

MỤC LỤC

MỤC LỤC	I
HƯỚNG DẪN.....	IV
BÀI 1: TỔNG QUAN VỀ .NET	1
 1.1 GIỚI THIỆU .NET FRAMEWORK.....	1
1.1.1 Tổng quan.....	1
1.1.2 Lịch sử hình thành.....	2
1.1.3 Quá trình phát triển.....	3
 1.2 ĐẶC ĐIỂM .NET.....	4
1.2.1 Kiến trúc .Net Framework.....	5
1.2.2 Common Language Runtime (CLR)	7
1.2.3 Thư viện lớp .NET Framework	8
1.2.4 Biên dịch và MS Intermediate Language.....	9
1.2.5 Garbage collection - bộ thu dọn rác	10
1.2.6 Namespace.....	11
 TÓM TẮT	13
 CÂU HỎI ÔN TẬP	13
BÀI 2: NGÔN NGỮ C#	15
 2.1 CÁC ĐẶC ĐIỂM CỦA NGÔN NGỮ C#.....	15
2.1.1 Giới thiệu	15
2.1.2 Đặc điểm của ngôn ngữ C#	16
2.1.3 Các loại ứng dụng C#	17
2.1.4 Ứng dụng C#	18
2.1.5 Cấu trúc chương trình C#	19
 2.2 NGÔN NGỮ C#	21
2.2.1 Kiểu dữ liệu – Data type.....	21
2.2.2 Từ khóa – Keyword	22
2.2.3 Quy tắc đặt tên, biến, hằng	23
2.2.4 Biến readonly	25
2.2.5 Chuyển đổi các kiểu dữ liệu trong C#.....	25
 TÓM TẮT	36
 CÂU HỎI ÔN TẬP	36
BÀI 3: LỚP VÀ GIAO DIỆN.....	38
 3.1 LỚP - CLASS	38
3.1.1 Giới thiệu	38
3.1.2 Khai báo	39
3.1.3 Constructor	41
3.1.4 Destructor.....	42
3.1.5 Method	42

3.1.6 Nạp chồng (method – overload)	43
3.1.7 Thuộc tính (Property)	44
3.2 KẾ THỪA VÀ ĐA HÌNH	45
3.2.1 Thừa kế	45
3.2.2 Up-cast và down-cast	46
3.2.3 Đa hình (Polymorphism)	48
TÓM TẮT	53
CÂU HỎI ÔN TẬP	54
BÀI 4: DELEGATE & EVENT	55
4.1 ỦY THÁC - DELEGATE	55
4.1.1 Khai báo	57
4.1.2 Instance delegate	57
4.1.3 Multicast delegate	58
4.1.4 Call Delegate	59
4.1.5 Demo – Buble sort	60
4.2 SỰ KIỆN - EVENT	62
4.2.1 Lập trình sự kiện	62
4.2.2 Event	63
TÓM TẮT	68
CÂU HỎI ÔN TẬP	69
BÀI 5: LINQ	70
5.1 LANGUAGE INTEGRATED QUERY (LINQ)	70
5.1.1 Nguồn dữ liệu (data source)	71
5.1.2 Truy vấn LINQ	72
5.1.3 Thực thi truy vấn	72
5.2 SỬ DỤNG TRUY VẤN LINQ	73
5.2.1 Cú pháp truy vấn và cú pháp phương thức	73
5.2.2 Danh sách truy vấn LINQ	74
TÓM TẮT	81
CÂU HỎI ÔN TẬP	82
BÀI 6: WINDOWS FORM	83
6.1 CÁC KHÁI NIỆM	83
6.1.1 Giao diện đồ họa – Graphical User Interface	83
6.1.2 Cơ chế lập trình sự kiện – Event-Driven Programming	85
6.2 ỨNG DỤNG WINDOWS FORM	86
6.2.1 GUI Components/Controls	87
TÓM TẮT	90
CÂU HỎI ÔN TẬP	90
BÀI 7: WINDOWS CONTROLS	91
7.1 TỔNG QUAN VỀ CONTROLS	91
7.2 CÁC CONTROLS CƠ BẢN	94

7.2.1 Label, TextBox, Button.....	94
7.2.2 GroupBox, Panel & TabControl	95
7.2.3 CheckBox, RadioButton.....	98
7.2.4 ListBox, ComboBox.....	100
7.2.5 ListView	103
TÓM TẮT	110
BÀI TẬP	111
BÀI 8: ADO.NET.....	120
8.1 TỔNG QUAN.....	120
8.1.1 Giới thiệu	120
8.1.2 Các đặc điểm của ADO.NET	122
8.2 CÁC ĐỐI TƯỢNG ADO.NET	124
8.2.1 Kiến trúc ADO.NET	124
8.2.2 Các đối tượng ADO.NET.....	126
TÓM TẮT	135
BÀI 9: DATA BINDING.....	136
9.1 GIỚI THIỆU.....	136
9.2 CÁC KỸ THUẬT.....	136
9.2.1 Simple Data Binding	137
9.2.2 Complex Data Binding.....	138
9.2.3 Data Binding dữ liệu cho Windows Controls	140
TÓM TẮT	143
BÀI 10: ENTITY FRAMEWORK (EF).....	144
10.1 ENTITY FRAMEWORK (EF)	144
10.1.1 Kiến trúc trong Entity Framework.....	144
10.1.2 Các thành phần trong Entity Framework	146
10.2 SỬ DỤNG ENTITY FRAMEWORK MÔ HÌNH CODE FIRST	147
10.2.1 Cài đặt EF.....	148
10.2.2 Tạo Entity Data Model	148
10.2.3 DbContext	150
10.2.4 Entity.....	151
10.2.5 Truy vấn với EDM	153
TÓM TẮT	155
CÂU HỎI ÔN TẬP	155
TÀI LIỆU THAM KHẢO.....	157

HƯỚNG DẪN

MÔ TẢ MÔN HỌC

Lập trình C# trên Windows là một trong những môn học nhằm cung cấp kiến thức cơ bản cho những ai muốn xây dựng được ứng dụng trên môi trường trên hệ điều hành Windows. Môn học này cung cấp những kiến thức cơ bản, tổng quan về công nghệ mới của Microsoft: .Net Framework, lập trình hướng đối tượng trên .Net, cũng như giới thiệu bộ thư viện .Net mà cung cấp sẵn (LINQ, Entity framework...), sinh viên có thể tạo giao diện ứng dụng, lồng kết nối các hệ cơ sở dữ liệu. Từ đó sinh viên có thể bước đầu có thể xây dựng những ứng dụng tin học hóa, hỗ trợ các doanh nghiệp trong việc quản lý hàng hóa, nhân sự, mua bán, v.v.

NỘI DUNG MÔN HỌC

- Bài 1. Tổng quan .NET và C#: Bài này cung cấp cho học viên khái niệm về chương trình nền tảng cho công nghệ .NET, CLR, CTS, CLS, MSIL, tập hợp class library thường dùng, quản lý sự thực thi của các chương trình .NET
- Bài 2. Ngôn ngữ C#: Bài này giới thiệu tổng quan về ngôn ngữ C#, đặc điểm ngôn ngữ, quá trình biên dịch, cấu trúc của một chương trình C#, các kiểu dữ liệu, lập trình với vòng lặp, phân nhánh, mảng dữ liệu, các loại tham số: ref, out, param. Giúp học viên bước đầu làm quen với kiến thức cơ bản của ngôn ngữ C#, xây dựng chương trình C# đơn giản đầu tiên.
- Bài 3. Lớp và giao diện: Bài này tập trung cung cấp kiến thức về lập trình hướng đối tượng, tìm hiểu tính đa hình trên ngôn ngữ C#. Giúp học viên mô tả được các đối tượng trong thế giới thực, xây dựng các lớp đối tượng với các đặc tính dữ liệu và hành vi.
- Bài 4. Cơ chế delegate & event: Giới thiệu event-handling model của C#. Giới thiệu các đặc điểm, và cài đặt của ủy thác (delegate) và sự kiện (event). Giúp học viên nắm được cơ chế lập trình sự kiện trên Windows, và các đối tượng lắng nghe sự kiện sẽ thực hiện ủy thác (delegate) như thế nào khi publisher dispatch sự kiện.
- Bài 5. LINQ: Giới thiệu truy vấn LINQ trong C#. Giúp học viên nắm được cách sử dụng danh sách truy vấn LINQ, Việc hiểu rõ Linq cũng giúp nắm bắt được những

khái niệm về lập trình hàm (Functional programming), nhằm tận dụng được hết sức mạnh mà C# và .NET mang lại.

- Bài 6: Windows Form. Giới thiệu lập trình ứng dụng với giao diện đồ họa (GUI) trên Windows. Giúp học viên hiểu rõ cấu trúc, thành phần của một ứng dụng GUI, làm quen với các lớp thư viện trong System.Windows.Form, làm quen với giao diện kéo thả, xây dựng ứng dụng desktop đầu tiên.
- Bài 7: Windows Controls: Tổng quan về Windows controls: Properties, Methods and Event của control. Giúp học viên nắm rõ các thuộc tính, hành vi, và sự kiện của windows control, từ đó xây dựng ứng dụng giao diện người dùng.
- Bài 8: ADO.Net: Lập trình cơ sở dữ liệu trên .Net.
- Bài 9: Data Binding. Giúp học viên kết nối dữ liệu với các đối tượng có giao diện như DataGridView, ComboBox,... Từ đó xây dựng nên ứng dụng hoàn chỉnh.
- Bài 10: Entity FrameWork (EF). Giúp học viên lập trình cơ sở dữ liệu bằng EF. Đặc trưng là mô hình code first với hướng tiếp cận có sẵn cơ sở dữ liệu.

KIẾN THỨC TIỀN ĐỀ

Môn học Lập trình C# trên Windows đòi hỏi sinh viên có nền tảng về Lập trình C, và lập trình hướng đối tượng.

YÊU CẦU MÔN HỌC

Người học phải dự học đầy đủ các buổi lên lớp và làm bài tập đầy đủ ở nhà.

CÁCH TIẾP NHẬN NỘI DUNG MÔN HỌC

Để học tốt môn này, người học cần ôn tập các bài đã học, trả lời các câu hỏi và làm đầy đủ bài tập; đọc trước bài mới và tìm thêm các thông tin liên quan đến bài học.

Đối với mỗi bài học, người học đọc trước mục tiêu và tóm tắt bài học, sau đó đọc nội dung bài học. Kết thúc mỗi ý của bài học, người đọc trả lời câu hỏi ôn tập và kết thúc toàn bộ bài học, người đọc làm các bài tập.

PHƯƠNG PHÁP ĐÁNH GIÁ MÔN HỌC

Môn học được đánh giá gồm:

- **Điểm thực hành (30%):** Hình thức thi thực hành, phù hợp với quy chế đào tạo và tình hình thực tế tại nơi tổ chức học tập.
- **Điểm quá trình (20%):** Hình thức và cách đánh giá do giảng viên dạy lý thuyết quyết định được phê duyệt của bộ môn.
- **Điểm thi cuối kỳ (50%):** Hình thức làm đồ án môn học và chấm thi vẫn đáp dựa trên nội dung đồ án môn học kết hợp với lý thuyết trong các bài học. Danh sách đồ án do giảng viên quyết định được bộ môn kiểm duyệt và cung cấp vào đầu khóa học.

BÀI 1: TỔNG QUAN VỀ .NET

Sau khi học xong bài này, học viên có thể:

- *Hiểu rõ nền tảng .NET Framework, CLR, CTS, CLS, MSI, Garbage collection, Namespace*
- *Kiến trúc .Net Framework*
- *Quá trình biên dịch CT C#*
- *Namespace.*

1.1 GIỚI THIỆU .NET FRAMEWORK

1.1.1 Tổng quan

Microsoft .NET gồm 2 phần chính: Framework và Integrated Development Environment (IDE).

- Framework cung cấp những gì cần thiết và căn bản, chữ Framework có nghĩa là khung hay khung cảnh trong đó ta dùng những hạ tầng cơ sở, theo một qui ước nhất định để công việc được trôi chảy.
- IDE thì cung cấp một môi trường giúp chúng ta triển khai dễ dàng, và nhanh chóng các ứng dụng dựa trên nền tảng .NET. Nếu không có IDE chúng ta cũng có thể dùng một trình soạn thảo ví như Notepad hay bất cứ trình soạn thảo văn bản nào và sử dụng command line để biên dịch và thực thi, tuy nhiên việc này mất nhiều thời gian. Tốt nhất là chúng ta dùng IDE phát triển các ứng dụng, và cũng là cách dễ sử dụng nhất.
- Thành phần Framework là quan trọng nhất .NET là cốt lõi và tinh hoa của môi trường, còn IDE chỉ là công cụ để phát triển dựa trên nền tảng đó thôi. Trong .NET toàn bộ các ngôn ngữ C#, Visual C++ hay Visual Basic.NET đều dùng cùng một IDE.

Tóm lại Microsoft .NET là nền tảng cho việc xây dựng và thực thi các ứng dụng phân tán thế hệ kế tiếp. Bao gồm các ứng dụng từ client đến server và các dịch vụ khác. Một số tính năng của Microsoft .NET cho phép những nhà phát triển sử dụng như sau:

- Một mô hình lập trình cho phép nhà phát triển xây dựng các ứng dụng dịch vụ web và ứng dụng client với XML (Extensible Markup Language - Ngôn ngữ đánh dấu mở rộng), một chuẩn trao đổi dữ liệu trên internet.
- Cung cấp các server phục vụ bao gồm: Windows 2000, SQL Server, và BizTalk Server, tất cả điều tích hợp, hoạt động, và quản lý các dịch vụ XML Web và các ứng dụng.
- Các phần mềm client như Windows XP và Windows CE giúp người phát triển phân phối sâu và thuyết phục người dùng kinh nghiệm thông qua các dòng thiết bị.

Nhiều công cụ hỗ trợ như Visual Studio .NET, để phát triển các dịch vụ Web XML, ứng dụng trên nền Windows hay nền web một cách dễ dàng và hiệu quả.

1.1.2 Lịch sử hình thành

Đầu năm 1998, sau khi hoàn tất phiên bản Version 4 của Internet Information Server (IIS), các đội ngũ lập trình ở Microsoft nhận thấy họ còn rất nhiều sáng kiến để kiện toàn IIS. Họ bắt đầu xây dựng một kiến trúc mới trên nền tảng ý tưởng đó và đặt tên là Next Generation Windows Services (NGWS).

Sau khi Visual Basic được trình làng vào cuối 1998, dự án kế tiếp mang tên Visual Studio 7 được xác nhập vào NGWS. Đội ngũ COM+/MTS góp vào một bộ runtime chung cho tất cả ngôn ngữ lập trình chung trong Visual Studio, và tham vọng của họ cung cấp cho các ngôn ngữ lập trình của các công ty khác dùng chung luôn. Công việc này được xúc tiến một cách hoàn toàn bí mật mãi cho đến hội nghị Professional Developers' Conference ở Orlando vào tháng 7/2000. Đến tháng 11/2000 thì Microsoft đã phát hành bản Beta 1 của .NET gồm 3 đĩa CD. Tính đến lúc này thì Microsoft đã làm việc với .NET gần 3 năm rồi, do đó bản Beta 1 này tương đối vững chắc.

.NET mang dáng dấp của những sáng kiến đã được áp dụng trước đây như p-code trong UCSD Pascal cho đến Java Virtual Machine. Có điều là Microsoft góp nhặt những sáng kiến của người khác, kết hợp với sáng kiến của chính mình để làm nên một sản

phẩm hoàn chỉnh từ bên trong lẫn bên ngoài. Hiện tại Microsoft đã công bố các phiên bản release của .NET.

1.1.3 Quá trình phát triển

Ngày 12/2/2002 đánh dấu bước quan trọng đầu tiên trong “cuộc đời” của .NET Framework, khi phiên bản 1.0 cùng với Visual Studio.NET 2002 được chính thức ra mắt. Chính .NET Framework 1.0 là điểm nhấn đáng chú ý nhất và làm cho Visual Studio.NET 2002 khác biệt hẳn với Visual Studio 6.0 đã phát hành năm 1998. Lần đầu tiên, Microsoft giới thiệu về “lập trình hợp nhất”, với việc lấy .NET Framework làm nền tảng.

Phiên bản 1.1 – phát hành năm 2003: một năm sau ngày .NET Framework 1.0 ra đời, ngày 24/4/2003, Microsoft đã có ngay bản cập nhật 1.1 ra mắt cùng với Visual Studio.NET 2003. .NET Framework 1.1 cũng mở ra một “truyền thống” là kể từ đây, các HĐH Windows đều được cài đặt sẵn phiên bản .NET Framework mới nhất. Windows Server 2003 tiên phong với phiên bản 1.1, sau đó là Windows Vista với .NET 3.0, và gần đây nhất là Windows 7/Server 2008 với .NET 3.5 SP1.

Phiên bản 2.0 phát hành năm 2005: Microsoft mất đến hơn 2 năm để phát triển .NET Framework 2.0 và Visual Studio 2005, và thời gian bỏ ra là thật sự đáng giá. Tháng 11/2005, hai sản phẩm này ra mắt với hàng loạt tính năng mới, trong đó đáng kể nhất là việc hỗ trợ hoàn toàn cho tính toán 64-bit, .NET Micro Framework, bổ sung và nâng cấp nhiều control của ASP.NET và đặc biệt là hỗ trợ Generics. .NET 2.0 hoàn toàn khác biệt so với các phiên bản trước.

Phiên bản 3.0: nếu như 3 phiên bản trước đó, .NET Framework đều gắn liền với một phiên bản Visual Studio nào đó, thì .NET Framework 3.0 đã “phá” truyền thống này khi ra mắt cùng với hệ điều hành Windows Vista vào cuối năm 2006. Ba “điểm nhấn” trong lần nâng cấp này là thành phần được kỳ vọng thay thế Winform – Windows Presentation Foundation – WPF, Windows Communication Foundation – WCF, Windows Workflow Foundation – WF, và Windows Card Space.

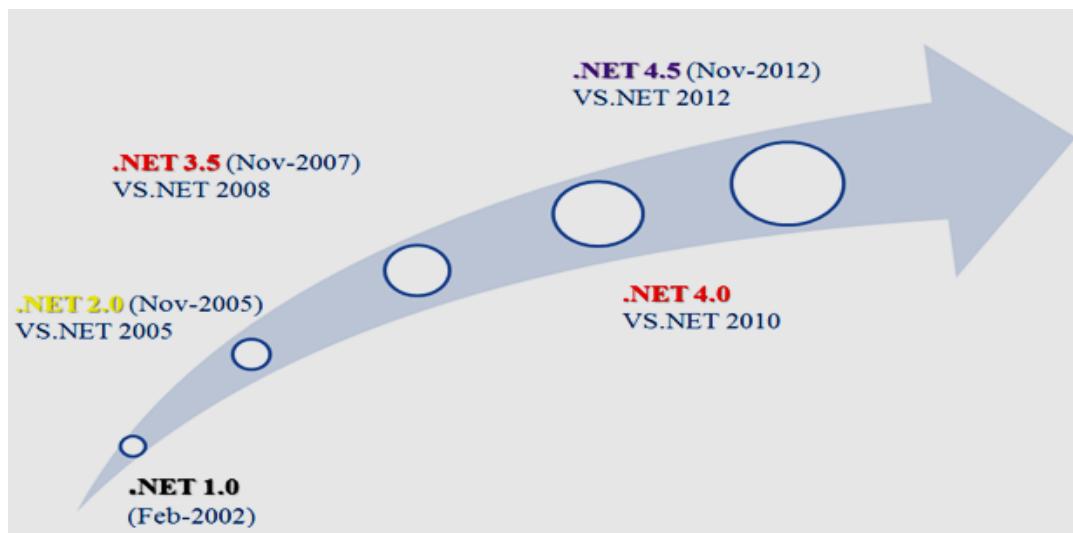
- Phiên bản 3.5 phát hành năm 2008, người dùng phải đợi đến tháng 11 năm 2007 mới được sử dụng một phiên bản Visual Studio hỗ trợ đầy đủ và toàn diện cho

.NET 3.0, và hơn thế nữa. Vâng, chúng ta đang nói đến VS 2008 và .NET Framework 3.5. Cũng như phiên bản 3.0, .NET 3.5, là một mở rộng trên nền .NET 2.0.

- LINQ [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] là phần nổi bật và đáng chú ý nhất trong .NET 3.5.

Phiên bản 4.0 – phát hành năm 2010: với mục tiêu nghiên cứu cách thức vận hành của nền kinh tế nói chung và hành vi của từng chủ thể trong nền kinh tế nói riêng, hoạt động kinh tế của xã hội có thể xem xét dưới nhiều góc độ khác nhau. Căn cứ vào đối tượng nghiên cứu, kinh tế học được phân thành hai nhánh: kinh tế học vi mô và kinh tế học vĩ mô.

Phiên bản 4.5: phát hành vào tháng 11/2012 cùng với VS 2012



Hình 1.1: Quá trình phát triển của .NET Framework

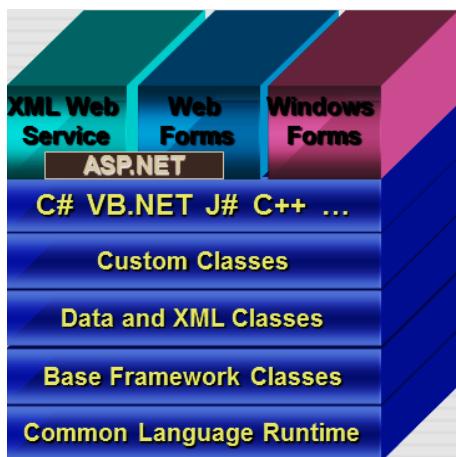
1.2 ĐẶC ĐIỂM .NET

.Net là một nền tảng hỗ trợ đa ngôn ngữ, chạy trên nền (.NET framework) với những đặc điểm nổi bật như sau:

- Mã nguồn được biên dịch qua MSIL
- MSIL được thông dịch qua mã máy lúc thực thi nhờ vào CLR

- Độc lập nền tảng (Về lý thuyết có thể chạy trên mọi nền!)
- Hỗ trợ thực thi chéo ngôn ngữ
- Cài đặt gói .NET Framework redistribute (dotnetfx.exe) để chạy ứng dụng .NET trên máy client.

1.2.1 Kiến trúc .Net Framework



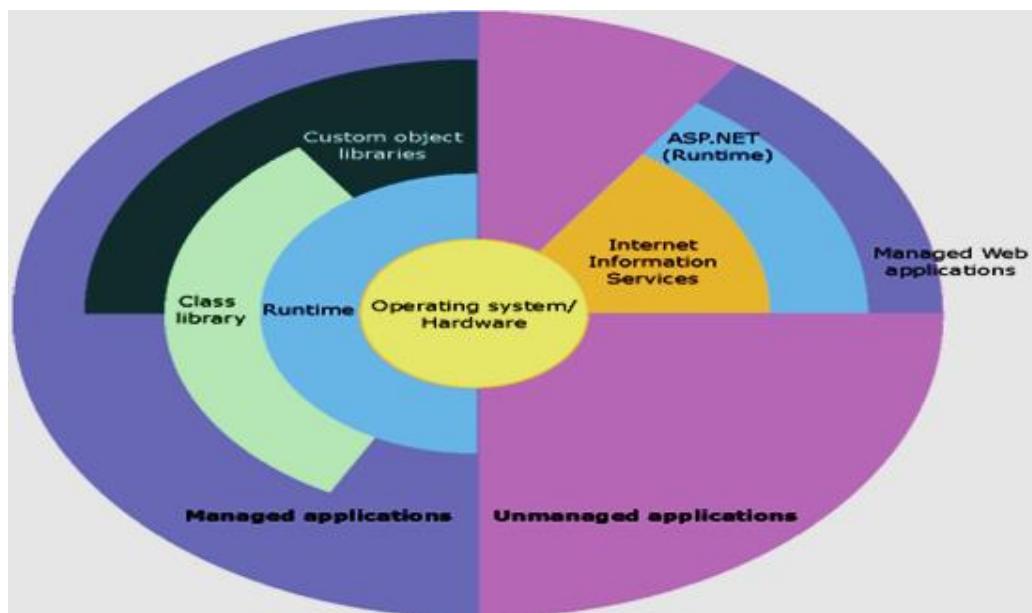
- Chương trình nền tảng cho công nghệ .NET
- Cung cấp tập hợp class library thường dùng
- Quản lý sự thực thi của các chương trình .NET
- Theo quan điểm của người lập trình, .NET có thể hiểu như môi trường thực thi mới và thư viện lớp cơ sở cải tiến.
- Môi trường thực thi là: Common Language Runtime – CLR. Vai trò chính CLR: locate, load, manage .NET types

Hình 1.2 Kiến trúc .Net Framework

.NET Framework là một platform 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:

- Để cung cấp một môi trường lập trình hướng đối tượng vững chắc, trong đó mã nguồn đối tượng được lưu trữ và thực thi một cách cục bộ. Thực thi cục bộ nhưng được phân tán trên Internet, hoặc thực thi từ xa.
- Để cung cấp một môi trường thực thi mã nguồn mà tối thiểu được việc đóng gói phần mềm và sự tranh chấp về phiên bản.
- Để cung cấp một môi trường thực thi mã nguồn mà đảm bảo việc thực thi an toàn mã nguồn, bao gồm cả việc mã nguồn được tạo bởi hàng thứ ba hay bất cứ hàng nào mà tuân thủ theo kiến trúc .NET.
- Để cung cấp một môi trường thực thi mã nguồn mà loại bỏ được những lỗi thực hiện các script hay môi trường thông dịch.

- Để làm cho những người phát triển có kinh nghiệm vững chắc có thể nắm vững nhiều kiểu ứng dụng khác nhau. Như là từ những ứng dụng trên nền Windows đến những ứng dụng dựa trên web.
- Để xây dựng tất cả các thông tin dựa trên tiêu chuẩn công nghiệp để đảm bảo rằng mã nguồn trên .NET có thể tích hợp với bất cứ mã nguồn khác. .NET Framework có hai thành phần chính: Common Language Runtime (CLR) và thư viện lớp .NET Framework. CLR là nền tảng của .NET Framework. Chúng ta có thể hiểu runtime như là một agent quản lý mã nguồn khi nó được thực thi, 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).
- Thư viện lớp, một thành phần chính khác của .NET Framework là một tập hợp hướng đối tượng của các kiểu dữ liệu được dùng lại, nó cho phép chúng ta có thể phát triển những ứng dụng từ những ứng dụng truyền thống command-line hay những ứng dụng có giao diện đồ họa (GUI) đến những ứng dụng mới nhất được cung cấp bởi ASP.NET, như là Web Form và dịch vụ XML Web.



Hình 1.3: Các thành phần của .Net Framework

1.2.2 Common Language Runtime (CLR)

Như đã đề cập thì CLR thực hiện quản lý bộ nhớ, quản lý thực thi tiểu trình, thực thi mã nguồn, xác nhận mã nguồn an toàn, biên dịch và các dịch vụ hệ thống khác. Những đặc tính trên là nền tảng cơ bản cho những mã nguồn được quản lý chạy trên CLR.

Do chú trọng đến bảo mật, những thành phần được quản lý được cấp những mức độ quyền hạn khác nhau, phụ thuộc vào nhiều yếu tố nguyên thủy của chúng như: liên quan đến Internet, hệ thống mạng trong nhà máy, hay một máy tính cục bộ. Điều này có nghĩa rằng, một thành phần được quản lý có thể có hay không có quyền thực hiện một thao tác truy cập tập tin, thao tác truy cập registry, hay các chức năng nhạy cảm khác.

CLR thúc đẩy việc mã nguồn thực hiện việc truy cập được bảo mật. Ví dụ, người sử dụng giới hạn rằng việc thực thi nhúng vào trong một trang web có thể chạy được hoạt hình trên màn hình hay hát một bản nhạc, nhưng không thể truy cập được dữ liệu riêng tư, tập tin hệ thống, hay truy cập mạng. Do đó, đặc tính bảo mật của CLR cho phép những phần mềm đóng gói trên Internet có nhiều đặc tính mà không ảnh hưởng đến việc bảo mật hệ thống.

CLR còn thúc đẩy cho mã nguồn được thực thi mạnh mẽ hơn bằng việc thực thi mã nguồn chính xác và sự xác nhận mã nguồn. Nền tảng của việc thực hiện này là Common Type System (CTS). CTS đảm bảo rằng những mã nguồn được quản lý thì được tự mô tả.

Sự khác nhau giữa Microsoft và các trình biên dịch ngôn ngữ của hãng thứ ba là việc tạo ra các mã nguồn được quản lý có thể thích hợp với CTS. Điều này thì mã nguồn được quản lý có thể sử dụng những kiểu được quản lý khác và những thể hiện, trong khi thúc đẩy nghiêm ngặt việc sử dụng kiểu dữ liệu chính xác và an toàn.

Thêm vào đó, môi trường được quản lý của runtime sẽ thực hiện việc tự động xử lý layout của đối tượng và quản lý những tham chiếu đến đối tượng, giải phóng chúng khi chúng không còn được sử dụng nữa. Việc quản lý bộ nhớ tự động này còn giải quyết hai lỗi chung của ứng dụng: thiếu bộ nhớ và tham chiếu bộ nhớ không hợp lệ.

Trong khi runtime được thiết kế cho những phần mềm của tương lai, nó cũng hỗ trợ cho phần mềm ngày nay và trước đây. Khả năng hoạt động qua lại giữa mã nguồn

được quản lý và mã nguồn không được quản lý cho phép người phát triển tiếp tục sử dụng những thành phần cần thiết của COM và DLL.

Runtime được thiết kế để cải tiến hiệu suất thực hiện. Mặc dù CLR cung cấp nhiều các tiêu chuẩn dịch vụ runtime, nhưng mã nguồn được quản lý không bao giờ được dịch. Có một đặc tính gọi là Just-in-Time (JIT) biên dịch tất cả những mã nguồn được quản lý vào trong ngôn ngữ máy của hệ thống vào lúc mà nó được thực thi. Khi đó, trình quản lý bộ nhớ xóa bỏ những phân mảnh bộ nhớ nếu có thể được và gia tăng tham chiếu bộ nhớ cục bộ, và kết quả là tăng hiệu quả thực thi.

.NET Framework – Common Type System (CTS)

- Mục đích hỗ trợ thực thi chéo ngôn ngữ
- Tất cả compiler hướng .NET đều phải tuân thủ theo CTS
- Định nghĩa kiểu dữ liệu tiền định và có sẵn trong IL: tất cả ngôn ngữ .NET sẽ được sinh ra mã cuối trên cơ sở kiểu dữ liệu này



Hình 1.4: ví dụ về kiểu dữ liệu chung cho các ngôn ngữ trong .Net

1.2.3 Thư viện lớp .NET Framework

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, như: quản lý chuỗi, thu thập hay chọn lọc dữ liệu, kết nối với cơ sở dữ liệu, và truy cập tập tin. Ngoài ra, 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 kiểu 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.

Các lớp .NET bao gồm các vấn đề

- Đặc tính lõi cung cấp IL, kiểu dữ liệu trong CTS
- Hỗ trợ Win GUI và control
- WebForm (ASP.NET)
- Data Access (ADO.NET)
- Directory Access
- File System, registry access
- Networking and web browsing
- .NET attributes and reflection
- COM interoperability

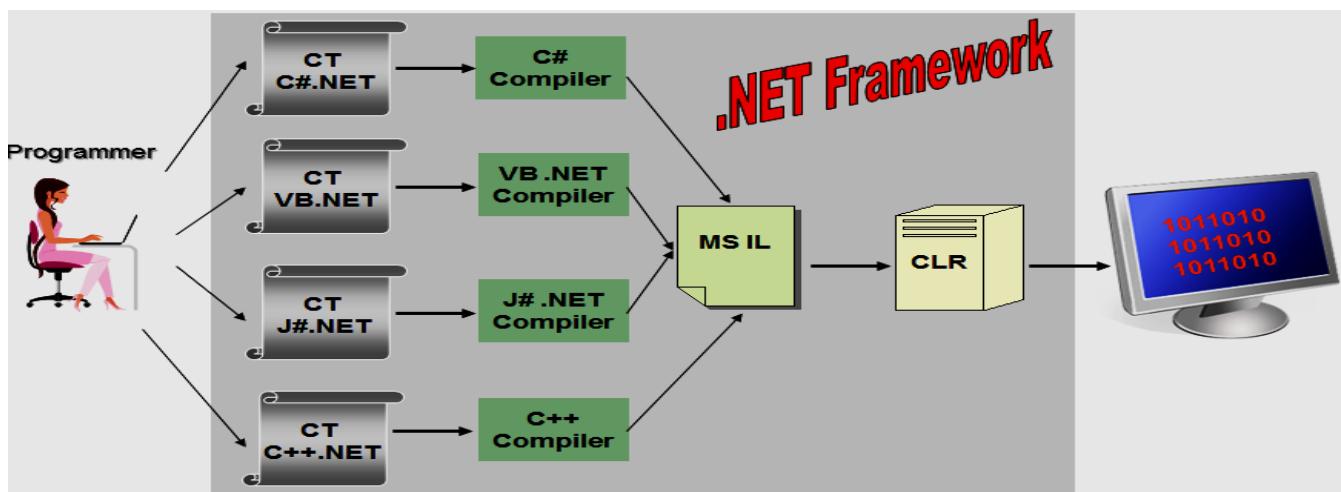
1.2.4 Biên dịch và MS Intermediate Language

Trong .NET Framework, chương trình không được biên dịch vào các tập tin thực thi mà thay vào đó chúng được biên dịch vào những tập tin trung gian gọi là Microsoft Intermediate Language (MSIL). Những tập tin MSIL được tạo ra từ C# cũng tương tự như các tập tin MSIL được tạo ra từ những ngôn ngữ khác của .NET, platform ở đây không cần biết ngôn ngữ của mã nguồn. Điều quan trọng chính yếu của CLR là chung (common), cùng một runtime hỗ trợ phát triển trong C# cũng như trong VB.NET.

Mã nguồn C# được biên dịch vào MSIL khi chúng ta build project. Mã MSIL này được lưu vào trong một tập tin trên đĩa. 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 đa 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.

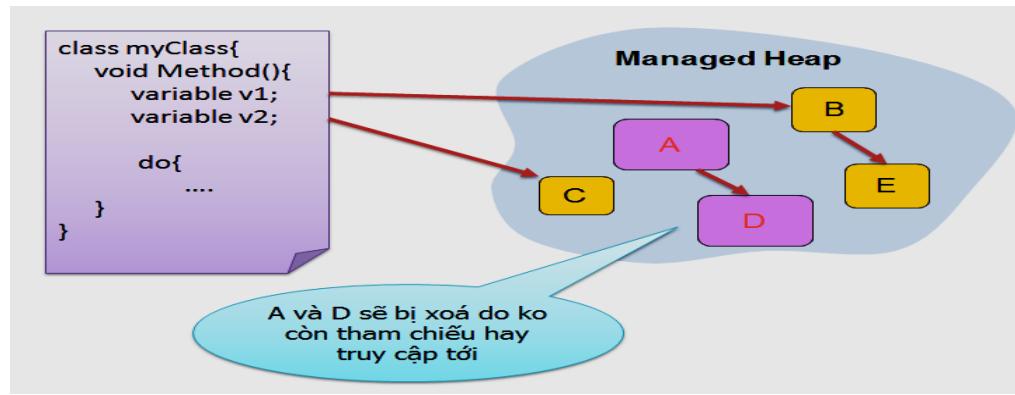
Do tất cả các ngôn ngữ .NET Framework cùng tạo ra sản phẩm MSIL giống nhau, nên kết quả là một đối tượng được tạo ra từ ngôn ngữ này có thể được truy cập hay được dẫn xuất từ một đối tượng của ngôn ngữ khác trong .NET. Ví dụ, người phát triển có thể tạo một lớp cơ sở trong VB.NET và sau đó dẫn xuất nó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



Hình 1.5: quá trình dịch các chương trình trên .Net

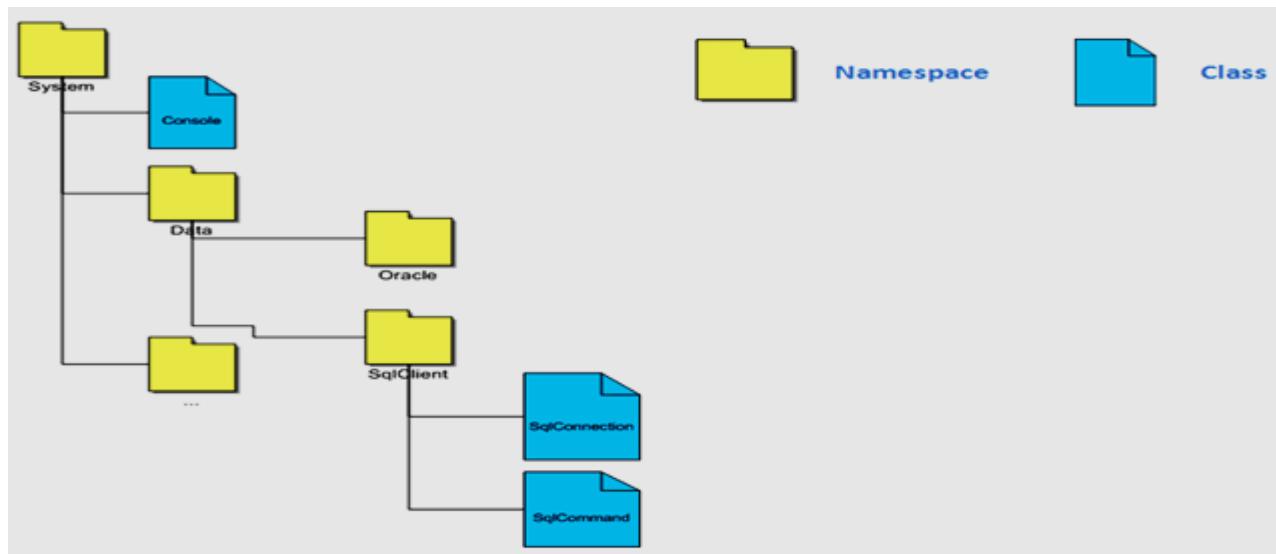
1.2.5 Garbage collection - bộ thu dọn rác

- GC xuất hiện (ko định trước) khi ko đủ bộ nhớ để cung cấp cho ứng dụng.
- GC thực hiện việc tìm kiếm những đối tượng trong managed heap, xoá nếu ko còn tham chiếu tới. Có thể gọi GC một cách tường minh



1.2.6 Namespace

Nhằm giải quyết bài toán xung đột tên class (cùng tên lớp) trong cùng một project. Cấu trúc chương trình được tổ chức theo dạng cây phân cấp như thư mục (directory). Một Namespace có thể chứa nhiều class.



Hình 1.6: ví dụ về một số namespaces, và lớp trong ngôn ngữ C#

Ta có thể hiểu Namespace là một gói những thực thể có thuộc tính và hành vi độc lập với bên ngoài. Những ưu điểm của namespace được liệt kê như sau:

- Tránh được sự trùng lặp tên giữa các class. Cho phép tổ chức mã nguồn một cách có khoa học và hợp lý.
- Cho phép nest
- Truy cập đầy đủ qua tên
- Tất cả data type có tiếp đầu ngữ là tên namespace

```
Khai báo một Namespace:
namespace NamespaceName
{
    // nơi chứa đựng tất cả các class
}
```

Trong đó:

- namespace: là từ khóa khai báo một NameSpace
- NamespaceName: là tên của một Namespace

Ví dụ:

```
namespace CSharpProgram{

    class Basic{

    }

    class Advance{

    }
}
```

Để truy xuất đến 1 class thì phải truy xuất từ Namespace.

Ví dụ: gọi phương thức ghi lên màn hình Console dòng thông báo ta làm như sau:

```
System.Console.WriteLine("Hello C Sharp");
```

Sử dụng từ khoá using để rút gọn việc truy xuất một class: Thường được khai báo ở đầu tài liệu C sharp.

```
using System;
```

và câu lệnh ở trên tương đương với dòng lệnh sau:

```
Console.WriteLine("Hello C Sharp");
```

TÓM TẮT

Trong bài này, học viên làm quen với .Net Framework bao gồm:

- .Net Framework Chương trình nền tảng cho công nghệ .NET. Nó cung cấp tập hợp class library thường dùng, cũng như quản lý sự thực thi của các chương trình .NET
- Common Language runtime là môi trường thực thi trên .Net. CLR đóng vai trò chính trong việc: locate, load, manage .NET types. Và CLR còn quản lý những phần ở mức thấp như: memory management, security check

Đặc điểm của ứng dụng .NET

- Chạy trên nền (.NET framework)
- Mã nguồn được biên dịch qua MSIL
- MSIL được thông dịch qua mã máy lúc thực thi nhờ vào CLR
- Độc lập nền tảng (về lý thuyết có thể chạy trên mọi nền!)
- Install .NET Framework redistribute packadge (dotnetfx.exe) để chạy ứng dụng .NET trên máy client.

CÂU HỎI ÔN TẬP

Câu 1: Một chương trình C# có thể chạy trên bất cứ máy nào?

Câu 2: Nếu muốn đưa chương trình mà ta viết cho một người bạn thì tập tin nào mà chúng ta cần đưa?

Câu 3: Sau khi tạo ra được tập tin thực thi .exe. Có cần thiết giữ lại tập tin nguồn không?

Câu 4: Nếu trình biên dịch C# đưa ra một trình soạn thảo, có phải nhất thiết phải sử dụng nó?

Câu 5: Hãy đưa ra 3 lý do tại sao ngôn ngữ C# là một ngôn ngữ lập trình tốt?

Câu 6: IL và CLR viết tắt cho từ nào và ý nghĩa của nó?

Câu 7: Đưa ra các bước cơ bản trong chu trình xây dựng chương trình?

Câu 8: Trong biên dịch dòng lệnh thì lệnh nào được sử dụng để biên dịch mã nguồn .cs và lệnh này gọi chương trình nào?

Câu 9: Phần mở rộng nào mà chúng ta nên sử dụng cho tập tin mã nguồn C#?

Câu 10: Câu hỏi 7: Ngôn ngữ máy là gì? Khi biên dịch mã nguồn C# ra tập tin .exe thì tập tin này là ngôn ngữ gì?

Câu 11: Tại sao phải khai báo static cho hàm Main của lớp?

Câu 12: Khái niệm và ý nghĩa của namespace trong C#? Điều gì xảy ra nếu như ngôn ngữ lập trình không hỗ trợ namespace?

BÀI 2: NGÔN NGỮ C#

Sau khi học xong bài này, học viên có thể:

- *Tại sao phải sử dụng ngôn ngữ C#*
- *Các đặc điểm của ngôn ngữ C#: C# là ngôn ngữ hiện đại, hướng đối tượng, mạnh mẽ và linh hoạt rất phổ biến*
- *Các bước xây dựng chương trình*
- *Chương trình C# đơn giản*
- *Phát triển chương trình minh họa*
- *Câu hỏi & bài tập.*

2.1 CÁC ĐẶC ĐIỂM CỦA NGÔN NGỮ C#

2.1.1 Giới thiệu

Ngôn ngữ C# ra đời cùng với sự phát triển của công nghệ .Net. Nó là một ngôn ngữ được dẫn xuất từ C và C++, nhưng nó được tạo từ nền tảng phát triển hơn. Microsoft bắt đầu với công việc trong C và C++ và thêm vào những đặc tính mới để làm cho ngôn ngữ này dễ sử dụng hơn. Nhiều trong số những đặc tính này khá giống với những đặc tính có trong ngôn ngữ Java. Không dừng lại ở đó, Microsoft đưa ra một số mục đích khi xây dựng ngôn ngữ này. Những mục đích này được tóm tắt như sau:

- Hướng đối tượng
- Đơn giản, dễ tiếp cận
- Mạnh mẽ (robust) và bền vững (durable)
- Anders Hejlsberg và MS team xây dựng C#

C# loại bỏ một vài sự phức tạp và rắc rối của những ngôn ngữ như Java và C++, bao gồm việc loại bỏ những macro, những template, đa kế thừa, và lớp cơ sở ảo

(virtual base class). Chúng là những nguyên nhân gây ra sự nhầm lẫn hay dẫn đến những vấn đề cho các người phát triển C++. Nếu chúng ta là người học ngôn ngữ này đầu tiên thì chắc chắn là ta sẽ không trải qua những thời gian để học nó! Nhưng khi đó ta sẽ không biết được hiệu quả của ngôn ngữ C# khi loại bỏ những vấn đề trên.

