MỤC LỤC

Bài 1
Microsoft .NET Framework

Cài đặt Visual Studio.NET Beta 2

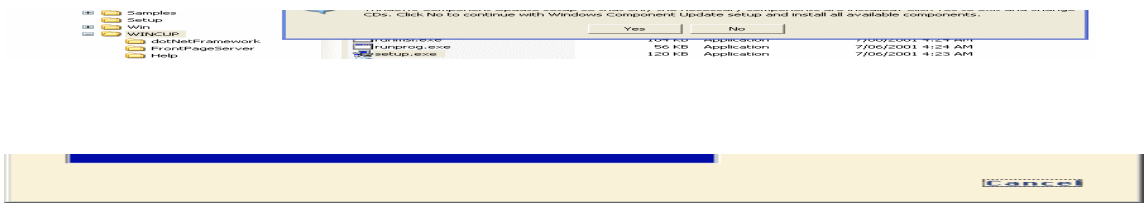
Visual Studio.NET nằm trong 3 CDs, gồm có 4 files:

- EN_VS_NET_BETA2_WINCUP.exe
- EN_VS_NET_BETA2_CD1.exe
- EN_VS_NET_BETA2_CD2.exe
- EN_VS_NET_BETA2_CD3.exe

Chạy EN_VS_NET_BETA2_WINCUP.exe để Unzip nó vào một folder trong một harddisk (eg: E:\CD\WINCUP).

Lần lượt chạy EN_VS_NET_BETA2_CD1.exe,

EN_VS_NET_BETA2_CD2.exe và EN_VS_NET_BETA2_CD3.exe để Unzip chúng vào chung một folder trong một harddisk (eg: E:\CD). Việc Unzip files có thể rất lâu, bạn phải kiên nhẫn chờ cho đến khi nó kết thúc. Nếu không, khi cài đặt có thể bị than phiền là thiếu files. Kế đó, trước hết chạy Setup.exe của Windows Component Update như dưới đây, click **No** khi Warning dialog hiện ra:



Tiếp theo, chạy Setup.exe của Visual Studio.NET:

Bạn nên có CPU Pentium III, 500MHz trở lên, với 256 MB RAM và ít nhất 10GB Harddisk. Về OS bạn nên dùng Windows 2000 (Professional

hay Server) hay Windows XP. Lý do chính là các versions Windows này hỗ trợ Unicode và có Internet Information Server (IIS) hỗ trợ ASPX để ta dùng cho ASP.NET.

Mở các Samples của QuickStart

Trước khi expand Samples của Quickstart bạn cần phải cài đặt IIS. Nếu chưa làm việc ấy bạn bỏ CD của Windows2000 hay WindowsXP vào để install IIS component.

QuickStart Samples của .NET Framework chứa các giải thích căn bản và nhiều thí dụ. Để expand các Samples doubleclick Webpage **Starthere.htm** như trong hình dưới đây:

Khi trang Web của QuickStart hiện ra, click **QuickStart, tutorials and samples** rồi sau đó theo chỉ dẫn từng bước.

Trang Framework SDK QuickStart Tutorials cho ta các bài tập của **ASP.NET, Windows Forms** và **How Do I...**

Nhớ để nguyên các folders của Unzipped files (E:\CD, E:\CD\WINCUP), đừng delete chúng, vì .NET sẽ còn dùng chúng. Ngoài ra, nếu sau này .NET bị corrupted vì conflict với các application software khác, bạn có thể cài đặt .NET lại.

Giới hạn của Software Tools hiện giờ

Architect của application software hiện giờ có nói chung ba tầng (three tiers): tầng giao diện (Presentation Tier), tầng giữa (Middle Tier) và tầng dữ kiện (Data Tier):

Presentation Tier: Trong desktop Client ta dùng VB6 và nối với middle tier qua DCOM. Trong browser based Client ta dùng Javascript hay Java applet. Từ browser based Client ta dùng http để nối với middle tier qua IIS/ASP (có thể dùng COM ở đây).

Middle Tier: Chứa các rules để validate data trên client và các business rules khác. Ta dùng VB6 ở đây, nhưng cách triển khai COM với những Object Oriented Programming concepts rắc rối hơn bình thường. Ta phải thiết kế sao cho các components scale well (dùng cho mọi cở). Có khi dùng Microsoft Transaction Server trên Windows NT hay COM+ Services trên Windows 2000. Lấp ráp các versions của components là một thách thức lớn.

Nhiều khi middle tier còn nói chuyện với các database qua HTTP, ADO và CDO (Collaborative Data Objects), .v.v..

Data Tier: Thường là relational database như Microsoft SQL Server hay Oracle. Ngoài ra còn có Exchange hay các database xưa của mainframe.

