi thay đổi, một chuỗi mới được tạo ra để chứa chuỗi kết quả.

Tạo một đối tượng chuỗi : Nhiều string được tạo từ các hằng chuỗi. Khi trình dịch bắt gặp một chuỗi ký tự bao giữa cặp nháy kép, nó tạo ra một đối tượng chuỗi có giá trị là chuỗi này. Bạn có thể dùng hằng chuỗi ở bất kỳ đâu bạn dùng đối tượng string

Bạn có thể tạo đối tượng chuỗi như bất kỳ đối tượng khác, dùng từ khoá new

```
string s = new string();
```

```
string s = new string("Hello World");  
hay có thể viết :
```

```
string s = "Hello World";
```

Truy cập các ký tự của chuỗi bằng chỉ số và sử dụng dấu [] như mảng, chỉ số có giá trị bắt đầu từ 0, ví dụ s[0]

Các thuộc tính và phương thức thường dùng của string

- int Length : cho chiều dài chuỗi

```
int len = source.Length;  
int len = "Hello World".Length;
```

- bool Equals(string value)
- bool Equals(object obj)
- bool Equals(string value, StringComparison comparisionType): kiểm tra hai chuỗi có giá trị bằng nhau không, trả về True nếu 2 chuỗi có giá trị bằng nhau, ngược lại trả về False, hoặc có thể sử dụng dấu == để so sánh chuỗi

Ví dụ:

```
string s1 = "Hello";  
string s2 = s1;  
Console.WriteLine("s1 equals s2 : " + s1.Equals(s2));  
//So sánh chuỗi không phân biệt hoa thường  
Console.WriteLine(s1.Equals(s2,  
                           StringComparison.OrdinalIgnoreCase));  
Console.WriteLine("s1 == s2 : " + (s1 == s2));
```

Kết quả là:

s1 equals s2 : True

True

s1 == s2 : True

-
- int CompareTo(string value): so sánh 2 chuỗi, trả về giá trị
nếu < 0 : chuỗi nhỏ hơn value

nếu > 0 : chuỗi lớn hơn value

nếu $= 0$: chuỗi bằng value

- int IndexOf(char value), int IndexOf(string value): trả về vị trí tìm thấy đầu tiên của ký tự hay chuỗi value
- int IndexOf(char value, int startIndex), int IndexOf(string value, int startIndex): trả về vị trí tìm thấy đầu tiên của ký tự hay chuỗi value kể từ vị trí startIndex đến cuối chuỗi.
- int IndexOf(string value, StringComparison comparisionType): trả về vị trí tìm thấy đầu tiên của chuỗi value với kiểu so sánh phân biệt hoa thường hay không.
- int IndexOf(char value, int startIndex, int count), int IndexOf(string value, int startIndex, int count): trả về vị trí tìm thấy đầu tiên của ký tự hay chuỗi value trong count ký tự của chuỗi kể từ vị trí startIndex.
- int IndexOf(string value, int startIndex, StringComparison comparisionType)
- int IndexOf(string value, int startIndex, int count, StringComparison comparisionType).

Tương tự phương thức IndexOf, có 9 phương thức LastIndexOf sử dụng để trả về vị trí tìm thấy cuối cùng:

- string Substring(int startIndex) : trả về chuỗi con của một chuỗi bắt đầu từ vị trí startIndex đến cuối chuỗi.
- string Substring(int startIndex, int length): trả về chuỗi con gồm length ký tự của một chuỗi bắt đầu từ vị trí startIndex.

```
string org = "This is a test";
string result1 = org.Substring(10); //result1 = "test"
string result2 = org.Substring(8,1); //result2 = "a"
```

- string Replace(string oldValue, string newValue) : thay thế tất cả các lần xuất hiện của chuỗi oldValue bằng chuỗi newValue
- string Trim() : cắt bỏ khoảng trắng đầu và cuối chuỗi
- string TrimEnd(): cắt bỏ khoảng trắng cuối chuỗi
- string TrimStart(): cắt bỏ khoảng trắng đầu chuỗi

- string ToLower() : trả về chuỗi thường của chuỗi
- string ToUpper() : trả về chuỗi chữ hoa của chuỗi

```
string s = "This is a test";
string upper = s.ToUpper();
```

Toán tử “+”: để kết nối hai đối tượng string, hay một đối tượng string và một giá trị khác thành đối tượng string

Ví dụ:

```
string s1 = "two";
Console.WriteLine("one" + s1 + "three");
Console.WriteLine("Word v. " + 9 + 7);
```

b. Kiểu mảng

Mảng là một cấu trúc lưu giữ các thành phần cùng kiểu. Mỗi thành phần của mảng được truy xuất bởi chỉ số của nó trong mảng

C# cũng hỗ trợ mảng không kiểu bằng cách tạo mảng các đối tượng như sau:

```
object[] arr = { 1, "Hi there", 5.5 };
```

Mảng 1 chiều

– Khai báo một biến tham chiếu đến mảng

Khai báo một biến dùng để tham chiếu đến mảng các thành phần kiểu ArrayType, nhưng không có mảng nào thật sự tồn tại: *ArrayType[] ArrayName;*

Kiểu dữ liệu thành phần có thể là bất kỳ kiểu giá trị hay tham chiếu

– Tạo một mảng

Sử dụng toán tử new để cấp phát bộ nhớ cho các thành phần và gán mảng đến biến đã khai báo

```
ArrayName = new ArrayType[ArraySize];
ArraySize : là số thành phần của mảng
```

Ví dụ: `m = new int[10];` // tạo một mảng số nguyên

Có thể kết hợp khai báo biến mảng và tạo mảng như sau :

```
ArrayType[] ArrayName = new ArrayType[ArraySize];
```

Ví dụ:

```
int[] m = new int[10];
string[] myStrings = new string[3];
```

- **Truy xuất thành phần của mảng:** ArrayName[index]

index: chỉ vị trí của thành phần trong mảng cần truy xuất, có thể là giá trị, biến hay biểu thức, và có giá trị từ 0 đến ArraySize-1

Ví dụ :

- **Lấy kích thước mảng:** ArrayName.Length
- **Khai báo, cấp phát bộ nhớ và khởi tạo giá trị đầu của mảng**

Mảng có thể khởi tạo khi khai báo sử dụng toán tử new hay không. Mảng khởi tạo là danh sách các biểu thức cách nhau dấu phẩy, đặt trong ngoặc móc. Chiều dài mảng là số thành phần giữa hai dấu { và }

Ví dụ 1:

```
int[] numbers = new int[5] {1, 2, 3, 4, 5};

//Có thể bỏ qua kích thước của mảng như sau:

int[] numbers = new int[] {1, 2, 3, 4, 5};

string[] names = new string[] {"Xuan", "Ha", "Thu"};

//Cũng có thể bỏ qua toán tử new như sau:

int[] numbers = {1, 2, 3, 4, 5};

string[] names = {"Xuan", "Ha", "Thu"};

int[] monthDays = {31,28,31,30,31,30,31,31,30,31,30,31};
```

Ví dụ 2:

```
static void Main(string[] args)
{
    for (int i = 0; i < args.Length; i++)
        Console.WriteLine("Tham doi thu " + i + ": " + args[i]);
    Console.ReadLine();
}
```

Khi chạy chương trình :

```
csc Thamdoi.cs ↵
ThamDoi Thu tham doi dong lenh ↵
Cho kết quả:      Tham doi thu 0 : Thu
                  Tham doi thu 1 : tham ...
```

Mảng đa chiều (Arrays of Arrays)

Mảng có thể chứa các thành phần là mảng.

- *Khai báo, cấp phát bộ nhớ mảng*

Để khai báo một biến mảng đa chiều cần xác định mỗi chiều của mảng bằng cách sử dụng các cặp dấu ngoặc vuông.

Ví dụ: *Khai báo mảng 2 chiều m là một mảng 4 dòng, và có thành phần là các số nguyên.*

```
int[][] m = new int[4][];
m[0] = new int[2]; //dòng 1 có 2 cột
m[1] = new int[5]; //dòng 2 có 5 cột
Console.WriteLine(m.Length); //số dòng của mảng là 4
```

Hay có thể khai báo như sau:

```
double[,] myDoubles = new double[2, 2];
int[, ,] myInts = new int[4, 5, 3];
```

– **Truy xuất thành phần của mảng**

$m[i][j]$: phần tử tại dòng i , cột j

với i là chỉ số dòng, có giá trị từ 0 đến số dòng -1

j là chỉ số cột, có giá trị từ 0 đến số cột - 1

Hay $m[i,j]$

– **Lấy kích thước mảng**

- Với mảng khai báo như sau:

```
int[][] m = new int[4][];
```

Kích thước của chiều thứ nhất, nghĩa là số dòng: $m.Length$

Kích thước của chiều thứ hai, nghĩa là số cột. Số phần tử trên dòng có thể khác nhau trên mỗi dòng. Ta có kích thước của dòng thứ i

$m[i].Length$

- Với mảng khai báo như sau:

```
int[,] m = new int[4, 5];
```

Số tất cả các phần tử mảng ở tất cả các chiều: $m.Length$

Kích thước của chiều thứ nhất, nghĩa là số dòng: $m.GetLength(0)$;

Kích thước của chiều thứ hai, nghĩa là số cột: $m.GetLength(1)$;

– **Khai báo, cấp phát bộ nhớ và khởi tạo giá trị đầu của mảng**

Ví dụ:

```
int[,] numbers = new int[3, 2] { { 1, 2 }, { 3, 4 }, { 5, 6 } };
int[][] numbers = new int[2][] { new int[] { 2, 3, 4 }, new int[]
{ 5, 6, 7, 8, 9 } };
string[,] nhom = new string[2, 2] { { "Hoa", "Thu" }, { "Hue",
"Cuc" } };
```

Có thể bỏ qua kích thước của mảng:

```
int[,] numbers = new int[,] { { 1, 2 }, { 3, 4 }, { 5, 6 } };
int[][] numbers = new int[][] { new int[] { 2, 3, 4 }, new int[]
{ 5, 6, 7, 8, 9 } };
int[][], numbers = { new int[] { 2, 3, 4 }, new int[] { 5, 6, 7,
8, 9 } };
```

```
string[,] nhom = new string[,] { { "Hoa", "Thu" }, { "Hue", "Cuc" } };
```

Cũng có thể bỏ qua toán tử new như sau:

```
int[,] numbers = { { 1, 2 }, { 3, 4 }, { 5, 6 } };
string[,] nhom = { { "Hoa", "Thu" }, { "Hue", "Cuc" } };
```

Ví dụ:

```
static void Main(string[] args)
{
    int[] myInts = { 5, 10, 15 };
    Console.WriteLine("myInts[0]: {0}, myInts[1]: {1}, myInts[2]: {2}",
                      myInts[0], myInts[1], myInts[2]);
    bool[][] myBools = new bool[2][];
    myBools[0] = new bool[3];
    myBools[1] = new bool[2];
    myBools[0][0] = true;
    myBools[0][1] = false;
    myBools[1][0] = true;
    Console.WriteLine("myBools[0][0]: {0}, myBools[1][0]: {1} ",
                      myBools[0][0], myBools[1][0]);
    double[,] myDoubles = new double[2, 2];
    myDoubles[0, 0] = 3.147;
    myDoubles[0, 1] = 7.157;
    myDoubles[1, 0] = 2.117;
    myDoubles[1, 1] = 56.00138917;
    Console.WriteLine("myDoubles[0, 0]: {0}, myDoubles[1, 0]: {1} ",
                      myDoubles[0, 0], myDoubles[1, 0]);
    string[] myStrings = new string[3];
    myStrings[0] = "Joe";
    myStrings[1] = "Matt";
    myStrings[2] = "Robert";
    Console.WriteLine("myStrings[0]: {0}, myStrings[1]: {1},
                      MyStrings [2]: {2}", myStrings[0], myStrings[1],
                      myStrings[2]);
    Console.ReadKey();
}
```

Ví dụ: Tạo ma trận các phần tử số nguyên có giá trị bằng chỉ số hàng + chỉ số cột của phần tử. In ma trận

```
static void Main(string[] args)
{
    int[][] a = new int[4][];
    for (int i = 0; i < a.Length; i++)
        a[i] = new int[5];
    //Nhập ma trận
    for (int i = 0; i < a.Length; i++)
        for (int j = 0; j < a[i].Length; j++)
            a[i][j] = i + j;
    //In ma trận
    for (int i = 0; i < a.Length; i++)
    {
        for (int j = 0; j < a[i].Length; j++)
            Console.Write(a[i][j] + "\t");
        Console.WriteLine();
    }
    Console.ReadKey();
}
```

Ví dụ: Viết lại ví dụ trên: tạo ma trận các phần tử số nguyên có giá trị bằng chỉ số hàng + chỉ số cột của phần tử. In ma trận

```
static void Main(string[] args)
{
    int[,] a = new int[4, 5];
    //Nhập ma trận
```

```
        for (int i = 0; i < a.GetLength(0); i++)
            for (int j = 0; j < a.GetLength(1); j++)
                a[i, j] = i + j;

        //In ma trận
        for (int i = 0; i < a.GetLength(0); i++)
        {
            for (int j = 0; j < a.GetLength(1); j++)
                Console.Write(a[i, j] + "\t");
            Console.WriteLine();
        }
        Console.ReadKey();
    }
```

c. Kiểu con trỏ

C# cũng hỗ trợ kiểu con trỏ như C++, nhưng hiếm khi được sử dụng, và chỉ sử dụng khi làm việc với những đoạn mã lệnh không được quản lý (unmanaged code). Mã lệnh không được quản lý là các lệnh được viết bên ngoài nền MS.Net, như là các đối tượng COM.

Khai báo biến kiểu con trỏ như sau:

type* identifier;

* identifier; //cho phép nhưng không nên dùng

Kiểu con trỏ có thể là kiểu sau : sbyte, byte, short, ushort, int, uint, long, ulong, char, float, double, decimal, or bool, enum, pointer, struct chứa các trường kiểu con trỏ

Biến kiểu con trỏ kiểu type* không chứa giá trị mà chứa tham chiếu đến biến kiểu type hay nói cách khác giá trị của biến con trỏ là địa chỉ của biến kiểu type

Ví dụ	Mô tả
int* p	p là một con trỏ tham chiếu đến biến nguyên
int** p	p là một con trỏ tham chiếu đến con trỏ đến biến nguyên

int*[] p	p là một mảng của các con trỏ tham chiếu đến biến nguyên
char* p	p là con trỏ tham chiếu đến biến char
void* p	p là con trỏ tham chiếu đến biến kiểu void

Khi bạn khai báo nhiều con trỏ cùng kiểu, chỉ sử dụng một dấu * cho nhiều biến của cùng một khai báo, không sử dụng mỗi dấu * cho mỗi biến :

int* p1, p2, p3; // đúng với C#

int *p1, *p2, *p3; // không đúng với C#

Toán tử * có thể được sử dụng để truy cập giá trị của biến tại vị trí được trỏ đến bởi biến con trỏ. Bạn không thể sử dụng toán tử * cho con trỏ kiểu void*, tuy nhiên có thể chuyển kiểu con trỏ void* đến bất kỳ kiểu con trỏ khác.

Bảng 2.3: Các phép toán có thể thực hiện trên biến con trỏ

Phép toán	Ý nghĩa
*	lấy giá trị của biến mà con trỏ tham chiếu đến
->	truy cập thành phần của cấu trúc struct được tham chiếu bởi con trỏ
[]	truy cập thành phần của mảng được tham chiếu bởi con trỏ
&	lấy địa chỉ của biến
++ and --	toán tử tăng và giảm trên biến con trỏ
+ and -	thực hiện phép toán số học trên biến con trỏ
==, !=, <, >, <=, and >=	để so sánh biến con trỏ
stackalloc	Định vị bộ nhớ trên stack

Ví dụ:

```
static void Main()
{
    char theChar = 'Z';
```

```
char* pChar = &theChar;
void* pVoid = pChar;
int* pInt = (int*)pVoid;
System.Console.WriteLine("Giá trị của biến theChar = {0}",
                         theChar);
System.Console.WriteLine("Địa chỉ của biến theChar = {0:X2}",
                         (int)pChar);
System.Console.WriteLine("Giá trị của biến pChar = {0}",
                         *pChar);
System.Console.WriteLine("Giá trị của biến pInt = {0}",
                         *pInt);
}
```

Ví dụ:

```
static void Main()
{
    int number;
    unsafe
    {
        // Gán địa chỉ của biến number đến con trỏ p
        int* p = &number;
        // Khởi tạo number
        *p = 0xffff;
        // Xuất giá trị của biến con trỏ p
        System.Console.WriteLine("Value at the location
                                 pointed to by p: {0:X}", *p);
        // Xuất địa chỉ lưu trữ trong con trỏ p
        System.Console.WriteLine("The address stored in p:
                                 {0}", p->ToString());
    }
    // Xuất giá trị của biến number
```

```
System.Console.WriteLine("Value of the variable number:  
{0:X}", number);  
}
```

Ví dụ:

```
struct CoOrds  
{  
    public int x;  
    public int y;  
}  
class AccessMembers  
{  
    static void Main()  
    {  
        CoOrds home;  
        unsafe  
        {  
            CoOrds* p = &home;  
            p->x = 25;  
            p->y = 12;  
            System.Console.WriteLine("The coordinates are: x={0},  
y={1}", p->x, p->y);  
        }  
    }  
}
```

Ví dụ:

Trong ví dụ sau, chú ý rằng biến thức charPointer[i] tương đương với biến thức *(charPointer + i)

```
class Pointers  
{
```

```
unsafe static void Main()
{
    char* charPointer = stackalloc char[123];
    for (int i = 65; i < 123; i++)
    {
        charPointer[i] = (char)i;
    }
    // In các ký tự hoa
    System.Console.WriteLine("Uppercase letters:");
    for (int i = 65; i < 91; i++)
    {
        System.Console.Write(charPointer[i]);
    }
    System.Console.WriteLine();
    // In các ký tự thường
    System.Console.WriteLine("Lowercase letters:");
    for (int i = 97; i < 123; i++)
    {
        System.Console.Write(charPointer[i]);
    }
}
```

Ví dụ:

```
class ClassConvert
{
    static void Main()
    {
        int number = 1024;
```

```
unsafe
{
    byte* p = (byte*)&number;
    System.Console.WriteLine("The 4 bytes of the integer:");
    for (int i = 0; i < sizeof(int); ++i)
    {
        System.Console.Write(" {0:X2}", *p);
        p++;
    }
    System.Console.WriteLine();
    System.Console.WriteLine("The value of the integer:
{0}", number);
}
}
```

2.4. Biến (Variable)

Là một vùng nhớ được định danh để lưu trữ dữ liệu hay địa chỉ dữ liệu. Mọi biến cần được định nghĩa tức là khai báo tên và kiểu dữ liệu trước khi sử dụng. Biến phải được khởi tạo trước khi sử dụng.

Khai báo biến thuộc kiểu dữ liệu cơ bản:

Kiểu TênBiến;

hay có thể khai báo và khởi tạo giá trị đầu cho biến:

Kiểu TênBiến = GiáTrị;

Nhiều biến cùng kiểu có thể khai báo cách nhau dấu phẩy, bạn có thể định nghĩa biến kiểu tham chiếu string như trên.

Ví dụ:

```
string s = "Hello"; // String s = "Hello";
int x = 5, y = 7; // Int32 x = 5, y=7;
bool b = true; // Boolean b = true;
```

Biến kiểu không tường minh (Implicitly typed local variable):

C# 3.0 bổ sung từ khóa var cho phép định nghĩa biến kiểu không tường minh (implicitly typed local variable). Chú ý giá trị biến phải được gán khi khai báo biến, khi thực hiện lệnh gán, biến đã được ngầm định khai báo kiểu tương ứng. Hơn nữa, giá trị gán cho biến phải khác null.

Ví dụ: var i = 1;

Kiểu dynamic và ràng buộc trễ (dynamic type and late binding):

C# 4.0 giới thiệu một kiểu mới, kiểu dynamic. Không như kiểu ràng buộc tĩnh (static binding), các biến khai báo kiểu dynamic ràng buộc trễ vào lúc chạy chương trình, nghĩa là khi biên dịch, CLR không biết kiểu thực sự của biến, vì vậy nó hỗ trợ bất kỳ thao tác vào lúc biên dịch, có thể gán giá trị, đổi tượng kiểu bất kỳ cho biến dynamic. Không giống như var, có thể khai báo biến dynamic, sau đó chúng ta gán giá trị cho biến.

Ví dụ:

```
dynamic d; d = 7;
//Biên dịch không lỗi

//Báo lỗi vào lúc chạy do d là một int, không định nghĩa thuộc
tính Length

Console.WriteLine(d.Length); d = "a string";

Console.WriteLine(d.Length); d = System.DateTime.Today;
```

2.5. Hằng

Hằng có hai loại: hằng giá trị (literal) và hằng được đặt tên (constant). Hằng đặt tên là một vùng nhớ để lưu trữ dữ liệu hay địa chỉ dữ liệu không thay đổi giá trị trong chương trình: const Kiểu TênHằng = GiáTrị;

Ví dụ: const int max = 10;

Hằng giá trị là các giá trị thuộc kiểu nào đó.

C# xem hằng nguyên thuộc kiểu int, trừ khi trị nguyên vượt quá phạm vi kiểu int, khi đó hằng nguyên thuộc kiểu long. Muốn có hằng nguyên nhỏ thuộc kiểu long, bạn thêm L hay L vào cuối trị số. Hằng nguyên có thể được gán cho biến thuộc kiểu byte hay short miễn là trị nguyên được gán không vượt quá phạm vi của các kiểu này.

Ví dụ: 25L

Hằng số thực được C# mặc nhiên hiểu là trị thuộc kiểu double, bạn có thể ghi thêm D hay F (d hay f) ở cuối trị số để khẳng định rõ trị số thuộc kiểu double hay float. Hằng số thực có thể chứa ký tự E hay e.

Ví dụ: 6.5, 6.5F, 12.36E-2

Hằng ký tự được ghi trong cặp dấu nháy đơn.

Ví dụ: 'a', '5'

Ký tự đặc biệt được thêm dấu \ gọi là ký tự thoát (escape character) phía trước.

Bảng 2.4. Các ký tự đặc biệt

Ký tự đặc biệt	Ý nghĩa
\n	Ký tự xuống dòng
\t	Ký tự tab ngang
\v	Ký tự tab dọc
\b	Ký tự xoá trái (backspace)
\r	Ký tự về đầu dòng (carriage return)
\f	Ký tự qua trang (formfeed)

\\"	Ký tự xô trái (backslash)
\'	Ký tự nháy đơn (single quote)
\\""	Ký tự nháy kép (double quote)
\xddd	Ký tự ứng với mã ASCII dạng bát phân ddd
\udddd	Ký tự ứng với mã Unicode dddd dạng thập lục phân
\0	Ký tự null
\a	Ký tự chuông

Hằng kiểu logic có giá trị true, false. Hằng chuỗi diễn đạt một chuỗi ký tự kiểu string, được ghi trong cặp dấu nháy kép.

Ví dụ:

```
Console.WriteLine("Phương trình có 2 nghiệm:\n\tx1=" + x1 + "\n\tx2=" + x2);
```

C# phiên bản trước chỉ cho phép kiểu tham chiếu có giá trị null là tham chiếu rỗng. C# 2.0 bổ sung hằng null cho kiểu giá trị có kiểu Nullable nghĩa là không có giá trị.

2.6. Biểu thức và toán tử

2.6.1. Biểu thức (Expression)

Biểu thức là một công thức tính toán một giá trị theo quy tắc toán học, cấu tạo từ các toán hạng (operand): hằng, phương thức, biến và nối kết với nhau bởi các toán tử (operator). Các toán hạng trong mọi biểu thức phải tương thích với nhau về kiểu. Một biểu thức có thể là biểu thức số học, logic, ký tự, chuỗi khi kết quả của biểu thức có kiểu tương ứng.

Ví dụ: Biểu thức đơn giản nhất là biến hay hằng

a

"Hello" true

2.6.2. Toán tử (Operator)

Toán tử là phép toán dùng để kết hợp các toán hạng, dùng trong các biểu thức.

2.6.2.1. Các toán tử số học (Arithmetic operator)

Phép toán số học sử dụng trên các toán hạng có kiểu số và trả về kết quả kiểu số:

Toán tử	Ý nghĩa	Cách dùng
+	Cộng hai toán hạng	op1 + op2
-	Trừ	op1 - op2
*	Nhân	op1 * op2
/	Chia	op1 / op2
%	Chia lấy số dư	op1 % op2

Với (op1, op2 là các toán hạng kiểu số)

Với phép chia op1/op2, nếu op1, op2 đều là số nguyên thì phép chia này cho kết quả là số nguyên, nếu một trong hai số op1, op2 hay cả hai số đều là số thực sẽ cho kết quả số thực.

Ví dụ:

```
static void Main(string[] args)
{
    int i = 67, j = 22;
    double x = 27.475, y = 7.22;
    Console.WriteLine("\ti / j = " + i / j);
    Console.WriteLine("\ti + j = " + (i + j));
    Console.WriteLine("\tx - y = " + (x - y));
    Console.WriteLine("\tx / y = {0}, \n\ti % j = {1}", x /
y, i % j);
}
```

2.6.2.2. Các toán tử tăng, giảm (Increment, decrement operator)

Phép toán tăng giảm sử dụng cho các biến kiểu số nguyên và kiểu ký tự:

- op ++ hay ++ op: Tăng biến số nguyên lên một đơn vị, hay cho ký tự Unicode kế tiếp

- op -- hay -- op: Giảm biến số nguyên xuống một đơn vị, hay cho ký tự Unicode kế trước.

Toán tử tăng, giảm có thể đặt ở trước hay sau biến để diễn đạt ý nghĩa khác nhau.

Ví dụ:

```
int x = 4, y;  
  
y = x++; //Giá trị x gán cho y trước, sau đó tăng lên một đơn vị, x=5, y=4  
  
y = ++x; //Giá trị x tăng lên một đơn vị, sau đó gán cho y, x=6, y=6  
  
x++; //x = 7  
  
Console.WriteLine(++x); //Tăng x lên một đơn vị sau đó in ra 8  
  
Console.WriteLine(x--); //In ra 8 sau đó giảm x đi 1
```

Ví dụ:

```
char c = "a"; c++; //c="b"
```

2.6.2.3. Toán tử quan hệ hay so sánh (Comparison operator)

Phép toán so sánh áp dụng trên các kiểu dữ liệu cơ bản và luôn cho kết quả có kiểu logic.

Toán tử	Ý nghĩa	Cách dùng
==	Bằng	op1 == op2
!=	Không bằng	op1 != op2
<	Nhỏ hơn	op1 < op2
>	Lớn hơn	op1 > op2
<=	Nhỏ hơn hoặc bằng	op1 <= op2
>=	Lớn hơn hoặc bằng	op1 >= op2

Ví dụ: Phép so sánh bằng có thể được sử dụng để so sánh số, ký tự hay chuỗi

155 > 25

4.5 < 8 ‘B’ == ‘A’

‘B’ < ‘A’

2.6.2.4. Toán tử logic (Logical operator)

Phép toán logic chỉ áp dụng cho các toán hạng kiểu logic, cho kết quả kiểu logic

Toán tử	Ý nghĩa	Cách dùng
& hay &&	Và (And)	op1 && op2
hay	Hoặc (Or)	op1 op2
^	Hoặc loại trừ (Xor)	op1 ^ op2
!	Phủ định (Not)	! op

Phép & hay && cho kết quả true chỉ khi 2 toán hạng op1,

op2 có giá trị true

Phép | hay || cho kết quả là false chỉ khi 2 toán hạng op1, op2 đều có giá trị false

Phép ^ cho kết quả là true chỉ khi 2 toán hạng op1, op2 khác giá trị nhau

Phép ! cho kết quả là true nếu op có trị false, và ngược lại

Ví dụ:

100 > 55 && ‘B’ < ‘A’

Diem >= 5 && Diem < 7 C == ‘Y’ || C == ‘y’

Ví dụ:

x > 3 && x < ++y

Phép & khác với phép &&: với phép && chỉ cần một toán hạng bên trái: x > 3 trả về false, lập tức kết quả phép toán trả về false, C# không tính đến toán hạng bên phải x < ++y, nghĩa là y không được tăng lên một đơn vị.

Tương tự với phép | và phép ||, ở phép || chỉ cần một toán hạng bên trái trả về true, toán hạng bên phải sẽ được bỏ qua không tính đến.

2.6.2.5. Các toán tử làm việc với bit

Là các toán tử làm việc trên từng bit của số nguyên:

Toán tử	Ý nghĩa	Cách dùng
&	Và (And)	op1 & op2
	Hoặc (Or)	op1 op2
^	Hoặc loại trừ (Xor)	op1 ^ op2
<<	Dịch trái n bit	op << n
>>	Dịch phải n bit và điền 0 vào những chỗ vừa dịch	op >> n
□	Đảo bit (Not)	~ op

Ví dụ:

0	0	0	0
---	---	---	---

0	1	0	1
---	---	---	---

5

0	0	0	0
---	---	---	---

0	0	0	1
---	---	---	---

$5 >> 2 = 1$

Phép dịch phải đi n bit cũng tương đương với phép chia cho 2^n

$$5 >> 2 = 5 / 2^2 = 1$$

$$16 >> 3 = 16 / 2^3 = 2$$

Khi dịch phải bít với số dương, bít bên trái sẽ điền 0, khi dịch phải bít với số âm, bít dấu bên trái vẫn giữ là 1

1	1	1	1
---	---	---	---

1	1	0	0
---	---	---	---

-4

1	1	1	1
---	---	---	---

1	1	1	0
---	---	---	---

$-4 >> 1 = -2$

Phép dịch trái đi n bit cũng tương đương với phép nhân cho 2^n

<table border="1" style="border-collapse: collapse; width: 100px;"> <tr><td>0</td><td>0</td><td>0</td><td>0</td></tr> </table>	0	0	0	0	<table border="1" style="border-collapse: collapse; width: 100px;"> <tr><td>0</td><td>1</td><td>0</td><td>0</td></tr> </table>	0	1	0	0	4
0	0	0	0							
0	1	0	0							
<table border="1" style="border-collapse: collapse; width: 100px;"> <tr><td>0</td><td>0</td><td>0</td><td>1</td></tr> </table>	0	0	0	1	<table border="1" style="border-collapse: collapse; width: 100px;"> <tr><td>0</td><td>0</td><td>0</td><td>0</td></tr> </table>	0	0	0	0	4 << 2 = 16
0	0	0	1							
0	0	0	0							

$$4 << 2 = 4 * 2^2 = 16$$

Phép dịch phải không dấu đi n bit sẽ điền 0 vào các bít bên trái cho cả số dương và âm :

2 biểu diễn nhị phân là: 00000000 00000000 00000000 00000010

-2 biểu diễn nhị phân là: 11111111 11111111 11111111 11111110

-2 >>>24 = 255: 00000000 00000000 00000000 11111111

2.6.2.6. Toán tử gán (Assignment Operator)

Toán tử gán = dùng để gán giá trị của một biểu thức Exp cho một biến Var. Phép toán gán trả về giá trị là giá trị của biểu thức Exp

Var = Exp;

Ví dụ : x = 10;

Có thể dùng toán tử gán nhiều lần liên tiếp trong câu lệnh.

Ví dụ: x = y = z = 0; //Cả ba biến x, y, z đều được gán giá trị 0

C# còn cung cấp các toán tử rút gọn (Shorthand Assignment Operator) như sau:

Toán tử	Ý nghĩa	Cách dùng
+=	Cộng rồi gán	var = var + Exp
-=	Trừ rồi gán	var = var - Exp
*=?	Nhân rồi gán	var = var * Exp
/=	Chia rồi gán	var = var / Exp
%=?	Chia lấy phần dư rồi gán	var = var % Exp
&=?	And từng bit rồi gán	var = var & Exp
=	Or từng bit rồi gán	var = Exp
^=?	Xor từng bit rồi gán	var ^= Exp
<<=?	Dịch trái rồi gán	var <<= Exp

>>=	Dịch phải rồi gán var >>= Exp	var >>= Exp
>>>=	Dịch phải, điền 0 vào những chỗ vừa dịch rồi gán var >>>= Exp	var >>>= Exp

Ví dụ: int a = 2;

```
a += 4;           //Tương đương với a = a + 4;
System.Console.WriteLine(a);
char c = 'A';
System.Console.WriteLine(c + 4);      //in ra 69
System.Console.WriteLine((char)(c + 4)); //in ra ký tự E
```

2.6.2.7. Một số toán tử khác

Toán tử	Ý nghĩa	Cách dùng
?:	op1 ? op2 : op3	Phép toán điều kiện: nếu toán hạng logic op1 có trị true thì trả về giá trị op2. Ngược lại trả về giá trị op3.
[]	op1[op2]	Truy cập đến một phần tử của mảng op1 với chỉ số op2
.	op1.op2	Truy cập đến biến và phương thức op2 của đối tượng hay lớp op1
()	op(params)	Định nghĩa hoặc gọi một phương thức có tên là op với các tham đối là params
(type)	(type) op	Chuyển đổi kiểu dữ liệu của toán hạng op sang kiểu type
new		Khởi tạo một đối tượng hay một mảng
is	op1 is op2	Phép toán cài đặt trả về true nếu đối tượng op1 là một đối tượng (thể hiện) của lớp op2
+	op1 + op2 +...	Ghép nối các chuỗi, nếu có toán hạng nào không phải chuỗi, C# tự động chuyển sang chuỗi

Ví dụ 1:

```
int a = 2, b = 5;
System.Console.WriteLine(a > b ? a : b);
```

```
System.Console.WriteLine(a > b ? "So a lon hon" : "So b lon
hon");
```

Ví dụ 2:

```
System.Console.WriteLine(" 2 + 2 = " + (2 + 2));
```

2.6.2.8. Độ ưu tiên các phép toán

Độ ưu tiên của các phép toán trong biểu thức thực hiện theo thứ tự từ trên xuống như trong bảng sau. Bạn có thể chủ động quy định thứ tự thực hiện phép toán trong biểu thức bằng những cặp ngoặc tròn (), trong ngoặc sẽ thực hiện trước, và ngoặc trong thực hiện trước, ngoặc ngoài thực hiện sau. Các phép toán trên cùng hàng sẽ có cùng độ ưu tiên.

Toán tử	Ghi chú	Quy tắc kết hợp
(), [], .	Dấu . để truy xuất biến, phương thức của đối tượng, lớp, [] truy xuất phần tử của mảng	Trái sang phải
++, --, +op, -op, !, ~	Dấu +op, -op là dấu dương, âm, ! là phép phủ định logic, ~ là đảo bít	Phải sang trái
new, (type) op	(type) op là phép chuyển đổi kiểu	Phải sang trái
*, /, %	Các phép toán số học	Trái sang phải
+, -		Trái sang phải
<<, >>, >>>	Phép toán dịch chuyển bit	Trái sang phải
<, >, <=, >=, is	Các phép toán quan hệ	Trái sang phải
==, !=		Trái sang phải
&	Các phép toán trên từng bit	Trái sang phải
^		Trái sang phải
		Trái sang phải
&&	Các phép toán logic	Trái sang phải
		Trái sang phải
? :	Phép toán điều kiện	Phải sang trái
=, +=, -=, *=, /=, %=, ^=, &=,	Các phép toán gán	Phải sang trái

=, <<=, >>=, >>>=		
-------------------	--	--

Ví dụ:

s += a += b += c; //Thứ tự thực hiện phép gán từ phải sang trái

2.7. Lệnh (Statement)

Lệnh có thể là lệnh đơn, khối lệnh hay cấu trúc điều khiển

2.7.1. Lệnh đơn

Lệnh đơn là thành phần cơ bản nhất trong chương trình C#, diễn đạt một thao tác riêng lẻ nào đó phải làm, được kết thúc bằng dấu chấm phẩy.

Lệnh đơn có thể là lời gọi phương thức, phép gán, phép tăng, giảm giá trị biến, các lệnh định nghĩa, khai báo

Ví dụ:

```
int i = 1;  
System.Console.WriteLine("Đây là một hằng chuỗi");  
x++;
```

2.7.2. Khối lệnh

Nhiều câu lệnh đơn có thể nhóm lại thành một câu lệnh phức hợp gọi là khối lệnh. Khối lệnh được mở đầu bằng dấu { và kết thúc bằng dấu }, có thể đặt trong một khối lệnh khác.

2.7.3. Cấu trúc rẽ nhánh

2.7.3.1. Cấu trúc IF

If là câu lệnh lựa chọn cho phép chương trình rẽ nhánh thực hiện lệnh theo hai hướng khác nhau căn cứ trên giá trị true, false của biểu thức điều kiện kiểu logic

Cú pháp:

Dạng 1: if (Biểu thức điều kiện)

Lệnh;

Dạng 2: if (Biểu thức điều kiện)

Lệnh 1;

else Lệnh 2;

Trong đó:

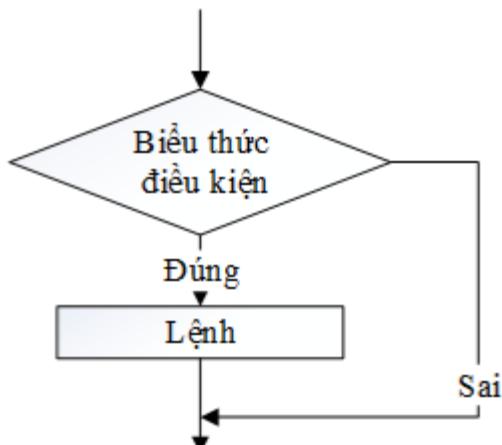
- Biểu thức điều kiện (*Boolean-Expression*) có kiểu logic bool.
- Lệnh (Statement) có thể là lệnh đơn, khối lệnh, hay cấu trúc điều khiển

Ý nghĩa

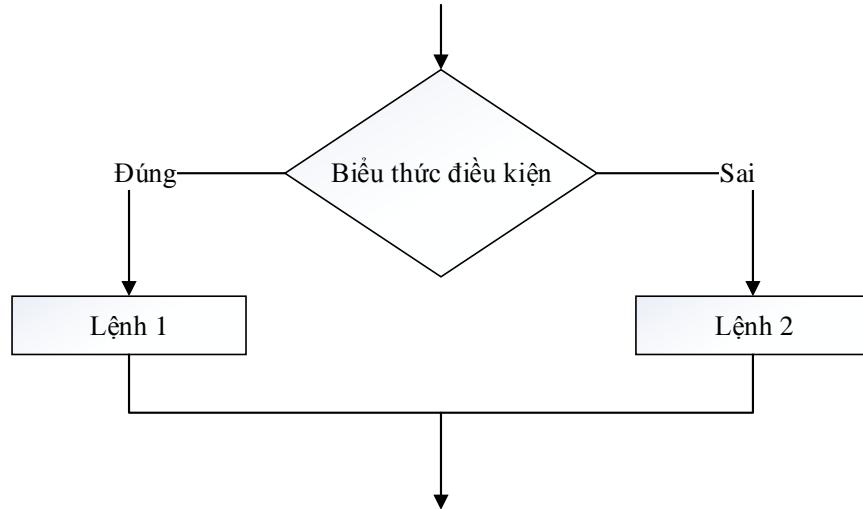
Nếu biểu thức điều kiện thỏa mãn (có giá trị true) thì lệnh 1 thực hiện, ngược lại nếu biểu thức điều kiện không thỏa mãn (có giá trị false) thì không làm gì cả (với dạng 1) hay thực hiện lệnh 2 (với dạng 2)

Sơ đồ khối

Dạng 1:



Dạng 2:



Ví dụ: Giải phương trình bậc 2: $ax^2 + bx + c = 0$

Cách giải:

```
using System;

namespace PTB2
{
    class PTB2
    {
        static void Main(string[] args)
        {
            Console.Write("Nhập hệ số a:");
            string s = Console.ReadLine(); double a =
double.Parse(s);

            Console.Write("Nhập hệ số b:");
            s = Console.ReadLine(); double b = double.Parse(s);
            Console.Write("Nhập hệ số c:");
        }
    }
}
```

```
s = Console.ReadLine(); double c = double.Parse(s);

if (a == 0)

    if (b == 0)

        if (c == 0) s = "co vo so nghiem";

        else s = "vo nghanem";

    else s = "co mot nghanem la " + (-c / b);

else

{

    double delta = b * b - 4 * a * c;

    if (delta < 0) s = "vo nghanem";

    else if (delta == 0) s = "co nghanem kep la " + (-b / (2 * a));

    else s = "co 2 nghanem x1= " + ((-b +
Math.Sqrt(delta)) / (2 * a)) +
                    " va x2= " + ((-b -
Math.Sqrt(delta)) / (2 * a));

}

Console.WriteLine("Phuong trinh " + s);

Console.ReadKey();

}

}
```

2.7.3.2. Cấu trúc Switch

Switch là câu lệnh lựa chọn, cho phép rẽ nhánh thực hiện lệnh theo nhiều hướng khác nhau căn cứ trên giá trị của một biểu thức

Cú pháp:

switch (BiểuThứcĐiềuKiện)

{

```
case Biểu thức 1: Lệnh 1;  
break;  
  
...  
  
case Biểu thức n: Lệnh n;  
break;  
  
default:  
  
    Lệnh n + 1;  
  
    break;  
  
}
```

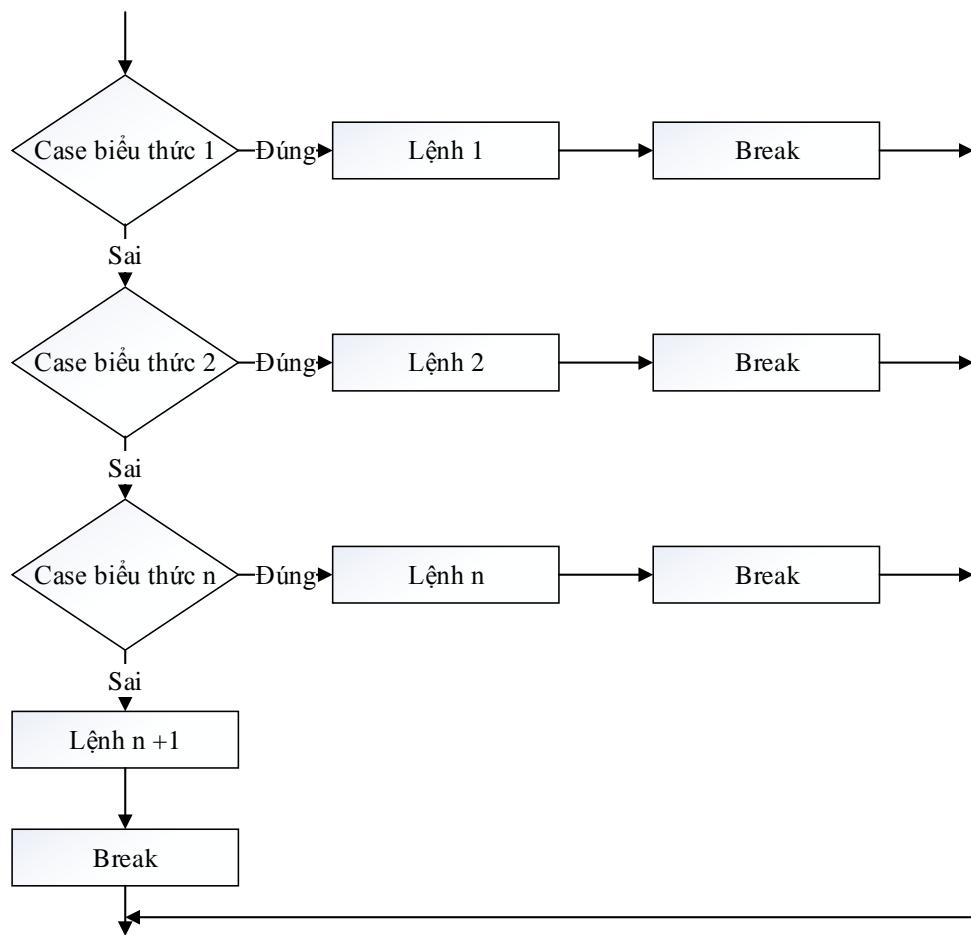
Trong đó:

- Biểu thức điều kiện: Có kiểu nguyên hay chuỗi
- Lệnh1, ...Lệnh n+1: Có thể là nhiều lệnh đơn, hay cấu trúc điều khiển
- Biểu thức 1, ...Biểu thức n: Các biểu thức hằng (toán hạng là các giá trị hay hằng)

Ý nghĩa:

Thực hiện lệnh 1, lệnh 2, ...lệnh n khi biểu thức có giá trị bằng biểu thức 1, biểu thức 2, ...biểu thức n tương ứng, nếu biểu thức không bằng các giá trị đó thì lệnh n+1 theo sau default được thực hiện (nếu có default). Câu lệnh break làm cho chương trình có thể thoát khỏi lệnh switch, nếu không có break, C# sẽ thông báo lỗi, nhưng đặc biệt C# cho phép case rỗng

Sơ đồ khối:



Ví dụ: Nhập vào một số nguyên từ 2 đến 8 và in ra thứ tương ứng: ví dụ nhập vào số 2 in ra “Thứ Hai”.

Cách giải

```

using System;
class Thu
{
    static void Main(string[] args)
    {
        Console.WriteLine("Nhập vào một số từ 2 đến 8");
        int thu = int.Parse(Console.ReadLine());
        switch (thu)
    
```

```
{  
    case 2: Console.WriteLine("Thu Hai"); break;  
    case 3: Console.WriteLine("Thu Ba"); break;  
    case 4: Console.WriteLine("Thu Tu"); break;  
    case 5: Console.WriteLine("Thu Nam"); break;  
    case 6: Console.WriteLine("Thu Sau"); break;  
    case 7: Console.WriteLine("Thu Bay"); break;  
    case 8: Console.WriteLine("Chu Nhat"); break;  
    default: Console.WriteLine("So ban vua nhap khong  
dung"); break;  
}  
Console.ReadLine();  
}  
}
```

2.7.4. Cấu trúc lặp

Ngoài for, while và do... while, C# còn bổ sung thêm cấu trúc lặp foreach.

2.7.4.1. Cấu trúc For

Cú pháp:

for (BiểuThứcKhởiTạo; BiểuThứcĐiềuKiện; BiểuThứcTăng)

Lệnh;

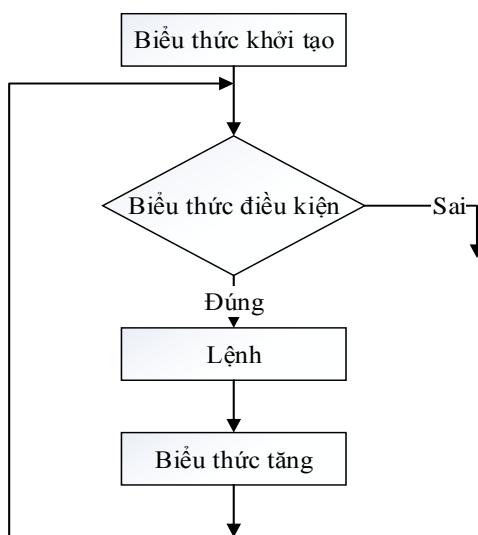
Trong đó:

- Lệnh (Statement) có thể là lệnh đơn, khối lệnh, hay cấu trúc điều khiển.
- Biểu thức khởi tạo: là các biểu thức khởi tạo giá trị cho các biến điều khiển vòng lặp, cách nhau dấu phẩy.
- Biểu thức điều kiện: có kiểu logic bool, thường là biểu thức so sánh giá trị của biến điều khiển với giá trị kết thúc vòng lặp.
- Biểu thức tăng: là các biểu thức tăng giảm giá trị biến điều khiển vòng lặp, cách nhau dấu phẩy.

Ý nghĩa:

Khi lệnh lặp for bắt đầu, phần khởi tạo được thực hiện trước, và thực hiện duy nhất một lần, sau đó biểu thức điều kiện được kiểm tra, nếu biểu thức đúng thì lệnh sẽ được thực hiện, tiếp theo thực hiện phần tăng của vòng lặp và quá trình kiểm tra biểu thức điều kiện, thực hiện lệnh, và thực hiện phần tăng sẽ lặp đi lặp lại cho đến khi biểu thức điều kiện sai.

Sơ đồ khái:



Ví dụ: Nhập vào một số nguyên n từ bàn phím và tính tổng S từ 1 tới n

Cách giải:

```
using System;
class TinhTong
{
    static void Main(string[] args)
    {
        Console.WriteLine("Nhập vào một số nguyên n");
        int n = int.Parse(Console.ReadLine());
        int s = 0;
        for (int i = 1; i <= n; i++)
    }
```

```
{  
    s = s + i;  
}  
Console.WriteLine("Tong S la {0}", s);  
Console.ReadLine();  
}  
}
```

2.7.4.2. Câu trúc While

Cú pháp:

while (Biểu Thức Điều Kiện)

Lệnh;

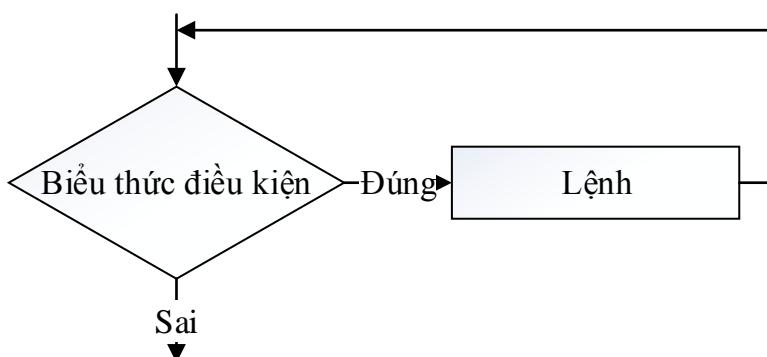
Trong đó:

- Lệnh (Statement) có thể là lệnh đơn, khối lệnh, hay câu trúc điều khiển.
- Biểu thức điều kiện: có kiểu logic

Ý nghĩa:

Kiểm tra chừng nào biểu thức điều kiện còn thỏa mãn thì còn thực hiện lệnh.

Sơ đồ khối:



Ví dụ: Tìm ước số chung lớn nhất của 2 số x, y

Phân tích: nếu $x = y$ thì USCLN của x và y là x ,
nếu $x > y$ thì USCLN của x , y cũng là USCLN của $x-y$ và y .

Do đó ta thay số lớn trong hai số x , y bằng hiệu số của chúng cho đến khi hai số bằng nhau, thì đó là ước số chung lớn nhất của hai số x , y .

Cách giải:

```
using System;
class UCLN
{
    static void Main(string[] args)
    {
        Console.WriteLine("Nhập vào số a");
        int a = int.Parse(Console.ReadLine());
        Console.WriteLine("Nhập vào số b");
        int b = int.Parse(Console.ReadLine());
        while (a != b)
        {
            if (a > b)
            {
                a = a - b;
            }
            else b = b - a;
        }
        Console.WriteLine("Ước chung lớn nhất là {0}", b);
        Console.ReadLine();
    }
}
```

2.7.4.3. Cấu trúc do ... while

Cú pháp:

do

Lệnh;

while (Biểu Thức Điều Kiện)

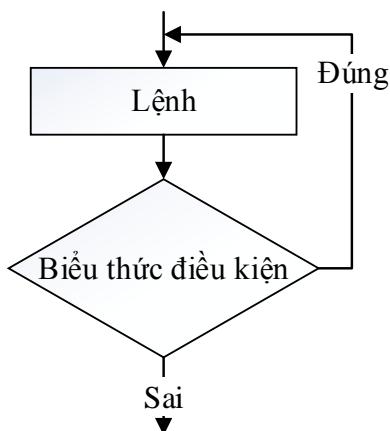
Trong đó:

- Lệnh (Statement) có thể là lệnh đơn, khối lệnh, hay cấu trúc điều khiển.
- Biểu thức điều kiện: có kiểu logic

Ý nghĩa:

Đầu tiên sẽ thực hiện lệnh, sau đó kiểm tra biểu thức điều kiện, nếu biểu thức đúng, thì vòng lặp tiếp tục, nếu biểu thức sai thì thoát vòng lặp

Sơ đồ khối:



Ví dụ: Nhập vào một số nguyên dương n từ bàn phím và tính tổng S từ 1 tới n.
Nếu n < 0 thì yêu cầu nhập lại n.

Cách giải:

```
using System;
class TinhTong
{
    static void Main(string[] args)
    {
        int n;
        do
        {
            Console.WriteLine("Nhập vào một số nguyên dương n");
            n = int.Parse(Console.ReadLine());
        } while (n < 0);

        int s = 0;
        for (int i = 1; i <= n; i++)
        {
            s = s + i;
        }
        Console.WriteLine("Tổng S là {0}", s);
        Console.ReadLine();
    }
}
```

2.7.4.4. Cấu trúc foreach

Cú pháp:

foreach (KiểuPhânTù TênPhânTù in TênDanhSách)

Lệnh;

Trong đó:

- Lệnh (Statement) có thể là lệnh đơn, khối lệnh, hay cấu trúc điều khiển.

- Kiểu phần tử (Item type): Là kiểu phần tử trong danh sách
- Tên phần tử (Item name): Là tên biến dùng để truy cập phần tử
- Tên danh sách (Collection name): Là tên biến danh sách

Ví dụ:

```
using System;
class Program
{
    static void Main(string[] args)
    {
        string[] nhom = { "Lan", "Thu", "Hoa", "Xuan" };
        foreach (string ten in nhom)
            Console.WriteLine("{0} ", ten);
        Console.ReadLine();
    }
}
```

2.7.5. Các kiểu lệnh khác

2.7.5.1. *Lệnh break*

Cú pháp:

break;
hay break Nhãn;

Ý nghĩa:

Lệnh break dùng để thoát tức thời ra khỏi vòng lặp while, do... while, for hay lệnh rẻ nhánh switch chứa nó và thực hiện lệnh tiếp theo.

Trong đó:

- Lệnh break Nhãn; dừng vòng lặp tức thời và quay về nhãn (Label). Tên nhãn đặt theo quy ước đặt tên của C#, ghi trong chương trình có dạng:

Nhãn:

Lệnh;

2.7.5.2. *Lệnh continue*

Cú pháp:

continue;

hay continue Nhãn;

Ý nghĩa:

Lệnh continue để bỏ qua các lệnh sau continue và quay trở về đầu vòng lặp chứa nó. Lệnh continue Nhãn bỏ qua các lệnh sau continue và quay về nhãn.

2.8. Ngoại lệ và xử lý ngoại lệ

Ngoại lệ trong C# là các đối tượng có kiểu lớp định nghĩa sẵn, biểu diễn trạng thái lỗi phát sinh trong trường hợp nào đó khi gọi phương thức hay trong trường hợp chia cho 0, cảnh báo bộ nhớ thấp... Một số phương thức định nghĩa sẵn trong trường hợp nào đó thì ngoại lệ sẽ phát sinh.

Khi một ngoại lệ xảy sinh, phải bắt ngoại lệ bởi lệnh try...catch, nếu không chương trình sẽ kết thúc thực hiện. Lệnh xử lý ngoại lệ thực hiện thông qua các từ khoá : try, catch, finally, throw.

2.8.1. *Lệnh try ...catch ...finally*

Cú pháp:

try {

lệnh;

}

catch (Kiểu ngoại lệ 1 [Đối tượng ngoại lệ 1])

{

Lệnh 1;

}

catch (Kiểu ngoại lệ 2 [Đối tượng ngoại lệ 2])

```
{  
    Lệnh 2;  
}  
...  
catch (Kiểu ngoại lệ n [Đối tượng ngoại lệ n])  
{  
    Lệnh n;  
}  
  
finally  
{  
    Lệnh n+1;  
}
```

Ý nghĩa:

- try : định nghĩa một khối lệnh mà ngoại lệ có thể xảy ra
- catch : đi kèm với try, để bắt ngoại lệ. Những câu lệnh trong chương trình mà bạn muốn bắt ngoại lệ được đưa vào giữa khối try, nếu một ngoại lệ xảy ra trong khối try, các lệnh còn lại trong khối try sẽ được bỏ qua và thân của mệnh đề catch có kiểu ngoại lệ tương ứng sẽ được thực hiện. Có thể có nhiều mệnh đề catch bắt các kiểu ngoại lệ khác nhau
- finally: thân của mệnh đề finally luôn luôn thực hiện ngay trước khi lệnh try kết thúc, cho dù có hay không có ngoại lệ, mệnh đề finally có thể có hay không.

Trong đó:

- Lệnh: Có thể là một hay nhiều lệnh đơn, hay cấu trúc điều khiển
- Đối tượng ngoại lệ (Exception instance): Là tên của đối tượng kiểu lớp ngoại lệ, có thể có hoặc không
- Kiểu ngoại lệ (Exception type) : Là kiểu lớp ngoại lệ

Ví dụ:

Nhập vào 2 số a, b nguyên từ bàn phím, tính thương bằng a chia b. Nếu a, b không phải nguyên dương thì thông báo nhập sai dữ liệu bắt nhập lại a,b. Nếu b bằng 0 thì thông báo lỗi chia cho 0 và nhập lại.

Cách giải:

```
using System;
class Program
{
    static void Main(string[] args)
    {
        bool kiemtra = true;
        do
        {
            try
            {
                Console.WriteLine("Nhập vào số a");
                int a = int.Parse(Console.ReadLine());
                Console.WriteLine("Nhập vào số b");
                int b = int.Parse(Console.ReadLine());
                int hieuso = a / b;
                Console.WriteLine("Thương a chia b là {0}",
                    hieuso);
                Console.WriteLine();
                kiemtra = true;
            }
            catch (FormatException)
            {
                Console.WriteLine("Bạn phải nhập vào a, b là các
so nguyên");
            }
        }
    }
}
```

```
        kiemtra = false;
    }
    catch (DivideByZeroException)
    {
        Console.WriteLine("Loi chia cho 0");
        kiemtra = false;
    }
} while (kiemtra == false);
}
```

2.8.2. Lệnh throw

Cú pháp:

throw Đôi tượng ngoại lệ;

Ý nghĩa:

Câu lệnh throw được đưa vào khối try, cho phép bạn điều khiển điều kiện phát sinh ngoại lệ. Khi gặp lệnh throw, các câu lệnh sau throw trong khối try sẽ được bỏ qua, và khối lệnh của mệnh đề catch có kiểu ngoại lệ tương ứng sẽ được thực hiện.

Trong đó:

- Đôi tượng ngoại lệ (Exception instance): Là một đôi tượng của kiểu ngoại lệ System.Exception, hay những đôi tượng được dẫn xuất từ kiểu dữ liệu này. Namespace System chứa một số các kiểu dữ liệu xử lý ngoại lệ mà chúng ta có thể sử dụng trong chương trình như:
 - ArgumentNullException
 - InvalidCastException
 - OverflowException...
- Tạo một đôi tượng kiểu ngoại lệ bằng toán tử new:

- throw new System.Exception();

2.8.3. Các lớp ngoại lệ

Exception là lớp cha của tất cả các ngoại lệ, cung cấp một số các phương thức và thuộc tính:

- Thuộc tính Message trả về thông tin phát sinh ngoại lệ.
- Thuộc tính HelpLink cung cấp một liên kết để trợ giúp cho các tập tin liên quan đến các ngoại lệ.
- Thuộc tính StackTrace cung cấp thông tin về câu lệnh lỗi.

Ta có bốn các kiểu ngoại lệ như sau

Kiểu ngoại lệ	Mô tả
MethodAccessException	<i>lỗi truy cập, do truy cập đến thành viên hay phương thức không được truy cập</i>
ArgumentException	<i>lỗi tham đổi</i>
ArgumentNullException	<i>tham đổi null, phương thức được truyền tham đổi null không được chấp nhận</i>
ArithmeticsException	<i>lỗi liên quan đến các phép toán</i>
ArrayTypeMismatchException	<i>kiểu thành phần mảng không đúng, khi lưu trữ kiểu không thích hợp vào mảng</i>
DivideByZeroException	<i>lỗi chia cho 0</i>
FormatException	<i>định dạng không chính xác</i>
IndexOutOfRangeException	<i>chỉ số truy cập thành phần mảng không hợp lệ</i>
InvalidCastException	<i>phép gán không hợp lệ</i>
NotFiniteNumberException	<i>không phải số hữu hạn, số không hợp lệ</i>
NotSupportedException	<i>phương thức không hỗ trợ, khi gọi một phương thức không tồn tại bên trong lớp</i>
NullReferenceException	<i>tham chiếu null không hợp lệ.</i>
OutOfMemoryException	<i>lỗi đầy bộ nhớ</i>

StackOverflowException	lỗi tràn stack
------------------------	----------------

2.9. Lớp System.Console

Lớp Console bao gồm các phương thức nhập, xuất và định dạng kết xuất, các luồng xử lý lỗi cho các ứng dụng dựa trên console. Lớp Console trong phiên bản .NET 2.0 đã được cải tiến thêm các đặc tính mới.