Những đặc tính như là xử lý ngoại lệ, thu gom bộ nhớ tự động, những kiểu dữ liệu mở rộng, và bảo mật mã nguồn là những đặc tính được mong đợi trong một ngôn ngữ hiện đại. C# chứa tất cả những đặc tính trên. Nếu là người mới học lập trình có thể chúng ta sẽ cảm thấy những đặc tính trên phức tạp và khó hiểu. Tuy nhiên, cũng đừng lo lắng chúng ta sẽ dần dần được tìm hiểu những đặc tính qua các chương trong giáo trình này.

Những đặc điểm chính của ngôn ngữ hướng đối tượng (Object-oriented language) là sự đóng gói (encapsulation), sự kế thừa (inheritance), và đa hình (polymorphism). C# hỗ trợ tất cả những đặc tính trên. Phần hướng đối tượng của C# sẽ được trình bày chi tiết trong một chương riêng ở phần sau.

C# là ngôn ngữ mạnh mẽ và cũng mềm dẻo, với ngôn ngữ C# chúng ta chỉ bị giới hạn ở chính bởi bản thân hay là trí tưởng tượng của chúng ta. Ngôn ngữ này không đặt những ràng buộc lên những việc có thể làm. C# được sử dụng cho nhiều các dự án khác nhau như là tạo ra ứng dụng xử lý văn bản, ứng dụng đồ họa, bản tính, hay thậm chí những trình biên dịch cho các ngôn ngữ khác.

2.1.2 Đặc điểm của ngôn ngữ C#

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

- Khái niệm delegate & event
- Garbage Collector, tự động thu hồi vùng nhớ không dùng
- Kiểm soát và xử lý ngoại lệ exception, đoạn mã bị lỗi sẽ không được thực thi
- Type – safe: không cho gán các kiểu dữ liệu khác nhau, đảm bảo sự tương thích giữa lớp con và lớp cha
- Mã nguồn C# (tập tin *.cs) được biên dịch qua MSIL, MSIL: tập tin .exe hoặc .dll, MSIL được CLR thông dịch qua mã máy,
- Dùng kỹ thuật JIT (just-in-time) để tăng tốc độ
- Là ngôn ngữ case sensitive, code phân biệt hoa thường

2.1.3 Các loại ứng dụng C#

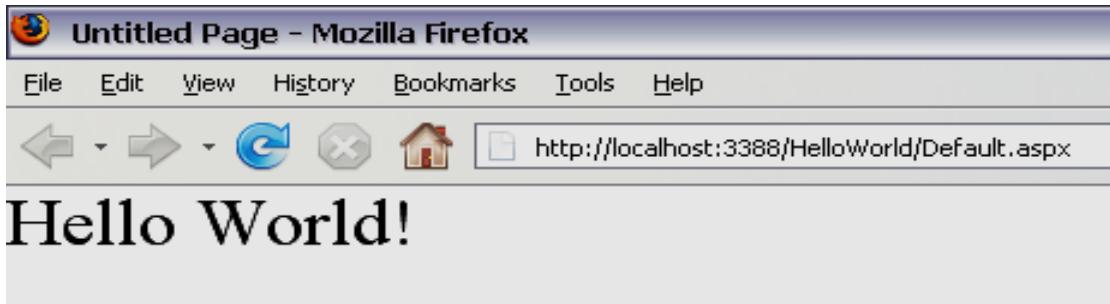
- Chương trình Console (TUI)
 - Giao tiếp với người dùng bằng bàn phím
 - Không có giao diện đồ họa (GUI)



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



- 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



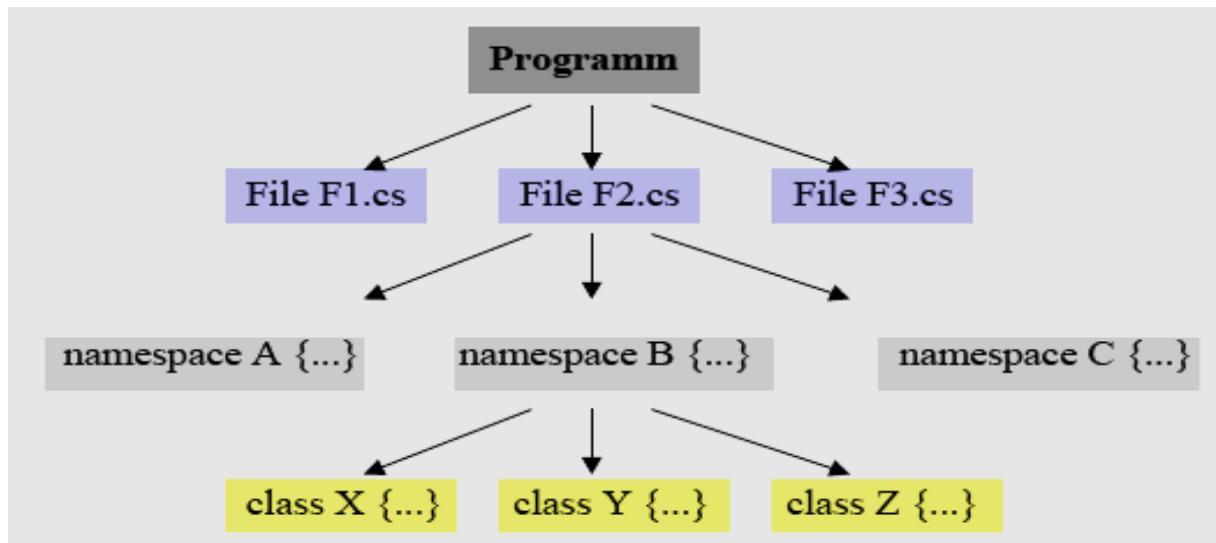
2.1.4 Ứng dụng C#

Thông thường, trong việc phát triển phần mềm, người phát triển phải tuân thủ theo quy trình phát triển phần mềm một cách nghiêm ngặt và quy trình này đã được chuẩn hóa. Tuy nhiên trong phạm vi của chúng ta là tìm hiểu một ngôn ngữ mới và viết những chương trình nhỏ thì không đòi hỏi khắt khe việc thực hiện theo quy trình. Nhưng để giải quyết được những vấn đề thì chúng ta cũng cần phải thực hiện đúng theo các bước sau. Đầu tiên là phải xác định vấn đề cần giải quyết.Nếu không biết rõ vấn đề thì ta không thể tìm được phương pháp giải quyết.Sau khi xác định được vấn đề, thì chúng ta có thể nghĩ ra các kế hoạch để thực hiện.Sau khi có một kế hoạch, thì có thể thực thi kế hoạch này.Sau khi kế hoạch được thực thi, chúng ta phải kiểm tra lại kết quả để xem vấn đề được giải quyết xong chưa.Logic này thường được áp dụng trong nhiều lĩnh vực khác nhau, trong đó có lập trình.

Khi tạo một chương trình trong C# hay bất cứ ngôn ngữ nào, chúng ta nên theo những bước tuần tự sau:

- Xác định mục tiêu của chương trình.
- Xác định những phương pháp giải quyết vấn đề.
- Tạo một chương trình để giải quyết vấn đề.
- Thực thi chương trình để xem kết quả.

2.1.5 Cấu trúc chương trình C#



Hình 2.1: một ví dụ về cấu trúc tệp, không gian tên, và lớp trong dự án

- Phần khai báo dùng namespace (option).

using: làm code gọn hơn, ko cần phải dùng tên của namespace

```

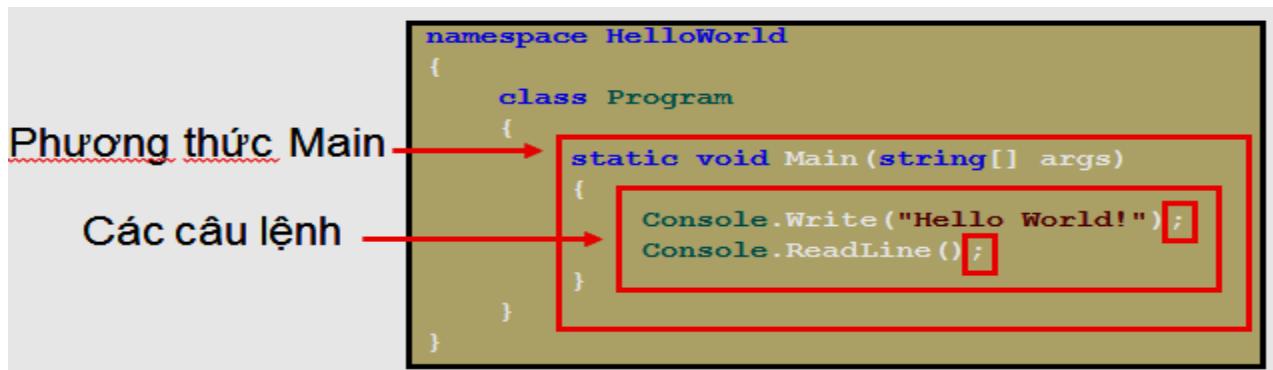
using System;
using System.Collections.Generic;
using System.Text;
  
```

- Phần định nghĩa lớp

class: tối thiểu có 1 lớp chứa hàm entry point Main của chương trình

public static void Main(): hàm entry point của chương trình C#

Các câu lệnh được viết trong thân của phương thức, thực hiện một công việc nào đó, kết thúc bởi dấu chấm phẩy (;



Hình 2.2: Cấu trúc hàm Main của ứng dụng Console trong C#

- Phần chú thích (option)

Chú thích (comment) được dùng để giải thích về chương trình và các câu lệnh, giúp cho chương trình dễ hiểu hơn, được bỏ qua khi biên dịch, không ảnh hưởng tới kết quả thực thi của chương trình

Gõ phần chú thích sau cặp ký tự `//`, hoặc giữa cặp ký tự `/*` và `*/`

```
/* Chương trình C# đầu tiên
   In ra câu chào "Hello World" */
using System;
namespace HelloWorld {
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!"); // Xuất ra câu chào
            Console.ReadLine(); // Chờ nhấn Enter
        }
    }
}
```

Hình 2.3: chương trình C# đơn giản

- XML Comment cho phép phát sinh ra sưu liệu dạng XML, thích hợp cho việc viết sưu liệu của dự án lớn. Chú thích XML bắt đầu với triple slash ("///") và các tag của XML

Chú thích XML dùng cho

- User defined types
- Class, delegate, enum and struct
- Member of user defined types

C# Code without XML Comment

```
using System;
namespace XMLCommentDemo
{
    public class Temperature
    {
        public static int CelsiusToFahrenheit(int degreesCelsius)
        {
            return ((int)((9/5)*degreesCelsius) + 32);
        }

        public static int FahrenheitToCelsius(int degreesFahrenheit)
        {
            return ((int)((5/9)*(degreesFahrenheit - 32)));
        }
    }
}
```

```

using System;
namespace XMLCommentDemo{
    /// <summary>
    /// Class temperature provides functions which convert among various
    /// temperature scales.
    /// </summary>
    public class Temperature {
        /// <summary>
        /// Converts degrees Celsius to degrees Fahrenheit
        /// </summary>
        /// <param name="degreesCelsius">Degrees Celsius</param>
        /// <returns>Returns degrees Fahrenheit</returns>
        public static int CelsiusToFahrenheit(int degreesCelsius) {
            return ((int)((9/5)*degreesCelsius) + 32);
        }
        /// <summary>
        /// Converts degrees Fahrenheit to degrees Celsius
        /// </summary>
        /// <param name="degreesFahrenheit">Degrees Fahrenheit</param>
        /// <returns>Returns degrees Celsius</returns>
        public static int FahrenheitToCelsius(int degreesFahrenheit) {
            return ((int)((5/9)*(degreesFahrenheit - 32)));
        }
    }
}

```

C# Code with XML Comment

48

Hình 2.4: Chú thích XML trong C#

2.2 NGÔN NGỮ C#

2.2.1 Kiểu dữ liệu – Data type

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
- object: kiểu dữ liệu cơ bản của tất cả các kiểu khác
- string: Được sử dụng để lưu trữ những giá trị kiểu chữ cho biến
- int: Sử dụng để lưu trữ giá trị kiểu số nguyên
- byte: sử dụng để lưu trữ giá byte
- float: Sử dụng để lưu trữ giá trị số thực
- bool: Cho phép một biến lưu trữ giá trị đúng hoặc sai
- char: Cho phép một biến lưu trữ một ký tự

Kiểu được người dùng định nghĩa (user-defined) do người lập trình tạo ra: class, struct, enum.

Bảng 2.1: Bảng mô tả các kiểu dữ liệu tiền định, kích thước bộ nhớ, và miền giá trị trong C#

Name	CTS Type	Size	Range
sbyte	System.SByte	8	-128..127
short	System.Int16	16	(-32768 .. 32767)
int	System. Int32	32	-2 ³¹ ..2 ³¹ -1
long	System. Int64	64	-2 ⁶³ ..2 ⁶³ -1
byte	System.SByte	8	0..255
ushort	System.UInt16	16	(0 .. 65535)
uint	System.UInt32	32	0..2 ³² -1
ulong	System.UInt64	64	0..2 ⁶⁴ -1
float	System.Single	32	xấp xỉ từ 3,4E - 38 đến 3,4E+38
double	System.Double	64	1,7E-308 đến 1,7E+308
decimal	System.Decimal	128	Có độ chính xác đến 28 con số
bool	System.Boolean		Kiểu true/false
char	System.Char	16	Ký tự unicode

C# phân tập hợp kiểu dữ liệu này thành hai loại:

- Kiểu dữ liệu giá trị (value) : Một biến (variable) khi được khai báo với kiểu dữ liệu tham trị thì vùng nhớ của nó sẽ chứa giá trị của dữ liệu. Danh sách kiểu dữ liệu tham trị: bool, byte, char, decimal, double, enum, float, int, long, sbyte, short, struct, uint, ulong, ushort
- Kiểu dữ liệu tham chiếu (reference) : Khác với kiểu dữ liệu tham trị, kiểu dữ liệu tham chiếu chỉ lưu trữ địa chỉ tham chiếu tới vùng nhớ chứa giá trị thật sự.

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 struct đều là kiểu dữ liệu tham chiếu.

2.2.2 Từ khóa – Keyword

Mọi ngôn ngữ đều chứa những từ khóa và những từ này hiểu được bởi người nói ra nó. Điều đó cũng đúng với C#. Từ khóa trong C# là những từ đặc biệt và mang nghĩa

đặc biệt chỉ dành riêng cho ngôn ngữ này. Trong VS.net những từ khóa của C# sẽ có màu xanh ra trờ. Trong ví dụ trên các từ khóa là using, namespace, int

Các từ khóa trong C#							
abstract	const	extern	in	operator	sbyte	throw	virtual
as	continue	false	int	out	sealed	true	void
base	decimal	finally	interface	override	set	try	volatile
bool	default	fixed	internal	params	short	typeof	where
break	delegate	float	is	partial	sizeof	uint	while
byte	do	for	lock	private	stackalloc	ulong	yield
case	double	foreach	long	protected	static	unchecked	
catch	else	get	namespace	public	string	unsafe	
char	enum	goto	new	readonly	struct	ushort	
checked	event	if	null	ref	switch	using	
class	explicit	implicit	object	return	this	value	

Hình 2.5 Các từ khóa trong C#

2.2.3 Quy tắc đặt tên, biến, hằng

2.2.3.1 Quy tắc đặt tên

Khi bạn đặt tên cần chú ý đến các nguyên tắc sau:

- Bao gồm chữ cái, chữ số, ký tự gạch dưới
- Không được bắt đầu bằng chữ số
 - **Chuong_Trinh, x25, z, _abc, XửLý** → hợp lệ
 - **2abc, Chuong-Trinh, Xu Ly, class** → không hợp lệ
- Phân biệt CHỮ HOA và chữ thường
 - **ChuongTrinh** và **chuongtrinh** là khác nhau
- Các định danh được khai báo trong cùng phạm vi (scope) không được trùng nhau
- Phải khác với từ khóa (dùng "@" khắc phục)

2.2.3.2 Biến

- Biến là nơi lưu trữ dữ liệu của chương trình.
- Phải khai báo trước khi dùng
- Dữ liệu của biến: nằm trong bộ nhớ vật lý (physical RAM), có thể thay đổi giá trị

- Phạm vi (scope): từ vị trí khai báo biến, được xác định bởi cặp dấu { và }, có thể sử dụng ở phạm vi nhỏ hơn
 - Trong thân phương thức: biến cục bộ
 - Trong thân lớp: thuộc tính
 - Biến trong C# chỉ có tác dụng trong phạm vi mà nó được khai báo
 - Đặt tên biến theo quy tắc đặt tên của C#
 - Khai báo biến: **<KIỂU DỮ LIỆU> <TÊN BIẾN>;**
- C# là ngôn ngữ phân biệt hoa thường nên, 2 biến này là 2 đối tượng khác nhau.

Ví dụ:

- int bien1;
- int Bien1;

2.2.3.3 HẰNG

- Một hằng là một biến nhưng trị không thay đổi


```
const int a = 100; // giá trị ko thể thay đổi
```
- Hằng bắt buộc phải được gán giá trị lúc khai báo
- Trị của hằng có thể được tính toán vào lúc biên dịch
- Hằng bao giờ cũng static
- Ưu điểm
 - Chương trình dễ đọc, khắc phục những con số "magic number" trong code, chương trình dễ sửa hơn.
 - Tránh lỗi dễ dàng hơn, trình biên dịch sẽ báo lỗi nếu gán lại giá trị cho hằng
- Ví dụ minh họa cách sử dụng biểu tượng hằng:

```
class MinhHoaC3
{
    const int DoSoi = 100; // Độ C
    const int DoDong = 0; // Độ C
    static void Main()
```

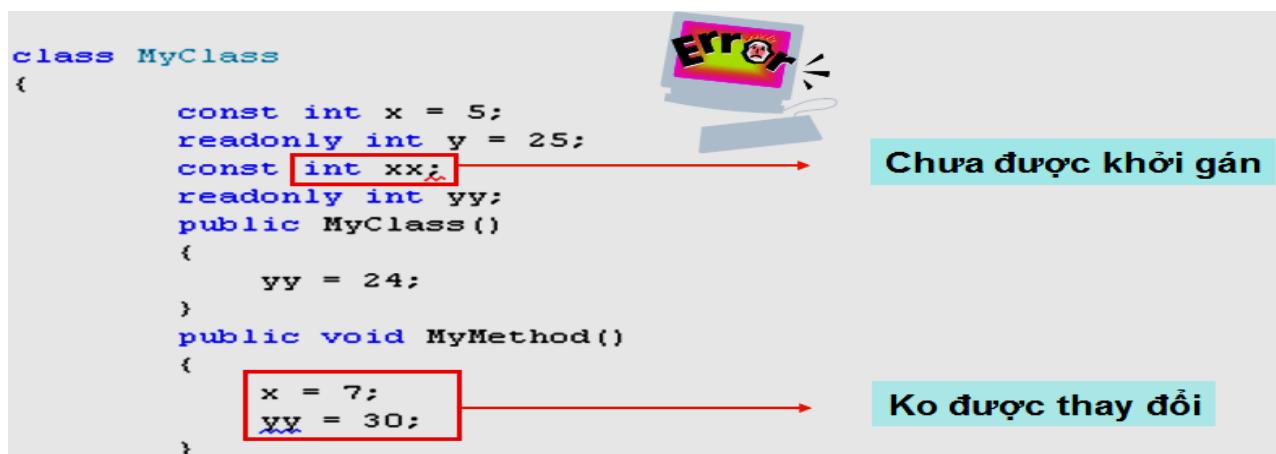
```

    {
        System.Console.WriteLine("Do dong cua nuoc {0}", DoDong);
        System.Console.WriteLine("Do soi cua nuoc {0}", DoSoi);
    }
}

```

2.2.4 Biến readonly

- const: phải được gán giá trị khi khai báo
- readonly: ko cần khởi tạo trước, khi gán giá trị thì sau đó ko thay đổi được



Hình 2.6: ví dụ về cách sử dụng hằng và biến chỉ đọc

Biến readonly thường được dùng để nạp cấu hình của hệ thống từ file config, hoặc database, và tránh không cho thay đổi sau khi gán giá trị.

2.2.5 Chuyển đổi các kiểu dữ liệu trong C#

Trong C# cung cấp cho chúng ta rất nhiều cách ép kiểu nhưng ở đây tớ chỉ xin giới thiệu tới mọi người 4 cách:

2.2.5.1 Parse

Phương thức Parse là phương thức được sử dụng khá phổ biến khi chúng ta muốn chuyển đổi một chuỗi sang một kiểu dữ liệu tương ứng.

Mỗi kiểu dữ liệu cơ bản trong C# đều có phương thức Parse để chuyển đổi sang kiểu dữ liệu đó. Một số ví dụ các câu lệnh minh họa cho việc chuyển đổi sử dụng phương thức Parse.

Ví dụ:

```
Int32 a = Int32.Parse("123"); //a sẽ mang giá trị 123
```

```
Float b = Float.Parse("20.7"); //b sẽ mang giá trị 20.7
```

```
bool c = Boolean.Parse("true"); //c sẽ mang giá trị true
```

Nếu như chuỗi chúng ta truyền vào là rỗng, không đúng định dạng hoặc vượt quá giá trị cho phép thì chúng ta sẽ nhận được các Exception tương ứng.

```
int a = Int32.Parse("Hello"); //sai định dạng, FormatException
```

```
byte b = Byte.Parse("10000000000"); //quá giới hạn, OverflowException
```

```
bool c = Boolean.Parse(null); //tham số là null, ArgumentNullException
```

2.2.5.2 TryParse

Giống như Parse, TryParse cũng là phương thức được tích hợp sẵn trong các lớp kiểu dữ liệu cơ bản của C#. Tuy nhiên, cú pháp của TryParse có phần khác với Parse. Cụ thể, tham số thứ nhất của TryParse là chuỗi cần chuyển đổi và tham số thứ hai là biến sẽ chứa giá trị đã được chuyển đổi, biến thứ hai này phải được đánh dấu là out .

```
bool b = Int32.TryParse("123", out a); //a sẽ có giá trị 123, kết quả trả về true
```

```
Boolean.TryParse("false", out b); //b sẽ mang giá trị false
```

```
bool c = Boolean.TryParse("", out b); //trả về c có giá trị false, b mang giá trị false
```

Điểm khác biệt thứ hai của TryParse so với Parse là phương thức TryParse không ném ra các ngoại lệ như Parse mà sẽ trả về các giá trị true (chuyển đổi thành công) hoặc false (chuyển đổi thất bại, biến mang giá trị mặc định).

Chú ý: Ngoài ra, phương thức TryParse sẽ thực thi nhanh hơn phương thức Parse vì TryParse không ném ra ngoại lệ

2.2.5.3 Convert

Lớp Convert là một lớp tiện ích trong C# cung cấp cho chúng ta rất nhiều phương thức tĩnh khác nhau để chuyển đổi từ một kiểu dữ liệu này sang kiểu dữ liệu khác. Tham số mà các phương thức trong Convert nhận không nhất thiết phải là chuỗi mà có thể ở nhiều kiểu dữ liệu khác nhau (int, bool, double...). Ví dụ:

```
int a = Convert.ToInt32("123"); //chuyển chuỗi 123 sang số nguyên
```

```
bool b = Convert.ToBoolean(13); //chuyển số 13 sang kiểu bool
```

Các phương thức trong lớp Convert sẽ trả về giá trị mặc định nếu như tham số truyền vào là null. Còn trong các trường hợp sai định dạng hoặc vượt quá giới hạn thì các phương thức đó sẽ ném ra các ngoại lệ tương tự như phương thức Parse. Ví dụ:

```
bool a = Convert.ToBoolean("khoaimon"); //FormatException
```

```
int b = Convert.ToInt32("123456787654"); //OverflowException
```

```
double d = Convert.ToDouble(null); //trả về giá trị mặc định
```

2.2.5.4 Casting - Ép kiểu

Ép kiểu là cách chúng ta có thể sử dụng khi muốn chuyển đổi giữa các kiểu dữ liệu có tính chất tương tự nhau.

- Ép từ kiểu lớn qua kiểu nhỏ: có thể mất giá trị (thường là số).

Ví dụ :

```
double x = 74.86;
int a = x;
float b = a; //chuyển đổi ngầm định, b = 100
int c = (int)b; //chuyển đổi rõ ràng, c = 100
```

- Ép từ lớp cơ sở qua lớp dẫn xuất. Ví dụ

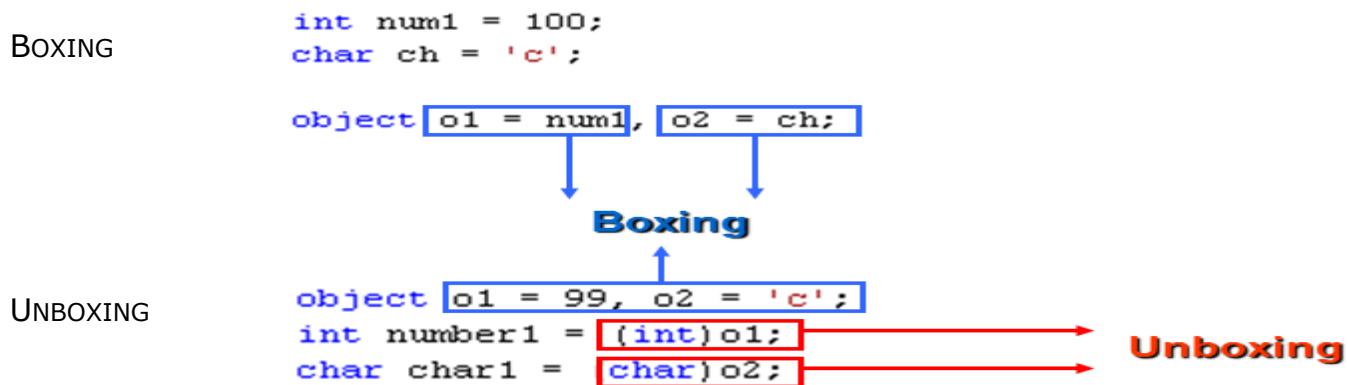
```
Object b = a; //boxing, b là kiểu tham chiếu chứa giá trị 100
int c = (int)b; //unboxing, c mang giá trị 100
string s = "Hello";
object o = s;
string s2 = o as String; // as là từ khóa chuyển đổi kiểu
```

Nhược điểm của việc sử dụng casting thuận túy là nếu việc casting thất bại thì chúng ta sẽ nhận được một exception cho việc thất bại đó. Tuy nhiên, nếu sử dụng

toán tử “as”, nếu việc casting không thành công thì chúng ta sẽ nhận về một giá trị null thay vì là một exception.

2.2.5.5 Boxing & Unboxing

Kiểu giá trị có thể được chuyển thành kiểu đối tượng



Hình 2.6: ví dụ về boxing và unboxing

2.2.5.6 CONSOLE I/O

Để đọc ký tự văn bản từ cửa sổ console. Ta có:

- Console.Read() giá trị trả về là int
- Console.ReadLine() giá trị trả về là string

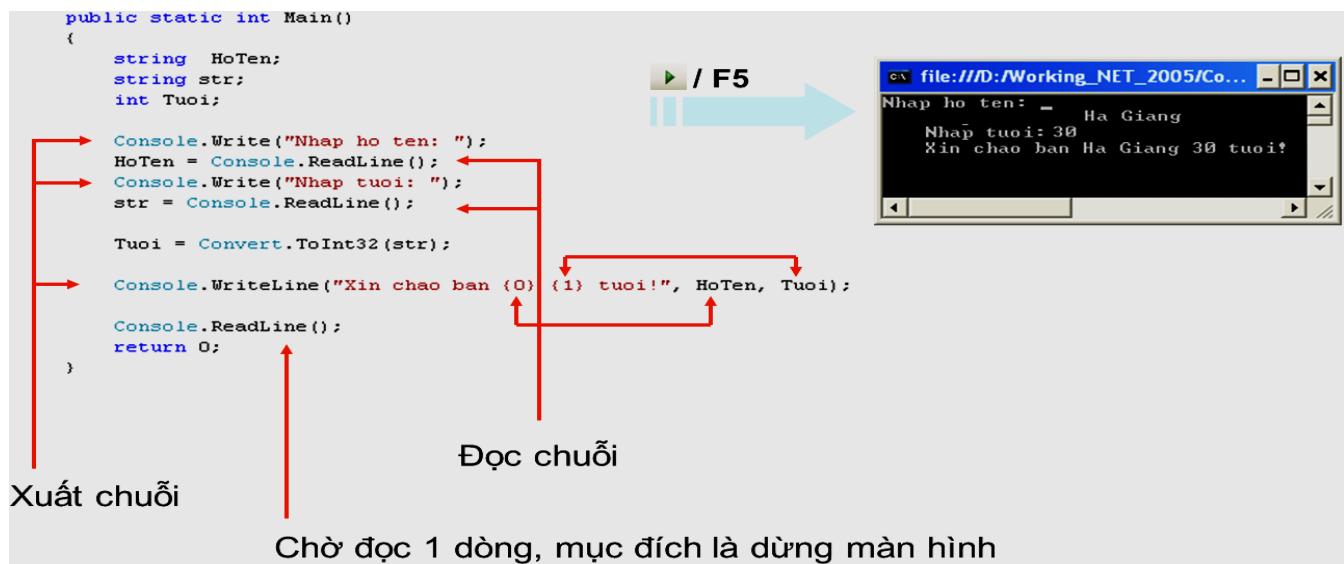
Để xuất chuỗi ký tự dùng ra màn hình. Trong đó:

- Console.WriteLine() cho phép xuất chuỗi ký tự ra màn hình theo định dạng:

```

int i = 10;
int j = 20;
Console.WriteLine("{0} plus {1} equals {2}", i, j, i + j);
  
```

- Console.WriteLine(): giống như Console.Write() nhưng sau khi xuất chuỗi ký tự thì con trỏ sẽ chuyển sang hàng mới cho lần xuất tiếp theo.

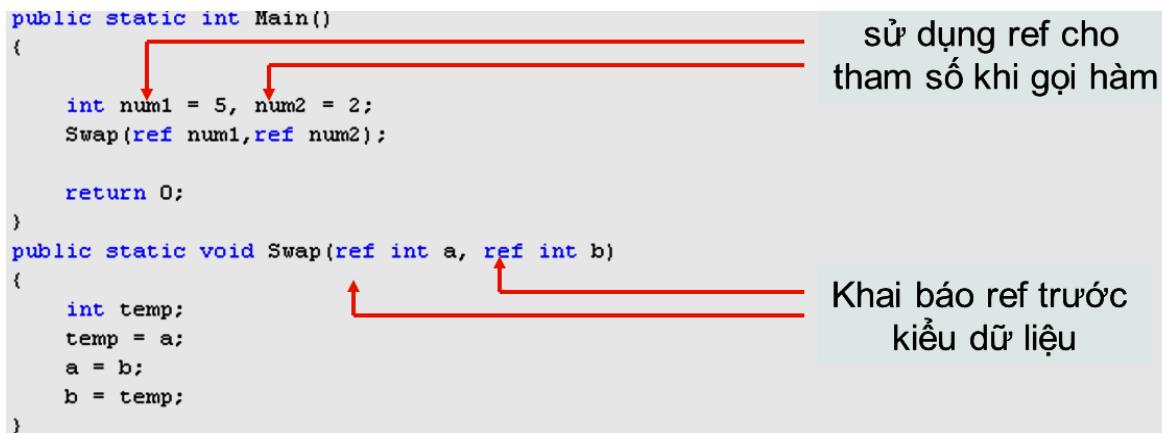


Hình 2.7: ví dụ nhập xuất trên ứng dụng console

2.2.5.7 Tham số ref, out

ref: tương tự như truyền tham chiếu trong C/C++. từ khoá ref phải được dùng lúc gọi hàm, các tham số truyền dạng ref phải được khởi tạo giá trị trước

Ví dụ:



out: tương tự như ref, khác ref là ko cần khởi tạo giá trị trước khi truyền

```

public static int Main()
{
    int a = 10, b;
    Subtraction(a, out b);
    return 0;
}
public static void Subtraction(int x, out int y)
{
    y = x - 2;
}

```

Dùng trước tham số khi gọi hàm

Khai báo cho tham số

Hình 2.8: ví dụ tham số tham chiếu (ref), tham số đầu ra (out)

2.2.5.8 Cấu trúc điều khiển

C# cung cấp hai cấu trúc điều khiển thực hiện việc lựa chọn điều kiện thực thi chương trình đó là cấu trúc if và switch...case

Cú pháp:

```

if (biểu thức điều kiện)
{
    // câu lệnh thực thi nếu biểu thức điều kiện đúng
}
[else
{
    // câu lệnh thực thi nếu biểu thức điều kiện sai
}]

```

Ví dụ:

```

if (20 % 4 > 0)
{
    Console.WriteLine("Số 20 không chia hết cho 4");
}
else
{
    Console.WriteLine("Số 20 chia hết cho số 4");
}

```

- **Cấu trúc switch ... case:**

```

// switch ... case
switch (Biến điều kiện)
{
    case giá trị 1:
        // Câu lệnh thực thi
        break;
    case giá trị 2:
        // Câu lệnh thực thi
}

```

```

        break;
    case giá trị 3:
        // Câu lệnh thực thi
        break;
    default:
        // Câu lệnh thực thi
        break;
}

```

- Biểu thức switch gồm: kiểu số, ký tự, enum và chuỗi
- Sử dụng break, goto, return để điều khiển luồng thực thi
- Nếu ko nhãn nào phù hợp → default
- Nếu ko có default → thực hiện lệnh sau switch

2.2.5.9 Vòng lặp

C# cung cấp một bộ mở rộng các câu lệnh lặp, bao gồm các câu lệnh lặp for, while do... while giống như C++, và bổ sung thêm foreach để duyệt tập hợp.

- Vòng lặp While: thực thi câu lệnh hoặc một loạt những câu lệnh đến khi điều kiện không được thỏa mãn.

Cú pháp:

```

while (biểu thức điều kiện) {
    // câu lệnh
}

```

Ví dụ:

```

using System;
class WhileTest
{
    public static void Main()
    {
        int n = 1;
        while (n < 10)
        {
            Console.WriteLine("Current value of n is {0}", n);
            n++;
        }
    }
}

```

- Vòng lặp do .. while: thực thi câu lệnh ít nhất một lần đến khi điều kiện không được thỏa mãn.

Cú pháp:

```
do {
    // câu lệnh
} while (biểu thức điều kiện);
```

Ví dụ:

```
using System;
public class TestDoWhile
{
    public static void Main ()
    {
        int x;
        int y = 0;
        do
        {
            x = y++;
            Console.WriteLine(x);
        }
        while(y < 10)
    }
}
```

- Vòng lặp for

```
for ([ phần khởi tạo] ; [biểu thức điều kiện]; [bước lặp])
{
    // thực thi câu lệnh
}
```

Ví dụ:

```
using System;
public class ForLoopTest
{
    public static void Main()
    {
        for (int i = 1; i <= 5; i++)
            Console.WriteLine(i);
    }
}
```

- Vòng lặp foreach: lệnh foreach cho phép chúng ta lặp qua tất cả các mục trong một mảng hay trong một tập hợp.

Cú pháp:

```
foreach (var item in collection )
{
    // thực hiện thông qua tương ứng với từng mục trong mảng hay tập hợp
}
```

Dữ liệu kiểu tập hợp chưa được đề cập tới trong các bài học trước nên bạn chỉ cần quan tâm đến vòng lặp foreach sử dụng với mảng.

Ví dụ:

```
using System;
public class UsingForeach
{
    public static int Main()
    {
        int[] intArray = {1,2,3,4,5,6,7,8,9,10};
        foreach( int item in intArray)
        {
            Console.Write("{0} ", item);
        }
        Console.ReadLine();
        return 0;
    }
}
```

Kết quả:

```
0 1 2 3 4 5 6 7 8 9 10
```

2.2.5.10 Array

- Chứa một số những biến có cùng kiểu dữ liệu.
- Truy xuất phần tử thông qua chỉ số (index)
- Chỉ số bắt đầu bằng 0.

Ví dụ:

```
string[] thuTT = {"Hai","Ba","Tư","Năm","Sáu","Bảy","Chủ Nhật"};
```

//khai báo mảng chuỗi các phần tử kiểu string lưu trữ các ngày trong tuần

```
int[] myInteger = new int[5];
```

- Lấy kích thước mảng qua thuộc tính Length

```
int Size = myArray.Length;
```

- Nếu thành phần của mảng là kiểu định trước, có thể dùng hàm Sort của Array để sắp xếp.

```
Array.Sort(myArray);
```

- Dùng hàm Reverse của Array để đảo thứ tự các phần tử trong mảng.

```
Array.Reverse(myArray);
```

Ví dụ:

```
public static int Main()
{
    string[] artists = { "Leonardo", "Monet", "Van Gogh", "Klee" };

    Array.Sort(artists);
    foreach (string name in artists)
        Console.WriteLine(name);

    Array.Reverse(artists);
    foreach (string name in artists)
        Console.WriteLine(name);

    Console.ReadLine();
    return 0;
}
```

Dùng phương thức tĩnh Sort
của lớp Array để sort artists

Dùng phương thức tĩnh Reverse
của lớp Array để đảo thứ tự artists

Hình 2.9: một ví dụ về mảng, sử dụng các method có sẵn của lớp Array

2.2.5.11 Enumeration

- Là một kiểu dữ liệu, mà mỗi phần tử của tập hợp là một hằng số, và có thể liệt kê khi khai báo. Tập hợp các giá trị hằng được đặt tên.
- Khai báo trực tiếp trong namespace.
- Enumeration kế thừa từ lớp object.

Ví dụ:

```
enum Color { Red, Green, Blue };
enum Access { personal = 1, group = 2, all = 4 };
```

Sử dụng

```
Color c = Color.Red;

Access a = Access.personal | Access.group;
If ((Access.personal & a) != 0)
    Console.WriteLine("access granted");
```

- Sau đây là bảng toán tử áp dụng cho Enum:

Compare	<code>if (c == Color.red) ... if (c > Color.red && c <= Color.green) ... c = c + 2;</code>
<code>+, -</code>	<code>c++;</code>
<code>++, --</code>	<code>if ((c & Color.red) == 0) ... c = c Color.blue;</code>
<code>&</code>	<code>c = ~ Color.red;</code>
<code> </code>	
<code>~</code>	

Ví dụ:

```
public enum TimeOfDay
{
    Morning = 0,
    Afternoon = 1,
    Evening = 2
}
public static int Main()
{
    TimeOfDay day;
    day = TimeOfDay.Evening;
    WriteGreeting(day);
    return 0;
}
static void WriteGreeting(TimeOfDay timeOfDay)
{
    switch (timeOfDay)
    {
        case TimeOfDay.Morning:
            Console.WriteLine("Good morning!");
            break;
        case TimeOfDay.Afternoon:
            Console.WriteLine("Good afternoon!");
            break;
        case TimeOfDay.Evening:
            Console.WriteLine("Good evening!");
            break;
        default:
            Console.WriteLine("Hello!");
            break;
    }
}
```

Hình 2.9: khai báo và sử dụng Enum

TÓM TẮT

Cú pháp khá giống với C/C++

Kiểu dữ liệu tham chiếu & giá trị

Truyền tham số kiểu giá trị cho hàm

Type-cast

Boxing & Unboxing

Điều khiển lặp foreach duyệt tập hợp

Kiểu dữ liệu mảng.

CÂU HỎI ÔN TẬP

Câu 1: Những ngôn ngữ nào khác được xem như là hướng đối tượng?

Câu 2: Tại sao trong kiểu số không nên khai báo kiểu dữ liệu lớn thay vì dùng kiểu dữ liệu nhỏ hơn?

Câu 3: Những ngôn ngữ nào khác hỗ trợ Common Type System (CTS) trong Common

Câu 4: Có thể sử dụng chuỗi với câu lệnh switch?

Câu 5: Có bao nhiêu cách khai báo comment trong ngôn ngữ C#, cho biết chi tiết?

Câu 6: Những từ sau từ nào là từ khóa trong C#: field, cast, as, object, throw, football, do, get, set, basketball.

Câu 7: Những khái niệm chính của ngôn ngữ lập trình hướng đối tượng?

Câu 8: Sự khác nhau giữa hai lệnh Write và WriteLine?

Câu 9: C# chia làm mấy loại kiểu dữ liệu? Nếu ta tạo một lớp tên myClass thì lớp này được xếp vào kiểu dữ liệu nào?

Câu 10: Kiểu chuỗi trong C# là kiểu dữ liệu giá trị, hay kiểu dữ liệu tham chiếu?

Câu 11: Dữ liệu của biến kiểu dữ liệu tham chiếu được lưu ở đâu trong bộ nhớ?

Câu 12: Kiểu dữ liệu nào trong .NET tương ứng với kiểu int trong C#?

Câu 13: Kết quả của $15\%4$ là bao nhiêu?

Câu 14: Sự khác nhau giữa chuyển đổi tường minh và chuyển đổi ngầm định?

Câu 15: Có thể chuyển từ một giá trị long sang giá trị int hay không?

Câu 16: Số lần tối thiểu các lệnh trong while được thực hiện?

Câu 17: Số lần tối thiểu các lệnh trong do while được thực hiện?

Câu 18: Lệnh nào dùng để thoát ra khỏi vòng lặp?

Câu 19: Lệnh nào dùng để qua vòng lặp kế tiếp?

Câu 20: Khi nào dùng biến và khi nào dùng hằng?

Câu 21: Cho biết giá trị CanhCut trong kiểu liệt kê sau:

```
enum LoaiChim
```

```
{
```

```
HaiAu,
```

```
BoiCa,
```

```
DaiBang = 50,
```

```
CanhCut
```

```
}
```

Câu 22: Cho biết các lệnh phân nhánh trong C#?

BÀI 3: LỚP VÀ GIAO DIỆN

Sau khi học xong bài này, học viên có thể nắm được:

- *Khai báo lớp*
- *Constructor & destructor*
- *Hàm thành viên*
- *Thuộc tính*
- *Đa hình trong C#*
- *Down cast – up cast*
- *Abstract class*
- *Sealed class*
- *Interface*

3.1 LỚP - CLASS

3.1.1 Giới thiệu

Class có thể được xem như những “khuôn mẫu” được định nghĩa trong chương trình của bạn để “đúc” ra những đối tượng cần thiết; cũng giống như trên thực tế, trong lĩnh vực hàng gia dụng, để “đúc” ra những đôi dép bán cho người tiêu dùng sử dụng, nhà sản xuất sẽ phải chế tạo, thiết kế ra các “khuôn mẫu” cần thiết cho loại dép sẽ sản xuất, sau đó dựa vào nguyên liệu và khuôn mẫu đã chế tạo sẽ sản xuất ra các sản phẩm là những đôi dép cung cấp cho thị trường tiêu dùng.

Trong lập trình hướng đối tượng, bạn cần phải định nghĩa Class như là những khuôn mẫu cần thiết để có thể sản xuất ra các đối tượng sẽ dùng đến cho mục đích nào đó trong chương trình của mình. Trong minh họa ở trên, hạt nhựa chính là nguyên liệu để tạo ra sản phẩm là những đôi dép, trong chương trình của bạn thì nguyên liệu để tạo ra đối tượng chính là dữ liệu. Dữ liệu ở đây có thể là các kiểu dữ liệu cơ bản (Primitive data type) mà C# cung cấp hoặc cũng có thể là những kiểu dữ liệu đặc biệt do bạn tạo ra như Struct, Enum hay những Class khác cần dùng cho việc lưu trữ thông tin trong Class muốn xây dựng.

Trước khi xây dựng chương trình, bạn thường phải thiết kế dựa trên những ý tưởng nào đó. Giả dụ trong Game về đua xe, người chơi được phép chọn xe có màu gì, ứng với loại xe đó thì tốc độ di chuyển là bao nhiêu, xe đó phải xử dụng nhiên liệu là xăng, dầu hay gas ... Hoặc giả như với game “Đá đấm” ở trên, chiến binh của người chơi có thể sẽ có chiều cao, cân nặng, hình dáng như thế nào, và giả sử các chiến binh trong chương trình của bạn đều có khả năng đá, đấm, phi thân khỏi mặt đất thì ứng với mỗi thể trạng của chiến binh mà cú đá, cú đấm, cú đấm ánh hưởng ra sao đến đối thủ, phi thân có thể cao hay thấp ...

Các thành phần cơ bản cần có trong class của lập trình hướng đối tượng: properties (thuộc tính) và method (phương thức), mỗi chiếc xe có thể có màu sơn khác nhau, sử dụng nguyên liệu khác nhau: đây chính là các properties cần có trong class “xeHoi” ở trên. Mỗi chiến binh có chiều cao, cân nặng, hình dáng khác nhau: đây chính là những properties cần xây dựng trong class “chienBinh”.