Do đú ta thấy:

- Desktop tools không thích hợp cho Distributed System hay Internet

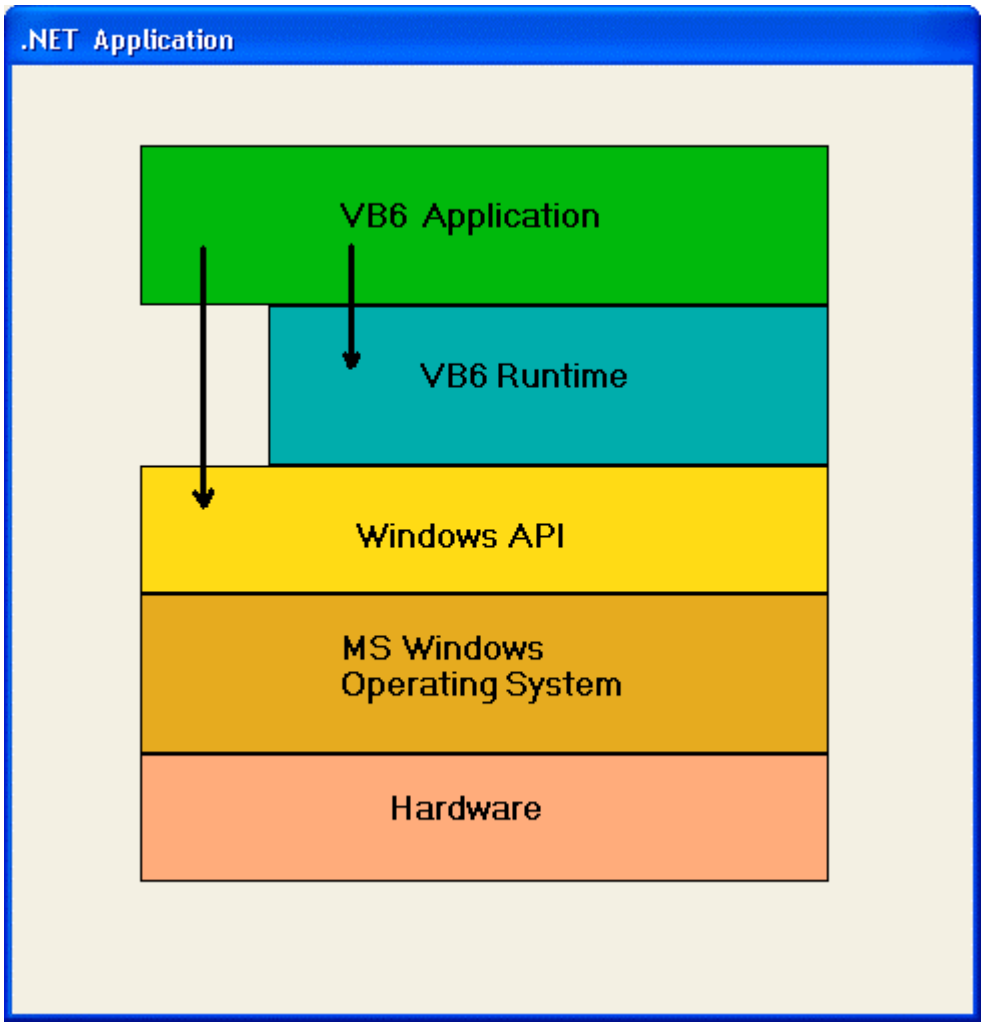
  Phải dùng nhiều thứ codes như VB6 Code, VBScript, JavaScript, Dynamic HTML, Cascading Style Sheet, VC++, Stored Procedures (Transact-SQL trong SQLServer hay PL-SQL trong Oracle).

Tùy thuộc quá nhiều vào central database: Ngay cả ADO dù là tiện dụng cũng đòi hỏi Client luôn luôn connect với Server. Điều này không thích

hợp cho Internet applications, vì bản chất của Internet là **stateless** (không giữ trạng thái), mỗi lần cần làm việc mới connect lại một chút thôi.

- DLL "Hell": Các ActiveX cần phải được registered với Windows Registry, chỉ có một version được chấp nhận. Nếu version mới nhất của một DLL không compatible với các versions trước đó mà applications trên máy đang cần thì có rắc rối.

Giới hạn của VB6



- Thiếu khả năng inheritance (thừa kế) và một số Object Oriented features khác.
- Khả năng Error handling giới hạn: On Error Goto ..., On Error Resume Next

- Nhiều khi cần phải gọi Windows API để làm những việc VB6 không hỗ trợ: việc này không tự nhiên và đôi khi nguy hiểm.
- Không có multi-threading: không thể đoán trước response của code chạy trong các windows của cùng một VB6 application. Ngay cả giải quyết vấn đề multitasking bằng Timers cũng không đáng tin cậy.
- Không dễ dùng chung với các ngôn ngữ khác như VC++.
- Không tiện cho Web development: WebClass không thành công lắm. Ít ai chịu cho ta cài ActiveX trên máy của họ.

.NET Framework

.NET được developed từ đầu năm 1998, lúc đầu có tên là **Next Generation Windows Services (NGWS)**. Nó được thiết kế hoàn toàn từ con số không để dùng cho Internet. Viển tượng của Microsoft là xây dựng một **globally distributed system**, dùng **XML** (chứa những databases tí hon) làm chất keo để kết hợp chức năng của những computers khác nhau trong cùng một tổ chức hay trên khắp thế giới.