2.9.1. Định dạng kết xuất

Lớp Console cung cấp một số đặc tính định dạng kết xuất sau:

- BackgroundColor, ForegroundColor: hai thuộc tính này cho phép bạn thiết lập màu nền và màu chữ cho kết xuất ra màn hình.
- BufferHeight, BufferWidth: hai thuộc tính này điều chỉnh chiều cao và chiều rộng cho buffer của console
- Clear(): phương thức xóa buffer và vùng hiển thị của console.
- WindowHeight, WindowWidth, WindowTop, WindowLeft: các thuộc tính này điều khiển các chiều cao, rộng, lề trên, lề trái của console so với buffer

Ví dụ:

```
using System;
class Program
{
    static void Main(string[] args)
    {
        Console.Title = "Doi Mau";
        Console.ForegroundColor = ConsoleColor.Yellow;
        Console.BackgroundColor = ConsoleColor.Blue;
        Console.WriteLine("*****");
        Console.WriteLine("**** Dinh Dang Ket Xuat ****");
        Console.WriteLine("*****");
        Console.ReadLine();
```

```
    }  
}
```

2.9.2. Nhập và xuất với lớp Console

Lớp Console định nghĩa một tập các phương thức để thực hiện việc nhập dữ liệu từ bàn phím và xuất ra màn hình, tất cả đều được định nghĩa là static và do đó có thể được sử dụng ở cấp class.

- Phương thức WriteLine() xuất chuỗi ra màn hình, sau đó xuống dòng
- Phương thức Write() xuất chuỗi ra màn hình
- Phương thức ReadLine() cho phép bạn nhập dữ liệu từ bàn phím cho đến khi nhập phím xuống dòng
- Phương thức Read() sử dụng để nhập ký tự từ bàn phím

Ví dụ: Viết chương trình nhập vào họ tên, và năm sinh với điều kiện họ tên không quá 25 ký tự, và năm sinh từ 1980 đến 2000.

Cách giải:

```
using System;  
  
namespace Nhap  
{  
    class Program  
    {  
        static void Main(string[] args)  
        {  
            ConfigureCUI();  
            Console.ReadLine();  
        }  
        private static void ConfigureCUI()  
        {  
            string s = null;  
            // Nhập họ tên
```

```
do
{
    Console.WriteLine("Nhập Họ và tên không quá 25 ký tự:");
    s = Console.ReadLine();
} while (s.Length > 25 || s.Length == 0);
Console.WriteLine("Họ và tên là : {0}", s);
//Nhập năm sinh
while (true)
{
    try
    {
        Console.Write("Nhập năm sinh : ");
        s = Console.ReadLine();
        int ns = int.Parse(s);
        if (ns < 1980 || ns > 1985) throw new
FormatException();
        Console.WriteLine("Năm sinh là : " + ns);
        Console.ReadLine();
        break;
    }
    catch (FormatException e)
    {
        Console.WriteLine("Bạn nhập lại năm sinh từ
1980 đến 2000");
    }
}
}
```

2.9.3. Định dạng kết xuất cho phương thức Write

Trong các ví dụ, bạn thường xuyên thấy sự xuất hiện của các ký tự đánh dấu {0}, {1}..., được chèn trong chuỗi kết xuất. .Net giới thiệu một cách định dạng chuỗi mới, cũng khá giống so với hàm printf() của C, nhưng không còn các cờ %d, %s hay %c nữa.

Ví dụ:

```
using System;
namespace ViDu
{
    class Program
    {
        static void Main(string[] args)
        {
            int theInt = 90;
            double theDouble = 9.99;
            bool theBool = true;
            Console.WriteLine("Int is: {0}\nDouble is: {1}\nBool
is: {2}",
            theInt, theDouble, theBool);
        }
    }
}
```

Tham đối đầu tiên của phương thức WriteLine() là một chuỗi và chứa các ký hiệu đánh dấu {0}, {1}, {2}... (số trong ngoặc được bắt đầu từ số không). Và các tham đối còn lại của WriteLine() chỉ đơn giản là các giá trị được thêm vào lần lượt tương ứng với các ký hiệu đánh dấu (trong trường hợp này là một int, một double và một bool).

Phương thức WriteLine() cũng cho phép bạn truyền vào một mảng các đối tượng, giống như sau:

```
object[] stuff = { "Hello", 20.9, 1, "There", "83", 99.99933 };  
Console.WriteLine("The Stuff: {0} , {1} , {2} , {3} , {4} , {5}  
", stuff);
```

Nó cũng cho phép bạn đặt cùng một đánh dấu ở nhiều nơi:

```
Console.WriteLine("{0}, Number {0}, Number {0}", 9);
```

Nếu bạn truyền giá trị không khớp với các đánh dấu, sẽ phát sinh ngoại lệ FormatException

Nếu bạn muốn định dạng thêm nhiều kiểu khác nữa, có thể dùng các cờ sau (viết hoa hay thường đều được), được viết sau dấu hai chấm sau ký hiệu đánh dấu ({0:C}, {1:d}, {2:X},...):

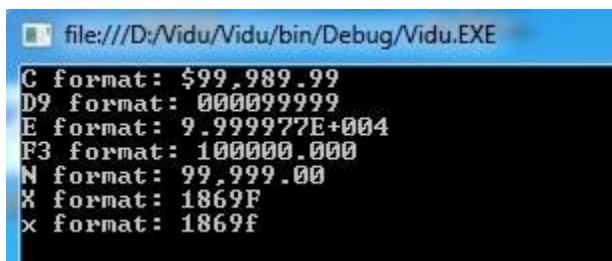
- **C** hay **c**: sử dụng để định dạng tiền tệ.
- **D** hay **d**: sử dụng để định dạng số thập phân.
- **E** hay **e**: sử dụng để thể hiện dạng số mũ.
- **F** hay **f**: sử dụng cho định dạng dấu chấm tĩnh.
- **G** hay **g**: viết tắt của general. Ký tự này dùng để định dạng kiểu chấm tĩnh hay số mũ.
- **N** hay **n**: sử dụng định dạng phần ngàn (với dấu phẩy)
- **X** hay **x**: sử dụng định dạng thập lục phân. Nếu bạn dùng X viết hoa thì ký tự thập lục cũng sẽ được viết hoa.

Ví dụ

```
using System;  
class Program  
{  
    static void Main(string[] args)  
    {  
        Console.WriteLine("C format: {0:C}", 99989.987);  
        Console.WriteLine("D9 format: {0:D9}", 99999);  
        Console.WriteLine("E format: {0:E}", 99999.76543);  
    }  
}
```

```
        Console.WriteLine("F3 format: {0:F3}", 99999.9999);
        Console.WriteLine("N format: {0:N}", 99999);
        Console.WriteLine("X format: {0:X}", 99999);
        Console.WriteLine("x format: {0:x}", 99999);
        Console.ReadLine();
    }
}
```

Kết quả:



Hoặc có thể sử dụng phương thức tĩnh string.Format() để định dạng chuỗi

Ví dụ:

```
using System;
class Program
{
    static void Main(string[] args)
    {
        string formatStr;
        formatStr = string.Format("Don't you wish you had {0:C} in your account?", 99989.987);
        Console.WriteLine(formatStr);
        Console.ReadLine();
    }
}
```

C. HÌNH THÚC VÀ PHƯƠNG PHÁP GIẢNG DẠY

- Trình chiếu powerpoint
- Đặt vấn đề, trao đổi
- Thực nghiệm kết hợp với máy tính

D. TÀI LIỆU THAM KHẢO

Tiếng Việt

[1] Nguyễn Ngọc Bình Phương, Thái Thanh Phong, *Các giải pháp lập trình C#*, Nhà sách Đất Việt

Tiếng Anh

[2] Erik brown, *Windows Forms Programming with C#*, Manning Publications Co

[3] James W. Cooper, *Introduction to Design Patterns in C#*

[4] Matthew MacDonald, *Pro .NET 2.0 Windows Forms and Custom Controls in C#*

Website

[5] <http://csharpcomputing.com/Tutorials/Lesson9.htm>

[6] <http://www.csharp-station.com/Tutorials/Lesson02.aspx>

[7] <http://msdn.microsoft.com/en-us/library/>

E. CÂU HỎI

1. Sự khác nhau giữa hằng định danh và hằng giá trị?
2. Sự khác nhau giữa kiểu dữ liệu giá trị và kiểu dữ liệu tham chiếu? Kiểu chuỗi, kiểu lớp trong C# là kiểu dữ liệu nào?
3. So sánh cấu trúc điều kiện if và switch
4. So sánh cấu trúc lặp while và do... while
5. Cho biết các lệnh rẽ nhánh có điều kiện và lệnh rẽ nhánh không điều kiện

F. BÀI TẬP

- 1) Viết chương trình đổi một ký tự thường thành ký tự in hoa (a-z thành A-Z)
- 2) Thời gian hoàn thành một công việc là x giây. Hãy đổi thời gian này ở dạng năm, tháng, ngày và giờ, phút giây
- 3) Tìm số lớn nhất trong 4 số a, b, c, d
- 4) Xếp loại sinh viên xuất sắc, giỏi, khá, trung bình khá, trung bình hay yếu dựa vào điểm trung bình

5) Giải phương trình bậc nhất

$$ax + b = 0$$

6) Giải phương trình bậc hai

$$ax^2 + bx + c = 0$$

7) Tính tổng sau trên n số nguyên dương đầu tiên

$$S = 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{n}$$

Lặp lại chương trình để có thể tính tổng với các giá trị n khác nhau

8) Tính tổng sau trên n số nguyên dương đầu tiên

$$S = \frac{1}{2^2} + \frac{3}{4^2} + \frac{5}{6^2} + \dots$$

9) Tính x^n

10) Tính n!

11) Tìm n số Fibonacii đầu tiên. Biết rằng dãy số Fibonacii như sau

$$1 \quad 1 \quad 2 \quad 3 \quad 5 \quad 8 \quad 13 \quad 21 \quad \dots$$

12) Viết chương trình in ra các số nguyên tố trong n số nguyên dương đầu tiên

13) Nhập epsilon, và tính tổng sau với độ chính xác epsilon (tính tổng chừng nào số hạng còn lớn hơn hay bằng sai số epsilon)

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots + \frac{x^n}{n!}$$

14) Nhập epsilon, và tính tổng sau với độ chính xác epsilon (tính tổng chừng nào số hạng còn lớn hơn hay bằng sai số epsilon)

$$\pi = 4 - \frac{4}{3} + \frac{4}{5} - \frac{4}{7} + \dots + (-1)^{n-1} \frac{4}{(2n-1)}$$

15) Nhập epsilon, và tính tổng sau với độ chính xác epsilon (tính tổng chừng nào số hạng còn lớn hơn hay bằng sai số epsilon)

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \dots + (-1)^n + \frac{x^{2n}}{2n!}$$

16) Nhập epsilon, và tính tổng sau với độ chính xác epsilon (tính tổng chừng nào số hạng còn lớn hơn hay bằng sai số epsilon)

$$\sin(x) = \frac{x}{1!} - \frac{x^3}{3!} + \frac{x^5}{5!} - \dots + (-1)^{n-1} \frac{x^{(2n-1)}}{(2n-1)!}$$

17) Nhập epsilon, và tính tổng sau với độ chính xác epsilon (tính tổng chừng nào số hạng còn lớn hơn hay bằng sai số epsilon)

$$1 - \frac{x}{1 \times 3} + \frac{x^2}{1 \times 3 \times 5} - \frac{x^3}{1 \times 3 \times 5 \times 7} + \dots + (-1)^n \frac{x^n}{1 \times 3 \times \dots \times (2n-1)(2n+1)}$$

Viết chương trình nhập vào một dãy số nguyên dương, kết thúc bằng -1.

- Tìm số lớn nhất và nhỏ nhất của dãy số này
- Đếm và tính tổng các số lớn hơn hoặc bằng 10 và nhỏ hơn hoặc bằng 20

18) Nhập chiều cao, xuất ra tam giác có dạng giống như hình:

*
* *
* * *
* * * *

19) Nhập chiều cao, xuất ra tam giác có dạng giống như hình:

* * * *
* * *
* *
*

20) Nhập chiều cao, xuất ra tam giác có dạng giống như hình:

*
* * *
* * * * *
* * * * * * *
* * * * * * * *

21) Viết chương trình in ra bảng cửu chương (BCC) (từ BCC 2 đến BCC 9)

22) Viết chương trình giải bài toán sau:

Trăm trâu trăm cỏ

Trâu đứng ăn năm

Trâu nằm ăn ba

Ba trâu già ăn một

Hỏi số trâu mỗi loại?

CHƯƠNG 3: LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG VỚI C#

A. MỤC TIÊU CHƯƠNG

1. VỀ KIẾN THỨC

Chương này cung cấp các kiến thức lập trình hướng đối tượng của C# và so sánh với ngôn ngữ Java và C++, giúp cho sinh viên nắm vững kiến thức lập trình hướng đối tượng với C# và vận dụng vào các ứng dụng Winform truy xuất cơ sở dữ liệu

- Định nghĩa lớp và tạo đối tượng
- Truy xuất thành phần của lớp
- Khai báo phương thức
- Phương thức khởi tạo
- Phương thức hủy
- Từ khóa this
- Nạp chồng phương thức
- Truyền tham đối cho phương thức
- Thuộc tính
- Ké thừa lớp
- Viết chồng hay che khuất phương thức
- Từ khóa base
- Kiểu delegate
- Kiểu cấu trúc
- Không gian tên
- Lớp, phương thức trừu tượng
- Lớp, phương thức hằng
- Định nghĩa và hiện thực giao tiếp
- Chuyển kiểu

2. VỀ KỸ NĂNG

- Biết cách tạo lớp và đối tượng

- Biết cách tạo và sử dụng phương thức
- Thuộc tính của đối tượng
- Tính kế thừa

B. NỘI DUNG

3.1. Xây dựng lớp, đối tượng

3.1.1. Định nghĩa lớp

Định danh lớp chính là tên của lớp do người xây dựng chương trình tạo ra. Lớp cơ sở là lớp mà đối tượng sẽ kế thừa để phát triển ta sẽ bàn sau. Tất cả các thành viên của lớp được định nghĩa bên trong thân của lớp, phần thân này sẽ được bao bọc bởi hai dấu ({ }).

Để định nghĩa một kiểu dữ liệu mới hay một lớp đầu tiên phải khai báo rồi sau đó mới định nghĩa các thuộc tính và phương thức của kiểu dữ liệu đó. Khai báo một lớp bằng cách sử dụng từ khoá class. Cú pháp đầy đủ của khai báo một lớp như sau:

[Thuộc tính] [Bổ sung truy cập] class <Định danh lớp> [: Lớp cơ sở]

{

<Phần thân của lớp: bao gồm định nghĩa các thuộc tính và phương thức hành động >

}

Thành phần thuộc tính của đối tượng sẽ được trình bày chi tiết trong chương sau, còn thành phần bổ sung truy cập cũng sẽ được trình bày tiếp ngay mục dưới.

*Ghi chú: Trong ngôn ngữ C# phần kết thúc của lớp không có dấu chấm phẩy giống như khai báo lớp trong ngôn ngữ C/C++. Tuy nhiên nếu người lập trình thêm vào thì trình biên dịch C# vẫn chấp nhận mà không đưa ra cảnh báo lỗi.

Trong C#, mọi chuyện đều xảy ra trong một lớp. Như các ví dụ mà chúng ta đã tìm hiểu trong chương 2, các hàm điều được đưa vào trong một lớp, kể cả hàm đầu vào của chương trình

(hàm Main()):

```
public class Tester  
{  
    static void Main  
    {  
    }  
}
```

Điều cần nói ở đây là chúng ta chưa tạo bất cứ thể hiện nào của lớp, tức là tạo đối tượng cho lớp Tester. Điều gì khác nhau giữa một lớp và thể hiện của lớp để trả lời cho câu hỏi này chúng ta bắt đầu xem xét sự khác nhau giữa kiểu dữ liệu int và một biến kiểu int. Ta có viết như sau:

```
int var1 = 10;  
tuy nhiên ta không thể viết được  
int = 10;
```

Ta không thể gán giá trị cho một kiểu dữ liệu, thay vào đó ta chỉ được gán dữ liệu cho một đối tượng của kiểu dữ liệu đó, trong trường hợp trên đối tượng là biến var1.

Khi chúng ta tạo một lớp mới, đó chính là việc định nghĩa các thuộc tính và hành vi của tất cả các đối tượng của lớp. Giả sử chúng ta đang lập trình để tạo các điều khiển trong các ứng dụng trên Windows, các điều khiển này giúp cho người dùng tương tác tốt với Windows, như là ListBox, TextBox, ComboBox,... Một trong những điều khiển thông dụng là ListBox, điều khiển này cung cấp một danh sách liệt kê các mục chọn và cho phép người dùng chọn các mục tin trong đó.

ListBox này cũng có các thuộc tính khác nhau như: chiều cao, bề dày, vị trí, và màu sắc thể hiện và các hành vi của chúng như: chúng có thể thêm bởi mục tin, sắp xếp...

Ngôn ngữ lập trình hướng đối tượng cho phép chúng ta tạo kiểu dữ liệu mới là lớp ListBox, lớp này bao bọc các thuộc tính cũng như khả năng như: các thuộc tính height, width, location, color, các phương thức hay hành vi như Add(), Remove(), Sort()...

Chúng ta không thể gán dữ liệu cho kiểu ListBox, thay vào đó đầu tiên ta phải tạo một đối tượng cho lớp đó:

ListBox myListBox;

Một khi chúng ta đã tạo một thẻ hiện của lớp ListBox thì ta có thể gán dữ liệu cho thẻ hiện đó. Tuy nhiên đoạn lệnh trên chưa thẻ tạo đối tượng trong bộ nhớ được, ta sẽ bàn tiếp. Bây giờ ta sẽ tìm hiểu cách tạo một lớp và tạo các thẻ hiện thông qua ví dụ minh họa 3.1. Ví dụ này tạo một lớp có chức năng hiển thị thời gian trong một ngày. Lớp này có hành vi thẻ hiện ngày, tháng, năm, giờ, phút, giây hiện hành. Để làm được điều trên thì lớp này có 6 thuộc tính hay còn gọi là biến thành viên, cùng với một phương thức như sau:

Ví dụ 3.1: Tạo một lớp Thoigian đơn giản như sau.

```
using System;
public class ThoiGian
{
    public void ThoiGianHienHanh()
    {
        Console.WriteLine("Hien thi thoi gian hien hanh");
    }
    // Các biến thành viên int Nam;
    int Thang; int Ngay; int Gio;
    int Phut;
    int Giay;
}
public class Tester
{
    static void Main()
    {
        ThoiGian t = new ThoiGian();
        t.ThoiGianHienHanh();
```

```
}
```

Kết quả:

Hien thi thoi gian hien hanh

Lớp ThoiGian chỉ có một phương thức chính là hàm ThoiGianHienHanh(), phần thân của phương thức này được định nghĩa bên trong của lớp ThoiGian. Điều này khác với ngôn ngữ C++, C# không đòi hỏi phải khai báo trước khi định nghĩa một phương thức, và cũng không hỗ trợ việc khai báo phương thức trong một tập tin và sau đó định nghĩa ở một tập tin khác. C# không có các tập tin tiêu đề, do vậy tất cả các phương thức được định nghĩa hoàn toàn bên trong của lớp. Phần cuối của định nghĩa lớp là phần khai báo các biến thành viên: Nam, Thang, Ngay, Gio, Phut, va Giay.

Sau khi định nghĩa xong lớp ThoiGian, thì tiếp theo là phần định nghĩa lớp Tester, lớp này có chứa một hàm khá thân thiện với chúng ta là hàm Main(). Bên trong hàm Main có một thê hiện của lớp ThoiGian được tạo ra và gán giá trị cho đối tượng t. Bởi vì t là thê hiện của đối tượng ThoiGian, nên hàm Main() có thể sử dụng phương thức của t:

t.ThoiGianHienHanh();

3.1.1.1. Thuộc tính truy cập

Thuộc tính truy cập quyết định khả năng các phương thức của lớp bao gồm việc các phương thức của lớp khác có thể nhìn thấy và sử dụng các biến thành viên hay những phương thức bên trong lớp. Bảng 3.1 tóm tắt các thuộc tính truy cập của một lớp trong C#.

Thuộc tính	Giới hạn truy cập
public	Không hạn chế. Những thành viên được đánh dấu public có thể được dùng bởi bất kì các phương thức của lớp bao gồm những lớp khác
private	Thành viên trong một lớp A được đánh dấu là private thì chỉ được truy cập bởi các phương thức của lớp A
protected	Thành viên trong lớp A được đánh dấu là protected thì

	chỉ được các phương thức bên trong lớp A và những phương thức dẫn xuất từ lớp A truy cập
internal	Thành viên trong lớp A được đánh dấu là internal thì được truy cập bởi những phương thức của bất cứ lớp nào trong cùng khối hợp ngữ với A
protected internal	Thành viên trong lớp A được đánh dấu là protected internal được truy cập bởi các phương thức của lớp A, các phương thức của lớp dẫn xuất của A, và bất cứ lớp nào trong cùng khối hợp ngữ của A

Mong muốn chung là thiết kế các biến thành viên của lớp ở thuộc tính private. Khi đó chỉ có phương thức thành viên của lớp truy cập được giá trị của biến. C# xem thuộc tính private là mặc định nên trong ví dụ 3.1 ta không khai báo thuộc tính truy cập cho 6 biến nên mặc định chúng là private:

```
// Các biến thành viên private int Nam;
int Thang; int Ngay; int Gio;
int Phut
int Giay;
```

Do lớp Tester và phương thức thành viên ThoiGianHienHanh của lớp ThoiGian được khai báo là public nên bất kỳ lớp nào cũng có thể truy cập được.

*Ghi chú: Thói quen lập trình tốt là khai báo tường minh các thuộc tính truy cập của biến thành viên hay các phương thức trong một lớp. Mặc dù chúng ta biết chắc chắn rằng các thành viên của lớp là được khai báo private mặc định. Việc khai báo tường minh này sẽ làm cho chương trình dễ hiểu, rõ ràng và tự nhiên hơn.

3.1.1.2. *Tham số của phương thức*

Trong các ngôn ngữ lập trình thì tham số và đối mục được xem là như nhau, cũng tương tự khi đang nói về ngôn ngữ hướng đối tượng thì ta gọi một hàm là một phương thức hay hành vi. Tất cả các tên này đều tương đồng với nhau.

Một phương thức có thể lấy bất kỳ số lượng tham số nào, Các tham số này theo sau bởi tên của phương thức và được bao bọc bên trong dấu ngoặc tròn (). Mỗi tham số

phải khai báo kèm với kiểu dữ liệu. ví dụ ta có một khai báo định nghĩa một phương thức có tên là Method, phương thức không trả về giá trị nào cả (khai báo giá trị trả về là void), và có hai tham số là một kiểu int và button:

```
void Method( int param1, button param2)  
{  
    //...  
}
```

Bên trong thân của phương thức, các tham số này được xem như những biến cục bộ, giống như là ta khai báo biến bên trong phương thức và khởi tạo giá trị bằng giá trị của tham số truyền vào. Ví dụ 3.2 minh họa việc truyền tham số vào một phương thức, trong trường hợp này thì hai tham số của kiểu là int và float.

Ví dụ 3.2: Truyền tham số cho phương thức.

```
using System;  
  
public class Class1  
{  
    public void SomeMethod(int p1, float p2)  
    {  
        Console.WriteLine("Ham nhan duoc hai tham so: {0} va  
        {1}", p1, p2);  
    }  
}  
  
public class Tester  
{  
    static void Main()  
    {  
        int var1 = 5;  
        float var2 = 10.5f; Class1 c = new Class1();  
        c.SomeMethod(var1, var2);  
    }  
}
```

```
    }  
}
```

□ Kết quả:

Ham nhan duoc hai tham so: 5 va 10.5

Phương thức SomeMethod sẽ lấy hai tham số int và float rồi hiển thị chúng ta màn hình bằng việc dùng hàm Console.WriteLine(). Những tham số này có tên là p1 và p2 được xem như là biến cục bộ bên trong của phương thức.

Trong phương thức gọi Main, có hai biến cục bộ được tạo ra là var1 và var2. Khi hai biến này được truyền cho phương thức SomeMethod thì chúng được ánh xạ thành hai tham số p1 và p2 theo thứ tự danh sách biến đưa vào.

3.1.2. Tạo đối tượng

Trong Chương 2 có đề cập đến sự khác nhau giữa kiểu dữ liệu giá trị và kiểu dữ liệu tham chiếu. Những kiểu dữ liệu chuẩn của C# như int, char, float, ... là những kiểu dữ liệu giá trị, và các biến được tạo ra từ các kiểu dữ liệu này được lưu trên stack. Tuy nhiên, với các đối tượng kiểu dữ liệu tham chiếu thì được tạo ra trên heap, sử dụng từ khóa new để tạo một đối tượng:

```
ThoiGian t = new ThoiGian();
```

t thật sự không chứa giá trị của đối tượng ThoiGian, nó chỉ chứa địa chỉ của đối tượng được tạo ra trên heap, do vậy t chỉ chứa tham chiếu đến một đối tượng mà thôi.

3.1.2.1. Bộ khởi dựng

Thử xem lại ví dụ minh họa 3.1, câu lệnh tạo một đối tượng cho lớp ThoiGian tương tự như việc gọi thực hiện một phương thức:

```
ThoiGian t = new ThoiGian();
```

Đúng như vậy, một phương thức sẽ được gọi thực hiện khi chúng ta tạo một đối tượng. Phương thức này được gọi là bộ khởi dựng (constructor). Các phương thức này được định nghĩa khi xây dựng lớp, nếu ta không tạo ra thì CLR sẽ thay mặt

chúng ta mà tạo phương thức khởi dụng một cách mặc định. Chức năng của bộ khởi dụng là tạo ra đối tượng được xác định bởi một lớp và đặt trạng thái này hợp lệ. Trước khi bộ khởi dụng được thực hiện thì đối tượng chưa được cấp phát trong bộ nhớ. Sau khi bộ khởi dụng thực hiện hoàn thành thì bộ nhớ sẽ lưu giữ một thể hiện hợp lệ của lớp vừa khai báo.

Lớp ThoiGian trong ví dụ 3.1 không định nghĩa bộ khởi dụng. Do không định nghĩa nên trình biên dịch sẽ cung cấp một bộ khởi dụng cho chúng ta. Phương thức khởi dụng mặc định được tạo ra cho một đối tượng sẽ không thực hiện bất cứ hành động nào, tức là bên trong thân của phương thức rỗng. Các biến thành viên được khởi tạo các giá trị tầm thường như thuộc tính nguyên có giá trị là 0 và chuỗi thì khởi tạo rỗng...Bảng 3.2 sau tóm tắt các giá trị mặc định được gán cho các kiểu dữ liệu cơ bản.

Kiểu dữ liệu	Giá trị mặc định
int, long, byte,...	0
Bool	false
Char	'\0' (null)
Enum	0
reference	null

Thường thường, khi muốn định nghĩa một phương thức khởi dụng riêng ta phải cung cấp các tham số để hàm khởi dụng có thể khởi tạo các giá trị khác ngoài giá trị mặc định cho các đối tượng. Quay lại ví dụ 3.1 giả sử ta muốn truyền thời gian hiện hành: năm, tháng, ngày,... để đối tượng có ý nghĩa hơn.

Để định nghĩa một bộ khởi dụng riêng ta phải khai báo một phương thức có tên giống như tên lớp đã khai báo. Phương thức khởi dụng không có giá trị trả về và được khai báo là public. Nếu phương thức khởi dụng này được truyền tham số thì phải khai báo danh sách tham số giống như khai báo với bất kỳ phương thức nào trong một lớp. Ví dụ 3.3 được viết lại từ ví dụ 3.1 và thêm một bộ khởi dụng riêng, phương thức khởi dụng này sẽ nhận một tham số là một đối tượng kiểu DateTime do C# cung cấp.

Ví dụ 3.3: Định nghĩa một bộ khởi dụng.

```
using System;

public class ThoiGian
{
    public void ThoiGianHienHanh()
    {
        Console.WriteLine("Thoi gian hien hanh la : {0}/{1}/{2}
{3}:{4}:{5}", Ngay,
        Thang, Nam, Gio, Phut, Giay);
    }

    // Hàm khởi động
    public ThoiGian(System.DateTime dt)
    {
        Nam = dt.Year; Thang = dt.Month; Ngay = dt.Day;
        Gio = dt.Hour; Phut = dt.Minute; Giay = dt.Second;
    }

    // Biến thành viên private int Nam;
    int Nam; int Thang; int Ngay; int Gio;
    int Phut;
    int Giay;
}

public class Tester
{
    static void Main()
    {
        System.DateTime currentTime = System.DateTime.Now;
        ThoiGian t = new
        ThoiGian(currentTime); t.ThoiGianHienHanh();
    }
}
```

Kết quả

Thoi gian hien hanh la: 7/12/2016 8:15:20

Trong ví dụ trên phương thức khởi dụng lấy một đối tượng DateTime và khởi tạo tất cả các biến thành viên dựa trên giá trị của đối tượng này. Khi phương thức này thực hiện xong, một đối tượng ThoiGian được tạo ra và các biến của đối tượng cũng đã được khởi tạo. Hàm ThoiGianHienHanh được gọi trong hàm Main() sẽ hiển thị giá trị thời gian lúc đối tượng được tạo ra.

Chúng ta thử bỏ một số lệnh khởi tạo trong phương thức khởi dụng và cho thực hiện chương trình lại thì các biến không được khởi tạo sẽ có giá trị mặc định là 0, do là biến nguyên. Một biến thành viên kiểu nguyên sẽ được thiết lập giá trị là 0 nếu chúng ta không gán nó trong phương thức khởi dụng. Chú ý rằng kiểu dữ liệu giá trị không thể không được khởi tạo, nếu ta không khởi tạo thì trình biên dịch sẽ cung cấp các giá trị mặc định theo bảng 3.2.

Ngoài ra trong chương trình 4.3 trên có sử dụng đối tượng của lớp DateTime, lớp DateTime này được cung cấp bởi thư viện System, lớp này cũng cung cấp các biến thành viên public như: Year, Month, Day, Hour, Minute, và Second tương tự như lớp ThoiGian của chúng ta.Thêm vào đó là lớp này có đưa ra một phương thức thành viên tĩnh tên là Now, phương thức Now sẽ trả về một tham chiếu đến một thể hiện của một đối tượng DateTime được khởi tạo với thời gian hiện hành.

Theo như trên khi lệnh :

```
System.DateTime currentTime = System.DateTime.Now();
```

Được thực hiện thì phương thức tĩnh Now() sẽ tạo ra một đối tượng DateTime trên bộ nhớ heap và trả về một tham chiếu và tham chiếu này được gán cho biến đối tượng currentTime. Sau khi đối tượng currentTime được tạo thì câu lệnh tiếp theo sẽ thực hiện việc truyền đối tượng currentTime cho phương thức khởi dụng để tạo một đối tượng ThoiGian:

```
ThoiGian t = new ThoiGian( currentTime );
```

Bên trong phương thức khởi dụng này tham số dt sẽ tham chiếu đến đối tượng DateTime là đối tượng vừa tạo mà currentTime cũng tham chiếu. Nói cách khác lúc này tham số dt và currentTime cùng tham chiếu đến một đối tượng DateTime trong bộ nhớ. Nhờ vậy phương thức khởi dụng ThoiGian có thể truy cập được các biến thành viên public của đối tượng DateTime được tạo trong hàm Main().

Có một sự nhấn mạnh ở đây là đối tượng DateTime được truyền cho bộ dựng ThoiGian chính là đối tượng đã được tạo trong hàm Main và là kiểu dữ liệu tham chiếu. Do vậy khi thực hiện truyền tham số là một kiểu dữ liệu tham chiếu thì con trỏ được ánh xạ qua chứ hoàn toàn không có một đối tượng nào được sao chép lại.

3.1.2.2. *Khởi tạo biến thành viên*

Các biến thành viên có thể được khởi tạo trực tiếp khi khai báo trong quá trình khởi tạo, thay vì phải thực hiện việc khởi tạo các biến trong bộ khởi dựng. Để thực hiện việc khởi tạo này rất đơn giản là việc sử dụng phép gán giá trị cho một biến:

```
private int Giay = 30; // Khởi tạo
```

Việc khởi tạo biến thành viên sẽ rất có ý nghĩa, vì khi xác định giá trị khởi tạo như vậy thì biến sẽ nhận giá trị mặc định mà trình biên dịch cung cấp. Khi đó nếu các biến này không được gán lại trong các phương thức khởi dụng thì nó sẽ có giá trị mà ta đã khởi tạo. Ví dụ 3.4 minh họa việc khởi tạo biến thành viên khi khai báo. Trong ví dụ này sẽ có hai bộ dựng ngoài bộ dựng mặc định mà trình biên dịch cung cấp, một bộ dựng thực hiện việc gán giá trị cho tất cả các biến thành viên, còn bộ dựng thứ hai thì cũng tương tự nhưng sẽ không gán giá trị cho biến Giay.

- Ví dụ 3.4: Minh họa sử dụng khởi tạo biến thành viên.

```
using System;  
  
public class ThoiGian  
{  
    public void ThoiGianHienHanh()  
    {
```

```
System.DateTime now = System.DateTime.Now;

System.Console.WriteLine("\n Hien tai: \t {0}/{1}/{2}
{3}:{4}:{5}", now.Day, now.Month, now.Year, now.Hour, now.Minute,
now.Second);

System.Console.WriteLine("Thoi Gian:\t {0}/{1}/{2}
{3}:{4}:{5}", Ngay, Thang, Nam, Gio, Phut, Giay);

}

public ThoiGian(System.DateTime dt)

{

    Nam = dt.Year; Thang = dt.Month; Ngay = dt.Day;

    Gio = dt.Hour; Phut = dt.Minute;

    Giay = dt.Second;      // có gán cho biến thành viên Giay

}

public ThoiGian(int Year, int Month, int Date, int Hour, int
Minute)

{

    Nam = Year; Thang = Month; Ngay = Date;

    Gio = Hour; Phut = Minute;

}

private int Nam;

private int Thang;

private int Ngay;

private int Gio;

private int phut;

private int Giay = 30; // biến được khởi tạo.

}

public class Tester
```

```
{  
    static void Main()  
  
    {  
        System.DateTime currentTime = System.DateTime.Now;  
  
        ThoiGian t1 = new ThoiGian(currentTime);  
  
        t1.ThoiGianHienHanh();  
  
        ThoiGian t2 = new ThoiGian(2001, 7, 3, 10, 5);  
  
        t2.ThoiGianHienHanh();  
  
        Console.ReadLine();  
    }  
}
```

□ Kết quả:

Hien tai:	5/6/2002 10:15:5
Thoi Gian:	5/6/2002 10:15:5
Hien tai:	5/6/2002 10:15:5
Thoi Gian:	3/7/2001 10:5:30

Nếu không khởi tạo giá trị của biến thành viên thì bộ khởi dụng mặc định sẽ khởi tạo giá trị là 0 mặc định cho biến thành viên có kiểu nguyên. Tuy nhiên, trong trường hợp này biến thành viên Giay được khởi tạo giá trị 30:

Giay = 30; // Khởi tạo

Trong trường hợp bộ khởi tạo thứ hai không truyền giá trị cho biến Giay nên biến này vẫn lấy giá trị mà ta đã khởi tạo ban đầu là 30:

ThoiGian t2 = new ThoiGian(2001, 7, 3, 10, 5);

t2.ThoiGianHienHanh();

Ngược lại, nếu một giá trị được gán cho biến Giay như trong bộ khởi tạo thứ nhất thì giá trị mới này sẽ được chồng lên giá trị khởi tạo.

Trong ví dụ trên lần đầu tiên tạo đối tượng ThoiGian do ta truyền vào đối tượng DateTime nên hàm khởi dựng thứ nhất được thực hiện, hàm này sẽ gán giá trị 5 cho biến Giay. Còn khi tạo đối tượng ThoiGian thứ hai, hàm khởi dựng thứ hai được thực hiện, hàm này không gán giá trị cho biến Giay nên biến này vẫn còn lưu giữ lại giá trị 30 khi khởi tạo ban đầu.

3.1.2.3. Bộ khởi dựng sao chép

Bộ khởi dựng sao chép thực hiện việc tạo một đối tượng mới bằng cách sao chép tất cả các biến từ một đối tượng đã có và cùng một kiểu dữ liệu. Ví dụ chúng ta muốn đưa một đối tượng ThoiGian vào bộ khởi dựng lớp ThoiGian để tạo một đối tượng ThoiGian mới có cùng giá trị với đối tượng ThoiGian cũ. Hai đối tượng này hoàn toàn khác nhau và chỉ giống nhau ở giá trị biến thành viên sao khi khởi dựng.

Ngôn ngữ C# không cung cấp bộ khởi dựng sao chép, do đó chúng ta phải tự tạo ra. Việc sao chép các thành phần từ một đối tượng ban đầu cho một đối tượng mới như sau:

```
public ThoiGian( ThoiGian tg)
{
    Nam = tg.Nam; Thang = tg.Thang; Ngay = tg.Ngay; Gio = tg.Gio;
    Phut = tg.Phut; Giay = tg.Giay;
}
```

Khi đó ta có thể sao chép từ một đối tượng ThoiGian đã hiện hữu như sau:

```
ThoiGian t2 = new ThoiGian( t1 );
```

Trong đó t1 là đối tượng ThoiGian đã tồn tại, sau khi lệnh trên thực hiện xong thì đối tượng t2 được tạo ra như bản sao của đối tượng t1.

3.1.2.4. Từ khóa this

Từ khóa this được dùng để tham chiếu đến thể hiện hiện hành của một đối tượng. Tham chiếu this này được xem là con trỏ ẩn đến tất cả các phương thức không có thuộc tính tĩnh trong một lớp. Mỗi phương thức có thể tham chiếu đến những phương thức khác và các biến thành viên thông qua tham chiếu this này.

Tham chiếu this này được sử dụng thường xuyên theo ba cách:

*Sử dụng khi các biến thành viên bị che lấp bởi tham số đưa vào, như trường hợp sau:

```
public void SetYear( int Nam )
{
    this.Nam = Nam;
}
```

Như trong đoạn mã trên phương thức SetYear sẽ thiết lập giá trị của biến thành viên Nam, tuy nhiên do tham số đưa vào có tên là Nam, trùng với biến thành viên, nên ta phải dùng tham chiếu this để xác định rõ các biến thành viên và tham số được truyền vào. Khi đó this.Nam chỉ đến biến thành viên của đối tượng, trong khi Nam chỉ đến tham số.

*Sử dụng tham chiếu this để truyền đối tượng hiện hành vào một tham số của một phương thức của đối tượng khác:

```
public void Method1( OtherClass otherObject )
{
    // Sử dụng tham chiếu this để truyền tham số là bản
    // thân đối tượng đang thực hiện. otherObject.SetObject( this );
}
```

Như trên cho thấy khi cần truyền một tham số là chính bản thân của đối tượng đang thực hiện thì ta bắt buộc phải dùng tham chiếu this để truyền.

*Cách thứ ba sử dụng tham chiếu this là mảng chỉ mục (indexer), phần này được trình bày chi tiết trong chương 2.

3.1.3. Sử dụng các thành viên tĩnh

Những thuộc tính và phương thức trong một lớp có thể là những thành viên thể hiện (instance members) hay những thành viên tĩnh (static members). Những thành viên thể hiện hay thành viên của đối tượng liên quan đến thể hiện của một kiểu dữ liệu. Trong khi thành viên tĩnh được xem như một phần của lớp. Chúng ta có thể truy cập đến thành viên tĩnh của một lớp thông qua tên lớp đã được khai báo. Ví dụ chúng ta có

một lớp tên là Button và có hai thẻ hiện của lớp tên là btnUpdate và btnDelete. Và giả sử lớp Button này có một phương thức tĩnh là Show(). Để truy cập phương thức tĩnh này ta viết :

```
Button.Show();
```

Đúng hơn là viết:

```
btnUpdate.Show();
```

*Ghi chú: Trong ngôn ngữ C# không cho phép truy cập đến các phương thức tĩnh và các biến thành viên tĩnh thông qua một thẻ hiện, nếu chúng ta cố làm điều đó thì trình biên dịch C# sẽ báo lỗi, điều này khác với ngôn ngữ C++.

Trong một số ngôn ngữ thì có sự phân chia giữa phương thức của lớp và các phương thức khác (tổng cục) tồn tại bên ngoài không phụ thuộc bất cứ một lớp nào. Tuy nhiên, điều này không cho phép trong C#, ngôn ngữ C# không cho phép tạo các phương thức bên ngoài của lớp, nhưng ta có thể tạo được các phương thức giống như vậy bằng cách tạo các phương thức tĩnh bên trong một lớp.

Phương thức tĩnh hoạt động ít nhiều giống như phương thức toàn cục, ta truy cập phương thức này mà không cần phải tạo bất cứ thẻ hiện hay đối tượng của lớp chứa phương thức toàn cục. Tuy nhiên, lợi ích của phương thức tĩnh vượt xa phương thức toàn cục vì phương thức tĩnh được bao bọc trong phạm vi của một lớp nơi nó được định nghĩa, do vậy ta sẽ không gặp tình trạng lạm xôn giữa các phương thức trùng tên do chúng được đặt trong namespace.

*Ghi chú: Chúng ta không nên bị cám dỗ bởi việc tạo ra một lớp chứa toàn bộ các phương thức linh tinh. Điều này có thể tiện cho công việc lập trình nhưng sẽ điều không mong muốn và giảm tính ý nghĩa của việc thiết kế hướng đối tượng. Vì đặc tính của việc tạo các đối tượng là xây dựng các phương thức và hành vi xung quanh các thuộc tính hay dữ liệu của đối tượng.

3.1.3.1. Gọi một phương thức tĩnh

Như chúng ta đã biết phương thức Main() là một phương thức tĩnh. Phương tĩnh được xem như là phần hoạt động của lớp hơn là của thể hiện một lớp. Chúng cũng không cần có một tham chiếu this hay bất cứ thể hiện nào tham chiếu tới.

Phương thức tĩnh không thể truy cập trực tiếp đến các thành viên không có tính chất tĩnh (nonstatic). Như vậy Main() không thể gọi một phương thức không tĩnh bên trong lớp.

```
using System;
public class Class1
{
    public void SomeMethod(int p1, float p2)
    {
        Console.WriteLine("Ham nhan duoc hai tham so: {0} va
{1}", p1, p2);
    }
}
public class Tester
{
    static void Main()
    {
        int var1 = 5;
        float var2 = 10.5f; Class1 c = new Class1();
        c.SomeMethod(var1, var2);
    }
}
```

Phương thức SomeMethod() là phương thức không tĩnh của lớp Class1, do đó để truy cập được phương thức của lớp này ta cần phải tạo một thể hiện là một đối tượng cho lớp Class1

Sau khi tạo thì có thể thông qua đối tượng c ta có thể gọi được được phương thức Some- Method().

3.1.3.2. Sử dụng bộ khởi tạo tĩnh

Nếu một lớp khai báo một bộ khởi tạo tĩnh (static constructor), thì được đảm bảo rằng phương thức khởi tạo tĩnh này sẽ được thực hiện trước bất cứ thê hiện nào của lớp được tạo ra.

*Ghi chú: Chúng ta không thể điều khiển chính xác khi nào thì phương thức khởi tạo tĩnh này được thực hiện. Tuy nhiên ta biết chắc rằng nó sẽ được thực hiện sau khi chương trình chạy và trước bất kỳ biến đối tượng nào được tạo ra.

Theo ví dụ 3.4 ta có thể thêm một bộ khởi tạo tĩnh cho lớp ThoiGian như sau:

```
static ThoiGian()
{
    Ten = "Thoi gian";
}
```

Lưu ý rằng ở đây không có bất cứ thuộc tính truy cập nào như public trước bộ khởi tạo tĩnh. Thuộc tính truy cập không cho phép theo sau một phương thức khởi tạo tĩnh. Do phương thức tĩnh nên không thể truy cập bất cứ biến thành viên không thuộc loại tĩnh, vì vậy biến thành viên Name bên trên cũng phải được khai báo là tĩnh:

```
private static string Ten;
```

Cuối cùng ta thêm một dòng vào phương thức ThoiGianHienHanh() của lớp ThoiGian:

```
public void ThoiGianHienHanh()
{
    System.Console.WriteLine("Ten: {0}", Ten);
    System.Console.WriteLine("Thoi Gian:\t {0}/{1}/{2}\n{3}:{4}:{5}", Ngay, Thang, Nam, Gio, Phut, Giay);
}
```

Sau khi thay đổi ta biên dịch và chạy chương trình được kết quả sau:

Ten: Thoi Gian

Thoi Gian: 5/6/2002 18:35:20

Mặc dù chương trình thực hiện tốt, nhưng không cần thiết phải tạo ra bộ khởi dụng tĩnh để phục vụ cho mục đích này. Thay vào đó ta có thể dùng chức năng khởi tạo biến thành viên như sau:

```
private static string Ten = "Thoi Gian";
```

Tuy nhiên, bộ khởi tạo tĩnh có hữu dụng khi chúng ta cần cài đặt một số công việc mà không thể thực hiện được thông qua chức năng khởi dụng và công việc cài đặt này chỉ được thực hiện duy nhất một lần.

3.1.3.3. Sử dụng bộ khởi dụng private

Như đã nói ngôn ngữ C# không có phương thức toàn cục và hằng số toàn cục. Do vậy chúng ta có thể tạo ra những lớp tiện ích nhỏ chỉ để chứa các phương thức tĩnh. Cách thực hiện này luôn có hai mặt tốt và không tốt. Nếu chúng ta tạo một lớp tiện ích như vậy và không muốn bắt cứ một thể hiện nào được tạo ra. Để ngăn ngừa việc tạo bất cứ thể hiện của lớp ta tạo ra bộ khởi dụng không có tham số và không làm gì cả, tức là bên trong thân của phương thức rỗng, và thêm vào đó phương thức này được đánh dấu là private. Do không có bộ khởi dụng public, nên không thể tạo ra bất cứ thể hiện nào của lớp.

3.1.3.4. Sử dụng các thuộc tính tĩnh

Một vấn đề đặt ra là làm sao kiểm soát được số thể hiện của một lớp được tạo ra khi thực hiện chương trình. Vì hoàn toàn ta không thể tạo được biến toàn cục để làm công việc đếm số thể hiện của một lớp.

Thông thường các biến thành viên tĩnh được dùng để đếm số thể hiện đã được tạo ra của một lớp. Cách sử dụng này được áp dụng trong minh họa sau:

*Ví dụ 4.5: Sử dụng thuộc tính tĩnh để đếm số thể hiện.

```
using System;  
public class Cat  
{  
    public Cat()
```

```
{  
    instance++;  
}  
public static void HowManyCats()  
{  
    Console.WriteLine("{0} cats", instance);  
}  
private static int instance = 0;  
}  
public class Tester  
{  
    static void Main()  
    {  
        Cat.HowManyCats(); Cat mun = new Cat();  
Cat.HowManyCats(); Cat muop =  
        new Cat();  
        Cat miu = new Cat(); Cat.HowManyCats();  
    }  
}
```

Kết quả:

0 cats

1 cats

3 cats

Bên trong lớp Cat ta khai báo một biến thành viên tĩnh tên là instance biến này dùng để đếm số thể hiện của lớp Cat, biến này được khởi tạo giá trị 0. Lưu ý rằng biến thành viên tĩnh được xem là thành phần của lớp, không phải là thành viên của thể hiện, do vậy nó sẽ không được khởi tạo bởi trình biên dịch khi tạo các thể hiện. Khởi tạo tường minh là yêu cầu bắt buộc với các biến thành viên tĩnh. Khi một thể hiện được tạo ra thì bộ dụng của lớp Cat sẽ thực hiện tăng biến instance lên một đơn vị.

3.1.4. Hủy đối tượng

Ngôn ngữ C# cung cấp cơ chế thu dọn (garbage collection) và do vậy không cần phải khai báo tường minh các phương thức hủy. Tuy nhiên, khi làm việc với các đoạn mã không được quản lý thì cần phải khai báo tường minh các phương thức hủy để giải phóng các tài nguyên. C# cung cấp ngần định một phương thức để thực hiện điều khiển công việc này, phương thức đó là `Finalize()` hay còn gọi là bộ kết thúc. Phương thức `Finalize` này sẽ được gọi bởi cơ chế thu dọn khi đối tượng bị hủy.

Phương thức kết thúc chỉ giải phóng các tài nguyên mà đối tượng nắm giữ, và không tham chiếu đến các đối tượng khác. Nếu với những đoạn mã bình thường tức là chứa các tham chiếu kiểm soát được thì không cần thiết phải tạo và thực thi phương thức `Finalize()`. Chúng ta chỉ làm điều này khi xử lý các tài nguyên không kiểm soát được.

Chúng ta không bao giờ gọi một phương thức `Finalize()` của một đối tượng một cách trực tiếp, ngoại trừ gọi phương thức này của lớp cơ sở khi ở bên trong phương thức `Finalize()` của chúng ta. Trình thu dọn sẽ thực hiện việc gọi `Finalize()` cho chúng ta.

3.1.4.1. Cách `Finalize` thực hiện

Bộ thu dọn duy trì một danh sách những đối tượng có phương thức `Finalize`. Danh sách này được cập nhật mỗi lần khi đối tượng cuối cùng được tạo ra hay bị hủy. Khi một đối tượng trong danh sách kết thúc của bộ thu dọn được chọn đầu tiên. Nó sẽ được đặt vào hàng đợi (queue) cùng với những đối tượng khác đang chờ kết thúc. Sau khi phương thức `Finalize` của đối tượng thực thi bộ thu dọn sẽ gom lại đối tượng và cập nhật lại danh sách hàng đợi, cũng như là danh sách kết thúc đối tượng.

3.1.4.2. Bộ hủy của C#

Cú pháp phương thức hủy trong ngôn ngữ C# cũng giống như trong ngôn ngữ C++. Nhưng về hành động cụ thể chúng có nhiều điểm khác nhau. Ta khao báo một phương thức hủy trong C# như sau:

```
~Class1() {}
```

Tuy nhiên, trong ngôn ngữ C# thì cú pháp khai báo trên là một shortcut liên kết đến một phương thức kết thúc Finalize được kết với lớp cơ sở, do vậy khi viết

```
~Class1()  
{  
    // Thực hiện một số công việc  
}
```

Cũng tương tự như viết :

```
Class1.Finalize()  
{  
    // Thực hiện một số công việc base.Finalize();  
}
```

Do sự tương tự như trên nên khả năng dẫn đến sự lộn xộn nhầm lẫn là không tránh khỏi, nên chúng ta phải tránh viết các phương thức hủy và viết các phương thức Finalize tường minh nếu có thể được.

3.1.4.3. Phương thức Dispose

Như chúng ta đã biết thì việc gọi một phương thức kết thúc Finalize trong C# là không hợp lệ, vì phương thức này dành cho bộ thu dọn thực hiện. Nếu chúng ta xử lý các tài nguyên không kiểm soát như xử lý các handle của tập tin và ta muốn được đóng hay giải phóng nhanh chóng bất cứ lúc nào, ta có thực thi giao diện IDisposable, phần chi tiết IDisposable sẽ được trình bày chi tiết trong Chương 8. Giao diện IDisposable yêu cầu những thành phần thực thi của nó định nghĩa một phương thức tên là Dispose() để thực hiện công việc dọn dẹp mà ta yêu cầu. Ý nghĩa của phương thức Dispose là cho phép chương trình thực hiện các công việc dọn dẹp hay giải phóng tài nguyên mong muốn mà không phải chờ cho đến khi phương thức Finalize() được gọi.

Khi chúng ta cung cấp một phương thức Dispose thì phải ngưng bộ thu dọn gọi phương thức Finalize() trong đối tượng của chúng ta. Để ngưng bộ thu dọn, chúng ta gọi một phương thức tĩnh của lớp GC (garbage collector) là GC.SuppressFinalize() và truyền tham số là tham chiếu this của đối tượng. Và sau đó phương thức Finalize() sử dụng để gọi phương thức Dispose() như đoạn mã sau:

```
public void Dispose()
{
    // Thực hiện công việc dọn dẹp
    // Yêu cầu bộ thu dọn GC trong thực hiện kết thúc
    GC.SuppressFinalize( this );
}

public override void Finalize()
{
    Dispose();
    base.Finalize();
}
```

3.1.4.4. Phương thức Close

Khi xây dựng các đối tượng, chúng ta có muốn cung cấp cho người sử dụng phương thức Close(), vì phương thức Close có vẻ tự nhiên hơn phương thức Dispose trong các đối tượng có liên quan đến xử lý tập tin. Ta có thể xây dựng phương thức Dispose() với thuộc tính là private và phương thức Close() với thuộc tính public. Trong phương thức Close() đơn giản là gọi thực hiện phương thức Dispose().

3.1.4.5. Câu lệnh using

Khi xây dựng các đối tượng chúng ta không thể chắc chắn được rằng người sử dụng có thể gọi hàm Dispose(). Và cũng không kiểm soát được lúc nào thì bộ thu dọn GC thực hiện việc dọn dẹp. Do đó để cung cấp khả năng mạnh hơn để kiểm soát việc giải phóng tài nguyên thì C# đưa ra cú pháp chỉ dẫn using, cú pháp này đảm bảo phương thức

Dispose() sẽ được gọi sớm nhất có thể được. Ý tưởng là khai báo các đối tượng với cú pháp using và sau đó tạo một phạm vi hoạt động cho các đối tượng này trong khối được bao bởi dấu ({}). Khi khối phạm vi này kết thúc, thì phương thức Dispose() của đối tượng sẽ được gọi một cách tự động.

Trong phần khai báo đầu của ví dụ thì đối tượng Font được khai báo bên trong câu lệnh using. Khi câu lệnh using kết thúc, thì phương thức Dispose của đối tượng Font sẽ được gọi.

Còn trong phần khai báo thứ hai, một đối tượng Font được tạo bên ngoài câu lệnh using. Khi ta quyết định dùng đối tượng này thì ta đặt nó vào câu lệnh using. Và cũng tương tự như trên khi khối câu lệnh using được thực hiện xong thì phương thức Dispose() của font được gọi.

3.1.5. Truyền tham số

Như đã thảo luận trong chương trước, tham số có kiểu dữ liệu là giá trị thì sẽ được truyền giá trị vào cho phương thức. Điều này có nghĩa rằng khi một đối tượng có kiểu là giá trị được truyền vào cho một phương thức, thì có một bản sao chép đối tượng đó được tạo ra bên trong phương thức. Một khi phương thức được thực hiện xong thì đối tượng sao chép này sẽ được hủy. Tuy nhiên, đây chỉ là trường hợp bình thường, ngôn ngữ C# còn cung cấp khả năng cho phép ta truyền các đối tượng có kiểu giá trị dưới hình thức là tham chiếu. Ngôn ngữ C# đưa ra một bổ sung tham số là ref cho phép truyền các đối tượng giá trị vào trong phương thức theo kiểu tham chiếu. Và tham số bổ sung out trong trường hợp muốn truyền dưới dạng tham chiếu mà không cần phải khởi tạo giá trị ban đầu cho tham số truyền. Ngoài ra ngôn ngữ C# còn hỗ trợ bổ sung params cho phép phương thức chấp nhận nhiều số lượng các tham số.

3.1.5.1. Truyền tham chiếu

Những phương thức chỉ có thể trả về duy nhất một giá trị, mặc dù giá trị này có thể là một tập hợp các giá trị. Nếu chúng ta muốn phương thức trả về nhiều hơn một giá trị thì cách thực hiện là tạo các tham số dưới hình thức tham chiếu. Khi đó trong phương thức ta sẽ xử lý và gán các giá trị mới cho các tham số tham chiếu này, kết quả là sau khi

phương thức thực hiện xong ta dùng các tham số truyền vào như là các kết quả trả về. Ví dụ 4.7 sau minh họa việc truyền tham số tham chiếu cho phương thức.

- Ví dụ 3.7: Trả giá trị trả về thông qua tham số.

```
using System;
public class Time
{
    public void DisplayCurrentTime()
    {
        Console.WriteLine("{0}/{1}/{2}/ {3}:{4}:{5}", Date,
Month, Year, Hour,
Minute, Second);
    }
    public int GetHour()
    {
        return Hour;
    }
    public void GetTime(int h, int m, int s)
    {
        h = Hour;
        m = Minute;
        s = Second;
    }
    public Time(System.DateTime dt)
    {
        Year = dt.Year; Month = dt.Month; Date = dt.Day; Hour
= dt.Hour;
        Minute = dt.Minute; Second = dt.Second;
    }
    private int Year;
    private int Month;
```

```
    private int Date;
    private int Hour;
    private int Minute;
    private int Second;
}
public class Tester
{
    static void Main()
    {
        System.DateTime currentTime = System.DateTime.Now;
        Time t = new
        Time(currentTime); t.DisplaycurrentTime();
        int theHour = 0; int theMinute = 0; int theSecond = 0;
        t.GetTime(theHour, theMinute, theSecond);
        System.Console.WriteLine("Current
time: {0}:{1}:{2}",
        theHour, theMinute, theSecond);
    }
}
```

□ Kết quả:

8/6/2002 14:15:20

Current time: 0:0:0

Như ta thấy, kết quả xuất ra ở dòng cuối cùng là ba giá trị 0:0:0, rõ ràng phương thức GetTime() không thực hiện như mong muốn là gán giá trị Hour, Minute, Second cho các tham số truyền vào. Tức là ba tham số này được truyền vào dưới dạng giá trị. Do đó để thực hiện như mục đích của chúng ta là lấy các giá trị của Hour, Minute, Second thì phương thức GetTime() có ba tham số được truyền dưới dạng tham chiếu. Ta thực hiện như sau, đầu tiên, thêm là thêm khai báo ref vào trước các tham số trong phương thức GetTime():

```
public void GetTime( ref int h, ref int m, ref int s)
{
    h = Hour;
    m = Minute;
    s = Second;
}
```

Điều thay đổi thứ hai là bổ sung cách gọi hàm GetTime để truyền các tham số dưới dạng tham chiếu như sau:

```
t.GetTime( ref theHour, ref theMinute, ref theSecond);
```

Nếu chúng ta không thực hiện bước thứ hai, tức là không đưa từ khóa ref khi gọi hàm thì trình biên dịch C# sẽ báo một lỗi rằng không thể chuyển tham số từ kiểu int sang kiểu ref int.

Cuối cùng khi biên dịch lại chương trình ta được kết quả đúng như yêu cầu. Bằng việc khai báo tham số tham chiếu, trình biên dịch sẽ truyền các tham số dưới dạng các tham chiếu, thay cho việc tạo ra một bản sao chép các tham số này. Khi đó các tham số bên trong GetTime() sẽ tham chiếu đến cùng biến đã được khai báo trong hàm Main(). Như vậy mọi sự thay đổi với các biến này đều có hiệu lực tương tự như là thay đổi trong hàm Main().

Tóm lại cơ chế truyền tham số dạng tham chiếu sẽ thực hiện trên chính đối tượng được đưa vào. Còn cơ chế truyền tham số giá trị thì sẽ tạo ra các bản sao các đối tượng được truyền vào, do đó mọi thay đổi bên trong phương thức không làm ảnh hưởng đến các đối tượng được truyền vào dưới dạng giá trị.