Mỗi loại xe có thể di chuyển tốc độ khác nhau trên mỗi loại địa hình, mỗi loại xe có ưu thế khác nhau khi vào cua tại những khúc quanh có góc mở khác nhau ... tốc độ, khả năng vào cua theo địa hình chính là method cho class “xeHoi”. Mỗi chiến binh có cú đá, cú đấm khác nhau hay khả năng của mỗi chiến binh khi phi thân là cao hay thấp, như vậy các khả năng đá, đấm và phi thân được xem như method của class “chienBinh”.

Khi xây dựng class trong C#, bạn phải định nghĩa các properties cùng với method cần thiết để có thể sử dụng cho đối tượng của mình trong chương trình.

3.1.2 Khai báo

```
[access modifier] class <tên_class>[: base class]
{
    [access modifier]<kieu_dl> <bien_thanh_vien>;
    ...
    [access modifier]<phương_thức>(tham_số, ...)
}
```

Trong đó:

- access modifier: khóa mô tả phạm vi truy cập. Có 4 khóa truy cập, public, protected, internal, private, trong đó chỉ public, và internal được sử dụng khi khai

báo lớp, còn 5 khóa còn lại được sử dụng cho khai báo dữ liệu, thuộc tính (properties), phương thức (methods), và sự kiện (events)

- Một class chứa trong namespace chỉ có 2 khóa truy xuất (mặc định khi không khai báo là Internal)
- public: cho phép bên ngoài assembly truy xuất
- internal: chỉ cho phép sử dụng bên trong assembly
- Access modifier cho khai báo dữ liệu, thuộc tính (properties), phương thức (methods), và sự kiện (events) là private:

Bảng mô tả tầm ảnh hưởng của Access Modifiers			
	Có thể truy xuất trong phạm vi chương trình	Có thể truy xuất trong phạm vi của lớp chứa khai báo	Có thể truy xuất từ các lớp thừa kế của lớp chứa khai báo
public	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
private	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
protected	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
internal	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

Hình 3.1: các từ khóa phạm vi truy cập trong C#

- Nếu ko khai báo lớp cơ sở thì C# mặc định xem lớp cơ sở là object
- Lớp luôn là kiểu dữ liệu tham chiếu trong C#
- Assembly là tập mã đã được biên dịch sang .NET
- Một assembly chứa nội dung thực thi chương trình hay thư viện động
- Assembly có thể chứa trong nhiều file
- Ví dụ:** định nghĩa 1 lớp (class) dùng cho mục đích chứa thông tin về học sinh cần quản lý trong chương trình với các thông tin lưu trữ mô tả cho các đặc điểm sau: mã số học sinh, họ tên, giới tính, số điện thoại, năm sinh.

```

public class HocSinh{
    int      _maHS;           //--- mã số của học sinh
    string   _hoTen;          //--- họ tên của học sinh
    bool     _gioiTinh;        //--- Nếu nam :true, ngược lại là false
    long    _soDT;            //--- Số điện thoại cá nhân
    int      _namSinh;         //--- Năm sinh của học sinh
}
  
```

Lớp có thể chứa các phần sau

- Constructor và destructor
- Field và constant
- Method
- Property
- Indexer
- Event
- Chứa các kiểu khác (nested): class, struct, enumeration, interface và delegate
- Tạo đối tượng: khai báo trong thân lớp, giống như thuộc tính, Trong thân phương thức, tương tự như biến
- Khởi tạo: bằng lệnh new



3.1.3 Constructor

Constructors là những hàm đặc biệt cho phép thực thi, điều khiển chương trình ngay khi khởi tạo đối tượng. Trong C#, Constructors có tên giống như tên của class và không trả lại giá trị:

- Cùng tên với lớp, được gọi tự động khi tạo đối tượng.
- Constructor ko tham số sẽ được tạo mặc định khi không có bất cứ constructor nào.
- Cho phép overload constructor để tạo ra nhiều cách khởi tạo đối tượng.
- Constructor mặc định:
 - Không có tham số.
 - Khởi tạo thể hiện (đối tượng) khi chưa biết thông tin gì về nó.
- Constructor sao chép:

- Tham số vào là đối tượng cùng lớp.
 - Tạo ra obj như bản sao của obj đầu vào.
- Constructor khác:
- Có một hay nhiều tham số vào.
 - Tạo object khi biết một số thông tin nào về nó.

```
class HocSinh
{
    //...
    public HocSinh()
    {
        hoTen = "unknown";
        namSinh = 1990;
        diemVan = diemToan = 0;
    }
    public HocSinh(HocSinh hs)
    {
        hoTen = hs.hoTen;
        namSinh = hs.namSinh;
        diemVan = hs.diemVan;
        diemToan = hs.diemToan;
    }
    public HocSinh(string ht)
    {
        hoTen = ht;
    }
}
```

Constructor mặc định

Constructor sao chép

**Constructor khác
(tạo học sinh khi biết họ tên)**

Hình 3.2 ví dụ xây dựng lớp Học sinh và các hàm khởi tạo

3.1.4 Destructor

Mỗi lớp chỉ có 1 destructor, nó thực hiện nhiệm vụ “clean” khi đối tượng bị hủy:

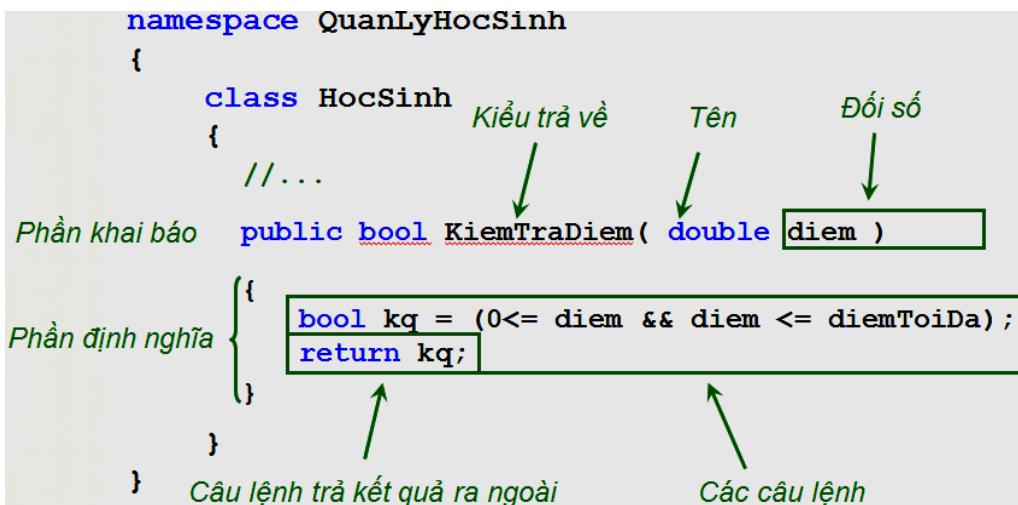
- Trùng tên lớp và có dấu “~” phía trước
- Không có tham số và access modifier

```
class HocSinh
{
    //...
    ~HocSinh()
    {
        siso--;
    }
}
```

3.1.5 Method

Method hay còn gọi là phương thức mô tả chức năng, hành vi giao tiếp với bên ngoài của lớp, là thủ tục khai báo trong class. Có 2 loại: static và non static

- Khai báo một non static method:



Hình 3.3: cấu trúc một phương thức non static

- Phương thức tĩnh (khai báo kèm với từ khóa static) hoạt động như phương thức toàn cục với tất cả các đối tượng của lớp. Khi cài đặt không được phép sử dụng bất kỳ thành phần dữ liệu, hay phương thức non static của lớp.
- Khi gọi non static method, ta gọi thông qua thể hiện của lớp, còn khi ta truy cập phương thức static mà không cần phải tạo bất cứ thể hiện của lớp, mà truy cập thông qua tên lớp. Ví dụ:

```

public class CSharp{
    public CSharp() {...}
    public static void StaticMethod() {...}
    public void NonStaticMethod() {...}

    public static void Main()
    {
        CSharp cs = new CSharp();
        cs.NonStaticMethod(); // truy cập qua thể hiện cs (instant)
        CSharp.StaticMethod(); // truy cập thông qua tên lớp
    }
}

```

3.1.6 Nạp chồng (method – overload)

C# cho phép nạp chồng phương thức, nghĩa là cho phép ra nhiều hàm có cùng tên trong cùng một lớp, với điều kiện các hàm này phải khác nhau danh sách tham số.

Nạp chồng toán tử được sử dụng khi muốn cài đặt hai phương thức có cùng tính chất, nhưng có hành vi khác nhau, hay xử lý các loại dữ liệu khác nhau. Như máy giặt các chức năng như giặt thường, ngâm, ngặt đồ nặng, v.v. Ví du:

```
static void ThongBao( double d )
{
    Console.WriteLine("Day la ThongBao(double)");
}
static void ThongBao( int i )
{
    Console.WriteLine("Day la ThongBao(int)");
}
static void ThongBao( int i1, int i2 )
{
    Console.WriteLine("Day la ThongBao(int, int)");
}
static void ThongBao( HocSinh hs )
{
    Console.WriteLine("Day la ThongBao(HocSinh)");
}
```

Hình 3.4 các phương thức có tham số đầu vào khác nhau

Khi chương trình gọi, tùy vào danh sách tham số mà các phương thức phù hợp sẽ được thực thi:

ThongBao(40);	Day la ThongBao(int)
ThongBao(6.8);	Day la ThongBao(double)
ThongBao(new HocSinh());	Day la ThongBao(HocSinh)
ThongBao(9,5);	Day la ThongBao(int, int)

Hình 3.5 Thực thi với các nạp chồng

3.1.7 Thuộc tính (Property)

Thuộc tính trong C# cung cấp khả năng bảo vệ các trường bằng cách đọc hoặc ghi thông qua một phương thức đặc biệt gọi là accessors là getter/setter.

Chúng được khai báo public hoặc protected với kiểu dữ liệu là kiểu của trường được bảo vệ. Thuộc tính trong C# cung cấp khả năng bảo vệ các trường bằng cách đọc hoặc ghi thông qua một phương thức đặc biệt gọi là accessors.

Ngoài đặc tính bảo vệ các trường dữ liệu, setter, còn cho phép kiểm tra tính hợp khi của dữ liệu nếu thuộc tính có ràng buộc.

```

namespace LopDiemKyTu
{
    public class CDiemKiTu
    {
        int x, y;
        char ch;

        public int x
        {
            get { return x; }
            set
            {
                if (KiemTraX(value))
                    x = value;
            }
        }

        #region "Nhóm kiểm tra ràng buộc"
        public bool KiemTraX(int xx)
        {
            return (0 <= xx && xx < Console.WindowWidth);
        }
        public bool KiemTraY(int yy)
        {
            return (0 <= yy && yy < Console.WindowHeight);
        }
        #endregion
    }
}

```

Hình 3.6: ví dụ về properties (các thuộc tính) của lớp

Getter cho phép từ bên ngoài lớp gián tiếp truy cập vào trường dữ liệu, nhằm thay đổi giá trị của trường dữ liệu, trong đó value là giá trị sẽ được gán cho thuộc tính.

Setter cho phép bên ngoài lớp truy cập và giá trị của trường dữ liệu.

Để truy cập getter/setter, ta thông qua instant của lớp. Ví dụ:

```

CDiemKiTu ch =new CDiemKiTu();
ch.X = 10
Console.WriteLine("X = {0}", ch.X)

```

3.2 KẾ THỪA VÀ ĐA HÌNH

3.2.1 Thừa kế

Thừa kế là một khái niệm quan trọng trong lập trình hướng đối tượng. Nó cho phép bạn tạo một hệ thống liên quan đến lớp và tái sử dụng chức năng xác định trong lớp hiện tại.

Ví dụ khi ta cần xây dựng một lóicó cùng những tính chất với lớp cơ sở (đã được xây dựng trước đó) như trường dữ liệu, phương thức, v.v, ngoài ra còn có thêm

những tính chất riêng của nó.Vậy làm thế nào để bạn có thể sử dụng được những tính chất của lớp đang có sẵn, mà không cần phải sao chép mã nguồn từ lớp sẵn có?

Thừa kế trong lập trình hướng đối tượng cho phép bạn có thể tạo ra 1 lớp mới có đầy đủ tất cả những tính chất và phương thức của 1 lớp đã có mà không phải viết lại bất cứ 1 dòng lệnh nào, và ngoài ra cho phép ghi đè các thành viên đã chọn và thêm vào các thành viên mới.

Cú pháp:

```
[access modifier] class <tên_class> : base_class
{
    [access modifier] <kieu_dl> <bien_thanh_vien>;
    ...
    [access modifier] <phuong_thuc>(tham_so, ...)
}
```

3.2.2 Up-cast và down-cast

Đây là 2 khái niệm dùng trong các lớp có thừa kế lẫn nhau.Trong đó:

- Up-cast là chuyển một đối tượng thuộc lớp CON thành một đối tượng lớp CHA
 - Luôn thành công
 - Thực hiện tự động (implicit)
- Down-cast là chuyển ngược một đối tượng thuộc lớp CHA thành một đối tượng thuộc lớp CON.
 - Trường hợp này phải ép kiểu từ CHA xuống CON, lý do 1 cha có thể có nhiều loại con nên phải ép về một kiểu xác định.
 - Phải chỉ định rõ (explicit), và tùy trường hợp mà ép kiểu có thể thành công, hoặc sinh ngoại lệ là chương trình bị lỗi khi runtime
 - Kiểm tra trước khi down-cast: để đảm bảo down-cast thành công, cần kiểm tra xem biến có phải đang giữ đối tượng phù hợp hay không. Sử dụng từ khóa để kiểm tra một biến có là đối tượng thuộc một kiểu cho trước không: **is**

Ví dụ:

using System;

```
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data.SqlClient;
using System.Data;

namespace Test
{
    // Ví dụ:
    public class PERSON
    {
        protected string name;
        public string Name
        {
            get { return name; }
            set { name = value; }
        }
        //...
    }
    public class STUDENT : PERSON
    {
        //...
    }
    public class WORKER : PERSON
    {
        //...
        public static void Main()
        {
            PERSON h1 = new PERSON();
            STUDENT s1 = new STUDENT();
            //Hợp lệ vì lớp cơ sở (CHA) có thể chứa đối tượng của lớp dẫn xuất (CON)
            PERSON h2 = s1;
            //Hợp lệ vì h2 chỉ đến s1 bản chất là STUDENT , nên chỉ cần ép kiểu (downcast)
            STUDENT s2 = (STUDENT)h2;
            // Dùng toán tử is để kiểm tra trước khi ép kiểu
            if (h2 is STUDENT)
            {
                s2 = h2 as STUDENT;
            }
            // Toán tử as không sinh ngoại lệ khi chuyển đổi kiểu
            s1 = h1 as STUDENT;

            // Exception vì ép kiểu thất bại nên s1=null,
            // không thể truy cập đến dữ liệu thành phần
            s1.Name = "ACZ";
            // Cần kiểm tra s1 có khác null không, trước khi tiếp tục sử dụng s1
            if (s1 != null)
            {
                Console.WriteLine("Name: " + s1.Name);
            }
            // Exception: vì đối tượng h1 nguyên thủy là PERSON, không thể ép về STUDENT
            s2 = (STUDENT)h1;

            // Lỗi. Vì không thể chuyển đổi kiểu PERSON cho kiểu đích STUDENT
            s1 = h1;
        }
    }
}
```

3.2.3 Đa hình (Polymorphism)

3.2.3.1 Abstract class

Lớp trừu tượng đơn giản được xem như một class cha cho tất cả các Class có cùng bản chất. Do đó mỗi lớp dẫn xuất (lớp con) chỉ có thể kế thừa từ một lớp trừu tượng. Bên cạnh đó lớp trừu tượng không cho phép tạo instance, nghĩa là sẽ không thể tạo được các đối tượng thuộc lớp đó:

- Sử dụng polymorphism đòi hỏi khai báo các phương thức là virtual hay abstract trong lớp cơ sở trừu tượng phải được
- Override phương thức là virtual hay abstract trong lớp dẫn xuất (lớp con)
- Lớp nào có một phương thức trừu tượng thì phải khai báo lớp là lớp trừu tượng
- So sánh cách cài đặt abstract class bằng abstract method, và virtual method:

Abstract Method	Virtual Method
<ul style="list-style-type: none"> - Từ khoá: virtual abstract - Chỉ có phần khai báo method và kết thúc là dấu ";", Không cần có phần thực thi cho phương thức abstract ở lớp abstract - Bắt buộc lớp dẫn xuất phải override lại - Từ khoá override trước phương thức ở lớp con 	<ul style="list-style-type: none"> - Từ khoá: virtual - Có phần thực thi cho phương thức virtual ở lớp cơ sở - Không bắt buộc lớp dẫn xuất phải override - Từ khoá override trước phương thức ở lớp con

Ví dụ:

```
//** Abstract class
public abstract class Employee
{
    //Khai báo Properties
    public abstract String ID
    {
        get;
        set;
    }
    //Khai báo Method
```

```

Public abstract String Add();

public virtual String Update()
{
    return "Update";
}
}

public class Worker : Employee
{
    //Không cần khai báo lại biến hằng.
    private string id;
    //Triển khai Properties với từ khóa override
    public override String ID
    {
        get { return id; }
        set { id = value; }
    }
    //Triển khai Method với access modifier như ở Abstract class.
    protected override String Add()
    {
        return "Add";
    }
    //Không cần triển khai lại Update() mà chỉ việc sử dụng .
    // Neu can thay doi thi co the overrid lai, khong bat buoc
    //public override String Update()
    //{
    //    return "Update Worker";
    //}
}
classProgram
{
    ///<summary>
    /// The main entry point for the application.
    ///</summary>
    public static void Main()
    {
        Employee ee = newWorker();
        // thuc thi phuong thuc Add cua lop Worker
        ee.Add();
        //Thuc thi phuong thuc Employee.Update neu khong override lai phuong thuc Update
        // Hoac thuc thi phuong thuc Worker.Update() neu co override lai tai lop Worker
        ee.Update();
    }
}

```

3.2.3.2 Đa hình

Đa hình là ý tưởng “sử dụng một giao diện chung cho nhiều phương thức khác nhau”, dựa trên phương thức ảo (virtual method).

Nói cách khác, tính đa hình cho phép một thao tác có các cách xử lý khác nhau trên các đối tượng khác nhau. Để thực hiện được tính đa hình ta thực hiện các bước sau:

- Lớp cơ sở đánh dấu phương thức ảo bằng từ khóa virtual hoặc abstract
- Các lớp dẫn xuất định nghĩa lại phương thức ảo này, đánh dấu bằng từ khóa override

Ví dụ:

```
public class Shape
{
    public virtual void Draw()
    {
        Console.WriteLine("Shape draw!");
    }
}
public class Line : Shape
{
    public override void Draw() —————→ Phù quyết hàm
    {
        Console.WriteLine("Line draw!");
    }
}
public class Circle : Shape
{
    public new void Draw() —————→ Hàm mới cùng tên Draw với
    {
        Console.WriteLine("Circle draw!");
    }
}
```

- Tham chiếu thuộc base class có thể trỏ đến object thuộc derived class và có thể truy cập virtual method đã define lại trong derived class. Nếu tham chiếu này trỏ tới object thuộc base class thì virtual method của lớp cơ sở được thực hiện.
- Nếu tham chiếu này trỏ tới object thuộc derived class thì virtual method đã được derived class định nghĩa lại được thực hiện.

Shape s1 = new Shape();	————→	Shape draw!
s1.Draw();		
Shape s2 = new Line();	————→	Line draw!
s2.Draw();		
Shape s3 = new Circle();	————→	Shape draw!
s3.Draw();		
Circle c = (Circle)s3;	————→	Circle draw!
c.Draw();		

Hình 3.7: ví dụ đa hình trong C#

3.2.3.3 Giao diện – interface

Interface được xem như một mặt nạ cho tất cả các **class** cùng cách thức hoạt động nhưng có thể khác nhau về bản chất. Từ đó lớp dẫn xuất có **implement** từ nhiều lớp **interface** để bổ sung đầy đủ cách thức hoạt động của mình (đa kế thừa - Multiple inheritance).

Ví dụ:

- Abstract class ConVat có các lớp con Chim, Ca.
- Abstract class MayMoc có các lớp con MayBay, Thuyen
- Interface: iBay, iBoi, iChay.

=> MayBay, Chim sẽ có cùng Interface là iBay. Rõ ràng mặc dù MayBay, Chim có cùng cách thức hoạt động là bay nhưng chúng khác nhau về bản chất.

=> MayBay cũng có interface là iChay nhưng Chim không thể nào kế thừa thêm abstract class MayMoc

Abstract Class vs Interface

- Nhìn chung cả 2 đều là “bản thiết kế” cho các lớp dẫn xuất, do đó chúng chỉ chứa các khai báo Properties và Method mà không quan tâm bên trong thực hiện những gì. Nhưng cụ thể thì Abstract Class là “bản thiết kế” cho Class còn Interface là “bản thiết kế” cho Method.
- Do được xem là bản thiết kế cho toàn Class, nên ngoài những khai báo Properties hoặc Method bắt buộc thì Abstarct class vẫn có thể chứa thêm các Method đã được triển khai hoặc các biến hằng. Ngược lại thì Interface hoàn toàn không thể làm được điều đó.
- Toàn bộ những Method và Properties trong Interface không cần khai báo Access modifier vì mặc định là Public. Abstract class thì phải bắt buộc khai báo (Public, Protected) và bắt buộc có từ khóa **abstract** trong các Method và Properties được khai báo trừu tượng.
- Một điểm khác biệt nữa là ở lớp dẫn xuất của Interface class bắt buộc phải thực hiện toàn bộ những Method và Properties đã khai báo, ngược lại lớp dẫn xuất của Abstract class chỉ cần thực hiện những Method và Properties được khai báo trừu tượng (có từ khóa **abstract**)
- Cú pháp: **[access modifier] interface <interface name> [: base interface list]**

- Một interface có thể kế thừa từ nhiều interface khác, Một lớp có thể kế thừa từ nhiều interface, Sự kế thừa interface phải đặt sau sự kế thừa lớp. Các interface thường được đặt tên với tiền tố là “I”, ví dụ: IFile, IComparable, IDisposable, IStorable, ICloneable...

```

/** Ví dụ về Interface
public interface IEmployee
{
    //Khai báo Properties
    string ID
    {
        get; set;
    }
    //Khai báo Method
    string Add();
    string Update();
}
/** Lớp dẫn xuất Interface
public class Employee1 : IEmployee
{
    // Biến hằng được định nghĩa ở đây.
    protected String id;
    protected String name;
    // Triển khai Properties
    String ID
    {
        get { return id; }
        set { id = value; }
    }
    //Triển khai tất cả các Method có ở Interface
    public String Add()
    {
        return "Add";
    }
    public String Update()
    {
        return "Update";
    }
    // Phần triển khai của Employee1
    ...
}

```

3.2.3.4 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.

```
// Ví dụ về sealed class
using System;
sealed class MyClass
{
    public int x;
    public int y;
}
class MainClass
{
    public static void Main()
    {
        MyClass mC = new MyClass();
        mC.x = 110; mC.y = 150;
        Console.WriteLine("x = {0}, y = {1}", mC.x, mC.y);
    }
}
```

Sealed Method: được sử dụng trước phương thức để ngăn ko cho lớp dẫn xuất override

```
// Ví dụ về sealed method
using System;
class MyClass1 {
    public int x; public int y;
    public virtual void Method() {
        Console.WriteLine("virtual method");
    }
}
class MyClass : MyClass1 {
    Public override sealed void Method() {
        Console.WriteLine("sealed method");
    }
}
class MainClass {
    public static void Main() {
        MyClass1 mC = new MyClass();
        mC.x = 110; mC.y = 150;
        Console.WriteLine("x = {0}, y = {1}", mC.x, mC.y);
        mC.Method();
    }
}
```

TÓM TẮT

Trong bài này, học viên làm quen với nền tảng của ngôn ngữ C# bao gồm:

- Các kiểu dữ liệu
- Khai báo và sử dụng hằng biến
- Lập trình với các cấu trúc điều khiển, vòng lặp trong C#.

- Định nghĩa lớp
- Các thành phần của lớp: field dữ liệu, method, property
- Năm được các nguyên tắc, khái niệm của lập trình hướng đối tượng trong C#
- Định nghĩa và sử dụng Lớp trừu tượng (abstract class)
- Định nghĩa và các sử dụng interface (giao tiếp)
- Tính đa hình trong C#

CÂU HỎI ÔN TẬP

Câu 1: Sự khác nhau giữa kiểu dữ liệu tham chiếu và kiểu dữ liệu giá trị?

Câu 2: Làm thế nào để tạo đối tượng là thể hiện của một lớp?

Câu 3: Quá trình boxing và unboxing có diễn ra với một đối tượng là kiểu cấu trúc hay không?

Câu 4: Toán tử as , is được dùng làm gì?

Câu 5: So sánh giữa lớp và giao diện?

Câu 6: Sự khác nhau giữa giao diện và lớp trừu tượng?

Câu 7: Các lớp thực thi giao diện sẽ phải làm gì?

Câu 8: Có bao nhiêu cách gọi một phương thức được khai báo trong giao diện?

Câu 9: Các thành viên của giao diện có thể có những thuộc tính truy cập nào?

Câu 10: Câu hỏi 6: Chúng ta có thể tạo thể hiện của giao diện một cách trực tiếp được không?

Câu 11: Giao diện là kiểu dữ liệu tham chiếu hay kiểu giá trị?

Câu 12: Số giao diện có thể được kế thừa cho một lớp?

BÀI 4: DELEGATE & EVENT

Sau khi học xong bài này, học viên có thể nắm được:

- *Delegate là gì, cách sử dụng*
- *Làm thế nào để tạo thể hiện delegate, và cách thức triệu gọi delegate*
- *Thực thi delegate*
- *Multicast delegate*
- *Event*
- *Khái niệm event*
- *Event & delegate*
- *Cơ chế publishing & subscribing*
- *Minh họa cơ chế event*

4.1 ỦY THÁC - DELEGATE

Trong lập trình chúng ta thường đối diện với tình huống là khi chúng ta muốn thực hiện một hành động nào đó, nhưng hiện tại thì chưa xác định được chính xác phương thức hay sự kiện trong đối tượng. Ví dụ như một nút lệnh button biết rằng nó phải thông báo cho vài đối tượng khi nó được nhấn, nhưng nó không biết đối tượng hay nhiều đối tượng nào cần được thông báo. Tốt hơn việc nối nút lệnh với đối tượng cụ thể, chúng ta có thể kết nối nút lệnh đến một cơ chế ủy quyền và sau đó thì chúng ta thực hiện việc ủy quyền đến phương thức cụ thể khi thực thi chương trình.

Trong thời kỳ đầu của máy tính, chương trình được thực hiện theo trình tự xử lý từng bước tuần tự cho đến khi hoàn thành, và nếu người dùng thực hiện một sự tương tác thì sẽ làm hạn chế sự điều khiển hoạt động khác của chương trình cho đến khi sự tương tác với người dùng chấm dứt.

Tuy nhiên, ngày nay với mô hình lập trình giao diện người dùng đồ họa (GUI: Graphical User Interface) đòi hỏi một cách tiếp cận khác, và được biết như là lập trình điều khiển sự kiện (event-driven programming). Chương trình hiện đại này đưa ra một giao diện tương tác với người dùng và sau đó thì chờ cho người sử dụng kích hoạt một hành động nào đó. Người sử dụng có thể thực hiện nhiều hành động khác nhau như: chọn các mục chọn trong menu, nhấn một nút lệnh, cập nhật các ô chứa văn bản,... Mỗi hành động như vậy sẽ dẫn đến một sự kiện (event) được sinh ra. Một số các sự kiện khác cũng có thể được xuất hiện mà không cần hành động trực tiếp của người dùng. Các sự kiện này xuất hiện do các thiết bị như đồng hồ của máy tính phát ra theo chu kỳ thời gian, thư điện tử được nhận, hay đơn giản là báo một hành động sao chép tập tin hoàn thành,...

Một sự kiện được đóng gói như một ý tưởng “chuyện g. đó xảy ra” và chương trình phải đáp ứng lại với sự kiện đó. Cơ chế sự kiện và ủy quyền gắn liền với nhau, bởi vì khi một sự kiện xuất hiện thì cần phải phân phát sự kiện đến trình xử lý sự kiện tương ứng. Thông thường một trình xử lý sự kiện được thực thi trong C# như là một sự ủy quyền.

Ủy quyền cho phép một lớp có thể yêu cầu một lớp khác làm một công việc nào đó, và khi thực hiện công việc đó thì phải báo cho lớp biết. Ủy quyền cũng có thể được sử dụng để xác nhận những phương thức chỉ được biết lúc thực thi chương trình, và chúng ta sẽ tìm hiểu kỹ vấn đề này trong phần chính của chương.

Ủy quyền (delegate): Trong ngôn ngữ C#, ủy quyền là lớp đối tượng đầu tiên (first-class object), được hỗ trợ đầy đủ bởi ngôn ngữ lập trình. Theo kỹ thuật thì ủy quyền là kiểu dữ liệu tham chiếu được dùng để đóng gói một phương thức với tham số và kiểu trả về xác định. Chúng ta có thể đóng gói bất cứ phương thức thích hợp nào vào trong một đối tượng ủy quyền. Trong ngôn ngữ C++ và những ngôn ngữ khác, chúng ta có thể làm được điều này bằng cách sử dụng con trỏ hàm (function pointer) và con trỏ đến hàm thành viên. Không giống như con trỏ hàm như trong C/C++, ủy quyền là hướng đối tượng, kiểu dữ liệu an toàn (type-safe) và bảo mật.

Một điều thú vị và hữu dụng của ủy quyền là nó không cần biết và cũng không quan tâm đến những lớp đối tượng mà nó tham chiếu tới. Điều cần quan tâm đến những đối tượng đó là các đối mục của phương thức và kiểu trả về phải phù hợp với đối tượng ủy quyền khai báo.

4.1.1 Khai báo

- Delegate (ủy thác , ủy quyền) : giúp giải quyết vấn đề muốn thực thi một phương thức nào đó của một đối tượng nào đó nhưng người lập trình có thể chưa rõ lúc thiết kế.
- Một delegate chứa tham chiếu tới hàm
- Một delegate định nghĩa một signature
 - Return type
 - Sequence of parameter types
- Cách khai báo: để tạo một ủy quyền ta dùng từ khóa delegate theo sau là kiểu trả về tên phương thức được ủy quyền và các đối mục cần thiết:

```
public delegate void MyDelegate1(int x, int y)
```

Delegate cho dạng hàm:
void Method(int, int)

```
public delegate string MyDelegate2(float f)
```

Delegate cho dạng hàm:
string Method(float)

- Khai báo trên định nghĩa một ủy quyền tên là MyDelegate1, nó sẽ đóng gói bất cứ phương thức nào lấy hai tham số kiểu int và không trả về giá trị (void).

4.1.2 Instance delegate

- Một khi mà ủy quyền được định nghĩa, chúng ta có thể đóng gói một phương thức thành viên bằng việc tạo một thể hiện của ủy quyền này, truyền vào trong một phương thức có khai báo kiểu trả về và các đối mục cần thiết.
- Sử dụng ủy quyền để xác nhận phương thức lúc thực thi. Ủy quyền như chúng ta đa biết là được dùng để xác định những loại phương thức có thể được dùng để xử lý các sự kiện và để thực hiện callback trong chương trình ứng dụng. Chúng cũng có thể được sử dụng để xác định các phương thức tĩnh và các instance của phương thức mà chúng ta không biết trước cho đến khi chương trình thực hiện.

- Khởi tạo delegate từ một phương thức có sẵn, bạn cần truyền phương thức mà delegate sẽ đại diện vào trong constructor. Bạn có thể khởi tạo delegate theo cách các cách sau:

```
public class TestDelegate1
{
    public delegate int DoSomething(int x, int y);

    public int Add(int x, int y)
    {
        return x + y;
    }

    public static void Main()
    {
        // cách 1
        DoSomething myDelegate = new DoSomething(Add);
        // cách 2
        DoSomething myDelegate1 = Add;

        // anonymous method
        DoSomething obj = delegate(int x, int y)
        {
            return x + y;
        };

        // lambda expression
        DoSomething obj1 = (x, y) => x + y;
    }
}
```

- Hoặc có thể sử dụng anonymous method hoặc lambda expression để tạo đối tượng như ví dụ trên.

4.1.3 Multicast delegate

- C# cho phép chúng ta “gắn” nhiều phương thức vào cùng một delegate, miễn là các phương thức này có cùng kiểu trả về và tham số với delegate. Và khi ta gọi delegate này, tất cả các phương thức đã được “gắn” sẽ thi hành cùng lúc. Kỹ thuật này gọi là “Multicast”.
- Chú ý : một Multicast delegate chỉ chấp nhận phương thức có kiểu trả về là void.
- Để thực hiện Multicast, ta tạo các thể hiện của một multicast delegate, gắn nó với các phương thức tương ứng. Sau đó dùng toán tử “+” để gom các delegate này vào thành 1 delegate duy nhất.
- Delegate instance có một danh sách các tham chiếu method:

- Cho phép add (+=) các method
- Có thể remove (-=) các method

4.1.4 Call Delegate

- Khi 1 delegate đc tạo để bao nhiêu method thì khi đc gọi, lần lượt các method đã được bao sẽ được gọi.
- Delegate thường dùng để gọi lại các method từ 1 method bên khác.

Ví dụ:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows.Forms;

namespace Test
{
    // Ví dụ về delegate
    public class TestDelegate2
    {
        public delegate void MyDelegate1(int x, int y);
        public delegate string MyDelegate2(float f);

        public void Method1(int x, int y)
        {
            //..
        }
        public string Method2(float f)
        {
            string kq = "";
            // ...
            return kq;
        }
        void Print(int x, int y)
        {
            Console.WriteLine("x = {0}, y = {1}", x, y);
        }
        void Sum(int x, int y)
        {
            Console.WriteLine("Tong = {0}", x + y);
        }
        public static void Main()
        {
            MyDelegate1 del1 = new MyDelegate1(Method1);
            int x = 5, y = 10;
            del1(x, y);
            del1(100, y);
            MyDelegate2 del2 = new MyDelegate2(Method2);
            string s = del2(100f);
            MyDelegate1 mulDel = new MyDelegate1(Print);
            mulDel += new MyDelegate1(Sum);
            mulDel(5, 10); //Thực thi lần lượt phương thức Print và Sum
        }
    }
}
```

```

        mulDel -= new MyDelegate1(Print);
        mulDel(5, 10); //Thuc thi phuong thuc Sum
    }
}

```

4.1.5 Demo – Bubble sort

- Sau đây ta sẽ xem 1 ví dụ cho thấy sự hữu ích của delegate. Ta sẽ tạo lớp Sorter. Lớp này thực thi 1 phương thức tĩnh **Sort()** lấy thông số đầu là 1 mảng đối tượng, và sắp xếp lại chúng tăng dần. Ví dụ để sắp xếp 1 mảng số nguyên bằng thuật toán Bubble sort :
- Thuật toán này tốt cho số nguyên, nhưng ta muốn phương thức **Sort()** sắp xếp cho mọi đối tượng, ta thấy vẫn đề nằm ở dòng `if(sortArray[j] < sortArray[i])` trong đoạn mã trên. bởi ta muốn so sánh 2 đối tượng trên mảng mà cái nào là lớn hơn. Chúng ta có thể sắp xếp kiểu int, nhưng làm thế nào để sắp xếp những lớp chưa biết hoặc không xác định cho đến lúc runtime. Câu trả lời là muốn sắp xếp phải truyền 1 delegate gói trong một phương thức sẽ làm công việc so sánh.
- Định nghĩa delegate như sau:
- delegate bool CompareOp(object lhs, object rhs);
- Và xây dựng phương thức sort() là :
- static public void Sort(object [] sortArray, CompareOp **gtMethod**)
- Phần hướng dẫn cho phương thức này sẽ nói rõ rằng **gtMethod** phải tham chiếu đến 1 phương thức static có 2 đối số, và trả về true nếu giá trị của đối số thứ 2 là 'lớn hơn' (nghĩa là nằm sau trong mảng) đối số thứ nhất. Mặc dù ta có thể sử dụng delegate ở đây, nhưng cũng có thể giải quyết vấn đề bằng cách sử dụng interface. .NET xây dựng 1 interface IComparer cho mục đích này. Tuy nhiên, ta sử dụng delegate vì loại vấn đề này thì thường có khuynh hướng dùng delegate.
- Để dùng lớp này ta cần định nghĩa 1 số lớp khác mà có thể dùng thiết lập mảng cần sắp xếp. Ví dụ công ty điện thoại có danh sách tên khách hàng, và muốn sắp danh sách theo lương. mỗi nhân viên trình bày bởi thể hiện của một lớp , Employee:
- Lưu ý để phù hợp với dấu ấn của delegate **CompareOp**, chúng ta phải định nghĩa **RhsIsGreater** trong lớp này lấy 2 đối tượng để tham khảo, hơn là tham khảo

employee như là thông số. Điều này có nghĩa là ta phải ép kiểu những thông số vào trong tham khảo employee để thực thi việc so sánh.

Ví dụ:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows.Forms;

namespace Test
{
    //Ví dụ:
    class BubbleSorter
    {
        static public void Sort(object[] sortArray, CompareOp gtMethod)
        {
            for (int i = 0; i < sortArray.Length; i++)
            {
                for (int j = i + 1; j < sortArray.Length; j++)
                {
                    if (gtMethod(sortArray[j], sortArray[i]))
                    {
                        object temp = sortArray[i];
                        sortArray[i] = sortArray[j];
                        sortArray[j] = temp;
                    }
                }
            }
        }
    }
    class Employee
    {
        private string name;
        private decimal salary;
        public Employee(string name, decimal salary)
        {
            this.name = name;
            this.salary = salary;
        }
        public override string ToString()
        {
            return string.Format(name + ", {0:C}", salary);
        }
        public static bool RhsIsGreater(object lhs, object rhs)
        {
            Employee empLhs = (Employee)lhs;
            Employee empRhs = (Employee)rhs;
            return (empRhs.salary > empLhs.salary) ? true : false;
        }
    }

    delegate bool CompareOp(object lhs, object rhs);
    class MainEntryPoint
    {
        public static void Main()
        {
            Employee[] employees = {
                new Employee("Karli Watson", 20000),
```

```

        new Employee("Bill Gates", 10000),
        new Employee("Simon Robinson", 25000),
        new Employee("Mortimer", (decimal)1000000.38),
        new Employee("Arabel Jones", 23000),
        new Employee("Avon from 'Blake's 7'", 50000)
    };

    CompareOp employeeCompareOp = new CompareOp(Employee.RhsIsGreater);
    BubbleSorter.Sort(employees, employeeCompareOp);
    for (int i = 0; i < employees.Length; i++)
    {
        Console.WriteLine(employees[i].ToString());
    }
}
}
}

```

- Chạy mã này sẽ thấy employees được sắp xếp theo lương như sau:
 - Bill Gates, £10,000.00
 - Karli Watson, £20,000.00
 - Arabel Jones, £23,000.00
 - Simon Robinson, £25,000.00
 - Avon from 'Blake's 7', £50,000.00
 - Mortimer, £1,000,000.38

4.2 SỰ KIỆN - EVENT

4.2.1 Lập trình sự kiện

- Mô hình sự kiện trong C# là một mô hình lập trình sự kiện phổ biến trong lập trình không đồng bộ. Điều căn bản trong mô hình này là khái niệm về "người xuất bản và người đăng ký" (publisher and subscribers.). Publisher sẽ làm 1 số việc và xuất bản ra 1 sự kiện, sau đó sẽ gửi chúng tới subscriber, subscriber sẽ mô tả và nhận sự kiện rõ ràng.
- Cơ chế thông điệp giữa các lớp hay các đối tượng:
 - Có thể thông báo cho lớp khác biết được khi một lớp có sự kiện phát sinh
 - Publisher: lớp phát sinh sự kiện
 - Subscriber: lớp nhận hay xử lý khi sự kiện xảy ra

- Trong C#, bất kỳ đối tượng nào mô tả sự kiện thì bất kỳ ứng dụng nào khác cũng có thể nhận biết mô tả này. Khi lớp publisher đưa lên 1 sự kiện thì tất cả các ứng dụng mô tả nó đều nhận ra.

Ví dụ:

- Trong môi trường giao diện GUIs (Graphical User Interfaces: GUIs): Button đưa ra sự kiện "Click", cho phép lớp khác có thể đáp ứng (xử lý) khi sự kiện này xảy ra.
- Button "Add" trong Form, khi sự kiện click xảy ra thì Form thực hiện lấy dữ liệu từ các TextBox đưa vào ListBox...

4.2.2 Event

- Event : Ứng với mỗi thao tác người dùng sẽ có một sự kiện phát sinh , và chương trình sẽ phải đáp trả cho mỗi sự kiện này
- Khái niệm Publishing và Subscribing:
 - Publishing: Một lớp phát sinh sự kiện.
 - Subscribing: Các lớp được subscribe sẽ nhận thông báo về sự kiện phát sinh, và thực thi.
- Event trong C# sẽ được cài đặt bằng delegate. Lớp publish định nghĩa một delegate. Khi một sự kiện phát sinh phương thức của lớp subscribe sẽ được gọi thông qua delegate
- Sự kiện trong C# được thực thi nhờ ủy thác:
 - Lớp publishing định nghĩa ủy thác
 - Những lớp subscribing phải thực thi
 - Khi sự kiện xuất hiện thì phương thức của lớp subscribing được gọi thông qua ủy thác.
- Phương thức để xử lý sự kiện gọi là trình xử lý sự kiện (event handler). Quy ước: Sau đây là những quy ước quan trọng khi sử dụng sự kiện:
 - Event Handlers trong .NET không có giá trị trả về và có 2 tham số.
 - 1 là nguồn phát sinh sự kiện, tức là đối tượng publisher.