Những computers này có thể là Servers, Desktop, Notebook hay Pocket Computers, đều có thể chạy cùng một software dựa trên một platform duy nhất, độc lập với hardware và ngôn ngữ lập trình. Đó là .NET Framework. Nó sẽ trở thành một phần của MS Windows và sẽ được port qua các platform khác, có thể ngay cả Unix.

Mặc dầu hãy còn là Beta, .NET Framework rất stable và Visual Studio.NET rất ít bugs, có thể dùng cho software development ngay từ bây giờ. Hiện nay đã có một số sách về lập trình .NET do Wrox và Oreilly xuất bản.

Các phần chính của Microsoft.NET Framework:

.NET application được chia ra làm hai loại: cho Internet gọi là **ASP.NET**,

gồm có **Web Forms** và **Web Services** và cho desktop gọi là **Windows Forms**.

Windows Forms giống như Forms của VB6. Nó hổ trợ **Unicode** hoàn toàn, rất tiện cho chữ Việt và thật sự Object Oriented. Web Forms có những **Server Controls** làm việc giống như các Controls trong Windows Forms, nhất là có thể dùng codes để xử lý Events y hệt như của Windows Forms.

Điểm khác biệt chánh giữa ASP (Active Server Pages) và ASP.NET là trong ASP.NET, phần đại diện visual components và code nằm riêng nhau, không lộn xộn như trong ASP. Ngoài ra ASP.NET code hoàn toàn Object Oriented.

Web Services giống như những Functions mà ta có thể gọi dùng từ các URL trên Internet, thí dụ như Credit Card authorisation.

ADO.NET là một loại cache database nho nhỏ (gọi là disconnected database) để thay thế ADO. Thay vì application connects vĩnh viễn với database mẹ qua ADO, application trong .NET làm việc với portable database chỉ chứa một hai tables, là copy từ database mẹ. Khi nào cần, portable database này (ADO.NET) sẽ được reconciled với database mẹ để update các thay đổi. Hai tables trong ADO.NET có thể được **related** nhau trong **Master/Details relationship**. Vì ADO.NET có chứa original data lẫn data mới nhất nên **Rollback** trong ADO.NET rất dễ dàng và nhẹ ký.

XML được yểm trợ tối đa. Nằm phía sau ADO.NET là XML. XML có thể là Table of records trong ADO.NET hay Tree of nodes trong **DOM (Document Object Model)**.

IO được hổ trợ bằng toàn bộ **Stream** kể cả **Memory Stream** và **StreamReader/StreamWriter**. Thêm vào là DataFormatting cho **Serialisation** để chứa Object xuống binary file hay text file.

TCP/IP và **http** là hai protocols thông dụng nhất trong .NET, nhưng chúng làm việc phía sau sân khấu giúp ta gọi một remote procedure (nằm trên computer khác) dễ dàng như một local procedure. Kỹ thuật ấy gọi là **Remoting**.

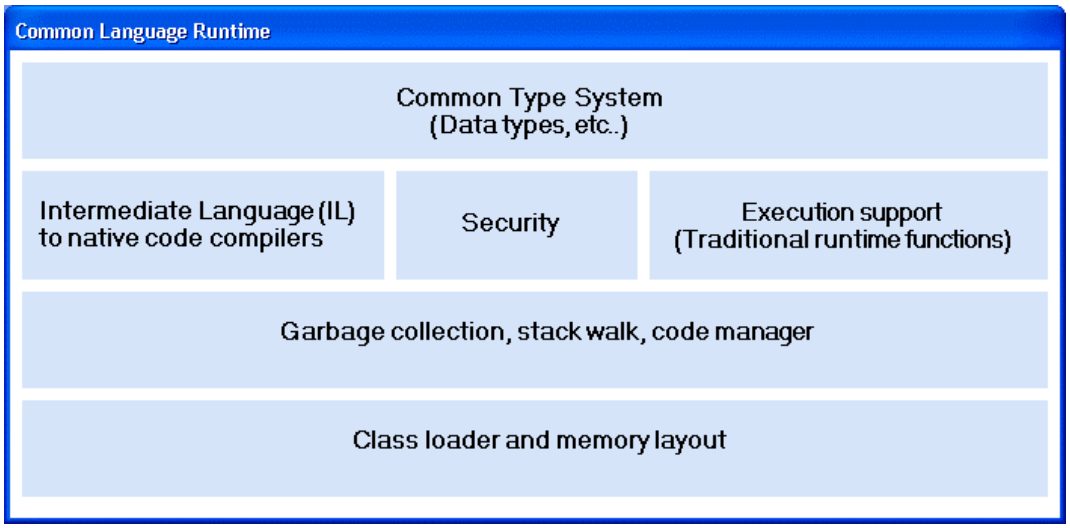
Security hổ trợ **Cryptography, Permissions** và **Policy**.