3.1.5.2. Truyền tham chiếu với biến chưa khởi tạo

Ngôn ngữ C# bắt buộc phải thực hiện một phép gán cho biến trước khi sử dụng, do đó khi khai báo một biến như kiểu cơ bản thì trước khi có lệnh nào sử dụng các biến này thì phải có lệnh thực hiện việc gán giá trị xác định cho biến. Như trong ví dụ 3.7 trên, nếu chúng ta không khởi tạo biến theHour, theMinute, và biến theSecond

trước khi truyền như tham số vào phương thức GetTime() thì trình biên dịch sẽ báo lỗi. Nếu chúng ta sửa lại đoạn mã của ví dụ 3.7 như sau:

```
int theHour; int theMinute; int theSecond;  
  
t.GetTime( ref int theHour, ref int theMinute, ref int  
theSecond);
```

Việc sử dụng các đoạn lệnh trên không phải hoàn toàn vô lý vì mục đích của chúng ta là nhận các giá trị của đối tượng Time, việc khởi tạo giá trị của các biến đưa vào là không cần thiết. Tuy nhiên khi biên dịch với đoạn mã lệnh như trên sẽ được báo các lỗi sau:

Use of unassigned local variable ‘theHour’ Use of unassigned local variable
‘theMinute’ Use of unassigned local variable ‘theSecond’

Để mở rộng cho yêu cầu trong trường hợp này ngôn ngữ C# cung cấp thêm một bổ sung tham chiếu là out. Khi sử dụng tham chiếu out thì yêu cầu bắt buộc phải khởi tạo các tham số tham chiếu được bỏ qua. Như các tham số trong phương thức GetTime(), các tham số này không cung cấp bất cứ thông tin nào cho phương thức mà chỉ đơn giản là cơ chế nhận thông tin và đưa ra bên ngoài. Do vậy ta có thể đánh dấu tất cả các tham số tham chiếu này là out, khi đó ta sẽ giảm được công việc phải khởi tạo các biến này trước khi đưa vào phương thức. Lưu ý là bên trong phương thức có các tham số tham chiếu out thì các tham số này phải được gán giá trị trước khi trả về. Ta có một số thay đổi cho phương thức GetTime() như sau:

```
public void GetTime( out int h, out int m, out int s)  
{  
    h = Hour;  
    m = Minute;  
    s = Second;  
}
```

và cách gọi mới phương thức GetTime() trong Main():

```
t.GetTime( out theHour, out theMinute, out theSecond);
```

Tóm lại ta có các cách khai báo các tham số trong một phương thức như sau: kiểu dữ liệu giá trị được truyền vào phương thức bằng giá trị. Sử dụng tham chiếu ref để truyền kiểu dữ liệu giá trị vào phương thức dưới dạng tham chiếu, cách này cho phép vừa sử dụng và có khả năng thay đổi các tham số bên trong phương thức được gọi. Tham chiếu out được sử dụng chỉ để trả về giá trị từ một phương thức. Ví dụ 3.8 sau sử dụng ba kiểu tham số như trên.

□ Ví dụ 3.8: Sử dụng tham số.

```
using System;

public class Time

{
    public void DisplayCurrentTime()

    {
        Console.WriteLine("{0}/{1}/{2} {3}:{4}:{5}", Date,
Month, Year, Hour,
Minute, Second);

    }

    public int GetHour()

    {
        return Hour;
    }

    public void SetTime(int hr, out int min, ref int sec)

    {
        // Nếu số giây truyền vào >30 thì tăng số Minute và
        Second = 0

        if (sec >= 30)
        {
    }
```

```
        Minute++; Second = 0;

    }

    Hour = hr; // thiết lập giá trị hr được truyền vào
    // Trả về giá trị mới cho min và sec min = Minute;
    sec = Second;

}

public Time(System.DateTime dt)
{
    Year = dt.Year; Month = dt.Month; Date = dt.Day; Hour
= dt.Hour;

    Minute = dt.Minute; Second = dt.Second;

}

// biến thành viên private private int Year;
private int Year; private int Month; private int Date;
private int Hour; private int Minute;

private int Second;

}

public class Tester
{

    static void Main()
    {

        System.DateTime currentTime = System.DateTime.Now;

        Time t = new
Time(currentTime); t.DisplayCurrentTime();

        int theHour = 3;

        int theMinute;
```

```
int theSecond = 20;

    t.SetTime(theHour, out theMinute, ref theSecond);
Console.WriteLine("The Minute is now: {0} and {1} seconds",
                  theMinute, theSecond);

    theSecond = 45;

    t.SetTime(theHour, out theMinute, ref theSecond);
Console.WriteLine("The Minute is now: {0} and {1} seconds",
                  theMinute, theSecond);

}
```

□ Kết quả:

8/6/2002 15:35:24

The Minute is now: 35 and 24 seconds

The Minute is now: 36 and 0 seconds

Phương thức SetTime trên đã minh họa việc sử dụng ba kiểu truyền tham số vào một phương thức. Tham số thứ nhất theHour được truyền vào dạng giá trị, mục đích của tham số này là để thiết lập giá trị cho biến thành viên Hour và tham số này không được sử dụng để về bất cứ giá trị nào.

Tham số thứ hai là theMinute được truyền vào phương thức chỉ để nhận giá trị trả về của biến thành viên Minute, do đó tham số này được khai báo với từ khóa out.

Cuối cùng tham số theSecond được truyền vào với khai báo ref, biến tham số này vừa dùng để thiết lập giá trị trong phương thức. Nếu theSecond lớn hơn 30 thì giá trị của biến thành viên Minute tăng thêm một đơn vị và biến thành viên Second được thiết lập về 0. Sau cùng thì theSecond được gán giá trị của biến thành viên Second và được trả về.

Do hai biến theHour và theSecond được sử dụng trong phương thức SetTime nên phải được khởi tạo trước khi truyền vào phương thức. Còn với biến theMinute thì không cần thiết vì nó không được sử dụng trong phương thức mà chỉ nhận giá trị trả về.

3.1.6. Nạp chồng phương thức

Thông thường khi xây dựng các lớp, ta có mong muốn là tạo ra nhiều hàm có cùng tên. Cũng như hầu hết trong các ví dụ trước thì các lớp điều có nhiều hơn một phương thức khởi dụng. Như trong lớp Time có các phương thức khởi dụng nhận các tham số khác nhau, như tham số là đối tượng DateTime, hay tham số có thể được tùy chọn để thiết lập các giá trị của các biến thành viên thông qua các tham số nguyên. Tóm lại ta có thể xây dựng nhiều các phương thức cùng tên nhưng nhận các tham số khác nhau. Chức năng này được gọi là nạp chồng phương thức.

Một ký hiệu (signature) của một phương thức được định nghĩa như tên của phương thức cùng với danh sách tham số của phương thức. Hai phương thức khác nhau khi ký hiệu của chúng khác là khác nhau tức là khác nhau khi tên phương thức khác nhau hay danh sách tham số khác nhau. Danh sách tham số được xem là khác nhau bởi số lượng các tham số hoặc là kiểu dữ liệu của tham số. Ví dụ đoạn mã sau, phương thức thứ nhất khác phương thức thứ hai do số lượng tham số khác nhau. Phương thức thứ hai khác phương thức thứ ba do kiểu dữ liệu tham số khác nhau:

```
void myMethod( int p1 );  
void myMethod( int p1, int p2 );  
void myMethod( int p1, string p2 );
```

Một lớp có thể có bất cứ số lượng phương thức nào, nhưng mỗi phương thức trong lớp phải có ký hiệu khác với tất cả các phương thức thành viên còn lại của lớp.

Ví dụ 3.9 minh họa lớp Time có hai phương thức khởi dụng, một phương thức nhận tham số là một đối tượng DateTime còn phương thức thứ hai thì nhận sáu tham số nguyên.

- Ví dụ 3.9: Minh họa nạp chồng phương thức khởi dụng.

```
using System;  
public class Time  
{
```

```
    public void DisplayCurrentTime()
    {
        Console.WriteLine("{0}/{1}/{2} {3}:{4}:{5}", Date,
Month, Year, Hour,
Minute, Second);
    }
    public Time(System.DateTime dt)
    {
        Year = dt.Year; Month = dt.Month; Date = dt.Day; Hour
= dt.Hour;
        Minute = dt.Minute; Second = dt.Second;
    }
    public Time(int Year, int Month, int Date, int Hour, int
Minute, int Second)
    {
        this.Year = Year; this.Month = Month; this.Date =
Date; this.Hour = Hour;
        this.Minute = Minute; this.Second = Second;
    }
    // Biến thành viên private private int Year;
    private int Year;
    private int Month;
    private int Date;
    private int Hour;
    private int Minute;
    private int Second;
    public class Tester
    {
        static void Main()
        {
```

```
System.DateTime currentTime = System.DateTime.Now;
Time t1 = new
    Time(currentTime); t1.DisplaycurrentTime();
    Time t2 = new Time(2002, 6, 8, 18, 15, 20);
    t2.DisplaycurrentTime();
}
}
```

□ Kết quả:

2/1/2002 17:50:17

8/6/2002 18:15:20

Như chúng ta thấy, lớp Time trong ví dụ minh họa 4.9 có hai phương thức khởi dụng. Nếu hai phương thức có cùng ký hiệu thì trình biên dịch sẽ không thể biết được gọi phương thức nào khi khởi tạo hai đối tượng là t1 và t2. Tuy nhiên, ký hiệu của hai phương thức này khác nhau vì tham số truyền vào khác nhau, do đó trình biên dịch sẽ xác định được phương thức nào được gọi dựa vào các tham số.

Khi thực hiện nạp chồng một phương thức, bắt buộc chúng ta phải thay đổi ký hiệu của phương thức, số tham số, hay kiểu dữ liệu của tham số. Chúng ta cũng có thể toàn quyền thay đổi giá trị trả về, nhưng đây là tùy chọn. Nếu chỉ thay đổi giá trị trả về thì không phải nạp chồng phương thức mà khi đó hai phương thức khác nhau, và nếu tạo ra hai phương thức cùng ký hiệu nhưng khác nhau kiểu giá trị trả về sẽ tạo ra một lỗi biên dịch.

□ Ví dụ 3.10: Nạp chồng phương thức.

```
using System;
public class Tester
{
    private int Triple(int val)
    {
```

```
        return 3 * val;  
    }  
  
    private long Triple(long val)  
    {  
        return 3 * val;  
    }  
  
    public void Test()  
    {  
  
        int x = 5;  
        int y = Triple(x);  
        Console.WriteLine("x: {0} y: {1}", x, y);  
        long lx = 10;  
        long ly = Triple(lx);  
        Console.WriteLine("lx: {0} ly:{1}", lx, ly);  
    }  
  
    static void Main()  
    {  
        Tester t = new Tester();  
        t.Test();  
    }  
}
```

- Kết quả: x: 5 y: 15 lx: 10 ly:30

Trong ví dụ này, lớp Tester nạp chồng hai phương thức Triple(), một phương thức nhận tham số nguyên int, phương thức còn lại nhận tham số là số nguyên long. Kiểu giá trị trả về của hai phương thức khác nhau, mặc dù điều này không đòi hỏi nhưng rất thích hợp trong trường hợp này.

3.1.7. Đóng gói dữ liệu với thành phần thuộc tính

Thuộc tính là khái niệm cho phép truy cập trạng thái của lớp thay vì thông qua truy cập trực tiếp các biến thành viên, nó sẽ được thay thế bằng việc thực thi truy cập thông qua phương thức của lớp.

Đây thật sự là một điều lý tưởng. Các thành phần bên ngoài (client) muốn truy cập trạng thái của một đối tượng và không muốn làm việc với những phương thức. Tuy nhiên, người thiết kế lớp muốn dấu trạng thái bên trong của lớp mà anh ta xây dựng, và cung cấp một cách gián tiếp thông qua một phương thức.

Thuộc tính là một đặc tính mới được giới thiệu trong ngôn ngữ C#. Đặc tính này cung cấp khả năng bảo vệ các trường dữ liệu bên trong một lớp bằng việc đọc và viết chúng thông qua thuộc tính. Trong ngôn ngữ khác, điều này có thể được thực hiện thông qua việc tạo các phương thức lấy dữ liệu (getter method) và phương thức thiết lập dữ liệu (setter method).

Thuộc tính được thiết kế nhằm vào hai mục đích: cung cấp một giao diện đơn cho phép truy cập các biến thành viên, Tuy nhiên cách thức thực thi truy cập giống như phương thức trong đó các dữ liệu được che dấu, đảm bảo cho yêu cầu thiết kế hướng đối tượng. Để hiểu rõ đặc tính này ta sẽ xem ví dụ 4.11 bên dưới:

- Ví dụ 4.11: Sử dụng thuộc tính.

```
using System;
public class Time
{
    public void DisplayCurrentTime()
    {
        Console.WriteLine("Time\t: {0}/{1}/{2} {3}:{4}:{5}",
date, month, year,
hour, minute, second);
    }
    public Time(System.DateTime dt)
```

```
    {
        year = dt.Year; month = dt.Month; date = dt.Day; hour
= dt.Hour;
        minute = dt.Minute;
        second = dt.Second;
    }
    public int Hour
    {
        get
        {
            return hour;
        }
        set
        {
            hour = value;
        }
    }
    // Biến thành viên
    private int year;
    private int month;
    private int date;
    private int hour; private int minute; private int second;
}
public class Tester
{
    static void Main()
    {
        System.DateTime currentTime = System.DateTime.Now;
        Time t = new
Time(currentTime); t.DisplayCurrentTime();
```

```
// Lấy dữ liệu từ thuộc tính Hour int theHour =  
t.Hour;  
  
    int theHour = t.Hour;  
  
    Console.WriteLine("Retrieved the hour: {0}", theHour);  
  
    theHour++;  
  
    t.Hour = theHour;  
  
    Console.WriteLine("Updated the hour: {0}", theHour);  
  
    Console.ReadLine();  
}  
}
```

□ Kết quả:

Time : 2/1/2003 17:55:1

Retrieved the hour: 17

Updated the hour: 18

Để khai báo thuộc tính, đầu tiên là khai báo tên thuộc tính để truy cập, tiếp theo là phần thân định nghĩa thuộc tính nằm trong cặp dấu ({}). Bên trong thân của thuộc tính là khai báo hai bộ truy cập lấy và thiết lập dữ liệu:

```
public int Hour  
{  
    get { return hour; }  
    set { hour = value; }  
}
```

Mỗi bộ truy cập được khai báo riêng biệt để làm hai công việc khác nhau là lấy hay thiết lập giá trị cho thuộc tính. Giá trị thuộc tính có thể được lưu trong cơ sở dữ liệu, khi đó trong phần thân của bộ truy cập sẽ thực hiện các công việc tương tác với cơ sở dữ liệu. Hoặc là giá trị thuộc tính được lưu trữ trong các biến thành viên của lớp như trong ví dụ:

```
private int hour;
```

3.1.7.1. Truy cập lấy dữ liệu (get accessor)

Phần khai báo tương tự như một phương thức của lớp dùng để trả về một đối tượng có kiểu dữ liệu của thuộc tính. Trong ví dụ trên, bộ truy cập lấy dữ liệu get của thuộc tính Hour cũng tương tự như một phương thức trả về một giá trị int. Nó trả về giá trị của biến thành viên hour nơi mà giá trị của thuộc tính Hour lưu trữ:

```
get
{
    return hour;
}
```

Trong ví dụ này, một biến thành viên cục bộ được trả về, nhưng nó cũng có thể truy cập dễ dàng một giá trị nguyên từ cơ sở dữ liệu, hay thực hiện việc tính toán tùy ý.

Bất cứ khi nào chúng ta tham chiếu đến một thuộc tính hay là gán giá trị thuộc tính cho một biến thì bộ truy cập lấy dữ liệu get sẽ được thực hiện để đọc giá trị của thuộc tính:

```
Time t = new Time( currentTime );
int theHour = t.Hour;
```

Khi lệnh thứ hai được thực hiện thì giá trị của thuộc tính sẽ được trả về, tức là bộ truy cập lấy dữ liệu get sẽ được thực hiện và kết quả là giá trị của thuộc tính được gán cho biến cục bộ theHour.

3.1.7.2. Bộ truy cập thiết lập dữ liệu (set accessor)

Bộ truy cập này sẽ thiết lập một giá trị mới cho thuộc tính và tương tự như một phương thức trả về một giá trị void. Khi định nghĩa bộ truy cập thiết lập dữ liệu chúng ta phải sử dụng từ khóa value để đại diện cho tham số được truyền vào và được lưu trữ bởi thuộc tính:

```
set
{
    hour = value;
}
```

Như đã nói trước, do ta đang khai báo thuộc tính lưu trữ dưới dạng biến thành viên nên trong phần thân của bộ truy cập ta chỉ sử dụng biến thành viên mà thôi. Bộ truy cập thiết lập hoàn toàn cho phép chúng ta có thể viết giá trị vào trong cơ sở dữ liệu hay cập nhật bất cứ biến thành viên nào khác của lớp nếu cần thiết.

Khi chúng ta gán một giá trị cho thuộc tính thì bộ truy cập thiết lập dữ liệu set sẽ được tự động thực hiện và một tham số ngầm định sẽ được tạo ra để lưu giá trị mà ta muốn gán:

```
theHour++;  
t.Hour = theHour;
```

Lợi ích của hướng tiếp cận này cho phép các thành phần bên ngoài (client) có thể tương tác với thuộc tính một cách trực tiếp, mà không phải hy sinh việc che dấu dữ liệu cũng như đặc tính đóng gói dữ liệu trong thiết kế hướng đối tượng.

3.1.8. Thuộc tính chỉ đọc

Giả sử chúng ta muốn tạo một phiên bản khác cho lớp Time cung cấp một số giá trị static để hiển thị ngày và giờ hiện hành. Ví dụ 3.12 minh họa cho cách tiếp cận này.

- Ví dụ 3.12: Sử dụng thuộc tính hằng static.

```
using System;  
  
public class RightNow  
{  
    // Định nghĩa bộ khởi tạo static cho các biến static static RightNow()  
    System.DateTime dt = System.DateTime.Now; Year = dt.Year;  
    Month = dt.Month; Date = dt.Day; Hour = dt.Hour;  
    Minute = dt.Minute; Second = dt.Second;  
}  
  
// Biến thành viên static public static int Year; public static int  
Month; public static int Date;  
  
public static int Hour; public static int Minute;
```

```
public static int Second;  
}  
  
public class Tester  
{  
    static void Main()  
    {  
        Console.WriteLine("This year: {0}", RightNow.Year.ToString());  
        RightNow.Year = 2003;  
        Console.WriteLine("This year: {0}", RightNow.Year.ToString());  
    }  
}
```

□ Kết quả:

This year: 2002

This year: 2003

Đoạn chương trình trên hoạt động tốt, tuy nhiên cho đến khi có một ai đó thay đổi giá trị của biến thành viên này. Như ta thấy, biến thành Year trên đã được thay đổi đến 2003. Điều này thực sự không như mong muốn của chúng ta.

Chúng ta muốn đánh dấu các thuộc tính tĩnh này không được thay đổi. Nhưng khai báo hằng cũng không được vì biến tĩnh không được khởi tạo cho đến khi phương thức khởi động static được thi hành. Do vậy C# cung cấp thêm từ khóa readonly phục vụ chính xác cho mục đích trên. Với ví dụ trên ta có cách khai báo lại như sau:

```
public static readonly int Year; public static readonly int  
Month; public static readonly int Date; public static readonly int  
Hour; public static readonly int Minute; public static readonly int  
Second;
```

Khi đó ta phải bỏ lệnh gán biến thành viên Year, vì nếu không sẽ bị báo lỗi:

```
// RightNow.Year = 2003; // error
```

Chương trình sau khi biên dịch và thực hiện như mục đích của chúng ta.

3.2. Kế thừa – Đa hình

Trong chương trước đã trình bày cách tạo ra những kiểu dữ liệu mới bằng việc xây dựng các lớp đối tượng. Tiếp theo chương này sẽ đưa chúng ta đi sâu vào mối quan hệ giữa những đối tượng trong thế giới thực và cách mô hình hóa những quan hệ trong xây dựng chương trình. Chương 3 cũng giới thiệu khái niệm đặc biệt hóa (specialization) được cài đặt trong ngôn ngữ C# thông qua sự kế thừa (inheritance).

Khái niệm đa hình (polymorphism) cũng được trình bày trong chương 5, đây là khái niệm quan trọng trong lập trình hướng đối tượng. Khái niệm này cho phép các thể hiện của lớp có liên hệ với nhau có thể được xử lý theo một cách tổng quát.

Cuối cùng là phần trình bày về các lớp cô lập (sealed class) không được đặt biệt hóa, hay các lớp trừu tượng sử dụng trong đặc biệt hóa. Lớp đối tượng Object là gốc của tất cả các lớp cũng được thảo luận ở phần cuối chương.

3.2.1. Đặc biệt hóa và tổng quát hóa

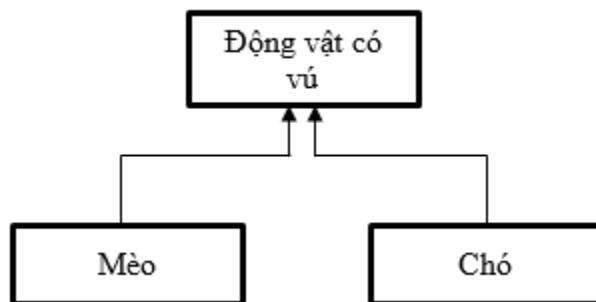
Lớp và các thể hiện của lớp tức đối tượng tuy không tồn tại trong cùng một khối, nhưng chúng tồn tại trong một mạng lưới sự phụ thuộc và quan hệ lẫn nhau. Ví dụ như con người và xã hội động vật cùng sống trong một thế giới có quan hệ loài với nhau.

Quan hệ là một (is-a) là một sự đặc biệt hóa. Khi chúng ta nói rằng mèo là một loại động vật có vú, có nghĩa là chúng ta đã nói rằng mèo là một trường hợp đặc biệt của loại động vật có vú. Nó có tất cả các đặc tính của bất cứ động vật có vú nào (như sinh ra con, có sữa mẹ và có lông...). Tuy nhiên, mèo có thêm các đặc tính riêng được xác định trong họ nhà mèo mà các họ động vật có vú khác không có được. Chó cũng là loại động vật có vú, chó cũng có tất cả các thuộc tính của động vật có vú, và riêng nó còn có thêm các thuộc tính riêng xác định họ loài chó mà khác với các thuộc tính đặc biệt của loài khác ví dụ như mèo chẳng hạn.

Quan hệ đặc biệt hóa và tổng quát hóa là hai mối quan hệ đối ngẫu và phân cấp với nhau. Chúng có quan hệ đối ngẫu vì đặc biệt được xem như là mặt ngược lại của tổng

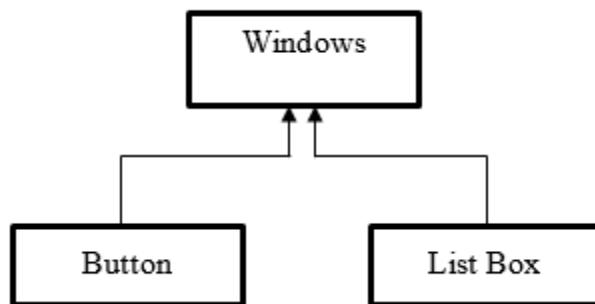
quát. Do đó, loài chó và mèo là trường hợp đặc biệt của động vật có vú. Ngược lại động vật có vú là trường hợp tổng quát từ các loài chó và mèo.

Mỗi quan hệ là phân cấp bởi vì chúng ta tạo ra một cây quan hệ, trong đó các trường hợp đặc biệt là những nhánh của trường hợp tổng quát. Trong cây phân cấp này nếu di chuyển lên trên cùng ta sẽ được trường hợp tổng quát hóa, và ngược lại nếu di chuyển xuống ngược nhánh thì ta được trường hợp đặc biệt hóa. Ta có sơ đồ phân cấp minh họa cho loài chó, mèo và động vật có vú như trên:



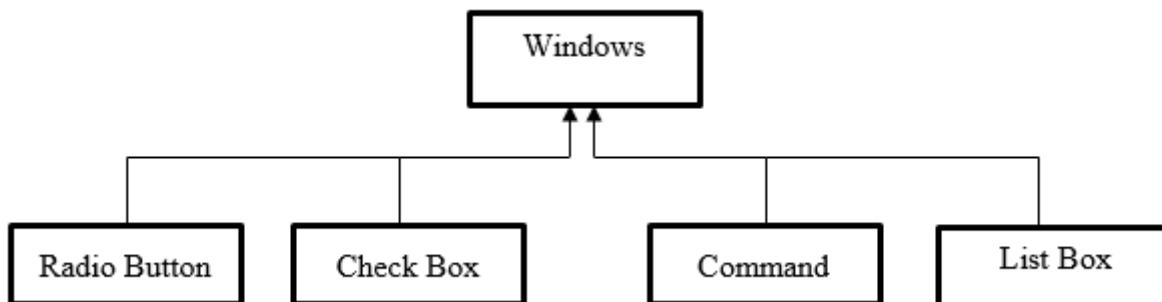
Hình 3.1. Sơ đồ phân cấp minh họa động vật có vú

Tương tự, khi chúng ta nói rằng ListBox và Button là những Window, ta phải chỉ ra những đặc tính và hành vi của những Window có trong cả hai lớp trên. Hay nói cách khác, Window là tổng quát hóa chia sẻ những thuộc tính của hai lớp ListBox và Button, trong khi đó mỗi trường hợp đặc biệt ListBox và Button sẽ có riêng những thuộc tính và hành vi đặc thù khác



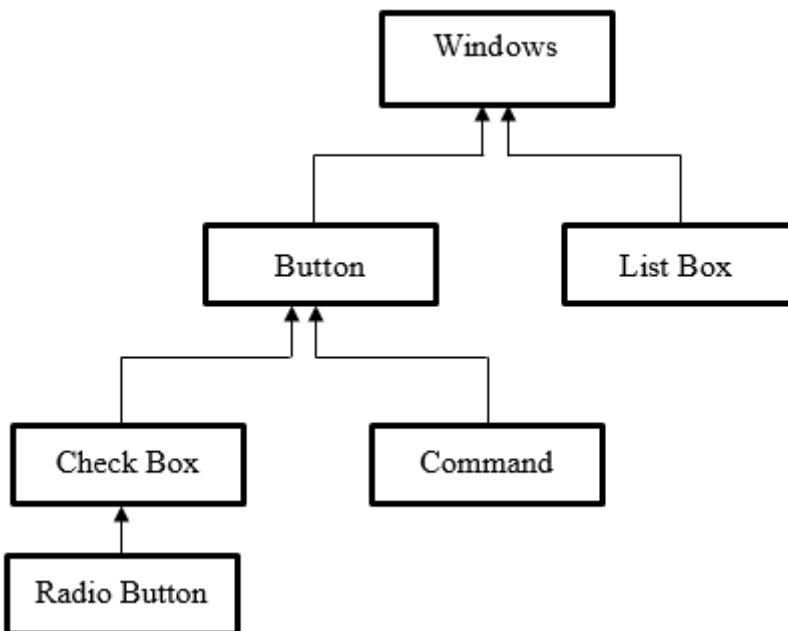
Hình 3.2. Sơ đồ phân cấp minh họa Windows đơn giản

Thông thường lưu ý rằng khi hai lớp chia sẻ chức năng với nhau, thì chúng được trích ra các phần chung và đưa vào lớp cơ sở chia sẻ. Điều này hết sức có lợi, vì nó cung cấp khả năng cao để sử dụng lại các mã nguồn chung và dễ dàng duy trì mã nguồn.



Hình 3.3. Sơ đồ phân cấp minh họa Windows được trích ra

Giả sử chúng ta bắt đầu tạo một loạt các lớp đối tượng theo hình vẽ 3.3 như bên trên. Sau khi làm việc với RadioButton, CheckBox, và CommandButton một thời gian ta nhận thấy chúng chia sẻ nhiều thuộc tính và hành vi đặc biệt hơn Window nhưng lại khá tổng quát cho cả ba lớp này. Như vậy ta có thể chia các thuộc tính và hành vi thành một nhóm lớp cơ sở riêng lấy tên là Button. Sau đó ta sắp xếp lại cấu trúc kế thừa như hình vẽ 3.4. Đây là ví dụ về cách tổng quát hóa được sử dụng để phát triển hướng đối tượng.



Hình 3.4. Sơ đồ tổng quát hóa được sử dụng để phát triển đối tượng

Trong mô hình UML trên được vẽ lại quan hệ giữa các lớp. Trong đó cả hai lớp Button và ListBox đều dẫn xuất từ lớp Window, trong đó Button có trường hợp đặc biệt là CheckBox và Command. Cuối cùng thì RadioButton được dẫn xuất từ CheckBox. Chúng ta cũng có thể nói rằng RadioButton là một CheckBox, và tiếp tục CheckBox là một Button, và cuối cùng Button là Window.

Sự thiết kế trên không phải là duy nhất hay cách tốt nhất để tổ chức những đối tượng, nhưng đó là khởi điểm để chúng ta hiểu về cách quan hệ giữa đối tượng với các đối tượng khác.

3.2.2. Sự kế thừa

Trong ngôn ngữ C#, quan hệ đặc biệt hóa được thực thi bằng cách sử dụng sự kế thừa. Đây không phải là cách duy nhất để thực thi đặc biệt hóa, nhưng nó là cách chung nhất và tự nhiên nhất để thực thi quan hệ này.

Trong mô hình trước, ta có thể nói ListBox kế thừa hay được dẫn xuất từ Window. Window được xem như là lớp cơ sở, và ListBox được xem như là lớp dẫn xuất. Như vậy, ListBox dẫn xuất tất cả các thuộc tính và hành vi từ lớp Window và thêm những phần đặc biệt riêng để xác nhận ListBox.

3.2.2.1. Thực thi kế thừa

Trong ngôn ngữ C# để tạo một lớp dẫn xuất từ một lớp ta thêm dấu hai chấm vào sau tên lớp dẫn xuất và trước tên lớp cơ sở:

```
public class ListBox : Window
```

Đoạn lệnh trên khai báo một lớp mới tên là ListBox, lớp này được dẫn xuất từ Window. Chúng ta có thể đọc dấu hai chấm có thể được đọc như là “dẫn xuất từ”.

Lớp dẫn xuất sẽ kế thừa tất cả các thành viên của lớp cơ sở, bao gồm tất cả các phương thức và biến thành viên của lớp cơ sở. Lớp dẫn xuất được tự do thực thi các phiên bản của một phương thức của lớp cơ sở. Lớp dẫn xuất cũng có thể tạo một phương

thức mới bằng việc đánh dấu với từ khóa new. Ví dụ 3.13 sau minh họa việc tạo và sử dụng các lớp cơ sở và dẫn xuất.

- Ví dụ 3.13: Sử dụng lớp dẫn xuất.

```
using System;

public class Window

{
    // Hàm khởi dựng lấy hai số nguyên chỉ
    // đến vị trí của cửa sổ trên console
    public Window( int top, int left)

    {
        this.top = top;
        this.left = left;
    }

    // mô phỏng vẽ cửa sổ
    public void DrawWindow()

    {
        Console.WriteLine("Drawing Window at {0}, {1}", top, left);
    }

    // Có hai biến thành viên private do đó
    // hai biến này sẽ không thấy bên trong lớp
    // dẫn xuất. private int top; private int left;
}

// ListBox dẫn xuất từ Window
public class ListBox: Window

{
```

```
// Khởi động có tham số

public ListBox(int top, int left, string theContents) : base(top,
left) // gọi khởi động của lớp cơ sở

{
    mListBoxContents = theContents;
}

// Tạo một phiên bản mới cho phương thức DrawWindow
// vì trong lớp dẫn xuất muốn thay đổi hành vi thực hiện
// bên trong phương thức này public new void DrawWindow()

{
    base.DrawWindow();

    Console.WriteLine(" ListBox write: {0}", mListBoxContents);
}

// biến thành viên private

private string mListBoxContents;

}

public class Tester

{
    public static void Main()

    {
        // tạo đối tượng cho lớp cơ sở Window w = new Window(5, 10);

        w.DrawWindow();

        // tạo đối tượng cho lớp dẫn xuất

        ListBox lb = new ListBox( 20, 10, "Hello world!");

        lb.DrawWindow();
    }
}
```

```
}
```

□ Kết quả:

Drawing Window at: 5, 10

Drawing Window at: 20, 10

ListBox write: Hello world!

Ví dụ 3.13 bắt đầu với việc khai báo một lớp cơ sở tên Window. Lớp này thực thi một phương thức khởi dựng và một phương thức đơn giản DrawWindow. Lớp có hai biến thành viên private là top và left, hai biến này do khai báo là private nên chỉ sử dụng bên trong của lớp Window, các lớp dẫn xuất sẽ không truy cập được. ta sẽ bàn tiếp về ví dụ này trong phần tiếp theo.

3.2.2.2. Gọi phương thức khởi dựng của lớp cơ sở

Trong ví dụ 3.13, một lớp mới tên là ListBox được dẫn xuất từ lớp cơ sở Window, lớp ListBox có một phương thức khởi dựng lấy ba tham số. Trong phương thức khởi dựng của lớp dẫn xuất này có gọi phương thức khởi dựng của lớp cơ sở. Cách gọi được thực hiện bằng việc đặt dấu hai chấm ngay sau phần khai báo danh sách tham số và tham chiếu đến lớp cơ sở thông qua từ khóa base:

```
public ListBox( int theTop, int theLeft,  
    string theContents):  
    base( theTop, theLeft) // gọi khởi tạo lớp cơ sở
```

Bởi vì các lớp không được kế thừa các phương thức khởi dựng của lớp cơ sở, do đó lớp dẫn xuất phải thực thi phương thức khởi dựng riêng của nó. Và chỉ có thể sử dụng phương thức khởi dựng của lớp cơ sở thông qua việc gọi tường minh.

Một điều lưu ý trong ví dụ 5.1 là việc lớp ListBox thực thi một phiên bản mới của phương thức DrawWindow():

```
public new void DrawWindow()
```

Từ khóa new được sử dụng ở đây để chỉ ra rằng người lập trình đang tạo ra một phiên bản mới cho phương thức này bên trong lớp dẫn xuất.

Nếu lớp cơ sở có phương thức khởi dựng mặc định, thì lớp dẫn xuất không cần bắt buộc phải gọi phương thức khởi dựng của lớp cơ sở một cách tường minh. Thay vào đó phương thức khởi dựng mặc định của lớp cơ sở sẽ được gọi một cách ngầm định. Tuy nhiên, nếu lớp cơ sở không có phương thức khởi dựng mặc định, thì tất cả các lớp dẫn xuất của nó phải gọi phương thức khởi dựng của lớp cơ sở một cách tường minh thông qua việc sử dụng từ khóa base.

*Ghi chú: Nếu chúng ta không khai báo bất cứ phương thức khởi dựng nào, thì trình biên dịch sẽ tạo riêng một phương thức khởi dựng cho chúng ta. Khi mà chúng ta viết riêng các phương thức khởi dựng hay là sử dụng phương thức khởi dựng mặc định do trình biên dịch cung cấp hay không thì phương thức khởi dựng mặc định không lấy một tham số nào hết. Tuy nhiên, lưu ý rằng khi ta tạo bất cứ phương thức khởi dựng nào thì trình biên dịch sẽ không cung cấp phương thức khởi dựng cho chúng ta.

3.2.2.3. Gọi phương thức của lớp cơ sở

Trong ví dụ 3.13, phương thức DrawWindow() của lớp ListBox sẽ làm ẩn và thay thế phương thức DrawWindow của lớp cơ sở Window. Khi ta gọi phương thức DrawWindow của một đối tượng của lớp ListBox thì phương thức ListBox.DrawWindow() sẽ được thực hiện, chứ không phải phương thức Window.DrawWindow() của lớp cơ sở Window. Tuy nhiên, ta có thể gọi phương thức DrawWindow() của lớp cơ sở thông qua từ khóa base:

```
base.DrawWindow(); // gọi phương thức cơ sở
```

Từ khóa base chỉ đến lớp cơ sở cho đối tượng hiện hành.

3.2.2.4. Điều khiển truy xuất

Khả năng hiện hữu của một lớp và các thành viên của nó có thể được hạn chế thông qua việc sử dụng các bổ sung truy cập: public, private, protected, internal, và protected internal.

Như chúng ta đã thấy, public cho phép một thành viên có thể được truy cập bởi một phương thức thành viên của những lớp khác. Trong khi đó private chỉ cho phép các phương thức thành viên trong lớp đó truy xuất. Từ khóa protected thì mở rộng thêm khả năng của private cho phép truy xuất từ các lớp dẫn xuất của lớp đó. Internal mở rộng khả năng cho phép bất cứ phương thức của lớp nào trong cùng một khối kết hợp (assembly) có thể truy xuất được. Một khối kết hợp được hiểu như là một khối chia sẻ và dùng lại trong CLR. Thông thường, khối này là tập hợp các tập tin vật lý được lưu trữ trong một thư mục bao gồm các tập tin tài nguyên, chương trình thực thi theo ngôn ngữ IL,...

Từ khóa internal protected đi cùng với nhau cho phép các thành viên của cùng một khối assembly hoặc các lớp dẫn xuất của nó có thể truy cập. Chúng ta có thể xem sự thiết kế này giống như là internal hay protected.

Các lớp cũng như những thành viên của lớp có thể được thiết kế với bất cứ mức độ truy xuất nào. Một lớp thường có mức độ truy xuất mở rộng hơn cách thành viên của lớp, còn các thành viên thì mức độ truy xuất thường có nhiều hạn chế. Do đó, ta có thể định nghĩa một lớp MyClass như sau:

```
public class MyClass  
{  
    //...  
    protected int myValue  
}
```

Như trên biến thành viên myValue được khai báo truy xuất protected mặc dù bản thân lớp được khai báo là public. Một lớp public là một lớp sẵn sàng cho bất cứ lớp nào khác muốn tương tác với nó. Đôi khi một lớp được tạo ra chỉ để trợ giúp cho những lớp khác trong một khối assembly, khi đó những lớp này nên được khai báo khóa internal hơn là khóa public.

3.2.3. Đa hình

Có hai cách thức khá mạnh để thực hiện việc kế thừa. Một là sử dụng lại mã nguồn, khi chúng ta tạo ra lớp ListBox, chúng ta có thể sử dụng lại một vài các thành phần trong lớp cơ sở như Window.

Tuy nhiên, cách sử dụng thứ hai chứng tỏ được sức mạnh to lớn của việc kế thừa đó là tính đa hình (polymorphism). Theo tiếng Anh từ này được kết hợp từ poly là nhiều và morph có nghĩa là form (hình thức). Do vậy, đa hình được hiểu như là khả năng sử dụng nhiều hình thức của một kiểu mà không cần phải quan tâm đến từng chi tiết.

Khi một tổng đài điện thoại gọi cho máy điện thoại của chúng ta một tín hiệu có cuộc gọi. Tổng đài không quan tâm đến điện thoại của ta là loại nào. Có thể ta đang dùng một điện thoại cũ dùng motor để rung chuông, hay là một điện thoại điện tử phát ra tiếng nhạc số. Hoàn toàn các thông tin về điện thoại của ta không có ý nghĩa gì với tổng đài, tổng đài chỉ biết một kiểu cơ bản là điện thoại mà thôi và điện thoại này sẽ biết cách báo chuông. Còn việc báo chuông như thế nào thì tổng đài không quan tâm. Tóm lại, tổng đài chỉ cần bảo điện thoại hãy làm điều gì đó để reng. Còn phần còn lại tức là cách thức reng là tùy thuộc vào từng loại điện thoại. Đây chính là tính đa hình.

3.2.3.1. Kiểu đa hình

Do một ListBox là một Window và một Button cũng là một Window, chúng ta mong muốn sử dụng cả hai kiểu dữ liệu này trong tình huống cả hai được gọi là Window. Ví dụ như trong một form giao diện trên MS Windows, form này chứa một tập các thẻ hiện của Window. Khi form được hiển thị, nó yêu cầu tất cả các thẻ hiện của Window tự thực hiện việc tô vẽ. Trong trường hợp này, form không muốn biết thành phần thẻ hiện là loại nào như Button, CheckBox, ... Điều quan trọng là form kích hoạt toàn bộ tập hợp này tự thực hiện việc vẽ. Hay nói ngắn gọn là form muốn đối xử với những đối tượng Window này một cách đa hình

3.2.3.2. Phương thức đa hình

Để tạo một phương thức hỗ trợ tính đa hình, chúng ta cần phải khai báo khóa virtual trong phương thức của lớp cơ sở. Ví dụ, để chỉ định rằng phương thức

DrawWindow() của lớp Window trong ví dụ 3.13 là đa hình, đơn giản là ta thêm từ khóa virtual vào khai báo như sau:

```
public virtual void DrawWindow()
```

Lúc này thì các lớp dẫn xuất được tự do thực thi các cách xử riêng của mình trong phiên bản mới của phương thức DrawWindow(). Để làm được điều này chỉ cần thêm từ khóa override để chồng lên phương thức ảo DrawWindow() của lớp cơ sở. Sau đó thêm các đoạn mã nguồn mới vào phương thức viết chồng này.

Trong ví dụ minh họa 3.14 sau, lớp ListBox dẫn xuất từ lớp Window và thực thi một phiên bản riêng của phương thức DrawWindow():

```
public override void DrawWindow()
{
    base.DrawWindow();
    Console.WriteLine("Writing string to the listbox: {0}",
listBoxContents);
}
```

Từ khóa override bảo với trình biên dịch rằng lớp này thực hiện việc phủ quyết lại phương thức DrawWindow() của lớp cơ sở. Tương tự như vậy ta có thể thực hiện việc phủ quyết phương thức này trong một lớp dẫn xuất khác như Button, lớp này cũng được dẫn xuất từ Window.

Trong phần thân của ví dụ 13.4, đầu tiên ta tạo ra ba đối tượng, đối tượng thứ nhất của Window, đối tượng thứ hai của lớp ListBox và đối tượng cuối cùng của lớp Button. Sau đó ta thực hiện việc gọi phương thức DrawWindow() cho mỗi đối tượng sau:

```
Window    win = new Window( 1, 2 );
ListBox   lb = new ListBox( 3, 4, "Stand alone list box");
Button    b = new Button( 5, 6 );
```

```
win.DrawWindow(); lb.DrawWindow(); b.DrawWindow();
```

Đoạn chương trình trên thực hiện các công việc như yêu cầu của chúng ta, là từng đối tượng thực hiện công việc tô vẽ của nó. Tuy nhiên, cho đến lúc này thì chưa có bất cứ sự đa hình nào được thực thi. Mọi chuyện vẫn bình thường cho đến khi ta muốn tạo ra một mảng các đối tượng Window, bởi vì ListBox cũng là một Window nên ta có thể tự do đặt một đối tượng ListBox vào vị trí của một đối tượng Window trong mảng trên. Và tương tự ta cũng có thể đặt một đối tượng Button vào bất cứ vị trí nào trong mảng các đối tượng Window, vì một Button cũng là một Window.

```
Window[ ] winArray = new Window[3];  
winArray[0] = new Window( 1, 2 );  
winArray[1] = new ListBox( 3, 4, "List box is array");  
winArray[2] = new Button( 5, 6 );
```

Chuyện gì xảy ra khi chúng ta gọi phương thức DrawWindow() cho từng đối tượng trong mảng winArray.

```
for( int i = 0; i < 3 ; i++ )  
{  
    winArray[i].DrawWindow();  
}
```

Trình biên dịch điều biết rằng có ba đối tượng Windows trong mảng và phải thực hiện việc gọi phương thức DrawWindow() cho các đối tượng này. Nếu chúng ta không đánh dấu phương thức DrawWindow() trong lớp Window là virtual thì phương thức DrawWindow() trong lớp Window sẽ được gọi ba lần. Tuy nhiên do chúng ta đã đánh dấu phương thức này ảo ở lớp cơ sở và thực thi việc phủ quyết phương thức này ở các lớp dẫn xuất.

Khi ta gọi phương thức DrawWindow trong mảng, trình biên dịch sẽ dò ra được chính xác kiểu dữ liệu nào được thực thi trong mảng khi đó có ba kiểu sẽ được thực thi là một Window, một ListBox, và một Button. Và trình biên dịch sẽ gọi chính xác phương

thức của từng đối tượng. Đây là điều cốt lõi và tinh hoa của tính chất đa hình. Đoạn chương trình hoàn chỉnh 3.14 minh họa cho sự thực thi tính chất đa hình.

- Ví dụ 3.14: Sử dụng phương thức ảo.

```
using System;

public class Window
{
    public Window( int top, int left )
    {
        this.top = top;
        this.left = left;
    }
    // phương thức được khai báo ảo
    public virtual void DrawWindow()
    {
        Console.WriteLine( "Window: drawing window at {0}, {1}", top,
left );
    }
}
// biến thành viên của lớp
protected int top; protected int left;
public class ListBox : Window
{
    // phương thức khởi dựng có tham số
    public ListBox( int top, int left, string contents ): base( top,
left )
    {
        listBoxContents = contents;
    }
    // thực hiện việc phủ quyết phương thức DrawWindow
    public override void DrawWindow()
```

```
{  
base.DrawWindow();  
Console.WriteLine(" Writing string to the listbox: {0}",  
listBoxContents);  
  
}  
// biến thành viên của ListBox private string listBoxContents;  
}  
}  
}  
}  
public class Button : Window  
{  
public Button( int top, int left ) : base( top, left )  
// phủ quyết phương thức DrawWindow của lớp cơ sở public override  
void DrawWindow()  
{  
Console.WriteLine(" Drawing a button at {0}: {1}", top, left);  
}  
}  
}  
public class Tester  
{  
static void Main()  
{  
Window win = new Window(1,2);  
ListBox lb = new ListBox( 3, 4, " Stand alone list box"); Button b  
= new Button( 5, 6 );  
win.DrawWindow(); lb.DrawWindow(); b.DrawWindow();  
Window[] winArray = new Window[3];  
winArray[0] = new Window( 1, 2 );  
winArray[1] = new ListBox( 3, 4, "List box is array");  
winArray[2] = new Button( 5, 6 );  
for( int i = 0; i < 3; i++)  
{
```

```
winArray[i].DrawWindow();  
}  
}  
}
```

□ Kết quả:

Window: drawing window at 1: 2

Window: drawing window at 3: 4

Writing string to the listbox: Stand alone list box

Drawing a button at 5: 6

Window: drawing Window at 1: 2

Window: drawing window at 3: 4

Writing string to the listbox: List box is array

Drawing a button at 5: 6

Lưu ý trong suốt ví dụ này, chúng ta đánh dấu một phương thức phủ quyết mới với từ khóa phủ quyết override:

```
public override void DrawWindow()
```

Lúc này trình biên dịch biết cách sử dụng phương thức phủ quyết khi gặp đối tượng mang hình thức đa hình. Trình biên dịch chịu trách nhiệm trong việc phân ra kiểu dữ liệu thật của đối tượng để sau này xử lý. Do đó phương thức ListBox.DrawWindow() sẽ được gọi khi một đối tượng Window tham chiếu đến một đối tượng thật sự là ListBox.

□Ghi chú: Chúng ta phải chỉ định rõ ràng với từ khóa override khi khai báo một phương thức phủ quyết phương thức ảo của lớp cơ sở. Điều này dễ làm lẩn với người lập trình C++ vì từ khóa này trong C++ có thể bỏ qua mà trình biên dịch C++ vẫn hiểu.

3.2.3.3. Từ khóa new và override

Trong ngôn ngữ C#, người lập trình có thể quyết định phủ quyết một phương thức ảo bằng cách khai báo tường minh từ khóa override. Điều này giúp cho ta đưa ra một phiên bản mới của chương trình và sự thay đổi của lớp cơ sở sẽ không làm

ảnh hưởng đến chương trình viết trong các lớp dẫn xuất. Việc yêu cầu sử dụng từ khóa override sẽ giúp ta ngăn ngừa vấn đề này.

Bây giờ ta thử bàn về vấn đề này, giả sử lớp cơ sở Window của ví dụ trước được viết bởi một công ty A. Cũng giả sử rằng lớp ListBox và RadioButton được viết từ những người lập trình của công ty B và họ dùng lớp cơ sở Window mua được của công ty A làm lớp cơ sở cho hai lớp trên. Người lập trình trong công ty B không có hoặc có rất ít sự kiểm soát về những thay đổi trong tương lai với lớp Window do công ty A phát triển.

Khi nhóm lập trình của công ty B quyết định thêm một phương thức Sort() vào lớp ListBox:

```
public class ListBox : Window
{
    public virtual void Sort( ) {....}
}
```

Việc thêm vào vẫn bình thường cho đến khi công ty A, tác giả của lớp cơ sở Window, đưa ra phiên bản thứ hai của lớp Window. Và trong phiên bản mới này những người lập trình của công ty A đã thêm một phương thức Sort() vào lớp cơ sở Window:

```
public class Window
{
    //.....
    public virtual void Sort( ) {....}
}
```

Trong các ngôn ngữ lập trình hướng đối tượng khác như C++, phương thức ảo mới Sort() trong lớp Window bây giờ sẽ hành động giống như là một phương thức cơ sở cho phương thức ảo trong lớp ListBox. Trình biên dịch có thể gọi phương thức Sort() trong lớp ListBox khi chúng ta có ý định gọi phương thức Sort() trong Window. Trong ngôn ngữ Java, nếu phương thức Sort() trong Window có kiểu trả về khác kiểu trả về

của phương thức Sort() trong lớp ListBox thì sẽ được báo lỗi là phương thức phủ quyết không hợp lệ.

Ngôn ngữ C# ngăn ngừa sự lẩn lộn này, trong C# một phương thức ảo thì được xem như là gốc rễ của sự phân phôi ảo. Do vậy, một khi C# tìm thấy một phương thức khai báo là ảo thì nó sẽ không thực hiện bất cứ việc tìm kiếm nào trên cây phân cấp kế thừa. Nếu một phương thức ảo Sort() được trình bày trong lớp Window, thì khi thực hiện hành vi của lớp Listbox không thay đổi.

Tuy nhiên khi biên dịch lại, thì trình biên dịch sẽ đưa ra một cảnh báo như sau:

...\\class1.cs(54, 24): warning CS0114: ‘ListBox.Sort()’ hides inherited member
‘Window.Sort()’.

To make the current member override that implementation, add the override keyword.
Otherwise add the new keyword.

Để loại bỏ cảnh báo này, người lập trình phải chỉ rõ ý định của anh ta. Anh ta có thể đánh dấu phương thức ListBox.Sort() với từ khóa là new, và nó không phải phủ quyết của bất cứ phương thức ảo nào trong lớp Window:

```
public class ListBox : Window
{
    public new virtual void Sort( ) {...}
```

Việc thực hiện khai báo trên sẽ loại bỏ được cảnh báo. Mặc khác nếu người lập trình muốn phủ quyết một phương thức trong Window, thì anh ta cần thiết phải dùng từ khóa override để khai báo một cách tường minh:

```
public class ListBox : Window
{
    public override void Sort( ) {...}
```

3.2.4. Lớp trừu tượng

Mỗi lớp con của lớp Window nên thực thi một phương thức DrawWindow() cho riêng mình. Tuy nhiên điều này không thực sự đòi hỏi phải thực hiện một cách bắt buộc. Để yêu cầu các lớp con (lớp dẫn xuất) phải thực thi một phương thức của lớp cơ sở, chúng ta phải thiết kế một phương thức một cách trừu tượng.

Một phương thức trừu tượng không có sự thực thi. Phương thức này chỉ đơn giản tạo ra một tên phương thức và ký hiệu của phương thức, phương thức này sẽ được thực thi ở các lớp dẫn xuất.

Những lớp trừu tượng được thiết lập như là cơ sở cho những lớp dẫn xuất, nhưng việc tạo các thẻ hiện hay các đối tượng cho các lớp trừu tượng được xem là không hợp lệ. Một khi chúng ta khai báo một phương thức là trừu tượng, thì chúng ta phải ngăn cấm bất cứ việc tạo thẻ hiện cho lớp này.

Do vậy, nếu chúng ta thiết kế phương thức DrawWindow() như là trừu tượng trong lớp Window, chúng ta có thể dẫn xuất từ lớp này, nhưng ta không thể tạo bất cứ đối tượng cho lớp này. Khi đó mỗi lớp dẫn xuất phải thực thi phương thức DrawWindow(). Nếu lớp dẫn xuất không thực thi phương thức trừu tượng của lớp cơ sở thì lớp dẫn xuất đó cũng là lớp trừu tượng, và ta cũng không thể tạo các thẻ hiện của lớp này được.

Phương thức trừu tượng được thiết lập bằng cách thêm từ khóa abstract vào đầu của phần định nghĩa phương thức, cú pháp thực hiện như sau:

```
abstract public void DrawWindow( );
```

Do phương thức không cần phần thực thi, nên không có dấu ({ }) mà chỉ có dấu chấm phẩy (;) sau phương thức. Như thế với phương thức DrawWindow() được thiết kế là trừu tượng thì chỉ cần câu lệnh trên là đủ.

Nếu một hay nhiều phương thức được khai báo là trừu tượng, thì phần định nghĩa lớp phải được khai báo là abstract, với lớp Window ta có thể khai báo là lớp trừu tượng như sau:

```
abstract public void Window
```

Ví dụ 3.15 sau minh họa việc tạo lớp Window trừu tượng và phương thức trừu tượng DrawWindow() của lớp Window.

□ Ví dụ 3.15: Sử dụng phương thức và lớp trừu tượng.

```
using System;  
  
abstract public class Window  
{  
    // hàm khởi dựng lấy hai tham số  
    public Window( int top, int left)  
    {  
        this.top = top;  
        this.left = left;  
    }  
    // phương thức trừu tượng minh họa việc vẽ ra cửa sổ  
    abstract public void DrawWindow();  
    // biến thành viên protected protected int top;  
    protected int left;  
}  
  
// lớp ListBox dẫn xuất từ lớp Window  
public class ListBox : Window  
{  
    // hàm khởi dựng lấy ba tham số  
    public ListBox( int top, int left, string contents) : base( top,  
left)  
    {  
        listBoxContents = contents;  
    }  
    // phủ quyết phương thức trừu tượng DrawWindow()  
    public override void DrawWindow( )  
    {
```

```
        Console.WriteLine("Writing string to the listBox: {0}",  
listBoxContents);  
    }  
// biến private của lớp  
private string listBoxContents;  
}  
// lớp Button dẫn xuất từ lớp Window  
public class Button : Window  
{  
// hàm khởi tạo nhận hai tham số  
    public Button( int top, int left) : base( top, left)  
    {}  
// thực thi phương thức trừu tượng  
    public override void DrawWindow()  
    {  
        Console.WriteLine("Drawing button at {0}, {1}\n", top, left);  
    }  
}  
public class Tester  
{  
    static void Main()  
    {  
        Window[] winArray = new Window[3];  
        winArray[0] = new ListBox( 1, 2, "First List Box");  
        winArray[1] = new ListBox( 3, 4, "Second List Box"); winArray[2] =  
new Button( 5, 6);  
        for( int i=0; i <3 ; i++)  
        {  
            winArray[i].DrawWindow( );  
        }  
    }  
}
```

```
}
```

Trong ví dụ 3.15, lớp Window được khai báo là lớp trừu tượng và do vậy nên chúng ta không thể tạo bất cứ thẻ hiện nào của lớp Window. Nếu chúng ta thay thế thành viên đầu tiên của mảng:

```
winArray[0] = new ListBox( 1, 2, "First List Box");
```

bằng câu lệnh sau:

```
winArray[0] = new Window( 1, 2);
```

Thì trình biên dịch sẽ báo một lỗi như sau:

```
Cannot create an instance of the abstract class or interface 'Window'
```

Chúng ta có thể tạo được các thẻ hiện của lớp ListBox và Button, bởi vì hai lớp này đã phủ quyết phương thức trừu tượng. Hay có thể nói hai lớp này đã được xác định (ngược với lớp trừu tượng).

Hạn chế của lớp trừu tượng

Mặc dù chúng ta đã thiết kế phương thức DrawWindow() như một lớp trừu tượng để hỗ trợ cho tất cả các lớp dẫn xuất được thực thi riêng, nhưng điều này có một số hạn chế. Nếu chúng ta dẫn xuất một lớp từ lớp ListBox như lớp DropDownListBox, thì lớp này không được hỗ trợ để thực thi phương thức DrawWindow() cho riêng nó.

*Ghi chú: Khác với ngôn ngữ C++, trong C# phương thức Window.DrawWindow() không thể cung cấp một sự thực thi, do đó chúng ta sẽ không thể lấy được lợi ích của phương thức DrawWindow() bình thường dùng để chia sẻ bởi các lớp dẫn xuất.

Cuối cùng những lớp trừu tượng không có sự thực thi căn bản; chúng thể hiện ý tưởng về một sự trừu tượng, điều này thiết lập một sự giao ước cho tất cả các lớp dẫn xuất. Nói cách khác các lớp trừu tượng mô tả một phương thức chung của tất cả các lớp được thực thi một cách trừu tượng.

Ý tưởng của lớp trừu tượng Window thể hiện những thuộc tính chung cùng với những hành vi của tất cả các Window, thậm chí ngay cả khi ta không có ý định tạo thể hiện của chính lớp trừu tượng Window.

Ý nghĩa của một lớp trừu tượng được bao hàm trong chính từ “trừu tượng”. Lớp này dùng để thực thi một “Window” trừu tượng, và nó sẽ được biểu lộ trong các thể hiện xác định của Windows, như là Button, ListBox, Frame,...

Các lớp trừu tượng không thể thực thi được, chỉ có những lớp xác thực tức là những lớp dẫn xuất từ lớp trừu tượng này mới có thể thực thi hay tạo thể hiện.

*Lớp cô lập (sealed class)

Ngược với các lớp trừu tượng là các lớp cô lập. Một lớp trừu tượng được thiết kế cho các lớp dẫn xuất và cung cấp các khuôn mẫu cho các lớp con theo sau. Trong khi một lớp cô lập thì không cho phép các lớp dẫn xuất từ nó. Để khai báo một lớp cô lập ta dùng từ khóa sealed đặt trước khai báo của lớp không cho phép dẫn xuất. Hầu hết các lớp thường được đánh dấu sealed nhằm ngăn chặn các tai nạn do sự kế thừa gây ra.

Nếu khai báo của lớp Window trong ví dụ 5.3 được thay đổi từ khóa abstract bằng từ khóa sealed (cũng có thể loại bỏ từ khóa trong khai báo của phương thức DrawWindow()). Chương trình sẽ bị lỗi khi biên dịch. Nếu chúng ta cố thử biên dịch chương trình thì sẽ nhận được lỗi từ trình biên dịch:

‘ListBox’ cannot inherit from sealed class ‘Window’

Đây chỉ là một lỗi trong số những lỗi như ta không thể tạo một phương thức thành viên protected trong một lớp khai báo là sealed.

Gốc của tất cả các lớp: Lớp Object

Tất cả các lớp của ngôn ngữ C# của bất cứ kiểu dữ liệu nào thì cũng được dẫn xuất từ lớp System.Object. Thú vị là bao gồm cả các kiểu dữ liệu giá trị.

Một lớp cơ sở là cha trực tiếp của một lớp dẫn xuất. Lớp dẫn xuất này cũng có thể làm cơ sở cho các lớp dẫn xuất xa hơn nữa, việc dẫn xuất này sẽ tạo ra một cây thừa kế

hay một kiến trúc phân cấp. Lớp gốc là lớp nằm ở trên cùng cây phân cấp thừa kế, còn các lớp dẫn xuất thì nằm bên dưới. Trong ngôn ngữ C#, lớp gốc là lớp Object, lớp này nằm trên cùng trong cây phân cấp các lớp.

Lớp Object cung cấp một số các phương thức dùng cho các lớp dẫn xuất có thể thực hiện việc phủ quyết. Những phương thức này bao gồm Equals() kiểm tra xem hai đối tượng có giống nhau hay không. Phương thức GetType() trả về kiểu của đối tượng. Và phương thức ToString() trả về một chuỗi thể hiện lớp hiện hành. Sau đây là bảng tóm tắt các phương thức của lớp Object.