- 2 là đối tượng thừa kế từ EventArgs.
- Event Handlers Những sự kiện là thuộc tính của lớp phát sinh sự kiện.(publisher)
- Sử dụng từ khóa event để điều khiển cách mà các thuộc tính event được truy cập bởi các lớp mô tả (subscriber)

4.2.2.1 Khai báo

- Khai báo delegate xử lý sự kiện
 - public **delegate** void HandlerName(object obj, EventArgs arg);
- Khai báo event
 - public **event** HandlerName OnEventName;
- Các lớp muốn xử lý khi sự kiện OnEventName phát sinh thì phải thực thi event handler

4.2.2.2 Minh họa

- Xây dựng 1 lớp thực hiện yêu cầu: “cứ mỗi giây sẽ phát sinh 1 sự kiện”
- Cho phép 2 lớp khác đăng ký xử lý sự kiện này, mỗi lớp có cách xử lý riêng:
 - Lớp A: hiển thị thời gian theo “mô phỏng đồng hồ analog”
 - Lớp B: hiển thị thời gian theo “mô phỏng đồng hồ digital”
- Tạo một lớp Clock:
 - Khai báo một event: OnSecondChange
 - Một phương thức Run: cứ 1s thì phát sinh sự kiện OnSecondChange
- Tạo 2 lớp: AnalogClock và DigitalClock nhận xử lý sự kiện OnSecondChange của lớp Clock
- Khai báo delegate xử lý event:



- Sau đây là source code mô tả chương trình cung cấp cho bạn cách tạo 1 đối tượng với 1 số sự kiện để thông báo đến subscriber bất cứ khi nào thời gian thay đổi mỗi giây:

```

using System;
using System.Threading;

namespace SecondChangeEvent
{
    /* ===== event Publisher ===== */
    // Our subject - it is this class that other classes will observe.
    // This class publishes one event : SecondChange. The observers subscribe to that event.
    public class Clock
    {
        // private Fields holding the hour, minute and second
        private int _hour;
        private int _minute;
        private int _second;

        // The delegate named SecondChangeHandler, which will encapsulate any method
        // that takes a clock object and a TimeInfoEventArgs object as the parameter and returns
        no value.
        // It's the delegate the subscribers must implement.
        public delegate void SecondChangeHandler(object clock, TimeInfoEventArgs
timeInformation);

        // The event we publish
        public event SecondChangeHandler SecondChange;

        // The method which fires the event
        protected void OnSecondChange(object clock, TimeInfoEventArgs timeInformation)
        {
            // Check if there are any Subscribers
            if (SecondChange != null)
            {
                // Call the event
                SecondChange(clock, timeInformation);
            }
        }

        // Set the clock running, it will raise an event for each new second
        public void Run()
        {
            for ( ; )
            {
                // Sleep 1 Second
                Thread.Sleep(1000);
            }
        }
    }
}

```

```
// Get the current time
System.DateTime dt = System.DateTime.Now;

// If the second has changed
// notify the subscribers
if (dt.Second != _second)
{
    // Create the TimeInfoEventArgs object to pass to the subscribers
    TimeInfoEventArgs timeInformation = new
TimeInfoEventArgs(dt.Hour,dt.Minute,dt.Second);

    // If anyone has subscribed, notify them
    OnSecondChange(this, timeInformation);
}

// update the state
_second = dt.Second;
_minute = dt.Minute;
_hour = dt.Hour;

}

}

// The class to hold the information about the event in this case it will hold only
information
// available in the clock class , but could hold additional state information
public class TimeInfoEventArgs : EventArgs
{
    public TimeInfoEventArgs(int hour, int minute, int second)
    {
        this.hour = hour;
        this.minute = minute;
        this.second = second;
    }

    public readonly int hour;
    public readonly int minute;
    public readonly int second;
}

/* ====== event Subscribers ===== */
// An observer. DisplayClock subscribes to the clock's event s.
// The job of DisplayClock is to display the current time
public class DisplayClock
{
    // Given a clock, subscribe to its SecondChangeHandler event
    public void Subscribe(Clock theClock)
    {
        theClock.SecondChange += new Clock.SecondChangeHandler(TimeHasChanged);
    }

    // The method that implements the delegate d functionality
    public void TimeHasChanged(object theClock, TimeInfoEventArgs ti)
    {
        Console.WriteLine("Current Time: {0}:{1}:{2}",
            ti.hour.ToString(),
            ti.minute.ToString(),
            ti.second.ToString());
    }
}
```

```

}

// A second subscriber whose job is to write to a file
public class LogClock
{
    public void Subscribe(Clock theClock)
    {
        theClock.SecondChange += new Clock.SecondChangeHandler(WriteLogEntry);
    }

    // This method should write to a file
    // we write to the console to see the effect
    // this object keeps no state
    public void WriteLogEntry(object theClock, TimeInfoEventArgs ti)
    {
        Console.WriteLine("Logging to file: {0}:{1}:{2}",
            ti.hour.ToString(),
            ti.minute.ToString(),
            ti.second.ToString());
    }
}
/* ===== Test Application ===== */
// Test Application which implements the Clock Notifier - Subscriber Sample
public class Test
{
    public static void Main()
    {
        // Create a new clock
        Clock theClock = new Clock();

        // Create the display and tell it to
        // subscribe to the clock just created
        DisplayClock dc = new DisplayClock();
        dc.Subscribe(theClock);

        // Create a Log object and tell it
        // to subscribe to the clock
        LogClock lc = new LogClock();
        lc.Subscribe(theClock);

        // Get the clock started
        theClock.Run();
    }
}
}

```

Lớp Clock từ ví dụ trên đơn giản là in ra thời gian mỗi khi phát sự kiện (giây thay đổi), vì sao phải sử dụng delegate? Điều tiện lợi của publisher/ subscriber là bất kỳ bao nhiêu lớp đều có thể được thông báo khi có 1 sự kiện được phát ra. Lớp subscriber ko cần biết lớp Clock làm việc như thế nào , và lớp Clock ko cần biết đối tượng nào sẽ tiếp nhận khi có sự kiện. Ví dụ đơn giản là 1 button có thể xuất bản ra 1 sự kiện OnClick , và bất kỳ đối tượng nào cũng có thể mô tả tới sự kiện này, nhận thông báo khi button được click.

Publisher và subscriber được tách riêng ra bởi delegate. Đặc tính này đã làm cho code được linh hoạt và rõ ràng hơn. Lớp Clock có thể thay đổi cách mà nó kiểm tra thời gian mà không ảnh hưởng gì đến lớp subscriber. Và ngược lại, lớp subscriber có thể thay đổi cách mà nó phản hồi lại sự kiện thời gian hay đổi mà không ảnh hưởng đến Clock. 2 lớp này độc lập với nhau và làm cho code dễ bảo trì.

TÓM TẮT

Trong bài này, học viên làm quen với mô hình lập trình sự kiện trong C# trên hệ điều hành Windows.

- *Delegate (ủy thác , ủy quyền) : giúp giải quyết vấn đề muốn thực thi một phương thức nào đó của một đối tượng nào đó nhưng người lập trình có thể chưa rõ lúc thiết kế.*
- *Một delegate chứa tham chiếu tới hàm*
- *Một delegate định nghĩa một signature, trong đó:*
 - *Return type*
 - *Sequence of parameter types*

Event : Ứng với mỗi thao tác người dùng sẽ có một sự kiện phát sinh , và chương trình sẽ phải đáp trả cho mỗi sự kiện này.

Khái niệm Publishing và Subscribing

- *Publishing : Một lớp phát sinh sự kiện*
- *Subscribing : Các lớp được subscribe sẽ nhận thông báo về sự kiện phát sinh , và thực thi*

Event trong C# sẽ được cài đặt bằng delegate. Lớp publish định nghĩa một delegate. Khi một sự kiện phát sinh phương thức của lớp subscribe sẽ được gọi thông qua delegate

CÂU HỎI ÔN TẬP

Câu 1: Tóm tắt những nét cơ bản về ủy quyền?

Câu 2: Con trỏ hàm là gì?

Câu 3: Công dụng của việc khai báo ủy quyền tĩnh? Khi nào thì nên khai báo ủy quyền tĩnh khi nào thì không nên?

Câu 4: Một ủy quyền có thể gọi được nhiều hơn một phương thức hay không? Chức năng nào trong C# hỗ trợ ủy quyền này?

Câu 5: Có phải tất cả các ủy quyền đều là ủy quyền Multicast hay không? Điều kiện để trở thành ủy quyền Multicast?

Câu 6: Các toán tử nào có thể dùng để thực hiện việc Multicast các ủy quyền?

Câu 7: Sự kiện là gì? Trong hệ thống ứng dụng nào thì sự kiện được sử dụng nhiều?

Câu 8: Những sự kiện trong C# được thực hiện thông qua cái gì?

Câu 9: Hãy tóm lược quá trình tạo một sự kiện và giải quyết sự kiện thông qua cơ chế ủy quyền trong C#?

BÀI 5: LINQ

Sau khi học xong bài này, học viên có thể nắm được:

- *Cách sử dụng thư viện LINQ*
- *Các thành phần trong LINQ*
- *Hai cú pháp sử dụng LINQ*
- *Biểu thức Lambda*
- *Sử dụng danh sách truy vấn LINQ*
- *LINQ to Objects*

5.1 Language integrated Query (LINQ)

Language-Integrated Query (LINQ) là một sự đổi mới được giới thiệu trong Visual Studio 2008 và .NET Framework từ phiên bản 3.5.

LINQ giúp nhúng truy vấn vào ngôn ngữ lập trình (đúng như tên gọi của nó). Tức là có thể sử dụng C# hay Visual Basic để có thể truy vấn đến mọi nguồn dữ liệu mà không nhất thiết phải biết các loại ngôn ngữ truy vấn riêng biệt để có thể thao tác với chúng.

Để sử dụng LINQ cần có ba thành phần: nguồn dữ liệu (data source), truy vấn (query), lời gọi thực hiện truy vấn (query execution). Khi kiểu dữ liệu trả về chưa được định nghĩa (kiểu vô danh), vẫn có thể sử dụng từ khóa **var** để thay thế.

Ví dụ: Thực hiện lấy các số lẻ trong mảng numbers

```
//1. Nguồn dữ liệu.
int[] numbers = new int[] { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

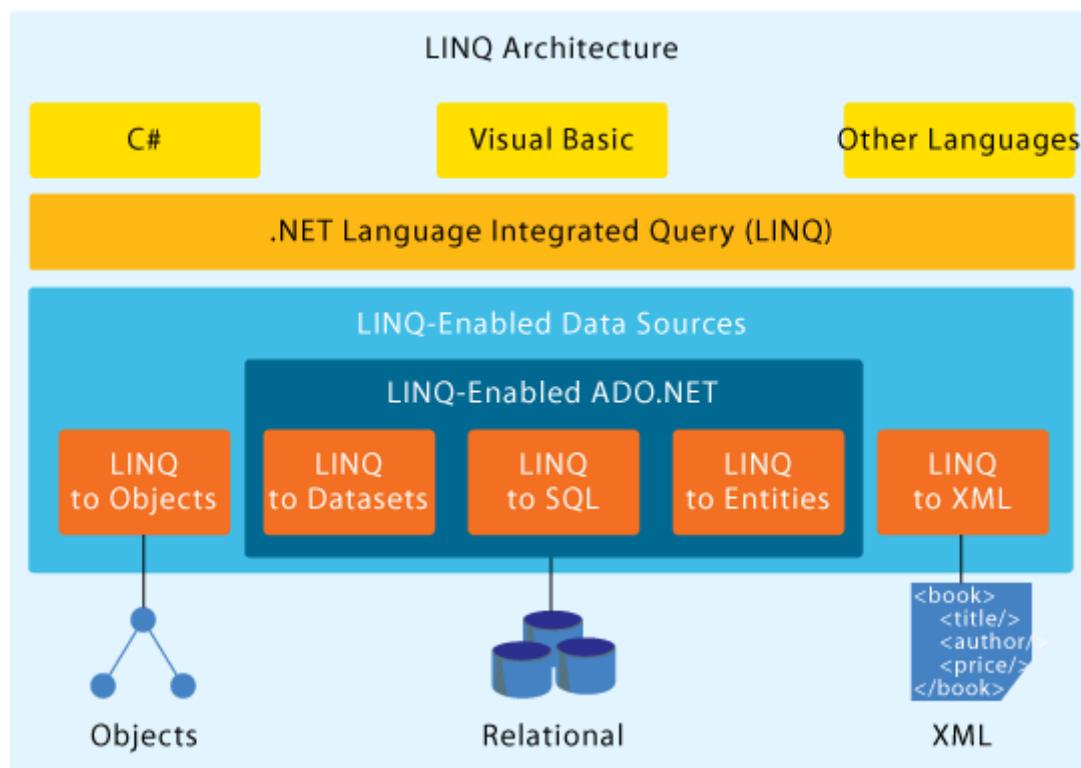
// 2. Tạo câu lệnh truy vấn:
// Lấy ra các số lẻ trong mảng numbers
var numQuery = from num in numbers
               where (num % 2) == 1
               select num;

// 3. Thực thi truy vấn.
foreach (int num in numQuery)
{
    Console.WriteLine("{0} ", num);
}
```

5.1.1 Nguồn dữ liệu (data source)

Các truy vấn LINQ trong C# đều tác động trên nguồn dữ liệu. Để người lập trình có thể sử dụng chung một cách thức viết code cho dù truy vấn từ nhiều loại nguồn dữ liệu khác nhau, LINQ luôn luôn làm việc với object. Do đó, đối với mỗi loại nguồn dữ liệu trong C# cần xây dựng thư viện hỗ trợ LINQ riêng giúp chuyển đổi dữ liệu về dạng object và ngược lại.

Hình minh họa dưới đây mô tả vai trò của LINQ trong quan hệ với ngôn ngữ lập trình C# và các nguồn dữ liệu



LINQ thực chất là một bộ thư viện phương thức mở rộng (extension method) cho các class thực thi hai giao diện (interface) `IEnumerable` và `IQueryable`

- Lớp `Enumerable` chứa các phương thức mở rộng dành cho các class thực thi giao diện `IEnumerable<T>`, bao gồm các kiểu dữ liệu quen thuộc nằm trong không gian tên `System.Collections.Generic` như `List<T>`, `Dictionary<Tkey, TValue>`, `HashSet<T>`...
- Lớp `Queryable` chứa các phương thức mở rộng dành cho các class thực thi giao diện `IQueryable<T>`, nằm trong không gian tên `System.Linq`.

Ví dụ: Sử dụng *IEnumerable* và *IQueryable*

```
// sử dụng trong System.Collections.Generic với các list, array...
List<string> stringList = new List<string>() { "C#", "Python" , "Java" };
IEnumerable<string> listStartC = stringList.Where(p => p.StartsWith("C")).ToList();

//sử dụng trong Linq to SQL, LINQ to Entity...
var context = new StudentModel();

IQueryable<Student> studentList = context.Students.Where(p => p.AverageScore > 5);
```

Vì lý do này, tất cả các class thực thi các giao diện trên đều có thể trở thành nguồn dữ liệu. Như vậy, với dữ liệu object “Linq to object”, với dữ liệu Xml phải xây dựng thêm provider “LINQ to Xml”, với cơ sở dữ liệu SQL Server phải xây dựng thêm provider “LINQ to SQL” , “LINQ to Entities”...

5.1.2 Truy vấn LINQ

Cú pháp truy vấn của LINQ được tích hợp trong ngôn ngữ C# (VB.NET) giúp loại bỏ sự khác biệt giữa ngôn ngữ lập trình và các ngôn ngữ dùng để truy vấn dữ liệu, cũng như tạo ra một giao diện lập trình thống nhất (sử dụng cùng một nhóm lệnh truy vấn) dùng cho nhiều loại nguồn dữ liệu khác nhau (Object, SQL server, XML, Entities...)

Có hai cách viết cho LINQ là cú pháp truy vấn (query syntax) và cú pháp phương thức (method syntax) được mô tả chi tiết trong hướng dẫn ở phần sau.

5.1.3 Thực thi truy vấn

Có hai kịch bản thực thi truy vấn trong LINQ:

- Thực thi trì hoãn (deferred execution): Truy vấn không thực thi cho đến khi chúng ta dùng vòng lặp duyệt qua danh sách các phần tử trả về của truy vấn, do đó, kết quả truy vấn có thể thay đổi trước khi vòng lặp thực thi. Đây là đặc trưng rất mạnh trong LINQ và là cách thực thi mặc định.

Ví dụ: *Lấy các số lẻ trong mảng numbers, trước khi thực thi thay đổi phần tử đầu tiên*

```
//1. Nguồn dữ liệu.
int[] numbers = new int[] { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
// 2. Tạo câu lệnh truy vấn: Lấy ra các số lẻ trong mảng numbers
var numQuery = from num in numbers
               where (num % 2) == 1
               select num;
```

```
// trước khi thực thi kết quả thử thay đổi phần từ đầu tiên trong mảng
numbers[0] = 0;
// 3. Thực thi truy vấn.
foreach (int num in numQuery)
{
    Console.Write("{0} ", num);
}
//Kết quả truy vấn: 3 5 7 9
```

- Thực thi ngay lập tức (Forcing Immediate Execution): Thực hiện truy vấn ngay tại quá trình tạo truy vấn, do đó, kết quả không thay đổi sau khi lệnh tạo truy vấn hoàn tất. Có thể dùng kết hợp các phương thức mở rộng như ToArray hayToList.

Ví dụ: Thực hiện lấy các số lẻ trong mảng numbers

```
//1. Nguồn dữ liệu.
int[] numbers = new int[] { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
// 2. Tạo câu lệnh truy vấn: Lấy ra các số lẻ trong mảng numbers
var numQuery =
    (from num in numbers
     where (num % 2) == 1
     select num).ToArray();
// trước khi thực thi kết quả thử thay đổi phần từ đầu tiên trong mảng
numbers[0] = 0;
// 3. Thực thi truy vấn.
foreach (int num in numQuery)
{
    Console.Write("{0} ", num);
}
//Kết quả truy vấn: 1 3 5 7 9
```

5.2 Sử dụng truy vấn LINQ

5.2.1 Cú pháp truy vấn và cú pháp phương thức

Có hai cách viết cho LINQ là cú pháp truy vấn (query syntax) và cú pháp phương thức (method syntax).

- Cú pháp truy vấn (Query syntax): Cú pháp này nhìn giống như một truy vấn select SQL đảo ngược với từ khóa đầu tiên là *from*, kết thúc là *select*. Cú pháp này có hình thức khác biệt với code C# thông thường với một số từ khóa mới.

```
// Nguồn dữ liệu
List<string> stringList = new List<string>() {
    "C#",
    "Python",
    "Java"
};
// LINQ Query Syntax
```

```
var result = from s in stringList
             where s.Contains("Tutorials")
             select s;
```

- Cú pháp phương thức (Method syntax): Giống như cách gọi một phương thức mở rộng bình thường trên object của class. Đây là lối viết cơ bản của LINQ.

```
// Nguồn dữ liệu
List<string> stringList = new List<string>() { "C#", "Python", "Java" };
// LINQ Method syntax (lambda expression: s => s.Contains("Tutorials"))
var result = stringList.Where(s => s.Contains("Tutorials"));
```

Trong ví dụ trên **Dấu " $=>$ " là dấu gì?**.

Toàn bộ cụm "`s => s.Contains("Tutorials")`" được gọi là gì? . Dấu " $=>$ " là dấu "go-to", toàn bộ cụm đi kèm là 1 **lambda expression**. Lamda Expression là một hàm không có tên (unnamed function) với các tham số (parameters) và nội dung thực thi (body). Nội dung thực thi của Lamda expression có thể là 1 khối lệnh hoặc 1 biểu thức. Dấu " $=>$ " tách biệt các tham số và nội dung thực thi.

Bằng cách dùng biểu thức lambda, chúng ta có thể viết các hàm cục bộ có thể được chuyển như các tham số hay được trả về như giá trị của hàm gọi. Biểu thức lambda rất hữu ích cho việc viết các biểu thức truy vấn LINQ.

5.2.2 Danh sách truy vấn LINQ

Danh sách phân loại các phương thức truy vấn của LINQ được liệt kê trong bảng dưới đây:

Loại	Phương thức sử dụng
Lọc dữ liệu	Where
Chọn dữ liệu	Select, SelectMany
Phân vùng dữ liệu	Take, Skip, TakeWhile, SkipWhile
Kết hợp	Join, GroupJoin
Sắp Xếp	OrderBy / ThenBy, Reverse
Gom nhóm	GroupBy
Toán tử tập hợp	Distinct, Union, Intersect, Except
Chuyển đổi kiểu dữ liệu	ToSequence, ToArray,ToList, ToDictionary, ToLookup, OfType, Cast
Phản tử	First, FirstOrDefault, Last, LastOrDefault, Single, SingleOrDefault, ElementAt, ElementAtOrDefault, DefaultIfEmpty

Các hàm tính toán kết hợp	Count, LongCount, Sum, Min, Max, Average, Aggregate
Toán tử phát sinh	Range, Repeat, Empty
Toán tử lượng hóa	Any, All, Contains
Trộn	Concat

Thông qua tên gọi chúng ta có thể hình dung sơ qua về khả năng của phương thức, chúng ta sẽ tìm hiểu một số phương thức hay sử dụng

Để bắt đầu với làm quen với cú pháp truy vấn của LINQ, trước tiên bạn hãy thêm namespace **System.Linq** vào lớp cần sử dụng. Chúng ta sẽ làm việc với hai mảng dữ liệu (names, ages) để minh họa cho các việc sử dụng danh sách truy vấn bằng cả 2 cách giữa Method syntax và Query syntax:

```
string[] names = new string[]{
    "Anh",
    "Bình",
    "Hải",
    "Thu",
    "Hương",
    "Tuấn",
    "Trung",
    "Dũng",
    "Dương",
    "Thủy",
    "Vân"
};

int[] ages = new int[] { 12, 34, 16, 78, 34, 25, 16, 25, 52, 16, 36 };
```

5.2.2.1 Lọc dữ liệu (Filter)

Lọc dữ liệu là câu lệnh truy vấn phổ biến ở dạng diễn giải Boolean (đúng hoặc sai). Câu truy vấn chỉ trả về các phần tử nếu diễn giải là đúng (true). Để lọc dữ liệu, chúng ta dùng mệnh đề where, trong đó mô tả các điều kiện lọc.

Ví dụ: *Lấy những tên (names) có từ bắt đầu bằng chữ "T"?*

```
// cách 1: sử dụng query syntax
IQueryable<string> x = from n
    in names
    where n.StartsWith("T")
    select n;

// cách 2: sử dụng method syntax (lambda expression: n => n.StartsWith("T"))
IQueryable<string> y = names.Where(n => n.StartsWith("T"));
```

Trong câu truy vấn ở ví dụ trên, chúng ta tìm những tên (names) có bắt đầu bằng chữ "T". Bạn có thể dùng kết hợp các toán tử AND và OR trong C# để áp dụng các diễn

giải lọc cần thiết trong mệnh đề where. Kết quả trả về trong 2 cách này đều giống nhau “Thu”, “Tuấn”, “Trung”, “Thủy”.

5.2.2.2 Các trường chọn dữ liệu

Toán tử **Select** dùng để lấy/trích ra dữ liệu từ mỗi phần tử trong tập dữ liệu.

Ví dụ: Lấy các độ dài của các tên tương ứng với names?

```
// cách 1: sử dụng query syntax
var x = from n in names select n.Length;

// cách 2: sử dụng method syntax (lambda expression: n => n.Length)
var y = names.Select(n => n.Length);
```

5.2.2.3 Kết hợp (join)

Tương tự như SQL, kết hợp (join) dữ liệu xảy ra giữa các tập đối tượng dữ liệu mà chưa được mô hình rõ ràng trong nguồn dữ liệu.

Ví dụ: Kết hợp mảng names và ages theo cùng vị trí index để tạo ra mảng có tên và tuổi?

```
// #1: query syntax
var innerJoinQuery = from name in names.Select((item, index) => new { item, index })
                     join age in ages.Select((item, index) => new { item, index }) on name.index
                     equals age.index
                     select new
                     {
                         Name = name.item,
                         Age = age.item
                     };
```

Trong LINQ, chúng ta không cần phải dùng mệnh đề join thường xuyên như trong SQL bởi vì các khóa ngoại trong LINQ thông thường được mô tả trong mô hình đối tượng (object model) như là các thuộc tính liên kết 1 tập các phần tử.

5.2.2.4 Phân vùng dữ liệu

Phân vùng trong LINQ là toán tử chia 1 tập dữ liệu đầu vào thành các phần, mà không cần sắp xếp lại các phần tử và trả về 1 trong các phần đó. Các phương thức toán tử truy vấn chuẩn thực thi việc phân vùng được sử dụng như Skip, SkipWhile, Take, TakeWhile.

Skip: Bỏ qua các phần tử theo 1 vị trí cụ thể trong tập dữ liệu

Take: Lấy các phần tử theo 1 vị trí cụ thể trong tập dữ liệu

SkipWhile: Bỏ qua các phần tử dựa trên một hàm cho trước cho đến khi một phần tử không thỏa mãn điều kiện.

TakeWhile: Lấy các phần tử dựa trên 1 hàm cho trước cho đến khi 1 phần tử không thỏa mãn điều kiện.

Ví dụ: *Lấy mảng các names bỏ qua 2 vị trí đầu tiên?*

```
// #1: query syntax
var x = from n in names.Skip(2) select n;

// #2: method syntax
var y = names.Skip(2);
```

Ví dụ: *Lấy mảng các tên của 2 vị trí đầu tiên trong mảng names?*

```
// #1: query syntax
var x = from n in names.Take(2) select n;

// #2: method syntax
var y = names.Take(2);
```

5.2.2.5 Sắp xếp (Order)

Mệnh đề OrderBy cho phép sắp xếp các phần tử theo thứ tự nào đó trong dữ liệu trả về, mặc định là xếp tăng dần. Bạn có thể thêm từ khóa descending phía sau để xếp theo chiều ngược lại (giảm dần).

Ví dụ: *Xếp mảng names theo độ dài giảm dần, nếu các tên có cùng độ dài sẽ xếp theo thứ tự alphabet?*

```
// #1: query syntax
var x = from n in names
        orderby n.Length descending, n
        select n;
// #2: method syntax
var y = names.OrderByDescending(n => n.Length).ThenBy(n => n);
```

5.2.2.6 Gom nhóm (Group)

Mệnh đề group cho phép gom nhóm kết quả dựa trên 1 khóa (key) được mô tả. Nếu thực hiện 1 câu truy vấn trong 1 mệnh đề group, kết quả là dạng 1 danh sách lồng 1 danh sách. Mỗi phần trong danh sách là 1 phần tử có 1 thành phần khóa và 1 danh sách các phần tử được gom nhóm theo khóa đó.

Ví dụ: Gom nhóm các tên theo độ dài length các tên?

```
// #1: query syntax
var x = from n in names
        group n by n.Length;

// #2: method syntax
var y = names.GroupBy(key => key.Length);
```

Kết quả là dạng 1 danh sách lồng 1 danh sách. Chúng ta có thể dùng vòng lặp foreach để lấy các nhóm tên theo cùng độ dài length, trong mỗi nhóm đó lại lấy xuất ra tên của từng người như ví dụ trên. Ví dụ trên có kết quả như dưới đây:

```
//sử dụng mang group x hoặc y ở trên
foreach (var customerGroup in x)
{
    Console.WriteLine(customerGroup.Key);
    foreach (string i in customerGroup)
    {
        Console.WriteLine("    {0}", i);
    }
}

//Kết quả: 3 nhóm length: nhóm 3 kí tự, nhóm 4 kí tự , nhóm 5 kí tự
```

```

3
    Anh
    Hải
    Thu
    Vân
4
    Bình
    Tuấn
    Dũng
    Thùy
5
    Hương
    Trung
    Dương

```

5.2.2.7 Các dạng chuyển đổi dữ liệu

Các phương thức chuyển đổi thay đổi dạng đối tượng đầu vào. Các toán tử chuyển đổi trong các truy vấn LINQ được sử dụng ở nhiều ứng dụng khác nhau. Một số phương thức hay sử dụng được mô tả sau đây:

ToArray: Chuyển 1 tập hợp thành 1 mảng. Phương thức này ép buộc việc thực thi truy vấn.

ToList: Chuyển 1 tập thành kiểu List. Phương thức này ép buộc việc thực thi truy vấn.

OfType: Lọc các giá trị, phụ thuộc vào khả năng của nó có thể được chuyển đổi thành 1 dạng cụ thể

Ví dụ: Tạo 1 class Student có các đối tượng Name. Lấy list các students có tên tương ứng như mảng names?

```
// Tạo class có đối tượng Name
class Student
{
    public string Name;
}

// #1: query syntax
List<Student> listStudent = (from n in names
                                select new Student() { Name = n }).ToList();

// #2: method syntax
List<Student> listStudent = names.Select(s => new Student() { Name = s }).ToList();
```

5.2.2.8 Các hàm tính toán tổng hợp

Hàm tổng hợp được yêu cầu để thực hiện các phép toán như Trung bình, Tổng, Đếm, Tối đa, Tối thiểu và Tổng trên thuộc tính số của các phần tử trong tập hợp và các phương thức này được gọi là phương thức mở rộng.

Average: Tính trung bình của các số trong bộ dữ liệu nguồn.

Count: Đếm số lượng phần tử

Max: Tìm giá trị lớn nhất

Min: Tìm giá trị nhỏ nhất

Sum: Tính tổng các giá trị

Ví dụ: Tính tuổi trung bình của mảng ages?

```
// #1: query syntax
var x = (from i in ages select i).Average();

// #2: method syntax
var y = ages.Average();
```

Ví dụ: Trong mảng ages có bao nhiêu name có độ dài lớn hơn bằng 4 kí tự ?

```
// #1: query syntax
var x = (from n in names
          where n.Length >= 4
          select n).Count();

// #2: method syntax
var y = names.Where(n=>n.Length>=4).Count();
```

5.2.2.9 Toán tử phần tử

Toán tử phần tử trả về 1 phần tử đơn, cụ thể từ 1 tập. Một số phương thức toán tử truy vấn hay sử dụng được mô tả dưới đây:

First: Lấy giá trị đầu tiên của 1 tập hợp hoặc giá trị đầu tiên thỏa mãn 1 điều kiện nào đó.

FirstOrDefault: Lấy giá trị đầu tiên của 1 tập hợp hoặc giá trị đầu tiên thỏa mãn 1 điều kiện nào đó. Trả về giá trị mặc định nếu không có phần tử nào thỏa mãn.

Last: Trả về phần tử cuối của 1 tập hợp, giá trị cuối thỏa mãn 1 điều kiện nào đó.

LastOrDefault: Trả về phần tử cuối của 1 tập hợp, giá trị cuối thỏa mãn 1 điều kiện nào đó. Trả về giá trị mặc định nếu không có phần tử nào thỏa mãn.

ElementAt: Trả về phần tử tại 1 vị trí cụ thể trong 1 tập hợp

ElementAtOrDefault: Trả về phần tử tại 1 vị trí cụ thể trong 1 tập hợp hoặc 1 giá trị mặc định nếu vị trí đó nằm ngoài tập hợp.

Ví dụ: Lấy ra tên của người đầu tiên có kí tự bắt đầu là 'H' trong mảng names?

```
// #1: query syntax
var x = (from n in names
          select n).First(n=>n.StartsWith("H"));

// #2: method syntax
var y = names.First(n=>n.StartsWith("H"));
```

5.2.2.10 Toán tử tập hợp

Toán tử tập hợp là các toán tử truy vấn sinh ra 1 tập kết quả dựa trên sự có mặt hoặc vắng mặt của các phần tử tương đương bên trong các tập hợp giống nhau hoặc tách rời. Một số phương thức toán tử truy vấn hay sử dụng được mô tả dưới đây:

Distinct: Bỏ các kết quả trùng lặp trong 1 tập hợp

Except: Trả về các phần tử của 1 tập mà không xuất hiện trong tập thứ 2

Union: Trả về tập hợp nhất của 2 tập, tập những phần tử phải ít nhất thuộc về 1 trong 2 tập

Intersect: Trả về một tập giao những phần tử xuất hiện trong 2 tập cho trước

Ví dụ: *Lấy ra mảng các tuổi được loại trừ phần tử giống nhau trong ages?*

```
// #1: query syntax
var x = (from n in names
          select n).Distinct();
// #2: method syntax
var y = ages.Distinct();
```

Như vậy LINQ là ngôn ngữ truy vấn đa năng mà chúng ta cần sử dụng chúng ở tất cả mọi nơi trong ứng dụng .NET. Khi đã quen LINQ, chúng ta sẽ thấy lambda là công cụ rất mạnh, linh hoạt, và ngắn gọn để viết các truy vấn.

TÓM TẮT

Trong bài này, học viên làm quen với cách sử dụng LINQ trong ngôn ngữ C# bao gồm:

- *Cú pháp truy vấn và cú pháp phương thức*
- *Danh sách truy vấn LINQ*
- *Năm được các truy vấn (lọc, chọn, phân vùng, gom nhóm, sắp xếp...)*

CÂU HỎI ÔN TẬP

Câu 1: Để sử dụng LINQ sử dụng thư viện nào?

Câu 2: Có bao nhiêu cú pháp truy vấn trong LINQ? Phân biệt sự khác nhau

Câu 3: Biểu thức Lambda là gì?

Câu 4: Sự khác nhau giữa **IEnumerable** và **IQueryable** là gì?

Câu 5: Sự khác nhau giữa **First** và **FirstOrDefault** trong LINQ?

Sử dụng LINQ viết theo 1 trong 2 cách để thực hiện các truy vấn trong danh sách list các đối tượng Player. Player gồm tên, đội, và điểm số ghi được. Như chương trình minh họa ở dưới.

```
class Player
{
    public string Name;
    public string Team;
    public float Score;
}

class Program
{
    static void Main(string[] args)
    {
        Console.OutputEncoding = Console.InputEncoding= Encoding.Unicode;
        var listPlayer = new List<Player> {
            new Player { Name = "Anh", Team = "A", Score = 10 },
            new Player { Name = "Vân", Team = "A", Score = 25 },
            new Player { Name = "Sơn", Team = "L", Score = 60 },
            new Player { Name = "Hải", Team = "L", Score = 20 },
            new Player { Name = "Dũng", Team = "A", Score = 40 },
            new Player { Name = "Vinh", Team = "L", Score = 30 },
            ///// có thể thêm nhiều dữ liệu để test
        };
        //Thực hiện các câu sau sử dụng LINQ (câu 6- 10)
        //...
    }
}
```

Câu 6: Lấy danh sách tên người chơi thuộc Team = "A" và có điểm Score > 25

Câu 7: Sắp xếp các tên các người chơi theo thứ tự điểm tăng dần?

Câu 8: Lấy ra tên 2 người chơi có điểm cao nhất?

Câu 9: Lấy ra tên các Team và điểm trung bình của team đó?

Câu 10: Lấy danh sách người chơi có tên chứa kí tự 'n' không phân biệt hoa thường?

BÀI 6: WINDOWS FORM

Sau khi học xong bài này, học viên có thể nắm được:

- *Graphical User Interface (GUI)*
- *Event Driven Programming*
- *Ứng dụng Windows Form dùng C#*
- *Khuôn mẫu của ứng dụng Windows Form chuẩn*
- *Cách tạo ứng dụng Windows Form trong VS 2005*
- *Tạo ứng dụng Form*
- *Chỉnh sửa form*
- *Thêm component vào form*
- *Viết phần xử lý cơ bản*

6.1 CÁC KHÁI NIỆM

6.1.1 Giao diện đồ họa – Graphical User Interface

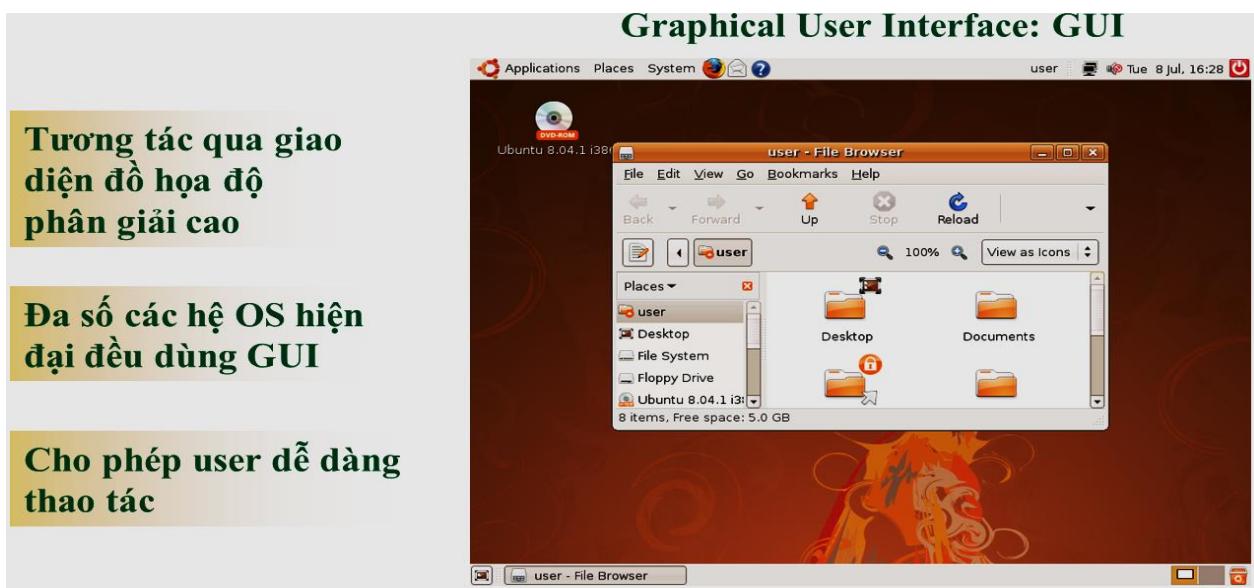
Trong các ứng dụng hiện đại, giao diện người dùng trực quan chiếm vị trí hết sức quan trọng. Việc trình diễn đúng thông tin, theo đúng cách và vào đúng thời điểm có thể đem lại những giá trị kinh tế xã hội đáng kể. Với những ứng dụng thương mại, chẳng hạn một ứng dụng bán hàng trực tuyến, việc cung cấp một giao diện người dùng mạnh có thể tạo nên sự khác biệt giữa một công ty với các đối thủ cạnh tranh, góp phần làm tăng doanh số và giá trị thương hiệu của hãng này so với hãng khác.

Microsoft .NET Framework chứa một tập phong phú các lớp dùng để tạo các ứng dụng dựa-trên-Windows truyền thống trong không gian tên System.Windows.Forms.Các lớp này có phạm vi từ các phần cơ bản như các lớp

TextBox, Button, và MainMenu đến các điều kiểm chuyên biệt như TreeView, LinkLabel, và NotifyIcon. Ngoài ra, bạn sẽ tìm thấy tất cả các công cụ cần thiết để quản lý các ứng dụng giao diện, đa tài liệu (Multiple Document Interface—MDI), tích hợp việc trợ giúp cảm-ngữ-cảnh, và ngay cả tạo các giao diện người dùng đa ngôn ngữ—tất cả đều không cần viện đến sự phức tạp của Win32 API.

Chương trình hiện đại đều dùng GUI trong đó:

- Graphical: text, window, menu, button...
- User: người sử dụng chương trình
- Interface: cách tương tác chương trình



Hình 6.1: Giao diện đồ họa người dùng

Trong Windows thành phần đồ họa điển hình:

- Window: một vùng bên trong màn hình chính
- Menu: liệt kê những chức năng
- Button: nút lệnh cho phép click vào
- TextBox: cho phép user nhập dữ liệu text

Windows Form là nền tảng GUI cho ứng dụng desktop (ngược với Web Form ứng dụng cho Web):

- Single Document Interface (SDI)

- Multiple Document Interface (MDI)

Các namespace chứa các lớp hỗ trợ GUI trong .NET

- System.Windows.Forms:
- Chứa GUI components/controls và form

- System.Drawing:

- Chức năng liên quan đến tô vẽ cho thành phần GUI
- Cung cấp chức năng truy cập đến GDI+ cơ bản

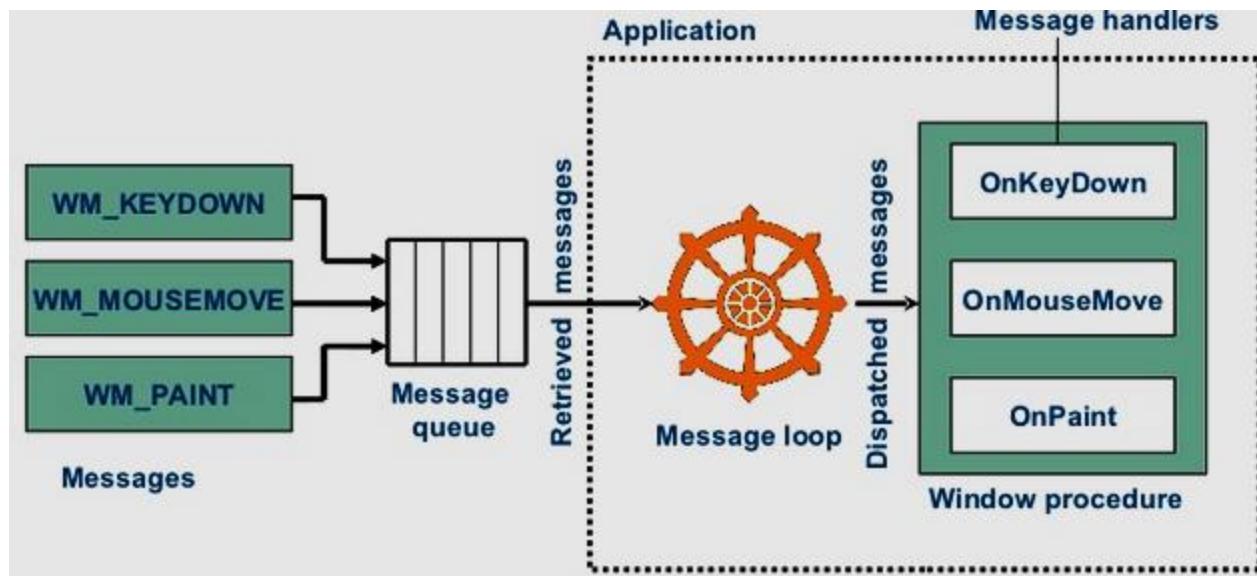
6.1.2 Cơ chế lập trình sự kiện – Event-Driven Programming

Mô hình sự kiện trong C# là một mô hình lập trình sự kiện phổ biến trong lập trình không đồng bộ. Điều căn bản trong mô hình này là khái niệm về “người xuất bản và người đăng ký” (publisher and subscribers.). Publisher sẽ làm 1 số việc và xuất bản ra 1 sự kiện, sau đó sẽ gửi chúng tới subscriber, subscriber sẽ mô tả và nhận sự kiện rõ ràng. Dưới đây là bảng so sánh giữa mô hình lập trình tuần tự, và mô hình sự kiện:

Lập trình tuần tự (DOS)	Mô hình sự kiện (Windows)
Danh sách các lệnh thực thi tuần tự	Các đối tượng có thể kích hoạt sự kiện và các đối tượng khác phản ứng với những sự kiện đó
Việc kế tiếp xảy ra chính là lệnh tiếp theo trong danh sách	Việc kế tiếp xảy ra phụ thuộc vào sự kiện kế tiếp
Chương trình được thực thi bởi máy tính	Luồng chương trình được điều khiển bởi sự tương tác User-Computer

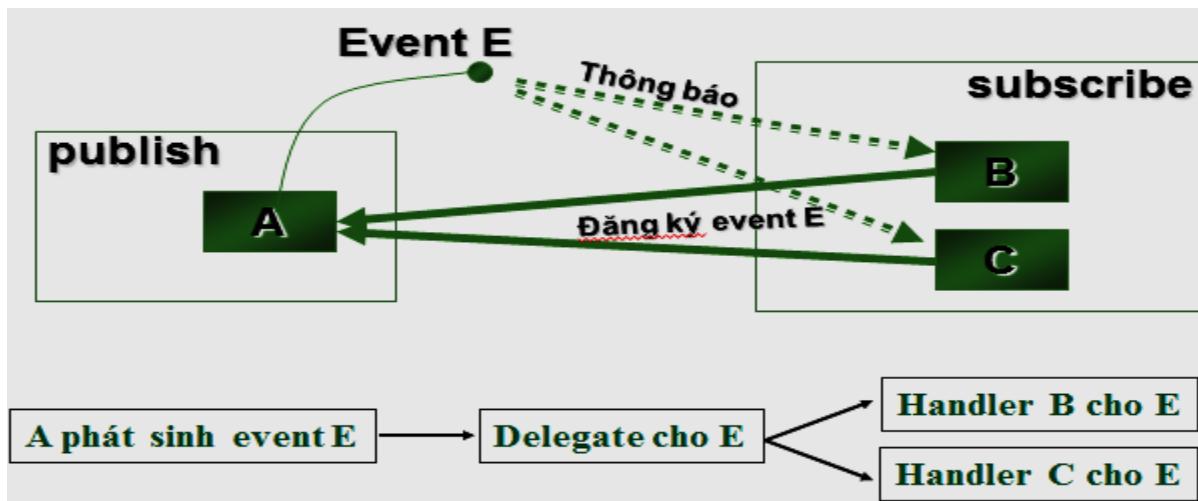
Chương trình GUI thường dùng Event-Driven Programming. chương trình chờ cho event xuất hiện và xử lý. Ví dụ sự kiện:

- Khi phát sinh một thay đổi từ chuột: di chuyển, nhấn phím trái, phím phải, v.v
- Phát sinh thông điệp từ bàn phím khi người dùng nhấn bất kỳ ký tự trên bàn phím



Hình 6.2: mô hình quản lý sự kiện của hệ điều hành

Các đối tượng đang active, có những thay đổi về chuột, bàn phím sẽ phát sinh ra một sự kiện (Firing an event), lúc này những đối tượng đăng ký lắng (Listener) nghe sự kiện sẽ đáp ứng với sự kiện bằng cách thực thi phương thức phản ứng lại sự kiện (Event handler).



Hình 6.3 mô tả cơ chế lập trình sự kiện

6.2 ỨNG DỤNG WINDOWS FORM

Ứng dụng desktop chạy trên môi trường Windows sử dụng GUI làm nền tảng, nó áp dụng mô hình Event-driven programming cho các đối tượng trên form

Ứng dụng dựa trên một “form” chứa các thành phần:

- Menu
- Toolbar
- StatusBar
- TextBox, Label, Button...