Diagnostics cho ta **Debug** và **Trace**.

Threading rất tiện và đơn giản để implement **Lightweight Process**. Vấn đề Timing trong .NET program rất linh động, hiệu quả và chính xác.

Việc thiết kế Common Language Runtime nhắm vào các mục tiêu chính sau đây:

- Việc triển khai đơn giản và nhanh hơn: developers sẽ dành thì giờ quyết định ráp những software components nào lại với nhau nhiều hơn là lập trình thật sự.
- Các công tác thiết yếu ("plumbing") như memory management, process communication .v.v. được lo liệu tự động.
- Các công cụ hổ trợ rất đầy đủ (no more API): .NET Framework Base classes rất phong phú cho file, network, serialisation, mã hóa, XML, database, v.v..
- Cài đặt đơn giản và an toàn (no more DLL "hell"): chỉ cần xcopy files, giống như thời vàng son của DOS. Lý do là .NET application chạy trên .NET framework, một khi ta đã cài .NET framework vào máy rồi thì có đầy đủ mọi .DLL cần thiết. Có lẽ trong tương lai Microsoft cài .NET framework chung với Windows.
- Dùng cho từ WindowsCE đến Desktop, đến Web (scalability).



Metadata

Metadata là các dữ kiện cắt nghĩa cho ta biết về dữ kiện. Thí dụ **XML Schema** của một XML file là metadata cắt nghĩa về data structure của data trong XML file. Chính cái XML Schema cũng là một XML file. Các .NET application components, gọi là **Assembly**, chứa rất nhiều metadata để cắt nghĩa về chính nó (self describing). Tìm biết về một .NET application để có thể làm việc với nó thì gọi là **Reflection**.

Hổ trợ và phối hợp mọi ngôn ngữ lập trình

Common Language Runtime (CLR) là trung tâm điểm của .NET Famework, nó là hầm máy để chạy các năng tính của .NET. Trong .NET, mọi ngôn ngữ lập trình đều được compiled ra **Microsoft Intermediate Language (IL)** giống giống như byte code của Java. Nhờ bắt buộc mọi ngôn ngữ đều phải dùng cùng các loại data types (gọi là **Common Type System**) nên Common Language Runtime có thể kiểm soát mọi interface, gọi giữa các components và cho phép các ngôn ngữ có thể hợp tác nhau một cách thông suốt. Tức là trong .NET, VB.NET program có thể inherit C# program và ngược lại một cách hoàn toàn tự nhiên. Điều này chẳng những giúp các VC++ hay Java programmers bắt đầu

dùng C# một cách dễ dàng mà còn làm cùng một dự án với VB.NET programmers nữa.

Nếu VC++ linh động và hiệu năng hơn VB6, thì C# chẳng khác gì VB.NET. Bạn có thể port C# code qua VB.NET code rất dễ dàng. Vì source code VC++ và Java gần gũi C# hơn VB6 với VB.NET nên ngoài đời có nhiều C# code hơn VB.NET. Do đó, mặc dầu hai ngôn ngữ VB.NET và C# đều ngang cơ nhau, nếu dùng C# bạn được lợi điểm có nhiều source code sẵn và nhất là lâu nay người ta vẫn mang ấn tượng rằng VC++ hay Java programmers mới thật sự là các cao thủ lập trình, và có khuynh hướng trả lương các guru VC++/Java cao hơn VB programmers.

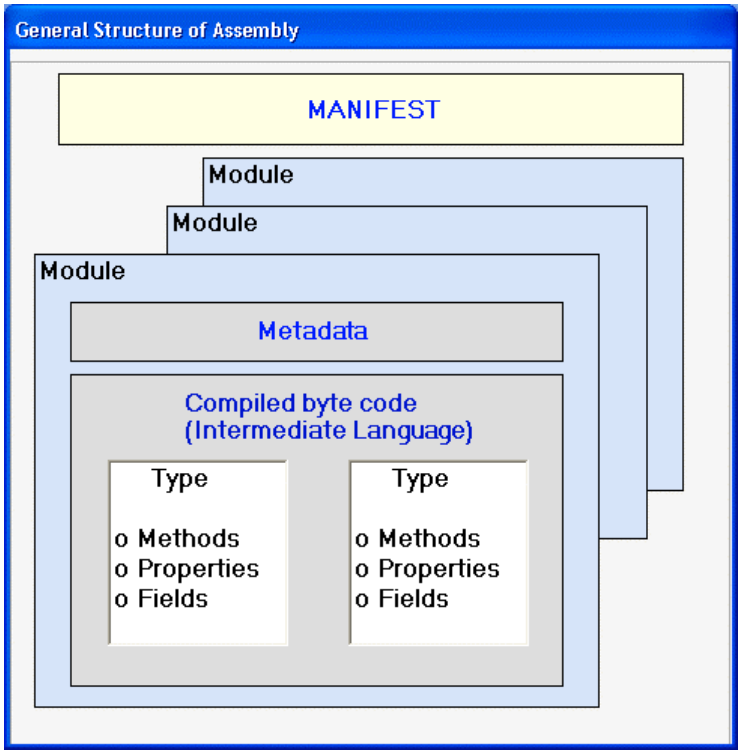
Khi chạy .NET application, nó sẽ được compiled bằng một **JIT (Just-In-Time) compiler** rất hiệu năng ra machine code để chạy. Điểm này giúp .NET application chạy nhanh hơn Java interpreted code trong Java Virtual Machine. Just-In-Time cũng có nghĩa là chỉ phần code nào cần xử lý trong lúc ấy mới được compiled. IL code chạy trong CLR được nói là **managed code**.