Phương thức	Chức năng
Equal()	So sánh bằng nhau giữa hai đối tượng
GetHashCode()	Cho phép những đối tượng cung cấp riêng những hàm băm cho sử dụng tập hợp
GetType()	Cung cấp kiểu của đối tượng
ToString()	Cung cấp chuỗi thể hiện của đối tượng
Finalize()	Dọn dẹp các tài nguyên
MemberwiseClone()	Tạo một bản sao từ đối tượng.

Ví dụ 3.16 sau minh họa việc sử dụng phương thức ToString() thừa kế từ lớp Object.

- Ví dụ 3.16: Thừa kế từ Object.

```
using System;  
  
public class SomeClass  
{  
    public SomeClass( int val )  
    {  
        value = val;  
    }  
}
```

```
// phủ quyết phương thức ToString của lớp Object
public virtual string ToString()
{
    return value.ToString();
}

// biến thành viên private lưu giá trị
private int value;
}

public class Tester
{
    static void Main( )
    {
        int i = 5;

        Console.WriteLine("The value of i is: {0}", i.ToString());
        SomeClass s = new SomeClass(7); Console.WriteLine("The value
of s is {0}", s.ToString());

        Console.WriteLine("The value of 5 is {0}", 5.ToString());
    }
}
```

□ Kết quả:

The value of i is: 5

The value of s is 7

The value of 5 is 5

Trong tài liệu của lớp Object phương thức ToString() được khai báo như sau:

```
public virtual string ToString();
```

Đây là phương thức ảo public, phương thức này trả về một chuỗi và không nhận tham số. Tất cả kiểu dữ liệu được xây dựng sẵn, như kiểu int, dẫn xuất từ lớp Object nên nó cũng có thể thực thi các phương thức của lớp Object.

Lớp SomeClass trong ví dụ trên thực hiện việc phủ quyết phương thức `ToString()`, do đó phương thức này sẽ trả về giá trị có nghĩa. Nếu chúng ta không phủ quyết phương thức `ToString()` trong lớp SomeClass, phương thức của lớp cơ sở sẽ được thực thi, và kết quả xuất ra sẽ có thay đổi như sau: The value of s is SomeClass

Như chúng ta thấy, hành vi mặc định đã trả về một chuỗi chính là tên của lớp đang thể hiện. Các lớp không cần phải khai báo tường minh việc dẫn xuất từ lớp Object, việc kế thừa sẽ được đưa vào một cách ngầm định. Như lớp SomeClass trên ta không khai báo bất cứ dẫn xuất của lớp nào nhưng C# sẽ tự động đưa lớp Object thành lớp dẫn xuất. Do đó ta mới có thể phủ quyết phương thức `ToString()` của lớp Object.

3.2.5. Các lớp lồng nhau

Các lớp chứa những thành viên, và những thành viên này có thể là một lớp khác có kiểu do người dùng định nghĩa (user-defined type). Do vậy, một lớp Button có thể có một thành viên của kiểu Location, và kiểu Location này chứa thành viên của kiểu dữ liệu Point. Cuối cùng, Point có thể chứa thành viên của kiểu int.

Cho đến lúc này, các lớp được tạo ra chỉ để dùng cho các lớp bên ngoài, và chức năng của các lớp đó như là lớp trợ giúp (helper class). Chúng ta có thể định nghĩa một lớp trợ giúp bên trong các lớp ngoài (outer class). Các lớp được định nghĩa bên trong gọi là các lớp lồng (nested class), và lớp chứa được gọi đơn giản là lớp ngoài.

Những lớp lồng bên trong có lợi là có khả năng truy cập đến tất cả các thành viên của lớp ngoài. Một phương thức của lớp lồng có thể truy cập đến biến thành viên private của lớp ngoài.

Hơn nữa, lớp lồng bên trong có thể ẩn đồi với tất cả các lớp khác, lớp lồng có thể là private cho lớp ngoài.

Cuối cùng, một lớp làm lồng bên trong là public và được truy cập bên trong phạm vi của lớp ngoài. Nếu một lớp Outer là lớp ngoài, và lớp Nested là lớp public lồng bên trong lớp Outer, chúng ta có thể tham chiếu đến lớp Nested như Outer.Nested, khi đó lớp bên ngoài hành động ít nhiều giống như một namespace hay một phạm vi.

□Ghi chú: Đối với người lập trình Java, lớp lồng nhau trong C# thì giống như những lớp nội static (static inner) trong Java. Không có sự tương ứng trong C# với những lớp nội nonstatic (nonstatic inner) trong Java.

Ví dụ 3.16 sau sẽ thêm một lớp lồng vào lớp Fraction tên là FractionArtist. Chức năng của lớp FractionArtist là vẽ một phân số ra màn hình. Trong ví dụ này, việc vẽ sẽ được thay thế bằng sử dụng hàm WriteLine xuất ra màn hình console.

□ Ví dụ 3.17: Sử dụng lớp lồng nhau.

```
using System;
using System.Text;
public class Fraction
{
    public Fraction( int numerator, int denominator )
    {
        this.numerator = numerator;
        this.denominator = denominator;
    }
    public override string ToString()
    {
        StringBuilder s = new StringBuilder();
        s.AppendFormat("{0}/{1}", numerator, denominator);
        return s.ToString();
    }
    internal class FractionArtist
    {
        public void Draw( Fraction f )
        {
            Console.WriteLine("Drawing the numerator {0}", f.numerator);
            Console.WriteLine("Drawing the denominator {0}",
f.denominator);
        }
    }
}
```

```
    }
}

// biến thành viên private
private int numerator; private int denominator;
public class Tester
{
    static void Main()
    {
        Fraction f1 = new Fraction( 3, 4); Console.WriteLine("f1:
{0}", f1.ToString());
        Fraction.FractionArtist fa = new Fraction.FractionArtist();
        fa.Draw( f1 );
    }
}
```

Lớp Fraction trên nói chung là không có gì thay đổi ngoại trừ việc thêm một lớp lồng bên trong và lược đi một số phương thức không thích hợp trong ví dụ này. Lớp lồng bên trong FractionArtist chỉ cung cấp một phương thức thành viên duy nhất, phương thức Draw(). Điều thú vị trong phương thức Draw() truy cập dữ liệu thành viên private là f.numerator và f.denominator. Hai biến thành viên private này sẽ không cho phép truy cập nếu FractionArtist không phải là lớp lồng bên trong của lớp Fraction.

Lưu ý là trong hàm Main() khi khai báo một thể hiện của lớp lồng bên trong, chúng ta phải xác nhận tên của lớp bên ngoài, tức là lớp Fraction:

```
Fraction.FractionArtist fa = new Fraction.FractionArtist();
```

Thậm chí khi lớp FractionArtist là public, thì phạm vi của lớp này vẫn nằm bên trong của lớp Fraction.

3.3. Nạp chồng toán tử

Hướng thiết kế của ngôn ngữ C# là tất cả các lớp do người dùng định nghĩa (user-defined classes) có tất cả các chức năng của các lớp được xây dựng sẵn. Ví dụ, giả sử chúng ta định nghĩa một lớp để thể hiện một phân số. Để đảm bảo rằng lớp này có tất cả các chức năng tương tự như các lớp được xây dựng sẵn, nghĩa là chúng ta cho phép thực hiện các phép toán số học trên các thể hiện của phân số chúng ta (như các phép toán cộng phân số, nhân hai phân số,...) và chuyển đổi qua lại giữa phân số và kiểu dữ liệu xây dựng sẵn như kiểu nguyên (int). Dĩ nhiên là chúng ta có thể dễ dàng thực hiện các toán tử bằng cách gọi một phương thức, tương tự như câu lệnh sau:

```
Fraction theSum = firstFraction.Add( secondFraction );
```

Mặc dù cách thực hiện này không sai nhưng về trực quan thì rất tệ không được tự nhiên như kiểu dữ liệu được xây dựng sẵn. Cách thực hiện sau sẽ tốt hơn rất nhiều nếu ta thiết kế được:

```
Fraction theSum = firstFraction + secondFraction;
```

Cách thực hiện này xem trực quan hơn và giống với cách thực hiện của các lớp được xây dựng sẵn, giống như khi thực hiện phép cộng giữa hai số nguyên int.

Trong chương này chúng ta sẽ tìm hiểu kỹ thuật thêm các toán tử chuẩn vào kiểu dữ liệu do người dùng định nghĩa. Và chúng ta sẽ tìm hiểu các toán tử chuyển đổi để chuyển đổi kiểu dữ liệu do người dùng định nghĩa một cách tường minh hay ngầm định sang các kiểu dữ liệu khác.

3.3.1. Sử dụng toán tử Operator

Trong ngôn ngữ C#, các toán tử là các phương thức tĩnh, giá trị trả về của nó thể hiện kết quả của một toán tử và những tham số là các toán hạng. Khi chúng ta tạo một toán tử cho một lớp là chúng ta đã thực hiện nạp chồng (overloaded) những toán tử đó, cũng giống như là chúng ta có thể nạp chồng bất cứ phương thức thành viên nào. Do đó, để nạp chồng toán tử cộng (+) chúng ta có thể viết như sau:

```
public static Fraction operator + ( Fraction lhs, Fraction rhs )
```

Trong toán tử trên ta có sự qui ước đặt tên của tham số là lhs và rhs. Tham số tên lhs thay thế cho “left hand side” tức là toán hạng bên trái, tương tự tham số tên rhs thay thế cho “right hand side” tức là toán hạng bên phải.

Cú pháp ngôn ngữ C# cho phép nạp chồng một toán tử bằng cách viết từ khóa operator và theo sau là toán tử được nạp chồng. Từ khóa operator là một bổ sung phương thức (method operator). Như vậy, để nạp chồng toán tử cộng (+) chúng ta có thể viết operator +. Khi chúng ta viết:

```
Fraction theSum = firstFraction + secondFraction;
```

Thì toán tử nạp chồng + được thực hiện, với firstFraction được truyền vào như là tham số đầu tiên, và secondFraction được truyền vào như là tham số thứ hai. Khi trình biên dịch gấp biểu thức:

```
firstFraction + secondFraction
```

thì trình biên dịch sẽ chuyển biểu thức vào:

```
Fraction.operator+(firstFraction, secondFraction)
```

Kết quả sau khi thực hiện là một đối tượng Fraction mới được trả về, trong trường hợp này phép gán sẽ được thực hiện để gán một đối tượng Fraction cho theSum.

□Ghi chú: Đối với người lập trình C++, trong ngôn ngữ C# không thể tạo được toán tử nonstatic, và do vậy nên toán tử nhị phân phải lấy hai toán hạng.

3.3.2. Hỗ trợ ngôn ngữ .Net khác

Ngôn ngữ C# cung cấp khả năng cho phép nạp chồng toán tử cho các lớp mà chúng ta xây dựng, thậm chí điều này không hoặc đề cập rất ít trong Common Language Specification (CLS). Những ngôn ngữ .NET khác như VB.NET thì không hỗ trợ việc nạp chồng toán tử, và một điều quan trọng để đảm bảo là lớp của chúng ta phải hỗ trợ các phương thức thay thế cho phép những ngôn ngữ khác có thể gọi để tạo ra các hiệu ứng tương tự.

Do đó, nếu chúng ta nạp chồng toán tử (+) thì chúng ta nên cung cấp một phương thức Add() cũng làm cùng chức năng là cộng hai đối tượng. Nạp chồng toán tử có thể là một cú pháp ngắn gọn, nhưng nó không chỉ là đường dẫn cho những đối tượng của chúng ta thiết lập một nhiệm vụ được đưa ra.

3.3.3. Sử dụng toán tử

Nạp chồng toán tử có thể làm cho mã nguồn của chúng ta trực quan và những hành động của lớp mà chúng ta xây dựng giống như các lớp được xây dựng sẵn. Tuy nhiên, việc nạp chồng toán tử cũng có thể làm cho mã nguồn phức tạp một cách khó quản lý nếu chúng ta phá vỡ cách thể hiện thông thường để sử dụng những toán tử. Hạn chế việc sử dụng tùy tiện các nạp chồng toán tử bằng những cách sử dụng mới và những cách đặc trưng.

Ví dụ, mặc dù chúng ta có thể hấp dẫn bởi việc sử dụng nạp chồng toán tử gia tăng (++) trong lớp Employee để gọi một phương thức gia tăng mức lương của nhân viên, điều này có thể đem lại rất nhiều nhầm lẫn cho các lớp client truy cập lớp Employee. Vì bên trong của lớp còn có thể có nhiều trường thuộc tính số khác, như số tuổi, năm làm việc,...ta không thể dành toán tử gia tăng duy nhất cho thuộc tính lương được. Cách tốt nhất là sử dụng nạp chồng toán tử một cách hạn chế, và chỉ sử dụng khi nào nghĩa nó rõ ràng và phù hợp với các toán tử của các lớp được xây dựng sẵn.

Khi thường thực hiện việc nạp chồng toán tử so sánh bằng (==) để kiểm tra hai đối tượng xem có bằng nhau hay không. Ngôn ngữ C# nhấn mạnh rằng nếu chúng ta thực hiện nạp chồng toán tử bằng, thì chúng ta phải nạp chồng toán tử nghịch với toán tử bằng là toán tử không bằng (!=). Tương tự, khi nạp chồng toán tử nhỏ hơn (<) thì cũng phải tạo toán tử (>) theo từng cặp. Cũng như toán tử (>=) đi tương ứng với toán tử (<=).

Theo sau là một số luật được áp dụng để thực hiện nạp chồng toán tử:

- Định nghĩa những toán tử trong kiểu dữ liệu giá trị, kiểu do ngôn ngữ xây dựng sẵn.
- Cung cấp những phương thức nạp chồng toán tử chỉ bên trong của lớp nơi mà những phương thức được định nghĩa.

- Sử dụng tên và những kí hiệu qui ước được mô tả trong Common Language Specification (CLS).

Sử dụng nạp chồng toán tử trong trường hợp kết quả trả về của toán tử là thật sự rõ ràng. Ví dụ, như thực hiện toán tử trừ (-) giữa một giá trị Time với một giá trị Time khác là một toán tử có ý nghĩa. Tuy nhiên, nếu chúng ta thực hiện toán tử or hay toán tử and giữa hai đối tượng Time thì kết quả hoàn toàn không có nghĩa gì hết.

Nạp chồng toán tử có tính chất đối xứng. Ví dụ, nếu chúng ta nạp chồng toán tử bằng (==) thì cũng phải nạp chồng toán tử không bằng (!=). Do đó khi thực hiện toán tử có tính chất đối xứng thì phải thực hiện toán tử đối xứng lại như: < với >, <= với >=.

Phải cung cấp các phương thức thay thế cho toán tử được nạp chồng. Đa số các ngôn ngữ điều không hỗ trợ nạp chồng toán tử. Vì nguyên do này nên chúng ta phải thực thi các phương thức thứ hai có cùng chức năng với các toán tử. Common Language Specification (CLS) đòi hỏi phải thực hiện phương thức thứ hai tương ứng.

Bảng 3.4 sau trình bày các toán tử cùng với biểu tượng của toán tử và các tên của phương thức thay thế các toán tử.

Biểu tượng	Tên phương thức thay thế	Tên toán tử
+	Add	Toán tử cộng
-	Subtract	Toán tử trừ
*	Multiply	Toán tử nhân
/	Divide	Toán tử chia
%	Mod	Toán tử chia lấy dư
^	Xor	Toán tử or loại trừ
&	BitwiseAnd	Toán tử and nhị phân
	BitwiseOr	Toán tử or nhị phân
&&	And	Toán tử and logic
	Or	Toán tử or logic
=	Assign	Toán tử gán
<<	LeftShift	Toán tử dịch trái
>>	RightShift	Toán tử dịch phải
==	Equals	Toán tử so sánh bằng
>	Compare	Toán tử so sánh lớn hơn

<	Compare	Toán tử so sánh nhỏ hơn
!=	Compare	Toán tử so sánh không bằng
>=	Compare	Toán tử so sánh lớn hơn hay bằng
<=	Compare	Toán tử so sánh nhỏ hơn hay bằng
*=	Multiply	Toán tử nhân rồi gán trở lại
-=	Subtract	Toán tử trừ rồi gán trở lại
^=	Xor	Toán tử or loại trừ rồi gán lại
<<=	LeftShift	Toán tử dịch trái rồi gán lại
%=	Mod	Toán tử chia dư rồi gán lại
+=	Add	Toán tử cộng rồi gán lại
&=	BitwiseAnd	Toán tử and rồi gán lại
=	BitwiseOr	Toán tử or rồi gán lại
/=	Divide	Toán tử chia rồi gán
--	Decrement	Toán tử giảm
++	Increment	Toán tử tăng
-	Negate	Toán tử phủ định một ngôi
+	Plus	Toán tử cộng một ngôi
~	OnesComplement	Toán tử bù

3.3.4. Toán tử so sánh bằng

Nếu chúng ta nạp chồng toán tử bằng (==), thì chúng ta cũng nên phủ quyết phương thức ảo Equals() được cung cấp bởi lớp object và chuyển lại cho toán tử bằng thực hiện. Điều này cho phép lớp của chúng ta thể tương thích với các ngôn ngữ .NET khác không hỗ trợ tính nạp chồng toán tử nhưng hỗ trợ nạp chồng phương thức. Những lớp FCL không sử dụng nạp chồng toán tử, nhưng vẫn mong đợi lớp của chúng ta thực hiện những phương thức cơ bản này. Do đó ví dụ lớp ArrayList mong muốn chúng ta thực thi phương thức Equals(). Lớp object thực thi phương thức Equals() với khai báo sau:

```
public override bool Equals( object o )
```

Bằng cách phủ quyết phương thức này, chúng ta cho phép lớp Fraction hành động một cách đa hình với tất cả những lớp khác. Bên trong thân của phương thức Equals() chúng ta cần phải đảm bảo rằng chúng ta đang so sánh với một Fraction khác, và nếu như

chúng ta đã thực thi một toán tử so sánh bằng thì có thể định nghĩa phương thức Equals() như sau:

```
public override bool Equals( object o )
{
    if ( !(o is Fraction) )
    {
        return false;
    }
    return this == (Fraction) o;
}
```

Toán tử is được sử dụng để kiểm tra kiểu của đối tượng lúc chạy chương trình có tương thích với toán hạng trong trường hợp này là Fraction. Do o là Fraction nên toán tử is sẽ trả về true.

3.3.5. Toán tử chuyển đổi

C# cho phép chuyển đổi từ kiểu int sang kiểu long một cách ngầm định, và cũng cho phép chúng ta chuyển từ kiểu long sang kiểu int một cách tường minh. Việc chuyển từ kiểu int sang kiểu long được thực hiện ngầm định bởi vì hiển nhiên bất kỳ giá trị nào của int cũng được thích hợp với kích thước của kiểu long. Tuy nhiên, điều ngược lại, tức là chuyển từ kiểu long sang kiểu int phải được thực hiện một cách tường minh (sử dụng ép kiểu) bởi vì ta có thể mất thông tin khi giá trị của biến kiểu long vượt quá kích thước của int lưu trong bộ nhớ:

```
int myInt = 5;

long myLong;

myLong = myInt; // ngầm định

myInt = (int) myLong; // tường minh
```

Chúng ta muốn thực hiện việc chuyển đổi này với lớp Fraction. Khi đưa ra một số nguyên, chúng ta có thể hỗ trợ ngầm định để chuyển đổi thành một phân số bởi vì bất kỳ giá trị nguyên nào ta cũng có thể chuyển thành giá trị phân số với mẫu số là 1 như ($24 == 24/1$). Khi đưa ra một phân số, chúng ta muốn cung cấp một sự chuyển đổi tường minh trở lại một số nguyên, điều này có thể hiểu là một số thông tin sẽ bị mất. Do đó, khi chúng ta chuyển phân số $9/4$ thành giá trị nguyên là 2.

Từ ngữ ngầm định (implicit) được sử dụng khi một chuyển đổi đảm thành công mà không mất bất cứ thông tin nào của dữ liệu nguyên thủy. Trường hợp ngược lại, tường minh (explicit) không đảm bảo toàn dữ liệu sau khi chuyển đổi do đó việc này sẽ được thực hiện một cách công khai.

Ví dụ 3.17 sẽ trình bày dưới đây minh họa cách thức mà chúng ta có thể thực thi chuyển đổi tường minh và ngầm định, và thực thi một vài các toán tử của lớp Fraction. Trong ví dụ này chúng ta sử dụng hàm Console.WriteLine() để xuất thông điệp ra màn hình minh họa khi phương thức được thi hành. Tuy nhiên cách tốt nhất là chúng ta sử dụng trình bebug để theo dõi từng bước thực thi các lệnh hay nhảy vào từng phương thức được gọi.

Ví dụ 3.18: Định nghĩa các chuyển đổi và toán tử cho lớp Fraction.

```
using System;
public class Fraction
{
    public Fraction(int numerator, int denominator)
    {
        Console.WriteLine("In Fraction Constructor( int, int )");
        this.numerator = numerator;
        this.denominator = denominator;
    }
    public Fraction(int wholeNumber)
    {
        Console.WriteLine("In Fraction Constructor( int )");
    }
}
```

```
    numerator = wholeNumber;
    denominator = 1;
}
public static implicit operator Fraction( int theInt )
{
    Console.WriteLine(" In implicit conversion to Fraction");
    return new Fraction( theInt );
}
public static explicit operator int( Fraction theFraction )
{
    Console.WriteLine("In explicit conversion to int");
    return theFraction.numerator / theFraction.denominator;
}
public static bool operator == ( Fraction lhs, Fraction rhs)
{
    Console.WriteLine("In operator ==");
    if (lhs.numerator == rhs.numerator &&lhs.denominator ==
rhs.denominator)
    {
        return true;
    }
    // thực hiện khi hai phân số không bằng nhau return false;
}
public static bool operator != ( Fraction lhs, Fraction rhs)
{
    Console.WriteLine("In operator !=");
    return !( lhs == rhs );
}
public override bool Equals( object o )
{
```

```
Console.WriteLine("In method Equals");

if ( !(o is Fraction ))
{
    return false;
}

return this == ( Fraction ) o;
}

public static Fraction operator+( Fraction lhs, Fraction rhs )
{
    Console.WriteLine("In operator +");

    if (lhs.denominator == rhs.denominator )
    {
        return new Fraction( lhs.numerator + rhs.numerator,
lhs.denominator);
    }

    //thực hiện khi hai mẫu số không bằng nhau
    int firstProduct = lhs.numerator * rhs.denominator;
    int secondProduct = rhs.numerator * lhs.denominator; return
new Fraction(
    firstProduct + secondProduct, lhs.denominator *
rhs.denominator);
}

public override string ToString()
{
    string s = numerator.ToString() + "/" +
denominator.ToString();

    return s;
}

//biến thành viên lưu tử số và mẫu số
private int numerator;
private int denominator;
```

```
}

public class Tester
{
    static void Main()
    {
        Fraction f1 = new Fraction( 3, 4);
        Console.WriteLine("f1:{0}",f1.ToString());
        Fraction f2 = new Fraction( 2, 4);
        Console.WriteLine("f2:{0}",f2.ToString());
        Fraction f3 = f1 + f2;
        Console.WriteLine("f1 + f2 = f3:{0}",f3.ToString());
        Fraction f4 = f3 + 5;
        Console.WriteLine("f4 = f3 + 5:{0}",f4.ToString());
        Fraction f5 = new Fraction( 2, 4);
        if( f5 == f2 )
        {
            Console.WriteLine("f5:{0}==f2:{1}", f5.ToString(),
f2.ToString());
        }
    }
}
```

Lớp Fraction bắt đầu với hai hàm khởi dựng: một hàm lấy một tử số và mẫu số, còn hàm kia lấy chỉ lấy một số làm tử số. Tiếp sau hai bộ khởi dựng là hai toán tử chuyển đổi. Toán tử chuyển đổi đầu tiên chuyển một số nguyên sang một phân số:

```
public static implicit operator Fraction( int theInt )
{
    return new Fraction( theInt );
}
```

Sự chuyển đổi này được thực hiện một cách ngầm định bởi vì bất cứ số nguyên nào cũng có thể được chuyển thành một phân số bằng cách thiết lập tử số bằng giá trị số nguyên và mẫu số có giá trị là 1. Việc thực hiện này có thể giao lại cho phương thức khởi dụng lấy một tham số. Toán tử chuyển đổi thứ hai được thực hiện một cách tường minh, chuyển từ một Fraction ra một số nguyên:

```
public static explicit operator int( Fraction theFraction )  
{  
    return theFraction.numerator / theFraction.denominator;  
}
```

Bởi vì trong ví dụ này sử dụng phép chia nguyên, phép chia này sẽ cắt bỏ phần phân chỉ lấy phần nguyên. Do vậy nếu phân số có giá trị là 16/15 thì kết quả số nguyên trả về là 1. Một số các phép chuyển đổi tốt hơn bằng cách sử dụng làm tròn số.

Tiếp theo sau là toán tử so sánh bằng (==) và toán tử so sánh không bằng (!=). Chúng ta nên nhớ rằng khi thực thi toán tử so sánh bằng thì cũng phải thực thi toán tử so sánh không bằng. Chúng ta đã định nghĩa giá trị bằng nhau giữa hai Fraction khi tử số bằng tử số và mẫu số bằng mẫu số. Ví dụ, như hai phân số 3/4 và 6/8 thì không được so sánh là bằng nhau. Một lần nữa, một sự thực thi tốt hơn là tối giản tử số và mẫu số khi đó 6/8 sẽ đơn giản thành 3/4 và khi đó so sánh hai phân số sẽ bằng nhau.

Trong lớp này chúng ta cũng thực thi phủ quyết phương thức Equals() của lớp object, do đó đối tượng Fraction của chúng ta có thể được đối xử một cách đa hình với bất cứ đối tượng khác. Trong phần thực thi của phương thức chúng ta ủy thác việc so sánh lại cho toán tử so sánh bằng cách gọi toán tử (==).

Lớp Fraction có thể thực thi hết tất cả các toán tử số học như cộng, trừ, nhân, chia. Tuy nhiên, trong phạm vi nhỏ hẹp của mình họa chúng ta chỉ thực thi toán tử cộng, và thậm chí phép cộng ở đây được thực hiện đơn giản nhất. Chúng ta thử nhìn lại, nếu hai mẫu số bằng nhau thì ta cộng tử số:

```
public static Fraction operator + ( Fraction lhs, Fraction rhs )
```

```
{  
    if ( lhs.denominator == rhs.denominator)  
    {  
        return new Fraction( lhs.numerator + rhs.numerator,  
                             lhs.denominator);  
    }  
}
```

Nếu mẫu số không cùng nhau, thì chúng ta thực hiện nhân chéo:

```
int firstProduct = lhs.numerator * rhs.denominator;  
  
int secondProduct = rhs.numerator * lhs.denominator;  
  
return new Fraction( firstProduct + secondProduct,  
                     lhs.denominator * rhs.denominator);
```

Cuối cùng là sự phủ quyết phương thức `ToString()` của lớp `object`, phương thức mới này thực hiện viết xuất ra nội dung của phân số dưới dạng : tử số / mẫu số:

```
public override string ToString()  
{  
    string s = numerator.ToString() + "/" + denominator.ToString();  
  
    return s;  
}
```

Chúng ta tạo một chuỗi mới bằng cách gọi phương thức `ToString()` của `numerator`. Do `numerator` là một đối tượng, nên trình biên dịch sẽ ngầm định thực hiện boxing số nguyên `numerator` và sau đó gọi phương thức `ToString()`, trả về một chuỗi thể hiện giá trị của số nguyên `numerator`. Sau đó ta nối chuỗi với “/” và cuối cùng là chuỗi thể hiện giá trị của mẫu số.

Với lớp `Fraction` đã tạo ra, chúng ta thực hiện kiểm tra lớp này. Đầu tiên chúng ta tạo ra hai phân số $3/4$, và $2/4$:

```
Fraction f1 = new Fraction( 3, 4);
Console.WriteLine("f1:{0}",f1.ToString());
Fraction f2 = new Fraction( 2, 4);
Console.WriteLine("f2:{0}",f2.ToString());
```

Kết quả thực hiện các lệnh trên như sau:

In Fraction Constructor(int, int) f1: 3/4

In Fraction Constructor(int, int)

f2: 2/4

Do trong phương thức khởi dựng của lớp Fraction chúng ta có gọi hàm WriteLine() để xuất ra thông tin bộ khởi dựng nên khi tạo đối tượng (new) thì cũng các thông tin này sẽ được hiển thị.

Dòng tiếp theo trong hàm Main() sẽ gọi toán tử cộng, đây là phương thức tĩnh. Mục đích của toán tử này là cộng hai phân số và trả về một phân số mới là tổng của hai phân số đưa vào:

```
Fraction f3 = f1 + f2;
Console.WriteLine("f1 + f2 = f3: {0}", f3.ToString());
```

Hai câu lệnh trên sẽ cho ra kết quả như sau:

```
In operator +
In Fraction Constructor( int, int)
f1 + f2 = f3: 5/4
```

Toán tử + được gọi trước sau đó đến phương thức khởi dựng của đối tượng f3. Phương thức khởi dựng này lấy hai tham số nguyên để tạo tử số và mẫu số của phân số mới f3.

Hai câu lệnh tiếp theo cộng một giá trị nguyên vào phân số f3 và gán kết quả mới về cho phân số mới f4:

Fraction f4 = f3 + 5;

```
Console.WriteLine("f3 + 5 = f4: {0}", f4.ToString());
```

Kết quả được trình bày theo thứ tự sau:

In implicit conversion to Fraction In Fraction Construction(int)

In operator+

In Fraction Constructor(int, int)

f3 + 5 = f4: 25/4

Ghi chú: rằng toán tử chuyển đổi ngầm định được gọi khi chuyển 5 thành một phân số. Phân số được tạo ra từ toán tử chuyển đổi ngầm định này gọi phương thức khởi dựng một tham số để tạo phân số mới 5/1. Phân số mới này sẽ được chuyển thành toán hạng trong phép cộng với phân số f3 và kết quả trả về là phân số f4 là tổng của hai phân số trên.

Thử nghiệm cuối cùng là tạo một phân số mới f5, rồi sau đó gọi toán tử nạp chồng so sánh bằng để kiểm tra xem hai phân số có bằng nhau hay không.

3.4. Cấu trúc

Cấu trúc là kiểu dữ liệu đơn giản do người dùng định nghĩa, kích thước nhỏ dùng để thay thế cho lớp. Những cấu trúc thì tương tự như lớp cũng chứa các phương thức, những thuộc tính, các trường, các toán tử, các kiểu dữ liệu lồng bên trong và bộ chỉ mục (indexer).

Có một số sự khác nhau quan trọng giữa những lớp và cấu trúc. Ví dụ, cấu trúc thì không hỗ trợ kế thừa và bộ hủy giống như kiểu lớp. Một điều quan trọng nhất là trong khi lớp là kiểu dữ liệu tham chiếu, thì cấu trúc là kiểu dữ liệu giá trị (Chương 3 đã thảo luận về kiểu dữ liệu tham chiếu và kiểu dữ liệu giá trị). Do đó cấu trúc thường dùng để hiển các đối tượng không đòi hỏi một ngữ nghĩa tham chiếu, hay một lớp nhỏ mà khi đặt vào trong stack thì có lợi hơn là đặt trong bộ nhớ heap.

Một sự nhận xét được rút ra là chúng ta chỉ nên sử dụng những cấu trúc chỉ với những kiểu dữ liệu nhỏ, và những hành vi hay thuộc tính của nó giống như các kiểu dữ liệu được xây dựng sẵn.

Cấu trúc có hiệu quả khi chúng ta sử dụng chúng trong mảng bộ nhớ (Chương 2). Tuy nhiên, cấu trúc sẽ kém hiệu quả khi chúng ta sử dụng dạng tập hợp (collections). Tập hợp được xây dựng hướng tới các kiểu dữ liệu tham chiếu.

Trong chương này chúng ta sẽ tìm hiểu các định nghĩa và làm việc với kiểu cấu trúc và cách sử dụng bộ khởi động để khởi tạo những giá trị của cấu trúc.

3.4.1. Định nghĩa một cấu trúc

Cú pháp để khai báo một cấu trúc cũng tương tự như cách khai báo một lớp:

```
[thuộc tính] [bổ sung truy cập] struct <tên cấu trúc> [: danh sách giao diện]
{
    [thành viên của cấu trúc]
}
```

Ví dụ 3.19 sau minh họa cách tạo một cấu trúc. Kiểu Location thể hiện một điểm trong không gian hai chiều. Lưu ý rằng cấu trúc Location này được khai báo chính xác như khi thực hiện khai báo với một lớp, ngoại trừ việc sử dụng từ khóa struct. Ngoài ra cũng lưu ý rằng hàm khởi động của Location lấy hai số nguyên và gán những giá trị của chúng cho các biến thành viên, x và y. Tọa độ x và y của Location được khai báo như là thuộc tính.

- Ví dụ 3.19 Tạo một cấu trúc.

```
using System;

public struct Location
{
    public Location( int xCoordinate, int yCoordinate )
    {
```

```
    xVal = xCoordinate;  
  
    yVal = yCoordinate;  
  
}  
  
public int x  
  
{  
  
    get {return xVal;}  
  
    set {xVal = value;}  
  
}  
  
public int y  
  
{  
  
    get {return yVal;}  
  
    set { yVal = value;}  
  
}  
  
public override string ToString()  
  
{  
  
    return (String.Format("{0}, {1}", xVal, yVal));  
  
}  
  
// thuộc tính private lưu toạ độ x, y  
  
private int xVal;  
  
private int yVal;  
  
}  
  
public class Tester  
  
{  
  
    public void myFunc( Location loc)  
  
    {
```

```
    loc.x = 50;  
  
    loc.y = 100;  
  
    Console.WriteLine("Loc1 location: {0}", loc);  
  
}  
  
static void Main()  
  
{  
  
    Location loc1 = new Location( 200, 300);  
Console.WriteLine("Loc1 location: {0}", loc1); Tester t = new  
Tester();  
  
    t.myFunc( loc1 );  
  
    Console.WriteLine("Loc1 location: {0}", loc1);  
  
}
```

Không giống như những lớp, cấu trúc không hỗ trợ việc thừa kế. Chúng được thừa kế ngầm định từ lớp object (tương tự như tất cả các kiểu dữ liệu trong C#, bao gồm các kiểu dữ liệu xây dựng sẵn) nhưng không thể kế thừa từ các lớp khác hay cấu trúc khác. Cấu trúc cũng được ngầm định là sealed, điều này có ý nghĩa là không có lớp nào hay bất cứ cấu trúc nào có thể dẫn xuất từ nó. Tuy nhiên, cũng giống như các lớp, cấu trúc có thể thực thi nhiều giao diện. Sau đây là một số sự khác nhau nữa là:

Không có bộ hủy và bộ khởi tạo mặc định tùy chọn: Những cấu trúc không có bộ hủy và cũng không có bộ khởi tạo mặc định không tham số tùy chọn. Nếu chúng ta không cung cấp bất cứ bộ khởi tạo nào thì cấu trúc sẽ được cung cấp một bộ khởi tạo mặc định, khi đó giá trị 0 sẽ được thiết lập cho tất cả các dữ liệu thành viên hay những giá trị mặc định tương ứng cho từng kiểu dữ liệu (bảng 3.2). Nếu chúng ta cung cấp bất cứ bộ khởi dụng nào thì chúng ta phải khởi tạo tất cả các trường trong cấu trúc.

Không cho phép khởi tạo: chúng ta không thể khởi tạo các trường thể hiện (instance fields) trong cấu trúc, do đó đoạn mã nguồn sau sẽ không hợp lệ:

```
private int xVal = 20;
```

```
private int yVal = 50;
```

mặc dù điều này thực hiện tốt đối với lớp.

Cấu trúc được thiết kế hướng tới đơn giản và gọn nhẹ. Trong khi các dữ liệu thành viên private hỗ trợ việc che dấu dữ liệu và sự đóng gói. Một vài người lập trình có cảm giác rằng điều này phá hỏng cấu trúc. Họ tạo một dữ liệu thành viên public, do vậy đơn giản thực thi một cấu trúc. Những người lập trình khác có cảm giác rằng những thuộc tính cung cấp một giao diện rõ ràng, đơn giản và việc thực hiện lập trình tốt đòi hỏi phải che dấu dữ liệu thậm chí với dữ liệu rất đơn giản. Chúng ta sẽ chọn cách nào, nói chung là phụ thuộc vào quan niệm thiết kế của từng người lập trình. Dù chọn cách nào thì ngôn ngữ C# cũng hỗ trợ cả hai cách tiếp cận.

3.4.2. Tạo cấu trúc

Chúng ta tạo một thể hiện của cấu trúc bằng cách sử dụng từ khóa new trong câu lệnh gán, như khi chúng ta tạo một đối tượng của lớp. Như trong ví dụ 3.18, lớp Tester tạo một thể hiện của Location như sau:

```
Location loc1 = new Location( 200, 300);
```

Ở đây một thể hiện mới tên là loc1 và nó được truyền hai giá trị là 200 và 300.

3.4.2.1. Cấu trúc là một kiểu giá trị

Phân định nghĩa của lớp Tester trong ví dụ 7.1 trên bao gồm một đối tượng Location là loc1 được tạo với giá trị là 200 và 300. Dòng lệnh sau sẽ gọi thực hiện bộ khởi tạo của cấu trúc Location:

```
Location loc1 = new Location( 200, 300);
```

Sau đó phương thức WriteLine() được gọi:

```
Console.WriteLine("Loc1 location: {0}", loc1);
```

Đĩ nhiên là WriteLine chờ đợi một đối tượng, nhưng Location là một cấu trúc (một kiểu giá trị). Trình biên dịch sẽ tự động boxing cấu trúc (cũng giống như trình biên dịch

đã làm với các kiểu dữ liệu giá trị khác). Một đối tượng sau khi boxing được truyền vào cho phương thức WriteLine(). Tiếp sau đó là phương thức ToString() được gọi trên đối tượng boxing này, do cấu trúc ngầm định kế thừa từ lớp object, và nó cũng có thể đáp ứng sự đa hình, bằng cách phủ quyết các phương thức như bất cứ đối tượng nào khác.

Loc1 location 200, 300

Tuy nhiên do cấu trúc là kiểu giá trị, nên khi truyền vào trong một hàm, thì chúng chỉ truyền giá trị vào hàm. Cũng như ta thấy ở dòng lệnh kế tiếp, khi đó một đối tượng Location được truyền vào phương thức myFunc():

```
t.myFunc( loc1 );
```

Trong phương thức myFunc() hai giá trị mới được gán cho x và y, sau đó giá trị mới sẽ được xuất ra màn hình:

Loc1 location: 50, 100

Khi phương thức myFunc() trả về cho hàm gọi (Main()) và chúng ta gọi tiếp phương thức WriteLine() một lần nữa thì giá trị không thay đổi:

Loc1 location: 200, 300

Như vậy cấu trúc được truyền vào hàm như một đối tượng giá trị, và một bản sao sẽ được tạo bên trong phương thức myFunc(). Nếu chúng ta thử đổi khai báo của Location là class như sau:

```
public class Location
```

Sau đó chạy lại chương trình thì có kết quả:

Loc1 location: 200, 3000

In myFunc loc: 50, 100

Loc1 location: 50, 100

Lúc này Location là một đối tượng tham chiếu nên khi truyền vào phương thức myFunc() thì việc gán giá trị mới cho x và y điều làm thay đổi đối tượng Location.

3.4.2.2. Gọi bộ khởi dựng mặc định

Như đề cập ở phần trước, nếu chúng ta không tạo bộ khởi dựng thì một bộ khởi dựng mặc định ngầm định sẽ được trình biên dịch tạo ra. Chúng ta có thể nhìn thấy điều này nếu bỏ bộ khởi dựng tạo ra:

```
/*
public Location( int xCoordinate , int yCoordinate)
{
    xVal = xCoordinate;
    yVal = yCoordinate;
}
*/
```

và ta thay dòng lệnh đầu tiên trong hàm Main() tạo Location có hai tham số bằng câu lệnh tạo không tham số như sau:

```
Location loc1 = new Location( 200, 300)

Location loc1 = new Location();
```

Bởi vì lúc này không có phương thức khởi dựng nào khai báo, một phương thức khởi dựng ngầm định sẽ được gọi. Kết quả khi thực hiện giống như sau:

```
Loc1 location 0, 0
In myFunc loc: 50, 100
Loc1 location: 0, 0
```

Bộ khởi tạo mặc định đã thiết lập tất cả các biến thành viên với giá trị 0.

□Ghi chú: Đối với lập trình viên C++ lưu ý, trong ngôn ngữ C#, từ khóa new không phải luôn luôn tạo đối tượng trên bộ nhớ heap. Các lớp thì được tạo ra trên heap, trong khi các cấu trúc thì được tạo trên stack. Ngoài ra, khi new được bỏ qua (sẽ bàn tiếp

trong phần sau), thì bộ khởi dựng sẽ không được gọi. Do ngôn ngữ C# yêu cầu phải có phép gán trước khi sử dụng, chúng ta phải khởi tạo tường minh tất cả các biến thành viên trước khi sử dụng chúng trong cấu trúc.

3.4.2.3. Tạo cấu trúc không gọi new

Bởi vì Location là một cấu trúc không phải là lớp, do đó các thể hiện của nó sẽ được tạo trong stack. Trong ví dụ 3.18 khi toán tử new được gọi:

```
Location loc1 = new Location( 200, 300);
```

kết quả một đối tượng Location được tạo trên stack.

Tuy nhiên, toán tử new gọi bộ khởi dựng của lớp Location, không giống như với một lớp, cấu trúc có thể được tạo ra mà không cần phải gọi toán tử new. Điều này giống như các biến của các kiểu dữ liệu được xây dựng sẵn (như int, long, char,...) được tạo ra. Ví dụ 3.19 sau minh họa việc tạo một cấu trúc không sử dụng toán tử new.

□Ghi chú: Đây là một sự khuyến cáo, trong ví dụ sau chúng ta minh họa cách tạo một cấu trúc mà không phải sử dụng toán tử new bởi vì có sự khác nhau giữa C# và ngôn ngữ C++ và sự khác nhau này chính là cách ngôn ngữ C# đối xử với những lớp khác những cấu trúc. Tuy nhiên, việc tạo một cấu trúc mà không dùng từ khóa new sẽ không có lợi và có thể tạo một chương trình khó hiểu, tiềm ẩn nhiều lỗi, và khó duy trì. Chương trình họ sau sẽ không được khuyến khích.

□ Ví dụ 3.20: Tạo một cấu trúc mà không sử dụng new.

```
using System;

public struct Location
{
    public Location( int xCoordinate, int yCoordinate )
    {
        xVal = xCoordinate;
        yVal = yCoordinate;
    }
}
```

```
}

public int x

{
    get { return xVal; }

    set { xVal = value; }

}

public int y

{
    get { return yVal; }

    set{ yVal = value; }

}

public override string ToString()

{
    return (string.Format("{0} ,{1}", xVal, yVal));
}

// biến thành viên lưu tọa độ x, y

public int xVal;

public int yVal;

}

public class Tester

{
    static void Main()

{
    Location loc1; loc1.xVal = 100; loc1.yVal = 250;

    Console.WriteLine("loc1");
}
```

```
    }  
}
```

Trong ví dụ 3.20 chúng ta khởi tạo biến thành viên một cách trực tiếp, trước khi gọi bất cứ phương thức nào của loc1 và trước khi truyền đối tượng cho phương thức WriteLine():

```
loc1.xVal = 100;
```

```
loc2.yVal = 250;
```

Nếu chúng ta thử bỏ một lệnh gán và biên dịch lại:

```
static void Main()
```

```
{
```

```
    Location loc1;
```

```
    loc1.xVal = 100;
```

```
//loc1.yVal = 250; Console.WriteLine( loc1 );
```

```
}
```

Chúng ta sẽ nhận một lỗi biên dịch như sau:

Use of unassigned local variable ‘loc1’

Một khi mà chúng ta đã gán tất cả các giá trị của cấu trúc, chúng ta có thể truy cập giá trị thông qua thuộc tính x và thuộc tính y:

```
static void Main()
```

```
{
```

```
    Location loc1;
```

```
// gán cho biến thành viên loc1.xVal = 100;
```

```
    loc1.yVal = 250;
```

```
// sử dụng thuộc tính loc1.x = 300;
```

```
    loc1.y = 400; Console.WriteLine( loc1 );
```

}

Hãy cẩn thận với việc sử dụng các thuộc tính. Mặc dù cấu trúc cho phép chúng ta hỗ trợ đóng gói bằng việc thiết lập thuộc tính private cho các biến thành viên. Tuy nhiên bản thân thuộc tính thật sự là phương thức thành viên, và chúng ta không thể gọi bất cứ phương thức thành viên nào cho đến khi chúng ta khởi tạo tất cả các biến thành viên.

Như ví dụ trên ta thiết lập thuộc tính truy cập của hai biến thành viên xVal và yVal là public vì chúng ta phải khởi tạo giá trị của hai biến thành viên này bên ngoài của cấu trúc, trước khi các thuộc tính được sử dụng.

3.5. Thực thi giao diện

Giao diện là ràng buộc, giao ước đảm bảo cho các lớp hay các cấu trúc sẽ thực hiện một điều gì đó. Khi một lớp thực thi một giao diện, thì lớp này báo cho các thành phần client biết rằng lớp này có hỗ trợ các phương thức, thuộc tính, sự kiện và các chỉ mục khai báo trong giao diện.

Một giao diện đưa ra một sự thay thế cho các lớp trừu tượng để tạo ra các sự ràng buộc giữa những lớp và các thành phần client của nó. Những ràng buộc này được khai báo bằng cách sử dụng từ khóa interface, từ khóa này khai báo một kiểu dữ liệu tham chiếu để đóng gói các ràng buộc.

Một giao diện thì giống như một lớp chỉ chứa các phương thức trừu tượng. Một lớp trừu tượng được dùng làm lớp cơ sở cho một họ các lớp dẫn xuất từ nó. Trong khi giao diện là sự trộn lẫn với các cây kế thừa khác.

Khi một lớp thực thi một giao diện, lớp này phải thực thi tất cả các phương thức của giao diện. Đây là một bắt buộc mà các lớp phải thực hiện.

Trong chương này chúng ta sẽ thảo luận cách tạo, thực thi và sử dụng các giao diện. Ngoài ra chúng ta cũng sẽ bàn tới cách thực thi nhiều giao diện cùng với cách kết hợp và mở rộng giao diện. Và cuối cùng là các minh họa dùng để kiểm tra khi một lớp thực thi một giao diện.

3.5.1. Thực thi một giao diện

Cú pháp để định nghĩa một giao diện như sau:

[thuộc tính] [bổ sung truy cập] interface <tên giao diện> [: danh sách cơ sở]

{

<phần thân giao diện>

}

Phần thuộc tính chúng ta sẽ đề cập sau. Thành phần bổ sung truy cập bao gồm: public, private, protected, internal, và protected internal đã được nói đến trong Chương 4, ý nghĩa tương tự như các bổ sung truy cập của lớp.

Theo sau từ khóa interface là tên của giao diện. Thông thường tên của giao diện được bắt đầu với từ I hoa (điều này không bắt buộc nhưng việc đặt tên như vậy rất rõ ràng và dễ hiểu, tránh nhầm lẫn với các thành phần khác). Ví dụ một số giao diện có tên như sau: IStorable, ICloneable, ...

Danh sách cơ sở là danh sách các giao diện mà giao diện này mở rộng, phần này sẽ được trình bày trong phần thực thi nhiều giao diện của chương. Phần thân của giao diện chính là phần thực thi giao diện sẽ được trình bày bên dưới.

Giả sử chúng ta muốn tạo một giao diện nhằm mô tả những phương thức và thuộc tính của một lớp cần thiết để lưu trữ và truy cập từ một cơ sở dữ liệu hay các thành phần lưu trữ dữ liệu khác như là một tập tin. Chúng ta quyết định gọi giao diện này là IStorage.

Trong giao diện này chúng ta xác nhận hai phương thức: Read() và Write(), khai báo này sẽ được xuất hiện trong phần thân của giao diện như sau:

interface IStorable

{

 void Read();

 void Write(object);

}

Mục đích của một giao diện là để định nghĩa những khả năng mà chúng ta muốn có trong một lớp. Ví dụ, chúng ta có thể tạo một lớp tên là Document, lớp này lưu trữ các dữ liệu trong cơ sở dữ liệu, do đó chúng ta quyết định lớp này này thực thi giao diện IStorable.

Để làm được điều này, chúng ta sử dụng cú pháp giống như việc tạo một lớp mới Document được thừa kế từ IStorable bằng dùng dấu hai chấm (:) và sau là tên giao diện

```
public class Document : IStorable
{
    public void Read()
    {
        .....
    }

    public void Write()
    {
        ....
    }
}
```

Bây giờ trách nhiệm của chúng ta, với vai trò là người xây dựng lớp Document phải cung cấp một thực thi có ý nghĩa thực sự cho những phương thức của giao diện IStorable. Chúng ta phải thực thi tất cả các phương thức của giao diện, nếu không trình biên dịch sẽ báo một lỗi. Sau đây là đoạn chương trình minh họa việc xây dựng lớp Document thực thi giao diện IStorable.

- Ví dụ 3.21: Sử dụng một giao diện.

```
using System;
// khai báo giao diện interface IStorable
```

```
{  
    // giao diện không khai báo bổ sung truy cập  
    // phương thức là public và không thực thi  
    void Read();  
  
    void Write(object obj);  
  
    int Status  
    {  
        get;  
        set;  
    }  
}  
  
// tạo một lớp thực thi giao diện IStorable  
public class Document : IStorable  
{  
    public Document( string s)  
    {  
        Console.WriteLine("Creating document with: {0}", s);  
    }  
  
    // thực thi phương thức Read()  
    public void Read()  
    {  
        Console.WriteLine("Implement the Read Method for IStorable");  
    }  
  
    // thực thi phương thức Write  
    public void Write( object o)
```

```
{  
    Console.WriteLine("Implementing the Write Method for IStorable");  
}  
  
// thực thi thuộc tính  
  
public int Status  
{  
    get { return status; }  
    set {status = value;}  
}  
  
// lưu trữ giá trị thuộc tính  
  
private int status = 0;  
}  
  
public class Tester  
{  
    static void Main()  
    {  
        // truy cập phương thức trong đối tượng Document  
  
        Document doc = new Document("Test Document"); doc.Status = -1;  
        doc.Read();  
  
        Console.WriteLine("Document Status: {0}", doc.Status);  
  
        // gán cho một giao diện và sử dụng giao diện IStorable  
        isDoc = (IStorable) doc; isDoc.Status = 0;  
        isDoc.Read();  
  
        Console.WriteLine("IStorable Status: {0}", isDoc.Status);  
    }  
}
```

```
}
```

Kết quả:

Creating document with: Test Document Implementing the Read Method for IStorable Document Status: -1

Implementing the Read Method for IStorable

IStorable Status: 0

Ví dụ 3.21 định nghĩa một giao diện IStorable với hai phương thức Read(), Write() và một thuộc tính tên là Status có kiểu là số nguyên.. Lưu ý rằng trong phần khai báo thuộc tính không có phần thực thi cho get() và set() mà chỉ đơn giản là khai báo có hành vi là get() và set():

```
int Status { get; set; }
```

Ngoài ra phần định nghĩa các phương thức của giao diện không có phần bổ sung truy cập (ví dụ như: public, protected, internal, private). Việc cung cấp các bổ sung truy cập sẽ tạo ra một lỗi. Những phương thức của giao diện được ngầm định là public vì giao diện là những ràng buộc được sử dụng bởi những lớp khác. Chúng ta không thể tạo một thể hiện của giao diện, thay vào đó chúng ta sẽ tạo thể hiện của lớp có thực thi giao diện.

Một lớp thực thi giao diện phải đáp ứng đầy đủ và chính xác các ràng buộc đã khai báo trong giao diện. Lớp Document phải cung cấp cả hai phương thức Read() và Write() cùng với thuộc tính Status. Tuy nhiên cách thực hiện những yêu cầu này hoàn toàn phụ thuộc vào lớp Document. Mặc dù IStorage chỉ ra rằng lớp Document phải có một thuộc tính là Status nhưng nó không biết hay cũng không quan tâm đến việc lớp Document lưu trữ trạng thái thật sự của các biến thành viên, hay việc tìm kiếm trong cơ sở dữ liệu. Những chi tiết này phụ thuộc vào phần thực thi của lớp.

3.5.1.1. Thực thi nhiều giao diện

Trong ngôn ngữ C# cho phép chúng ta thực thi nhiều hơn một giao diện. Ví dụ, nếu lớp Document có thể được lưu trữ và dữ liệu cũng được nén. Chúng ta có thể chọn thực thi cả hai giao diện IStorable và ICompressible. Như vậy chúng ta phải

thay đổi phần khai báo trong danh sách cơ sở để chỉ ra rằng cả hai giao diện đều được thực thi, sử dụng dấu phẩy (,) để phân cách giữa hai giao diện:

```
public class Document : IStorable, ICompressible
```

Do đó Document cũng phải thực thi những phương thức được xác nhận trong giao diện ICompressible:

```
public void Compress()
{
    Console.WriteLine("Implementing the Compress Method");
}

public void Decompress()
{
    Console.WriteLine("Implementing the Decompress Method");
}
```

Bổ sung thêm phần khai báo giao diện ICompressible và định nghĩa các phương thức của giao diện bên trong lớp Document.

3.5.1.2. Mở rộng giao diện

C# cung cấp chức năng cho chúng ta mở rộng một giao diện đã có bằng cách thêm các phương thức và các thành viên hay bổ sung cách làm việc cho các thành viên. Ví dụ, chúng ta có thể mở rộng giao diện ICompressible với một giao diện mới là ILoggedCompressible. Giao diện mới này mở rộng giao diện cũ bằng cách thêm phương thức ghi log các dữ liệu đã lưu:

```
interface ILoggedCompressible : ICompressible
{
    void LogSavedBytes();
}
```

Các lớp khác có thể thực thi tự do giao diện ICompressible hay ILoggedCompressible tùy thuộc vào mục đích có cần thêm chức năng hay không. Nếu một lớp thực thi giao diện ILoggedCompressible, thì lớp này phải thực thi tất cả các phương thức của cả hai giao diện ICompressible và giao diện ILoggedCompressible. Những đối tượng của lớp thực thi giao diện ILoggedCompressible có thể được gán cho cả hai giao diện ILoggedCompressible và ICompressible.

3.5.1.3. Kết hợp các giao diện

Một cách tương tự, chúng ta có thể tạo giao diện mới bằng cách kết hợp các giao diện cũ và ta có thể thêm các phương thức hay các thuộc tính cho giao diện mới. Ví dụ, chúng ta quyết định tạo một giao diện IStorableCompressible. Giao diện mới này sẽ kết hợp những phương thức của cả hai giao diện và cũng thêm vào một phương thức mới để lưu trữ kích thước nguyên thuỷ của các dữ liệu trước khi nén:

```
interface IStorableCompressible : IStoreable, ILoggedCompressible
{
    void LogOriginalSize();
}
```

- Ví dụ 3.22: Minh họa việc mở rộng và kết hợp các giao diện.

```
using System;

interface IStorable
{
    void Read();
    void Write(object obj);
    int Status { get; set; }
}

// giao diện mới
```

```
interface ICompressible
{
    void Compress();
    void Decompress();
}

// mở rộng giao diện

interface ILoggedCompressible : ICompressible
{
    void LogSavedBytes();
}

// kết hợp giao diện

interface IStorableCompressible : IStorable, ILoggedCompressible
{
    void LogOriginalSize();
}

interface IEncryptable
{
    void Encrypt();
    void Decrypt();
}

public class Document : IStorableCompressible, IEncryptable
{
    // bộ khởi tạo lớp Document lấy một tham số public Document(
    string s)

        Console.WriteLine("Creating document with: {0}", s);
```

```
// thực thi giao diện IStorable public void Read()
    Console.WriteLine("Implementing the Read Method for
IStorable");

    public void Write( object o)

    {
        Console.WriteLine("Implementing the Write Method for IStorable");
    }

    public int Status

    {
        get {return status;}
        set {status = value;}
    }

// thực thi ICompressible

    public void Compress()

    {
        Console.WriteLine("Implementing Compress");
    }

    public void Decompress()

    {
        Console.WriteLine("Implementing Decompress");
    }

// thực thi giao diện ILoggedCompressible

    public void LogSavedBytes()

    {
        Console.WriteLine("Implementing LogSavedBytes");
    }
```

```
}

// thực thi giao diện IStorableCompressible

public void LogOriginalSize()

{

    Console.WriteLine("Implementing LogOriginalSize");

}

// thực thi giao diện

public void Encrypt()

{

    Console.WriteLine("Implementing Encrypt");

}

public void Decrypt()

{

    Console.WriteLine("Implementing Decrypt");

}

// biến thành viên lưu dữ liệu cho thuộc tính

private int status = 0;

}

public class Tester

{

    public static void Main()

    {

        // tạo đối tượng document

        Document doc = new Document("Test Document");

        // gán đối tượng cho giao diện
```

```
IStorable isDoc = doc as IStorable;  
  
if ( isDoc != null)  
{  
    isDoc.Read();  
}  
  
else  
{  
    Console.WriteLine("IStorable not supported");  
}  
  
ICompressible icDoc = doc as ICompressible;  
  
if ( icDoc != null )  
{  
    icDoc.Compress();  
}  
  
else  
{  
    Console.WriteLine("Compressible not supported");  
}  
  
ILoggedCompressible ilcDoc = doc as ILoggedCompressible;  
  
if ( ilcDoc != null )  
{  
    ilcDoc.LogSavedBytes();  
  
    ilcDoc.Compress();  
  
    // ilcDoc.Read(); // không thể gọi được  
}
```

```
        else
    {
        Console.WriteLine("LoggedCompressible not supported");
    }

    IStorableCompressible isc = doc as IStorableCompressible;
    if ( isc != null )
    {
        isc.LogOriginalSize();
    }
    else
    {
        Console.WriteLine("StorableCompressible not
supported");
    }

    IEncryptable ie = doc as IEncryptable;
    if ( ie != null )
    {
        ie.Encrypt();
    }
    else
    {
        Console.WriteLine("Encryptable not supported");
    }
}
```

Kết quả:

Creating document with: Test Document Implementing the Read Method for IStorable Implementing Compress

Implementing LogSavedBytes Implementing Compress Implementing LogOriginalSize Implementing LogSaveBytes Implementing Compress

Implementing the Read Method for IStorable

Implementing Encrypt

Ví dụ 3.22 bắt đầu bằng việc thực thi giao diện IStorable và giao diện ICompressible. Sau đó là phần mở rộng đến giao diện ILoggedCompressible rồi sau đó kết hợp cả hai vào giao diện IStorableCompressible. Và giao diện cuối cùng trong ví dụ là IEncrypt.

Chương trình Tester tạo đối tượng Document mới và sau đó gán lần lượt vào các giao diện khác nhau. Khi một đối tượng được gán cho giao diện ILoggedCompressible, chúng ta có thể dùng giao diện này để gọi các phương thức của giao diện ICompressible bởi vì ILogged- Compressible mở rộng và thừa kế các phương thức từ giao diện cơ sở:

```
ILoggedCompressible ilcDoc = doc as ILoggedCompressible;  
if ( ilcDoc != null )  
{  
    ilcDoc.LogSavedBytes();  
    ilcDoc.Compress();  
    // ilcDoc.Read(); // không thể gọi được  
}
```

Tuy nhiên, ở đây chúng ta không thể gọi phương thức Read() bởi vì phương thức này của giao diện IStorable, không liên quan đến giao diện này. Nếu chúng ta thêm lệnh này vào thì chương trình sẽ biên dịch lỗi.