Lớp cơ sở cho các form của ứng dụng là **Form**:

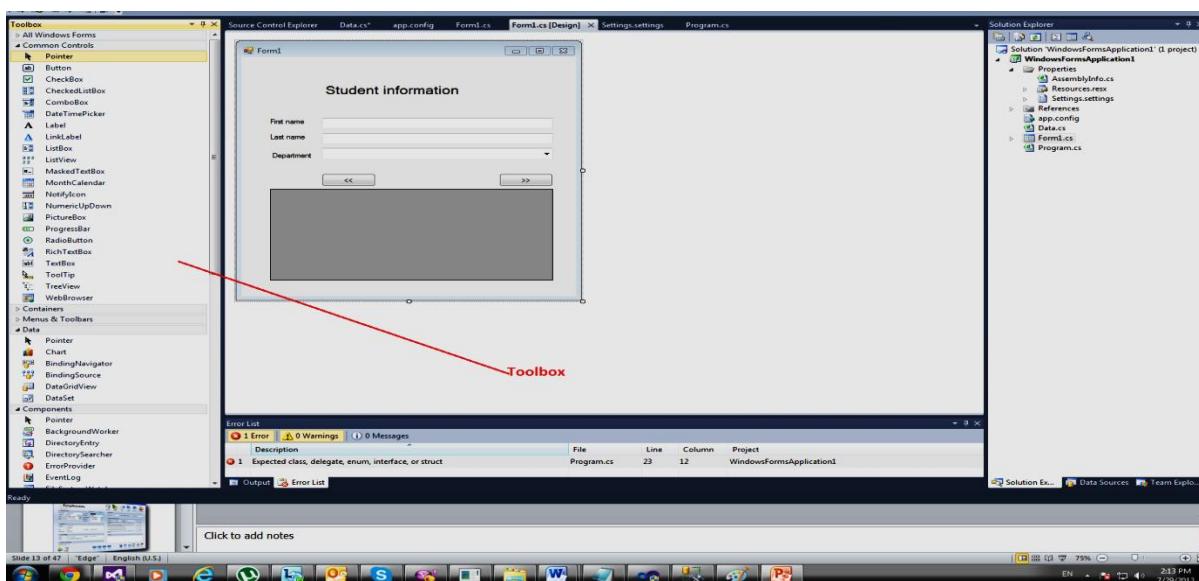


6.2.1 GUI Components/Controls

Components/controls được tổ chức vào các lớp thừa kế, cho phép dễ dàng chia sẻ các thuộc tính. Mỗi component/control định nghĩa các:

- Thuộc tính
- Phương thức
- Sự kiện

Cách dễ nhất là sử dụng VS .NET Toolbox để thêm control và component vào form



Hình 6.4: giao diện Visual studio 2012

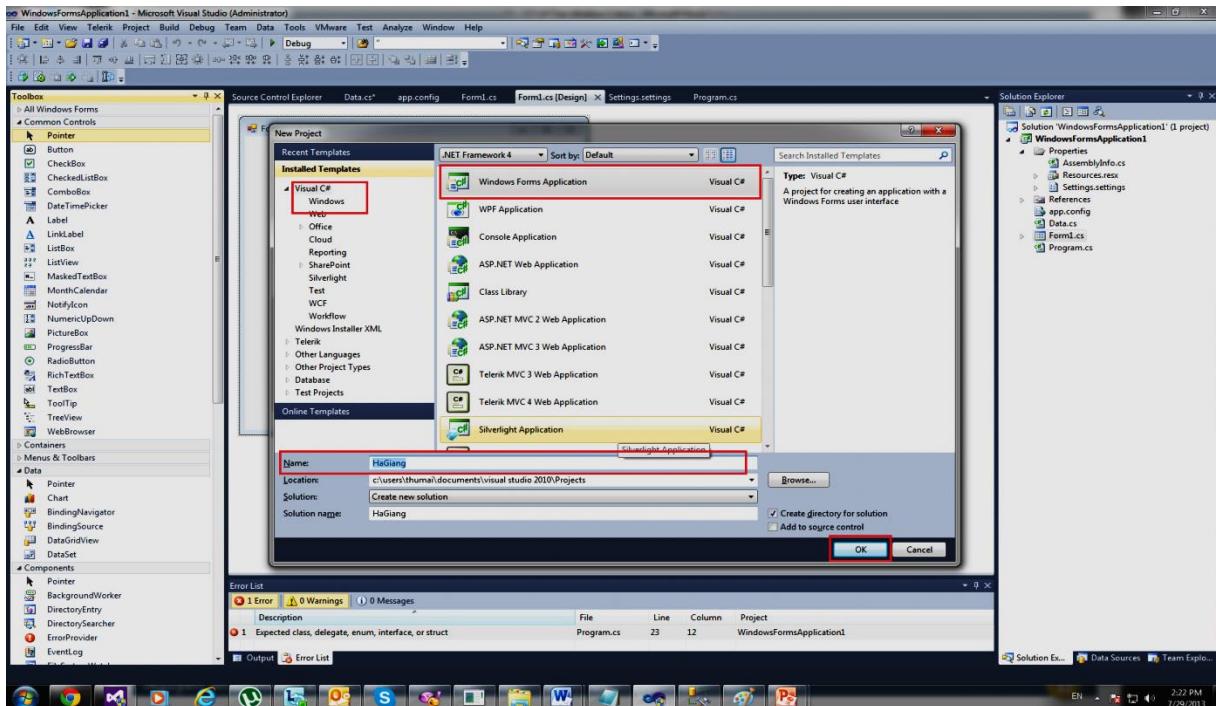
Các bước tạo ứng dụng Window Form:

- Tạo lớp kế thừa từ lớp Form cơ sở

- Bổ sung các control vào form
- Thêm các label, menu, button, textbox...
- Thiết kế layout cho form (bố trí control)
- Hiệu chỉnh kích thước, trình bày, giao diện cho form, controls chứa trong form
- Viết các xử lý cho các control trên form và các xử lý khác
- Hiển thị Form
- Thông qua lớp Application gọi phương thức Run
- Hãy tuân theo các bước sau để xây dựng một chương trình Windows đơn giản:
- Tạo ứng dụng Windows Application từ Visual Studio:

Chọn Menu File -> New -> Project, một cửa sổ sẽ hiện ra cho phép lập trình viên chọn tên ngôn ngữ lập trình (C#), chọn loại ứng dụng, và nhập tên dự án

- Visual Studio sẽ tạo project với Form1 mặc định và lớp Program.cs chứa hàm Main entry point để chạy ứng dụng



Hình 6.5: các bước tạo ứng dụng trên Windows

- Bạn có thể thêm Form vào ứng dụng, và chỉ định Form đầu tiên mà ứng dụng hiển thị tại hàm Main thông qua phương thức Run của lớp Application:
 - Tạo instant của Form và truyền vào Application.Run

```
Application.Run(new Form1());
```

```
EmployeeForm.cs [Design] Program.cs* X Form1.cs [Design] Source Control Explorer
HaGiang.Program
using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows.Forms;

namespace HaGiang
{
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new EmployeeForm());
        }
    }
}
```

- Mở cửa sổ code để xem các thành phần của một Form

Form1.cs

```
using System;
using System.Windows.Forms;
using System.Drawing;

namespace HaGiang
{
    public class Form1 : Form
    {
        Label title;
        public Form1()
        {
            this.Text = "Hello World!";
            this.Size = new Size(400, 200);
            title = new Label();
            title.Text = "Hello World";
            title.Location = new Point(50, 50);
            this.Controls.Add(title);
        }
        public static void Main(string[] args)
        {
            Application.Run(new Form1());
        }
    }
}
```

Lớp Form cơ sở

Control kiểu Label

Thiết kế form & control

Add control vào form

Chạy ứng dụng với Form1 làm form chính

TÓM TẮT

Trong bài này, học viên làm quen với các khái niệm lập trình ứng dụng có giao diện (Graphical User Interface) trên môi trường Windows bằng ngôn ngữ C#.

Chương trình hiện đại đều dùng GUI trong đó:

- *Graphical: text, window, menu, button...*
- *User: người sử dụng chương trình*
- *Interface: cách tương tác chương trình*
- *Nắm rõ cơ chế lập trình sự kiện – Event-Driven Programming:*
- *Các đối tượng có thể kích hoạt sự kiện và các đối tượng khác phản ứng với những sự kiện đó*
- *Việc kế tiếp xảy ra phụ thuộc vào sự kiện kế tiếp*
- *Luồng chương trình được điều khiển bởi sự tương tác User-Computer*

CÂU HỎI ÔN TẬP

Câu 1: Các thành phần của một ứng dụng trên Windows?

Câu 2: Các namespace chứa các lớp tạo nên giao diện cho ứng dụng Windows là gì?

Câu 3: Mô tả mô hình lập trình sự kiện trên Windows

Câu 4: Giải thích khái niệm publisher and subscribers trong lập trình sự kiện

Câu 5: Các bước để xây dựng nên ứng dụng Windows Form đơn giản?

BÀI 7: WINDOWS CONTROLS

Sau khi học xong bài này, học viên có thể nắm được:

Tổng quan controls

Property & layout của control

- *Anchor*
- *Docking*

Các control thông dụng

- *Label, TextBox, Button*
- *ListBox, ComboBox*
- *GroupBox, Panel & TabControl*
- *CheckBox, RadioButton*
- *ListView*
- *ImageList*

7.1 TỔNG QUAN VỀ CONTROLS

Một số thuộc tính chung của các Windows controls

- Text: mô tả text xuất hiện trên control
- Focus: phương thức chuyển focus vào control
- TabIndex: thứ tự của control nhận focus (mặc định được VS.NET thiết lập)
- Enable: thiết lập trạng thái truy cập của control
- Visible: ẩn control trên form, có thể dùng phương thức Hide
- Size: xác nhận kích thước của control

Common properties	Description
BackColor	Màu nền của control
BackgroundImage	Ảnh nền của control
ForeColor	Màu hiển thị text trên form
Enabled	Xác định khi control trạng thái enable
Focused	Xác định khi control nhận focus
Font	Font hiển thị text trên control
TabIndex	Thứ tự tab của control
TabStop	Nếu true, user có thể sử dụng tab để select control
Text	Text hiển thị trên form
TextAlign	Canh lề text trên control
Visible	Xác định hiển thị control

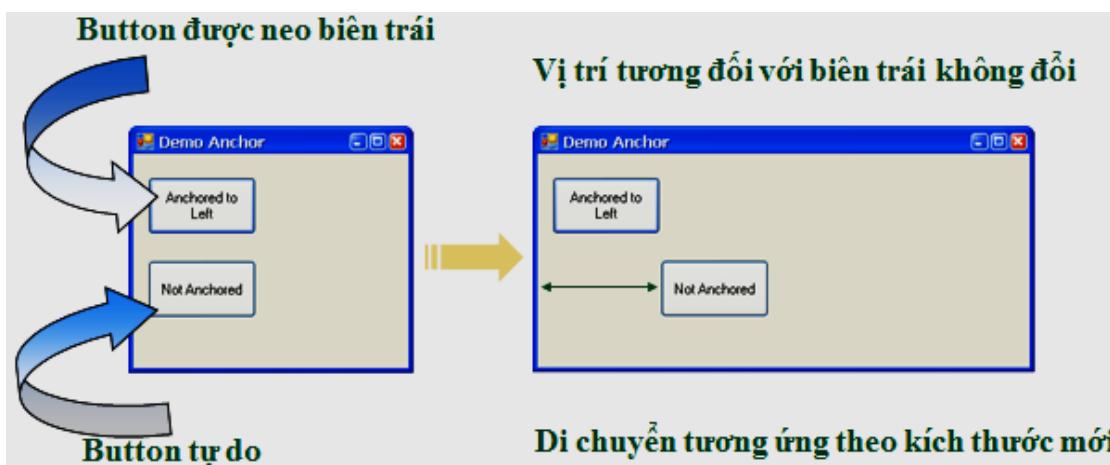
Bảng mô tả các thuộc tính chung của windows controls

- Anchor: Neo giữ control ở vị trí xác định

Khi đổi tượng Form có thuộc tính FormBorderStyle = Sizable, thì form cho phép thay đổi kích thước khi Runtime, khi thay đổi kích thước **Form** sự bối trí của control trên Form cũng thay đổi!

Thuộc tính Anchor Cho phép

- Control phản ứng lại với thao tác resize của form
- Control có thể thay đổi vị trí tương ứng với việc resize của form

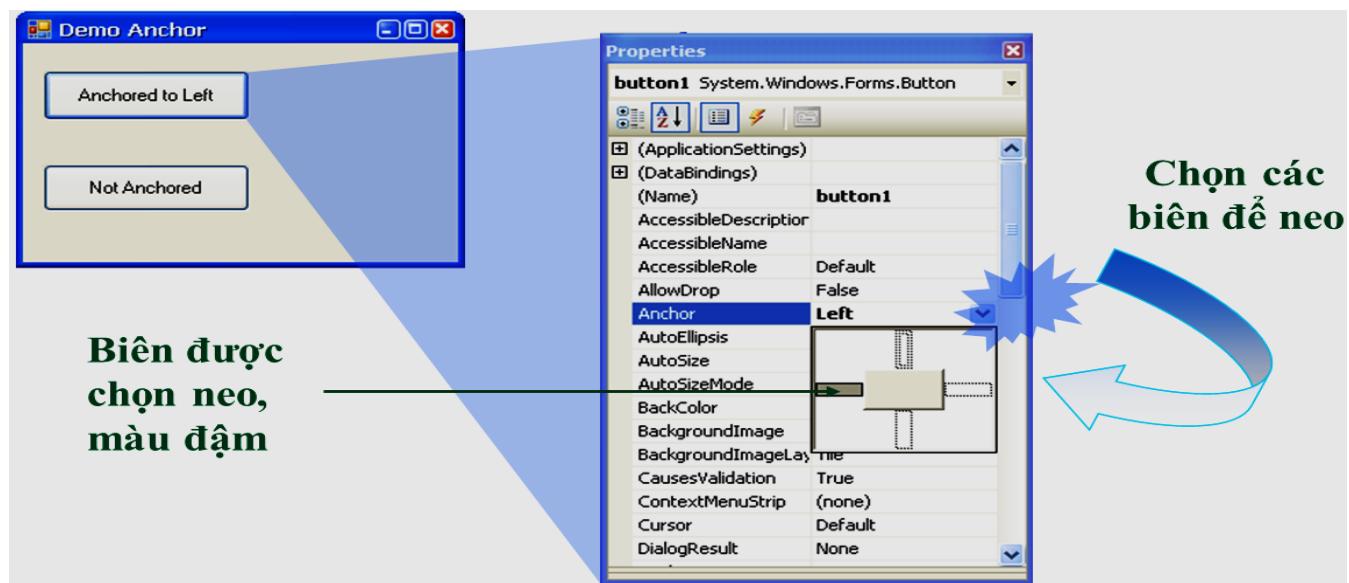


Hình 7.1: Neo biên trong ứng dụng Windows

Các trạng thái neo

- Left: cố định theo biên trái
- Right: cố định theo biên phải
- Top: cố định theo biên trên
- Bottom: cố định theo biên dưới

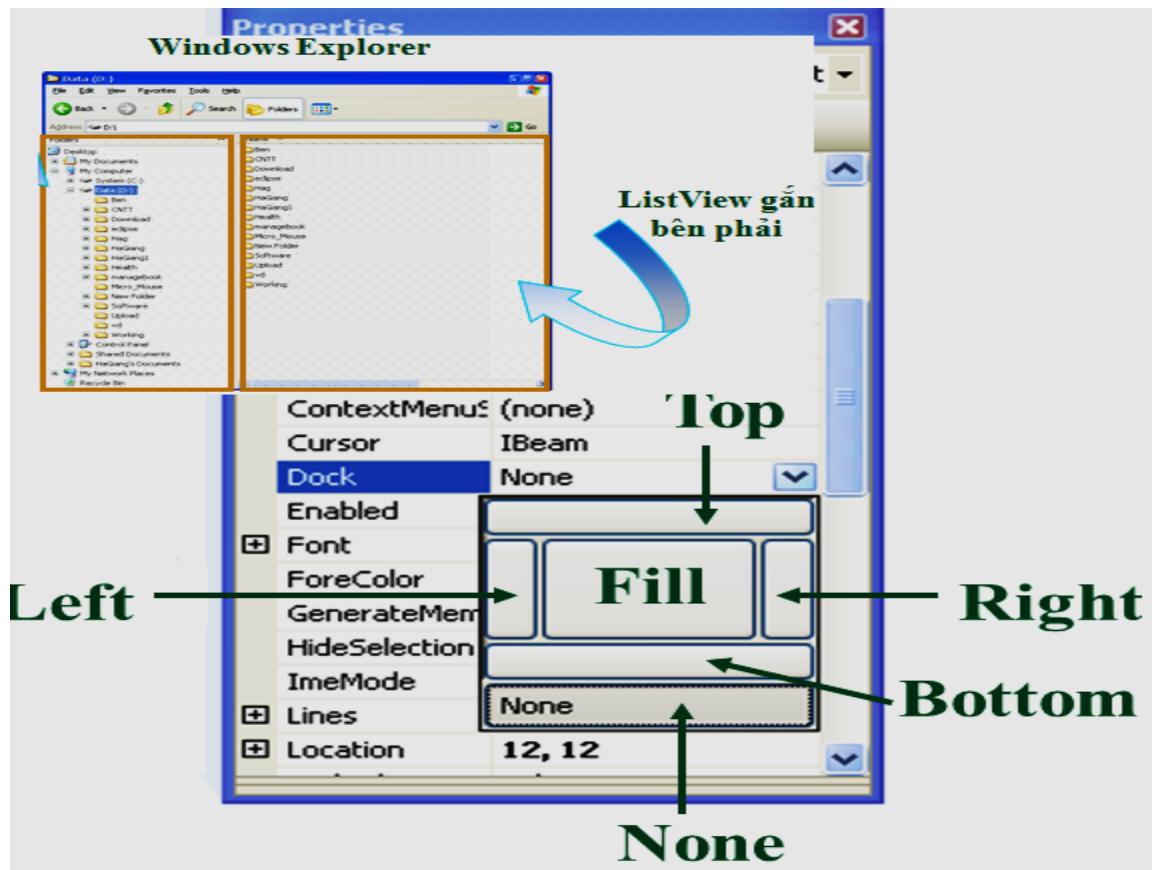
Khi lập trình trên Windows, người lập trình viên có thể dễ dàng tạo giao diện người dùng bằng cách kéo thả controls từ cửa sổ Toolbox, và thay đổi các thuộc tính của các controls ở Properties Windows để nhanh chóng tạo nên giao diện



Hình 7.2: chỉnh sửa thuộc tính Anchor cho các controls

- Docking : thuộc tính dùng để bố trí các controls trên Form, hay các container

Các control có thể gắn (dock) với một cạnh nào đó của form, hoặc container của control.



Hình 7.3: chỉnh sửa thuộc tính Dock

7.2 CÁC CONTROLS CƠ BẢN

7.2.1 Label, TextBox, Button

- Lớp Label : cung cấp chuỗi thông tin chỉ dẫn

Label - Thuộc tính thường dùng	
Font	Font hiển thị của text
Text	Nội dung text hiển thị
TextAlign	Canh lề text
ForeColor	Màu text
Visible	Trạng thái hiển thị

- TextBox: cho phép user nhập dữ liệu

TextBox

<i>Thuộc tính thường dùng</i>	
AcceptsReturn	Nếu true: nhấn enter tạo thành dòng mới trong chế độ multiline
Multiline	Nếu true: textbox ở chế độ nhiều dòng, mặc định là false
PasswordChar	Chỉ hiển thị ký tự đại diện cho text
ReadOnly	Nếu true: TextBox hiển thị nền xám, và ko cho phép nhập liệu, mặc định là false
ScrollBars	Thanh cuộn cho chế độ Multiline
<i>Event thường dùng</i>	
TextChanged	Kích hoạt khi text bị thay đổi, trình xử lý được khởi tạo mặc định khi kích đúp vào textbox trong màn hình design view

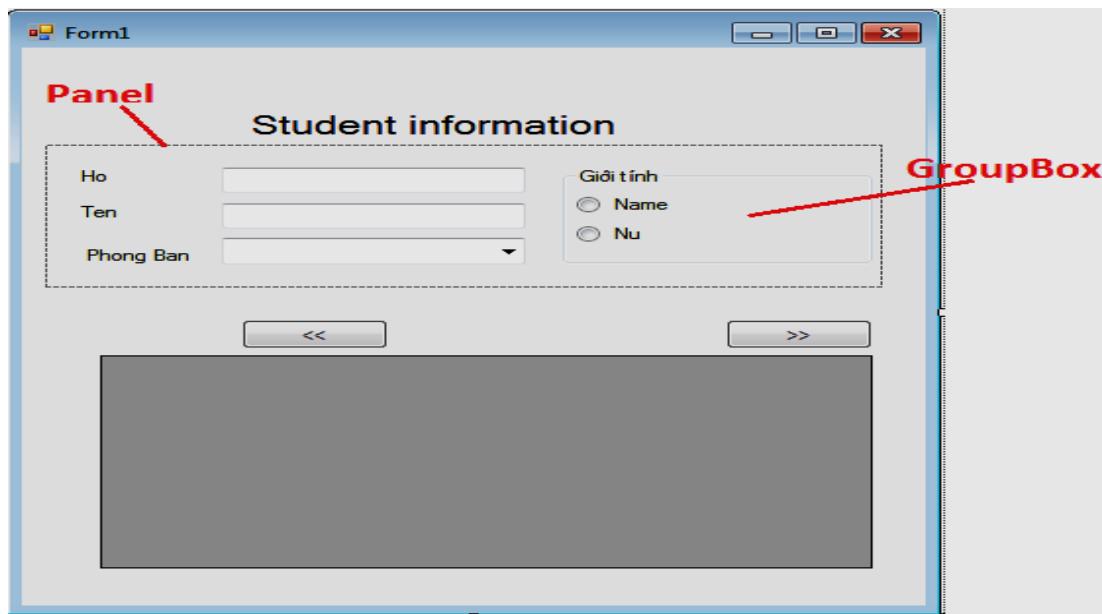
- Button: cho phép cài đặt 1 hành động.

Button	
<i>Thuộc tính thường dùng</i>	
Text	Chuỗi hiển thị trên bề mặt button
<i>Event thường dùng</i>	
Click	Kích hoạt khi user kích vào button, khai báo mặc định khi người lập trình kích đúp vào button trong màn hình Design View của Form.

7.2.2 GroupBox, Panel & TabControl

Đây là nhóm controls có thể chứa được những control khác (container), nhằm ố trí controls trên GUI. Trong đó:

- **GroupBox :**
 - Hiển thị một khung bao quanh một nhóm control
 - Thuộc tính **Text**: hiển thị một tiêu đề cho GroupBox
 - Khi xóa một GroupBox thì các control chứa trong nó bị xóa theo
 - Lớp GroupBox kế thừa từ System.Windows.Forms.Control



Hình 7.4 ví dụ về Panel và GroupBox

- Panel :

- Chứa nhóm các control
- Không có caption
- Có thanh cuộn (scrollbar) nhằm hiển thị nhiều control khi kích thước panel giới hạn

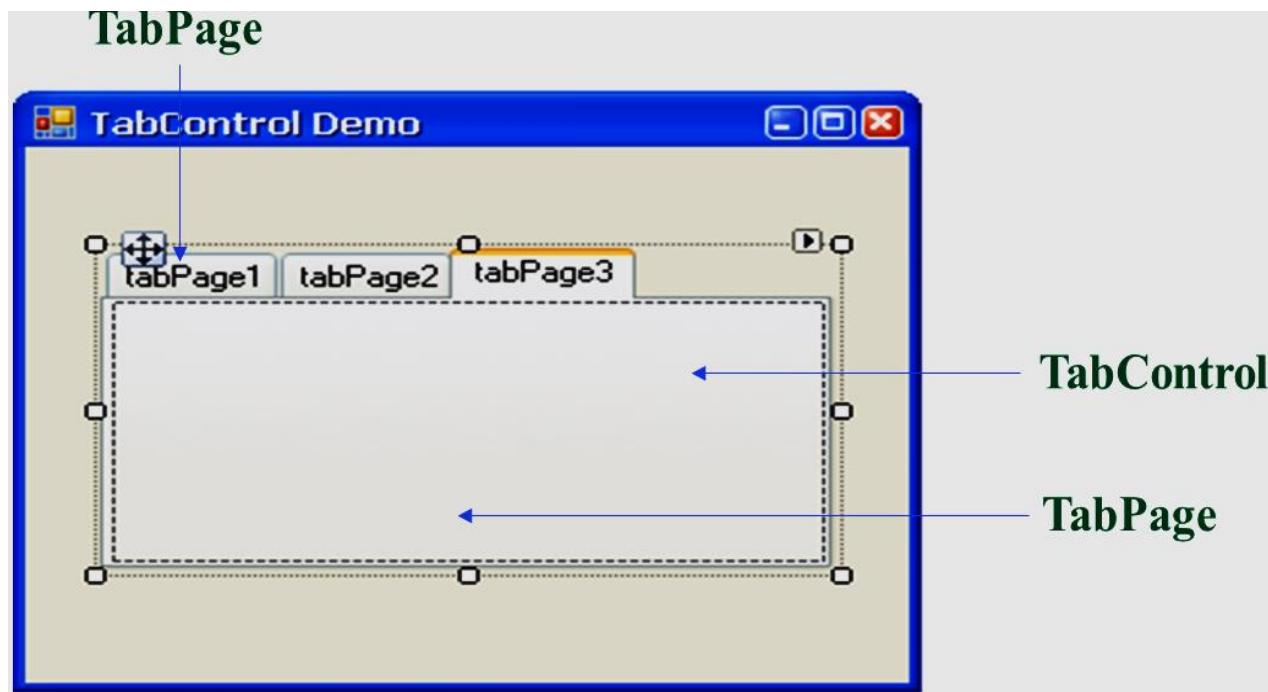
GroupBox	Mô tả
<i>Thuộc tính thường dùng</i>	
Controls	Danh sách control chứa trong GroupBox.
Text	Caption của GroupBox

Panel	
<i>Thuộc tính thường dùng</i>	
AutoScroll	Xuất hiện khi panel quá nhỏ để hiển thị hết các control, mặc định là false
BorderStyle	Biên của panel, mặc định là None, các tham số khác như Fixed3D, FixedSingle
Controls	Danh sách control chứa trong panel

- TabControl: dạng container chứa các control khác

- Cho phép thể hiện nhiều page trên một form
- Các control có cùng nhóm chức năng sẽ được tổ chức trong một tab (page)

- Cho phép thể hiện nhiều page trên một form duy nhất
- Mỗi page chứa các control tương tự như group control khác.
- Mỗi page có tag chứa tên của page
- Kích vào các tag để chuyển qua lại giữa các page



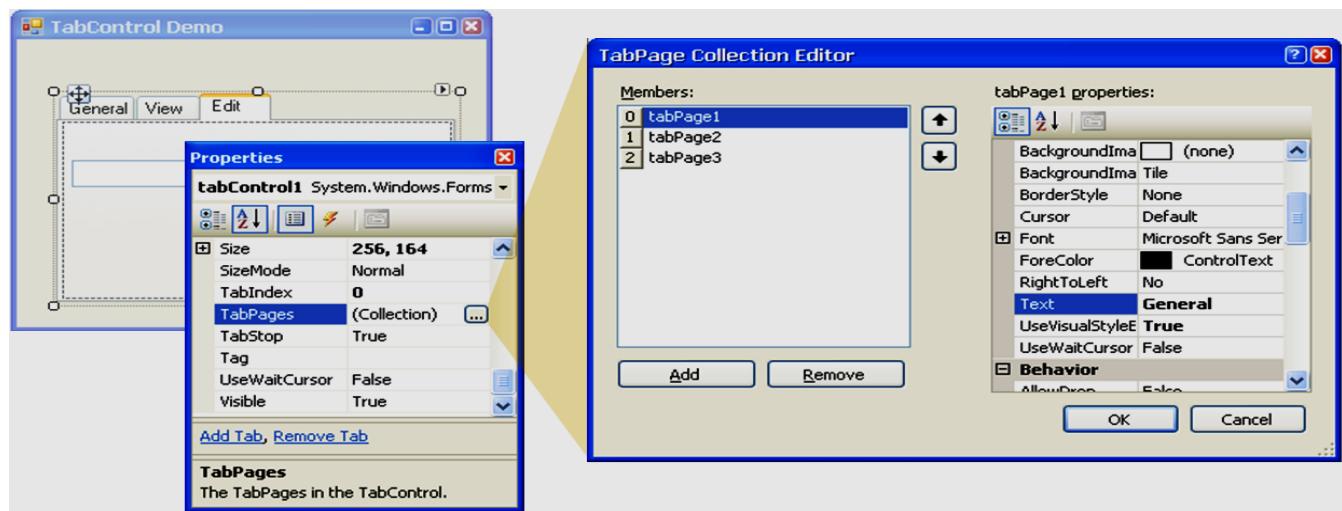
Hình 7.5: ví dụ về Tab Control

TabControl	
<i>Thuộc tính thường dùng</i>	
TabPages	Chứa tập các page
SelectedIndex	Index của active page (có giá trị từ 0 đến số page -1)
<i>Event thường dùng</i>	
SelectedIndexChanged	Sự kiện xảy ra khi change active page của TabControl

Chỉnh sửa các TabPage:

- Chọn thuộc tính TabPages của TabControl
- Sử dụng màn hình TabPage Collection Editor để chỉnh sửa
- Bổ sung Control vào TabControl:

- Chọn TabPage cần thêm control
- Kéo control từ ToolBox thả vào TabPage đã chọn



Hình 7.6: chỉnh sửa thuộc tính cho các TabPages

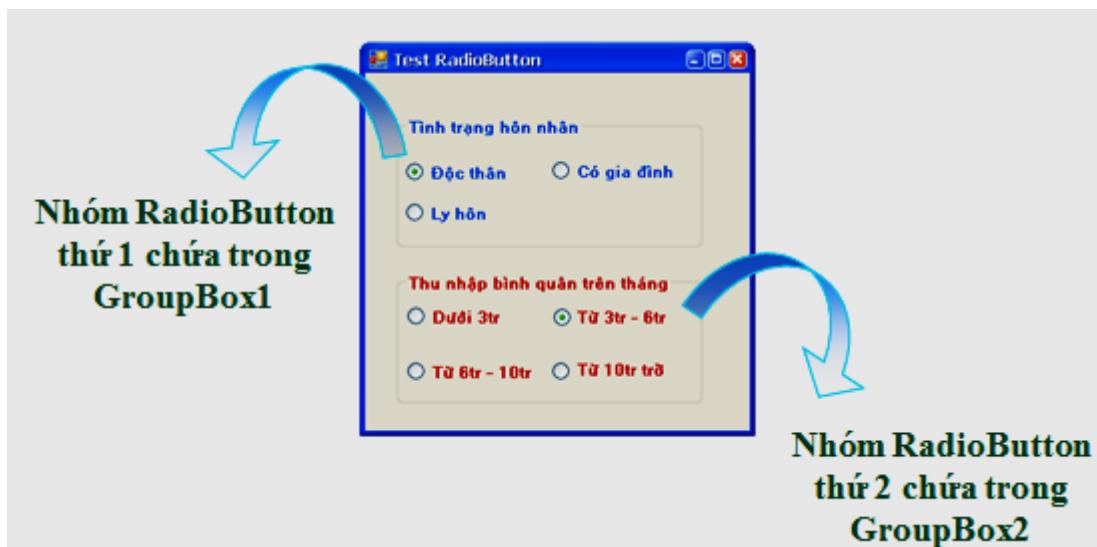
7.2.3 CheckBox, RadioButton

- CheckBox: ontrol đưa ra một giá trị cho trước và user có thể
 - Thuộc tính Checked
 - Khi checkbox được chọn: Checked = true
 - Không chọn giá trị: Checked = false
 - Thuộc tính ThreeState: thông thường CheckBox chỉ có 2 tình trạng là (true, false), nhưng nếu set thuộc tính ThreeState = true : cho phép thiết lập 3 trạng thái:
 - Checkstate = Indeterminate: không xác định
 - CheckState= Checked: chọn
 - CheckState= Unchecked: không chọn



Hình 7.7: ví dụ CheckBox

- Sự kiện mặc định CheckedChanged: xảy ra khi có sự thay đổi khi người dùng chọn (check), hoặc bỏ chọn(uncheck) lên checkbox
- RadioButton: cho phép user chọn một option trong số nhóm option
 - Khi user chọn 1 option thì tự động option được chọn trước sẽ uncheck
 - Các radio button chứa trong 1 container (form, GroupBox, Panel, TabControl) thuộc một nhóm.
 - Khác với nhóm CheckBox cho phép chọn nhiều option, còn RadioButton chỉ cho chọn một trong số các option.



Hình 7.8: ví dụ về RadioButton

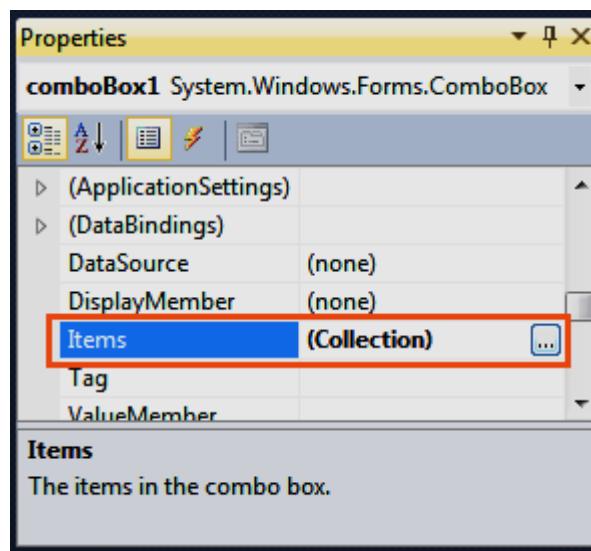
- Sự kiện mặc định CheckedChanged: xảy ra khi có sự thay đổi khi người dùng chọn (check), hoặc bỏ chọn(uncheck) lên checkbox

7.2.4 ListBox, ComboBox

- **ListBox** hiểu đơn thuần là một hộp nhỏ (vùng/ khung dữ liệu) chứa các danh sách dữ liệu được sắp xếp liền kề nhau theo chiều thẳng đứng. Mỗi phần tử trong danh sách đó gọi là 1 item, mỗi item có chứa 1 biến duy nhất kiểu string, nghĩa là mọi dữ liệu để lưu trữ trong ListBox đều phải chuyển về dạng string. Trong ListBox không thể chứa ListBox con khác.

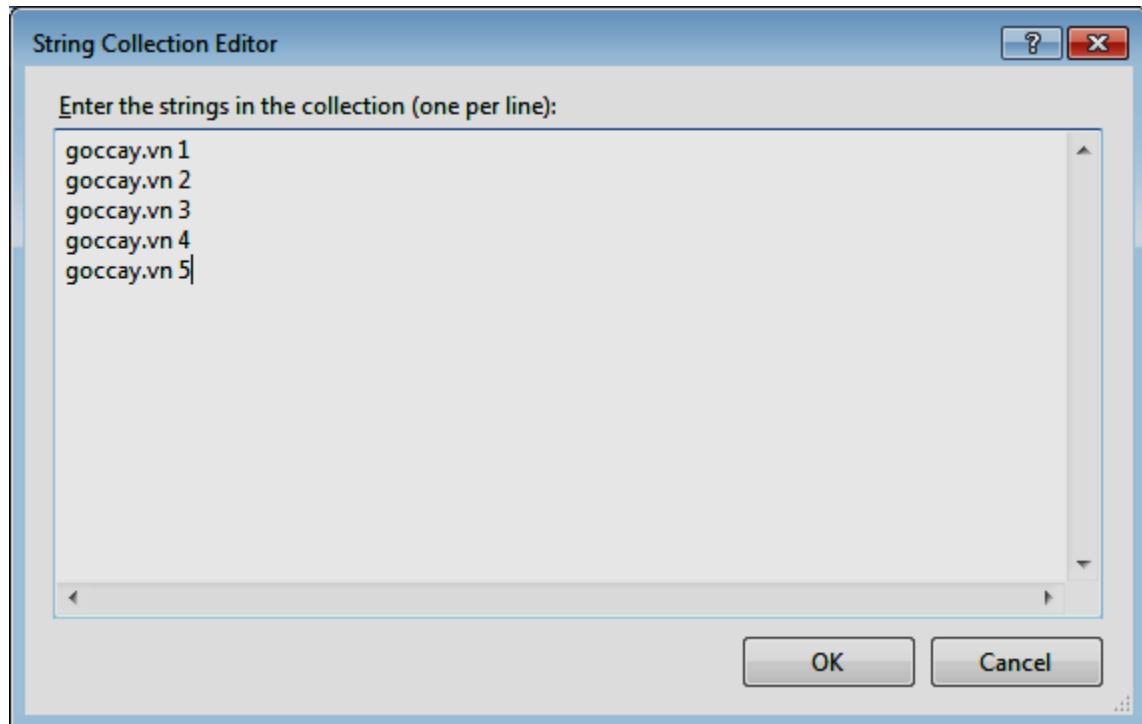
Các thuộc tính và sự kiện thường sử dụng:

- SelectedIndex: Trả về chỉ số của phần tử đang được chọn trong ListBox. Chỉ số phần tử là một số nguyên bắt đầu từ 0. Hoặc thiết lập để định vị phần tử được chọn thông qua chỉ số.
- SelectedValue: Trả về giá trị của phần tử đang được chọn
- DataSource: Đổ dữ liệu vào ListBox. Dữ liệu đưa vào phải có cấu trúc danh sách, mảng...
- Items: chứa tập item để hiển thị danh sách chọn lựa, để thêm phần tử vào danh sách, ta có hai cách tương tự ListBox



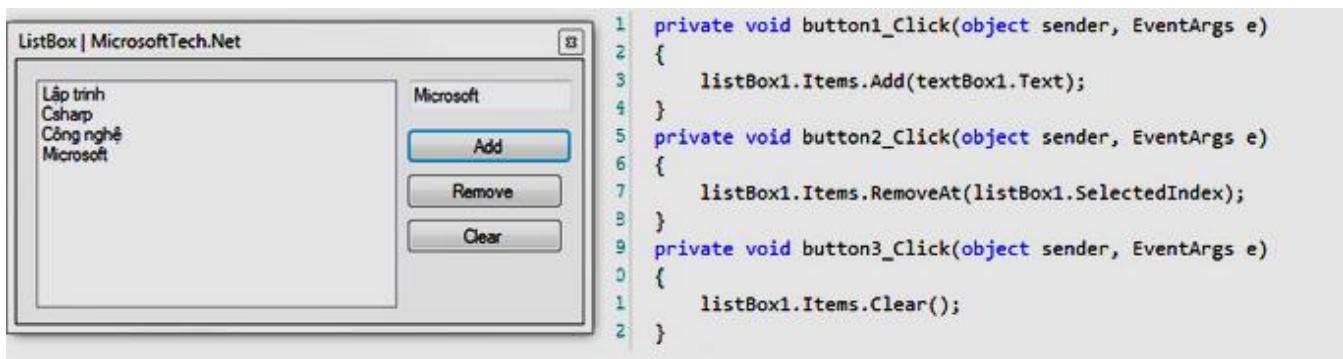
Hình 7.9 chỉnh sửa thuộc tính Items của ListBox với cửa sổ Properties

- Items.Count: Trả về số lượng phần tử có trong ListBox.
- Sau đó nhấn vào nút ... để nhập các phần tử và nhấn OK



- Hoặc thêm/loại bỏ phần tử vào bằng phương thức:
- Items.Add(): Thêm phần tử cho ListBox.
- Remove/ RemoveAt(): Xóa phần tử ra khỏi danh sách.
- Items.Clear(): Xóa bỏ tất cả các phần tử trong ListBox.
- Event SelectedIndexChanged: sinh ra khi có sự thay đổi lựa chọn các phần tử trên ListBox.

Sau đây là ví dụ đơn giản trong việc thêm, xóa dữ liệu trong ListBox.



Hình 7.10: thêm dữ liệu vào ListBox từ cửa sổ code

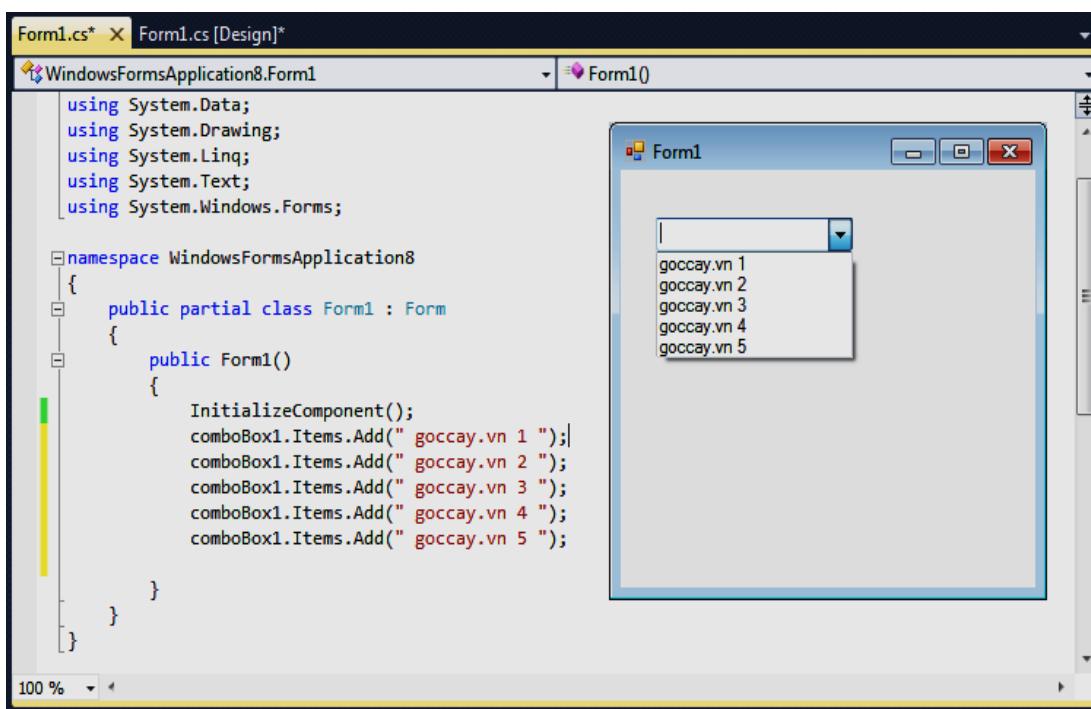
ComboBox là một trong những control dùng để thiết giao diện phổ biến nhất. Nó được sử dụng để cung cấp cho người sử dụng cơ sở lựa chọn một mục từ một danh

sách hoặc nhập vào một văn bản mới. Trong bài viết này tôi sẽ chỉ cho các bạn một số chức năng phổ biến và hữu ích của ComboBox trong C # bằng cách sử dụng Microsoft Visual Studio. Net 2010.

- **ComboBox** là 1 control kết hợp giữa 1 TextBox và 1 ListBox, vì vậy nó có các đặc tính của cả 2 control này. Trong 1 ComboBox, bạn có thể chọn 1 mục nào đó trong danh sách có sẵn của nó hay nhập 1 nội dung bất kỳ trong TextBox bên trên.

Các thuộc tính thường sử dụng:

- **DropDownStyle**: dùng để quy định kiểu của ComboBox, nó có thể dùng 1 trong 3 giá trị sau:
 - Simple: Người dùng chỉ có thể gõ một văn bản trong hộp văn bản. Danh sách các mục không được hiển thị.
 - DropDown: Người sử dụng hoặc có thể gõ một văn bản vào phần TextBox bên trên hoặc chọn một mục từ danh sách.
 - DropDown List: Người dùng chỉ có thể chọn một mục từ danh sách.
- **Items**: chứa tập item để hiển thị danh sách chọn lựa, để thêm phần tử vào danh sách, ta có hai cách tương tự ListBox



Hình 7.11: Lập trình thao tác dữ liệu với ComboBox

- Sự kiện thường sử dụng Event SelectedIndexChanged: sinh ra khi có sự thay đổi lựa chọn các phần tử trên ComboBox.

7.2.5 ListView

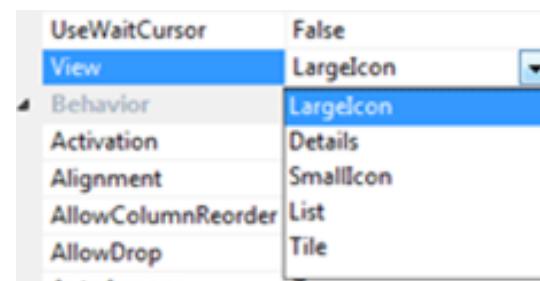
ListView là một control dùng để hiển thị một danh sách các item với các biểu tượng. Chúng ta có thể sử dụng một ListView để tạo ra một giao diện giống như cửa sổ bên phải của Windows Explorer.