.NET code có thể chạy chung với ActiveX, nhưng code trong ActiveX được gọi là **unmanaged code**, tức là CLR không chịu trách nhiệm.

Ngoài việc allocation và management of memory, CLR còn giữ các refrerence đến objects và đỗ rác (handle **garbage collection**), tức là thâu lại các mảnh vụn memory khi chúng không cần dùng nữa. Trước đây, mỗi khi một DLL được loaded vào memory, system sẽ ghi nhận có bao nhiêu task dùng nó để khi task cuối cùng chấm dứt thì system unload DLL và trả lại phần memory nó dùng trước đây để system dùng cho chuyện khác. Chớ nếu allocate memory để dùng mà không nhớ dispose nó thì sẽ bị memory leak (rỉ), lần lần ta dùng hết memory, bị bắt buộc phải reboot

OS. Nhưng bây giờ .NET dùng một process độc lập để làm việc garbage collection. Cái phản ứng phụ của việc này là khi ta đã **Dispose** một Object rồi, ta vẫn không biết chắc chắn chừng nào nó mới thật sự biến mất. Vì garbage collector là một low priority process làm việc trong background, chỉ khi nào system memory gần cạn nó mới nâng cao priority lên. Dĩ nhiên, nếu muốn, ta có thể đòi hỏi system Dispose một Object ngay lập tức.

Assembly



.NET application xây dựng từ các assemblies. Mỗi **assembly** phải có một **manifest**. Có thể nó nằm riêng trong một file hay nằm bên trong một module. Manifest chứa những metadata sau đây:

- Tên và Version number của assembly
- Những assembly khác (kể cả version number của assembly) mà assembly này tùy thuộc vào để chạy

- Types (classes và members) mà assembly này cho xuất khẩu
- Assembly này đòi hỏi điều kiện an ninh nào (security permissions)

Manifest cho phép ta dùng hơn một version của assembly (tương đương với DLL trước đây) cùng một lúc. Từ đây không còn register DLL nữa. Thay vào đó, ta chỉ cần copy các assembly vào một subfolder /bin của chương trình chính.

Quyết định của bạn

Sau khi biết qua về .NET, câu hỏi bạn sẽ đặt ra là bạn có nên học lập trình trên .NET hay không. Nói chung, về lập trình có nhiều tôn giáo như VC++/Java, VB6, Delphi ..v.v.. Bạn có thể chọn giữa C# và VB.NET. Đối với VB6 programmers, học lập trình VB.NET sẽ mất một thời gian, nhưng không khó. Nên nhớ rằng .NET không phải chỉ cho ta các ngôn ngữ lập trình, mà cả một hệ thống triển khai phần mềm chú trọng vào mục tiêu hơn là cách thức. Các lợi ích .NET cống hiến cho bạn cách thực tiễn là:

- Kỹ thuật .NET sẽ hoành hành trên giang hồ trong từ 5 đến 10 năm tới.
- Tính trung bình, lập trình trong .NET sẽ tiết kiệm thì giờ cho bạn từ 25% đến 50% so với trước đây. Lý do là trong .NET bạn sẽ nghiên cứu để dùng component nào nhiều hơn là thật sự viết code. Hơn nữa, hầu như code nào bạn cần phải viết, bạn sẽ dùng nó lại trong tương lai. Và bảo trì .NET code thì lại càng khỏe hơn trước đây, vì chính bạn có viết bao nhiêu code (có thể bị bugs) đâu mà bảo trì.

Do đó, có lẽ trong tương lai .NET programmers chúng ta sẽ ngủ đến 10 giờ sáng mới thức, một ngày chỉ cần làm việc vài tiếng, rảnh rang để làm vườn, câu cá.

Trong bài tới ta sẽ học về Visual Studio.NET interface và bắt đầu viết thử một program Demo .