Nếu chúng ta gán vào giao diện IStorableCompressible, do giao diện này kết hợp hai giao diện IStorable và giao diện ICompressible, chúng ta có thể gọi tất cả những phương thức của IStorableCompressible, ICompressible, và IStorable:

```
IStorableCompressible isc = doc as IStorableCompressible;

if ( isc != null )

{
    isc.LogOriginalSize();      // IStorableCompressible
    isc.LogSaveBytes();        // ILoggedCompressible isc.Compress();
    // ICompress

    isc.Read();    // IStorable
}
```

3.5.2. Truy cập phương thức giao diện

Chúng ta có thể truy cập những thành viên của giao diện IStorable như thế là các thành viên của lớp Document:

```
Document doc = new Document("Test Document");

doc.status = -1;

doc.Read();
```

hay là ta có thể tạo thể hiện của giao diện bằng cách gán đối tượng Document cho một kiểu dữ liệu giao diện, và sau đó sử dụng giao diện này để truy cập các phương thức:

```
IStorable isDoc = (IStorable) doc;

isDoc.status = 0;

isDoc.Read();
```

□Ghi chú: Cũng như đã nói trước đây, chúng ta không thể tạo thể hiện của giao diện một cách trực tiếp. Do đó chúng ta không thể thực hiện như sau:

```
IStorable isDoc = new IStorable();
```

Tuy nhiên chúng ta có thể tạo thể hiện của lớp thực thi như sau:

```
Document doc = new Document("Test Document");
```

Sau đó chúng ta có thể tạo thể hiện của giao diện bằng cách gán đối tượng thực thi đến kiểu dữ liệu giao diện, trong trường hợp này là IStorable:

```
IStorable isDoc = (IStorable) doc;
```

Chúng ta có thể kết hợp những bước trên như sau:

```
IStorable isDoc = (IStorable) new Document("Test Document");
```

Nói chung, cách thiết kế tốt nhất là quyết định truy cập những phương thức của giao diện thông qua tham chiếu của giao diện. Do vậy cách tốt nhất là sử dụng isDoc.Read(), hơn là sử dụng doc.Read() trong ví dụ trước. Truy cập thông qua giao diện cho phép chúng ta đổi xử giao diện một cách đa hình. Nói cách khác, chúng ta tạo hai hay nhiều hơn những lớp thực thi giao diện, và sau đó bằng cách truy cập lớp này chỉ thông qua giao diện.

3.5.2.1. Gán đối tượng cho một giao diện

Trong nhiều trường hợp, chúng ta không biết trước một đối tượng có hỗ trợ một giao diện đưa ra. Ví dụ, giả sử chúng ta có một tập hợp những đối tượng Document, một vài đối tượng đã được lưu trữ và số còn lại thì chưa. Và giả sử chúng ta đã thêm giao diện giao diện thứ hai, ICompressible cho những đối tượng để nén dữ liệu và truyền qua mail nhanh chóng:

```
interface ICompressible
{
    void Compress();
    void Decompress();
```

}

Nếu đưa ra một kiểu Document, và ta cũng không biết là lớp này có hỗ trợ giao diện IStorable hay ICompressible hoặc cả hai. Ta có thể có đoạn chương trình sau:

```
Document doc = new Document("Test Document");

IStorable isDoc = (IStorable) doc;

isDoc.Read();

ICompressible icDoc = (ICompressible) doc;

icDoc.Compress();
```

Nếu Document chỉ thực thi giao diện IStorable:

```
public class Document : IStorable
```

phép gán cho ICompressible vẫn được biên dịch bởi vì ICompressible là một giao diện hợp lệ. Tuy nhiên, do phép gán không hợp lệ nên khi chương trình chạy thì sẽ tạo ra một ngoại lệ (exception):

A exception of type System.InvalidCastException was thrown.

Phần ngoại lệ sẽ được trình bày trong phần tiếp theo

3.5.2.2. Toán tử is

Chúng ta muốn kiểm tra một đối tượng xem nó có hỗ trợ giao diện, để sau đó thực hiện các phương thức tương ứng. Trong ngôn ngữ C# có hai cách để thực hiện điều này. Phương pháp đầu tiên là sử dụng toán tử is.

Cú pháp của toán tử is là:

<biểu thức> is <kiểu dữ liệu>

Toán tử is trả về giá trị true nếu biểu thức thường là kiểu tham chiếu có thể được gán an toàn đến kiểu dữ liệu cần kiểm tra mà không phát sinh ra bất cứ ngoại lệ nào. Ví

dụ 3.23 minh họa việc sử dụng toán tử `is` để kiểm tra Document có thực thi giao diện `IStorable` hay `ICompressible`.

- Ví dụ 3.23: Sử dụng toán tử `is`.

```
using System;

interface IStorable
{
    void Read();
    void Write(object obj);
    int Status { get; set; }
}

// giao diện mới

interface ICompressible
{
    void Compress();
    void Decompress();
}

// Document thực thi IStorable

public class Document : IStorable
{
    public Document( string s )
    {
        Console.WriteLine("Creating document with: {0}", s );
    }

    // IStorable

    public void Read()
    {
```

```
        Console.WriteLine("Implementing the Read Method for
IStorable");
    }

// IStorable.WriteLine()

public void Write( object o)
{
    Console.WriteLine("Implementing the Write Method for
IStorable");
}

// IStorable.Status

public int Status
{
    get { return status; }
    set {status = value;}
}

// bien thanh vien luu gia tri cua thuoc tinh

Status private int status = 0;

}

public class Tester
{
    static void Main()
    {
        Document doc = new Document("Test Document");
        // chỉ gán khi an toàn
        if ( doc is IStorable )
        {
            IStorable isDoc = (IStorable) doc;
            isDoc.Read();
        }
    }
}
```

```
// việc kiểm tra này sẽ sai  
if ( doc is ICompressible )  
{  
    ICompressible icDoc = (ICompressible) doc;  
    icDoc.Compress();  
}  
}  
}
```

Trong ví dụ 3.23, hàm Main() lúc này sẽ thực hiện việc gán với interface khi được kiểm tra hợp lệ. Việc kiểm tra này được thực hiện bởi câu lệnh if:

```
if ( doc is IStorable )
```

Biểu thức điều kiện sẽ trả về giá trị true và phép gán sẽ được thực hiện khi đối tượng có thực thi giao diện bên phải của toán tử is.

Tuy nhiên, việc sử dụng toán tử is đưa ra một việc không có hiệu quả. Để hiểu được điều này, chúng ta xem đoạn chương trình được biên dịch ra mã IL. Ở đây sẽ có một ngoại lệ nhỏ, các dòng bên dưới là sử dụng hệ thập lục phân:

IL_0023: isinst ICompressible

IL_0028: brfalse.s IL_0039

IL_002a: ldloc.0

IL_002b: castclass ICompressible

IL_0030: stloc.2

IL_0031: ldloc.2

IL_0032: callvirt instance void ICompressible::Compress() IL_0037: br.s
IL_0043

IL_0039: ldstr “Compressible not supported”

Điều quan trọng xảy ra là khi phép kiểm tra ICompressible ở dòng 23. Từ khóa isinst là mã

MSIL tương ứng với toán tử is. Nếu việc kiểm tra đối tượng (doc) đúng kiểu của kiểu bên phải. Thì chương trình sẽ chuyển đến dòng lệnh 2b để thực hiện tiếp và castclass được gọi. Điều không may là castcall cũng kiểm tra kiểu của đối tượng. Do đó việc kiểm tra sẽ được thực hiện hai lần. Giải pháp hiệu quả hơn là việc sử dụng toán tử as.

3.5.2.3. Toán tử as

Toán tử as kết hợp toán tử is và phép gán bằng cách đầu tiên kiểm tra hợp lệ phép gán (kiểm tra toán tử is trả về true) rồi sau đó phép gán được thực hiện. Nếu phép gán không hợp lệ (khi phép gán trả về giá trị false), thì toán tử as trả về giá trị null.

□Ghi chú: Từ khóa null thể hiện một tham chiếu không tham chiếu đến đâu cả (null reference). Đối tượng có giá trị null tức là không tham chiếu đến đối tượng nào.

Sử dụng toán tử as để loại bỏ việc thực hiện các xử lý ngoại lệ. Đồng thời cũng né tránh việc thực hiện kiểm tra dư thừa hai lần. Do vậy, việc sử dụng tối ưu của phép gán cho giao diện là sử dụng as.

Cú pháp sử dụng toán tử as như sau:

<biểu thức> as <kiểu dữ liệu>

Đoạn chương trình sau thay thế việc sử dụng toán tử is bằng toán tử as và sau đó thực hiện việc kiểm tra xem giao diện được gán có null hay không:

```
static void Main()
{
    Document doc = new Document("Test Document");
    IStorable isDoc = doc as IStorable;
    if ( isDoc != null )
    {
        isDoc.Read();
    }
}
```

```
else
{
    Console.WriteLine("IStorable not supported");
}
ICompressible icDoc = doc as ICompressible;
if ( icDoc != null)
{
    icDoc.Compress();
}
else
{
    Console.WriteLine("Compressible not supported");
}
}
```

Ta có thể so sánh đoạn mã IL sau với đoạn mã IL sử dụng toán tử `is` trước sẽ thấy đoạn mã sau có nhiều hiệu quả hơn:

```
IL_0023:  isinst      ICompressible
IL_0028:  stloc.2
IL_0029:  ldloc.2
IL_002a:  brfalse.s  IL_0034
IL_002c:  ldloc.2
IL_002d:  callvirt    instance void ICompressible::Compress()
```

□Ghi chú: Nếu mục đích của chúng ta là kiểm tra một đối tượng có hỗ trợ một giao diện và sau đó là thực hiện việc gán cho một giao diện, thì cách tốt nhất là sử dụng toán tử `as` là hiệu quả nhất. Tuy nhiên, nếu chúng ta chỉ muốn kiểm tra kiểu dữ liệu và không thực hiện phép gán ngay lúc đó. Có lẽ chúng ta chỉ muốn thực hiện việc kiểm tra nhưng không thực hiện việc gán, đơn giản là chúng ta muốn thêm nó vào danh sách nếu

chúng thực sự là một giao diện. Trong trường hợp này, sử dụng toán tử `is` là cách lựa chọn tốt nhất.

3.5.2.4. Giao diện đối lập với lớp trừu tượng

Giao diện rất giống như các lớp trừu tượng. Thật vậy, chúng ta có thể thay thế khai báo của `IStorable` trở thành một lớp trừu tượng:

```
abstract class Storable  
{  
    abstract public void Read();  
    abstract public void Write();  
}
```

Bây giờ lớp `Document` có thể thừa kế từ lớp trừu tượng `IStorable`, và cũng không có gì khác nhiều so với việc sử dụng giao diện.

Tuy nhiên, giả sử chúng ta mua một lớp `List` từ một hàng thứ ba và chúng ta muốn kết hợp với lớp có sẵn như `Storable`. Trong ngôn ngữ C++ chúng ta có thể tạo ra một lớp `StorableList` kế thừa từ `List` và cả `Storable`. Nhưng trong ngôn ngữ C# chúng ta không thể làm được, chúng ta không thể kế thừa từ lớp trừu tượng `Storable` và từ lớp `List` bởi vì trong C# không cho phép thực hiện đa kế thừa từ những lớp.

Tuy nhiên, ngôn ngữ C# cho phép chúng ta thực thi bất cứ những giao diện nào và dẫn xuất từ một lớp cơ sở. Do đó, bằng cách làm cho `Storable` là một giao diện, chúng ta có thể kế thừa từ lớp `List` và cũng từ `IStorable`. Ta có thể tạo lớp `StorableList` như sau:

```
public class StorableList : List, IStorable  
{  
    // phương thức List...  
    ....  
    public void Read()  
    {...}
```

```
public void Write( object o)  
{...}  
//....  
}
```

3.5.3. Thực thi phủ quyết giao diện

Khi thực thi một lớp chúng ta có thể tự do đánh dấu bất kỳ hay tất cả các phương thức thực thi giao diện như là một phương thức ảo. Ví dụ, lớp Document thực thi giao diện IStorable và có thể đánh dấu các phương thức Read() và Write() như là phương thức ảo. Lớp Document có thể đọc và viết nội dung của nó vào một kiểu dữ liệu File. Những người phát triển sau có thể dẫn xuất một kiểu dữ liệu mới từ lớp Document, có thể là lớp Note hay lớp EmailMessage, và những người này mong muốn lớp Note đọc và viết vào cơ sở dữ liệu hơn là vào một tập tin.

Ví dụ 3.24 mở rộng từ ví dụ 3.23 và minh họa việc phủ quyết một thực thi giao diện. Phương thức Read() được đánh dấu như phương thức ảo và thực thi bởi Document.Read() và cuối cùng là được phủ quyết trong kiểu dữ liệu Note được dẫn xuất từ Document.

- Ví dụ 3.24: Phủ quyết thực thi giao diện.

```
using System;  
  
interface IStorable  
{  
    void Read();  
    void Write();  
}  
// lớp Document đơn giản thực thi giao diện IStorable  
public class Document : IStorable  
{  
    // bộ khởi dựng
```

```
public Document( string s)
{
    Console.WriteLine("Creating document with: {0}", s);
}

// đánh dấu phương thức Read ảo
public virtual void Read()
{
    Console.WriteLine("Document Read Method for IStorable");
}

// không phải phương thức ảo
public void Write()
{
    Console.WriteLine("Document Write Method for IStorable");
}

}

// lớp dẫn xuất từ Document
public class Note : Document
{
    public Note( string s ) : base(s)
    {
        Console.WriteLine("Creating note with: {0}", s);
    }

    // phủ quyết phương thức Read()
    public override void Read()
    {
        Console.WriteLine("Overriding the Read Method for Note!");
    }

    // thực thi một phương thức Write riêng của lớp
}
```

```
public void Write()
{
    Console.WriteLine("Implementing the Write method for
Note!");
}

}

public class Tester
{
    static void Main()
    {
        // tạo một đối tượng Document
        Document theNote = new Note("Test Note");
        IStorable isNote = theNote as IStorable;
        if ( isNote != null)
        {
            isNote.Read();
            isNote.Write();
        }
        Console.WriteLine("\n");
        // trực tiếp gọi phương thức
        theNote.Read();
        theNote.Write();
        Console.WriteLine("\n");
        // tạo đối tượng Note
        Note note2 = new Note("Second Test");
        IStorable isNote2 = note2 as IStorable;
        if ( isNote != null )
        {
```

```
    isNote2.Read();

    isNote2.Write();

}

Console.WriteLine("\n");

// trực tiếp gọi phương thức note2.Read(); note2.Write();

}

}
```

□ Kết quả:

Creating document with: Test Note Creating note with: Test Note Overriding the Read method for Note! Document Write Method for IStorable

Overriding the Read method for Note! Document Write Method for IStorable

Creating document with: Second Test Creating note with: Second Test Overriding the Read method for Note! Document Write Method for IStorable

Overriding the Read method for Note! Implementing the Write method for Note!

Trong ví dụ trên, lớp Document thực thi một giao diện đơn giản là IStorable:

interface IStorable

{

void Read();

void Write();

}

Người thiết kế của lớp Document thực thi phương thức Read() là phương thức ảo nhưng không tạo phương thức Write() tương tự như vậy:

public virtual void Read()

Trong ứng dụng thế giới thực, chúng ta cũng đánh dấu cả hai phương thức này là phương thức ảo. Tuy nhiên trong ví dụ này chúng ta minh họa việc người phát triển có thể tùy ý chọn các phương thức ảo của giao diện mà lớp thực thi.

Một lớp mới Note dẫn xuất từ Document:

```
public class Note : Document
```

Việc phủ quyết phương thức Read() trong lớp Note là không cần thiết, nhưng ở đây ta tự do làm điều này:

```
public override void Read()
```

Trong lớp Tester, phương thức Read() và Write() được gọi theo bốn cách sau:

- Thông qua lớp cơ sở tham chiếu đến đối tượng của lớp dẫn xuất
- Thông qua một giao diện tạo từ lớp cơ sở tham chiếu đến đối tượng dẫn xuất
- Thông qua một đối tượng dẫn xuất
- Thông qua giao diện tạo từ đối tượng dẫn xuất

Thực hiện cách gọi thứ nhất, một tham chiếu Document được tạo ra, và địa chỉ của một đối tượng mới là lớp dẫn xuất Note được tạo trên heap và gán trở lại cho đối tượng Document:

```
Document theNote = new Note("Test Note");
```

Một tham chiếu giao diện được tạo ra và toán tử as được sử dụng để gán Document cho tham chiếu giao diện IStorable:

```
IStorable isNote = theNote as IStorable;
```

Sau đó gọi phương thức Read() và Write() thông qua giao diện. Kết xuất của phương thức Read() được thực hiện một cách đa hình nhưng phương thức Write() thì không, do đó ta có kết xuất sau:

Overriding the Read method for Note! Document Write Method for IStorable

Phương thức Read() và Write() cũng được gọi trực tiếp từ bản thân đối tượng:

```
theNote.Read();
```

```
theNote.Write();
```

và một lần nữa chúng ta thấy việc thực thi đa hình làm việc:

Overriding the Read method for Note! Document Write Method for IStorable

Trong trường hợp này, phương thức Read() của lớp Note được gọi, và phương thức Write()

của lớp Document được gọi.

Để chứng tỏ rằng kết quả này của phương thức phủ quyết, chúng ta tiếp tục tạo đối tượng Note thứ hai và lúc này ta gán cho một tham chiếu Note. Điều này được sử dụng để minh họa cho những trường hợp cuối cùng (gọi thông qua đối tượng dẫn xuất và gọi thông qua giao diện được tạo từ đối tượng dẫn xuất):

```
Note note2 = new Note("Second Test");
```

Một lần nữa, khi chúng ta gán cho một tham chiếu, phương thức phủ quyết Read() được gọi. Tuy nhiên, khi những phương thức được gọi trực tiếp từ đối tượng Note:

```
note2.Read();
```

```
note2.Write();
```

kết quả cho ta thấy rằng cách phương thức của Note được gọi chứ không phải của một phương thức Document:

Overriding the Read method for Note! Implementing the Write method for Note!

3.5.4. Thực thi giao diện tương minh

Trong việc thực thi giao diện cho tới giờ, những lớp thực thi (trong trường hợp này là Document) tạo ra các phương thức thành viên cùng ký hiệu và kiểu trả về như là phương thức được mô tả trong giao diện. Chúng ta không cần thiết khai báo tương minh rằng đây là một thực thi của một giao diện, việc này được hiểu ngầm bởi trình biên dịch.

Tuy nhiên, có vấn đề xảy ra khi một lớp thực thi hai giao diện và cả hai giao diện này có các phương thức cùng một ký hiệu. Ví dụ 8.5 tạo ra hai giao diện: IStorable và

ITalk. Sau đó thực thi phương thức Read() trong giao diện ITalk để đọc ra tiếng nội dung của một cuốn sách. Không may là phương thức này sẽ tranh chấp với phương thức Read() của IStorable mà Document phải thực thi.

Bởi vì cả hai phương thức IStorable và ITalk có cùng phương thức Read(), việc thực thi lớp Document phải sử dụng thực thi tường minh cho mỗi phương thức. Với việc thực thi tường minh, lớp thực thi Document sẽ khai báo tường minh cho mỗi phương thức:

```
void ITalk.Read();
```

Điều này sẽ giải quyết việc tranh chấp, nhưng nó sẽ tạo ra hàng loạt các hiệu ứng thú vị.

Đầu tiên, không cần thiết sử dụng thực thi tường minh với những phương thức khác của Talk:

```
public void Talk();
```

vì không có sự tranh chấp cho nên ta khai báo như thông thường.

Điều quan trọng là các phương thức thực thi tường minh không có bù sung truy cập:

```
void ITalk.Read();
```

Phương thức này được hiểu ngầm là public.

Thật vậy, một phương thức được khai báo tường minh thì sẽ không được khai báo với các từ khóa bù sung truy cập: abstract, virtual, override, và new.

Một điều quan trọng khác là chúng ta không thể truy cập phương thức thực thi tường minh thông qua chính đối tượng. Khi chúng ta viết:

```
theDoc.Read();
```

Trình biên dịch chỉ hiểu rằng chúng ta thực thi phương thức giao diện ngầm định cho IStorable. Chỉ một cách duy nhất truy cập các phương thức thực thi tường minh là thông qua việc gán cho giao diện để thực thi:

```
ITalk itDoc = theDoc as ITalk;  
  
if ( itDoc != null )  
  
{  
  
    itDoc.Read();  
  
}
```

Sử dụng thực thi tường minh được áp dụng trong ví dụ 3.25

- Ví dụ 3.25: Thực thi tường minh.

```
using System;  
  
interface IStorable  
{  
  
    void Read();  
  
    void Write();  
  
}  
  
interface ITalk  
{  
  
    void Talk();  
  
    void Read();  
  
}  
  
// lớp Document thực thi hai giao diện  
public class Document : IStorable, ITalk  
{  
  
    // bộ khởi dựng  
  
    public Document( string s)  
    {  
  
        Console.WriteLine("Creating document with: {0}",s);  
  
    }  
  
    // tạo phương thức ảo
```

```
public virtual void Read()
{
    Console.WriteLine("Implementing IStorable.Read");
}

// thực thi bình thường
public void Write()
{
    Console.WriteLine("Implementing IStorable.Write");
}

// thực thi tường minh
void ITalk.Read()
{
    Console.WriteLine("Implementing ITalk.Read");
}

public void Talk()
{
    Console.WriteLine("Implementing ITalk.Talk");
}

}

public class Tester
{
    static void Main()
    {
        // tạo đối tượng Document
        Document theDoc = new Document("Test Document");
        IStorable isDoc = theDoc as IStorable;
        if ( isDoc != null )
        {
    }
```

```
        isDoc.Read();  
    }  
  
    ITalk itDoc = theDoc as ITalk;  
  
    if ( itDoc != null )  
    {  
        itDoc.Read();  
    }  
  
    theDoc.Read();  
  
    theDoc.Talk();  
}  
}
```

□ Kết quả:

Creating document with: Test Document Implementing IStorable.Read
Implementing ITalk.Read

Implementing IStorable.Read

Implementing ITalk.Talk

3.5.5. Lựa chọn thể hiện phương thức giao diện

Những người thiết kế lớp có thể thu được lợi khi một giao diện được thực thi thông qua thực thi tường minh và không cho phép các thành phần client của lớp truy cập trừ phi sử dụng thông qua việc gán cho giao diện.

Giả sử nghĩa của đối tượng Document chỉ ra rằng nó thực thi giao diện IStorable, nhưng không muốn phương thức Read() và Write() là phần giao diện public của lớp Document. Chúng ta có thể sử dụng thực thi tường minh để chắc chắn chỉ có thể truy cập thông qua việc gán cho giao diện. Điều này cho phép chúng ta lưu trữ ngữ nghĩa của lớp Document trong khi vẫn có thể thực thi được giao diện IStorable. Nếu thành phần client muốn đối tượng thực thi giao diện IStorable, nó có thể thực hiện gán tường minh cho giao diện để gọi các phương thức thực thi giao diện.

Nhưng khi sử dụng đối tượng Document thì nghĩa là không có phương thức Read() và Write().

Thật vậy, chúng ta có thể lựa chọn thể hiện những phương thức thông qua thực thi tường minh, do đó chúng ta có thể trưng bày một vài phương thức thực thi như là một phần của lớp Document và một số phương thức khác thì không. Trong ví dụ 8.5, đối tượng Document trưng bày phương thức Talk() như là phương thức của lớp Document, nhưng phương thức Talk.Read() chỉ được thể hiện thông qua gán cho giao diện. Thậm chí nếu IStorable không có phương thức Read(), chúng ta cũng có thể chọn thực thi tường minh phương thức Read() để phương thức không được thể hiện ra bên ngoài như các phương thức của Document.

Chúng ta lưu ý rằng vì thực thi giao diện tường minh ngăn ngừa việc sử dụng từ khóa virtual, một lớp dẫn xuất có thể được hỗ trợ để thực thi lại phương thức. Do đó, nếu Note dẫn xuất từ Document, nó có thể được thực thi lại phương thức Talk.Read() bởi vì lớp Document thực thi phương thức Talk.Read() không phải ảo.

3.5.6. Ẩn thành viên

Ngôn ngữ C# cho phép ẩn các thành viên của giao diện. Ví dụ, chúng ta có một giao diện IBase với một thuộc tính P:

```
interface IBase  
{  
    int P { get; set; }  
}
```

và sau đó chúng ta dẫn xuất từ giao diện này ra một giao diện khác, IDerived, giao diện mới này làm ẩn thuộc tính P với một phương thức mới P():

```
interface IDerived : IBase  
{  
    new int P();  
}
```

Việc cài đặt này là một ý tưởng tốt, bây giờ chúng ta có thể ẩn thuộc tính P trong lớp cơ sở. Một thực thi của giao diện dẫn xuất này đòi hỏi tối thiểu một thành

viên giao diện tường minh. Chúng ta có thể sử dụng thực thi tường minh cho thuộc tính của lớp cơ sở hoặc của phương thức dẫn xuất, hoặc chúng ta có thể sử dụng thực thi tường minh cho cả hai. Do đó, ba phiên bản được viết sau đều hợp lệ:

```
class myClass : IDerived
{
    // thực thi tường minh cho thuộc tính cơ sở
    int IBase.P { get{...} }

    // thực thi ngầm định phương thức dẫn xuất
    public int P() {...}

}

class myClass : IDerived
{
    // thực thi ngầm định cho thuộc tính cơ sở
    public int P { get{...} }

    // thực thi tường minh phương thức dẫn xuất
    int IDerived.P() {...}

}

class myClass : IDerived
{
    // thực thi tường minh cho thuộc tính cơ sở
    int IBase.P { get{...} }

    // thực thi tường minh phương thức dẫn xuất
    int IDerived.P(){...}

}
```

Truy cập lớp không cho dẫn xuất và kiểu giá trị

Nói chung, việc truy cập những phương thức của một giao diện thông qua việc gán cho giao diện thì thường được thích hơn. Ngoại trừ đối với kiểu giá trị (như cấu trúc) hoặc với các lớp không cho dẫn xuất (sealed class). Trong trường hợp này, cách ưu chuộng hơn là gọi phương thức giao diện thông qua đối tượng.

Khi chúng ta thực thi một giao diện trong một cấu trúc, là chúng ta đang thực thi nó trong một kiểu dữ liệu giá trị. Khi chúng ta gán cho một tham chiếu giao diện, có một boxing ngầm định của đối tượng. Chẳng may khi chúng ta sử dụng giao diện để bổ sung đối tượng, nó là một đối tượng đã boxing, không phải là đối tượng nguyên thủy cần được bổ sung. Xa hơn nữa, nếu chúng ta thay đổi kiểu dữ liệu giá trị, thì kiểu dữ liệu được boxing vẫn không thay đổi. Ví dụ 3.26 tạo ra một cấu trúc và thực thi một giao diện IStorable và minh họa việc boxing ngầm định khi gán một cấu trúc cho một tham chiếu giao diện.

- Ví dụ 3.26: Tham chiếu đến kiểu dữ liệu giá trị.

```
using System;  
  
// khai báo một giao diện đơn  
  
interface IStorable  
{  
  
    void Read();  
  
    int Status { get; set; }  
  
}  
  
// thực thi thông qua cấu trúc  
  
public struct myStruct : IStorable  
{  
  
    public void Read()  
    {  
  
        Console.WriteLine("Implementing IStorable.Read");  
  
    }  
}
```

```
public int Status
{
    get{ return status;}
    set{status = value;}
}

// biến thành viên lưu giá trị thuộc tính
Status private int status;
}

public class Tester
{
    static void Main()
    {
        // tạo một đối tượng myStruct
        myStruct theStruct = new myStruct();
        theStruct.Status = -1; // khởi tạo
        Console.WriteLine("theStruct.Status: {0}",
theStruct.Status);

        // thay đổi giá trị the
        Struct.Status = 2;
        Console.WriteLine("Changed object");
        Console.WriteLine("theStruct.Status: {0}",
theStruct.Status);

        // gán cho giao diện
        // boxing ngầm định
        IStorable isTemp = (IStorable) theStruct;
        // thiết lập giá trị thông qua tham chiếu giao diện
        isTemp.Status = 4;
        Console.WriteLine("Changed interface");
```

```
Console.WriteLine("theStruct.Status: {0}, isTemp: {1}",
    theStruct.Status, isTemp.Status);
// thay đổi giá trị một lần nữa theStruct.Status = 6;
Console.WriteLine("Changed object.");
Console.WriteLine("theStruct.Status: {0}, isTemp: {1}",
    theStruct.Status, issTemp.Status);
}
}
```

□ Kết quả:

theStruct.Status: -1

Changed object. theStruct.Status: 2

Changed interface theStruct.Status: 2, isTemp: 4

Changed object

theStruct.Status: 6, isTemp: 4

Trong ví dụ 3.26, giao diện IStorable có một phương thức Read() và một thuộc tính là Status. Giao diện này được thực thi bởi một cấu trúc tên là myStruct:

```
public struct myStruct : IStorable
```

Đoạn mã nguồn thú vị bên trong Tester. Chúng ta bắt đầu bằng việc tạo một thể hiện của cấu trúc và khởi tạo thuộc tính là -1, sau đó giá trị của status được in ra:0

```
myStruct theStruct = new myStruct();
theStruct.Status = -1; // khởi tạo
Console.WriteLine("theStruct.Status: {0}", theStruct.status);
```

Kết quả là giá trị của status được thiết lập:

```
theStruct.Status = -1;
```

Kết tiếp chúng ta truy cập thuộc tính để thay đổi status, một lần nữa thông qua đối tượng giá trị:

```
// thay đổi giá trị  
  
theStruct.Status = 2;  
  
Console.WriteLine("Changed object");  
  
Console.WriteLine("theStruct.Status: {0}", theStruct.Status);
```

kết quả chỉ ra sự thay đổi: Changed object theStruct.Status: 2

Tại điểm này, chúng ta tạo ra một tham chiếu đến giao diện IStorable, một đối tượng giá trị theStruct được boxing ngầm và gán lại cho tham chiếu giao diện. Sau đó chúng ta dùng giao diện để thay đổi giá trị của status bằng 4:

```
// gán cho một giao diện  
  
// boxing ngầm định  
  
IStorable isTemp = (IStorable) theStruct;  
  
// thiết lập giá trị thông qua tham chiếu giao diện  
  
isTemp.Status = 4;  
  
Console.WriteLine("Changed interface");  
  
Console.WriteLine("theStruct.Status:{0},isTemp:{1}",  
theStruct.Status, isTemp.Status);
```

nhiều chúng ta đã thấy kết quả thực hiện có một điểm khác biệt:

Changed interface theStruct.Status: 2, isTemp: 4

Điều xảy ra là: đối tượng được giao diện tham chiếu đến thay đổi giá trị status bằng 4, nhưng đối tượng giá trị cấu trúc không thay đổi. Thậm chí có nhiều thứ vị hơn khi chúng ta truy cập phương thức thông qua bản thân đối tượng:

```
// thay đổi giá trị lần nữa
```

```
theStruct.Status = 6;
```

```
Console.WriteLine("Changed object");
Console.WriteLine("theStruct.Status: {0}, isTemp: {1}",
theStruct.Status, isTemp.Status);
```

kết quả đổi tượng giá trị thay đổi nhưng đổi tượng được boxing và được giao diện tham chiếu không thay đổi:

Changed object theStruct.Status: 6,

isTemp: 4

C. HÌNH THÚC VÀ PHƯƠNG PHÁP GIẢNG DẠY

- Trình chiếu powerpoint
- Đặt vấn đề, trao đổi
- Thực nghiệm kết hợp với máy tính

D. TÀI LIỆU THAM KHẢO

- [1] Professional C#, 2nd Edition, Wrox Press Ltd.
- [2] A programmer's Introduction to C#, Eric Gunnerson, Apress, 2000
- [3] Programming C#, Jesse Liberty, O'Reilly, First Edition, 2001
- [4] C# bible, Jeff Ferguson et al, Wiley Publishing, 2002
- [5] Thinking in C#, Larry O'Brien, Bruce Eckel, Prentice Hall.
- [6] Presenting C#, Sams Publishing, 2002
- [7] C# Language Reference, Anders Hejlsberg and Scott Wiltamuth, Microsoft Corp.

E. CÂU HỎI

Câu hỏi 1: Có phải chúng ta chỉ nên sử dụng lớp với các dữ liệu thành viên?

Trả lời 1: Nói chung là chúng ta không nên sử dụng lớp chỉ với dữ liệu thành viên. Ý nghĩa của một lớp hay của lập trình hướng đối tượng là khả năng đóng gói các chức năng và dữ liệu vào trong một gói đơn.

Câu hỏi 2: Có phải tất cả những dữ liệu thành viên luôn luôn được khai báo là public để bên ngoài có thể truy cập chúng?

Trả lời 2: Nói chung là không. Do vấn đề che dấu dữ liệu trong lập trình hướng đối tượng, xu hướng là dữ liệu bên trong chỉ nên dùng cho các phương thức thành viên. Tuy nhiên, như chúng ta đã biết khái niệm thuộc tính cho phép các biến thành viên được truy cập từ bên ngoài thông qua hình thức như là phương thức.

Câu hỏi 3: Có phải có rất nhiều lớp được xây dựng sẵn và tôi có thể tìm chúng ở đâu?

Trả lời 3: Microsoft cung cấp rất nhiều các lớp gọi là các lớp cơ sở .NET. Những lớp này được tổ chức bên trong các namespace. Chúng ta có thể tìm tài liệu về các lớp này trong thư viện trực tuyến của Microsoft. Và một số lớp thường sử dụng cũng được trình bày lần lượt trong các ví dụ của giáo trình này.

Câu hỏi 4: Sự khác nhau giữa tham số (parameter) và đối mục (argument)?

Trả lời 4: Tham số được định nghĩa là những thứ được truyền vào trong một phương thức. Một tham số xuất hiện với định nghĩa của phương thức ở đầu phương thức. Một đối mục là giá trị được truyền vào phương thức. Chúng ta truyền những đối mục vào phương thức phù hợp với những tham số đã khai báo của phương thức.

Câu hỏi 5: Chúng ta có thể tạo phương thức bên ngoài của lớp hay không?

Trả lời 5: Mặc dù trong những ngôn ngữ khác, chúng ta có thể tạo các phương thức bên ngoài của lớp. Nhưng trong C# thì không, C# là hướng đối tượng, do vậy tất cả các mã nguồn phải được đặt bên trong một lớp.

Câu hỏi 6: Có phải những phương thức và lớp trong C# hoạt động tương tự như trong các ngôn ngữ khác như C++ hay Java?

Trả lời 6: Trong hầu hết các phần thì chúng tương tự như nhau. Tuy nhiên, mỗi ngôn ngữ cũng có những khác biệt riêng. Một ví dụ sự khác nhau là C# không cho phép tham số mặc định bên trong một phương thức. Trong ngôn ngữ C++ thì chúng ta có thể khai báo các tham số mặc định lúc định nghĩa phương thức và khi gọi phương thức thì có thể không cần truyền giá trị vào, phương thức sẽ dùng giá trị mặc định. Trong C# thì không được phép. Nói chung là còn nhiều sự khác nhau nữa, nhưng xin dành cho bạn đọc tự tìm hiểu.

Câu hỏi 7: Phương thức tĩnh có thể truy cập được thành viên nào và không truy cập được thành viên nào trong một lớp?

Trả lời 7: Phương thức tĩnh chỉ truy cập được các thành viên tĩnh trong một lớp.

Câu hỏi 8: Có cần thiết phải chỉ định từ khóa override trong phương thức phủ quyết của lớp dẫn xuất hay không?

Trả lời 8: Có, chúng ta phải khai báo rõ ràng từ khóa override với phương thức phủ quyết phương thức ảo (của lớp cơ sở) bên trong lớp dẫn xuất.

Câu hỏi 9: Lớp trừu tượng là thế nào? Có thể tạo đối tượng cho lớp trừu tượng hay không?

Trả lời 9: Lớp trừu tượng không có sự thực thi, các phương thức của nó được tạo ra chỉ là hình thức, tức là chỉ có khai báo, do vậy phần định nghĩa bắt buộc phải được thực hiện ở các lớp dẫn xuất từ lớp trừu tượng này. Do chỉ là lớp trừu tượng, không có sự thực thi nên chúng ta không thể tạo thể hiện hay tạo đối tượng cho lớp trừu tượng này.

Câu hỏi 10: Có phải khi tạo một lớp thì phải kế thừa từ một lớp nào không?

Trả lời 10: Không nhất thiết như vậy, tuy nhiên trong C#, thì tất cả các lớp được tạo điều phải dẫn xuất từ lớp Object. Cho dù chúng có được khai báo tường minh hay không. Do đó Object là lớp gốc của tất cả các lớp được xây dựng trong C#. Một điều thú vị là các kiểu dữ liệu giá trị như kiểu nguyên, thực, ký tự cũng được dẫn xuất từ Object.

Câu hỏi 11: Lớp lồng bên trong một lớp là như thế nào?

Trả lời 11: Lớp lồng bên trong một lớp hay còn gọi là lớp nội được khai báo với từ khóa internal, chứa bên trong phạm vi của một lớp. Lớp nội có thể truy cập được các thành viên private của lớp mà nó chứa bên trong

Câu hỏi 12: Có thể kế thừa từ một lớp cơ sở được viết trong ngôn ngữ khác ngôn ngữ C#?

Trả lời 12: Được, một trong những đặc tính của .NET là các lớp có thể kế thừa từ các lớp được viết từ ngôn ngữ khác. Do vậy, trong C# ta có thể kế thừa một lớp được viết

từ ngôn ngữ khác của .NET. Và những ngôn ngữ khác cũng có thể kế thừa từ các lớp C# mà ta tạo ra

Câu hỏi 13: Có phải khi xây dựng các lớp chúng ta chỉ cần dùng nạp chồng toán tử với các chức năng tính toán ?

Trả lời 13: Đúng là như vậy, việc thực hiện nạp chồng toán tử rất tự nhiên và trực quan. Tuy nhiên một số ngôn ngữ .NET như VB.NET không hỗ trợ việc nạp chồng toán tử nên, tốt nhất nếu muốn cho lớp trong C# của chúng ta có thể được gọi từ ngôn ngữ khác không hỗ trợ nạp chồng toán tử thì nên xây dựng các phương thức tương đương để thực hiện cùng chức năng như: Add, Sub, Mul,..

Câu hỏi 14: Những điều lưu ý nào khi sử dụng nạp chồng toán tử trong một lớp?

Trả lời 14: Nói chung là khi nào thật cần thiết và ít gây ra sự nhầm lẫn. Ví dụ như ta xây dựng lớp Employee có nhiều thuộc tính số như lương, thâm niên, tuổi... Chúng ta muốn xây dựng toán tử ++ cho lương nhưng có thể làm nhầm lẫn với việc tăng số năm công tác, hay tăng tuổi. Do vậy việc sử dụng nạp chồng toán tử cũng phải cân nhắc tránh gây nhầm lẫn. Tốt nhất là sử dụng trong lớp có ít thuộc tính số...

Câu hỏi 15: Khi xây dựng toán tử so sánh thì có phải chỉ cần dùng toán tử so sánh bằng?

Trả lời 15: Đúng là nếu cần dùng toán tử so sánh nào thì chúng ta có thể chỉ tạo ra duy nhất toán tử so sánh đó mà thôi. Tuy nhiên, tốt hơn là chúng ta cũng nên xây dựng thêm toán tử so sánh khác như: so sánh khác, so sánh nhỏ hơn, so sánh lớn hơn... Việc này sẽ làm cho lớp của chúng ta hoàn thiện hơn.

Câu hỏi 16: Có sự khác nhau giữa cấu trúc và lớp?

Trả lời 16: Đúng có một số sự khác nhau giữa cấu trúc và lớp. Như đã đề cập trong lý thuyết thì lớp là kiểu dữ liệu tham chiếu còn cấu trúc là kiểu dữ liệu giá trị. Điều này được xem là sự khác nhau căn bản giữa cấu trúc và lớp. Ngoài ra cấu trúc cũng không cho phép có hàm hủy và tạo bộ khởi động không tham số tường minh. Cấu trúc cũng khác lớp là cấu trúc là kiểu có lập tường minh, tức là không cho phép kế thừa từ nó. Và

nó cũng không kế thừa được từ bất cứ lớp nào khác. Mặc nhiên, các cấu trúc vẫn kế thừa từ Object như bất cứ kiểu dữ liệu giá trị nào khác trong C#.

Câu hỏi 17: Trong hai dạng mảng và tập hợp thì lại nào chứa cấu trúc tốt hơn?

Trả lời 17: Cấu trúc có hiệu quả khi sử dụng trong mảng hơn là lưu chúng dưới dạng tập hợp. Dạng tập hợp tốt với kiểu dữ liệu tham chiếu.

Câu hỏi 18: Cấu trúc được lưu trữ ở đâu?

Trả lời 18: Cấu trúc như đã đề cập là kiểu dữ liệu giá trị nên nó được lưu trữ trên stack của chương trình. Ngược với kiểu tham chiếu được đặt trên heap.

Câu hỏi 19: Khi truyền cấu trúc cho một phương thức thì dưới hình thức nào?

Trả lời 19: Do là kiểu giá trị nên khi truyền một đối tượng cấu trúc cho một phương thức thì nó được truyền dưới dạng tham trị chứ không phải tham chiếu.

Câu hỏi 20: Vậy làm thế nào truyền cấu trúc dưới dạng tham chiếu cho một phương thức?

Trả lời 20: Cũng giống như truyền tham chiếu một kiểu giá trị như int, long, char. Ta khai báo khóa ref cho các tham số kiểu cấu trúc. Và khi gọi phương thức thì thêm từ khóa ref vào trước đối mục cấu trúc được truyền vào.

Câu hỏi 21: So sánh giữa lớp và giao diện?

Trả lời 21: Giao diện khác với lớp ở một số điểm sau: giao diện không cung cấp bất cứ sự thực thi mã nguồn nào cả. Điều này sẽ được thực hiện tại các lớp thực thi giao diện. Một giao diện đưa ra chỉ để nói rằng có cung cấp một số sự xác nhận hướng dẫn cho những điều gì đó xảy ra và không đi vào chi tiết. Một điều khác nữa là tất cả các thành viên của giao diện được giả sử là public ngầm định. Nếu chúng ta cố thay đổi thuộc tính truy cập của thành viên trong giao diện thì sẽ nhận được lỗi.

Giao diện chỉ chứa những phương thức, thuộc tính, sự kiện, chỉ mục. Và không chứa dữ liệu thành viên, bộ khởi động, và bộ hủy. Chúng cũng không chứa bất cứ thành viên static nào cả.

Câu hỏi 22: Sự khác nhau giữa giao diện và lớp trùu tượng?

Trả lời 22: Sự khác nhau cơ bản là sự kế thừa. Một lớp có thể kế thừa nhiều giao diện cùng một lúc, nhưng không thể kế thừa nhiều hơn một lớp trùu tượng.

Câu hỏi 23: Các lớp thực thi giao diện sẽ phải làm gì?

Trả lời 23: Các lớp thực thi giao diện phải cung cấp các phần thực thi chi tiết cho các phương thức, thuộc tính, chỉ mục, sự kiện được khai báo trong giao diện.

Câu hỏi 24: Có bao nhiêu cách gọi một phương thức được khai báo trong giao diện?

Trả lời 24: Có 4 cách gọi phương thức được khai báo trong giao diện:

- Thông qua lớp cơ sở tham chiếu đến đối tượng của lớp dẫn xuất
- Thông qua một giao diện tạo từ lớp cơ sở tham chiếu đến đối tượng dẫn xuất
- Thông qua một đối tượng dẫn xuất
- Thông qua giao diện tạo từ đối tượng dẫn xuất

Câu hỏi 25: Các thành viên của giao diện có thể có những thuộc tính truy cập nào?

Trả lời 25: Mặc định các thành viên của giao diện là public. Vì mục tiêu của giao diện là xây dựng cho các lớp khác sử dụng. Nếu chúng ta thay đổi thuộc tính này như là internal, protected hay private thì sẽ gây ra lỗi.

Câu hỏi 26: Chúng ta có thể tạo thể hiện của giao diện một cách trực tiếp được không?

Trả lời 26: Không thể tạo thể hiện của giao diện trực tiếp bằng khai báo new được. Chúng ta chỉ có thể tạo thể hiện giao diện thông qua một phép gán với đối tượng thực thi giao diện.

CHƯƠNG 4: WINDOWS FORM

A. MỤC TIÊU CHƯƠNG

1. VỀ KIẾN THỨC

Cung cấp cho sinh viên những kiến thức về:

- Lập trình Windows Form
- Các bước xây dựng ứng dụng Windows Form
- Các lớp điều khiển và xử lý sự kiện

2. VỀ KỸ NĂNG

Sau khi học xong chương này sinh viên có thể vận dụng những kiến thức nền tảng về lập trình Windows form vào các ứng dụng trong nhiều lĩnh vực.

B. NỘI DUNG

4.1. Bắt đầu với Windows form

Trong chương 2, chúng ta đã được học ngôn ngữ C# được sử dụng vào ứng dụng Console. Chương này chúng ta sẽ thảo luận làm thế nào để làm việc với một ứng dụng Windows Form. Nó là tập hợp những lớp chứa giao diện đồ họa người dùng (GUI) của ứng dụng desktop cổ điển. Việc tạo ra các Form giao tiếp giúp người dùng dễ dàng hơn trong các thao tác của mình.

Trước đây thì mỗi ngôn ngữ lập trình đều có cách khác nhau để tạo ra Windows, Textbox, Buttons,... Giờ đây thì chức năng này đã đưa vào thư viện của lớp .NET Framework, trở thành phần của namespace System.Windows.Forms

Trong chương này, form được xem như là bộ phận trung tâm của ứng dụng desktop. Chúng ta sẽ xem xét Form được khởi tạo như thế nào và nó phát sinh sự kiện ra sao. Bên cạnh đó, chúng ta sẽ khảo sát những Windows Controls để xây dựng nên những ứng dụng Windows hiệu quả.

4.1.1. Ứng dụng Windows Form đơn giản

Đầu tiên chúng ta hãy bắt đầu với ứng dụng Winform đơn giản hiển thị dòng chữ “Hello word” trong một nhãn Label.

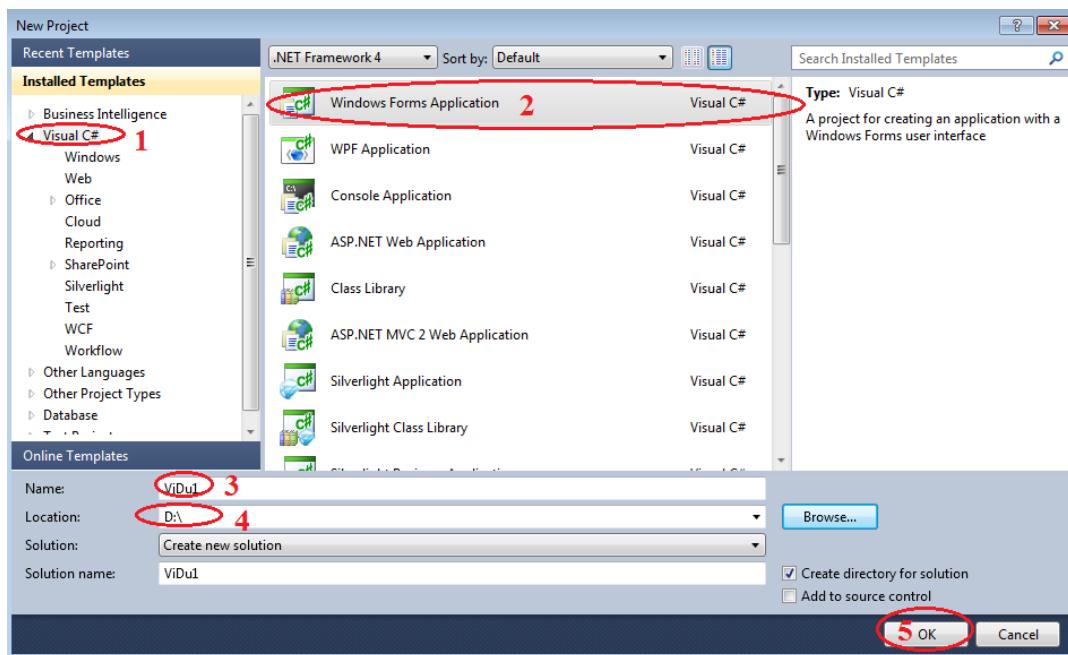
Cách 1: Chúng ta có thể soạn thảo trong chương trình Notepad, và biên dịch bằng dòng lệnh csc.exe

```
using System;
using System.Windows.Forms;
public class Form1 : System.Windows.Forms.Form
{
    private System.Windows.Forms.Label label1;
    public Form1() // constructor
    {
        InitializeComponent();
        this.label1.Text = "Hello world!";
    }
    static void Main()
    {
        Application.Run(new Form1()); // Run our form
    }
    private void InitializeComponent() // helper method to keep
constructor clean
    {
        this.SuspendLayout(); // for better performance
        // textBox1
        this.label1 = new System.Windows.Forms.Label();
        this.label1.Location = new System.Drawing.Point(28, 55);
        this.label1.Name = "lbHello";
        this.label1.Size = new System.Drawing.Size(100, 20);
        this.label1.TabIndex = 1;
        // Form1
        this.AutoScaleDimensions = new System.Drawing.SizeF(6.0F,
```

```
13.0F);  
  
    this.AutoScaleMode =  
System.Windows.Forms.AutoScaleMode.Font;  
  
    this.ClientSize = new System.Drawing.Size(284, 264);  
  
    this.Name = "Form1";  
  
    this.Text = "Form1";  
  
    this.Controls.Add(this.label1);  
  
    this.ResumeLayout(false);  
  
}  
  
}
```

Cách 2: Tuy nhiên thay vì cách thủ công thì chúng ta sử dụng môi trường phát triển và tích hợp Visual Studio.Net bằng cách:

- Bước 1: Khởi động visual Studio 2010
- Bước 2: Vào File > New > Project
- Bước 3: Khai báo



Hình 4.1. Ví dụ cách tạo Windows Form bằng IDE

Mở hộp ToolBox: Menu View > ToolBox > chứa các control

Mở cửa sổ Properties: Menu View > Properties > chứa thuộc tính

Mở cửa sổ Solution Explorer: Menu View > Solution Explorer > cửa sổ Project xuất hiện

- Bước 4: Thiết kế Form – Viết Code

Thiết kế Form: nhấp vào View Designer (trong cửa sổ Solution Explorer)

Viết Code: nhấp vào View Code (trong cửa sổ Solution Explorer)



Hình 4.2. Windows Form đơn giản

Để đặt một đối tượng điều khiển trên Form, bạn cần phải thực hiện các bước sau:

Bước 1. Tạo một đối tượng của lớp điều khiển

Bước 2. Cấu hình các thuộc tính của điều khiển (đặc biệt là kích thước và tọa độ vị trí).

Bước 3. Đặt điều khiển vào Form hoặc điều khiển chứa

Bước 4. Xử lý các sự kiện trên các điều khiển

4.1.2. Tạo đối tượng của lớp điều khiển

Để sử dụng các điều khiển, chúng ta cần phải tạo một đối tượng của lớp điều khiển. Ở đây, các điều khiển được thừa kế từ lớp System.Windows.Forms.Control. Hãy xét ví dụ sau:

Ví dụ:

```
this.label1 = new System.Windows.Forms.Label();
```

Khi đã tạo đối tượng điều khiển, bạn có thể thiết lập giá trị các thuộc tính của nó:

```
this.label1 = new System.Windows.Forms.Label();  
this.label1.Location = new System.Drawing.Point(28, 55);  
this.label1.Name = "lbHello";  
this.label1.Size = new System.Drawing.Size(100, 20);  
this.label1.TabIndex = 1;
```

4.1.3. Tập các đối tượng điều khiển con (Controls collection)

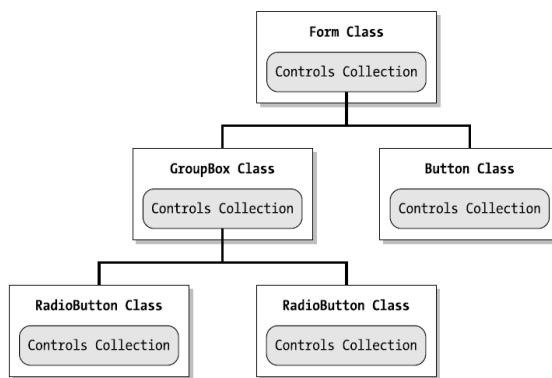
Một điều khiển có thể chứa một hoặc là nhiều các điều khiển khác. Ở đây lớp System.Windows.Forms.Control cung cấp thuộc tính Controls tức là chúng ta có thể biểu diễn một tập các điều khiển con.

Hãy xét ví dụ sau: một Windows Form sử dụng thuộc tính Controls để lưu các điều khiển trực tiếp xuất hiện trên Form. Nếu bạn có điều khiển chứa các điều khiển khác trên Form, như Group boxes, nó sẽ có tập điều khiển con của chính nó.



Hình 4.3. Ví dụ Form chứa điều khiển nhiều cấp

Điều khiển cấp cao nhất chứa các điều khiển khác là Form, ta có cấu trúc các điều khiển trên Form như sau:



Hình 4.4. Cấu trúc các điều khiển trên Form.

4.1.4. Đặt, xóa một đối tượng điều khiển trên Form hay điều khiển khác

Giống như hầu hết các lớp collection khác, tập điều khiển Controls của Form cung cấp một số phương thức chuẩn như Add() and Remove().

Ví dụ: Đặt đối tượng điều khiển Label label1 trên Form. Nhãn (Label) sẽ xuất hiện trên đối tượng Form hiện hành:

```
this.Controls.Add(this.label1);
```

Nếu bạn muốn đối tượng Label label1 đặt trong một đối tượng tấm nền (Panel) panel1, bạn làm như sau:

```
this.Controls.Add(this.panel1);  
this.panel1.Controls.Add(this.label1);
```

Một điều khiển có thể xóa khỏi form sử dụng phương thức Remove() của tập điều khiển con Controls:

```
this.panel1.Controls.Remove (this.label1);
```

4.1.5. Thừa kế lớp Form

Khi chúng ta sử dụng Visual Studio để tạo một đối tượng Form mới của lớp System.Windows.Forms.Form chương trình sẽ tự động tạo ra một lớp kế thừa lớp Form ví dụ như Form Hello. Vì vậy, chúng ta có thể kế thừa tất cả các đặc tính của lớp Form để bổ sung cho các đặc tính cho lớp đó.

4.1.6. Truy cập các đối tượng điều khiển

Khi đối tượng Form được khởi tạo thì chúng ta có hai phương pháp sau để có thể truy cập các điều khiển trên form đó:

- Thông qua tập các điều khiển con (Controls collection)
- Sử dụng các biến thành viên của lớp Form mà bạn tạo

Đối với một lớp Form chứa một Label được tham chiếu với biến thành viên label1, chúng ta có thể dễ dàng truy cập nó trong các phương thức của lớp Form sử dụng bằng cách sau:

```
label1.Text = "Hello Form";
```

Mặc định, tất cả các đối tượng điều khiển trong C# được khai báo Private, vì vậy chúng ta không thể truy cập được trong lớp khác. Tuy nhiên, ở đây chúng ta có thể truy cập bất kỳ hiểu biết thông qua tập điều khiển con Controls, và nó luôn luôn ở trạng thái Public.

Tuy nhiên, các điều khiển được đánh chỉ số trong tập điều khiển, không sử dụng tên. Vì vậy để tìm một điều khiển, bạn cần sử dụng thuộc tính Name của điều khiển, được tự động thiết lập khi bạn khai báo tạo một đối tượng điều khiển trong lớp Form của bạn.

```
label1.Name = "label1";
```

Mã tìm một điều khiển trong tập điều khiển con như sau:

```
foreach (Control ctrl in Controls)
{
    if (ctrl.Name == "label1")
    {
        Controls.Remove(ctrl);
    }
}
```

4.1.7. Thành phần (Components)

Trong .NET Framework, components là một lớp thực hiện giao diện từ giao diện System.ComponentModel.IComponent, trực tiếp hoặc gián tiếp từ một lớp đã thực hiện Component.

Không giống như điều khiển, Components không chiếm vị trí trên Form, nó thường được sử dụng cho một đối tượng là các đối tượng tái sử dụng và có thể tương tác với các đối tượng khác.

Ví dụ, .NET bao gồm các Components có thể hiển thị cửa sổ trợ giúp, thông báo lỗi, biểu tượng hệ thống, hay một hộp thoại chuẩn. Loại Components khác không có biểu diễn trực quan, và chỉ biểu diễn một tính năng nào đó. (Ví dụ lớp Timer và SqlConnection). Tuy nhiên, các Components cũng giống như các điều khiển có thể kết hợp vào Form và cấu hình khi thiết kế

4.2. Lớp Control

Control là lớp (class) các thành phần được thêm vào Windows Forms để tương tác giữa người sử dụng với Windows.

Có rất nhiều loại Control trong Windows Forms như: Label, TextBox, ListBox, Combobox, Button,...

Các control trong Windows Forms dùng namespace System.Windows.Forms.

4.2.1. Các thành viên cơ sở của lớp Control

Thuộc tính là những thông tin mà ta có thể thay đổi nội dung, cách trình bày... của người thiết kế để ứng dụng vào control

Mỗi lớp (class) có nhiều thuộc tính khác nhau, tuy nhiên vẫn có nhiều thuộc tính giống nhau được xây dựng từ lớp баз đầu gọi là các thành viên cơ sở của lớp Control

Thành viên	Mô tả
Name	Tên đại diện của control đó. Nó như một tên biến rất quan trọng
Anchor	Cố định control này khi thay đổi kích thước Form
Dock	Gắn giống với Anchor nhưng nó chiếm toàn bộ phần được đặt
BackColor	Màu nền của control đó
BackgroundImage	Hình nền của control đó
ContextMenuStrip	Menu khi ấn chuột phải lên control
Cursor	Hình con chuột khi rê lên control
Font	Chứa các thuộc tính về màu, cỡ, kiểu chữ mô tả
Location	Vị trí của Form đó trên control
Tag	Là nhãn phân biệt giữa các control giống nhau trong cùng 1 Form
TabIndex	Thứ tự focus khi nhấn phím Tab (trên bàn phím) của control so với các control khác cùng nằm trong control chứa nó
TagStop	Chỉ định control có được phép “bắt” (True) / không được phép bắt “False” phím Tab. Nếu không được phép thì TabIndex cũng không dùng được
Text	Nội dung hiển thị trong Control
Top	Là khoảng cách theo chiều dọc từ cạnh trên của Control đến

	cạnh trên của Control chứa nó
Enabled	Cho phép điều khiển đáp ứng sự kiện (true) hay không (false)
Visible	Cho phép Control hiện “true” hay không hiện “False” khi chạy ứng dụng
Width	Là chiều rộng của Control tính từ cạnh trái của Control đến cạnh phải của Control
Dispose()	Phương thức này giải phóng tài nguyên sử dụng bởi điều khiển. Bạn có thể gọi phương thức thủ công để hủy điều khiển trong bộ nhớ hay để cho bộ thu gom rác tự động hủy. Khi bạn gọi Dispose() trên đối tượng chứa, tất cả các điều khiển con cũng bị hủy

4.2.2. Phím truy cập

Một số điều khiển như nút lệnh (button), nhãn (label) và mục menu cho phép một ký tự trong chuỗi hiển thị trên điều khiển (caption) được gạch chân và sử dụng như một phím truy cập. Chẳng hạn button có caption là Save. Nếu người dùng nhấn Alt - S, button được tự động click. Để cấu hình phím tắt này, cần bổ sung ký tự & trước ký tự đặc biệt đó, như “&Save”

4.2.3. Focus và thứ tự Tab

Thuộc tính TabIndex xác định thứ tự chuyển focus bởi phím Tab, điều khiển có TabIndex là 0 sẽ nhận focus đầu tiên. Trong Visual Studio, bạn có thể chọn View/ Tab Order để thiết lập thứ tự nhận focus bởi phím Tab

Các điều khiển không nhìn thấy (thuộc tính Visible là false) hay không có khả năng đáp ứng sự kiện (thuộc tính Enabled là false) sẽ không nhận focus

Thành viên	Mô tả
TabIndex	Thứ tự điều khiển trên Form khi ấn phím Tab
Focused	Trả về true nếu điều khiển nhận focus
ContainsFocus	Trả về true nếu điều khiển hay điều khiển con nhận focus
Focus()	Chuyển focus đến điều khiển, trả về true nếu chuyển thành công
SelectNextControl()	Chuyển focus đến điều khiển kế tiếp, trả về true nếu chuyển thành công

GetNextControl(Control ctrl, bool forward)	Trả về điều khiển kế tiếp (nếu forward là true) hay kế trước (forward là false) điều khiển đang nhận focus
--	--

4.3. Xử lý sự kiện

4.3.1. Sự kiện là gì

Ta cứ tưởng tượng khi chúng ta nhảy, thì đó là sự kiện ghi nhận hành động nhảy. Con chim hót thì là sự kiện chim hót

Trong Winform C# sự kiện là một hành động gì đó vừa được xảy ra. Ví dụ sự kiện đơn giản nhất nếu người dùng ấn vào một nút nào đó, thì đó là sự kiện Click hoặc ấn phím nào đó là sự kiện KeyPress

Chúng ta cứ tưởng tượng sự kiện như là một câu lệnh điều kiện if, nếu xảy ra sự kiện gì đó thì ta sẽ xử lý để làm việc gì đó. Ví dụ như kích chuột vào button đăng nhập thì nó sẽ kiểm tra tài khoản và mật khẩu.