ListView còn là control phổ biến hiện thị một danh sách item. Các item có thể có các item con gọi là subitem.

Ví dụ như Windows Explorer hiển thị thông tin thư mục, tập tin...

Có thể hiển thị thông tin theo nhiều dạng thông qua thuộc tính View

- Xem dạng chi tiết thông tin
- Xem dạng icon nhỏ
- Xem dạng icon lớn
- Xem dạng tóm tắt

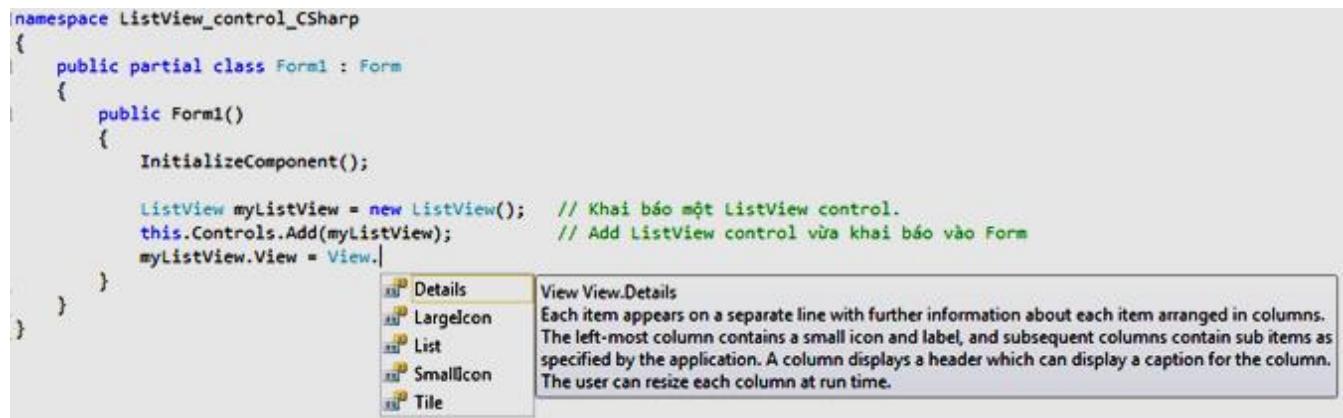


- Lớp ListView dẫn xuất từ System.Windows.Forms.Control. Sử dụng code để add ListView vào Form:

```
ListView myListview = new ListView(); // Khai báo một ListView
control.
myListview.Size = new System.Drawing.Size(390, 100); // Kích
thước hiển thị
this.Controls.Add(myListview);
```

Thay đổi chế độ xem (Changing the display modes)

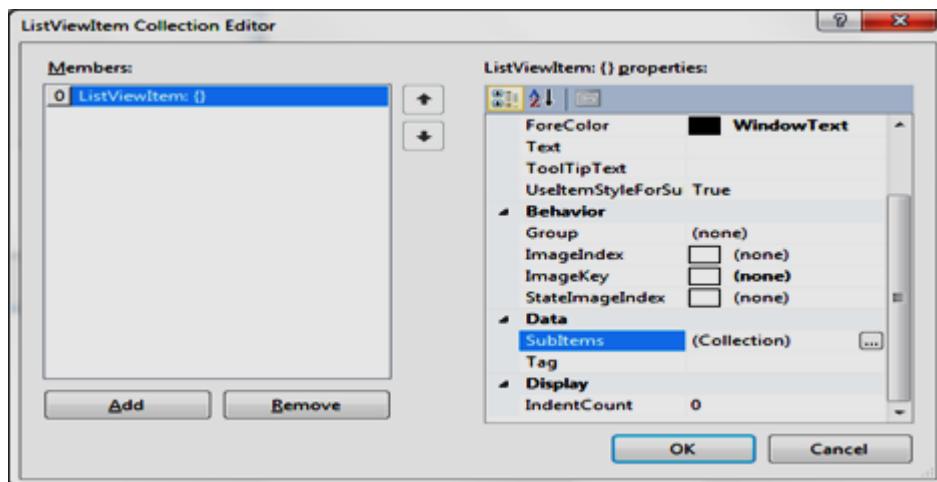
- Tùy chỉnh thuộc tính View trong cửa sổ Properties của Listview. Sẽ có 4 thuộc tính hiển thị để chúng ta lựa chọn: LargeIcon, Details, SmallIcon, List Tile.
- Sử dụng code để tùy chỉnh thuộc tính view:



Hình 7.12: Thuộc tính khung nhìn (View) với ListView

Add các item vào ListView (Khi ListView không theo cách hiển thị Details)

- Sử dụng thuộc tính Items trong cửa sổ Properties. Khi click vào button ... ở thuộc tính Items. Thì cửa sổ như hình dưới sẽ hiện ra để bạn add item vào.



Hình 7.13: Thuộc tính Items của ListView

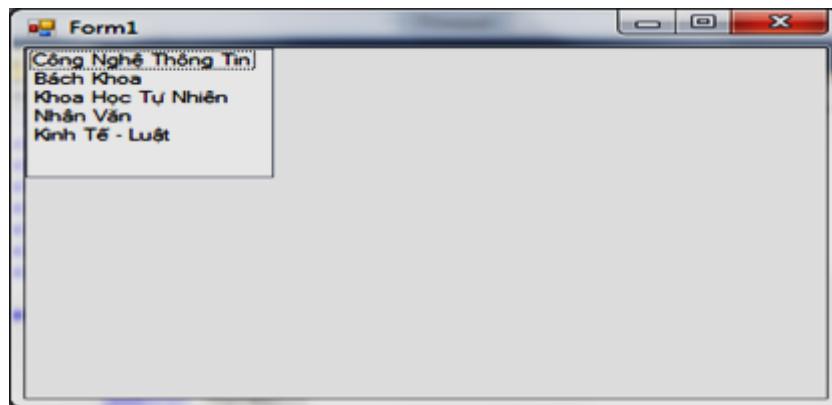
- Mỗi Item add sẽ có các thuộc tính như: Text, ForeColor, Text, ImageIndex...
- Chúng ta cũng có thể viết code để add các item vào ListView với mục đích tương tự cách làm trên. Ví dụ:

```

myListView.Items.Add("Công Nghệ Thông Tin");
myListView.Items.Add("Bách Khoa");
myListView.Items.Add("Khoa Học Tự Nhiên");
myListView.Items.Add("Nhân Văn");
myListView.Items.Add("Kinh Tế - Luật");

```

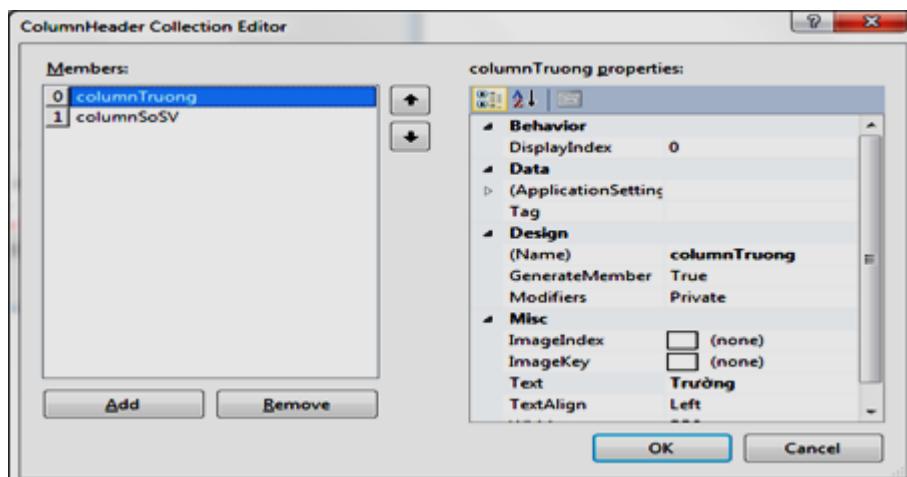
Và kết quả sẽ là:



Hình 7.14: Lập trình với thuộc tính Items của ListView

Add các cột vào ListView (Adding columns to the ListView)

Chúng ta cũng có thể thực hiện một cách đơn giản như cách add các items ở trên:



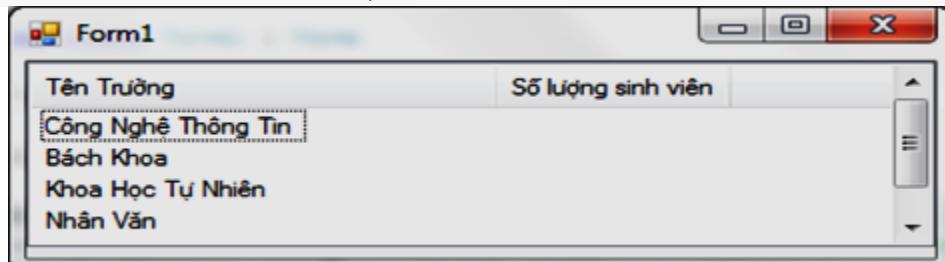
Hình 7.15: Tiêu đề cột của ListView ở khung nhìn chi tiết

- Hoặc cũng có thể sử dụng code:

```
myListView.Columns.Add("Tên Trường", 200);
myListView.Columns.Add("Số lượng sinh viên", 100);
```

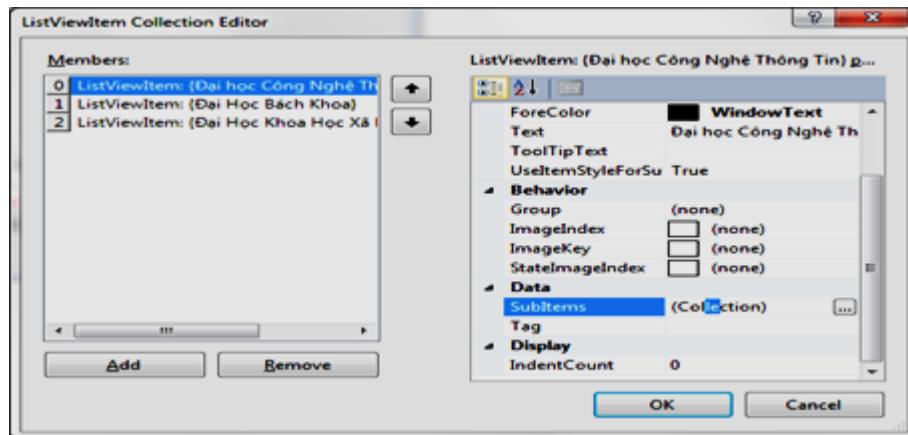
- Lưu ý: Để có thể hiển thị các columns thì chúng ta phải chọn chế độ xem là Details

```
myListView.View = View.Details;
```



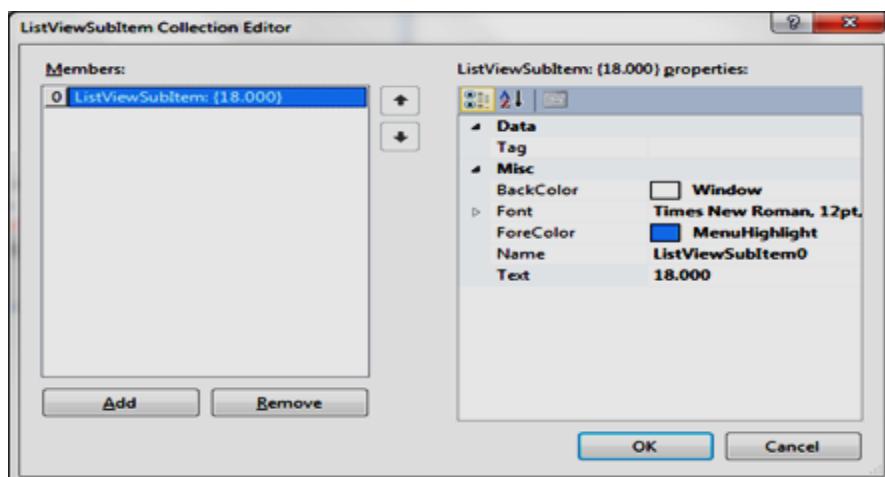
Add sub Item vào Listview (Khi ListView ở chế độ xem Details)

- Sử dụng giống như cách add các items trong phần 3 đã trình bày. Ở chúng ta click vào thuộc tính SubItem một cửa sổ mới sẽ hiện ra khá giống với cửa sổ add items



Hình 7.16: Thuộc tính SubItems của từng Item trong ListView

- Giờ chúng ta có thể add các item con cho item chính một cách bình thường giống như khi add item chính.



- Chúng ta cũng có thể sử dụng code để add các giá trị con cho item như:

```
// Add subitem
ListViewItem cntt = new ListViewItem("Công Nghệ Thông Tin");
ListViewItem.ListViewSubItem svcntt = new
ListViewItem.ListViewSubItem(cntt, "3.000 sinh viên");
cntt.SubItems.Add(svcntt);
myListView.Items.Add(cntt);

ListViewItem bk = new ListViewItem("Bách Khoa");
ListViewItem.ListViewSubItem svbk = new ListViewItem.ListViewSubItem(bk,
"18.00 sinh viên");
bk.SubItems.Add(svbk);
myListView.Items.Add(bk);
```

```

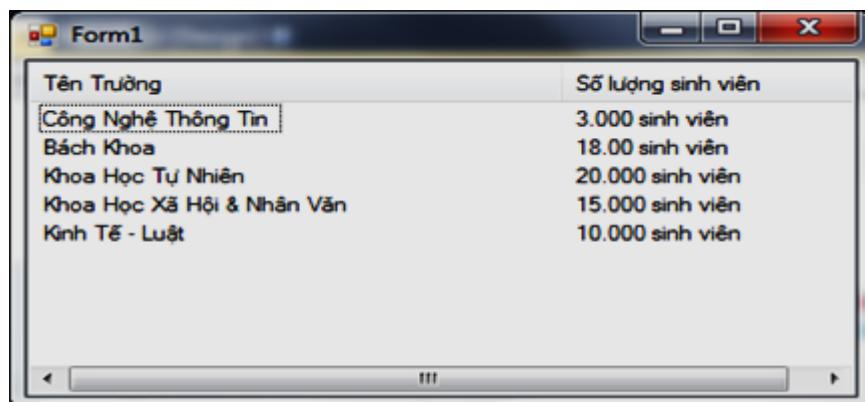
ListViewItem khtn = new ListViewItem("Khoa Học Tự Nhiên");
ListViewItem.ListViewSubItem svkhtn = new
ListViewItem.ListViewSubItem(khtn, "20.000 sinh viên");
khtn.SubItems.Add(svkhtn);
myListView.Items.Add(khtn);

ListViewItem nv = new ListViewItem("Khoa Học Xã Hội & Nhân Văn");
ListViewItem.ListViewSubItem svnv = new ListViewItem.ListViewSubItem(nv,
"15.000 sinh viên");
nv.SubItems.Add(svnv);
myListView.Items.Add(nv);

ListViewItem ktl = new ListViewItem("Kinh Tế - Luật");
ListViewItem.ListViewSubItem svktl = new
ListViewItem.ListViewSubItem(ktl, "10.000 sinh viên");
ktl.SubItems.Add(svktl);
myListView.Items.Add(ktl);

```

- Và kết quả sẽ là:



Thêm Style cho SubItems

Trong chế độ xem Details, chúng ta muốn thêm các hiển thị khác nhau của cách SubItems từ Item cha chúng ta sử dụng thuộc tính UseItemStyleForSubItem = true; Như vậy chúng ta sẽ xác định được các kiểu khác nhau cho các subitems.

Ví dụ:

```

ListViewItem ktl = new ListViewItem("Kinh Tế - Luật");
ListViewItem.ListViewSubItem svktl = new ListViewItem.ListViewSubItem(ktl,
"10.000 sinh viên");

```

```
ktl.SubItems.Add(svktl);
myListView.Items.Add(ktl);
ktl.UseItemStyleForSubItems = true;
```

Xóa item (Removing item)

Để xóa toàn bộ các item trong ListView có tên là myListview ta thực hiện lệnh
myListView.Clear();

- Để xóa item nào ta gọi phương thức Remove():

```
ListViewItem cntt = new ListViewItem("Công Nghệ Thông Tin");
cntt.Remove();
```

- Xóa item ở vị trí thứ a trong ListView ta sử dụng phương thức RemoveAt():

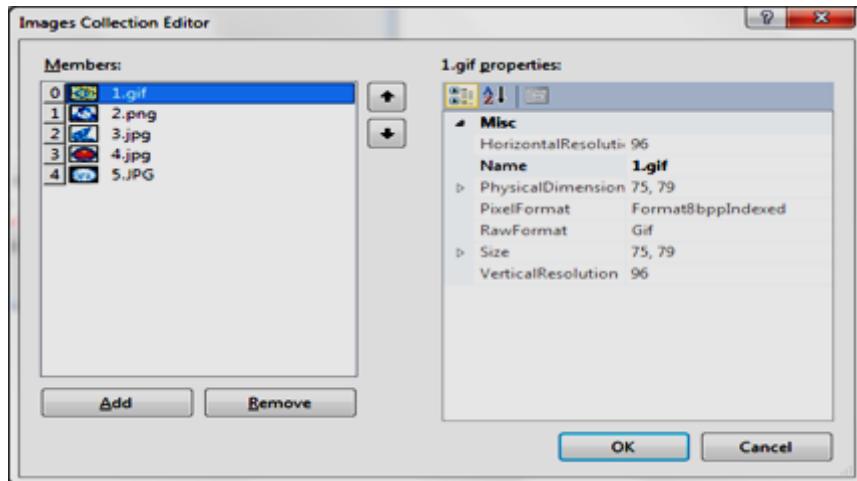
```
myListView.Items.RemoveAt(2);
```

Liên kết hình ảnh với danh sách các items

- Đây là một tính năng làm đẹp cho ListView. Để liên kết các items trong danh sách chúng ta cần phải có một imageList với một tập hợp các ảnh. Điều này được thực hiện trong phương thức ListView.Items.Add(...), sử dụng đối số imageIndex – là chỉ mục liên kết với hình ảnh trong imageList.
- Đầu tiên kéo một imageList từ Toolbox vào Form (tên mặc định sẽ là imageList1)



- Trong thuộc tính Images của imageList1 sẽ được sử dụng để add hình ảnh vào imageList1 như:



Hình 7.17: Thiết lập tập hình ảnh cho Images List

- Bây giờ ta sử dụng đối số imageIndex trong phương thức add item vào listview để liên kết hình ảnh với imageList1:

```
myListView.SmallImageList = imageList1; // Liên kết danh sách hình ảnh nhỏ với
imageList1
```

```
ListViewItem cntt = new ListViewItem("Công Nghệ Thông Tin", 0);
ListViewItem.ListViewSubItem svcntt = new ListViewItem.ListViewSubItem(cntt,
"3.000 sinh viên");
cntt.SubItems.Add(svcntt);
myListView.Items.Add(cntt);

ListViewItem bk = new ListViewItem("Bách Khoa", 1);
ListViewItem.ListViewSubItem svbk = new ListViewItem.ListViewSubItem(bk, "18.00
sinh viên");
bk.SubItems.Add(svbk);
myListView.Items.Add(bk);

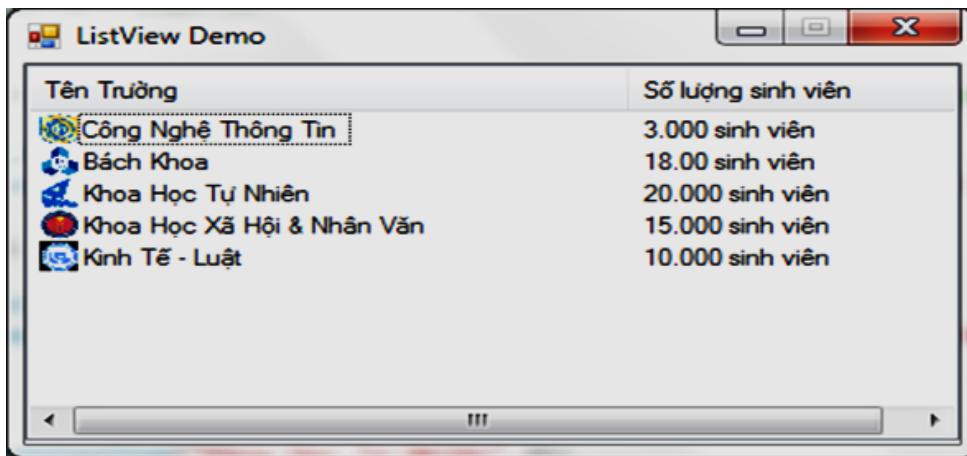
ListViewItem khtn = new ListViewItem("Khoa Học Tự Nhiên",2);
ListViewItem.ListViewSubItem svkhtn = new ListViewItem.ListViewSubItem(khtn,
"20.000 sinh viên");
khtn.SubItems.Add(svkhtn);
myListView.Items.Add(khtn);

ListViewItem nv = new ListViewItem("Khoa Học Xã Hội & Nhân Văn", 3);
ListViewItem.ListViewSubItem svnv = new ListViewItem.ListViewSubItem(nv,
"15.000 sinh viên");
nv.SubItems.Add(svnv);
myListView.Items.Add(nv);

ListViewItem ktl = new ListViewItem("Kinh Tế - Luật", 4);
```

```
ListViewItem.ListViewSubItem svktl = new ListViewItem.ListViewSubItem(ktl,
    "10.000 sinh viên");
ktl.SubItems.Add(svktl);
myListView.Items.Add(ktl);
```

- Lưu ý đối số thứ 2 trong phương thức add item chính là chỉ mục tham chiếu tới hình ảnh trong imageList1.
- Và kết quả mà chúng ta có được:



TÓM TẮT

Trong bài này, học viên làm quen với các Windows Control s:

Property & layout của control

- *Anchor*
- *Docking*

Năm được các thuộc tính và sự kiện thường sử dụng của các controls:

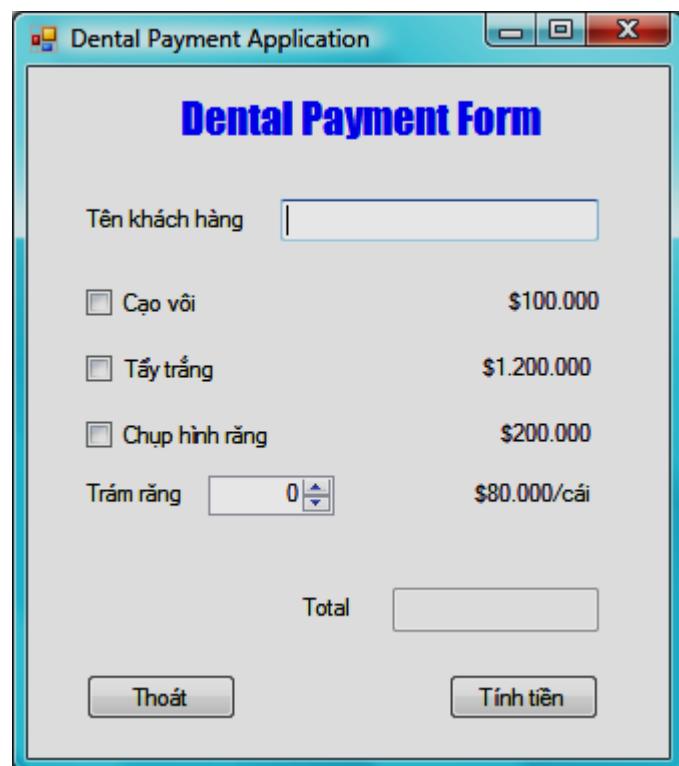
- *Label, Textbox, Button*
- *ListBox, ComboBox*
- *GroupBox, Panel & TabControl*
- *CheckBox, RadioButton*
- *ListView*
- *ImageList*

BÀI TẬP

- Sử dụng Visual Studio .NET 2005 (hoặc 2008) tạo ứng dụng dạng Windows Forms.
- Làm quen với việc sử dụng các control cơ bản trên form chính
 - CheckBox: cho phép user chọn một option
 - Label: hiển thị các thông tin chỉ dẫn
 - Button: cho phép user kích chọn để gọi chức năng cài sẵn
 - TextBox: hộp nhập liệu thông tin
 - NumericUpDown: sử dụng nút up down để nhập giá trị số
 - MessageBox: hiển thị thông tin đến user
- Khai báo trình xử lý sự kiện **Click** cho button trên Form: viết code xử lý cho trình xử lý sự kiện Click.

Nội dung

- Tạo một ứng dụng Windows Form cơ bản tính tiền công dịch vụ cho một lần đi khám tại phòng nha.
- Với mỗi khách hàng, các dịch vụ cung cấp gồm: tẩy răng, cao vôi, chụp hình răng và trám răng. Mỗi loại sẽ có chi phí riêng. Cuối cùng tính tổng các chi phí mà người khách phải trả. Lưu ý: chỉ tính tiền khi phần thông tin tên khách hàng đã được nhập (nếu thông tin này chưa có thì chương trình phát sinh MessageBox cảnh báo).
- Ứng dụng có giao diện đơn giản như hình 1

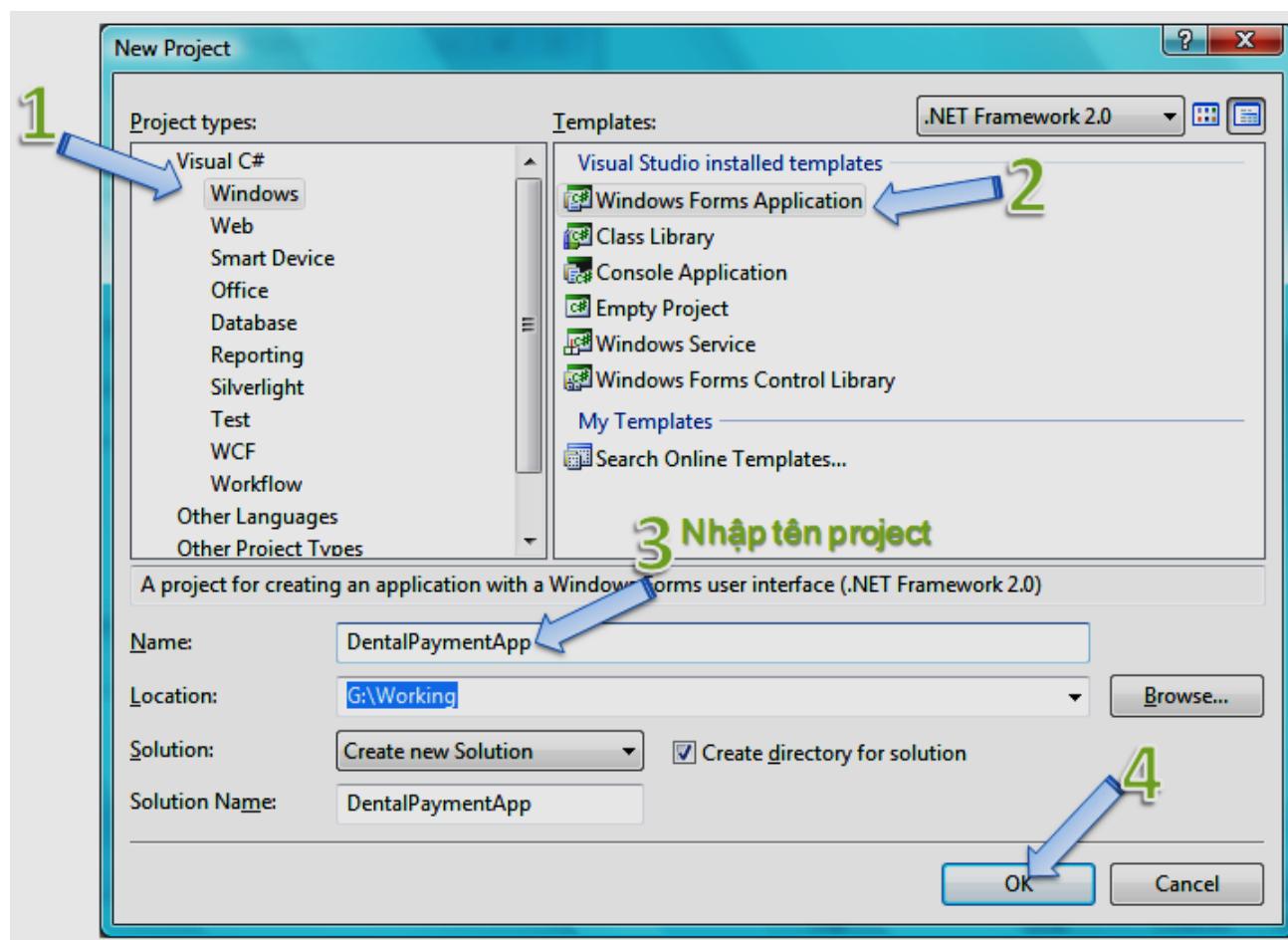


Hình 1: Màn hình chính của ứng dụng

Hướng dẫn

1. Tạo ứng dụng Windows Form có tên **DentalPaymentApp** theo các bước sau
 - a. Tạo project mới (Ctrl+Shift +N),
 - b. Trong cửa sổ new project chọn Visual C# - Windows
 - c. Phần template chọn Windows Forms Application
 - d. Đặt tên project trong phần Name

Xem hình 2 mô tả các bước tạo ứng dụng Windows Form



Hình 2: Tạo ứng dụng Windows Form

2. Sau khi hoàn tất các bước trên VS.NET sẽ phát sinh ra một project Windows Form mẫu, cho phép người lập trình bắt đầu xây dựng các ứng dụng. Giao diện của VS.NET 2005 cho ứng dụng vừa tạo có dạng như hình 3 bên dưới.

Màn hình VS.NET cho ứng dụng Windows Form bao gồm các phần cơ bản

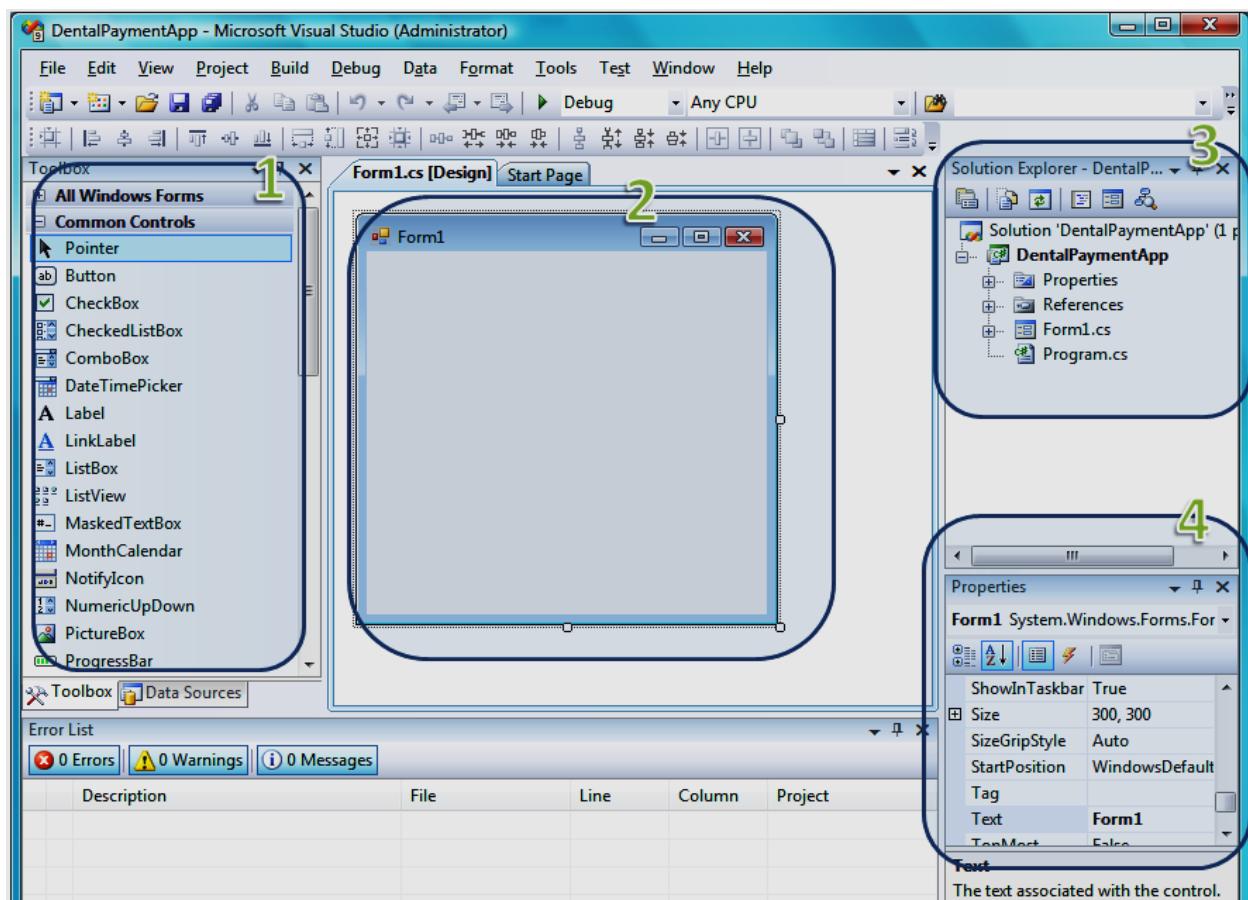
(1): Toolbox: chứa các control cho phép kéo thả vào form

(2): Màn hình thiết kế form, có thể chuyển sang phần code editor...

(3): Cửa sổ Solution Explorer: cho phép người lập trình có thể quản lý các thành phần trong project, hỗ trợ định vị nhanh chóng đến các file mã nguồn.

(4): Cửa sổ property: cho phép user có thể custom lại các thành phần control trên form như: thiết lập các thuộc tính cho control, form, component, cho phép khai báo trình xử lý sự kiện của các control trên form...

(Trước khi viết chương trình, sinh viên nên làm quen với các thành phần trên giao diện này)



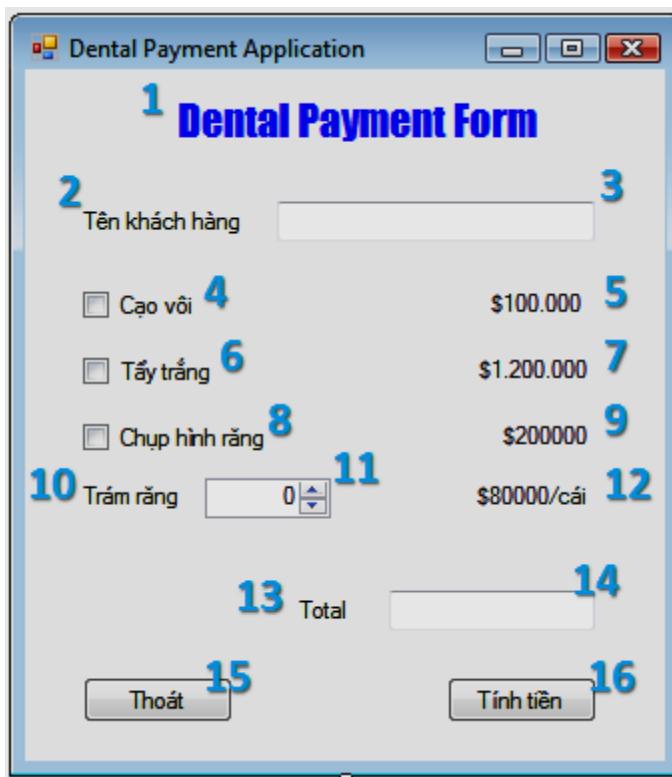
Hình 3: Màn hình VS. NET phục vụ cho việc tạo project Windows Form

3. Thiết kế form theo mô tả như sau

Bảng 1: Mô tả các control trên form

STT	Name	Control	Thiết lập các thuộc tính cho control
-----	------	---------	--------------------------------------

1	lblTitle	Label	Text = "Dental Payment Form", Font = "Impact, Size = 17", ForeColor = Blue
2	lblName	Label	Text = "Tên khách hàng"
3	txtName	TextBox	
4	chkClean	CheckBox	Text = "Cạo vôi"
5	lblCleanCost	Label	Text = "\$100000"
6	chkWhitening	CheckBox	Text = "Tẩy trắng"
7	lblWhiteningCost	Label	Text = "\$1200000"
8	chkXRay	CheckBox	Text = "Chụp hình răng"
9	lblXRayCost	Label	Text = "\$200000"
10	lblFilling	Label	Text = "Trám răng"
11	numFilling	NumericUpDown	
12	lblFillCost	Label	Text = "\$80000"
13	lblTotal	Label	Text = "Total"
14	txtTotal	TextBox	Enable = False
15	btnExit	Button	Text = "Thoát"
16	btnCalc	Button	Text = "Tính tiền"

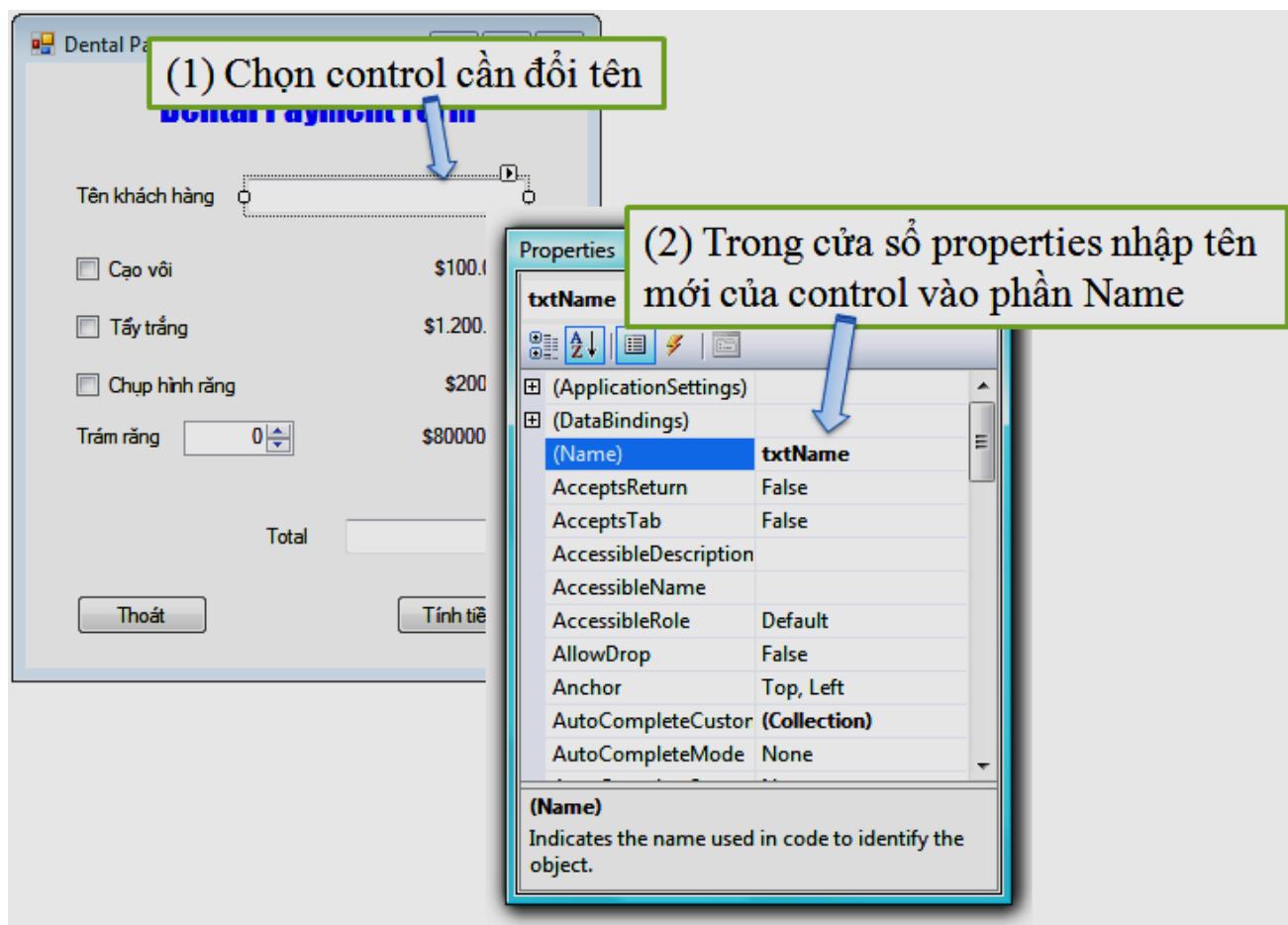


Hình 4: Giao diện của Form chương trình

Cách thực hiện:

- Chọn trong ToolBox control tương ứng rồi kéo thả vào vị trí xác định trên form.
- Sử dụng cửa sổ properties thiết lập các thông tin:
 - Đổi tên của control theo mô tả ở bảng trên

- Thiết lập các giá trị cho control theo mô tả ở bảng 1



Hình 5: Minh họa việc đổi tên của TextBox trên Form

- Tạo trình xử lý sự kiện click cho button "Thoát": chức năng này khi thực hiện sẽ kết thúc ứng dụng (đóng form lại)

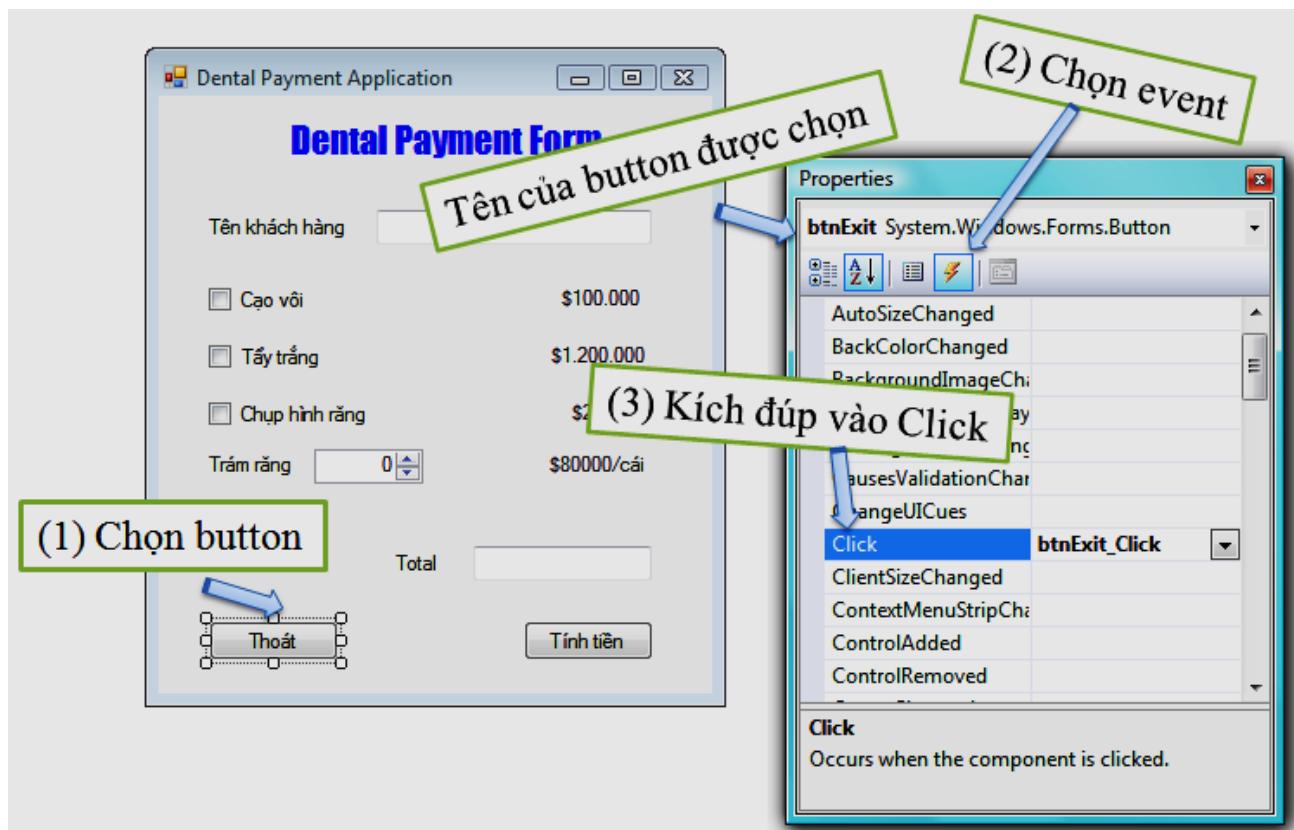
Cách tạo trình xử lý sự kiện: có thể làm theo một trong hai cách

- Cách 1:** Kích đúp vào button cần tạo trình xử lý sự kiện trong màn hình Form design view: khi đó VS sẽ tạo trình xử lý sự kiện gắn với sự kiện Click của button "Thoát"



Hình 6: Minh họa cách kích đúp vào button để tạo event handler

- **Cách 2:** chọn button cần tạo trình xử lý, sau đó kích tab event trong cửa sổ Properties, kích đúp vào mục Click trong cửa sổ event.