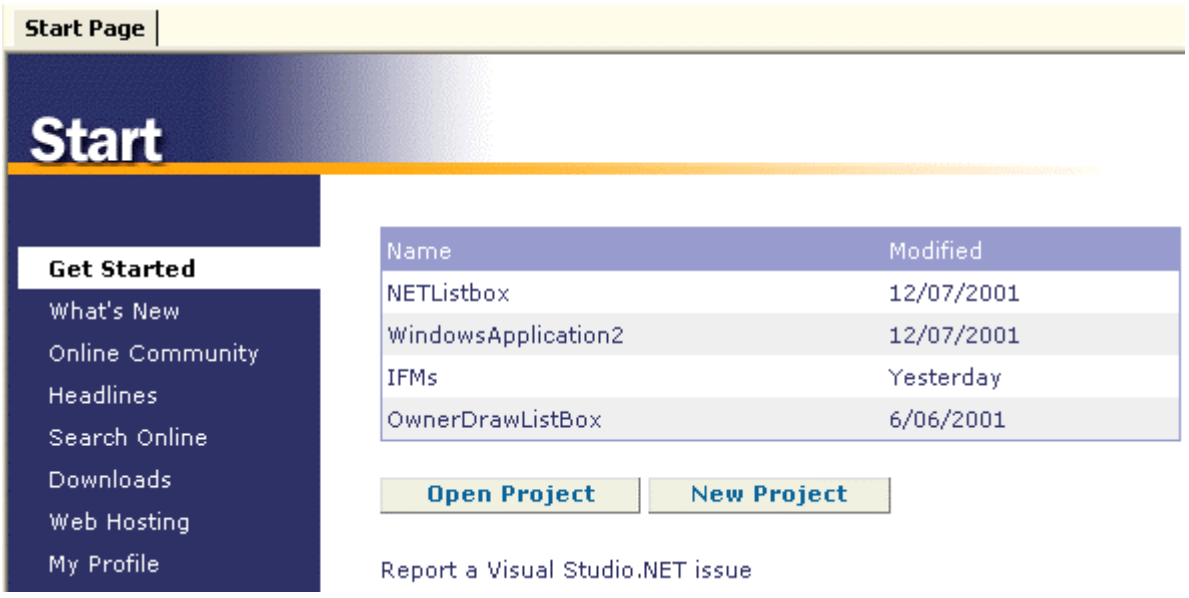
Bài 2
Visual Studio.NET

Visual Studio.NET

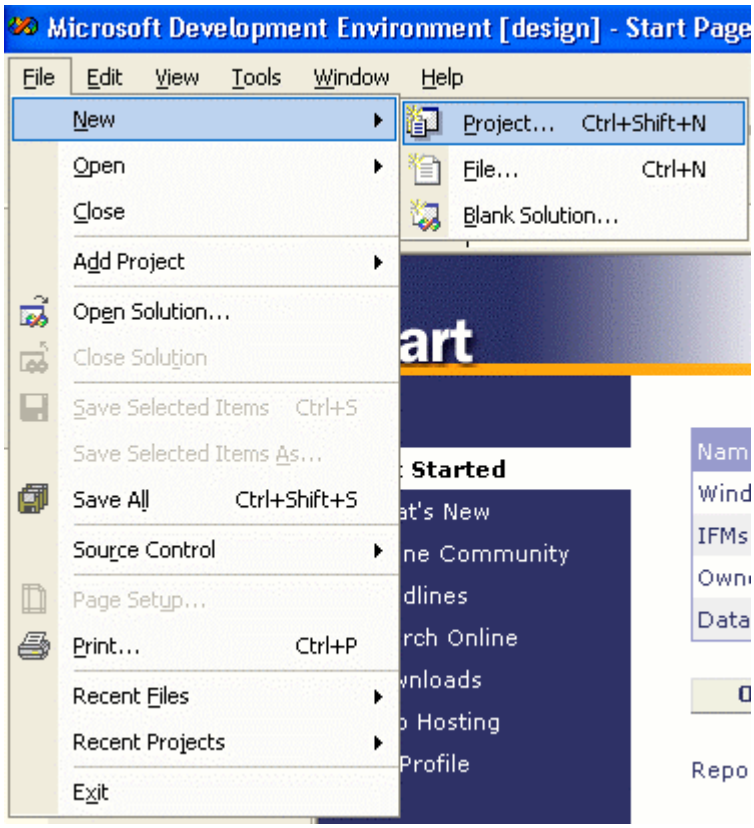
Để chạy VisualStudio.NET bạn cần phải Set Time của Windows lại trước cuối tháng 7,2001, eg: 1-July-2001.

Visual Studio.NET Beta 2 hiện nay có Service Pak 2, nó cho phép ta dùng Visual Studio.NET Beta 2 sau ngày 31-July-2001, tức là không có time-bomb. Nếu VS.NET version của bạn bị giới hạn về thời gian nói trên, từ trong VS.NET bạn có thể download Service Pak 2 để cài đặt bằng cách dùng IDE Menu Command **Help | Check for Updates**.