Người dùng có thể khởi chạy các sự kiện bằng cách tương tác với chương trình ví dụ việc nhập một button sẽ khởi chạy event Click của button. Các control có thể tự chạy sự kiện riêng của mình và đồng thời chúng ta có thể khởi chạy các sự kiện bằng cách gọi chúng như một phương thức.

C# hỗ trợ cho chúng ta một bộ lắng nghe sự kiện EventHandler, đơn giản nó sẽ dựa trên cơ chế ngắt của vi xử lý gửi và nhận các tín hiệu và trả về các phương thức.

4.3.2. Các sự kiện thường dùng

Bảng sự kiện bàn phím của lớp Control

Sự kiện	Mô tả
KeyDown	Bắt đầu ấn phím
KeyUp	Đã ấn phím xong
KeyPress	Trong khi ấn phím

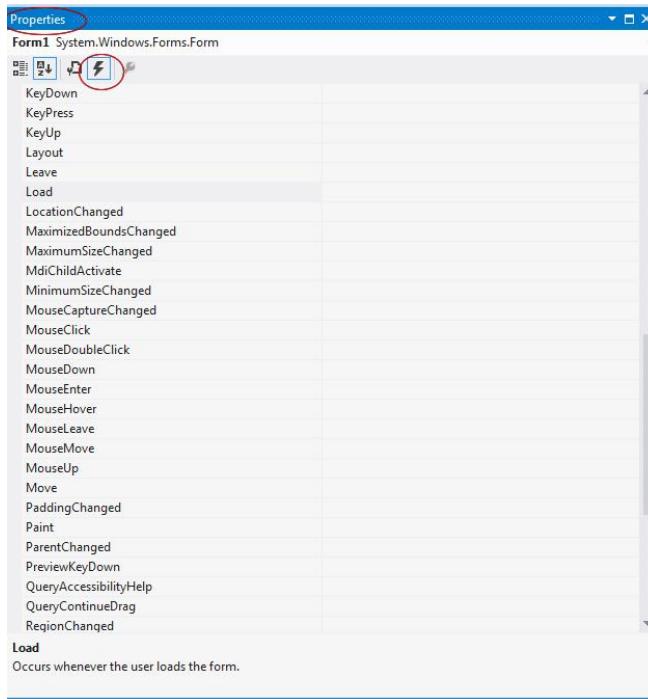
Bảng sự kiện mouse của lớp Control

Sự kiện	Mô tả
Click	Ấn vào control đó
DoubleClick	Nháy kép vào control đó

MouseDown	Ấn chuột
MouseUp	Thả chuột
MouseHover	Rê chuột qua control
MouseLeave	Rê chuột khỏi control

4.3.3. Thêm một sự kiện vào phương thức

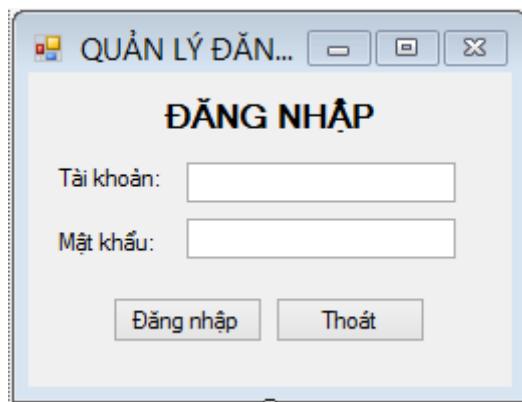
Chúng ta có thể thêm sự kiện bằng code, hoặc click đúp vào control muốn thêm để thêm sự kiện thường dùng của nó, hoặc đúp chuột vào tên sự kiện cần thêm trong khung sự kiện của ô Properties khi đang thiết kế form. Khi đó Visual Studio sẽ tự động generate code cho sự kiện đó vào <YourForm>.Designer.cs và gọi hàm đó vào class <YourForm> để các bạn thiết lập xử lý sự kiện



Hình 4.5. Cách thêm sự kiện bằng khung Properties

4.4. Form

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

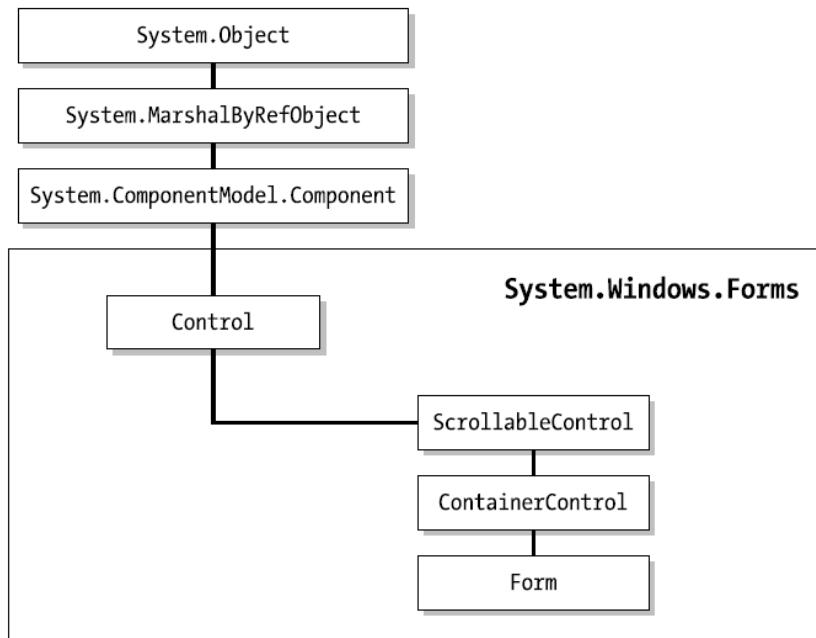


Hình 4.6. Form đăng nhập đơn giản

Form trên có chứa các Textbox, cho phép chúng ta nhập dữ liệu tài khoản và mật khẩu. Cũng trên form này còn có 2 nút Button đăng nhập và thoát. Button đăng nhập này thực thi nhiệm vụ mở hệ thống nếu tài khoản và mật khẩu trùng khớp và Button thoát để thoát khỏi form đăng nhập.

4.4.1. Form và các thành viên cơ sở của lớp Form

Mỗi form đều được kế thừa từ lớp form thuộc namespace System.Windows.Forms của hệ thống. Lớp form có nguồn gốc từ các lớp trong cùng một namespace. Vì vậy, tất cả các form đều được thừa kế các thuộc tính, phương thức và sự kiện liên quan đến các cơ sở của nó. Ta có bảng như sau



Hình 4.7. Form và các thành viên của lớp Form

Thuộc tính của Form được sử dụng để xác định cách hiển thị tại thời điểm chạy. Mỗi form đều có rất nhiều thuộc tính khác nhau, chẳng hạn như kích thước, màu nền, vị trí...

Các thuộc tính thường được sử dụng của một form được mô tả trong bảng sau:

Thuộc tính	Mô tả
Name	Tên của Form. Mặc định tên form đầu tiên là Form1
BackColor	Sử dụng để xác định màu nền của một form
BackgroundImage	Sử dụng để xác định hình nền của form
Font	Xác định kiểu chữ, cỡ chữ, phông chữ của văn bản của các điều khiển trên Form
Size	Kích thước chiều cao và rộng của form
StartPosition	Vị trí form trên màn hình Manual: vị trí và kích thước của form mà nó xác định khi chạy WindowsDefaultLocation: form hiển thị tại vị trí mặc định và có kích thước xác định bởi thuộc tính Size WindowsDefaultBounds: form hiển thị tại ví trí mặc định và có kích thước form xác định bởi hệ điều hành Windows CenterParent: hiển thị form ở trung tâm của form chứa nó
Text	Văn bản hiển thị dùng để xác định các chú thích trên thanh tiêu đề của Form
WindowState	Xác định trạng thái hiển thị cửa sổ dạng bình thường (Normal), cực đại (Maximized), hay cực tiểu (Minimized)

Bảng các sự kiện của lớp Form

Sự kiện	Mô tả
Load	Xảy ra khi form được nạp đầu tiên vào bộ nhớ. Sự kiện này sử dụng để khởi tạo các biến của form, cấu hình các điều khiển
FormClosing	Xảy ra khi form đang đóng
FormClosed	Xảy ra khi form đã đóng
Click	Xảy ra khi người dùng kích trái mouse bất kỳ vị trí trên

	form
MouseMove	Xảy ra khi mouse di chuyển trên form
MouseDown	Xảy ra khi nhấn nút mouse trái trên form
MouseUp	Xảy ra khi nhả nút mouse sau khi kích

Bảng các phương thức của lớp Form

Phương thức	Mô tả
Show()	Được dùng để hiển thị form
ShowDialog()	Hiển thị form như một hộp thoại, nghĩa là phải đóng form mới có thể thực hiện thao tác trên form khác
Hide()	Dùng để ẩn form
Activate()	Kích hoạt form và thiết lập focus vào nó, form sẽ hiển thị trên cùng của ứng dụng và thanh tiêu đề sáng lên
Close()	Dùng để đóng form
SetDesktopLocation(int x, int y)	Thiết lập vị trí của form theo tọa độ màn hình trong lúc chạy. Phương thức này lấy tọa độ X và Y màn hình như các thông số của nó

4.4.2. Thiết lập vị trí Form

Lớp Form cung cấp những thuộc tính Location (vị trí) và Size kích thước mà mỗi control đều có. Thuộc tính Location cho biết khoảng cách ở góc top left của cửa sổ so với góc top left của màn hình (hoặc desktop area) trôi đi. Ngoài ra, Location sẽ bị bỏ qua trừ khi thuộc tính StartPosition được cho về Manual. Các trị của thuộc tính StartPosition sẽ lấy từ enum FormStartPosition, như dưới đây:

Thuộc tính StartPosition sử dụng để xác định vị trí của form trên màn hình. Thuộc tính này có thể có các giá trị sau:

- Manual: vị trí và kích thước của form quyết định vị trí bắt đầu của form
- CenterScreen: form sẽ được canh giữa màn hình
- WindowsDefaultLocation: form hiển thị tại vị trí mặc định và vị trí của form xác định bởi kích thước của form. Nói cách khác sẽ không biết biểu mẫu sẽ được canh về đâu.

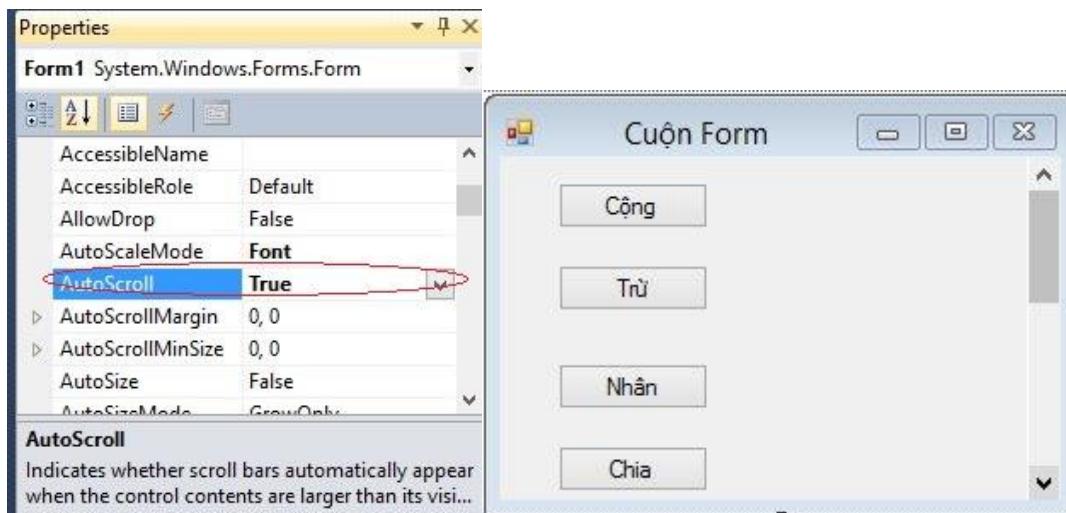
- WindowsDefaultBounds: form hiển thị tại vị trí mặc định và vị trí form xác định bởi hệ điều hành Windows

Ví dụ: đặt form giữa màn hình khi nạp form lần đầu tiên vào bộ nhớ

```
private void form1_Load(System.Object sender, System.EventArgs e)
{
    Screen scr = Screen.PrimaryScreen;
    this.Left = (scr.WorkingArea.Width - this.Width) / 2;
    this.Top = (scr.WorkingArea.Height - this.Height) / 2;
}
```

4.4.3. Cuộn Form

Thiết lập thuộc tính AutoScroll về true để cuộn form.



Hình 4.8. Cách cuộn form

4.4.4. Tạo cửa sổ hộp thoại



Hình 4.9. Hộp thoại đơn giản

Để tạo cửa sổ hộp thoại như trên ta thực hiện như sau:

```
public partial class DialogForm : System.Windows.Forms.Form
{
    public enum SelectionTypes
    {
        OK,
        Cancel
    }

    public SelectionTypes UserSelection;

    private void cmdOK_Click(object sender, EventArgs e)
    {
        UserSelection = SelectionTypes.OK;
        this.Close();
    }

    private void cmdCancel_Click(object sender, EventArgs e)
    {
        UserSelection = SelectionTypes.Cancel;
        this.Close();
    }
}

DialogForm frmDialog = new DialogForm();
frmDialog.ShowDialog();

switch (frmDialog.UserSelection)
{
    case DialogForm.SelectionTypes.OK:
        // Thực hiện thao tác khi kích OK
```

```
        break;

        case DialogResult.SelectionTypes.Cancel:
            // Thực hiện thao tác khi kích Cancel
            break;

    }

frmDialog.Dispose();
```

4.4.5. Form chủ (form ownership)

.NET cho phép một form làm chủ các form khác. Hãy xem xét một ví dụ nếu form 1 là form chủ của form 2, thì nếu form 1 được đóng lại thì form 2 cũng được đóng. Và khi form con nằm chồng lên form chủ thì form con luôn nằm phía trên

Ta có bảng các thành viên xác định Form chủ

Thành viên	Mô tả
Owner	Xác định form chủ của form này
OwnedForms	Cung cấp một mảng các form con của form hiện hành. Mảng này là chỉ đọc
AddOwnedForm() and RemovedOwnedForm()	Thêm và xóa form con khỏi form chủ

4.4.6. Làm việc với ứng dụng MDI

Có 2 loại ứng dụng là SDI (Single Document Interface) và MDI (Multiple Document Interface).

SDI sử dụng giao diện một tài liệu (ví dụ WordPad) và nó có thể làm việc chỉ với một cửa sổ trong một thời điểm

MDI bao gồm hai phần, form cha và form con (ví dụ Microsoft Word). Nó có thể chứa nhiều form cha và mỗi form cha có nhiều form con. Chúng ta có thể tạo MDI lúc thiết kế bằng cách thiết lập thuộc tính IsMdiContainer của form là true



Hình 4.10. Cách tạo ứng dụng MDI

4.4.7. Form định nghĩa trước

4.4.7.1. Hộp thoại thông báo đơn giản

Winform hỗ trợ một số dạng hộp thoại xây dựng sẵn

Ví dụ



Hình 4.11. Hộp thoại thông báo đơn giản

Ta sử dụng MessageBox, hiển thị một hộp thoại thông báo đơn giản

```
DialogResult thoat = new DialogResult();  
  
thoat = MessageBox.Show("Bạn có muốn thoát hay không",  
"Thông báo", MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
```

Ngoài ra ta có một số kiểu Button sử dụng cho phương thức như: AbortRetryIgnore, OK, OKCancel, RetryCancel, YesNo, YesNoCancel

Và một số Icon biểu tượng cho hộp thoại như: Asterisk hay Information, Error, Hand, hay Stop, Exclamation hay Warning, None, Question

Ngoài ra .NET còn cung cấp các hộp thoại khác như font chữ, màu, hộp thoại lưu mở tập tin, và máy in. Các lớp này kế thừa từ lớp System.Windows.Forms.CommonDialog

4.4.7.2. Hộp thoại phông chữ và màu

Ví dụ: Hiển thị hộp thoại màu

```
ColorDialog colorDialog = new ColorDialog();  
if (colorDialog.ShowDialog() == DialogResult.OK)  
    shape.ForeColor = colorDialog.Color;
```

Lớp ColorDialog cung cấp thuộc tính:

- AllowFullOpen, nếu là false ngăn cản người dùng chọn màu tùy ý
- ShowHelp nếu là true cho phép người dùng nhấn F1 để hiển thị trợ giúp

4.4.7.3. Hộp thoại lưu và mở tập tin

Để sử dụng hộp thoại mở tập tin chúng ta sử dụng điều khiển OpenFileDialog và lưu tập tin chúng ta sử dụng điều khiển SaveFileDialog.

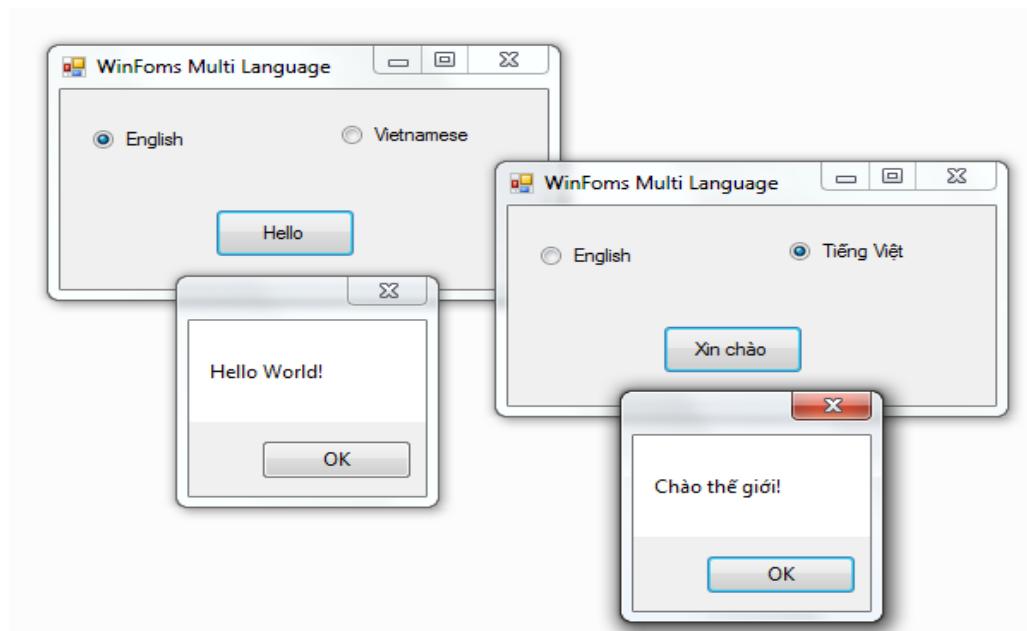
4.4.7.4. Hộp thoại in

Để sử dụng hộp thoại in chúng ta bổ sung điều khiển PrintDialog và điều khiển PrintDocument vào form hay tạo đối tượng của lớp PrintDialog và PrintDocument. Thiết lập thuộc tính Document của đối tượng PrintDialog là đối tượng của lớp PrintDocument

4.4.8. Form đa ngôn ngữ

Form đa ngôn ngữ tức là form này có thể được triển khai ở nhiều ngôn ngữ khác nhau. Để tạo một ứng dụng đa ngôn ngữ, cách điển hình là tạo các string table để lưu trữ các văn bản, chuỗi cho mỗi ngôn ngữ. Trong .NET, bạn có thể dùng các Resource để tạo ra các string table này và dùng lớp ResourceManager để quản lý và lấy ra các chuỗi cụ thể dựa vào một đối tượng CultureInfo.

Bạn có thể tạo nhiều tập tin resource cho nhiều quốc gia khác nhau rồi biên dịch chúng thành Assembly. Khi chạy ứng dụng, .NET sẽ tự động sử dụng đúng Assembly dựa trên các thiết lập bản địa (locale setting) của máy tính hiện hành.



Hình 4.12. Form đa ngôn ngữ

Bạn có thể đọc và ghi các tập tin tài nguyên bằng mã lệnh. Tuy nhiên, Visual Studio .NET cũng hỗ trợ việc thiết kế các form được bản địa hóa:

1. Trước tiên, thiết lập thuộc tính Localizable của form là true trong cửa sổ Properties.
2. Thiết lập thuộc tính Language của form là quốc gia bạn muốn nhập thông tin cho nó. Ké đó, cấu hình các thuộc tính có thể bản địa hóa của tất cả các điều kiém trên form. Thay vì lưu trữ những thay đổi này trong phần mã thiết kế form, Visual Studio .NET tạo một tập tin tài nguyên mới để lưu trữ dữ liệu của bạn.
3. Lặp lại bước 2 cho mỗi ngôn ngữ bạn muốn hỗ trợ. Mỗi lần như thế, một tập tin tài nguyên mới sẽ được tạo ra. Nếu bạn thay đổi thuộc tính Language thành quốc gia mà bạn đã cấu hình thì các thiết lập trước đó sẽ xuất hiện trở lại, và bạn có thể chỉnh sửa chúng. Bây giờ, bạn có thể biên dịch và thử nghiệm ứng dụng trên các hệ thống quốc gia khác nhau. Bạn cũng có thể

buộc ứng dụng chấp nhận một quốc gia cụ thể bằng cách thay đổi thuộc tính Thread.CurrentCulture. Tuy nhiên, bạn phải thay đổi thuộc tính này trước khi form được nạp.

4.5. Các điều khiển cơ bản

Trong phần này chúng ta sẽ tìm hiểu về các điều khiển cơ bản như nhãn (Label), LinkLabel), hộp văn bản (TextBox), nút lệnh (Button), hộp đánh dấu (CheckBox), nút lựa chọn (RadioButton), hình ảnh (PictureBox) và hộp danh sách (ListBox, CheckedListBox, ComboBox) và điều khiển nhóm (GroupBox)

4.5.1. Nhãn (Label)

Trong form, điều khiển Label cho phép ta đặc văn bản miêu tả, Label thường được đặt trước các điều khiển khác với văn bản mô tả cách sử dụng điều khiển.

Để tạo Label chúng ta sẽ kéo và thả Label lên form, gán văn bản mô tả cho nó thông qua thuộc tính text.

Bảng Thuộc tính của lớp Label

Thuộc tính	Mô tả
AutoSize	Điều khiển kích thước đổi tương cho vừa với chiều dài chuỗi ký tự
Font Name	Quy định Font chữ cho văn bản
Bold	Nếu là True: in đậm, nếu là False: bỏ in đậm
Italic	Nếu là True: in nghiêng, nếu là False: bỏ nghiêng
Size	Quy định cỡ chữ cho văn bản
Underline	Nếu là True: gạch dưới, nếu là False: bỏ gạch dưới
TextAlign	Canh lề

4.5.2. Nhãn liên kết (LinkLabel)

Tương tự như Label, LinkLabel là một nhãn đặc biệt kế thừa từ lớp Label, và bổ sung một số thuộc tính của nhãn biểu diễn dạng liên kết. Ví dụ, ứng dụng có thể cung cấp một liên kết đến website công ty trong cửa sổ About.

Thuộc tính LinkArea xác định phạm vi chuỗi hiển thị như một liên kết bao gồm vị trí ký tự đầu tiên và chiều dài liên kết. Phụ thuộc vào thuộc tính LinkBehavior, nhãn liên

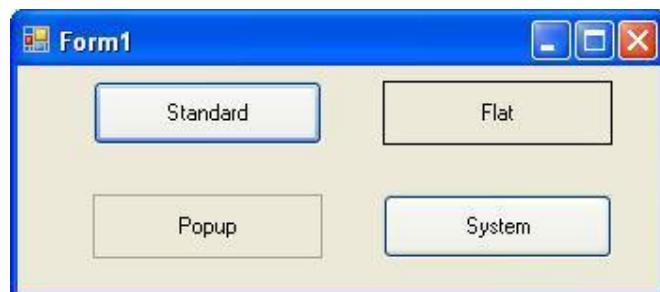
kết có thể gạch chân (AlwaysUnderline) hoặc không gạch chân (NeverUnderline), gạch chân khi di chuột đến nhän (HoverUnderline) hoặc có giá trị mặc định (SystemDefault)

Các thuộc tính khác của lớp LinkLabel

Thuộc tính	Mô tả
ActiveLinkColor	Màu của liên kết đang hoạt động
DisabledLinkColor	Màu của liên kết không thể chọn
LinkColor	Màu của liên kết
VisitedLinkColor	Màu của liên kết đã thăm
LinkVisited	Nếu là true, liên kết xuất hiện với màu liên kết đã thăm

4.5.3. Nút lệnh (Button)

Control Button là một trong những control được sử dụng nhiều nhất trong các ứng dụng Windows. Sự kiện Click là sự kiện mà chúng ta thường xuyên sử dụng khi lập trình với control này. Lớp Button kế thừa hai thuộc tính quan trọng từ lớp ButtonBase, đó là thuộc tính Flatstyle và thuộc tính Image. Thuộc tính Flatstyle có bốn tùy chọn: Flat, popup, Standard (mặc định) và System. Hình minh họa sau sẽ thể hiện bốn tùy chọn trên.



Hình 4.13. Ví dụ về thuộc tính Button

4.5.4. Hộp văn bản (Textbox)

Textbox thường được sử dụng để nhập thông tin người dùng ở dạng văn bản. Trong control Textbox chương trình có thể cho phép chỉnh sửa dữ liệu (hoặc không), cũng như cho phép cắt, dán, copy như những chương trình soạn thảo cơ bản trong Windows.

Bảng các thành viên khác của lớp TextBox

Thành viên	Mô tả
Multiline	Thuộc tính này cho phép chúng ta thiết lập là true hoặc false. Nếu là true textbox sẽ chứa được nhiều dòng, ngược lại textbox chỉ có một dòng. Mặc định của thuộc tính này là false.
Passwordchar	Định nghĩa kí tự mặt nạ hiển thị các kí tự do người dùng nhập vào. Thuộc tính này chỉ áp dụng cho textbox một dòng (Multiline = false) và nó chỉ ảnh hưởng đến hình thức trình bày mà không ảnh hưởng đến giá trị nhập vào textbox.
ScrollBars:	Thuộc tính này chỉ áp dụng cho textbox dạng Multiline, nó xác định xem thanh cuộn có xuất hiện trên textbox hay không. C#.Net hỗ trợ bốn tùy chọn cho thuộc tính này: None (mặc định, không có thanh cuộn), Horizontal (thanh cuộn ngang), Vertical (thanh cuộn dọc) và Both (có cả thanh cuộn ngang và dọc).
TextAlign	Thuộc tính này xác định cách thức canh lề cho textbox. C#.Net hỗ trợ ba tùy chọn cho thuộc tính này: Center (canh giữa), Left (canh trái) và Right (canh phải).

4.5.5. Hộp đánh dấu (CheckBox) và nút lựa chọn (RadioButton)

Trong một form, control Checkbox cho phép người dùng chọn nhiều tùy chọn cùng lúc (ngược lại với RadioButton). Thuộc tính quan trọng nhất của CheckBox là Checked, nếu thuộc tính này là True thì CheckBox được chọn, ngược lại là không được chọn.

Nút Radio cho phép người dùng chọn một tùy chọn từ một tập hợp những tùy chọn. Ta chỉ có thể chọn được một trong danh sách các mục chọn. Thông thường chương trình ít khi trả lời sự kiện của control RadioButton. Thay vào đó, chương trình sẽ kiểm tra giá trị của chúng (thông qua thuộc tính Checked) khi bắt đầu thao tác xử lý.

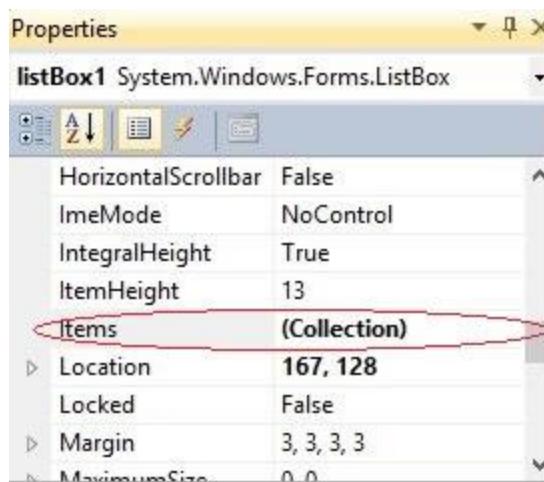
4.5.6. PictureBox

PictureBox có chức năng điều khiển hình ảnh, ta sử dụng thuộc tính Image để hiển thị hình ảnh mong muốn. Nó có các thuộc tính như SizeMode thiết lập giá trị chọn từ enumeration PictureBoxSizeMode như StretchImage

4.5.7. Các điều khiển danh sách (List Controls)

Các điều khiển danh sách kế thừa từ lớp trừu tượng ListControl bao gồm ba điều khiển danh sách: ListBox, CheckedListBox và ComboBox. Chúng cung cấp các tính năng cơ sở cho phép sử dụng điều khiển danh sách để ràng buộc dữ liệu (data binding).

Control Listbox được sử dụng để hiển thị một danh sách các mục cho người sử dụng chọn. Người dùng có thể chọn một mục từ danh sách. Chúng ta có thể thêm các mục cho Control này bằng cách sử dụng cửa sổ Properties chọn mục Items như hình vẽ:



Hình 4.14. Thêm Items trong ListBox

Một cửa sổ hiện ra chúng ta thêm các mục để chọn lựa. Hoặc chúng ta có thể thêm các mục này trong phần mã lệnh ở sự kiện Load:

```
listBox1.Items.Add("Nam");
listBox1.Items.Add("Nữ");
listBox1.Items.Add("Giới tính khác");
```

Bảng các thuộc tính thường được sử dụng của Listbox được mô tả trong bảng sau đây

Thành viên	Mô tả
MultiColumn	Xác định ListBox có một hay nhiều cột. Nếu thuộc tính này bằng True thì ListBox có nhiều cột, ngược lại chỉ có một cột.
IntegralHeight	Nếu là true, chiều cao tự động điều chỉnh, để nhiều hàng, nhưng không có hàng không nhìn thấy
Items	Tập các mục trong list control
SelectedIndex, SelectedIndices, SelectedItem, SelectedItems, and Text	Mục chọn (Text) và chỉ số mục chọn hiện hành
Text	Trả về tiêu đề của phần tử được chọn
Sorted	Nếu là true, các mục tự động sắp xếp

Combobox gần giống với Control ListBox, nó cho phép người dùng chọn một mục từ danh sách cho trước và cho phép người dùng nhập thông tin mới vào. Cũng tương tự như ListBox, Combobox cho phép người dùng thêm các mục bằng hai cách. Để sử dụng combobox, tương tự như các control khác, đơn giản ta chỉ cần kéo và thả control Combobox lên form.

Bảng các thuộc tính của ComboBox

Thành viên	Mô tả
Text	Cho ta biết phần tử mà ta vừa chọn
Items	Trả về đối tượng ComboBox.ObjectCollection
Add	Thêm một phần tử item vào trong ComboBox
Clear	Xóa tất cả các phần tử trong ComboBox.
Remove	Xóa phần tử value trong ComboBox.
RemoveAt	Xóa một phần tử trong ComboBox tại vị trí index.

CheckedListBox được sử dụng để hiển thị một ListBox với những CheckBox được hiển thị bên trái của đối tượng

Bảng các thuộc tính của CheckedListBox

Thành viên	Mô tả
CheckedIndices	Tập hợp các thứ tự CheckBox đã chọn trong CheckedListBox này
CheckedItems	Tập hợp các mục đã chọn trong CheckedListBox này

CheckedOnClick	Sử dụng để nhận hoặc thiết lập giá trị để chắc chắn rằng các CheckBox sẽ được chuyển đổi thuộc tính khi được thiết lập
----------------	--

4.5.8. Điều khiển font

Domain controls hạn chế người dùng vào chỉ trong một tập giá trị đúng. ListBox chuẩn là một ví dụ của domain control, bởi vì một người dùng có thể chọn chỉ một mục trong danh sách

Nó bao gồm các điều khiển như

DomainUpDown tương tự với ListBox đều cung cấp danh sách các tùy chọn. Sự khác nhau là người dùng có thể duyệt danh sách chỉ sử dụng phím mũi tên (up/down arrow buttons). Để sử dụng điều khiển DomainUpDown, bổ sung mục vào tập Items. Thuộc tính Text hay SelectedIndex trả về mục chọn hay chỉ số mục chọn

NumericUpDown cho phép người dùng chọn một giá trị số bằng cách sử dụng phím mũi tên hay gõ trực tiếp.

Track bar cho phép người dùng chọn giá trị bằng cách di chuyển biểu tượng mũi tên theo thanh dọc hay ngang

Thanh tiến trình (ProgressBar) hoàn toàn khác với các domain controls khác bởi vì nó không cho phép người dùng chọn giá trị. Thay vì vậy, bạn có thể sử dụng thanh tiến trình để cung cấp phản hồi về tiến trình thực hiện nhiệm vụ.

4.6. Các điều khiển chuyên biệt

4.6.1. Điều khiển NotifyIcon

Điều khiển NotifyIcon sử dụng để tạo biểu tượng hiển thị trên MenuStrip, ToolStrip hay StatusStrip trong vùng chỉ định. Lớp này không thừa kế được

Các thuộc tính của điều khiển NotifyIcon mô tả như bảng sau:

- **BalloonTipIcon:** sử dụng để nhận hoặc thiết lập các biểu tượng để hiển thị trên phần ghi chú của balloon kết hợp với NotifyIcon

- **BalloonTipText:** sử dụng để nhận hoặc thiết lập nội dung hiển thị trên phần ghi chú của balloon kết hợp với NotifyIcon
- **BalloonTipTitle:** sử dụng để nhận hoặc thiết lập tiêu đề hiển thị trên phần ghi chú của balloon kết hợp với NotifyIcon.

4.6.2. Trình đơn (MenuStrip), thanh trạng thái (StatusStrip), thanh công cụ (ToolStrip)

Các menu được sử dụng hầu hết trong các ứng dụng Windows, chúng cung cấp một cách tuyệt vời để giao tiếp với người dùng với các tùy chọn để họ làm việc theo các chức năng có sẵn

Có 2 loại Menu:

- **Menu ngang (Menu Bar):** sử dụng điều khiển MenuStrip là đầu vào của một cửa sổ và thường bao gồm các mục như File, Edit, và Help.
- **Menu ngữ cảnh (Context menu):** sử dụng điều khiển ContextMenuStrip cho phép người sử dụng truy cập đến thông tin về các chủ đề hay mục đặc biệt.

4.6.2.1. *MenuStrip*

MenuStrip là phần cần thiết của ứng dụng giao diện đồ họa. Bổ sung điều khiển MenuStrip vào form.

Chúng ta có thể kéo lê mục menu đến vị trí mong muốn hoặc xóa mục menu bằng cách kích phải vào mục menu chọn Delete. Ngoài ra chúng ta cũng có thể sử dụng thanh phân chia để chia các mục menu thành các nhóm bằng cách kích phải trên mục menu, chọn Insert Separator đồng thời bổ sung biểu tượng đánh dấu trước văn bản trên mục menu, cho biết mục menu đang được chọn hay thôi chọn, bằng cách chọn mục menu, thiết lập thuộc tính Checked là true.

4.6.2.2. *Menu ngữ cảnh (Context Menu)*

Menu ngữ cảnh được ẩn cho đến khi người dùng nhấn chuột phải, sau đó menu được hiển thị tại vị trí con trỏ. Lớp Context Menu thì có thể thêm các menu theo ngữ

cánh cho một ứng dụng. Lớp này cũng chứa một tập hợp đối tượng các MenuItems, nhưng ContextMenu có thể xuất hiện trong bất cứ vị trí nào trong một form

Để bổ sung điều khiển ContextMenuStrip vào form. Có hai cách hiển thị Menu ngũ cảnh :

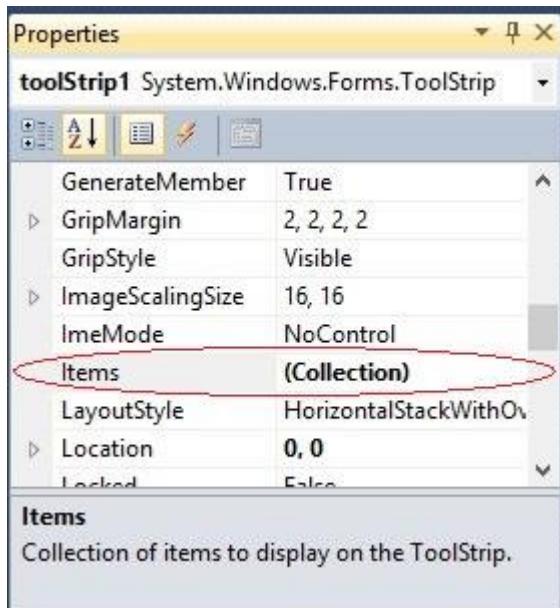
- Thiết lập thuộc tính ContextMenuStrip của Form1 là đối tượng contextMenuStrip1
- Hoặc viết hàm đáp ứng sự kiện kích phải mouse trên Form1 như sau, sẽ hiển thị menu ngũ cảnh tại vị trí kích phải mouse (cột x, dòng y)

```
private void Form1_MouseClick(object sender, MouseEventArgs e)
{
    if (e.Button == MouseButtons.Right)
        contextMenuStrip1.Show(this, e.X, e.Y);
}
```

4.6.2.3. Thanh công cụ (ToolStrip)

ToolStrip là điều khiển cho phép tạo thanh công cụ, nó cung cấp truy cập dễ dàng và nhanh chóng để các tùy chọn được sử dụng thường xuyên nhất. Bạn có thể đã nhận thấy tầm quan trọng của Thanh công cụ trong khi làm việc trong ứng dụng dựa trên cửa sổ. Bất cứ khi nào bạn muốn thay đổi các định dạng của văn bản bất kỳ, bạn sử dụng các biểu tượng có mặt trên thanh công cụ thay vì chọn các tùy chọn từ menu. Có thể bổ sung buttons dễ dàng đến thanh công cụ khi thiết kế hay chạy chương trình.

Để bổ sung buttons khi thiết kế, bạn cần bổ sung một điều khiển ToolStrip vào form sau đó từ cửa sổ Properties của ToolStrip, bạn vào mục thuộc tính, như trong hình sau:

**Hình 4.15. Cửa sổ thuộc tính của toolStrip**

Lưu ý: Điều khiển ToolStrip có thể chứa Button, Label, SplitButton, DropDownButton, Separator, ComboBox, TextBox, and ProgressBar controls

4.6.2.4. Thanh trạng thái (StatusStrip)

StatusStrip là bắt nguồn từ lớp System.Windows.Forms.ToolStrip. Đây là đối tượng chứa có thể chứa tất cả điều khiển khác gọi là đối tượng con. StatusStrip được sử dụng để hiển thị thông tin về các đối tượng trên form hiện hành, và cung cấp phản hồi tiến trình bất kỳ thao tác thực hiện trên form. Mặc định, StatusStrip nằm ở cuối của form.

Một số thuộc tính của StatusStrip mô tả như sau:

Thành viên	Mô tả
Items	Là tập hợp các đối tượng hiển thị trên StatusStrip
LayoutStyle	Chỉ định kiểu định dạng của StatusStrip
Dock	Định nghĩa biên của StatusStrip là biên của đối tượng chứa

Điều khiển StatusStrip chứa bốn điều khiển con mà không thể sử dụng độc lập. Các điều khiển này có thể được thêm vào StatusStrip khi cần thiết và chứa thuộc tính, phương thức và sự kiện tương tự điều khiển cha. Các điều khiển con của điều khiển StatusStrip là:

- StatusLabel
- ProgressBar
- DropDownButton
- SplitButton

4.6.2.5. Điều khiển StatusLabel

Điều khiển này sử dụng để hiển thị thông tin trạng thái, nó cũng được dùng để nhắc nhở người dùng nhập dữ liệu hợp lệ. StatusLabel có các thuộc tính khác nhau, chẳng hạn như các thuộc tính Text và thuộc tính Spring trong đó:

- Text property: thay đổi thông tin hiển thị trên điều khiển StatusLabel
- Spring: xác định điều khiển StatusLabel có dièn vào bất kỳ không gian còn lại trên điều khiển StatusStrip hay không

Ví dụ

```
StatusLabel1.Spring = true;  
StatusLabel1.Text = "Kết thúc quá trình";
```

4.6.2.6. Điều khiển ProgressBar

Điều khiển này biểu diễn thanh tiến trình Windows và sử dụng để hiển thị trạng thái thực hiện của bất kỳ nhiệm vụ. Các thuộc tính của điều khiển ProgressBar như sau:

- Minimum: xác định biên thấp nhất của dãy giá trị của thanh tiến trình
- Maximum: xác định biên cao nhất của dãy giá trị của thanh tiến trình
- Value: giá trị hiện hành của thanh tiến trình
- Step: xác định khoảng tăng của giá trị của thanh tiến trình khi phương thức PerformStep() được gọi.

4.6.2.7. Điều khiển DropDownButton

Điều khiển này hiển thị danh sách ToolStripDropDown cho người dùng có thể chọn mục duy nhất từ nhiều mục khác nhau. Nó thường được sử dụng khi các mục được hiển thị trên bộ điều khiển StatusStrip không có đủ vị trí cho tất cả.

Các thuộc tính của điều khiển DropDownButton:

- DisplayStyle: xác định hiển thị image hay text
- DoubleClickEnabled: xác định đáp ứng sự kiện kích đúp hay không
- DropDownItems: xác định ToolStripItem sẽ hiển thị khi mục được kích chọn

4.6.2.8. Điều khiển SplitButton

Điều khiển này là sự kết hợp của button chuẩn phía bên trái một nút thả xuống bên phải. Điều khiển này được sử dụng khi yêu cầu cả hai tính năng button chuẩn và hộp danh sách button.

Các thuộc tính của điều khiển SplitButton:

- DisplayStyle: xác định image hay text được hiển thị
- DoubleClickEnabled: chỉ rõ đáp ứng sự kiện kích đúp mouse hay không
- Padding: xác định khoảng trống bên trong mục

4.6.3. Các điều khiển ngày (Date Controls)

Hai điều khiển ngày C# cung cấp là DateTimePicker và MonthCalendar.



Hình 4.16. Hai loại điều khiển ngày giờ

4.6.3.1. The DateTimePicker

DateTimePicker cho phép người dùng chọn một ngày cụ thể. Người dùng có thể duyệt tháng, và năm để tìm kiếm ngày chỉ rõ. Một đặc điểm hữu ích của DateTimePicker là tự động tuân theo thiết lập dạng ngày của máy tính. Bạn có thể xác định định dạng ngày cho thuộc tính DateTimePicker.Format là yyyy/mm/dd hay dd/mm/yyyy tùy thuộc vào thiết lập dạng ngày. Bạn có thể gán chuỗi định dạng ngày tùy ý cho thuộc tính CustomFormat property.

Một số thuộc tính của DateTimePicker mô tả như sau:

Thành viên	Mô tả
Text	Trả về chuỗi ngày đã được định dạng
Value	Trả về đối tượng DateTime
Format	Định dạng hiển thị: Long, short, time, custom
CustomFormat	dd/mm/yyyy: Ngày/tháng/năm hh:mm:ss: Giờ:Phút:Giây
MinDate, MaxDate	Chỉ ra vùng ngày mà người sử dụng có thể chọn

4.6.3.2. MonthCalendar

Điều khiển MonthCalendar trông giống như DateTimePicker, chỉ khác là nó luôn hiển thị tháng, và không cho phép người dùng nhập ngày bằng cách gõ vào hộp nhập.

Một số thuộc tính của MonthCalendar mô tả như sau:

Thành viên	Mô tả
SelectionStart, SelectionEnd	Trả về các ngày đã chọn
SelectionRange	Cấu trúc chứa SelectionStart và SelectionEnd
MinDate, MaxDate, MaxSelectionCount	Ngày tối thiểu và tối đa được chọn trong MonthCalendar, và số ngày liên tục có thể chọn tại một thời điểm

4.6.4. Các điều khiển Container

.NET Framework định nghĩa các điều khiển được thiết kế để chứa các đối tượng khác.

4.6.4.1. Điều khiển nhóm (GroupBox)

Nếu form chứa hai hoặc nhiều nhóm lựa chọn (đặc biệt là lựa chọn RadioButton), ta nên đặt mỗi nhóm trong control GroupBox. Trong Visual Studio ta đơn giản kéo và thả control GroupBox lên trên form. Đặt tiêu đề (thuộc tính Text) cho GroupBox, sau đó đặt các control cùng nhóm vào trong GroupBox

Chúng ta cũng có thể thiết lập các thuộc tính Text tại thời điểm chạy như sau:

```
groupBox1.Text = "Chọn giới tính";
```

4.6.4.2. Tâm nền (Panel)

Điều khiển này là điều khiển chứa các điều khiển khác, hỗ trợ thanh cuộn. Thuộc tính thường dùng của đối tượng này là AutoScroll dùng để hiển thị hoặc không hiển thị thanh cuộn. Đây là sự khác biệt với GroupBox. Khi người dùng thiết lập AutoScroll là True, đối tượng này sẽ hiển thị một thanh cuộn mà GroupBox không thể có. Tuy nhiên Panel thì không có thuộc tính Text

4.6.4.3. SplitContainer

Được sử dụng để bố trí các control khác trong hai tấm ngăn cách bởi một thanh di chuyển. Nó có thể được sử dụng để tạo ra các giao diện người dùng phức tạp.

Một số thuộc tính của SplitContainer mô tả như sau:

Thành viên	Mô tả
IsSplitterFixed	Sử dụng để thiết lập giá trị false cho phép di chuyển, và true là cố định. Mặc định là false
Orientation	Sử dụng để nhận hoặc thiết lập các giá trị định hướng chỉ ra các bảng trong việc kiểm soát SplitContainer

4.6.4.4. TabControl

Tùy vào lượng thông tin mà chương trình cần hiển thị, nếu thông tin quá nhiều ta có thể tách thông tin hiển thị trên nhiều bảng Tab khác nhau. Tab là control tương tác ngăn ta có thể dùng để tài liệu hoặc phân cách tài liệu trong những cặp hồ sơ. Mỗi một ngăn như thế ta gọi là một trang (page). Khi sử dụng control Tab ta có thể đặt nhiều control trên từng Page của Tab.

Để tạo Tab, đơn giản ta chỉ cần kéo thả control Tab lên form. Sau đó, nếu ta muốn thêm Page ta chỉ cần click chuột phải lên Tab rồi chọn Add Tab. Ta cũng có thể sử dụng thuộc tính TabPages để thêm các trang vào Tab.

4.6.4.5. FlowLayoutPanel và TableLayoutPanel

FlowLayoutPanel sử dụng để sắp xếp các control khác như TextBox và Label theo hàng ngang hoặc hàng dọc.

Một số thuộc tính của FlowLayoutPanel mô tả như sau:

Thành viên	Mô tả
FlowDirection	Định hướng các control sẽ được định vị trí
WrapContents	Nếu là true sẽ cho phép các control khác được bao bọc bên trong

TableLayoutPanel sử dụng để bố trí các control khác như TextBox và Label trong một bộ cục tương tự như lưới. Tùy thuộc vào các thuộc tính thiết lập, nó có thể mở rộng bản thân tự động để nạp thêm một control mới.

Một số thuộc tính của TableLayoutPanel mô tả như sau:

Thành viên	Mô tả
RowCount	Sử dụng để thiết lập hoặc nhận lại số lượng hàng
ColumnCount	Sử dụng để thiết lập hoặc nhận lại số lượng cột
GrowStyle	Sử dụng để thiết lập hoặc nhận lại giá trị quy định cụ thể có thể mở rộng tự động để thích ứng với các control trong nó

4.6.5. ListView và TreeView

ListView là một control dùng để hiển thị một danh sách các item với các biểu tượng. Chúng ta có thể sử dụng một ListView để tạo ra một giao diện giống như cửa sổ bên phải của Windows Explorer.

Một số thuộc tính của ListView mô tả như sau:

Thành viên	Mô tả
AllowColumnReorder	Cho phép sắp xếp cột ở chế độ thi hành (mặc định False)
Checkboxes	Xuất hiện Checkbox bên cạnh từng phần tử trên điều khiển ListView
Columns	Khai báo cột (có header)
Group	Khai báo nhóm để phân loại các phần tử sau khi trình bày trên ListView
View	Chế độ trình bày : List, Details, largeIcon, SmallIcon, Title
Sorting	Sắp xếp phần tử tăng dần (Ascending)
MultiSelect	True nếu chọn nhiều phần tử
SelectedItems	Trả về danh sách phần tử được chọn
CheckedItems	Trả về danh sách phần tử được Check

TreeView là một control trong Windows dùng để hiển thị phân cấp của các nút. Giống như các file và folder được hiển thị trong cửa sổ phía bên phải của Windows Explorer. Nó rất hữu ích trong nhiều trường hợp, giúp hiển thị các dữ liệu một cách có hệ thống hơn rõ ràng hơn.

Một số thuộc tính của TreeView mô tả như sau:

Thành viên	Mô tả
Checkboxes	Xuất hiện Checkboxes bên cạnh từng nút
Nodes	Khai báo số Node (có header)
FullRowSelect	Tô màu cho hàng được chọn (mặc định False)
Showline	True là cho phép đường viền ứng với từng nút
LabelEdit	True nếu thay đổi chuỗi của mỗi nút

4.6.6. Điều khiển Timer và biểu tượng động

Ví dụ:

Bạn cần tạo một biểu tượng động trong khay hệ thống (chẳng hạn, cho biết tình trạng của một tác vụ đang chạy). Sử dụng một Timer, Timer này sẽ phát sinh một cách định kỳ và cập nhật thuộc tính NotifyIcon.Icon. Bổ sung điều khiển NotifyIcon vào form. Thiết lập tập tin biểu tượng cho thuộc tính Icon. Không giống với các điều khiển khác, NotifyIcon sẽ tự động hiển thị menu ngữ cảnh. Chương trình sau sử dụng tám biểu tượng từ trăng này đến trăng khuyết, bằng cách thay đổi các hình này một cách định kỳ, sẽ có ảo giác hình động

```
using System;
using System.Windows.Forms;
using System.Drawing;
public class AnimatedSystemTrayIcon : System.Windows.Forms.Form
{
    Icon[] images;
    int offset = 0;
    private void Form1_Load(object sender, System.EventArgs e) {
        // Nạp vào tám icon.
        images = new Icon[8];
        images[0] = new Icon("moon01.ico");
        images[1] = new Icon("moon02.ico");
        images[2] = new Icon("moon03.ico");
        images[3] = new Icon("moon04.ico");
```

```
    images[4] = new Icon("moon05.ico");
    images[5] = new Icon("moon06.ico");
    images[6] = new Icon("moon07.ico");
    images[7] = new Icon("moon08.ico");
}

private void timer_Elapsed(object sender,
System.Timers.ElapsedEventArgs e)
{
    // Thay đổi icon.
    notifyIcon.Icon = images[offset];
    offset++;
    if (offset > 7) offset = 0;
}
```

4.6.7. Sự kiện kéo lê (Drag và Drop)

Chức năng này cho phép bạn kéo các mục từ vị trí này đến vị trí khác trong một ứng dụng đồng thời cũng có thể kéo lê một mục từ ứng dụng này tới ứng dụng khác.

Một số sự kiện kéo thả

Sự kiện	Mô tả
ItemDrag	Kéo lê một mục từ TreeView hay ListView
DragEnter	Kéo lê mục vào biên của điều khiển
DragOver	Kéo lê mục trên điều khiển
DragDrop	Hoàn thành thao tác kéo lê

Để điều khiển đích nhận dữ liệu kéo lê người dùng phải thực hiện một loạt các bước. Các bước này là:

Gọi phương thức DoDragDrop của điều khiển nguồn. Lúc này, bạn cần cung cấp dữ liệu và chỉ định kiểu hoạt động sẽ được hỗ trợ (chép, di chuyển...).

Để có thể nhận dữ liệu được kéo lê đến, điều khiển phải có thuộc tính AllowDrop là true. Điều khiển này sẽ nhận sự kiện DragEnter khi chuột kéo lê dữ liệu lên nó.

Bước cuối cùng là viết code cho sự kiện DragDrop của control đích quy định cụ thể dữ liệu đã xử lý chép vào như thế nào.

4.6.8. Kiểm tra dữ liệu hợp lệ

Để kiểm tra dữ liệu của điều khiển hợp lệ, chúng ta sử dụng sự kiện Validating của điều khiển. Sự kiện Validating ngăn cản người dùng chuyển focus sang một đối tượng khác trong khi dữ liệu nhập vào chưa đúng. Ví dụ trong form đăng ký chúng ta không thể di chuyển đến bất kỳ control khác trên form cho đến khi các thông tin xác nhận là đúng

Ví dụ:



Hình 4.17. Form đăng ký tài khoản

Yêu cầu đặt ra

- Tên tài khoản không được bỏ trống và không có dấu cách

```
private void textBox1_Validating(object sender,  
CancelEventArgs e) {  
    if (textBox1.Text.Length == 0) { //Thông báo nhắc nhở  
    }  
}
```

- Mật khẩu phải từ 6 ký tự trở lên

```
private void textBox2_Validating(object sender,
CancelEventArgs e) {
    if (textBox2.Text.Length >= 5) { //Thông báo nhắc nhở
    }
}
– Nhắc lại mật khẩu phải trùng với mật khẩu
private void textBox3_Validating(object sender,
CancelEventArgs e) {
    if (textBox3.Text != textBox2.Text) { //Thông báo nhắc
nhở
    }
}
```

Để hiển thị thông báo nhắc nhở trong sự kiện Validating, có thể sử dụng các phương pháp sau:

- Lớp MessageBox
- Sử dụng điều khiển ErrorProvider
- Sử dụng thanh trạng thái StatusStrip

4.6.8.1. Sử dụng lớp MessageBox

- textBox1: tên đăng nhập không được để trống

```
private void textBox1_Validating(object sender,
CancelEventArgs e) {
    if (textBox1.Text.Length == 0) {
        MessageBox.Show("Bạn phải nhập tên đăng nhập",
"Nhập tên đăng nhập", MessageBoxButtons.OK,
MessageBoxIcon.Exclamation);
    }
}
```

- textBox2: mật khẩu phải lớn hơn 6 ký tự

```
private void textBox2_Validating(object sender,
CancelEventArgs e) {
```

```
if (textBox2.Text.Length >= 5) {
    MessageBox.Show("Bạn phải nhập mật khẩu nhiều
hơn 6 ký tự", "Nhập mật khẩu", MessageBoxButtons.OK,
MessageBoxIcon.Exclamation) ;
}

}
- textBox3: nhắc lại mật khẩu phải trùng với mật khẩu
private void textBox3_Validating(object sender, CancelEventArgs e)
{
    if (textBox3.Text != textBox2.Text) {
        MessageBox.Show("Bạn phải nhập mật khẩu trùng nhau",
"Nhập mật khẩu", MessageBoxButtons.OK,
MessageBoxIcon.Exclamation) ;
    }
}
```

4.6.8.2. Sử dụng thanh trạng thái StatusStrip

Xem mục 4.6.2.4

4.6.8.3. Điều khiển ErrorProvider

ErrorProvider sử dụng để hiển thị một thông báo lỗi khi người sử dụng di chuyển chuột qua biểu tượng lỗi. Nó sẽ hiển thị bên cạnh control mà người dùng đã nhập vào dữ liệu không hợp lệ.