Hình 7: Minh họa các bước khai báo event handler từ cửa sổ properties của button

Nội dung của trình xử lý sự kiện Click của button btnExit như sau

```
private void btnExit_Click(object sender, EventArgs e)
{
    // đóng form lại ==> kết thúc ứng dụng
    this.Close();
}
```

5. Tạo chức năng tính tiền, chức năng này được kích hoạt khi button “Tính tiền” được chọn.

Mô tả chức năng **GetPay()** như sau (GetPay() là phương thức thành viên của lớp Form chính:

- Kiểm tra xem tên khách hàng có được nhập hay không?
- Nếu chưa: → xuất thông báo, yêu cầu nhập tên khách.
- Đã nhập: thực hiện các bước sau

Total = 0

If (cạo vôi) Total += 100.000

If (tẩy trắng) Total += 1.200.000

If (chụp hình răng) Total +=200000

Total += (số răng trám)*80000

Xuất số tiền ra TextBox txtTotal

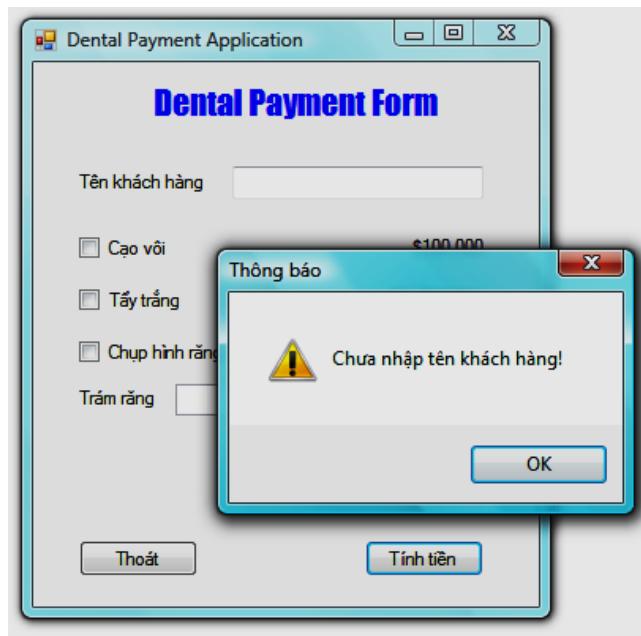
Sinh viên tự viết phương thức GetPay theo mô tả bên trên!

Tạo trình xử lý sự kiện cho button btnCalc rồi trong trình xử lý sự kiện này gọi chức năng GetPay.

```
private void btnCalc_Click(object sender, EventArgs e)
{
    // gọi chức năng tính tiền
    GetPay();
}
```

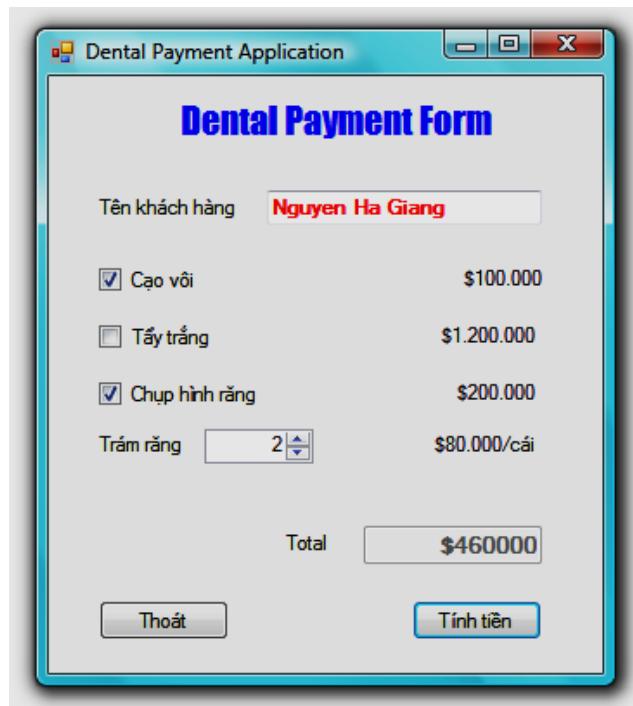
Kết quả chương trình

- Trường hợp không nhập tên khách hàng: phát sinh message box cảnh báo



Hình 8: Không tính tiền khi chưa nhập tên khách hàng

- Màn hình sau khi tính tiền cho khách



Hình 9: Màn hình tính tiền cho khách

Phần mở rộng

1. Bổ sung một ListBox vào form tính tiền, ListBox này dùng để lưu trữ các thông tin tính tiền của khách hàng. Mỗi thông tin tính tiền sẽ được lưu trên một dòng trong ListBox (một item của listbox). Một item gồm hai thông tin: <tên khách hàng> - <tổng số tiền thanh toán>
2. Bổ sung chức năng Lưu: cho phép lưu trữ các thông tin tính tiền của khách hàng trong một file text. File text này có định dạng mỗi dòng là một thông tin tính tiền: gồm tên khách hàng + tổng số tiền.
3. Bổ sung chức năng Đọc file: cho phép load thông tin tính tiền khách hàng từ một file lưu trữ (theo mô tả trong phần 2).
4. Tạo chức năng Tùy chọn: cho phép người tính tiền phòng nha có thể chỉnh lại đơn giá cho từng dịch vụ. Hiện tại ứng dụng trên các đơn giá là fix, ta sẽ cải tiến lại chức năng này. Sinh viên nên tạo file chứa đơn giá từng dịch vụ, tạo form cho phép người quản lý phòng nha có thể hiệu chỉnh lại giá tiền này, lưu lại file đó, và mỗi lần ứng dụng chạy thì đọc file đó để lấy đơn giá.

BÀI 8: ADO.NET

Sau khi học xong bài này, học viên có thể nắm được:

- *ADO.NET*
- *Quá trình phát triển*
- *Đặc điểm ADO.NET*
- *.NET Data Provider*
- *DataSet*
- *Biết được cách thực thi câu lệnh SQL từ ứng dụng*

8.1 TỔNG QUAN

8.1.1 Giới thiệu

Mỗi ứng dụng ít nhiều đều cần lưu trữ dữ liệu. Dữ liệu từ người dùng nhập vào, dữ liệu được lưu trữ trong ứng dụng và dữ liệu từ các hệ thống khác, ... tất cả đều là nguồn thông tin mà ứng dụng cần xử lý với chức năng chính là hỗ trợ tìm kiếm, tính toán, thống kê và ra quyết định.

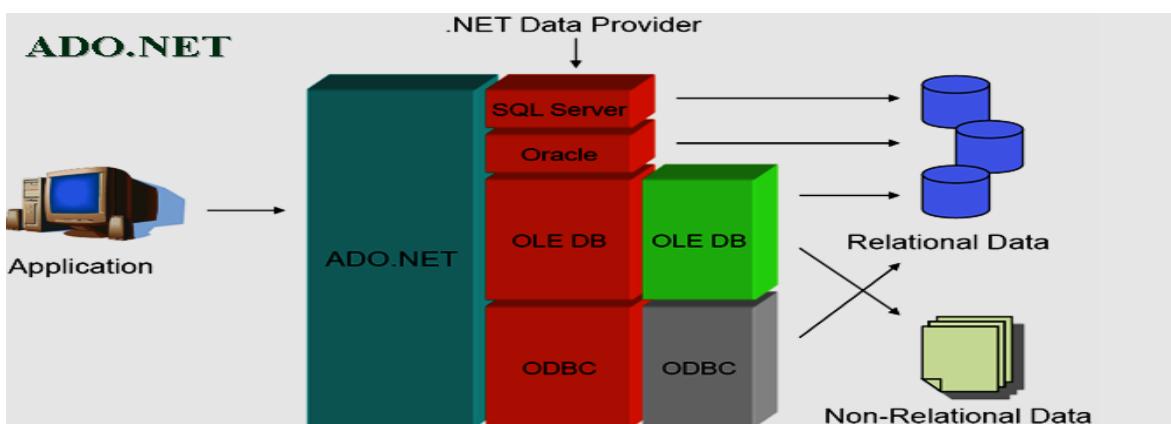
Để thực hiện các chức năng xử lý dữ liệu, người lập trình cần phải có các công cụ lập trình chuyên dụng. Dữ liệu không đơn giản lưu trữ trên các file văn bản hay file nhị phân với cấu trúc record do người lập trình định nghĩa. Thay vào đó hầu hết các ứng dụng tổ chức dữ liệu logic dựa trên cấu trúc cơ sở dữ liệu quan hệ và lưu trữ vật lý dựa vào các hệ quản trị cơ sở dữ liệu quan hệ như Access, SQL Server, Oracle, DB2, ... Khi dữ liệu trở thành trung tâm của ứng dụng thì việc cung cấp các chức năng tới người dùng phụ thuộc vào khả năng xử lý và thao tác với dữ liệu. Vấn đề mà người thiết kế và xây dựng ứng dụng quan tâm khi làm việc với dữ liệu là :

- Lưu trữ dữ liệu tập trung.

- Đảm bảo toàn vẹn dữ liệu.
- Đảm bảo khả năng truy xuất đồng thời của nhiều người dùng trên dữ liệu.
- Đảm bảo thời gian hồi đáp ngắn cho mỗi người dùng.
- Bảo mật dữ liệu.
- Trao đổi dữ liệu giữa các hệ thống khác nhau.

Những vấn đề này được giải quyết nhờ vào khả năng của các hệ quản trị cơ sở dữ liệu (HQT CSDL) và cách phần mềm xử lý dữ liệu do hệ quản trị cơ sở dữ liệu cung cấp. Với các database server sử dụng HQT CSDL như Oracle, SQL Server, ... dữ liệu được đảm bảo lưu trữ tập trung, toàn vẹn và truy xuất đồng thời cũng như bảo mật.

ADO.NET là mô hình mà các ứng dụng .NET sử dụng để có thể làm việc với các hệ quản trị cơ sở dữ liệu cũng như các nguồn dữ liệu khác (data sources). Trước khi .NET xuất hiện thì mô hình được sử dụng để kết nối là ADO (ActiveX Data Object). Với sự ra đời của .NET thì ADO.NET cũng được phát triển để giúp lập trình viên viết ứng dụng làm việc với các nguồn dữ liệu dễ dàng hơn và thống nhất hơn nhờ việc sử dụng chung các kiểu dữ liệu, các mẫu thiết kế (design pattern) và các quy tắc đặt tên.



Hình 8.1: Các thành phần của ADO.NET

Chúng ta biết rằng ADO.NET cho phép tương tác với các loại dữ liệu và kiểu database. Mỗi loại dữ liệu cần một cách thức khác nhau để có thể truy xuất. Các loại dữ liệu cũ sử dụng giao thức ODBC, các loại dữ liệu mới hơn sử dụng giao thức OleDb. Vì vậy cần có một thư viện thống nhất để làm việc với chúng, đây chính là lý do mà ADO.NET được tạo ra.

Đặc điểm chính của ADO.NET là làm việc với dữ liệu không kết nối. Dữ liệu được lưu trữ trong bộ nhớ như một CSDL thu nhỏ gọi là DataSet, nhằm tăng tốc độ tính toán, xử lý tối đa và hạn chế việc sử dụng tài nguyên trên Database Server. Đặc điểm quan trọng thứ hai là khả năng xử lý dữ liệu dạng chuẩn XML. Dữ liệu ở dạng XML có thể trao đổi giữa bất kỳ hệ thống nào nên ứng dụng của bạn sẽ có nhiều khả năng làm việc với nhiều ứng dụng khác.

8.1.2 Các đặc điểm của ADO.NET

8.1.2.1 Interoperability–Tương tác giữa nhiều hệ thống khác nhau

Trong các hệ thống phân tán, dữ liệu được chuyển ở dạng disconnected recordset giữa Data provider và Data consumer. Vì ở dạng RecordSet, cả provider và consumer cần phải sử dụng COM để trao đổi dữ liệu dẫn đến việc khó mở rộng hệ thống vì COM không làm việc qua Firewall và Network Address Translator (NAT).

ADO.NET giải quyết vấn đề này bằng cách thay đổi về bản chất của việc đóng gói dữ liệu trước khi truyền trên mạng.

- Với ADO, disconnected RecordSet được đóng gói ở dạng Network Data Representation (NDR) trước khi truyền trên mạng. NDR vẫn là dạng đối tượng mà ở tầng Application, data provider và data consumer phải sử dụng kỹ thuật COM để xử lý.
- ADO.NET thay thế NDR bằng định nghĩa mới : XML. Với bản chất Text, XML hoàn toàn có thể sử dụng HTTP để trao đổi dữ liệu. Hơn nữa XML document dễ dàng được xử lý mà không cần đến kỹ thuật COM. XML là chuẩn để trao đổi dữ liệu mới nhất và đang được hỗ trợ rất rộng rãi.

8.1.2.2 IScalability – Hỗ trợ nhiều người dùng

ADO.NET sử dụng dữ liệu ở dạng disconnected data :

- Client tạo kết nối đến Server để lấy dữ liệu.
- Server gửi dữ liệu về cho Client.
- Client ngắt kết nối với Server.

- Khi cần cập nhật dữ liệu, kết nối giữa Client và Server được phục hồi.

Với cơ chế disconnected data, thời gian kết nối giữa Client và Server không còn lâu dài như trước, Server có khả năng phục vụ nhiều Client hơn hẳn so với cơ chế của ADO trước đây.

Cơ chế disconnected data có 2 điểm đáng chú ý :

1. Dữ liệu sẽ không phản ánh kịp thời những thay đổi dữ liệu trong một hệ thống Client/Server với nhiều người dùng.
 - Trên thực tế, người dùng khi làm việc với các hệ thống phân tán hiểu và chấp nhận dữ liệu có một độ trễ và sai lệch nhỏ, không đáng kể (tương tự như các ứng dụng Web).
 - ADO trước đây nếu sử dụng RecordSet ở dạng OpenStatic thì cũng không tốt hơn cơ chế disconnected data là bao.
2. Thời gian cho một lần tạo kết nối giữa Client và Server là khá lớn, việc ngắt kết nối sau đó tạo lại kết nối để cập nhật dữ liệu sẽ chậm.
 - Vấn đề này được ADO.NET giải quyết khá đơn giản bằng cơ chế connection pooling, thông tin về mỗi kết nối vẫn còn được lưu giữ ở Server trong khoảng thời gian nhất định và khi Server còn đủ tài nguyên, mỗi khi Client cần kết nối lại với Server, thông tin kết nối được lấy ra từ connection pooling nên tốc độ không bị ảnh hưởng

8.1.2.3 Productivity – Mở rộng khả năng làm việc với CSDL

ADO.NET vẫn giữ kiến trúc các đối tượng độc lập và về mặt ý nghĩa nói chung, các đối tượng trong ADO.NET và ADO cũng gần giống nhau, vẫn có Connection, Command và đối tượng đại diện cho dữ liệu cần xử lý. Điểm mạnh hơn của ADO.NET là DataSet tích hợp vào nó nhiều chức năng hơn hẳn những gì mà RecordSet làm được.

Với ADO.NET, có sự tập trung rõ ràng hơn về các chức năng của các đối tượng. Ví dụ, Connection không còn lo việc thi hành một câu lệnh SQL hay một Stored Procedure nữa, công việc này được giao hoàn toàn cho Command đảm nhận.

Đáng chú ý hơn, ADO.NET phát triển trên .NET là component có thể kế thừa được. Người lập trình hoàn toàn có khả năng kế thừa các đối tượng của ADO.NET để xây dựng các đối tượng mới tốt hơn và phù hợp với nhu cầu phát triển ứng dụng của mình.

Cuối cùng, Visual Studio .NET với Component Designer giúp người lập trình tạo ra các DataSet nhanh chóng với nội dung lệnh rõ ràng hơn những gì Data Form Wizard của Visual Studio 6 trước đây tạo ra.

8.1.2.4 Performance – Hiệu quả cao trong xử lý dữ liệu

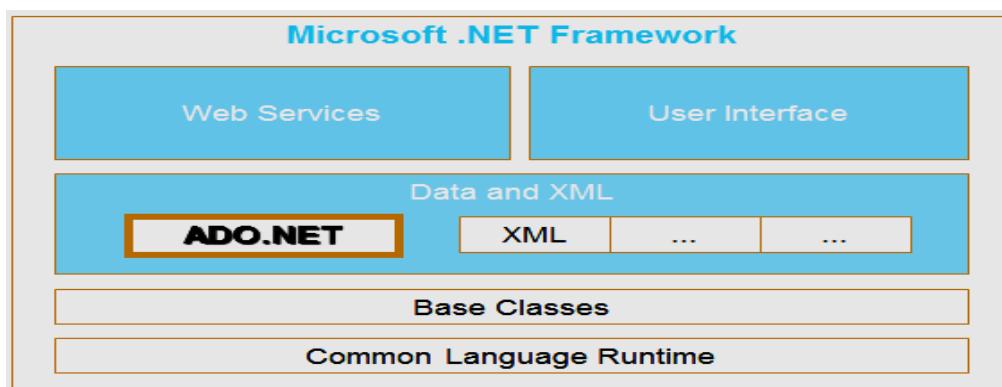
Với các thay đổi trong bản chất cơ chế thực hiện, hiệu quả của ứng dụng dùng ADO.NET sẽ cao hơn nhiều.

- Disconnected data đảm bảo ứng dụng trên Client ít gặp lỗi khi xử lý dữ liệu đồng thời giúp mở rộng ứng dụng tới nhiều người dùng hơn. Các managed provider component hỗ trợ tính năng thiết lập connection pooling ngầm định để giảm bớt thời gian tái kết nối.
- Định dạng XML thay thế cho NDR trong trao đổi dữ liệu giữa Client và Server giúp cho dữ liệu có thể được xử lý bởi nhiều client hơn dù ứng dụng trên Client có hay không sử dụng COM. XML cũng giúp việc truyền thông nhanh hơn vì không cần đóng gói phức tạp như NDR hay chuyển qua chuyển lại giữa dữ liệu thô sang dữ liệu ở dạng đối tượng.

8.2 CÁC ĐỐI TƯỢNG ADO.NET

8.2.1 Kiến trúc ADO.NET

ADO.NET là một bộ các thư viện hướng đối tượng (OOP) cho phép bạn tương tác với dữ liệu nguồn. Thông thường thì dữ liệu nguồn là một cơ sở dữ liệu (database), nhưng nó cũng có thể là file text, excel hoặc XML. Theo những mục tiêu của hướng dẫn này, chúng ta sẽ chỉ xem xét tới cách ADO.NET làm việc với database.



Hình 8.2: Kiến trúc .Net Framework

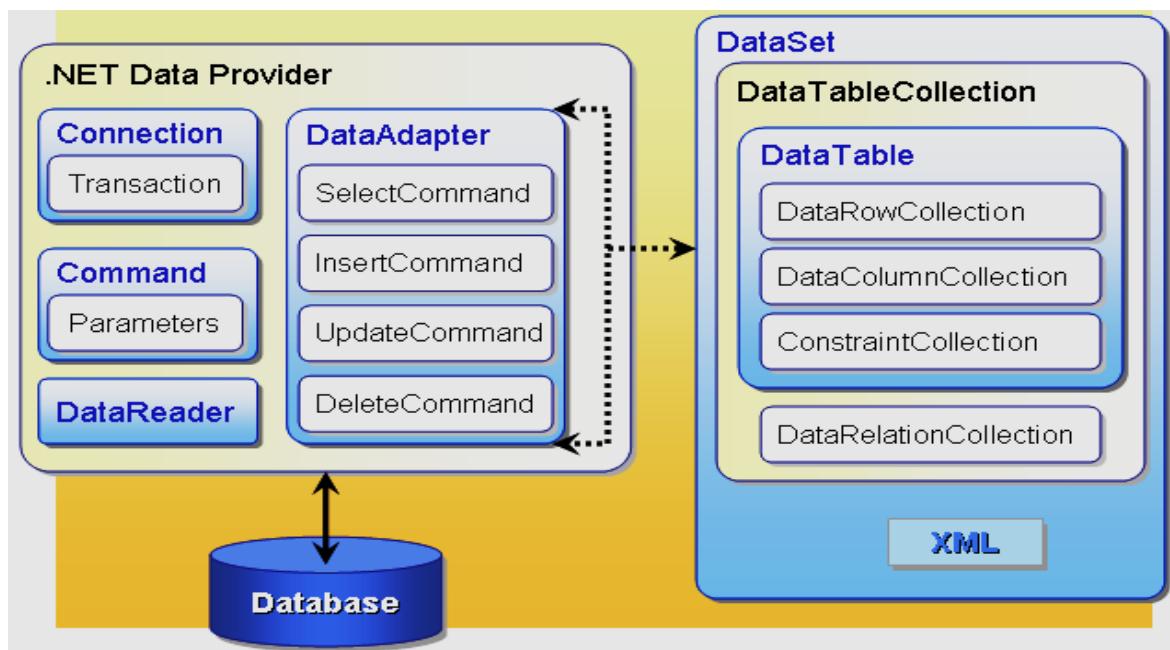
Như bạn có thể biết rằng, có rất nhiều loại database hiện nay như Microsoft SQL Server, Microsoft Access, Oracle, Borland Interbase, và IBM DB2,... Để làm rõ hơn phạm vi của loạt bài này, tất cả ví dụ sẽ sử dụng SQL Server.

ADO.NET cung cấp một cách thức chung để tương tác với nguồn dữ liệu, nhưng với mỗi loại dữ liệu bạn phải sử dụng một thư viện khác nhau. Các thư viện này được gọi là Data Provider và thường được đặt tên theo giao thức hoặc loại dữ liệu mà chúng cho phép bạn truy xuất. Table 1 liệt kê các data provider phổ biến, tiền tố (prefix) API mà chúng sử dụng và kiểu dữ liệu mà bạn có thể làm việc.

Provider name	Prefix	namespace	Mô tả
ODBC Data Provider	Odbc	System.Data.Odbc	
OleDb Data Provider	OleDb	System.Data.OleDb	
Oracle Data Provider	Oracle	System.Data.OracleClient	
SQL Data Provider	Sql	System.Data.SqlClient	

Bảng mô tả các thư viện truy vấn dữ liệu của ADO.Net thông qua các loại provider khác nhau. Các thư viện trên về giao tiếp lập trình là giống nhau:

- Dùng thư viện SqlClient truy xuất SQL Server nhanh hơn OleDb
- Tương tự cho OracleClient



Hình 8.3: Mô hình đối tượng ADO.NET

Kiến trúc ADO.NET có thể chia làm hai phần chính :

- Phần kết nối (Managed Provider Component) : bao gồm các đối tượng như DataAdapter, DataReader, ... giữ nhiệm vụ làm việc trực tiếp với dữ liệu như database, file, ...:
 - Connection: quản lý việc đóng mở DB
???Connection: SqlConnection, OleDbConnection
 - Command: lệnh truy vấn, tương tác dữ liệu khi đang lập kết nối
???Command: SqlCommand, OleDbCommand
 - DataReader: đọc dữ liệu, chỉ xử lý 1 dòng dữ liệu tại một thời điểm
???DataReader: SqlDataReader, OleDbDataReader
 - DataAdapter: cầu nối giữa DB và DataSet
- Phần ngắt kết nối (Content Component): bao gồm các đối tượng như DataSet, DataTable, ... đại diện cho dữ liệu thực sự cần làm việc.
 - DataReader là đối tượng giúp truy cập dữ liệu nhanh chóng nhưng forward-only và read-only.
 - DataSet có thể coi là một bản sao gọn nhẹ của CSDL trong bộ nhớ với nhiều bảng và các mối quan hệ.

DataAdapter là đối tượng kết nối giữa DataSet và CSDL, nó bao gồm hai đối tượng Connection và Command để cung cấp dữ liệu cho DataSet cũng như cập nhật dữ liệu từ DataSet xuống CSDL.

8.2.2 Các đối tượng ADO.NET

8.2.2.1 Connection



Hình 8.4: Tương tác giữa ứng dụng và cơ sở dữ liệu thông qua kết nối

Để tương tác với database, bạn phải có một kết nối tới nó. Kết nối giúp xác định database server, database name, user name, password, và các tham số cần thiết để kết nối tới database. Một đối tượng connection được dùng bởi đối tượng command vì thế chúng sẽ biết database nào để thực thi lệnh.

Thuộc tính và phương thức:

- ConnectionString: chuỗi kết nối đến nguồn dữ liệu (datasource)

Database	ODBC/OLE DB Connection (*)
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; Data Source=ServerName; Initial Catalog=DatabaseName; UserId=Username; Password=Password

Hình 8.5: Cách thiết lập ConnectionString

- Open(): thiết lập kết nối đến datasource
- Close(): đóng kết nối đến datasource

```
using System;
using System.Data;
using System.Data.SqlClient;
using System.Configuration;
using System.Collections.Generic;

public class DBAccess
{
    public string cnnString = ConfigurationManager.ConnectionStrings["ConnectionString"].ConnectionString;

    public DBAccess()
    {}

    public DBAccess(string connection)
    {
        cnnString = connection;
    }

    public DataTable GetDataTable(string query)
    {
        DataTable myData = new DataTable();
```

```

using (SqlConnection cnn = new SqlConnection(cnnString))
{
    cnn.Open();
    SqlCommand myCommand = new SqlCommand(query, cnn);
    SqlDataAdapter myAdapter = new SqlDataAdapter(myCommand);

    myAdapter.Fill(myData);
}
return myData;
}

public string Single(string query)
{
    string ret = "";
    using (SqlConnection cnn = new SqlConnection(cnnString))
    {
        SqlDataAdapter myAdapter = new SqlDataAdapter(query, cnn);
        DataTable myData = new DataTable();
        myAdapter.Fill(myData);
        if (myData.Rows.Count > 0)
        {
            DataRow myRow = myData.Rows[0];
            ret = myRow[0].ToString();
        }
    }
    return ret;
}

public bool Execute(string query)
{
    bool success = false;

    using (SqlConnection cnn = new SqlConnection(cnnString))
    {
        cnn.Open();

        SqlCommand myCommand = new SqlCommand(query, cnn);
        success = myCommand.ExecuteNonQuery() > 0;
    }
    return success;
}

```

8.2.2.2 Command



Quá trình tương tác với database cần phải biết hành động nào bạn muốn xảy ra. Điều này được thực hiện bởi đối tượng command. Bạn dùng đối tượng command để gửi một câu lệnh SQL tới database. Một đối tượng command dùng một đối tượng

connection để xác định database nào sẽ được truy xuất. Bạn có thể dùng một đối tượng command riêng lẻ để thực thi lệnh trực tiếp, hoặc để gắn một tham chiếu của đối tượng command cho một SqlDataAdapter – đối tượng giữ các command sẽ làm việc trên một nhóm dữ liệu như sẽ đề cập tới trong phần dưới.

Thuộc tính và phương thức:

- Connection: kết nối để thực hiện lệnh
- CommandText: câu lệnh cần thực hiện
- CommandType: loại câu lệnh (Text,TableDirect, StoredProcedure)
- ExecuteScalar(): thực hiện câu lệnh và trả về giá trị đơn
- ExecuteNonQuery(): gọi các lệnh SQL, store, trả về số row bị tác động (Insert, Update, Delete...)
- ExecuteReader(): thực hiện lệnh và trả về DataReader

```
// ví dụ đếm số dòng trong bảng Customer
public int GetNumberOfCustomers()
{
    SqlConnection cnn = new SqlConnection();
    cnn.ConnectionString="server=.\SQLEXPRESS;database=Northwind;
    Trusted_connection=true";
    SqlCommand cmd = new SqlCommand();
    cmd.CommandText = "Select COUNT(*) From Customer";
    cmd.Connection = cnn;
    cnn.Open();
    int count = (int) cmd.ExecuteScalar();
    cnn.Close();
    return count;
}
```

Tham số hóa câu lệnh

Một số câu lệnh như tìm kiếm sinh viên có mã do người dùng nhập vào lúc chương trình đang chạy, hay cập nhật thông tin của một sinh viên. Người lập trình phải tạo câu query với parameter, tương tự như vậy ADO.Net cũng cho phép truyền dữ liệu từ chương trình cho các câu lệnh như thế thông qua SqlParameter

```

public bool InsertStudent(int id, string name, DateTime birthdate, int sex,
string address, int phone)
{
    int count = 0;
    SqlConnection conn = new SqlConnection();
    conn.ConnectionString= "server=Server10;database=Test;integrated security=true";

    string query = "INSERT INTO Sinhvien VALUES (@MS, @HT, @NS, @GT, @DC, @DT)";
    SqlCommand cmd = new SqlCommand(query, conn);
    // tao instant cho param
    SqlParameter param = new SqlParameter("@MS", SqlDbType.Int);
    param.Value = id;
    cmd.Parameters.Add(param);
    cmd.Parameters.Add(new SqlParameter("@DT", phone));
    //add truc tiep
    cmd.Parameters.Add("@HT", SqlDbType.NVarChar);
    cmd.Parameters.Add("@NS", SqlDbType.DateTime);
    cmd.Parameters.Add("@GT", SqlDbType.Bit);
    cmd.Parameters.Add("@DC", SqlDbType.NVarChar);
    // gan gia tri cho param
    cmd.Parameters["@MS"].Value = id;
    cmd.Parameters["@HT"].Value = name;
    cmd.Parameters["@NS"].Value = birthdate;
    cmd.Parameters["@GT"].Value = sex;
    cmd.Parameters["@DC"].Value = address;
    cmd.Parameters["@DT"].Value = 5120791;
    try
    {
        conn.Open();
        count = (int)cmd.ExecuteNonQuery();
    }
    catch (SqlException ex)
    {
        //
    }
    finally {
        if (conn.State == ConnectionState.Open)
        {
            conn.Close();
        }
    }
    return count > 0;
}

```

8.2.2.3 DataReader

Nhiều thao tác dữ liệu đòi hỏi bạn chỉ lấy một luồng dữ liệu để đọc. Đối tượng data reader cho phép bạn lấy được kết quả của một câu lệnh SELECT từ một đối tượng command. Để tăng hiệu suất, dữ liệu trả về từ một data reader là một luồng dữ liệu fast forward-only. Có nghĩa là bạn chỉ có thể lấy dữ liệu từ luồng theo một thứ tự nhất định. Các tính chất của DataReader:

- Truy xuất tuần tự

- Cơ chế kết nối



Thuộc tính và phương thức:

- HasRow: cho biết câu truy vấn có trả về dữ liệu
- Read(): đọc một mẫu tin
- [i]: truy xuất đến cột i của mẫu tin được đọc
- Close(): đóng

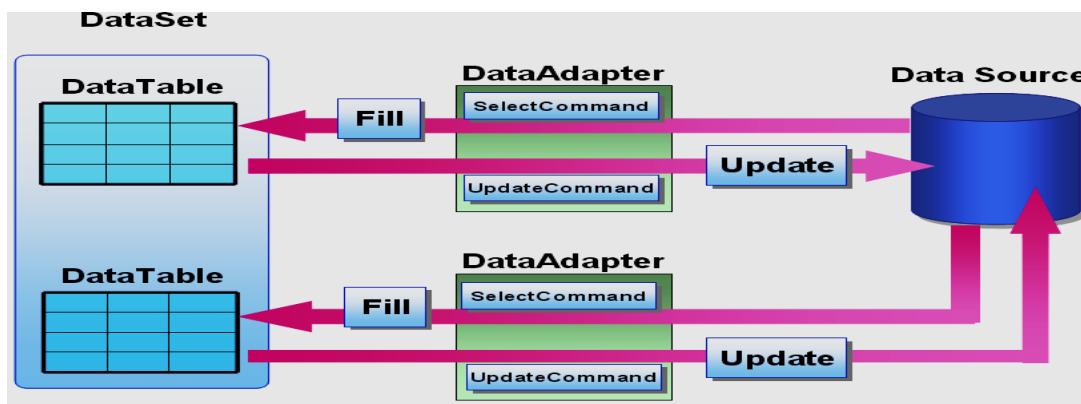
Mặc dù điều này có lợi về mặt tốc độ, nhưng nếu bạn cần phải thao tác dữ liệu, thì một DataSet sẽ là một đối tượng tốt hơn để làm việc.

```
SqlCommand cmd = new SqlCommand("Select * From Sinhvien", conn);
conn.Open();
SqlDataReader reader = cmd.ExecuteReader();
while (reader.Read()) {
    listBox1.Items.Add(reader["Hoten"]);
}
reader.Close();
conn.Close();
```

8.2.2.4 DataAdapter

Đôi lúc dữ liệu mà bạn làm việc là read-only và bạn ít khi cần thay đổi dữ liệu nguồn. Vài trường hợp cần lưu trữ tạm dữ liệu trong bộ nhớ để hạn chế truy xuất đến database. Data adapter làm điều này dễ dàng bằng cách giúp bạn quản lý dữ liệu trong chế độ ngắt kết nối. Data adapter sẽ đổ vào DataSet khi đọc dữ liệu và thực hiện thay đổi dữ liệu một lượt vào database.

Data adapter chứa một tham chiếu đến đối tượng connection và mở/đóng kết nối tự động khi đọc và ghi dữ liệu vào database. Hơn nữa, data adapter chứa đối tượng command cho những thao tác SELECT, INSERT, UPDATE và DELETE trên dữ liệu. Bạn sẽ có một data adapter được định nghĩa cho mỗi table trong một DataSet và nó sẽ quản lý các giao tiếp với database cho bạn. Tất cả những gì bạn cần làm là chỉ cho data adapter khi nào nạp hoặc ghi vào database



Hình 8.6: Thao tác dữ liệu với DataAdapter và DataSet

Thuộc tính & Phương thức

- Fill(DataSet): sử dụng SelectCommand lấy dữ liệu từ Data Source đổ vào Data Set
- Update(DataSet): InsertCommand, UpdateCommand, DeleteCommand cập nhật dữ liệu trong DataSet vào DataSource

```

public void Test()
{
    string strConn = "Server=.;Database=StudentDB; Trusted_Connection=true";
    string query = "Select * From Sinhvien";
    SqlDataAdapter adapter = new SqlDataAdapter(query, strConn);
    DataSet ds = new DataSet();
    adapter.Fill(ds);
    // thao tác trên dataset
    //...
    adapter.Update(ds);
}
    
```

8.2.2.5 DataSet

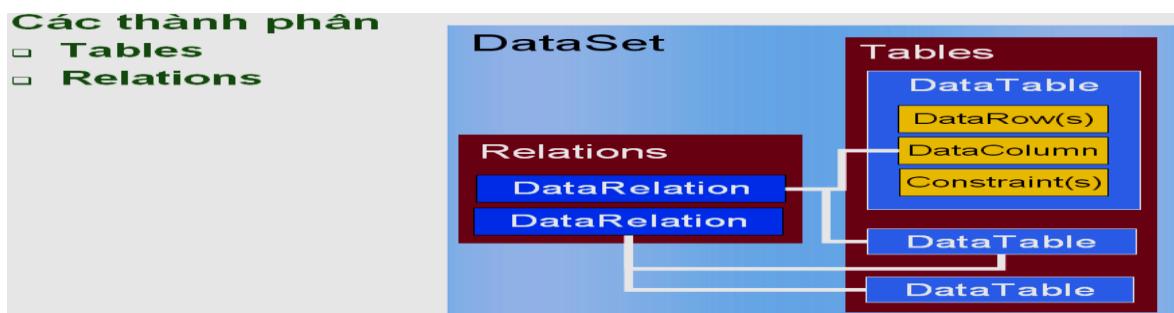
Đối tượng DataSet là đối tượng ngắt kết nối, là một thể hiện của dữ liệu trong bộ nhớ. Chúng chứa nhiều đối tượng DataTable, bên trong DataTable lại có nhiều column

và row, giống như các database table thông thường. Bạn thậm chí có thể định nghĩa dữ liệu giữa các table để tạo các quan hệ parent-child. DataSet được thiết kế đặc biệt để giúp quản lý dữ liệu trong bộ nhớ và để hỗ trợ các thao tác không cần kết nối (disconnected) trên dữ liệu. DataSet là một đối tượng được dùng bởi tất cả Data Provider, đó là lý do tại sao nó không có một Data Provider prefix trong tên gọi.

DataSet hoạt động theo cơ chế không kết nối, nhờ đối tượng DataAdapter làm trung gian.

Hỗ trợ đầy đủ đặc tính XML

Thao tác được với tất cả mô hình lưu trữ hiện tại: flat, relational, hierarchical



Hình 8.7: Các thành phần của DataSet

DataTable: thể hiện một bảng trong CSDL

- TableName: tên của bảng dữ liệu
- Columns: danh sách các cột
- Rows: danh sách các mẫu tin
- PrimaryKey: danh sách các cột là khóa chính
- NewRow(): tạo một mẫu tin mới

DataColumn: đại diện cho một cột trong bảng

- ColumnName: tên cột
- DataType: kiểu dữ liệu

DataRow: đại diện cho mẫu tin trong bảng

- RowState: trạng thái Added, Modified, Deleted,...
- [i]: truy xuất đến cột i

- Delete(): đánh dấu xóa mẫu tin

Sử dụng SqlCommandBuilder, tham chiếu đến SqlDataAdapter để hỗ trợ cho việc cập nhật dữ liệu từ DataSet vào data source, nếu SqlDataAdapter không cài đặt đầy đủ InsertCommand/UpdateCommand/DeleteCommand

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data.SqlClient;
using System.Data;

namespace WindowsFormsApplication1
{
    public class Demo
    {
        string strConn = Properties.Settings.Default.ConnectionString;
        string strCmd = "SELECT * FROM Student";

        public void UpdateRow()
        {
            SqlDataAdapter da = new SqlDataAdapter(strCmd, strConn);
            SqlCommandBuilder builder = new SqlCommandBuilder(da);
            DataSet ds = new DataSet();
            da.Fill(ds);
            foreach (DataRow dr in ds.Tables[0].Rows)
                dr["Ngaysinh"] = DateTime.Now;

            da.Update(ds);
        }

        public void DeleteRow()
        {
            SqlDataAdapter da = new SqlDataAdapter(strCmd, strConn);
            SqlCommandBuilder builder = new SqlCommandBuilder(da);
            DataSet ds = new DataSet();
            da.Fill(ds);
            DataTable table = ds.Tables[0];
            DataRow[] rows = table.Select("Ngaysinh<'1/1/1980'");
            foreach (DataRow r in rows)
                r.Delete();
            da.Update(ds);
        }
    }
}
```

TÓM TẮT

Trong bài này, học viên làm quen với lập trình cơ sở dữ liệu với ADO.NET

ADO.NET được thiết kế với các tính năng: kiến trúc dữ liệu không kết nối (disconnected data architecture), tích hợp với XML, và được tối ưu để kết nối với cơ sở dữ liệu (và các nguồn dữ liệu khác).

ADO.NET là một kĩ thuật .NET để thao tác với nguồn dữ liệu.

- *Bạn có một vài Data Provider, cho phép bạn giao tiếp với các nguồn dữ liệu khác nhau, dựa trên giao thức mà chúng dùng hoặc kiểu database. Không cần quan tâm đến điều này, với mỗi Data Provider được sử dụng, bạn sẽ dùng các đối tượng tương tự nhau để thao tác với dữ liệu.*
- *Đối tượng SqlConnection cho phép bạn quản lý một kết nối đến nguồn dữ liệu. SqlCommand cho phép bạn gửi lệnh đến dữ liệu.*
- *Để đọc dữ liệu nhanh theo cơ chế forward-only, sử dụng SqlDataReader.*
- *Nếu bạn muốn làm việc với dữ liệu đã ngắt kết nối, dùng một DataSet và hiện thực việc đọc và ghi đến dữ liệu nguồn bằng một SqlDataAdapter.*

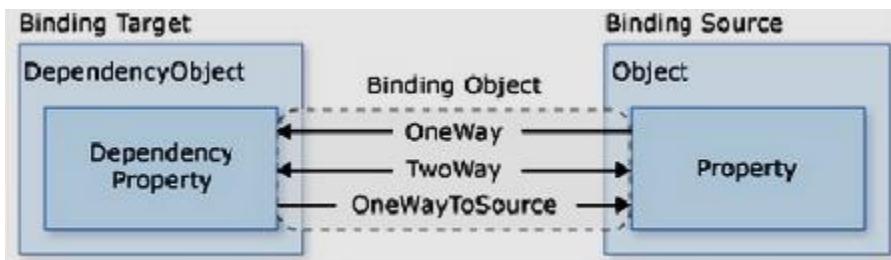
BÀI 9: DATA BINDING

Sau khi học xong bài này, học viên có thể nắm được:

- Ôn lại kiến thức về Windows Forms
- Biết được cách tạo kết nối và truy xuất dữ liệu từ SQL Server
- Hiểu cơ bản về Data Binding
- Data Binding với TextBox, ComboBox, ListBox, DataGridView

9.1 GIỚI THIỆU

Data binding là kỹ thuật tạo ra cầu nối giữa các thành phần giao diện chương trình (GUI) và các nguồn dữ liệu (data source)



Hình 9.1: Binding dữ liệu

Ví dụ:

- Giao diện: TextBox, ListBox, ComboBox, ...
- Nguồn dữ liệu: sql data source, list objects, linq to sql, ...
- Có thể bind một cột (col) vào một TextBox qua thuộc tính Text hoặc có thể bind cả một table vào DataGrid như DataGridView.

9.2 CÁC KỸ THUẬT

Có 2 cách binding WinForm control vào dữ liệu :

- Simple Data Binding

- Complex Data Binding

9.2.1 Simple Data Binding

Là cách liên kết một-một giữa một thuộc tính của control và một thành phần của data source, và sử dụng control để hiển thị duy nhất một giá trị một lần.

```
txtCustomer.DataBindings.Add("Text", ds.Tables[0], "CustomerID");
```

Một control chỉ dc binding 1 lần, muốn binding lại thì phải clear binding cũ.

- Các checkbox khi binding thì phải thay chữ Text = Checked

```
cbGender.DataBindings.Add("Checked", ds.Tables[0], "Gender");
```

Ví dụ :

- Tạo Winform App project, tại Form1 bạn cho thêm 2 textbox vào.
- Sau đó trong sự kiện : Form1_Load bạn chèn thêm đoạn code sau:
- Sau đó run thì thấy FirstnameTextBox có giá trị firstname đầu tiên trong record và LastnameTextBox là giá trị lastname tương ứng.

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Data.SqlClient;

namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            string sql = @"SELECT * FROM Employee ";
            SqlConnection conn = new
SqlConnection(Properties.Settings.Default.ConnectionString);

            SqlDataAdapter da = new SqlDataAdapter(sql, conn);
            DataSet ds = new DataSet();
            da.Fill(ds, "Employee");

            FirstnameTextBox.DataBindings.Add("Text", ds, "Employee.FirstName");
            LastnameTextBox.DataBindings.Add("Text", ds.Tables["Employee"], "LastName");
        }
    }
}
```

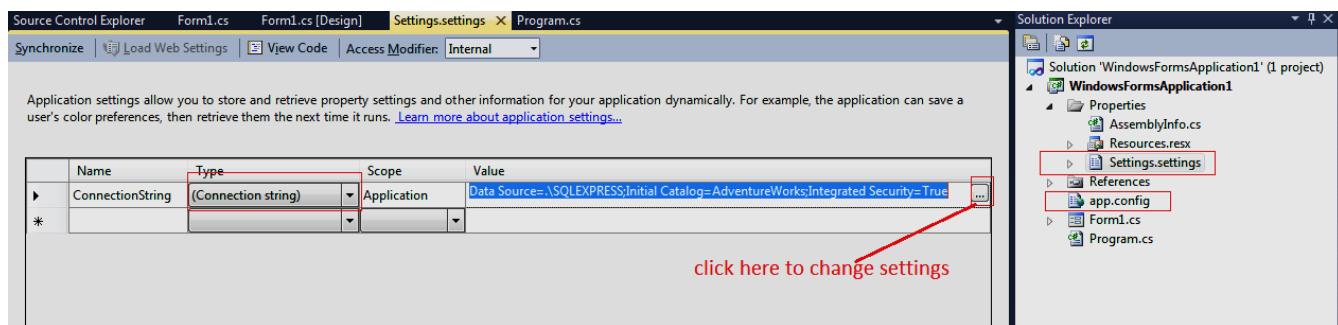
```

    }
}

}

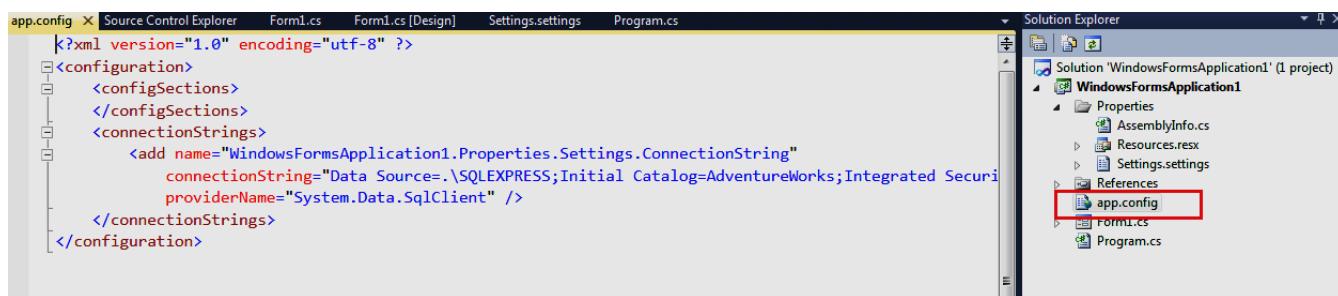
```

Chú ý: khi tạo kết nối cơ sở dữ liệu, các bạn không nên tạo chuỗi kết nối ConnectionString fix trong code như những ví dụ trước, vì như thế khi các bạn cài đặt ứng dụng ở môi trường khác, không giống với môi trường gốc khi phát triển ứng dụng thì ứng dụng có thể không hoạt động vì không tìm thấy nguồn dữ liệu.