Có hai cách để bắt đầu một project mới trong VS.NET. Hoặc Click **New Project** trong trang Web **StartPage** như dưới đây:

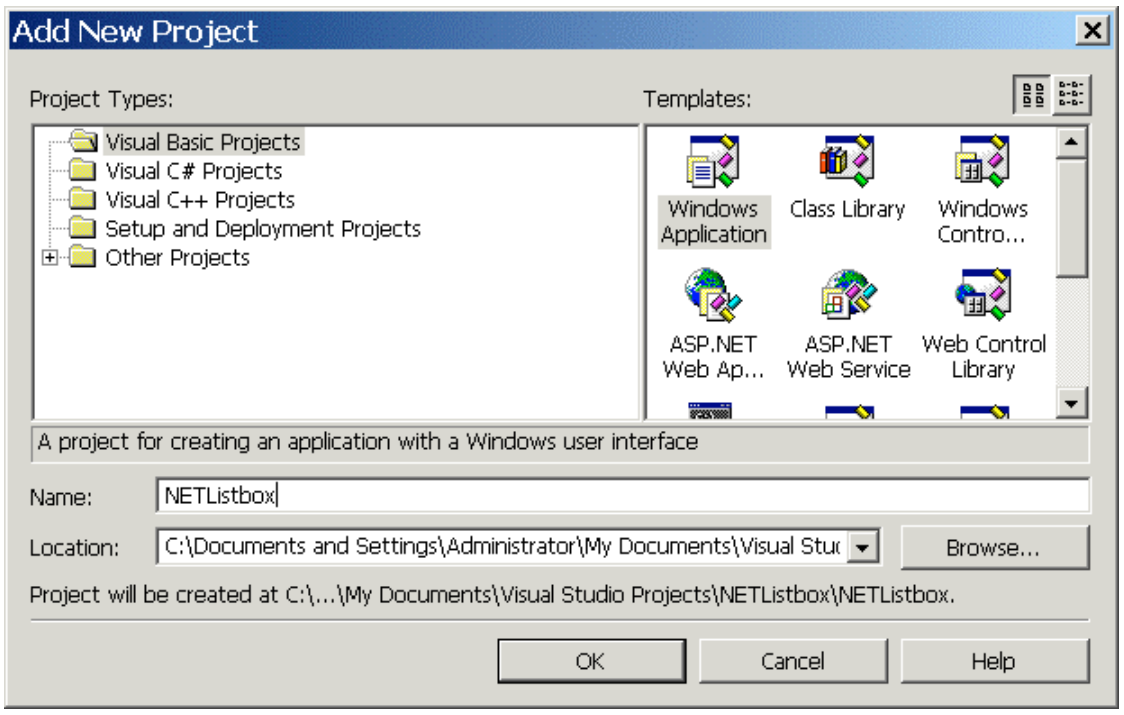


Hoặc dùng Menu command **File | New | Project** giống như trong VB6 IDE:



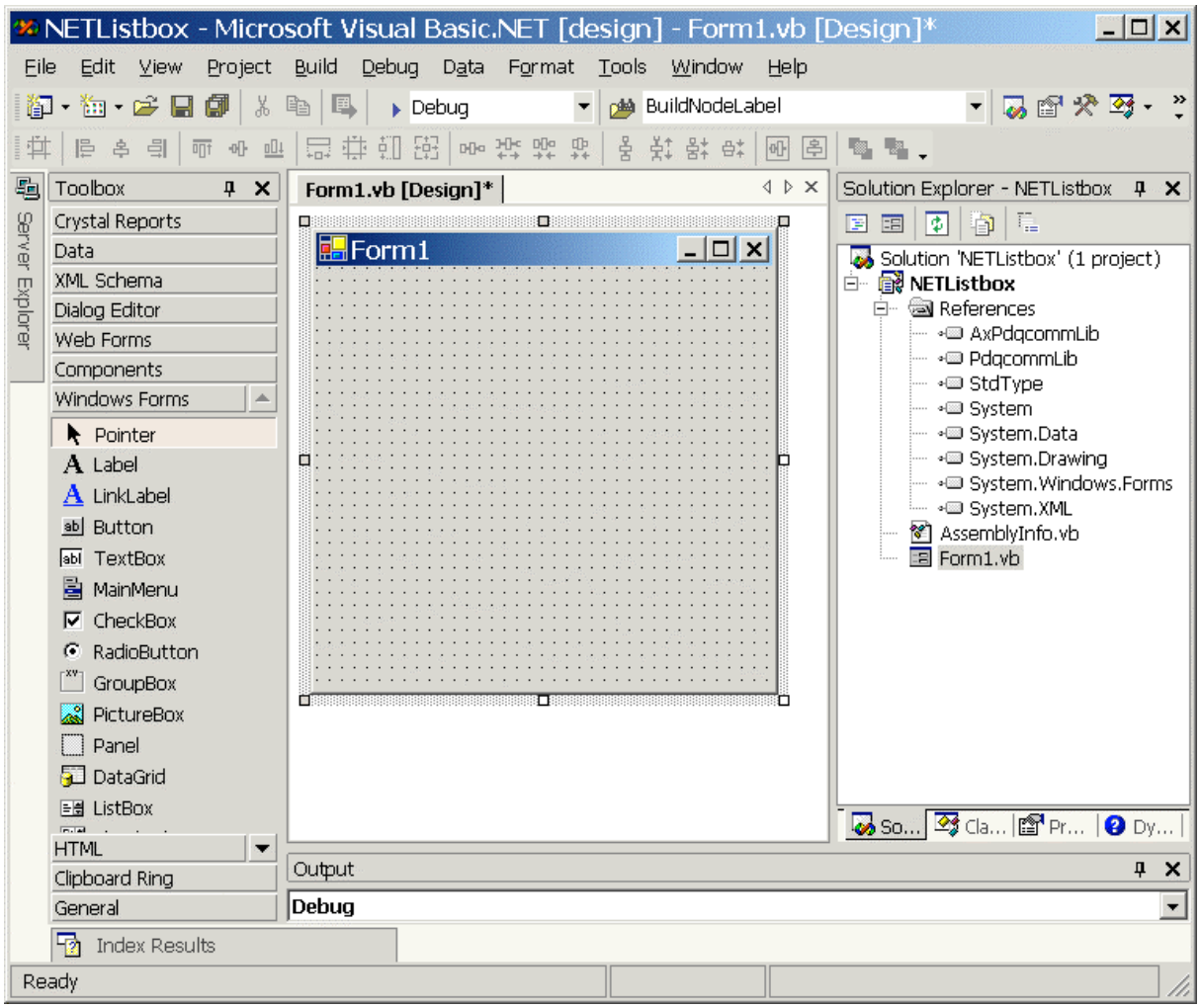
Khi **Add New Project**, **Name** sẽ là subfolder name của New Project. Bạn có thể chọn VB.NET, C# hay VC++.NET project. Trong tương lai chúng ta sẽ chỉ nhắm vào VB.NET và C# thôi. Ngoài ra SetUp and Deployment bây giờ là một loại project nằm trong IDE của Visual Studio.NET, ta không cần phải chạy riêng chương trình Package and Deployment bên ngoài VB6 IDE như trước đây.