Một số thuộc tính điều khiển ErrorProvider

Sự kiện	Mô tả
ContainerControl	Thiết lập Form chứa điều khiển
BlinkStyle	Thay đổi dạng nhấp nháy của biểu tượng lỗi

Một số phương thức điều khiển ErrorProvider

Sự kiện	Mô tả
Clear	Xóa tất cả lỗi kết hợp với điều khiển ErrorProvider
Dispose	Hủy tất cả tài nguyên sử dụng bởi điều khiển ErrorProvider

GetError	Trả về chuỗi mô tả lỗi hiện hành của điều khiển chỉ rõ
SetError	Thiết lập mô tả lỗi cho điều khiển chỉ rõ

C. HÌNH THỨC VÀ PHƯƠNG PHÁP GIẢNG DẠY

- Trình chiếu powerpoint
- Đặt vấn đề, trao đổi
- Thực nghiệm kết hợp với máy tính

D. TÀI LIỆU THAM KHẢO

Tiếng Việt

- [1] Nguyễn Ngọc Bình Phuong, Thái Thanh Phong, Các giải pháp lập trình C#, Nhà sách Đất Việt

Tiếng Anh

- [2] Erik brown, Windows Forms Programming with C#, Manning Publications Co.
- [3] James W. Cooper, Introduction to Design Patterns in C#
- [4] Matthew MacDonald, Pro .NET 2.0 Windows Forms and Custom Controls in C#

Website

- [5] <http://csharpcomputing.com/Tutorials/Lesson9.htm>
- [6] <http://www.csharp-station.com/Tutorials/Lesson02.aspx>
- [7] <http://msdn.microsoft.com/en-us/library/>

E. BÀI TẬP

- 1) Xây dựng ứng dụng hiển thị danh sách tất cả phông chữ vào hộp danh sách, khi người dùng chọn phông chữ trong hộp danh sách, chuỗi sẽ hiển thị theo phông chữ người dùng chọn



2) Tạo một ứng dụng MDI. Ứng dụng có một form cha và hai form con hiển thị dữ liệu vào trong cửa sổ riêng. Bổ sung menu cho form cha để người dùng có thể chuyển đổi giữa các form

1. Các form dữ liệu vào gồm form Chi tiết sinh viên và form Chi tiết giảng viên.
2. Form Chi tiết sinh viên bao gồm tên sinh viên, tuổi, địa chỉ, phái, khóa học, thời gian, thư viện, máy tính.
3. Form Chi tiết giảng viên gồm tên giảng viên, tuổi, địa chỉ, phái, khóa học và chất lượng.
4. Bạn cần đảm bảo không điều khiển nào để trống trong form Chi tiết sinh viên, và Chi tiết giảng viên

Người dùng có thể truy cập mục menu bởi phím tắt

Bổ sung menu ngữ cảnh để truy cập các mục menu gồm Sinh viên, Giảng viên và Thoát

Bổ sung Toolbar để truy cập các mục menu dễ dàng gồm Chi tiết sinh viên, Chi tiết giảng viên và Thoát

3) Tạo ứng dụng giao diện đồ họa như sau:

- Tạo lớp số phức chứa:

Thuộc tính:

- Phần thực

- Phần ảo

Hàm:

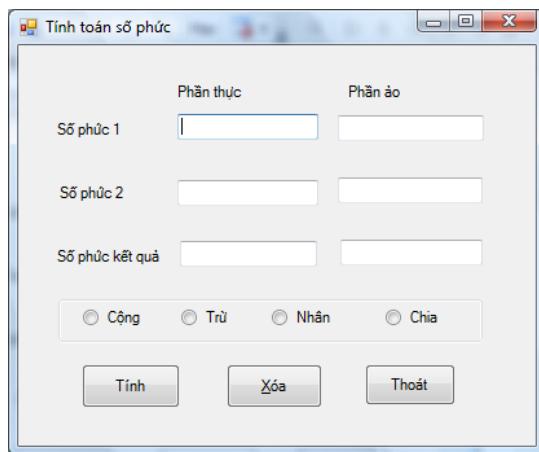
- Khởi tạo số phức

- In số phức

- Nạp chồng phép cộng số phức

- Nạp chồng phép trừ số phức

- Nạp chồng phép nhân số phức
 - Nạp chồng phép chia số phức
- Tạo lớp giao diện đồ họa như sau sử dụng các hàm của lớp số phức, với yêu cầu:
- + Chọn một trong các RadioButton Cộng, Trừ, Nhân hay Chia, sẽ thực hiện cộng, trừ, nhân hay chia 2 số phức 1 và 2, và hiển thị kết quả
 - + Chọn Button Xóa, sẽ xóa các TextField
 - + Chọn Button Thoát, sẽ thoát chương trình



4) Xây dựng ứng dụng quản lý sinh viên bao gồm:

1. Lớp SinhVien gồm:

Thuộc tính:

- Mã sinh viên
- Họ tên sinh viên
- Năm sinh
- Lớp
- Địa chỉ
- Điểm tổng kết

Phương thức:

- Phương thức khởi tạo các thuộc tính của lớp SinhVien
- Các phương thức get và set để nhận và thiết lập các giá trị thuộc tính cần thiết
- Phương thức trả về xếp loại của sinh viên, biết rằng sinh viên xếp loại

Điểm tổng kết	Xếp loại
9 - 10	Xuất sắc
8 - <9	Giỏi
7 - <8	Khá
5 - <7	Trung bình
<5	Không đạt

2. Xây dựng lớp DSSV bao gồm:

Thuộc tính:

- Mảng các đối tượng sinh viên
- Số phần tử sinh viên hiện có trong mảng

Phương thức:

- Thêm một sinh viên vào mảng
- Thống kê số lượng sinh viên đạt
- Tìm sinh viên tương ứng với mã sinh viên đã biết
- Xóa sinh viên tương ứng với mã sinh viên
- Xóa tất cả sinh viên thuộc lớp đã biết
- Kiểm tra sinh viên có thuộc lớp nào đó không

3. Xây dựng lớp giao diện đồ họa, hiển thị giao diện nhập thông tin một sinh viên, và các nút lệnh Thêm, Thống kê, Tìm, Xóa, Xóa lớp, Kiểm tra. Khi người dùng kích vào các nút lệnh, các phương thức tương ứng đã xây dựng trong lớp DSSV sẽ được gọi:

- Nhập thông tin sinh viên trên giao diện đồ họa, khi kích nút Thêm, sẽ thêm hồ sơ sinh viên này vào danh sách
- Kích nút Thống kê, sẽ hiển thị số lượng sinh viên đạt trên giao diện đồ họa
- Nhập mã sinh viên, khi kích nút Tìm, sẽ hiển thị thông tin sinh viên trên giao diện đồ họa
- Nhập mã sinh viên, khi kích nút Xóa, sẽ xóa hồ sơ sinh viên từ danh sách
- Nhập lớp, khi kích nút Xóa lớp, sẽ xóa tất cả sinh viên thuộc lớp đó
- Nhập mã sinh viên, và mã lớp, khi kích vào nút Kiểm tra, sẽ kiểm tra và thông báo sinh viên này có thuộc lớp hay không

5) Xây dựng ứng dụng quản lý danh sách các sách trong thư viện:

1. Xây dựng lớp Sach gồm:

Thuộc tính:

- Mã sách
- Tên sách
- Năm xuất bản
- Đơn giá
- Số lượng

Phương thức:

- Phương thức khởi tạo các thuộc tính của lớp Sach
- Các phương thức get và set để nhận và thiết lập các giá trị thuộc tính cần thiết
- Phương thức tính giá sách = số lượng x đơn giá
- Phương thức tính giá tiền vận chuyển biết rằng:

Giá vận chuyển

Nếu số lượng ≤ 50

0

Ngược lại, nếu số lượng ≤ 500 2% của giá sách

Ngược lại 5% của giá sách

2. Xây dựng lớp DMSach, để quản lý danh sách các sách trong thư viện bao gồm:

Thuộc tính:

- Mảng các đối tượng sách
- Số lượng sách hiện có trong mảng

Phương thức:

- Thêm một sách vào mảng
- Tính giá sách (kể cả giá vận chuyển) trong mảng
- Xóa sách tương ứng với mã sách đã biết
- Tìm sách có số lượng lớn nhất trong thư viện
- Tìm sách có mã sách nào đó
- Đếm các sách có số lượng từ 50 đến 100 trong mảng

3. Xây dựng lớp giao diện đồ họa, hiển thị giao diện nhập thông tin sách, và các nút lệnh Thêm, Tổng, Xóa, Tìm lớn nhất, Tìm sách, Đếm. Khi người dùng kích vào các nút lệnh, các phương thức tương ứng đã xây dựng trong lớp DMSach sẽ được gọi:

- Nhập thông tin sách trên giao diện đồ họa, khi kích nút Thêm, sẽ thêm sách này vào danh sách
- Kích nút Tổng, sẽ tính tổng giá sách (kể cả giá vận chuyển) của các sách trong danh sách và hiển thị trên giao diện đồ họa
- Nhập mã sách, khi kích nút Xóa sẽ sách tương ứng với mã sách này trong danh sách
- Kích nút Tìm lớn nhất, sẽ tìm sách có số lượng lớn nhất trong thư viện và hiển thị thông tin sách này trên giao diện đồ họa
- Kích nút Tìm sách, sẽ tìm sách có mã sách nhập vào, và hiển thị thông tin sách trên giao diện đồ họa

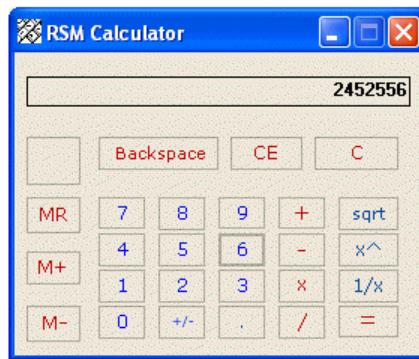
- Kích nút Đếm, sẽ đếm các sách có số lượng từ 50 đến 100, và hiển thị kết quả trên giao diện đồ họa

6) Tạo máy tính tay như ứng dụng Calculator của Windows, gồm các phép tính cơ bản: cộng, trừ, nhân, chia, khai căn, lũy thừa, 1/x, xóa màn hình, xóa ký tự kế trước, các phép toán trên bộ nhớ như MC, MR, M+, M-,

Hướng dẫn:

Xây dựng lớp Calculator_Core gồm các thao tác của máy tính tay

Xây dựng lớp giao diện đồ họa, sử dụng các hàm của lớp Calculator_Core



7) Viết chương trình xử lý chuỗi với yêu cầu như sau:

- Chọn File/ Exit, hoặc nút Exit, hoặc kích Mouse vào nút Close: thoát chương trình
- Gõ dữ liệu tùy ý vào hộp nhập, chọn khối dữ liệu trong hộp nhập, rồi chọn:

Edit/ Clear All: xoá hộp nhập

Edit/ Clear Selection: xoá khối chọn

Edit/ LTrim: cắt các khoảng trắng bên trái khối chọn

Edit/ RTrim: cắt các khoảng trắng bên phải khối chọn

Edit/ CTrim: cắt các khoảng trắng thừa giữa các từ của khối chọn

- Gõ dữ liệu tùy ý vào hộp nhập, chọn khối dữ liệu trong hộp nhập, rồi chọn:

Format/ Change Case/ Sentence case: đổi chữ đầu của khối chọn thành chữ hoa

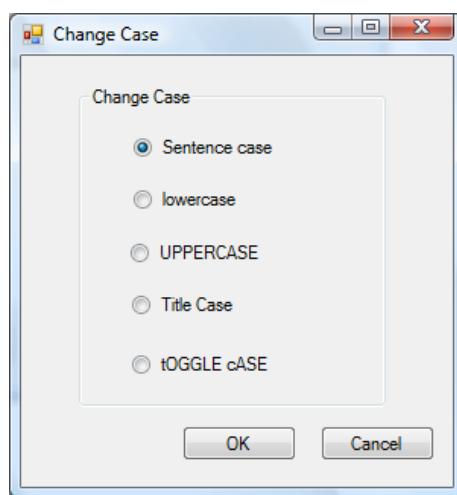
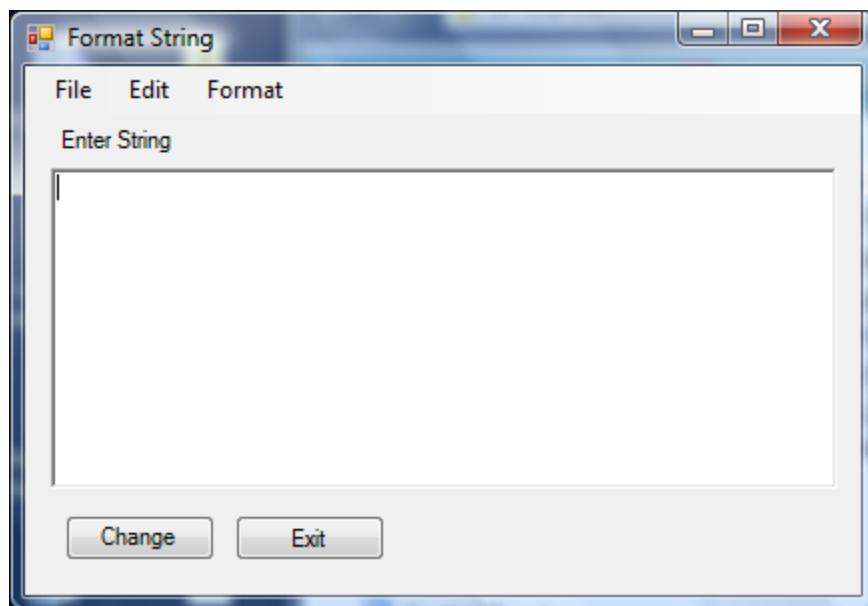
Format/ Change Case/ lowercase: đổi khối chọn thành chữ thường

Format/ Change Case/ UPPERCASE: đổi khối chọn thành chữ hoa

Format/ Change Case/ Title Case: đổi các chữ đầu mỗi từ thành chữ hoa

Format/ Change Case/ tOGGLE cASE: đổi các chữ hoa thành chữ thường và đổi các chữ thường thành chữ hoa

- Khi chọn nút Change Case, hiển thị hộp đổi thoại gồm các mục có chức năng như các mục của menu Format/ Change Case:



- Tương tự, sinh viên mở rộng thêm các chức năng để tạo ứng dụng soạn thảo văn bản đơn giản như Notepad

8) Xây dựng ứng dụng Windows Form Hangman game

- Cho tập tin văn bản lưu mỗi từ trên một dòng, dòng đầu là số từ trong tập tin. Đọc tập hợp các từ trong tập tin này vào danh sách
- Mỗi khi bắt đầu game, hiển thị câu hỏi “Bạn muốn chơi nữa không?”, nếu người chơi trả lời “Vâng”, một từ bí mật ngẫu nhiên từ danh sách sẽ hiển thị, mỗi ký tự của từ hiển thị một dấu ?, nếu người chơi trả lời “Không”, chương trình hiển thị bao nhiêu game người chơi đã thắng hay thua.
- Mỗi game cho phép người chơi đoán ký tự, không trùng với ký tự đã đoán, nếu đoán trùng sẽ đếm lỗi và hiển thị thông báo lỗi
- Nếu người chơi đoán ký tự không trùng ký tự đã đoán, và ký tự đoán không trùng với một trong các ký tự của từ, sẽ đếm lỗi và hiển thị một thông báo lỗi, nếu đoán đúng một thông báo đoán trúng sẽ hiển thị và ký tự đoán trùng hiển thị thay cho dấu ?
- Nếu tất cả ký tự của từ đã được đoán, một thông báo chúc mừng người chơi đã thắng game, nếu từ chưa được đoán hết và số lỗi đạt đến 10 lỗi, một thông báo chia buồn thua game hiển thị cùng với kết quả của từ bí mật, và game sẽ bắt đầu, cho phép người chơi đoán từ khác hay kết thúc game tùy người chơi trả lời “Vâng” hay “Không” cho câu hỏi “Bạn muốn chơi nữa không”

Hướng dẫn:

Thiết kế cấu trúc lớp ứng dụng:

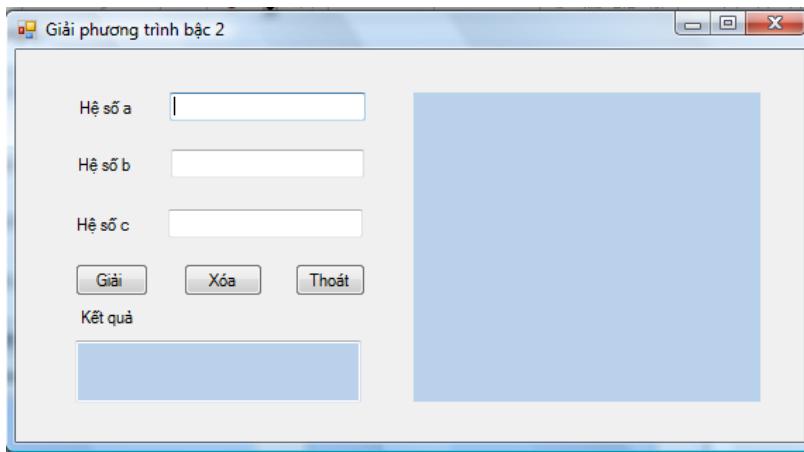
Xây dựng các lớp xử lý cần thiết

Xây dựng lớp giao diện đồ họa

9) Giải phương trình bậc 2 có vẽ đồ thị với yêu cầu như sau:

- Khi chọn nút Giải, từ các hệ số phương trình a, b, c nhập từ các hộp nhập, giải phương trình bậc 2, hiển thị kết quả ra hộp kết quả, và vẽ đồ thị phương trình $y = ax^2+bx+c$ ở Panel bên phải
- Khi chọn nút Xoá, các hộp nhập xoá về rỗng

- Khi chọn nút Thoát, đóng cửa sổ ứng dụng, thoát chương trình



10) Xây dựng ứng dụng mô phỏng ứng dụng Paint của Microsoft Windows gồm các chức năng cơ bản:

- Thanh công cụ vẽ, chọn màu vẽ, màu nền, menu
- Các công cụ vẽ đường thẳng, hình chữ nhật, vuông, hình tròn, elip, đường cong, hình chữ nhật tròn gốc...
- Chức năng menu: File/ New, Open, Save, Exit
- Tương tự, sinh viên mở rộng thêm các chức năng để tạo ứng dụng vẽ hình như ứng dụng Paint

CHƯƠNG 5: TRUY XUẤT CƠ SỞ DỮ LIỆU

A. MỤC TIÊU CHƯƠNG

1. VỀ KIẾN THỨC

Trong chương này, sinh viên sẽ được cung cấp kiến thức truy cập cơ sở dữ liệu bởi ADO.NET, làm thế nào để xây dựng các ứng dụng truy cập cơ sở dữ liệu, đặc biệt là ứng dụng quản lý bởi C# và ADO.NET

- ❖ Giới thiệu ADO.NET
 - Kiến trúc ADO.NET
- ❖ Kết nối và truy cập cơ sở dữ liệu
 - Đối tượng connection
 - Truy cập cơ sở dữ liệu bởi ADO.NET theo mô hình kết nối
 - Đối tượng command
 - Đối tượng data reader
 - Truy cập cơ sở dữ liệu bởi ADO.NET theo mô hình phi kết nối
 - Đối tượng data adapter
 - Đối tượng DataSet
 - Đối tượng DataTable
 - Đối tượng DataView
 - Đối tượng DataRelation
 - Ràng buộc dữ liệu cho các điều khiển
 - Tạo lập báo cáo sử dụng Crystal Report

2. VỀ KỸ NĂNG

Sau khi học xong chương này sinh viên có thể vận dụng những kiến thức về truy xuất cơ sở dữ liệu để có thể triển khai những dự án

B. NỘI DUNG

5.1. Giới thiệu ADO.NET

5.1.1. Giới thiệu chung

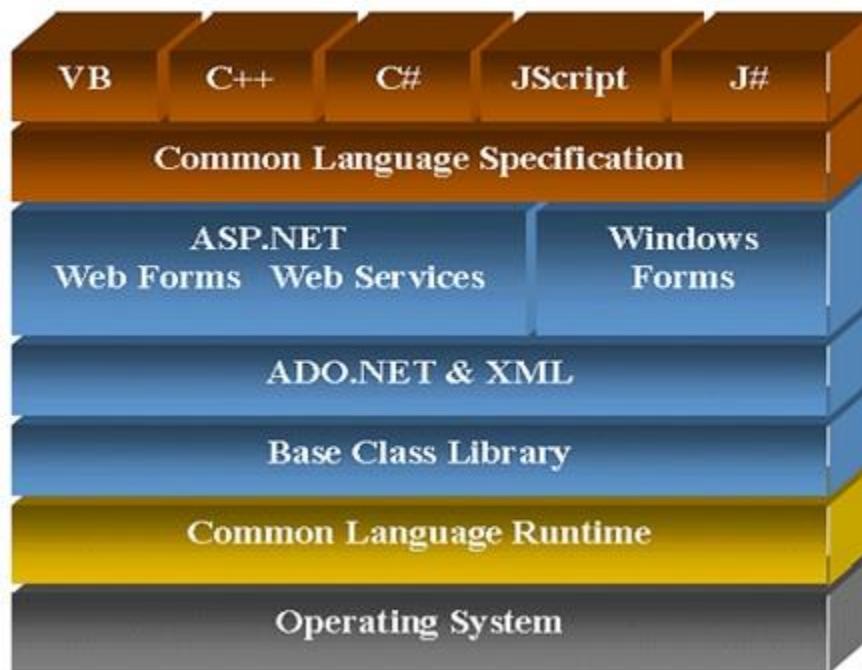
ADO.NET là một tập các lớp nằm trong bộ thư viện lớp cơ sở của .NET Framework, cho phép các ứng dụng windows (như C#, VB.NET, C++, C++ for Windows) hay ứng dụng web (như ASP.NET) thao tác dễ dàng với các nguồn dữ liệu.

Mục tiêu chính của ADO.NET là:

- Cung cấp các lớp để thao tác CSDL trong cả hai môi trường là ngắt kết nối (Disconnected data) và kết nối (Connected data).
- Tích hợp chặt chẽ với XML (Extensible Markup Language)
- Tương tác với nhiều nguồn dữ liệu thông qua mô tả dữ liệu chung.
- Tối ưu truy cập nguồn dữ liệu (OLE DB & SQL server).
- Làm việc trên môi trường Internet (môi trường phi kết nối – Disconnected environment).

Các lớp của ADO.NET được đặt trong Namespace là System.Data/ System.Data.oledb. ADO.NET bao gồm 2 Provider (2 bộ thư viện) (thường dùng) để thao tác với các CSDL là: OLE DB Provider (nằm trong System.Data.OLEDB) dùng để truy xuất đến bất kỳ CSDL nào có hỗ trợ OLEDB; SQL Provider (nằm trong System.Data.SqlClient) chuyên dùng để truy xuất đến CSDL SQL Server (Không qua OLE DB nên nhanh hơn).

- Vị trí của ADO.NET trong kiến trúc của .NET Framework



Hình 5.1. Vị trí của ADO.NET trong kiến trúc của .net Framework

Từ kiến trúc ta thấy rằng: ADO.NET là một thành phần nội tại () của .NET framework, do vậy nó có thể được sử dụng trong tất cả các ngôn ngữ hỗ trợ .NET như C#, VB.NET... mà không có sự khác biệt nào (Tức là các chức năng cũng như cách sử dụng hoàn toàn giống nhau).

5.1.2. So sánh với phiên bản ADO

Hai mô hình lập trình tương tự nhau

- ADO.NET được thiết kế làm việc với cả dữ liệu phi kết nối trong môi trường đa tầng (Multi-Tier). Nó sử dụng XML để trao đổi dữ liệu phi kết nối, do vậy dễ dàng khi giao tiếp giữa các ứng dụng không phải trên nền windows.
- ADO.NET là thành phần nội tại (có sẵn) trong .NET Framework, do vậy dễ dàng khi phát triển bằng nhiều ngôn ngữ khác nhau.
- ADO.NET Hỗ trợ XML hoàn toàn (ADO thì không), nghĩa là chúng ta có thể nạp dữ liệu từ một tệp XML và thao tác như một CSDL, sau đó cũng có thể lưu kết quả ngược trở lại tệp XML.
- ADO.NET được thiết kế hoàn toàn hướng đối tượng : đây là đặc điểm chi phối toàn bộ các sản phẩm Microsoft .NET

ADO lưu trữ dữ liệu dưới dạng nhị phân nên có thể bị chặn bởi Firewall, còn ADO.NET lưu trữ dữ liệu dưới dạng XML nên có thể đi qua một cách dễ dàng

Trước ADO.NET, Microsoft đã có ADO là một bộ thư viện để xử lý các thao tác liên quan đến dữ liệu. ADO có tính linh hoạt, dễ sử dụng và được tích hợp trong các ngôn ngữ như Visual Basic, ASP 3.0.

Ta có bảng so sánh ADO và ADO.NET

Đặc điểm	ADO	ADO.NET
Dữ liệu xử lý được đưa vào bộ nhớ dưới dạng	Recordset : tương đương 1 bảng dữ liệu trong database	Dataset : tương đương 1 database
Duyệt dữ liệu	Recordset chỉ cho phép duyệt tuần tự, từng dòng một.	Dataset : cho phép duyệt “tự do, ngẫu nhiên”, truy cập thẳng tới bảng ,dòng ,cột mong muốn.
Dữ liệu ngắt kết nối	Recordset cũng có thể ngắt kết nối nhưng tư tưởng thiết kế ban đầu của Recordset là hướng kết nối, do đó việc ngắt kết nối cũng không được hỗ trợ tốt nhất.	Dataset được thiết kế với tư tưởng ban đầu là “ngắt kết nối” và hỗ trợ mạnh mẽ “ngắt kết nối”.
Khả năng vượt tường lửa	Khi trao đổi dữ liệu với ADO qua Internet, thường sử dụng chuẩn COM, chuẩn COM rất khó vượt qua được tường lửa. Do vậy khả năng trao đổi dữ liệu ADO qua Internet thường có nhiều hạn chế.	ADO.NET trao đổi dữ liệu qua Internet rất dễ dàng vì ADO.NET được thiết kế theo chuẩn XML, là chuẩn dữ liệu chính được sử dụng để trao đổi trên Internet.

5.1.3. Kiến trúc ADO.NET

Kiến trúc ADO.NET bao gồm 2 phần chính:

a. Phần kết nối

.Net Data Provider cho phép truy cập các nguồn dữ liệu xác định (System.Data.SqlClient dùng truy cập SQL Server 7.0 trở lên, System.Data.OleDb dùng truy cập bất kỳ nguồn dữ liệu nào hỗ trợ OLE DB). Bao gồm đối tượng kết nối cơ sở dữ liệu connection, đối tượng command thực hiện truy vấn dữ liệu hay thao tác dữ liệu từ cơ sở dữ liệu vào bộ nhớ, data reader bộ đọc dữ liệu và data adapter bộ điều phối dữ liệu, đối tượng DataAdapter để thay đổi dữ liệu nguồn và một DataSet.

Các giao thức như ODBC, OleDb, hay khác hơn là các giao thức khác thông qua thư viện .NET của lớp ADO.NET

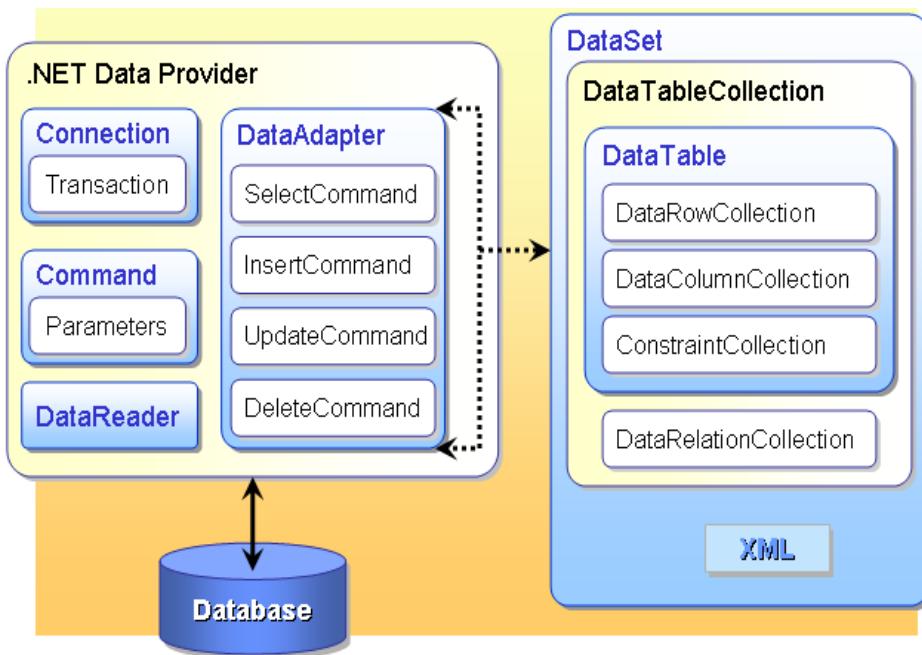
Một số giao thức thường dùng như:

Provider Name	Tiền tố API	Data Source
ODBC Data Provider	Odbc	Dùng cho cơ sở dữ liệu cũ
OleDb Data Provider	OleDb	Dùng cho cơ sở dữ liệu như Access, Excel
Oracle Data Provider	Oracle	Dùng cho cơ sở dữ liệu Oracle
SQL Data Provider	Sql	Dùng cho cơ sở dữ liệu SQL Server
Borland Data Provider	Bdp	Dùng cho cơ sở dữ liệu Interbase, SQL Server, IBM DB2, Oracle

b. Phân ngắt kết nối (Dataset)

Mỗi công nghệ truy cập dữ liệu đều đã được cải thiện khái niêm không kết nối, nhưng đến ADO.Net mới cung cấp giải pháp cách đầy đủ. ADO.Net được thiết kế dùng cho Internet. ADO.Net sử dụng XML như là các định dạng truyền tải. ADO.Net cung cấp một đối tượng mới cho việc caching dữ liệu trên máy client. Đối tượng này là DataSet.

DataSet là phần cơ sở dữ liệu được lưu trữ trong bộ nhớ (in-memory database), cơ chế không kết nối, nhờ đối tượng DataAdapter làm trung gian, hỗ trợ đầy đủ đặc tính XML.



Hình 5.2. Kiến trúc ADO.NET

Phần kết nối(Data provider) sử dụng khi kết nối CSDL và thao tác dữ liệu, phải thực hiện kết nối khi thao tác. Tuy nhiên .Net cung cấp các thư viện thao tác trên các nguồn dữ liệu khác nhau nhưng cách thức về giao tiếp lập trình giống nhau.

Các đối tượng của phần này là:

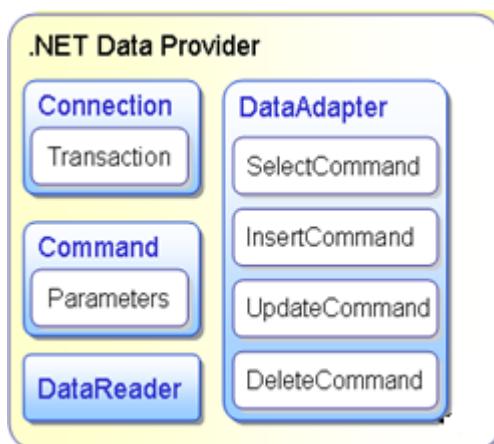
Connection : Đối tượng quản lý đóng /mở kết nối tới Database. Tùy nguồn dữ liệu, thư viện .Net cung cấp đối tượng connection khác nhau như OleDbConnection, SqlConnection...

Command : Đối tượng thực hiện các câu lệnh SQL, thủ tục lưu trữ (store procedure), khi đã thiết lập kết nối tới dữ liệu và trả về kết quả. Tương tự như connection, command cũng có các thư viện hỗ trợ tùy theo nguồn dữ liệu là gì như OleDbCommand và SqlCommand.

DataReader : Đối tượng xử lý đọc dữ liệu. Chỉ Xử lý 1 dòng dữ liệu tại một thời điểm. Phù hợp với ứng dụng Windows Form và Web form vì xử lý nhanh, nhẹ không chiếm bộ nhớ. Hai thư viện hỗ trợ thao tác dữ liệu tùy theo dữ liệu nguồn : OleDbDataReader và SqlDataReader. Dữ liệu của đối tượng được tạo ra khi đối tượng Command thực hiện câu lệnh ExecuteReader().

DataAdapter : Là cầu nối của database và dataset (dataset là đối tượng ngắt kết nối), bởi vì đối tượng “ngắt kết nối” dataset không thể liên lạc trực tiếp với database nên nó cần một đối tượng trung gian lấy dữ liệu từ database cho nó. Và đó chính là DataAdapter. Vì DataAdpater khi thao tác với Database vẫn phải duy trì kết nối nên nó được liệt kê vào dạng “kết nối”, nhưng bản chất là phục vụ cho việc “ngắt kết nối”.

Dưới đây là mô hình kết nối là một phần trong kiến trúc của ADO.NET



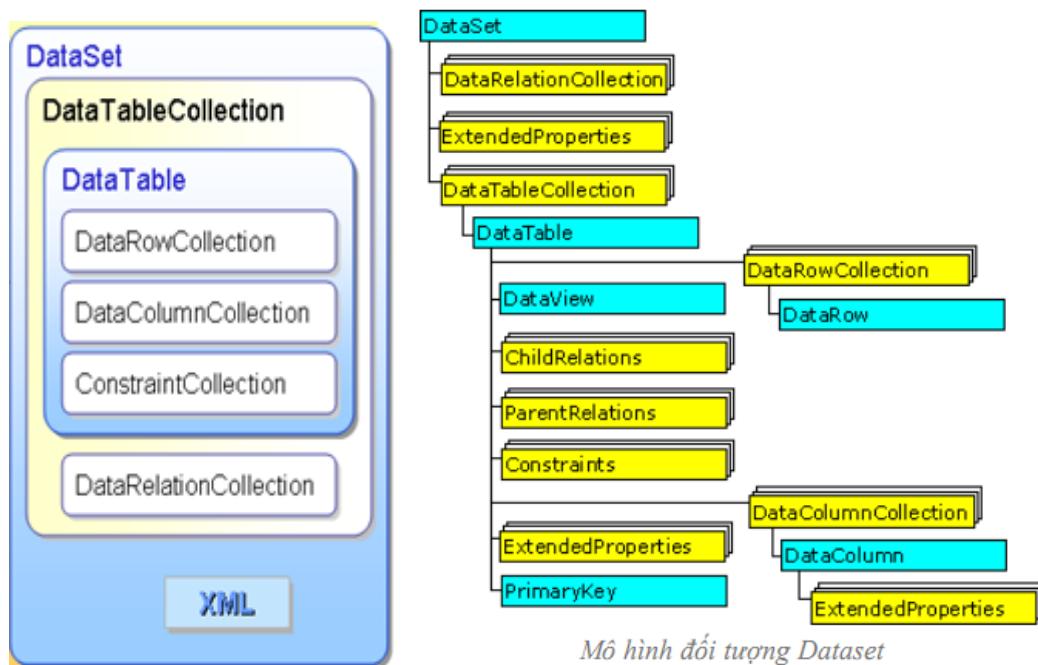
Hình 5.3. Kiến trúc kết nối

Ngắt kết nối (Data set).

DataSet là thành phần chính cho đặc trưng kết nối không liên tục của kiến trúc ADO.NET. DataSet được thiết kế để có thể thích ứng với bất kỳ nguồn dữ liệu nào.

DataSet chứa một hay nhiều đối tượng DataTable mà nó được tạo từ tập các dòng và cột dữ liệu, cùng với khoá chính, khoá ngoại, ràng buộc và các thông tin liên quan đến đối tượng DataTable này. Bản thân DataSet được dạng như một tập tin XML. Dataset có thể được xem như là thể hiện của cả một cơ sở dữ liệu con, lưu trữ trên vùng nhớ cache của máy người dùng mà không kết nối đến cơ sở dữ liệu.

Dưới đây là mô hình ngắt kết nối là một phần trong kiến trúc của ADO.NET



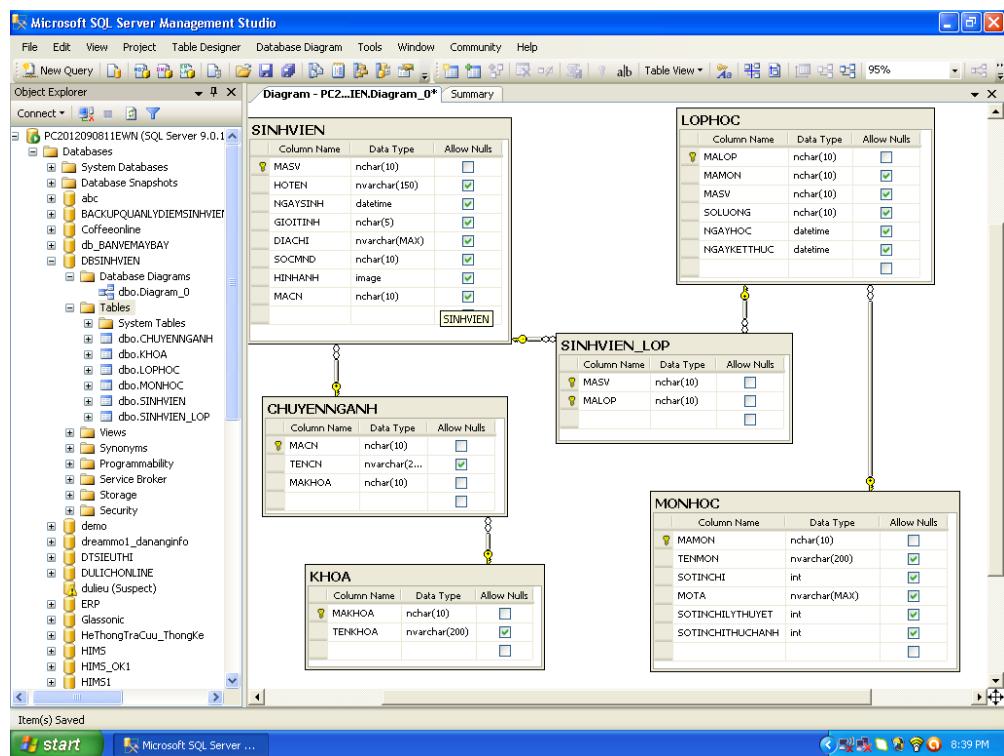
Hình 5.4. Kiến trúc ngắn kết nối

Các thư viện ADO.NET hỗ trợ thao tác dữ liệu trên hai đối tượng kết nối và ngắn kết nối

Name space	Cung cấp
System.Data	Các lớp tạo và truy cập dữ liệu như DataSet, DataTable, DataRelation
System.Data.Common	Các lớp dùng chung bởi các data provider khác nhau
System.Data.SqlClient	Các lớp truy cập cơ sở dữ liệu Microsoft SQL Server như SqlConnection, SqlCommand
System.Data.SqlTypes	Các kiểu của SQL Server
System.Data.OracleClient	Các lớp truy cập cơ sở dữ liệu Oracle (Microsoft .NET Framework phiên bản 1.1 trở đi)
System.Data.OleDb	Các lớp truy cập các cơ sở dữ liệu như Microsoft Access, MySQL như OleDbConnection, OleDbCommand
System.Xml	Các lớp xử lý dữ liệu XML

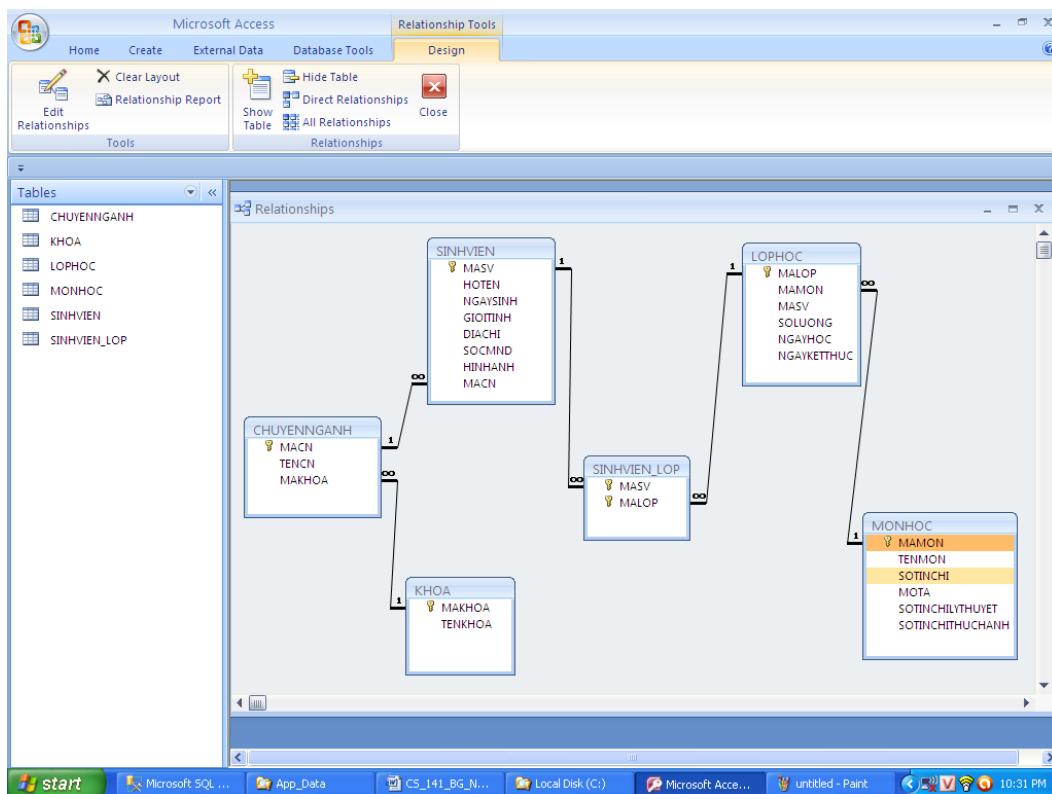
5.2. Kết nối và truy xuất cơ sở dữ liệu

Cơ sở dữ liệu DBSINHVIEN SQL Server



Hình 5.5. Cơ sở dữ liệu DBSINHVIEN

Cơ sở dữ liệu MICROSOFT ACCESS DBSINHVIEN.mdb



Hình 5.6. Cơ sở dữ liệu MICROSOFT ACCESS DBSINHVIEN.mdb

5.2.1. Đối tượng Connection

Đối tượng Connection là đối tượng chịu trách nhiệm quản lý kết nối tới nguồn dữ liệu (DataSource). Có 2 dạng Connection tương ứng với 2 kiểu dữ liệu SQL Server và OLE DB đó là : SqlConnection và OleDbConnection. Cả 2 đối tượng này đều implement từ interface IDbConnection. Bằng cách sử dụng Interface IDbConnection, các nhà cung cấp dịch vụ Database khác nhau có thể tạo ra các cài đặt phù hợp cho Database riêng của họ. Đối tượng Connection của ADO.NET chỉ nhận một tham số đầu vào là chuỗi kết nối (connection string). Trong chuỗi kết nối, các thông số được cách nhau bằng dấu “;”, connection string có các thông số sau:

- Provider : Tên nhà cung cấp Database, đối với OLEDB cần khai báo là SQLOLEDB. Đối với SQL Server thì không thuộc tính này.
- DataSource (hoặc Server) : tên/địa chỉ database server cần kết nối tới.
- Initial catalog (hoặc Database) : tên của Database cần truy xuất.
- Uid : username để đăng nhập vào Database Server.
- Pwd : password để đăng nhập vào Database Server.

Sau đây là ví dụ về 1 chuỗi kết nối đối với Database dạng OLE DB:

Database	ODBC/OLE DB Connection (*)	Mô tả
MS Access	Driver = {Microsoft Access Driver (*.mdb)}; DBQ = <đường dẫn file access>	Provider=Microsoft.Jet.OLEDB.4.0; Data Source = <đường dẫn file access>
SQL Server	Driver = {SQLServer}; Server = ServerName; Database= DatabaseName; Uid=Username; Pwd=Password;	Provider=SQLOLEDB.1; Data Source=MySQL; Initial Catalog=NorthWind; uid=sa; pwd=sa;

Còn đây là ví dụ về 1 chuỗi kết nối đối với Database dạng SQL Server:

Server=CSC; database=Northwind; uid=sa; pwd=sa;

Dưới đây là các thuộc tính, phương thức, sự kiện thông dụng của cả SqlConnection và OleDbConnection :

Tên	Mô tả
ConnectionString	Cung cấp thông tin như datasource, tên cơ sở dữ liệu, được sử dụng để thiết lập kết nối với một cơ sở dữ liệu
Open()	Mở một kết nối với datasource được khai báo tại ConnectionString
Close()	Được sử dụng để đóng kết nối với data source
State	Được sử dụng để kiểm tra trạng thái của một kết nối. 0: kết nối đang đóng, 1: kết nối đang mở.
ConnectionTimeout	Thuộc tính thiết lập / lấy thời gian chờ trong khi truy xuất vào database. Khi truy xuất vào Database, chương trình sẽ chờ đúng khoảng thời gian này nếu chờ qua khoảng thời gian này mà vẫn không kết nối được vào database thì chương trình sẽ báo lỗi.
Database	Thuộc tính thiết lập/ lấy tên database của đối tượng connection hiện thời.
DataSource	Thuộc tính thiết lập/lấy tên của database server của đối tượng connection hiện thời.
BeginTransaction	Sử dụng cho trường hợp xử lý giao tác của ứng dụng. Việc xử lý giao tác rất có lợi trong khi xử lý dữ liệu từ database vì có lúc trong khi xử lý dữ liệu gặp lỗi thì có thể thực hiện câu lệnh như Rollback hay trong lúc thao tác cũng có thể thực hiện được các giao tác chính như trên SQL Server như Commit...
InfoMessage	Xảy ra khi provider gửi ra 1 lời cảnh cáo hay thông tin.
StateChange	Xảy ra khi trạng thái của connection thay đổi.

Một số chuỗi kết nối như sau:

Với Access chuẩn:

`Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\mydatabase.mdb;`

`User Id=admin;Password=;`

Với Access qua mạng:

`Provider=Microsoft.Jet.OLEDB.4.0;`

```
Data Source=\\serverName\\shareName\\folder\\myDatabase.mdb;
```

```
User Id=admin;Password=;
```

Với Excel chuẩn:

```
Provider=Microsoft.Jet.OLEDB.4.0;
```

```
Data Source=C:\\MyExcel.xls;
```

```
Extended Properties="Excel 8.0;HDR=Yes;IMEX=1";
```

Với SQL Server:

```
Provider=Microsoft.Jet.OLEDB.4.0;
```

```
Data Source=C:\\MyExcel.xls;
```

```
Extended Properties="Excel 8.0;HDR=Yes;IMEX=1";
```

Với Attach Database:

```
Server=.\\SQLEXPRESS;
```

```
AttachDbFilename=|DataDirectory|MyDataFile.mdf;
```

```
Database=dbname;Trusted_Connection=Yes;
```

Với MySQL:

```
Server=myServerAddress;
```

```
Database=myDataBase;
```

```
Uid=myUsername;
```

```
Pwd=myPassword;
```

Với Oracle:

```
Data Source=T0RCL;User Id=myUsername;Password=myPassword;
```

Ví dụ 1: Kết nối cơ sở dữ liệu sử dụng thư viện OleDB đối với dữ liệu nguồn là Access.

```
OleDbConnection con = new OleDbConnection();
```

```
con.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;  
Data Source= DBSINHVIEN.mdb;  
Persist Security Info=False";  
con.Open();  
//xử lý trong quá trình kết nối  
...  
con.Close();
```

Ví dụ 2: Kết nối cơ sở dữ liệu sử dụng thư viện SQLClient đối với dữ liệu nguồn là

SQL Server

```
tringconnString="server=(local);uid=sa;pwd=;  
database= DBSINHVIEN ";  
SqlConnection con = new SqlConnection(connString);  
con.Open();  
//xử lý trong quá trình kết nối  
...  
con.Close();
```

Ví dụ 3: Kết nối cơ sở dữ liệu sử dụng thư viện SQLOLEDB đối với dữ liệu nguồn là SQL Server.

```
tringconnString=" Provider=SQLOLEDB.1;  
Data Source=localhost;  
Initial Catalog= DBSINHVIEN; uid=sa; pwd=sa;";  
OleDbConnection con = new OleDbConnection(connString);  
con.Open();
```

//xử lý trong quá trình kết nối

...

con.Close();

5.2.2. Đối tượng command

Sau khi kết nối vào nguồn dữ liệu, chúng ta cần phải thực hiện thao tác các dữ liệu. Để thao tác được với các dữ liệu chúng ta phải dùng đối tượng Command. Đối tượng Command là đối tượng rất “đa năng”, nó vừa xử lý được sqlserver stored procedures vừa xử lý được các giao tác (transaction).

Tương tự như đối tượng Connection, đối tượng Command cũng chia ra làm 2 loại tùy theo nguồn dữ liệu : SqlCommand (cho SQL Server) và OleDbCommand (cho OLE DB).

5.2.2.1. Các thuộc tính của đối tượng command

- Thuộc tính CommandText.

Cho phép bạn khai báo câu truy vấn SQL hay thủ tục(stored procedure) truy cập cơ sở dữ liệu.

Chú ý: Thuộc tính CommandText phụ thuộc vào giá trị của thuộc tính CommandType. Cụ thể khi CommandType có giá trị là văn bản, CommandText tài sản cần có văn bản của một truy vấn mà phải được chạy trên máy chủ. Nó cũng có thể chứa một số câu lệnh SQL, tách biệt với dấu chấm phẩy. Bạn có thể sử dụng các thông số trong tất cả các câu. Nếu có một số báo cáo, trả về dữ liệu, sử dụng SQLiteDataReader.NextResult để có được tất cả các kết quả đọc.

Khi giá trị được thiết lập để TableDirect, CommandText phải là tên của một bảng bạn muốn để có được tất cả các dữ liệu từ. Tất cả các hàng và các cột được lấy ra.

Khi thuộc tính CommandType được thiết lập cho StoredProcedure, giá trị thuộc tính CommandText phải là tên thủ của thủ tục . Lệnh thực hiện thủ tục này có hiệu lực khi bạn gọi phương thức Execute.

Khi thuộc tính CommandType được thiết lập cho TableDirect, giá trị thuộc tính CommandText phải là tên bảng. Tất cả các hàng và các cột của bảng được đặt tên hoặc bảng sẽ được trả lại khi bạn gọi phương pháp Execute.

Bạn không thể thiết lập kết nối, CommandType, và tài sản CommandText nếu kết nối hiện tại đang thực hiện một hoạt động thực hiện hoặc lấy.

Các nhà cung cấp OLE DB.NET không hỗ trợ các thông số được đặt tên theo truyền tham số cho một bản Tuyên Bố SQL hoặc một thủ tục lưu trữ được gọi bằng một OleDbCommand khi CommandType được thiết lập để Text. Trong trường hợp này, dấu hỏi giữ chỗ (?) Phải được sử dụng. Ví dụ:

```
SELECT * FROM khachhang WHERE ID khachhang =?
```

Do đó, thứ tự mà đối tượng OleDbParameter được thêm vào OleDbParameterCollection phải tương ứng trực tiếp đến vị trí của các giữ chỗ dấu hỏi cho tham số.

Để biết thêm thông tin, xem thông số cấu hình và loại thông số dữ liệu.

Ví dụ:

```
public void CreateMyOleDbCommand()
{
    OleDbCommand command = new OleDbCommand();
    command.CommandText = "SELECT * FROM Sinhvien";
    command.CommandTimeout = 20;
}
```

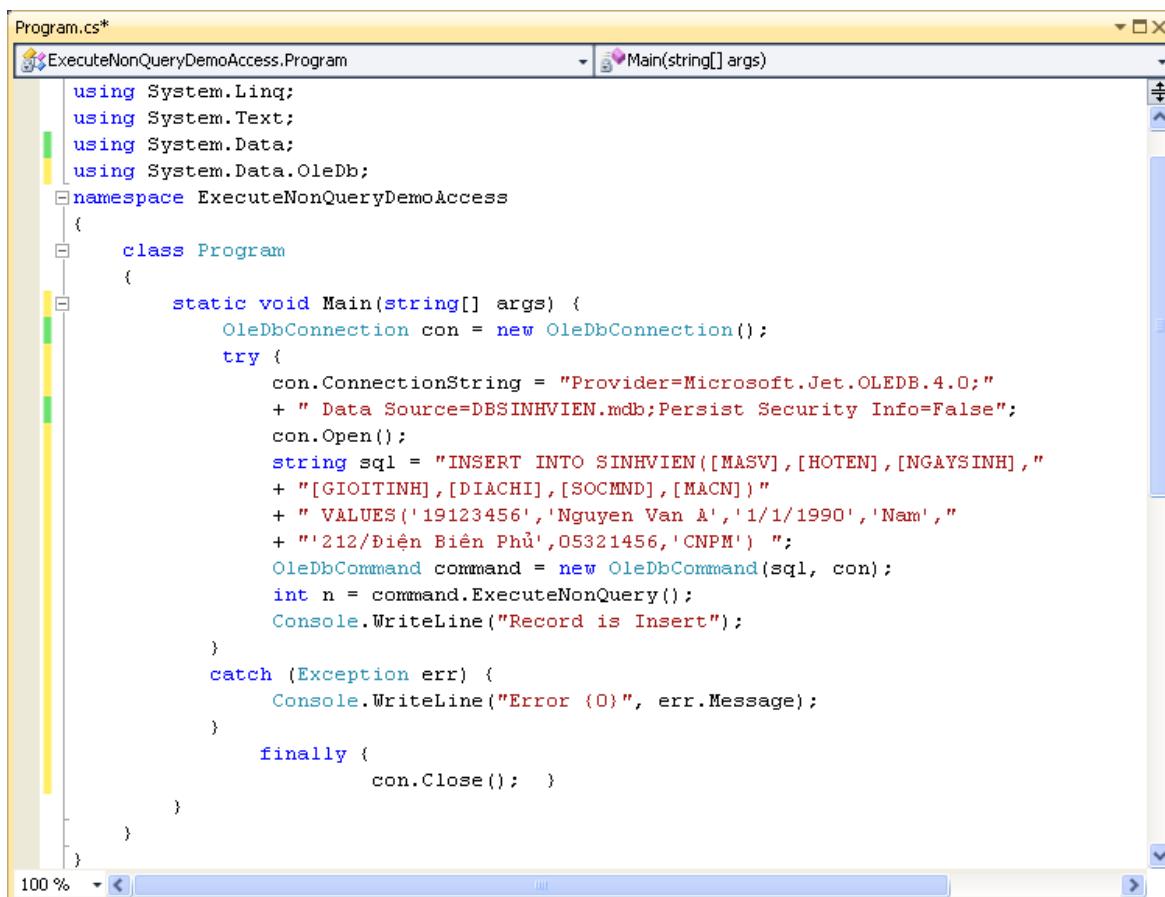
- b. Thuộc tính CommandTimeout
- c. Thuộc tính CommandType
- d. Thuộc tính Connection

5.2.2.2. Các phương thức của đối tượng command

a. Phương thức ExecuteNonQuery

Để thực thi phát biểu T-SQL (Transact-SQL) để thao tác dữ liệu như: INSERT, UPDATE, DELETE, CREATE, DROP, EXEC và SET, và trả về số bản ghi đã xử lý

Ví dụ 1: Kết nối và tạo đối tượng Command thực hiện lệnh INSERT trên cơ sở dữ liệu Access



```
Program.cs*
ExecuteNonQueryDemoAccess.Program Main(string[] args)
{
    using System.Linq;
    using System.Text;
    using System.Data;
    using System.Data.OleDb;
    namespace ExecuteNonQueryDemoAccess
    {
        class Program
        {
            static void Main(string[] args)
            {
                OleDbConnection con = new OleDbConnection();
                try {
                    con.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;" +
                        "Data Source=DBSINHVIEN.mdb;Persist Security Info=False";
                    con.Open();
                    string sql = "INSERT INTO SINHVIEN([MASV],[HOTEN],[NGAYSINH]," +
                        "[GIOITINH],[DIACHI],[SOCMND],[MACN])" +
                        "VALUES('19123456','Nguyen Van A','1/1/1990','Nam',"
                        "'212/Diện Biên Phủ',05321456,'CNPM') ";
                    OleDbCommand command = new OleDbCommand(sql, con);
                    int n = command.ExecuteNonQuery();
                    Console.WriteLine("Record is Insert");
                }
                catch (Exception err)
                {
                    Console.WriteLine("Error (0)", err.Message);
                }
                finally {
                    con.Close();
                }
            }
        }
    }
}
```

Hình 5.7. Kết nối và tạo đối tượng Command thực hiện lệnh INSERT
trên cơ sở dữ liệu Access

Ví dụ 2: Kết nối và tạo đối tượng Command thực hiện lệnh INSERT trên cơ sở dữ liệu SQL Server sử dụng thư viện SQLClient

The screenshot shows a Windows Form application window titled "ExecuteNonQueryDemoSQLServer.Program". The code in the main class "Program" contains a static void Main method that connects to a local SQL Server database, executes an INSERT query to add a new student record, and then outputs the message "Thêm mới sinh viên thành công" (Student added successfully) to the console.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data;
using System.Data.SqlClient;
namespace ExecuteNonQueryDemoSQLServer
{
    class Program
    {
        static void Main(string[] args)
        {
            SqlConnection con = new SqlConnection();
            try
            {
                con.ConnectionString = "Data Source=localhost; " +
                    "Initial Catalog=DBSINHVIEN;Integrated Security=True";
                con.Open();
                string sql = "INSERT INTO SINHVIEN([MASV],[HOTEN],[NGAYSINH], " +
                    "[GIOITINH],[DIACHI],[SOCMND],[MACN]) " +
                    "VALUES('191234587','Nguyen Van A', '1/1/1990', 'Nam', " +
                    "'212/Diện Biên Phủ', 05321456, 'CNPM') ";
                SqlCommand command = new SqlCommand(sql, con);
                int n = command.ExecuteNonQuery();
                Console.WriteLine("Thêm mới sinh viên thành công");
                Console.ReadKey();
            }
            catch (Exception err)
            {
                Console.WriteLine("Error {0}", err.Message);
            }
            finally
            {
                con.Close(); }
        }
    }
}
```

A separate window titled "file:///C:/ExecuteNonQueryDemoAcces..." shows the output of the console application: "Thêm mới sinh viên thành công".

Hình 5.8. Kết nối và tạo đối tượng Command thực hiện lệnh INSERT trên cơ sở dữ liệu SQL Server sử dụng thư viện SQLClient

Ví dụ 3: Kết nối và tạo đối tượng Command thực hiện lệnh INSERT trên cơ sở dữ liệu SQL Server sử dụng OleDb

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data;
using System.Data.OleDb;
namespace ExecuteNonQueryDemoSQLServer
{
    class Program
    {
        static void Main(string[] args)
        {
            OleDbConnection con = new OleDbConnection();
            try
            {
                con.ConnectionString = "Provider=SQLOLEDB;Data Source=.;"
                + "Integrated Security=SSPI;Initial Catalog=DBSINHVIEN";
                con.Open();
                string sql = "INSERT INTO SINHVIEN([MASV],[HOTEN],[NGAYSINH],"
                + "[GIOITINH],[DIACHI],[SOCMND],[MACN])"
                + " VALUES('1912345','Nguyen Van B','1/1/1990','Nam',"
                + "'212/Điện Biên Phú',05321456,'CNPM') ";
                OleDbCommand command = new OleDbCommand(sql, con);
                int n = command.ExecuteNonQuery();
                Console.WriteLine("Them moi sinh vien thanh cong");
                Console.ReadKey();
            }
            catch (Exception err)
            {
                Console.WriteLine("Error {0}", err.Message);
            }
            finally
            {
                con.Close(); }
        }
    }
}

```

Hình 5.9. Kết nối và tạo đối tượng Command thực hiện lệnh INSERT trên cơ sở dữ liệu SQL Server sử dụng OleDb

b. Phương thức ExecuteScalar

Phương thức ExecuteScalar thực thi và trả về giá trị đơn từ phát biểu SQL dạng SELECT chỉ có một cột và một hàng. Chẳng hạn, trong trường hợp muốn lấy tổng số sinh viên tồn tại trong bảng sinhviên, bạn khai báo như sau.

Ví dụ 1: Kết nối và tạo đối tượng Command thực hiện lệnh SELECT trả về một giá trị trên cơ sở dữ liệu Sql server

The screenshot shows a Windows Form application window titled "file:///C:/ExecuteNonQueryDemoAcce...". Inside the window, the text "Tong so sinh vien:1" is displayed, indicating the result of the SQL query executed in the code.

```

Program.cs
ExecuteNonQueryDemoSQLServer.Program
Main(string[] args)

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data;
using System.Data.SqlClient;
namespace ExecuteNonQueryDemoSQLServer
{
    class Program
    {
        static void Main(string[] args)
        {
            SqlConnection con = new SqlConnection();
            try
            {
                con.ConnectionString = "Data Source=localhost; " +
                    "Initial Catalog=DBSINHVIEN;Integrated Security=True";
                con.Open();
                string sql = "Select count(*) from SINHVIEN ";
                SqlCommand command = new SqlCommand(sql, con);
                int TSV = (int)command.ExecuteScalar();
                Console.WriteLine("Tong so sinh vien:" + TSV);
                Console.ReadKey();
            }
            catch (Exception err)
            {
                Console.WriteLine("Error (0)", err.Message);
            }
            finally
            {
                con.Close(); }
        }
    }
}

```

Hình 5.10. Kết nối và tạo đối tượng Command thực hiện lệnh SELECT trả về một giá trị trên cơ sở dữ liệu Sql server

c. Phương thức ExecuteReader

Phương thức ExecuteReader trả về một đối tượng DataReader. Bạn có thể sử dụng phương thức này để đọc dữ liệu từ cơ sở dữ liệu vào đối tượng DataReader (trình bày trong phần đối tượng DataReader)

Các thuộc tính của đối tượng command

Đối tượng command có một số thuộc tính thường sử dụng như CommandText, Connection, CommandTimeout...