Hình 9.2: Thiết lập chuỗi kết nối cho toàn ứng dụng

Để cải thiện nhược điểm này, Visual Studio cho phép lưu cấu hình của ứng dụng trong phần Settings, khi cài đặt ứng dụng bạn chỉ cần sửa lại file app.config (XML) mà không cần sử code và build lại toàn bộ ứng dụng



Hình 9.3: App.config

9.2.2 Complex Data Binding

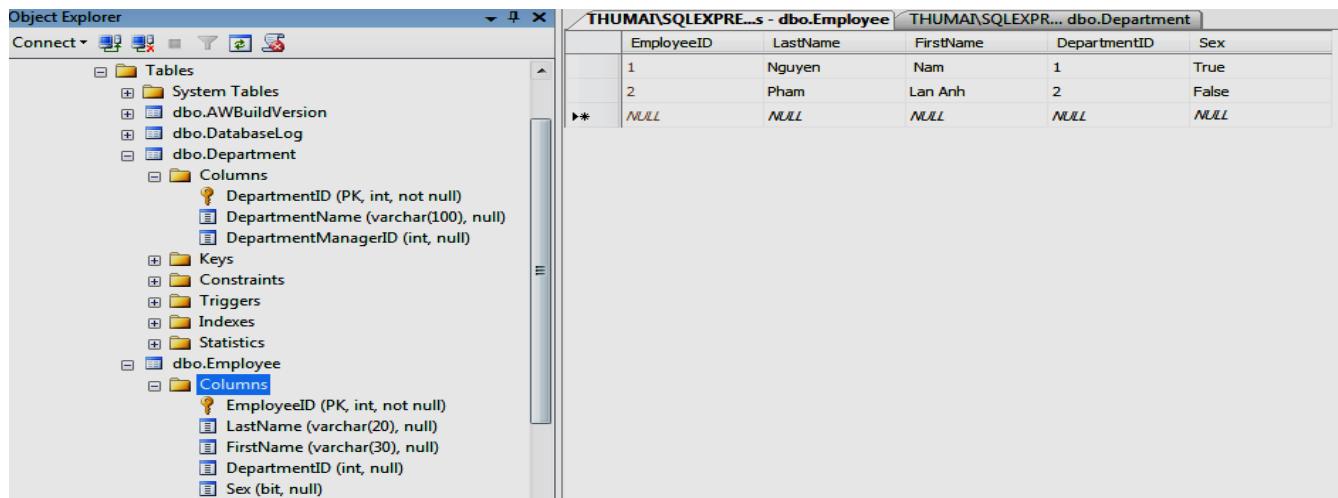
Là liên kết một control với một hoặc nhiều thành phần data của data source, có thể hiển thị nhiều hơn 1 giá trị một lần.

Ta phải sử dụng lớp **BindingManagerBase** để hỗ trợ cho việc liên kết các binding của cùng một record.

Thuộc tính **BindingManagerBase.Position**: cho phép dịch chuyển vị trí giữa các record dữ liệu

Xét ví dụ xây dựng Form hiển thị danh sách nhân viên với DataGridView, và sử dụng các nút di chuyển để duyệt thông tin chi tiết từ đầu đến cuối danh sách:

- Tạo một WinForm App project.
- Viết code để lấy dữ liệu từ hai bảng Employee, Department của database AdventureWorks



Hình 9.4: Cơ sở dữ liệu của ứng dụng

- Thêm vào một DataGridView để hiển thị danh sách nhân viên, 2 textbox, và 2 button
- Khai báo sử dụng đối tượng BindingManager Class
- Binding dữ liệu cho dataGridView có thể viết code để binding Datasource với danh sách nhân viên (ds.Tables["Employee"]), hoặc sử dụng wizard (sau đó hiện ra DataGridView Task chọn Choose Data Source -> Add Project -> chọn Database -> New Connection -> SQL Server, AdventureWorks database -> Next đến khi nào thấy mục chọn Table cho DataSet thì chọn table : Employee rồi Finish)
- Binding dữ liệu cho các control còn lại như đoạn code bên dưới.
- Build -> Ctrl + F5 xem kết quả thu được

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Drawing;
```

```

using System.Text;
using System.Windows.Forms;
using System.Data.SqlClient;

namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        // Tạo BindingManager
        private BindingManagerBase bMgr;
        SqlConnection conn = new SqlConnection(Properties.Settings.Default.ConnectionString);
        SqlDataAdapter da = new SqlDataAdapter();
        DataSet ds = new DataSet();

        public Form1()
        {
            InitializeComponent();
        }
        // Sự kiện Form1_Load
        private void Form1_Load(object sender, EventArgs e)
        {
            string sql = @"SELECT * FROM Employee ";
            da.SelectCommand = new SqlCommand(sql, conn);
            da.Fill(ds, "Employee");
            FirstnameTextBox.DataBindings.Add("Text", ds, "Employee.FirstName");
            LastnameTextBox.DataBindings.Add("Text", ds.Tables["Employee"], "LastName");
            // binding DataTable ds.Tables["Employee"] cua ds voi dataGridView1
            // Cach 1
            dataGridView1.DataSource = ds;
            dataGridView1.DataMember = "Employee";
            // Cach 2
            //dataGridView1.DataSource = ds.Tables["Employee"];

            //binnding bMgr voi DataTable Employee cua DataSet ds
            bMgr = this.BindingContext[ds, "Employee"];
        }
        // Lấy record tiếp theo
        private void button1_Click_1(object sender, EventArgs e)
        {
            if (bMgr.Position < ds.Tables[0].Rows.Count)
                bMgr.Position += 1;
        }
        // Trở lại record trước
        private void button2_Click_1(object sender, EventArgs e)
        {
            if (bMgr.Position > 0)
                bMgr.Position -= 1;
        }
    }
}

```

9.2.3 Data Binding dữ liệu cho Windows Controls

- Để binding dữ liệu với các Windows controls, các bạn chú ý các thuộc tính sau:

- Datasource: nguồn dữ liệu (DataTable, List objects, ArrayList, Array, ...), áp dụng cho tất cả các control như ListBox, ComboBox, DataGridView, ..

- Với ComboBox và ListBox bạn phải chỉ định thuộc tính giá trị và hiển thị
 - ValueMember : tên column (nguồn dữ liệu là DataTable), hoặc tên thuộc tính của lớp (trong trường hợp nguồn dữ liệu là list objects, array, ..) chứa giá trị của SelectedValue của ListBox, hoặc ComboBox
 - DisplayMember: tên column (nguồn dữ liệu là DataTable), hoặc tên thuộc tính của lớp (trong trường hợp nguồn dữ liệu là list objects, array, ..) chứa giá trị hiển thị của ListBox, hoặc ComboBox
- Với DataGridView: nguồn dữ liệu còn có thể là DataSet, nhưng trong trường hợp này bạn phải chỉ định DataMember là tên DataTable được chứa trong DataSet

Ví dụ:

```
// binding DataTable ds.Tables[“Employee”] cua ds voi dataGridView1
// Cach 1
dataGridView1.DataSource = ds;
dataGridView1.DataMember = “Employee”; // data table name
// Cach 2
//dataGridView1.DataSource = ds.Tables[“Employee”];
```

```
//Ví dụ:
using System;
using System.Collections.Generic;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Data.SqlClient;

namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        // Tạo BindingManager
        private BindingManagerBase bMgr;
        SqlConnection conn = new SqlConnection(Properties.Settings.Default.ConnectionString);
        SqlDataAdapter da = new SqlDataAdapter();
        DataSet ds = new DataSet();

        public Form1()
        {
            InitializeComponent();
        }
        // Sự kiện Form1_Load
        private void Form1_Load(object sender, EventArgs e)
        {
            string sql = @"SELECT * FROM Employee ";
            da.SelectCommand = new SqlCommand(sql, conn);
            da.Fill(ds, “Employee”);
            // binding DataTable ds.Tables[“Employee”] cua ds voi dataGridView1
```

```

// Cach 1
dataGridView1.DataSource = ds;
dataGridView1.DataMember = "Employee"; // data table name
// Cach 2
//dataGridView1.DataSource = ds.Tables["Employee"];
// binding combo box voi danh sach phong ban
comboBox1.DataSource = LoadDepartmentData();
comboBox1.ValueMember = "ID";
comboBox1.DisplayMember = "DepartmentName";

//binnding bMgr voi DataTable Employee cua DataSet ds
bMgr = this.BindingContext[ds, "Employee"];

FirstnameTextBox.DataBindings.Add("Text", ds, "Employee.FirstName");
LastnameTextBox.DataBindings.Add("Text", ds, "Employee.LastName");
comboBox1.DataBindings.Add("SelectedValue", ds, "Employee.DepartmentID");
}

// Lấy record tiếp theo
private void button1_Click_1(object sender, EventArgs e)
{
    if (bMgr.Position < ds.Tables[0].Rows.Count)
        bMgr.Position += 1;
}
// Trở lại record trước
private void button2_Click_1(object sender, EventArgs e)
{
    if (bMgr.Position > 0)
        bMgr.Position -= 1;
}
private List<Department> LoadDepartmentData()
{
    SqlDataReader reader = new SqlCommand("SELECT * FROM Department", conn);
    List<Department> list = new List<Department>();

    conn.Open();
    reader.ExecuteReader();
    while (reader.Read())
    {
        Department cl = new Department();
        cl.ID = reader.GetInt32(0);
        cl.DepartmentName = reader.GetString(1);
        list.Add(cl);
    }
    return list;
}
}
}

```

Khi chương trình chạy bạn sẽ nhận được Form hiển thị danh sách nhân viên như sau:

	EmployeeID	LastName	FirstName	Department
▶	1	Nguyen	Nam	1
	2	Pham	Lan Anh	2
*				

Hình 9.5: Giao diện ứng dụng

TÓM TẮT

Trong bài này, học viên làm quen với lập trình cơ sở dữ liệu với ADO.NET.

Năm được các cơ chế DataBinding với các Windows Controls: *ListBox*, *ComboBox*, *DataGridView*

Năm được cơ chế DataBinding: *Simple DataBinding*, *Complex DataBinding*

BÀI 10: Entity FrameWork (EF)

Sau khi học xong bài này, học viên có thể nắm được:

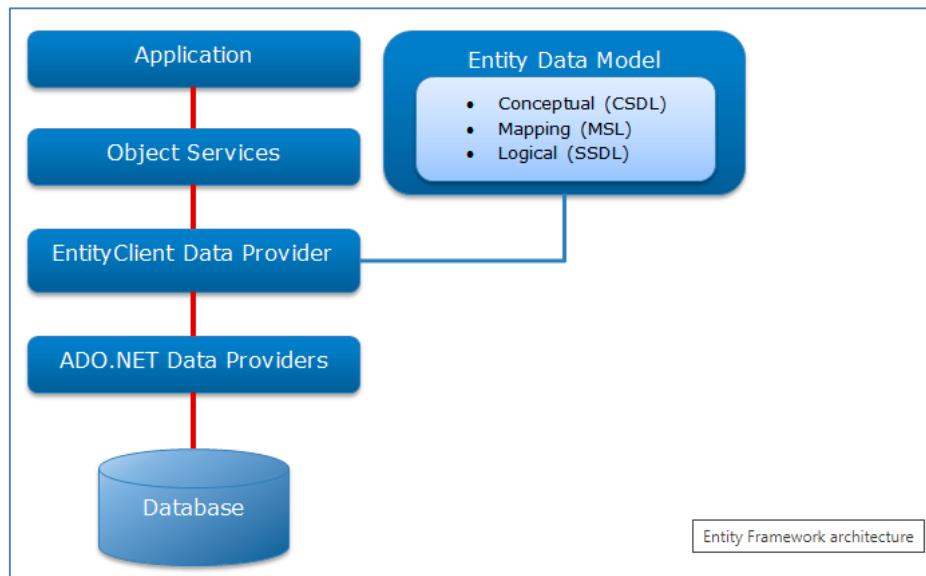
- *Lợi ích của Entity FrameWork (EF)*
- *Các thành phần trong EF*
- *Cách tạo EDM*
- *Entity Set*
- *Sử dụng EF với mô hình code first*
- *Các thao tác insert, update, delete trên EF*
- *LINQ to Entities, Entity SQL và Native SQL*

10.1 Entity FrameWork (EF)

Entity Framework (EF) là một framework ánh xạ quan hệ đối tượng (ORM) dành cho ADO.NET, là 1 phần của .NET Framework. EF cho phép các nhà phát triển tương tác với cơ sở dữ liệu quan hệ theo phương pháp hướng đối tượng. Lợi ích lớn nhất của EF là giúp lập trình viên giảm thiểu việc lập trình mã nguồn cần thiết để truy cập và tương tác với cơ sở dữ liệu. EF được Microsoft hỗ trợ phát triển lâu dài và bền vững, bằng cách sử dụng Entity Framework, truy vấn LINQ, thì việc truy xuất và thao tác với cơ sở dữ liệu cũng như các đối tượng trở nên mạnh mẽ hơn.

10.1.1 Kiến trúc trong Entity FrameWork

Kiến trúc của Entity Framework được minh họa như sau:



10.1.1.1 Tầng Ứng dụng (Application)

Application (ứng dụng) là tầng chứa giao diện (Console, forms, web ...) với các đoạn mã nguồn (C#...) để tương tác dữ liệu với các tầng khác trong mô hình thông qua Object Services.

10.1.1.2 Object Services

Object Services (**các dịch vụ đối tượng**) là tầng chứa quá trình tương tác giữa ứng dụng và cơ sở dữ liệu (database), hay nói cách khác là nơi chủ yếu để truy cập dữ liệu từ database và trả ngược kết quả về giao diện. Object Services cung cấp các tiện ích để truy vết các thay đổi và quản lý nhận dạng, đồng thời là các quan hệ và thay đổi ở database.

10.1.1.3 Entity Data Model

EDM (**mô hình dữ liệu thực thể**) chứa 3 phần chính: mô hình khái niệm (CSDL – Conceptual schema definition language), mô hình ánh xạ (MSL – mapping specification language) và mô hình lưu trữ (SSDL – store schema definition language). EDM khác với EntityClient Data Provider ở chỗ EDM sử dụng LINQ là ngôn ngữ truy vấn tương tác với database.

Mô hình khái niệm (phần mở rộng *.CSDL): Mô hình khái niệm chứa các lớp mô hình và mối quan hệ giữa các lớp này. Nó độc lập với thiết kế Table trong cơ sở dữ liệu.

Mô hình lưu trữ (phần mở rộng *.SSDL): Mô hình lưu trữ là 1 mô hình thiết kế database bao gồm các bảng, view, stored procedure, mối quan hệ giữa chúng và các khóa. Mô hình này thể hiện gần giống mô hình quan hệ các bảng trong database.

Mô hình ánh xạ (phần mở rộng *.MSL): Mô hình ánh xạ gồm thông tin về cách mô hình khái niệm được ánh xạ đến mô hình lưu trữ

10.1.1.4 EntityClient Data Provider

Đây là tầng cung cấp các kết nối, diễn dịch các truy vấn thực thể thành truy vấn nguồn dữ liệu (chuyển LINQ to Entity hay các truy vấn thực thể SQL thành truy vấn SQL), trả về data reader để EF dùng chuyển dữ liệu thực thể thành các đối tượng. Phần này kết nối ADO.NET Data Providers để gửi hoặc lấy dữ liệu từ database. Tầng này hoàn toàn khác với EDM (Entity Data Model) khi thực thi các truy vấn tương tự như cách thực hiện ở ADO.NET Provider.

10.1.1.5 ADO.Net Data Provider

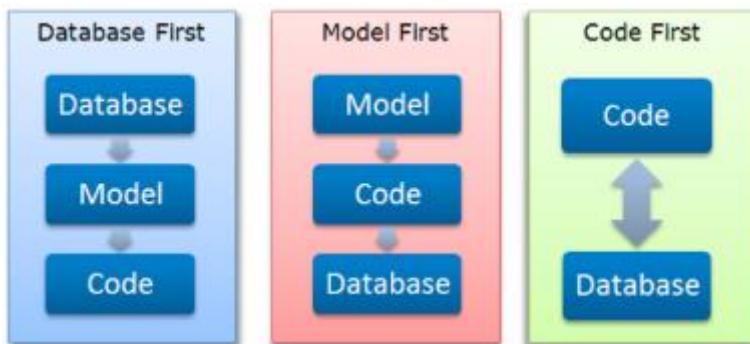
Đây là tầng thấp nhấp để dịch các truy vấn LINQ to Entity thông qua câu lệnh thành các câu lệnh SQL và thực thi các câu lệnh trong hệ thống DBMS (database management system – hệ quản lý dữ liệu) nào đó. Tầng này kết với database sử dụng ADO.NET.

10.1.2 Các thành phần trong Entity FrameWork

Các thành phần trong Entity Framework:

- Code là mã lệnh tạo thành các lớp đối tượng dữ liệu cho phép thao tác với dữ liệu.
- Model là sơ đồ gồm các hộp mô tả các thực thể và các đường nối kết mô tả các quan hệ.
- Database là cơ sở dữ liệu (có thể là SQL Server, Compact SQL Server, Local database, MySQL, Oracle,...)

Có 3 cách sử dụng Entity Framework: Code First, Models First, Database First.



Vì lý do này, tất cả các class thực thi các giao diện trên đều có thể trở thành nguồn dữ liệu.

- **Database first:** nên dùng khi ta đã có sẵn CSDL (không phải tạo), EF Wizard sẽ tạo Model và Code cho ta.

- **Model first:** nên dùng khi ta bắt đầu thiết kế CSDL từ đầu (từ chưa có gì). Ta sẽ thiết kế mô hình CSDL (Model) EF sẽ tự tạo code cho ta, sau đó nhờ EF Wizard tạo CSDL.

- **Code first:** Riêng với cách này có thể làm việc với 2 lựa chọn, làm việc với database có sẵn hoặc sẽ tạo mới. Với 1 bài toán yêu cầu nghiệp vụ thay đổi liên tục và phát triển nhanh chóng thì code first sẽ là mô hình tiếp cập phù hợp.

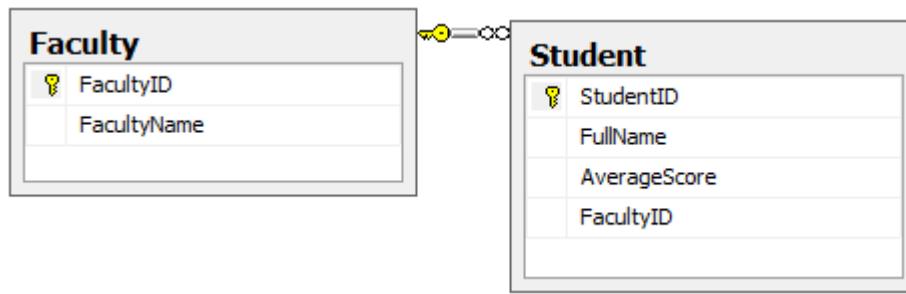
10.2 Sử dụng Entity Framework mô hình code first

Với *lựa chọn có trước cơ sở dữ liệu*: giả sử với CSDL *QuanLySinhVien* có 2 bảng Student và Faculty được mô tả như sau:

Student: Mỗi sinh viên có 1 mã số duy nhất (StudentID), Họ và tên (FullName), Điểm trung bình (AverageScore) và mã khoa (FacultyID)

Faculty: Mỗi khoa có 1 mã khoa duy nhất (FacultyID) và tên khoa (FacultyName).

Sơ đồ Diagram:

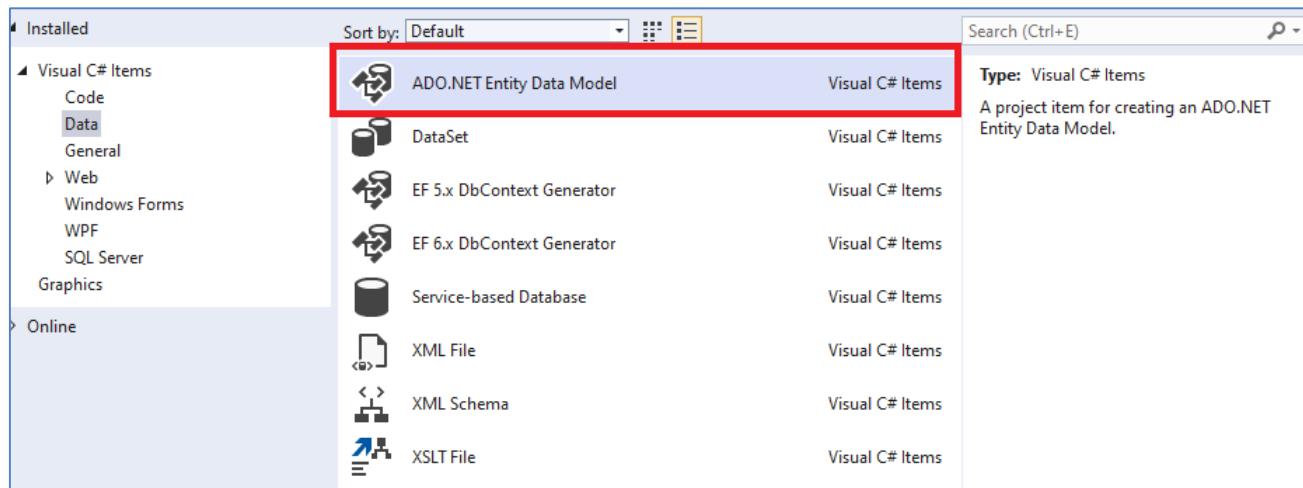


10.2.1 Cài đặt EF

Có thể cài đặt EF (EntityFramework.dll) qua NuGet đối với bất kỳ phiên bản nào của VS. Hoặc khi tạo xong Entity Data Model thì VS sẽ tự động thêm thư viện EntityFramework.dll nếu chưa có.

10.2.2 Tạo Entity Data Model

Bước 1: Chọn new ADO.NET Entity Data Model, Đặt tên cho EDM, ví dụ là "StudentModel" và Add.

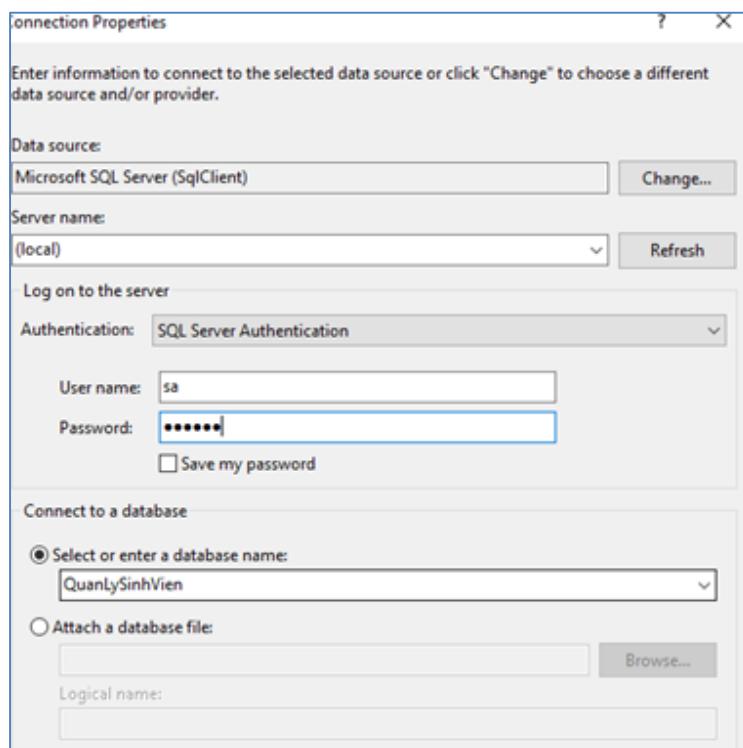


Entity Data Model Wizard trong VS mở ra với 4 lựa chọn:

- **EF Designer from database:** cho phương pháp tiếp cận Database First
- **Empty EF Designer model:** cho phương pháp tiếp cận Model First
- **Empty Code First model:** cho phương pháp tiếp cận Code First khi chưa có sẵn database
 - **Code First from database:** cho phương pháp tiếp cận Code First khi có sẵn database

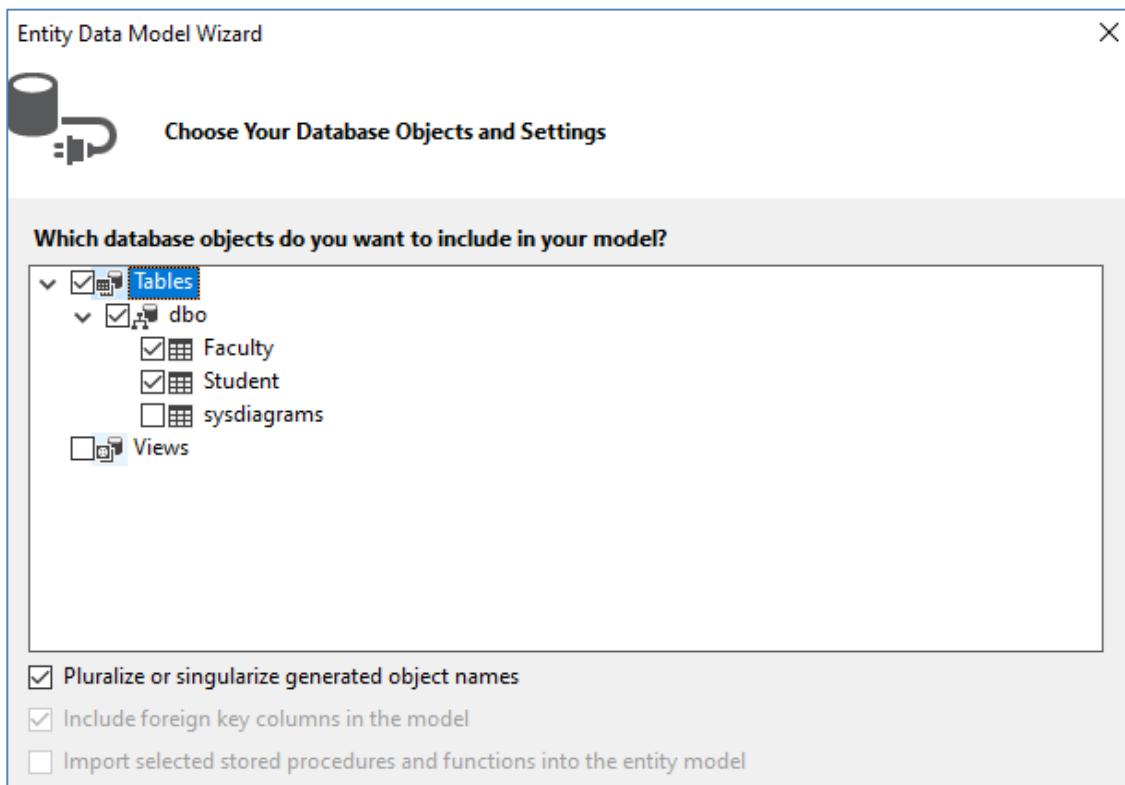
Trong bài hướng dẫn cơ bản này chọn **Code First from database** và click **Next**.

Bước 2: Bạn có thể chọn từ kết nối database sẵn có hoặc tạo một kết nối mới bằng cách bấm nút 'New Connection'. Chúng ta sẽ sử dụng DB connection sẵn có tới CSDL *QuanLySinhVien*. Thao tác này cũng sẽ thêm một connection string tới file **app.config** với hậu tố mặc định là tên CSDL và có thể dễ dàng thay đổi nếu muốn. Click 'Next' sau khi cài đặt DB connection.



Ví dụ sử dụng SQL server với server name local, tài khoản sa và tên database

Bước 3: Chọn các bảng, store procedure...để tạo ra các thực thể (entity) tương ứng. Ở Ví dụ này giả sử muốn tạo ra object cho sinh viên (Student) và Khoa (Faculty) Click finish để hoàn thành.VS sẽ tự động sinh ra các class tương ứng



10.2.3 DBContext

DbContext là một phần quan trọng của Entity Framework. Nó là một cầu nối giữa lớp domain hoặc thực thể và CSDL của bạn. Lớp kế thừa từ DbContext được gọi là lớp context trong Entity Framework.

Trong phần trước Tạo Entity Data Model, EDM khởi tạo lớp *StudentModel* được dẫn xuất từ lớp System.Data.Entity.DbContext như bên dưới.

```

1  namespace EFStudentCodeFirst
2  {
3      using System;
4      using System.Data.Entity;
5      using System.ComponentModel.DataAnnotations.Schema;
6      using System.Linq;
7
8      public partial class StudentModel : DbContext
9      {
10         public StudentModel()
11             : base("name=StudentModel")
12         {
13         }
14
15         public virtual DbSet<Faculty> Faculties { get; set; }
16         public virtual DbSet<Student> Students { get; set; }
17
18         protected override void OnModelCreating(DbModelBuilder modelBuilder)
19         {
20         }
21     }
22 }

```

DbContext là lớp chính chịu trách nhiệm cho việc tương tác với dữ liệu như là đối tượng. DbContext chịu trách nhiệm cho các hoạt động sau:

EntitySet: DbContext chứa tập thực thể (DbSet< TEntity >) cho tất cả thực thể nối với những bảng của CSDL.

Querying: DbContext chuyển đổi những truy vấn LINQ-to-Entities thành truy vấn SQL và gửi nó tới CSDL.

Change Tracking: Nó giữ việc theo dõi những thay đổi xảy ra trong những thực thể sau khi nó đã truy vấn từ CSDL.

Persisting Data: Nó cũng thực hiện các thao tác Insert, Update và Delete tới CSDL dựa trên những gì mà thực thể thể hiện.

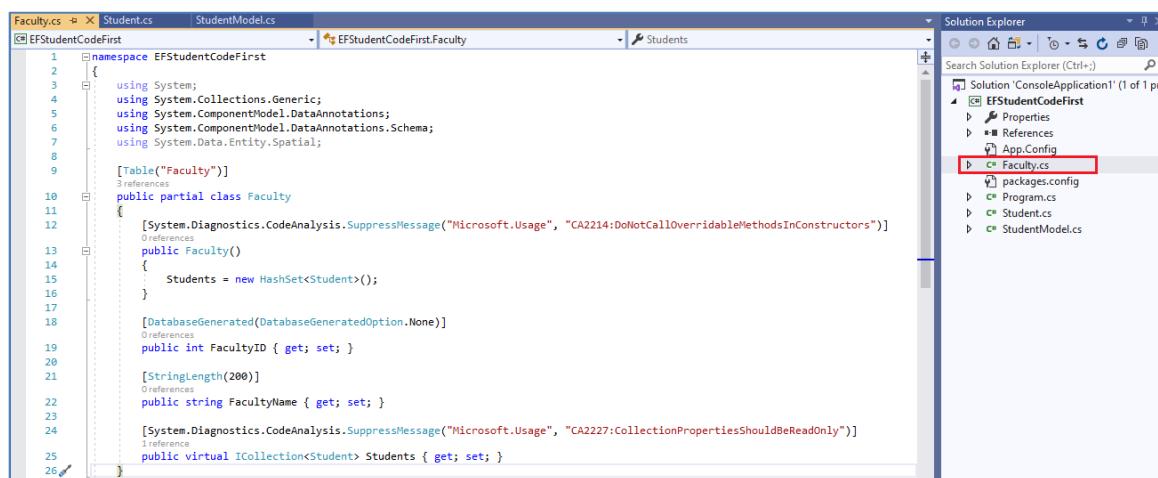
Caching: DbContext mặc định thực hiện caching mức đầu tiên. Nó lưu những thực thể đã được nhận suốt vòng đời của một lớp context.

Manage Relationship: DbContext cũng quản lý những quan hệ sử dụng CSDL

Object Materialization: DbContext chuyển đổi bảng dữ liệu thô vào những đối tượng thực thể.

10.2.4 Entity

Các thực thể (Entity) đại diện cho các class tương ứng được sinh ra (khi được chọn từ EDM).



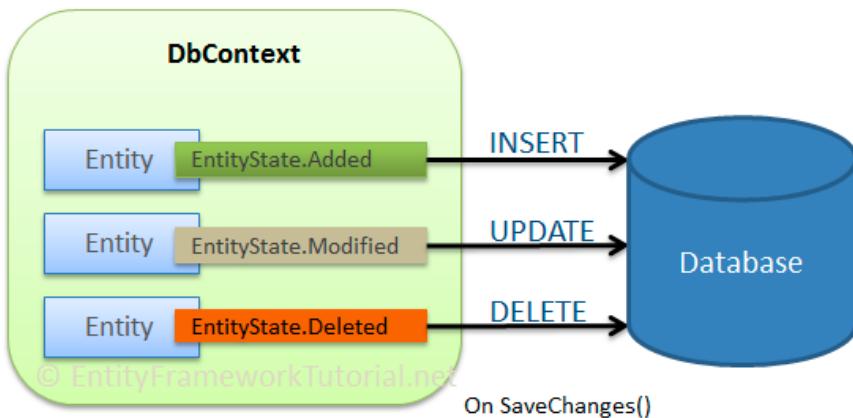
```

Faculty.cs  Student.cs  StudentModel.cs
EFStudentCodeFirst  EFStudentCodeFirst  Students
EFStudentCodeFirst.Faculty
1  namespace EFStudentCodeFirst
2  {
3      using System;
4      using System.Collections.Generic;
5      using System.ComponentModel.DataAnnotations;
6      using System.ComponentModel.DataAnnotations.Schema;
7      using System.Data.Entity.Spatial;
8
9      [Table("Faculty")]
10     public partial class Faculty
11     {
12         [System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2214:DoNotCallOverridableMethodsInConstructors")]
13         public Faculty()
14         {
15             Students = new HashSet<Student>();
16         }
17
18         [DatabaseGenerated(DatabaseGeneratedOption.None)]
19         public int FacultyID { get; set; }
20
21         [StringLength(200)]
22         public string FacultyName { get; set; }
23
24         [System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly")]
25         public virtual ICollection<Student> Students { get; set; }
26     }
}

```

Ở ví dụ trên, EDM khi chọn 2 bảng (Student, Faculty) tương ứng được sinh ra Student.cs và Faculty.cs

Sử dụng cùng EDM cho các hoạt động CRUD (Create, Read, Update, và Delete) mà đã tạo trong chương Tạo mô hình dữ liệu thực thể. Khi thêm, cập nhật hoặc xóa dữ liệu sẽ đưa vào trên **EntityType**. Hình dưới đây minh họa các thao tác CUD (Tạo, Cập nhật, Xóa).



Sau khi phương thức DbContext.**SaveChanges** () được gọi. Entity Framework sẽ xây dựng và thực thi các câu lệnh INSERT, UPDATE và DELETE cho các thực thể có EntityState.

10.2.4.1 Thêm dữ liệu

Để thêm một thực thể mới vào một DbContext: Sử dụng phương thức DbSet.**Add**

Để lưu vào cơ sở dữ liệu: gọi phương thức ***SaveChanges()***.

Ví dụ: Thực hiện thêm 1 khoa mới 'CNTT' có mã khoa 100 vào CSDL.

```
using (var context = new StudentModel())
{
    var f = new Faculty() { FacultyID = 100, FacultyName= "CNTT" };
    context.Faculties.Add(f); //Add vào DbSet<Faculty>
    context.SaveChanges(); //lưu vào csdl
}
```

Ví dụ: Thực hiện thêm mới 1 sinh viên vào Student

```
using (var context = new StudentModel())
{
    var s = new Student() { StudentID = "1799062090", FullName="Lê Tuấn Sang", AverageScore
= 7.6, FacultyID= 100};

    context.Students.Add(s); //Add vào DbSet<Student>
    context.SaveChanges(); //lưu vào csdl
}
```

10.2.4.2 Cập nhật dữ liệu

Khi chỉnh sửa dữ liệu trên DbSet trong DbContext, EF sẽ tự động đánh dấu EntityState thành Modified.

Để cập nhật vào cơ sở dữ liệu: gọi phương thức **SaveChanges()**.

Ví dụ: Cập nhật lại họ tên và điểm của sinh viên có mã "1799062090"

```
using (var context = new StudentModel())
{
    var std = context.Students.First(p => p.StudentID == "1799062090");//lấy ra sinh viên
    //có mã số 1799062090

    //Set các giá trị muốn cập nhật
    std.FullName = "Lê Tuấn Sang";
    std.AverageScore = 8.6;

    context.SaveChanges(); //lưu vào csdl
}
```

10.2.4.3 Xóa dữ liệu

Để xóa dữ liệu ở DbSet trong DbContext: Sử dụng phương thức DbSet.**Remove()**

Để thao tác xóa thực hiện vào cơ sở dữ liệu: gọi phương thức **SaveChanges()**.

Ví dụ: Xóa sinh viên có mã số "1799062090"

```
using (var context = new StudentModel())
{
    var std = context.Students.First(p => p.StudentID == "1799062090");//lấy ra sinh
    //viên có mã số 1799062090

    context.Students.Remove(std); //Remove trong DBSet
    context.SaveChanges(); //lưu vào csdl
}
```

10.2.5 Truy vấn với EDM

Chúng ta đã tạo EDM, DbContext và những lớp thực thể trong những phần trước. Bạn sẽ học những kiểu truy vấn khác nhau mà Entity Framework, nó lần lượt chuyển đổi thành truy vấn SQL cho CSDL bên dưới.

Entity Framework hỗ trợ 3 kiểu truy vấn: LINQ to Entities, Entity SQL và Native SQL.

10.2.5.1 LINQ to Entities

LINQ-to-Entities hoạt động trên những thực thể Entity Framework để truy cập dữ liệu từ CSDL bên dưới. Bạn có thể sử dụng cú pháp LINQ method hoặc cú pháp truy vấn khi truy vấn với EDM (xem lại bài LINQ).

Lưu ý: LINQ có hai cú pháp khác nhau: query syntax và method syntax. Việc sử dụng cú pháp nào hoàn toàn do quyết định cá nhân.

Ví dụ: Xuất ra danh sách sinh viên có điểm ≥ 5 và tên khoa tương ứng.

```
var context = new StudentModel();
// #2: method syntax (lambda expression s=>s.AverageScore>=5)
List<Student> studentList = context.Students.Where(s => s.AverageScore >= 5).ToList();

// Xuất danh sách sinh viên và tên khoa
foreach (Student s in studentList)
    Console.WriteLine("Mã SV = {0}, Tên SV = {1}, Điểm = {2}, Tên Khoa = {3}",
s.StudentID, s.FullName, s.AverageScore, s.Faculty.FacultyName);
```

10.2.5.2 Native SQL

Có thể thực thi những truy vấn native SQL cho một CSDL quan hệ

Ví dụ: Lấy tất cả các sinh viên có điểm ≥ 5 trong cơ sở dữ liệu

```
using (var ctx = new StudentModel())
{
    var studentList = ctx.Students.SqlQuery("SELECT * from Student WHERE
AverageScore>=5").ToList<Student>();
}
```

10.2.5.3 Entity SQL

Entity SQL là cách khác để tạo một truy vấn. Nó được xử lý bởi Object Services của Entity Framework Object Services trực tiếp. Nó trả về ObjectQuery thay vì IQueryable. Bạn cần ObjectContext để tạo một truy vấn sử dụng Entity SQL.

Ví dụ: Khi sử dụng database first, Lấy các sinh viên có điểm >5

```
// SchoolDBEntities được tạo khi sử dụng DatabaseFirst
string sqlString = "SELECT VALUE st FROM SchoolDBEntities.Students " +
                    "AS st WHERE st.AverageScore >= 5";
var objctx = (ctx as IObjectContextAdapter).ObjectContext;
ObjectQuery<Student> student = objctx.CreateQuery<Student>(sqlString);
var studentList = student.ToList<Student>();
```

TÓM TẮT

Trong bài này, học viên làm quen với entity framework và sử dụng EF mô hình code first với cơ sở dữ liệu có sẵn trên SQL server bao gồm:

- Tạo Entity Data Model, DBContext, Sử dụng EDM mô hình code first cho hướng tiếp cận có sẵn database
- Các thao tác với Create, Update, Delete
- Truy vấn với LINQ to Entities

CÂU HỎI ÔN TẬP

Câu 1: Lợi ích lớn nhất của EF?

Câu 2: Các thành phần trong Entity FrameWork?

Câu 3: Sự khác nhau Database First và Code first theo hướng tiếp cận đã có CSDL?

Câu 4: SaveChange có ý nghĩa gì?

Câu 5: Entity Framework hỗ trợ các kiểu truy vấn gì?

Sử dụng cơ sở dữ liệu sau cho các câu hỏi từ 6-10

Cho cơ sở dữ liệu cho nhân viên-phòng ban mô tả như sau

NHÂN VIÊN: mỗi nhân viên có 1 Mã nhân viên duy nhất, Tên nhân viên, và thuộc một mã phòng (là khóa ngoại tham chiếu đến phòng ban).

PHÒNG BAN: mỗi phòng ban có 1 Mã phòng duy nhất, tên phòng, và mã trưởng phòng (cho phép null, là khóa ngoại tham chiếu đến mã nhân viên).

Thêm 1 số dữ liệu vào cơ sở dữ liệu trên

Sử dụng mô hình Entity Framework để thực hiện các yêu cầu sau

Câu 6: Thực hiện thêm mới 1 phòng ban? Rồi thêm các nhân viên thuộc phòng đó?

Câu 7: Thay đổi lại tên các nhân viên vừa thêm ở câu trên?

Câu 8: Xóa phòng ban vừa thêm và các nhân viên tương ứng thuộc phòng ban đó?

Câu 9: Xuất danh sách từng phòng ban, ứng với mỗi phòng ban xuất ra các nhân viên tương ứng?

Câu 10: Giả sử người ta muốn thêm 1 cột người quản lý (mặc định cho phép null, là khóa ngoại tham chiếu đến mã nhân viên) vào bảng nhân viên. Hãy chỉnh sửa diagram, EDM, dữ liệu.... tương ứng để vẫn đáp ứng được các yêu trước.

Xuất ra danh sách nhân viên ứng với từng người quản lý trực tiếp nếu có?

TÀI LIỆU THAM KHẢO

1. Bài giảng Lập trình C# trên Windows, Ths. Nguyễn Hà Giang, 2010.
2. C# and .NET programing, msdn.microsoft.com, 2012.
3. Pro C# 2005 and the .NET 2.0 Platform, Andrew Troelsen, Apress, 2005.
4. C# 2.0 Practical Guide for Programmers, Michel de Champlain, Brian G. Patrick, Morgan Kaufmann publishers. 2005.
5. Windows Forms Programming with C#, Erik Brown, Manning Publications, 2008.
6. Microsoft Visual C# 2010 Step by Step, Microsoft Press, 2010.
7. Windows Forms 2.0 Programming, Chris Sells, Michael Weinhardt, Additon Wesley Professional, 2003.
8. Teach yourself .NET Windows Forms in 21 Days, Chris Payne, SAMS, 2003.
9. Source code tham khảo ở <http://www.wrox.com>.
10. Các topic lập trình ở www.codeguru.com, www.codeproject.com.