- Thuộc tính CommandText cho phép bạn khai báo phát biểu SQL hay thủ tục truy cập cơ sở dữ liệu như sau:

- Thuộc tính Connection dùng để tham chiếu đến đối tượng kết nối cơ sở dữ liệu
- Thuộc tính CommandTimeout (tính bằng giây) là thời gian dùng để thực thi phát biểu SQL hay thủ tục khi truy cập cơ sở dữ liệu

Ví dụ:

Thay cho đoạn lệnh sau:

The screenshot shows a Visual Studio IDE window with the code editor open. The file is named 'Program.cs' and contains the following C# code:

```
using System.Linq;
using System.Text;
using System.Data;
using System.Data.OleDb;
namespace ExecuteNonQueryDemoSQLServer
{
    class Program
    {
        static void Main(string[] args)
        {
            OleDbConnection con = new OleDbConnection();
            try
            {
                con.ConnectionString = "Provider=SQLOLEDB;Data Source=.;"
                    + "Integrated Security=SSPI;Initial Catalog=DBSINHVIEN";
                con.Open();
                string sql = " select * from sinhvien";
                OleDbCommand command = new OleDbCommand(sql, con);
                command.CommandText = sql;
                command.Connection = con;
                command.CommandTimeout = 15;
                OleDbDataReader reader;
                reader = command.ExecuteReader();

                while (reader.Read())
                {
                    Console.WriteLine("Ho ten:(0), Ngay sinh:(1)", reader["Hoten"], reader["Ngaysinh"]);
                }
                reader.Close();
                Console.ReadKey();
            }
            catch (Exception err)
            {
                Console.WriteLine("Error (0)", err.Message);
            }
            finally
            {
                con.Close(); 
            }
        }
    }
}
```

To the right of the code editor, a terminal window displays the output of the program. The output shows three rows of data, each containing a name and a date:

```
file:///C:/ExecuteNonQueryDemoAccess/ExecuteNonQueryDemo...
Ho ten:Nguyen Van A, Ngay sinh:1/1/1990 12:00:00 AM
Ho ten:Nguyen Van B, Ngay sinh:1/1/1990 12:00:00 AM
Ho ten:Nguyen Van A, Ngay sinh:1/1/1990 12:00:00 AM
```

Hình 5.11. Ví dụ về phương thức **ExecuteReader**

5.2.3. Đối tượng data reader

Lớp data reader là lớp chỉ đọc (ReadOnly) và dữ liệu được đọc theo một chiều (Forward only). Lớp này rất thông dụng khi bạn cần tiết kiệm bộ nhớ

Bạn đọc dữ liệu từ cơ sở dữ liệu vào đối tượng data reader bằng phương thức ExecuteReader, của đối tượng command. Data reader chứa tập các bản ghi truy vấn từ một hay nhiều bảng trong cơ sở dữ liệu

.NET có các lớp data reader tương ứng với các loại trình điều khiển cơ sở dữ liệu, SqlDataReader sử dụng cho cơ sở dữ liệu SQL, OleDbDataReader sử dụng cho cơ sở dữ liệu OLE DB, OdbcDataReader sử dụng cho các nguồn dữ liệu ODBC, OracleDataReader hỗ trợ cơ sở dữ liệu Oracle và một số data reader khác như MySqlDataReader...

Tuy nhiên, trong khi sử dụng đối tượng data reader, kết nối cơ sở dữ liệu không thể thực hiện cho các thao tác khác. Trong trường hợp bạn cần sử dụng kết nối cho thao tác khác, bạn phải đóng đối tượng data reader

Đối tượng data reader không có khái niệm định hướng di chuyển bản ghi, không lưu trữ dữ liệu trong bộ nhớ. Do đó, khi bạn cần hiển thị kết quả ra màn hình từ cơ sở dữ liệu, đối tượng này là giải pháp tốt nhất

Các phương thức của đối tượng data reader

- Read: di chuyển và trả đến bản ghi kế tiếp, nếu hết bản ghi phương thức trả về false
- Close: đóng đối tượng data reader
- Các phương thức GetXXX(int index): để đọc dữ liệu từ trường thứ index của bản ghi hiện hành, ứng với các kiểu dữ liệu tương ứng. Chẳng hạn bạn muốn lấy giá trị là chuỗi bạn sử dụng phương thức GetString, tương tự như vậy đối với các kiểu dữ liệu khác như GetInt, GetInt16, GetBoolean, GetDateTime, GetDouble, GetDecimal... index là thứ tự trường, trường đầu tiên là trường thứ 0

Bạn sử dụng phương thức phù hợp với kiểu dữ liệu muốn lấy ra mà .NET hỗ trợ để cải thiện tốc độ đọc dữ liệu. Trong trường hợp lấy dữ liệu hiển thị mà không quan tâm đến loại dữ liệu, bạn có thể sử dụng phương thức GetString. Nếu sử dụng phương thức

này, hệ thống không quan tâm đến kiểu dữ liệu trong cột, mà chỉ lấy và hiển thị dữ liệu dạng chuỗi

Ngoài ra mỗi bản ghi từ data reader là một tập hợp dữ liệu của các trường, vì vậy có thể sử dụng cặp dấu ngoặc vuông để truy xuất dữ liệu từ tập hợp bởi chỉ số, hay tên trường

Ví dụ 1: Kết nối và hiển thị dữ liệu từ cơ sở dữ liệu SQL Server sử dụng đối tượng data reader

```
using System.Data;
using System.Data.SqlClient;
public class ExecuteNonQueryDemo {
    public static void Main(string[] args) {
        int count;
        void Page_Load(Object sender, EventArgs e) {
            SqlConnection cnn = new SqlConnection();
            try {
                cnn.ConnectionString = "server=.\SQLExpress; database=Northwind;
Trusted_connection=true";
                command.CommandText = query;
                command.Connection = con;
                command.CommandTimeout = 15;
                cmd = new SqlCommand("Select * From Sinhvien", conn);
                SqlDataReader reader;
                conn.Open();
                reader = cmd.ExecuteReader();
                while (reader.Read())

```

```
        Console.WriteLine(reader["Hoten"]);

    reader.Close();

    conn.Close();

}

catch (Exception err) {

    Console.WriteLine("Error {0}", err.Message);

}

finally {

    con.Close();

}

}

}

}
```

5.2.4. Đối tượng data adapter

Để truy cập cơ sở dữ liệu thông qua ADO.NET sử dụng đối tượng data adapter, thực hiện các bước sau:

Tạo đối tượng kết nối connection để kết nối với nguồn dữ liệu vật lý như cơ sở dữ liệu SQL Server, Access, Oracle, MySQL, XML...

- Tạo đối tượng data adapter
- Đối tượng OleDbConnection để kết nối với cơ sở dữ liệu của Microsoft Access hay MySQL. Đối tượng OleDbDataAdapter để thực hiện lệnh và trả về dữ liệu
- Đối tượng SqlConnection để kết nối cơ sở dữ liệu của Microsoft SQL Server. Đối tượng SqlCommand để thực hiện lệnh và trả về dữ liệu
- Đối tượng OracleConnection để kết nối cơ sở dữ liệu của Oracle. Đối tượng OracleDataAdapter để thực hiện lệnh và trả về dữ liệu
- Tạo đối tượng data set sử dụng đối tượng adapter

- Sử dụng đối tượng data set để hiển thị dữ liệu trong cơ sở dữ liệu bằng cách duyệt data set hay kỹ thuật data binding
- Cập nhật cơ sở dữ liệu từ đối tượng data set sử dụng đối tượng adapter
- Đóng kết nối cơ sở dữ liệu

Ví dụ: Tạo các đối tượng dữ liệu và hiển thị dữ liệu từ cơ sở dữ liệu Microsoft SQL Server

```
void Page_Load(object sender, EventArgs e)
{
    // (1) Tạo đối tượng connection.
    string strcon =
"server=(local);uid=sa;pwd=;database=QuanLySinhVien";
    SqlConnection sqlCon = new SqlConnection(strcon);

    // (2) Tạo đối tượng data adapter.
    SqlDataAdapter dataAdapter = new SqlDataAdapter("select *
from Sinhvien",sqlCon);

    // (3) Tạo đối tượng data set.
    DataSet dataSet = new DataSet();

    // (4) Điền dữ liệu vào data set.
    dataAdapter.Fill(dataSet, "mathang");

    // (5) Duyệt dữ liệu trong data set, xem thêm phần Data
    set
}
```

ADO.NET sử dụng các thuộc tính của đối tượng adapter là InsertCommand, DeleteCommand và UpdateCommand để cập nhật cơ sở dữ liệu từ data set. Khi bạn tạo đối tượng, cần thiết lập các thuộc tính này trước khi có thể gọi phương thức Update của đối tượng adapter

Để thiết lập các thuộc tính InsertCommand, DeleteCommand, và UpdateCommand, thực hiện các bước sau:

Thiết lập thuộc tính SelectCommand của đối tượng adapter bởi phương thức khởi tạo của đối tượng data adapter hay sử dụng thuộc tính SelectCommand. ADO.NET sử dụng thuộc tính CommandText của SelectCommand để thiết lập các thuộc tính InsertCommand, DeleteCommand, và UpdateCommand

Gọi phương thức Update của đối tượng adapter để thực hiện các phát biểu SQL Insert, Delete, Update đã được thiết lập bởi các thuộc tính InsertCommand, DeleteCommand hay UpdateCommand

Đối tượng data adapter dùng để truy cập vào cơ sở dữ liệu, sau đó điền vào bộ nhớ thường trú data set. Khi dữ liệu trong đối tượng data set có thay đổi chúng ta yêu cầu đối tượng data adapter kết nối trở lại cơ sở dữ liệu và cập nhật dữ liệu dựa trên các thay đổi được thực hiện trong data set

Có nhiều loại data adapter ứng với các đối tượng connection như: SqlDataAdapter, OleDbDataAdapter, OdbcDataAdapter, OracleDataAdapter, MySQLDataAdapter... Những đối tượng này cung cấp 4 thuộc tính (SelectCommand, InsertCommand, UpdateCommand, DeleteCommand) định nghĩa 4 phát biểu SQL tương ứng như: Select, Insert, Update, Delete dùng để thao tác dữ liệu trên đối tượng DataSet

Đối tượng data adapter bao gồm các phương thức như sau:

- Fill: thực thi lệnh SelectCommand và điền dữ liệu vào đối tượng data set từ dữ liệu nguồn
- FillSchema: sử dụng lệnh SelectCommand để trích dữ liệu trong table và tạo ra một table rỗng trong đối tượng data set
- Update: gọi đối tượng Command để thực thi 4 phát biểu SQL tương ứng trong thuộc tính SelectCommand, InsertCommand, UpdateCommand, DeleteCommand trên đối tượng data set

Ví dụ 1:

```
void Page_Load(object sender, EventArgs e)
{
    // Tạo đối tượng connection.

    string strcon =
"server=(local);uid=sa;pwd=;database=banhang";

    SqlConnection sqlCon = new SqlConnection(strcon);

    // Tạo đối tượng data adapter.

    SqlDataAdapter dataAdapter = new SqlDataAdapter();
    dataAdapter.SelectCommand = new SqlCommand("select * from
mathang",
sqlCon);

    // Tạo đối tượng data set.

    DataSet dataSet = new DataSet();

    // Điền dữ liệu vào data set.

    dataAdapter.Fill(dataSet, "mathang");

    // Duyệt dữ liệu trong data set
}
```

ADO.NET hỗ trợ truyền tham đối vào lệnh trong biểu thức SQL, với các tham đối truyền trong câu lệnh SQL có dấu @ đi trước. Chú ý, mỗi lần thực hiện xong một lệnh, bạn phải truyền lại CommandText nếu không lệnh sẽ không có hiệu lực

Chú ý:

Với OleDbDataAdapter và OdbcDataAdapter, câu lệnh SQL là:

```
string selectSQL = "select * from mathang where mahang = ?";  
string insertSQL = "insert into mathang (mahang, tenhang)  
values(?, ?);"
```

```
string updateSQL = "update mathang set mahang=?, tenhang=? where  
mahang=?";  
  
string deleteSQL = "delete from mathang where mahang = ?";
```

Ví dụ 2:

```
void Page_Load(object sender, EventArgs e)  
{  
    string strcon =  
"server=(local);uid=sa;pwd=;database=banhang";  
    SqlConnection sqlCon = new SqlConnection(strcon);  
    SqlDataAdapter dataAdapter = new SqlDataAdapter("select *  
from mathang",  
        sqlCon);  
    dataAdpater.UpdateCommand = new SqlCommand(  
        "update mathang set dongia = @dongia where mahang =  
@mahang", sqlCon);  
    dataAdpater.UpdateCommand.Parameters.Add(  
        "@mahang", SqlDbType.VarChar, 15, "mahang").Value =  
"03";  
    dataAdpater.UpdateCommand.Parameters.Add("@dongia",  
SqlDbType.Int).Value =  
    220000;  
    DataSet dataSet = new DataSet();  
    dataAdapter.Fill(dataSet, "mathang");  
    DataRow row = dataSet.Tables["mathang"].Rows[0];  
    row["tenhang"] = "Ao thun";  
    dataAdpater.Update(dataSet, "mathang");  
    // Duyệt dữ liệu trong data set  
}
```

5.2.5. Đối tượng DataSet

Đối tượng DataSet là một đối tượng của ADO.NET để trình bày và thao tác dữ liệu. Lưu ý rằng, trong ASP.NET, DataSet được xem là ảnh của một cơ sở dữ liệu thu nhỏ có khả năng hoạt động độc lập khi ngắt kết nối với dữ liệu nguồn. Một DataSet có thể chứa tất cả các bảng (DataTable), View (DataView), quan hệ (Relation), ràng buộc (Constraints) như một cơ sở dữ liệu thật sự. Duyệt dữ liệu trong DataSet như sau:

Ví dụ 1:

```
void Page_Load(object sender, EventArgs e)
{
    string strcon =
"server=(local);uid=sa;pwd=;database=QuanlySinhvien";

    SqlConnection sqlCon = new SqlConnection(strcon);

    SqlDataAdapter dataAdapter = new SqlDataAdapter("select *
from Sinhvien",
sqlCon);

    dataAdpater.UpdateCommand = new SqlCommand(
        "update sinhvien set Hoten = @Hoten where MaSV =
@Masv", sqlCon);

    dataAdpater.UpdateCommand.Parameters.Add(
        "@Hoten", SqlDbType.VarChar, 15, "Sinh vien").Value =
"19233455";

    DataSet dataSet = new DataSet();

    dataAdapter.Fill(dataSet, "Sinhvien");

    DataTable dt = dataSet.Tables["sinhvien"];
```

```
foreach(DataRow row in dt.Rows)
{
    foreach(DataColumn col in dt.Columns)
        Console.WriteLine.Write(row[col]);
}
```

Ví dụ 2:

DataSet có thể chứa nhiều bảng dữ liệu vì thế khi xử lý bảng cần nào cần phải gán vào DataTable một tên bảng cụ thể. Và cuối cùng dùng DataColumn và DataRow để xử lý DataTable đó

```
SqlDataAdapter da = new SqlDataAdapter(sql, conn);
DataSet ds= new DataSet();
da.Fill(ds, "sinhvien");
DataTable dt = ds.Tables["sinhvien"];
foreach(DataRow row in dt.Rows)
{
    foreach(DataColumn col in dt.Columns)
        Console.WriteLine(row[col]);
}
```

Ví dụ 3:

Phân loại và sắp xếp trong DataSet

```
SqlConnection conn = new SqlConnection(connectionString);
string sql1=@"select * from masv";
string sql2 = @"select * from sinhvien where Masv = '192345';
string sql = sql1 + sql2;
```

```
SqlDataAdapter da = new SqlDataAdapter();  
da.SelectCommand = new SqlCommand(sql, conn);  
DataSet ds = new DataSet();  
da.Fill(ds, "sinhvien");  
DataTableCollection dtc = ds.Tables;  
//Thiết lập tiêu chuẩn lọc  
string filter = "Masv='192345'";  
//Thiết lập tiêu chuẩn sắp xếp  
string sort="Hoten ASC"  
foreach (DataRow row in dtc["sinhvien"].Select(filter, sort))  
{  
    Console.WriteLine(row["Masv"].ToString());  
    Console.WriteLine(row["Hoten"]);  
}
```

Ví dụ 4:

```
//Chỉnh sửa  
da.Fill(ds, "sinhvien");  
DataTable dt = ds.Tables["Hoten"];  
dt.Columns["tenhang"].AllowDBNull = true ;  
dt.Rows[0]["Hoten"] = "Le B";  
//Thêm một dòng mới  
DataRow newRow = dt.NewRow();  
newRow["Masv"] = "1923456";  
newRow["Hoten"] = "Le C"; //...
```

```
foreach (DataRow row in dt.Rows)
{
    row[“Masv”].ToString();
    row[“Hoten”].ToString();
}
```

5.2.6. Đôi tượng DataTable

Lớp DataTable dùng để xử lý dữ liệu của bảng trong tập hợp Tables. Lớp DataTable là một phần của không gian tên System.Data, và không có các lớp con

Lưu ý rằng, khi làm việc với đôi tượng DataTable, bạn sử dụng hai đôi tượng liên quan, DataRow và DataColumn là nền tảng tạo nên DataTable. Đôi tượng DataRow sử dụng để trình bày một hàng trong lớp DataTable. Đôi tượng DataColumn trình bày một cột.

Một DataTable có thể định nghĩa một khóa chính, bao gồm một hoặc nhiều cột, và cũng có thể chứa các ràng buộc của các cột.

Bảng một số thuộc tính của lớp DataTable

Thuộc tính	Điễn giải
Columns	Columns là thuộc tính chỉ đọc trả về tập DataColumnCollection của cột trong bảng nếu không tồn tại cột nào trong bảng thì trả về giá trị null
DataSet	Thuộc tính trả về tập dữ liệu trong bảng, điều này có nghĩa là trả về đối tượng DataSet
DefaultView	Thuộc tính trả về giá trị DataView mà nó có thể tùy chỉnh khung nhìn của bảng. Khung nhìn dữ liệu trả về bảng dữ liệu có thể lọc, sắp xếp, tìm kiếm
Primarykey	Thuộc tính cho phép đọc ghi, giữ liệu trả về hay gán mảng đối tượng

	DataColumn dùng cho khoá chính trong bảng dữ liệu. Ngoại lệ phát sinh nếu các cột này trong bảng là khoá ngoại
Rows	Rows là thuộc tính chỉ đọc trả về tập đối tượng DataColumnCollection lưu tất cả các đối tượng DataRow tạo nên dữ liệu trong bảng, nếu không tồn tại hàng nào trong bảng, giá trị trả về là null
TableName	Tên của bảng dữ liệu, thuộc tính TableName có kiểu giá trị string được sử dụng khi tìm kiếm tên bảng trong tập Table

Bảng một số phương thức của lớp DataTable

Phương thức	Điễn giải
void Clear()	Xoá tất cả các hàng dữ liệu trong bảng, nếu có ràng buộc quan hệ với bảng khác thì ngoại lệ sẽ phát sinh, trừ khi quan hệ có khai báo Cascade delete
object Compute (string expression, string filter)	Tính toán biểu thức expression trên các hàng hiện tại thỏa mãn điều kiện lọc filter. expression phải chứa đựng các hàm COUNT hay SUM
void Reset()	Xác lập lại đối tượng DataTable về trạng thái ban đầu
DataRow NewRow()	Tạo mới một đối tượng DataRow cùng lược đồ trong DataTable. Nghĩa là, một hàng mới phải có các cột như trong cấu trúc của bảng dữ liệu

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, xác định cột là chỉ đọc.... Khi tạo một cột, tốt hơn hết là nên đặt cho nó một cái tên; nếu không, sẽ tự động phát sinh một cái tên theo định dạng Columnn.

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. Các cột dữ liệu có thể được tạo để lưu giữ các kiểu dữ liệu của .NET Framework sau: Boolean, Char, Byte, Sbyte, Short, Int16,

Int32, Int64, UInt16, UInt32, UInt64, Single, Double, Decimal, String, DateTime, TimeSpan

Một khi đã được tạo, bước tiếp theo là thiết lập các thuộc tính khác cho đối tượng DataColumn, chẳng hạn như tính khả năng nullability, giá trị mặc định.

Các thuộc tính sau có thể được cài đặt trong một DataColumn:

Thuộc tính	Điễn giải
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 việc tính toán trên cột này

Khi sử dụng đối tượng DataTable, bạn cần kết hợp với hai đối tượng khác là DataView và DataSet.

Ví dụ 1: Tạo một bảng dữ liệu có tên mathang và định nghĩa các cột mahang, tenhang, mota, dongia

```
void Page_Load(object sender, EventArgs e){  
  
    DataTable dataTable = new DataTable ("Sinhvien");  
  
    //Định nghĩa cấu trúc Table  
  
    DataColumn dataColumn = new DataColumn();
```

```
//Khai báo kiểu dữ liệu cột thứ nhất  
  
dataColumn.DataType = System.Type.GetType ("System.String");  
  
dataColumn.ColumnName = "MaSV";  
  
dataColumn.ReadOnly = true;  
  
dataColumn.Unique = true;  
  
dataColumn.AllowDBNull = false;  
  
// Thêm cột mahang vào đối tượng DataTable  
  
dataTable.Columns.Add (dataColumn);  
  
//Định nghĩa cột dữ liệu thứ hai  
  
dataColumn = new DataColumn();  
  
dataColumn.DataType= System.Type.GetType ("System.String");  
  
dataColumn.ColumnName = "Hoten";  
  
dataColumn.AutoIncrement = false;  
  
dataColumn.Caption = "Ho va Ten";  
  
dataColumn.ReadOnly = false;  
  
dataColumn.Unique = false;  
  
dataTable.Columns.Add (dataColumn);  
  
//Định nghĩa cột dữ liệu thứ ba  
  
dataColumn = new DataColumn();  
  
dataColumn.DataType= System.Type.GetType  
("System.DateTime");  
  
dataColumn.ColumnName = "Ngaysinh";  
  
dataColumn.AutoIncrement = false;  
  
dataTable.Columns.Add(dataColumn);  
  
dataColumn = new DataColumn();
```

```
dataColumn.DataType = System.Type.GetType("System.Int32");

dataColumn.ColumnName = "Namnhaphoc";

dataColumn.Caption = "Nam Nhập Học";

dataColumn.ReadOnly = false;

dataColumn.Unique = false;

dataTable.Columns.Add (dataColumn);

//Khai báo khoá chính cho cột dữ liệu thứ nhất

DataColumn[] primaryKeyColumns= new DataColumn[1];

primaryKeyColumns[0] = dataTable.Columns["Masv"];

dataTable.PrimaryKey = primaryKeyColumns;

//Thêm đối tượng DataTable vào đối tượng DataSet

DataSet dataSet = new DataSet();

dataSet.Tables.Add(dataTable);

//Dùng đối tượng DataRow, DataColumn

//và DataTable để thêm 4 bản ghi vào bảng

DataRow dataRow;

int i;

for (i=1;i<=3;i++) {

    //Khai báo thêm bản ghi vào bảng dữ liệu

    dataRow = dataTable.NewRow();

    dataRow["Masv"] ="19234567";

    dataRow["Hoten"] = "Le D ";

    dataRow["Ngaysinh"] = DateTime.Parse("1/1/1991");

    dataRow["Namnhaphoc"] = 2005;

    //Thêm bản ghi vào bảng dữ liệu
```

```
        dataTable.Rows.Add (dataRow);

        foreach(DataRow row in dataTable.Rows)

        {

            foreach(DataColumn col in dataTable.Columns)

                Console.WriteLine(row[col]);

        }

    }
```

5.2.7. Đổi tượng DataView

Đổi tượng DataView cho phép bạn trình bày tập dữ liệu con từ đối tượng DataTable, hay nhận toàn bộ dữ liệu của đối tượng DataTable nhưng hiển thị dưới các góc độ khác nhau

Lớp DataView cho phép tạo một khung nhìn khác so với thuộc tính DefaultView trong đối thoại DataTable. Bạn có thể sử dụng lớp này để đọc (filtering), sắp xếp (sorting), tìm kiếm (searching) và điều hướng (navigation) trong DataTable.

Trong thực tế, bạn có thể tạo nhiều đối tượng DataView trên đối tượng DataTable mà bạn muốn. Bởi vì DataView là lớp cho phép bạn trình bày dữ liệu dưới nhiều cách nhìn, kết quả mong muốn hay tiêu chuẩn chọn lọc trước dựa trên DataTable. Giống như hai lớp DataTable và DataSet, lớp DataView không có lớp con

Bảng thuộc tính của lớp DataView

Thuộc tính	Diễn giải
AllowDelete	Thuộc tính này trả về hay gán giá trị có kiểu dữ liệu bool, cho phép xoá hàng trên DataView hay DataTable hay không
AllowEdit	Thuộc tính này trả về hay gán giá trị kiểu dữ liệu bool, cho phép chỉnh sửa hàng trên DataView hay DataTable hay không
AllowNew	Thuộc tính này trả về hay gán giá trị có kiểu dữ liệu bool, cho

	phép thêm mới hàng trên DataView hay DataTable hay không
ApplyDefaultSort	Thuộc tính này trả về hay gán giá trị có kiểu bool, cho phép thuộc tính sắp xếp có hiệu lực hay không
Count	Thuộc tính chỉ đọc giá trị chỉ số hàng trong DataView. Thuộc tính này không có hiệu lực khi sử dụng thuộc tính RowFilter và RowStateFilter
RowStateFilter	Thuộc tính này trả về hay gán trạng thái lọc dữ liệu được sử dụng trong DataView. Có nghĩa là bạn lọc các hàng dựa trên trạng thái như Added, Deleted, Unchanged...
RowFilter	Thuộc tính kiểu string, xác định biểu thức điều kiện lọc các hàng trong DataView
Sort	Kiểu string, xác định cột sắp xếp giảm dần hay tăng dần bằng cách sử dụng từ khóa ASC (tăng dần) hay DESC (giảm dần) sau mỗi tên trường
Table	Thuộc tính này trả về hay gán DataTable, điều này có nghĩa là DataView cung cấp dữ liệu từ bảng dữ liệu

Bảng phương thức của lớp DataView

Phương thức	Diễn giải
DataRowView AddNew()	Thêm một hàng vào DataView , giá trị trả về có kiểu dữ liệu DataRowView. Chú ý rằng, khi gọi phương thức này, thuộc tính AllowNew có giá trị true, nếu false ngoại lệ phát sinh

void CopyTo (objtec [] array, int index)	Sao chép các hàng trong DataView vào mảng array từ chỉ mục index
Delete (int intIndex)	Xoá một hàng tại chỉ mục intIndex
int Find(object key) int Find(object[] key)	Tìm chỉ mục của hàng trong DataView có giá trị khóa chính bằng các giá trị key
DataRowView FindRow(object key)	Trả về một mảng các đối tượng DataRowView từ DataView có giá trị khóa chính bằng các giá trị key
DataRowView FindRow(object key)	

Ví dụ: Sử dụng DataView để lọc các bản ghi trong DataTable

```
void Page_Load(object sender, EventArgs e){
    //Khai báo và khởi tạo đối tượng DataTable
    DataTable dataTable = new DataTable ("Sinhvien");
    //Định nghĩa cấu trúc Table
    //Khai báo kiểu dữ liệu cột thứ nhất
    DataColumn dataColumn = new DataColumn();
    dataColumn.DataType = System.Type.GetType ("System.String");
    dataColumn.ColumnName = "Masv";
    dataColumn.ReadOnly = true;
    dataColumn.Unique = true;
    dataTable.Columns.Add(dataColumn);
    //Định nghĩa cột dữ liệu thứ hai
    dataColumn = new DataColumn();
    dataColumn.DataType= System.Type.GetType ("System.String");
    dataColumn.ColumnName = "Hoten";
```

```
dataColumn.AutoIncrement = false;
dataColumn.Caption = "tenhang";
dataColumn.ReadOnly = false;
dataColumn.Unique = false;
dataTable.Columns.Add (dataColumn);
//Định nghĩa cột dữ liệu thứ ba
dataColumn = new DataColumn();
dataColumn.DataType= System.Type.GetType ("System.DateTime");
dataColumn.ColumnName = "Ngaysinh";
dataColumn.AutoIncrement = false;
dataTable.Columns.Add(dataColumn);
//Định nghĩa cột dữ liệu thứ tư
dataColumn = new DataColumn();
dataColumn.DataType = System.Type.GetType("System.Int32");
dataColumn.ColumnName = "Namnhaphoc";
dataColumn.Caption = "Nam nhap hoc";
dataColumn.ReadOnly = false;
dataColumn.Unique = false;
dataTable.Columns.Add (dataColumn);
//Khai báo khoá chính cho cột dữ liệu thứ nhất
DataColumn[] primaryKeyColumns= new DataColumn[1];
primaryKeyColumns[0] = dataTable.Columns["Masv"];
dataTable.PrimaryKey = primaryKeyColumns;
//Thêm đối tượng DataTable vào đối tượng DataSet
DataSet dataSet = new DataSet();
dataSet.Tables.Add(dataTable);
//Dùng đối tượng DataRow, DataColumn và DataTable để thêm 4
bản ghi vào bảng
//Khai báo đối tượng DataRow
DataRow dataRow;
int i;
```

```
for (i=1;i<=3;i++) {  
    //Khai báo thêm bản ghi vào bảng dữ liệu  
    dataRow = dataTable.NewRow();  
    dataRow[ "MaSV" ] = i.ToString();  
    dataRow[ "tenhang" ] = "Le Van " + i.ToString();  
    dataRow[ "Ngaysinh" ] = DateTime.pares("1/1/1990");  
    dataRow[ "Namnhaphoc" ]  
=DateTime.parse("1/1/1990").YearAdd(i);  
    //Thêm bản ghi vào bảng dữ liệu  
    dataTable.Rows.Add (dataRow);  
}  
//Khai báo một đối tượng DataView với thuộc tính  
//RowFilter và Sort để lọc dữ liệu  
DataView dataView = new DataView(dataTable);  
dataView.RowFilter = "Namnhaphoc>2005";  
dataView.Sort = "Hoten ASC";  
}
```

5.2.8. Đối tượng DataRelation

Đối tượng DataRelation dùng để biểu diễn mối quan hệ một – nhiều (cha/ con) giữa hai đối tượng DataTable với nhau. Điều này được thực hiện bằng mối liên kết thông qua hai đối tượng DataColumn tương ứng với nhau của hai đối tượng DataTable

Trong DataSet (có một hay nhiều DataTable, DataView), bạn có thể truy xuất tất cả DataRelation thông qua thuộc tính Relations của DataSet

Khi thiết lập một đối tượng quan hệ DataRelation trong đối tượng DataSet, dữ liệu trong các cột liên kết của hai đối tượng DataTable cha và DataTable con phải được kiểm tra trước khi mối quan hệ tạo ra, nếu dữ liệu của một trong hai cột tương ứng của DataTable trong đối tượng không thỏa các ràng buộc, ngoại lệ sẽ phát sinh. Ngoại lệ phát sinh do đối tượng quan hệ tạo ra bao gồm một trong hai loại sau:

- ArgumentNullException: phát sinh khi một trong hai cột có giá trị null

- InvalidConstraintException: phát sinh khi hai đối tượng DataTable quan hệ với nhau không cùng nằm trong một DataSet hay kiểu dữ liệu của hai cột trong DataTable khác nhau

Khi tạo đối tượng DataRelation, trước tiên bạn xác định hai cột dữ liệu có quan hệ cha con (parent/ child) với nhau của hai đối tượng DataTable trong đối tượng DataSet.

Bảng thuộc tính của đối tượng DataRelation

Thuộc tính	Ý nghĩa
ChildColumns	Thuộc tính chỉ đọc, trả về đối tượng DataColumn là các cột khóa ngoại của mối quan hệ
ChildKeyConstraint	Thuộc tính chỉ đọc, trả về đối tượng ForeignKeyConstraint là ràng buộc khóa ngoại của mối quan hệ
ChildTable	Thuộc tính chỉ đọc, trả về đối tượng DataTable là bảng phía bên nhiều (bảng con) của mối quan hệ
DataSet	Thuộc tính chỉ đọc, trả về DataSet chứa mối quan hệ
ParentColumns	Thuộc tính chỉ đọc, trả về một mảng các đối tượng DataColumn là cột khóa chính (cột cha) của mối quan hệ
ParentKeyConstraint	Thuộc tính chỉ đọc, trả về đối tượng UniqueConstraint ràng buộc các giá trị trong cột khóa chính là duy nhất
ParentTable	Thuộc tính chỉ đọc trả về đối tượng DataTable chỉ bảng cha là bảng bên 1 của mối quan hệ
RelationName	Gán hay trả về tên quan hệ sử dụng để nhận một đối tượng DataRelation từ tập quan hệ DataRelationCollection

Ví dụ:

```

using System;
using System.Data;
using System.Data.SqlClient;
public class _Default
{
    void Page_Load(object sender, EventArgs e)

```

```
{  
    string strcon =  
"server=(local);uid=sa;pwd=;database=Quanlysinhvien";  
  
    SqlConnection sqlCon = new SqlConnection(strcon);  
  
    SqlDataAdapter dataAdapter = new SqlDataAdapter("select  
* from Lop", sqlCon);  
  
    DataSet dataSet = new DataSet();  
  
    dataAdapter.Fill(dataSet, "Lop");  
  
    SqlDataAdapter mathangAdapter = new SqlDataAdapter(  
  
        "SELECT * FROM sinhvien", sqlCon);  
  
    mathangAdapter.Fill(dataSet, "sinhvien");  
  
    DataColumn parentColumn =  
dataSet.Tables["Lop"].Columns["malop"];  
  
    DataColumn childColumn =  
dataSet.Tables["Sinhvien"].Columns["malop"];  
  
    DataRelation relation = new DataRelation("Lop-sinhvien",  
  
        parentColumn, childColumn);  
  
    dataSet.Relations.Add(relation);  
  
    Response.Write("Mỗi quan hệ giữa hai bảng đã được tạo "  
+ relation.RelationName);  
  
    //Duyệt qua các Table  
  
    foreach (DataRow loaihangRow in  
relation.ParentTable.Rows)  
  
    {  
  
        Response.Write("<p>Mã Lớp: " + loaihangRow["malop"]+  
"</p>");
```

```
        foreach (DataRow sinhvienRow in  
lopRow.GetChildRows(relation))  
  
        {  
  
            Console.WriteLine("Masv: {0}",  
mathangRow[ "Masv"]);  
  
            Console.WriteLine("Hoten: {0}",  
mathangRow[ "Hoten"]);  
  
        }  
  
    }  
  
}
```

5.2.9. Ràng buộc dữ liệu vào các điều khiển

5.2.9.1. Ràng buộc dữ liệu vào ListBox

Để ràng buộc dữ liệu vào ListBox, thiết lập thuộc tính DataSource của ListBox là nguồn dữ liệu như DataTable, DataSet, thuộc tính DisplayMember là tên trường dữ liệu hiển thị trong ListBox

Ví dụ: Hiển thị tên hàng vào ListBox

```
private void Form1_Load(object sender, EventArgs e) {  
  
    //Tạo đối tượng SqlDataAdapter  
  
    string strcon="server=(local);uid=sa;pwd=;database=banhang";  
  
    SqlDataAdapter dataAdapter = new SqlDataAdapter();  
  
    dataAdapter.SelectCommand = new SqlCommand("select * from  
sinhvien", strcon);  
  
    //Tạo đối tượng DataTable  
  
    DataTable dataTable = new DataTable();
```

```
//Điền dữ liệu vào DataTable  
  
dataAdapter.Fill(dataTable);  
  
//Lọc dữ liệu  
  
dataTable.DefaultView.RowFilter = "ngaysinh > 1/1/1999";  
  
//Sắp xếp dữ liệu  
  
dataTable.DefaultView.Sort = "Hoten";  
  
//Hiển thị dữ liệu vào ListBox  
  
listBox1.DataSource = table;  
  
listBox1.DisplayMember = "Hoten";  
  
}
```

5.2.9.2. Ràng buộc dữ liệu vào các điều khiển khác

```
private void Form1_Load(object sender, EventArgs e) {  
  
    //  
  
    //Tạo đối tượng SqlDataAdapter  
  
    string strcon="server=(local);uid=sa;pwd=;database=banhang";  
  
    //SqlDataAdapter dataAdapter=new SqlDataAdapter("select *  
from Sinhvien", strcon);  
  
    SqlDataAdapter dataAdapter = new SqlDataAdapter();  
  
    dataAdapter.SelectCommand = new SqlCommand("select *  
from sinhvien", strcon);  
  
    //Tạo đối tượng DataTable  
  
    DataTable dataTable = new DataTable();  
  
    //Điền dữ liệu vào DataTable  
  
    dataAdapter.Fill(dataTable);  
  
    //Ràng buộc dữ liệu trong DataTable với các Textbox
```

```
        txtMasv.DataBindings.Add("Text", dataTable,
    "Masv");

        txtHoten.DataBindings.Add("Text", dataTable,
    "Hoten");

        txtNgaysinh.DataBindings.Add("Text", dataTable,
    "ngaysinh");

        txtNamnhaphoc.DataBindings.Add("Text", dataTable,
    "namnhaphoc");

    }
```

5.2.9.3. Ràng buộc dữ liệu vào DataGridView

Điều khiển DataGridView là điều khiển có thể ràng buộc dữ liệu, có thể thực hiện các thao tác như sắp xếp, lọc, kiểm tra dữ liệu hợp lệ, hiển thị và cập nhật bản ghi, và xử lý lỗi

- Ràng buộc dữ liệu vào DataGridView từ DataTable:

```
dataGridView1.DataSource = dataTable;
```

Ví dụ:

```
private void Form1_Load(object sender, EventArgs e) {

    //Tạo đối tượng SqlDataAdapter
    string
strcon="server=(local);uid=sa;pwd=;database=Sinhvien";

    SqlDataAdapter dataAdapter = new SqlDataAdapter();

    dataAdapter.SelectCommand = new SqlCommand("select * from
sinhvien", strcon);

    //Tạo đối tượng DataTable
    DataTable dataTable = new DataTable();
    //Điền dữ liệu vào DataTable
```

```
        dataAdapter.Fill(dataTable);

        //Hiển thị dữ liệu vào DataGridView

        dataGridView1.DataSource = dataTable;

    }
```

- DataGridView chỉ có thể hiển thị dữ liệu từ một DataTable, có thể ràng buộc dữ liệu từ một DataSet như sau:

```
        dataGridView1.DataSource = dataSet1;

        dataGridView1.DataMember = "sinhvien";
```

- Xử lý dữ liệu từng cột trong DataTable:

```
        dataGridView1.Columns["mahang"].ReadOnly = true;

        - DataGridView tự động hiển thị mô tả ngắn khi di chuyển mouse đến ô, có thể tắt mô tả ngắn như sau:

            dataGridView1.Columns["mahang"].ShowCellToolTips = false;

        - DataGridView hỗ trợ tự động sắp xếp.

        - DataGridView cho phép chỉnh sửa dữ liệu trong các ô, bằng cách kích đúp trong ô, hay ấn F2. Có thể xóa hàng, bằng cách hàng, ấn phím delete, và chèn hàng mới bằng cách gõ dữ liệu vào hàng trống
```

```
        dataGridView1.AllowUserToAddRows = false, // không chèn mới
        dataGridView1.AllowUserToDeleteRows = false, // không xóa
```

```
        dataGridView1EditMode=dataGridViewEditMode.EditProgrammatic
ally;
```

- Cấu trúc DataTable:

- Thuộc tính Columns là tập hợp các đối tượng DataGridViewColumn. DataGridViewColumn để cấu hình định dạng cột, tiêu đề cột, thay kích thước, sắp xếp, hay che dấu cột chỉ rõ

- Thuộc tính Rows là tập hợp các DataGridViewRow objects. DataGridViewRow và DataGridViewCell để nhận dữ liệu. Khi chỉnh sửa dữ liệu trong ô, sự kiện DataGridView change phát sinh, và dữ liệu nguồn được chỉnh sửa

Ví dụ: hiển thị mã hàng của mặt hàng có đơn giá lớn hơn 100

```
foreach (DataGridViewRow row in dataGridView1.Rows)
{
    if (!row.IsNewRow && Convert.ToInt32
        (row.Cells["dongia"].Value) > 100)
    {
        Console.WriteLine(row.Cells["mahang"].Value);
    }
}
```

Thuộc tính DataGridView.IsNewRow đảm bảo không tìm dòng mới trống cuối của DataGridView

Ví dụ: Hiển thị cột tenhang và mô tả, và che dấu các cột còn lại

```
foreach (DataGridViewColumn col in dataGridView1.Columns)
{
    col.Visible = false;
}
dataGridView1.Columns["Hoten"].Visible = true;
-
```

- DataGridViewButtonColumn, DataGridViewButtonCell hiển thị văn bản như một nút lệnh.
- DataGridViewLinkColumn, DataGridViewLinkCell hiển thị văn bản như một liên kết

- DataGridViewCheckBoxColumn, DataGridViewCheckBoxCell hiển thị một check box. Cột này tự động sử dụng dữ liệu kiểu bool
- Thay đổi tiêu đề cột:

```
dataGridView1.Columns["Hoten"].HeaderCell.Value = "Họ và  
Tên";
```

- DataGridViewComboBoxColumn, DataGridViewComboBoxCell hiển thị hộp danh sách
- DataGridViewImageColumn, DataGridViewImageCell hiển thị hình ảnh.
- DataGridViewTextBoxColumn, DataGridViewTextBoxCell hiển thị văn bản, cho phép chỉnh sửa
- Cập nhật dữ liệu trong DataGridView vào CSDL
- Có thể thêm, sửa, xóa trong DataGridView, sau đó để cập nhật dữ liệu từ DataGridView vào CSDL:

```
SqlCommandBuilder cb = new SqlCommandBuilder(sqlAdap);  
sqlAdap.Update(table);
```

Trong đó: table là DataTable ràng buộc dữ liệu vào DataGridView

5.2.10. Điều khiển tạo lập báo cáo sử dụng Crystal Report

5.2.10.1. Các bước tạo Crystal Reports

- Kích phải tên dự án trong cửa sổ Solution Explorer, chọn Add/ New Item, chọn Crystal Report, chọn Using the Report Wizard
- Ở mục Create New Connection, mở rộng mục OLE DB (ADO), chọn Microsoft OLE DB Provider for SQL Server, chọn Next, gõ tên SQL server, user ID, password, và tên Database, Next, Finish
- Mở rộng Tables, chọn tên bảng, kích nút >, Next, chọn các trường, kích nút >, Next
- Chọn trường nhóm Group by, kích nút >

- Chọn trường Sumarized Field, kích chọn mục Sum of..., có thể chọn các hàm Sum, Average, Maximum, Minimum, Count... từ hộp danh sách để tính toán
- Chọn dạng trình bày Report Style, Finish

5.2.10.2. Các bước tạo Crystal Reports

- Report Header: chứa tiêu đề đầu report, xuất hiện đầu của report
- Page Header: chứa tiêu đề đầu trang, xuất hiện đầu mỗi trang
- Details: hiển thị nội dung chi tiết của report
- Report Footer: chứa tiêu đề cuối report, xuất hiện cuối report
- Page Footer: chứa tiêu đề cuối trang, xuất hiện cuối mỗi trang

5.2.10.3. Tự thiết kế Report

- Chèn văn bản vào Report: Kích phải tại vùng Report, chọn Insert/ Text Object, gõ văn bản
- Kích phải trên Text Object, chọn Format / Object, để định dạng Text Object
- Cửa sổ Field Explorer: chứa các mục, có thể bổ sung vào cửa sổ thiết kế report
 - o Formula Fields: chèn trường tính toán vào report. Kích phải Formula Fields, chọn New, gõ tên trường, chọn Use Expert để xác định công thức tính toán
 - o Group Name Fields: sử dụng để nhóm và sắp xếp dữ liệu trong report. Kích phải Group Name Fields, chọn Insert Group
 - o Special Fields: sử dụng để chèn trường như số thứ tự bản ghi, số trang. Kéo lê trường cần chèn vào report
 - o Unbound Fields: chứa công thức tính toán
- Kéo lê Unbound Fields kiểu string vào report, kích phải, chọn edit formula, gõ công thức:
"Ngày " + ToText (Day(CurrentDate))
+ " Tháng " + ToText (Month(CurrentDate))

+ " Năm " + ToText(Year(CurrentDate))

5.2.10.4. Xem Crystal Report

- Bổ sung điều khiển CrystalReportViewer vào Form
- Ràng buộc điều khiển CrystalReportViewer đến Crystal Reports đã tạo bằng cách chọn thuộc tính của CrystalReportViewer là ReportSource, kích nút Browse để chọn Crystal Reports cần xem
- Để xem Crystal Reports, chạy ứng dụng

5.2.10.5. Điều khiển CrystalReportViewer

Các chức năng của điều khiển:

- Go to First Page: di chuyển về trang đầu tiên của Report
- Go to Previous Page: di chuyển về trang trước của Report
- Go to Next Page: di chuyển về trang tiếp của Report
- Go to Last Page: di chuyển về trang cuối cùng của Report
- Go to Page: di chuyển về trang chỉ rõ của Report
- Close Current View: đóng cửa sổ xem Report
- Print Report: in Report
- Refresh: xem lại Report
- Export Report: xuất Report ra định dạng khác, như Adobe Acrobat, Microsoft Excel, Microsoft Rich Text, HTML, và Microsoft Word

* Có thể tương tác giữa điều khiển CrystalReportViewer và các điều khiển khác bằng cách xử lý sự kiện. Một số sự kiện của điều khiển CrystalReportViewer:

Load: phát sinh khi điều khiển CrystalReportViewer được nạp vào bộ nhớ

ReportRefresh: phát sinh khi report được xem lại

Ví dụ muốn hiển thị Crystal Reports được chọn bởi người dùng vào lúc chạy chương trình, viết mã sử dụng lớp OpenFileDialog để hiển thị Crystal Reports. Sử dụng

thuộc tính ReportSource của điều khiển CrystalReportViewer để hiển thị Crystal Reports được chọn

5.2.10.6. Cải tiến Crystal reports

Ví dụ, phân tích tình hình bán mặt hàng bán trong mỗi tháng sử dụng đồ thị (Chart)

- Kích phải trên cửa sổ thiết kế Report, chọn Insert/ Chart
- Ở tab Type, chọn kiểu đồ thị Chart type
- Ở tab Data, Group, ở mục On change of, chọn trường nhóm dữ liệu, ở mục Show, chọn dữ liệu tổng hợp của mỗi nhóm
- Ở tab Text, gõ tiêu đề report ở mục Title, Subtitle

5.3. LINQ (Language Integrated Query)

5.3.1. Giới thiệu LINQ

Linq được viết tắt từ cụm từ Language Integrated Query, hay ta có thể tạm dịch là ngôn ngữ truy vấn được tích hợp vào ngôn ngữ lập trình. Linq là một thành phần trong gói .Net của hãng Microsoft. Nó có thể thêm câu hỏi truy vấn dữ liệu vào ngôn ngữ .net với cấu trúc gần giống với cấu trúc SQL. Nhiều những khái niệm mà LINQ được giới thiệu trước đây đã được kiểm tra ở Dự án nghiên cứu CW của hãng Microsoft. LINQ được phát triển tự do như một phần của gói .NET 3.5.

LINQ định nghĩa là một tập hợp những toán tử truy vấn có thể được sử dụng để truy vấn hoặc lọc dữ liệu từ những lớp dữ liệu chuẩn như XML, CSDL quan hệ, những nguồn dữ liệu của hãng thứ ba. Nó cũng cho phép truy vấn đến bất kỳ nguồn dữ liệu nào nhưng với yêu cầu dữ liệu đó phải được đóng gói như những đối tượng. Vì thế, nếu nguồn dữ liệu không lưu trữ dữ liệu như những đối tượng, thì dữ liệu phải là ma trận có miền giá trị của đối tượng. Những câu hỏi đã viết sử dụng những toán tử truy vấn được thực hiện theo cách khác bởi bộ xử lý câu hỏi LINQ hoặc qua một cơ chế mở rộng, được thoả thuận với những nhà cung cấp LINQ. Những kết quả của câu hỏi được trả về như một tập hợp bộ nhớ trong của những đối tượng.

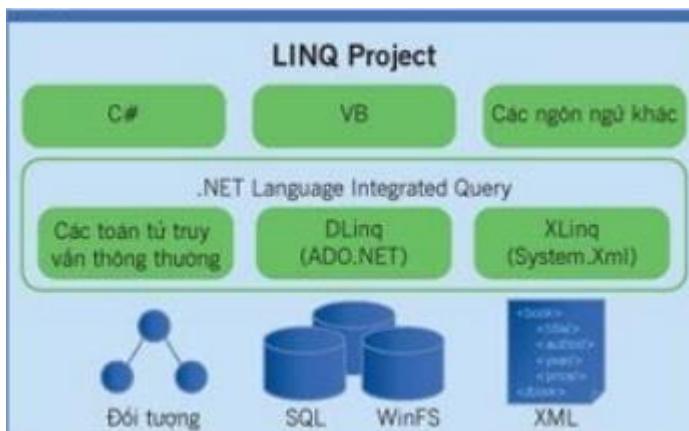
Giải pháp lập trình hợp nhất, đem đến khả năng truy vấn dữ liệu theo cú pháp SQL trực tiếp trong C# hay VB.NET, áp dụng cho tất cả các dạng dữ liệu từ đối tượng đến CSDL quan hệ và XML.

Xử lý thông tin hay dữ liệu là nhiệm vụ quan trọng nhất của bất kỳ phần mềm nào và một trong những trở ngại chính mà các nhà phát triển hiện nay phải đổi mới là khác biệt giữa ngôn ngữ lập trình hướng đối tượng và ngôn ngữ truy vấn dữ liệu, vấn đề càng phức tạp hơn với sự xuất hiện của XML (eXtensible Markup Language - ngôn ngữ đánh dấu mở rộng).

Hiện tại, cách phổ biến nhất để ứng dụng lấy dữ liệu từ các hệ cơ sở dữ liệu (CSDL) là sử dụng SQL (Structure Query Language - ngôn ngữ truy vấn cấu trúc). SQL có cú pháp rất khác với những ngôn ngữ lập trình phổ dụng như C# và VB.NET, do vậy lập trình viên phải học công "hàn gắn" hai thực thể khác biệt này với nhau trong mỗi dự án phần mềm.

Một vấn đề khác với SQL là nó chỉ dùng để truy vấn dữ liệu trong các CSDL dạng quan hệ. Nếu muốn truy cập dữ liệu XML hay dạng khác (như trang HTML, email...), nhà phát triển lại phải sử dụng cú pháp truy vấn khác (XPath/XQuery).

Để giảm gánh nặng thao tác trên nhiều ngôn ngữ khác nhau và cải thiện năng suất lập trình, Microsoft đã phát triển giải pháp tích hợp dữ liệu cho .NET Framework có tên gọi là LINQ (Language Integrated Query), đây là thư viện mở rộng cho các ngôn ngữ lập trình C# và Visual Basic.NET (có thể mở rộng cho các ngôn ngữ khác) cung cấp khả năng truy vấn trực tiếp dữ liệu đối tượng, CSDL và XML.



Hình 5.12. LINQ Project

5.3.2. Truy vấn dữ liệu LINQ

5.3.2.1. Truy vấn dữ liệu đối tượng trong bộ nhớ

Dữ liệu cần phải đổ vào bộ nhớ để xử lý, nhưng một khi tách khỏi nơi gốc của nó thì khả năng truy vấn rất kém. Bạn có thể dễ dàng truy vấn thông tin khách hàng mọc nối với thông tin đơn hàng của họ từ CSDL SQL Server nhưng không dễ gì thực hiện tương tự với thông tin trong bộ nhớ. Trong môi trường .NET, thông tin (trong bộ nhớ) thường được thể hiện ở dạng các đối tượng và trước LINQ, không có cách nào để mọc nối các đối tượng hay thực hiện bất kỳ thao tác truy vấn nào. LINQ chính là giải pháp cho vấn đề này.

Ví dụ, trong SQL Server, chúng ta có thể truy vấn tất cả record (mẫu tin hay hàng) từ bảng (table) Customer theo cách sau:

```
SELECT * FROM Sinhvien
```

Giá trị trả về là tập kết quả ("result set") tương tự như bảng dữ liệu, chứa tất cả các trường (field) của bảng Sinhvien.

Sử dụng LINQ, chúng ta có thể thực hiện truy vấn tương tự bằng chính lệnh C# hay VB.NET, chỉ khác là truy vấn danh sách đối tượng trong bộ nhớ thay vì bảng trong CSDL. Ví dụ đơn giản dưới đây sử dụng "nguồn dữ liệu" là một mảng chuỗi, trong VB.NET:

```
String names = {"Long", "Lân", "Qui", "Phụng"}
```

Các đối tượng trong mảng names có tên là name.

```
Foreach (String name in names) { }
```

Dùng cú pháp LINQ, chúng ta có thể truy vấn "nguồn dữ liệu" này tương tự như truy vấn bảng bằng SQL.

Select name From name in names

Danh sách đối tượng (mảng) names ở đây tương đương với bảng Customer trong câu lệnh SQL ở trên.

Vì .NET là môi trường đối tượng, mọi thứ đều dựa trên đối tượng, thuộc tính và phương thức. Vì vậy cả nguồn dữ liệu mà chúng ta truy vấn cũng như tập kết quả trả về cũng đều là đối tượng. Do vậy chúng ta cần khai báo biến cho phát biểu Select (hay kết quả của phát biểu Select), ví dụ:

```
IEnumerable<String> result = Select name From name in names
```

Tương tự, trong C#:

```
IEnumerable<String> result = from name in names select name;
```

LINQ có đủ các toán tử truy vấn trên dữ liệu đối tượng tương tự như SQL trên CSDL, chẳng hạn như xếp thứ tự (order), điều kiện (where) hay móc nối (join)...

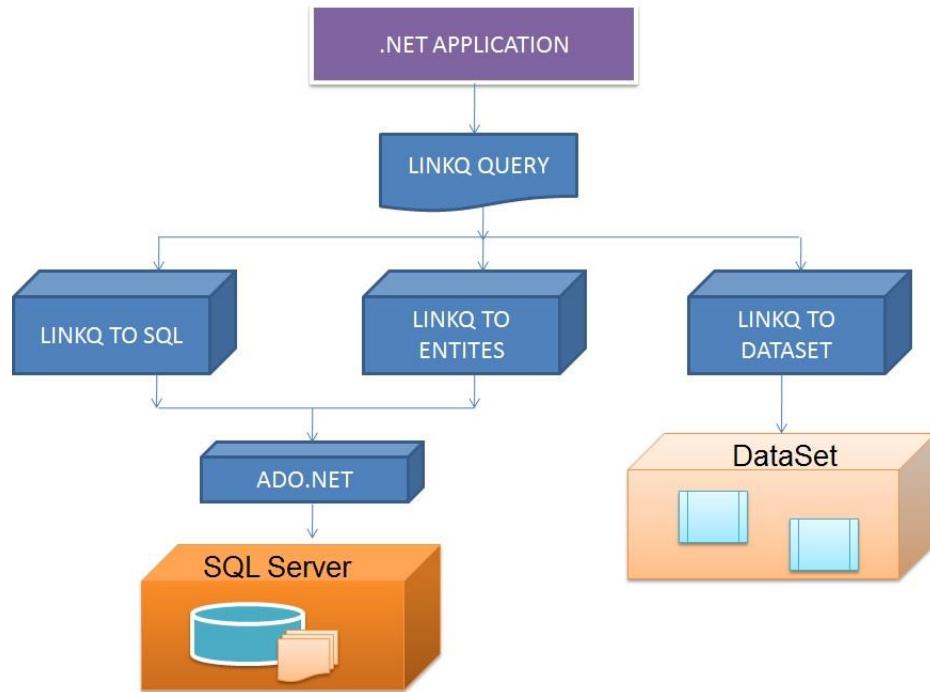
Tính năng truy vấn các đối tượng trong bộ nhớ mở ra nhiều khả năng thú vị. Ví dụ, bạn có thể truy vấn tất cả các textbox trong một form có giá trị nhất định, và móc nối chúng với các đối tượng của một tập hợp được "hợp" (union) với tập kết quả truy vấn từ CSDL hay tài liệu XML.

Truy vấn CSDL "thực"

Tất nhiên, dữ liệu không chỉ nằm trong bộ nhớ. Có 2 nơi quan trọng khác thường chứa dữ liệu là hệ CSDL (SQL Server) và tài liệu XML (các dữ liệu "thực" này được lưu trữ vật lý, có thời gian "sống" lâu hơn dữ liệu "ảo" trong bộ nhớ). LINQ có 2 bộ hàm API dùng để truy vấn các nguồn dữ liệu này: DLINQ dùng truy vấn CSDL quan hệ (SQL) và XLINQ dùng truy vấn dữ liệu phân cấp (XML).

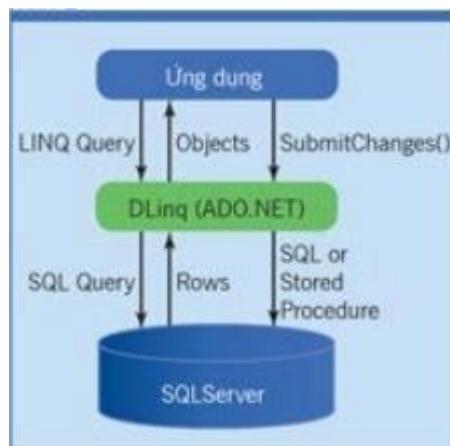
5.3.2.2. DLINQ

LinkQ to ADO.NET



Hình 5.13. Kiến trúc LINQ

DLINQ là tập các lớp đặc biệt cho phép thể hiện các bảng và hàng dữ liệu theo dạng đối tượng, nhờ vậy có thể sử dụng LINQ để truy vấn trực tiếp CSDL



Hình 5.14. Mô hình hoạt động DLINQ

DLINQ dùng đối tượng `DataContext` để mở kết nối đến CSDL. Sau đó dùng lớp `Table<>` để thể hiện bảng dữ liệu, và với đối tượng này, chúng ta có thể sử dụng cú pháp LINQ để truy vấn.

Ví dụ sau đây truy vấn tất cả khách hàng (customer) có tên công ty bắt đầu bằng chữ "T" từ CSDL Northwind của SQL Server, dùng cú pháp lệnh của C#:

```
DataContext context = new DataContext("Initial  
Catalog=Northwind;" + "Integrated Security=sspi");  
  
Table<CustomerTable> customers =  
context.GetTable<CustomerTable>();  
  
var result = from c in customers where c.CompanyName.StartsWith("T")  
select c;
```

5.3.2.3. XLINQ

Những gì mà DLINQ thực hiện với CSDL thì XLINQ thực hiện với XML. Xét chuỗi XML sau:

```
<customers>  
  
<customer>  
  
<companyName>PC World Vietnam</companyName>
```

```
<contactName>The Gioi Vi Tinh</contactName>

</customer>

<customer>

...

</customer>

</customers>
```

XLINQ cho phép đưa chuỗi XML này vào đối tượng XElement để truy vấn với cú pháp LINQ.

```
XElement names = XElement.Parse(xmlString);

var result = from n in names.Descendants("customer") where
n.Descendants("companyName")

.Value.StartsWith("T")

select n.Descendants("contactName").Value;
```

Truy vấn này trả về danh sách chuỗi chứa tên người liên hệ của tất cả khách hàng (customer) có tên công ty bắt đầu bằng chữ "T".

XLINQ còn có tính năng hấp dẫn khác: tạo XML. Việc này cũng thực hiện với XElement và các đối tượng XLINQ khác.

Ví dụ, xét dữ liệu XML sau:

```
<root>

<sub> Test </sub>

</root>
```

Chúng ta có thể tạo dữ liệu XML trên với các đối tượng XLINQ.

```
XElement xml = new XElement("root", new
XElement("sub", "Test");
```

```
Console.WriteLine(xml.ToString());
```

C. HÌNH THÚC VÀ PHƯƠNG PHÁP GIẢNG DẠY

- Trình chiếu powerpoint
- Đặt vấn đề, trao đổi
- Thực nghiệm kết hợp với máy tính

D. TÀI LIỆU THAM KHẢO

Tiếng Việt

- [1] Nguyễn Ngọc Bình Phương, Thái Thanh Phong, Các giải pháp lập trình C#, Nhà sách Đất Việt

Tiếng Anh

- [2] Erik brown, Windows Forms Programming with C#, Manning Publications Co.
[3] James W. Cooper, Introduction to Design Patterns in C#
[4] Matthew MacDonald, Pro .NET 2.0 Windows Forms and Custom Controls in C#

Website

- [5] <http://csharpcomputing.com/Tutorials/Lesson9.htm>
[6] <http://www.csharp-station.com/Tutorials/Lesson02.aspx>
[7] <http://msdn.microsoft.com/en-us/library/>

XÉT DUYỆT CỦA TRƯỞNG BỘ MÔN

.....
.....
.....

ĐÀ NẴNG, NGÀY THÁNG NĂM

KẾT QUẢ KIỂM TRA TẬP BÀI GIẢNG

.....
.....
.....

ĐÀ NẴNG, NGÀY THÁNG NĂM

PHÒNG THANH TRA

.....
.....
.....

ĐÀ NẴNG, NGÀY THÁNG NĂM