Solution trong VS.NET có thể chứa hơn một Project và bao gồm tất cả những files bạn liệt ra là cần thiết cho Solution. Nếu một trong những files ấy bị thay đổi bên ngoài VS.NET, khi VS.NET khám phá ra nó sẽ load vào trong VS.NET cho bạn nếu bạn đồng ý.



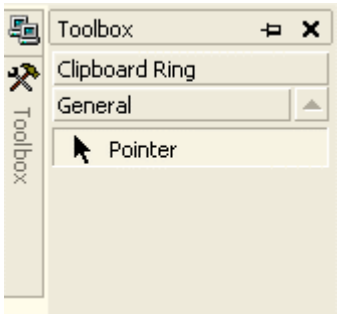
Để mở một Solution/Project có sẵn, bạn có thể click **link** của tên project trên trang **StartPage**, hay dùng Menu command **File | Open | Project** , hay Menu Command **File | Recent Projects**.

Visual Studio.NET IDE

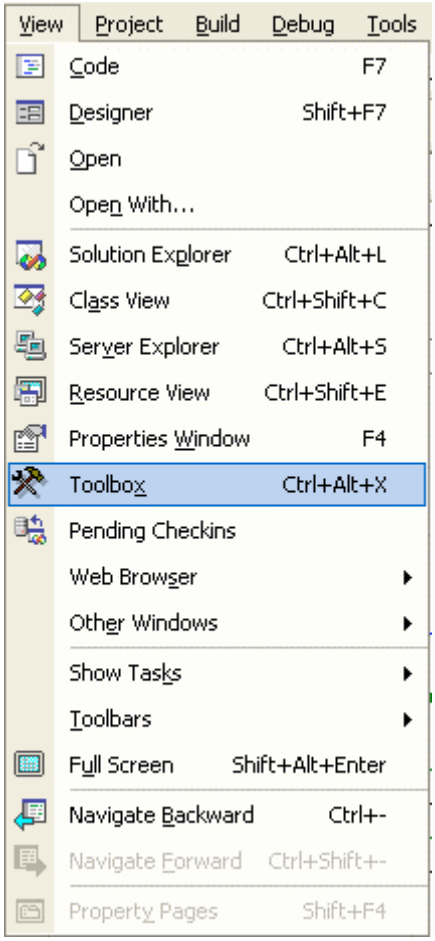


Giao diện của VS.NET có những đặc điểm giống như Delphi và Visual InterDev ở chỗ code được generated tự động, Windows nằm chung thành Tab set hay float khắp nơi, và hỗ trợ Solution rất thích hợp cho việc triển khai của cả đội.

Toolbox bên trái chứa Controls cho **Windows Forms, Web Forms, General Components, Data Components, HTML tags, XML Schema tools** v.v... Khi một Form đang hiển thị, click lên một button trong Toolbox để chọn Tool Set bạn cần. Toolbox ở trạng thái **Fixed displayed** (như trong hình trên) khi cây ghim phía trên đâm xuống. Bạn có thể click cây ghim cho nó nằm ngang và vertical Toolbox tab hiện ra bên trái. Lúc ấy, Toolbox ở trong trạng thái **Auto Hide** (hiện ra/rút vào) như dưới đây:



Bạn có thể gọi hầu hết các Windows hiển thị bằng cách dùng Menu commands **View**, **View | Other Windows** và **Debug | Windows**:

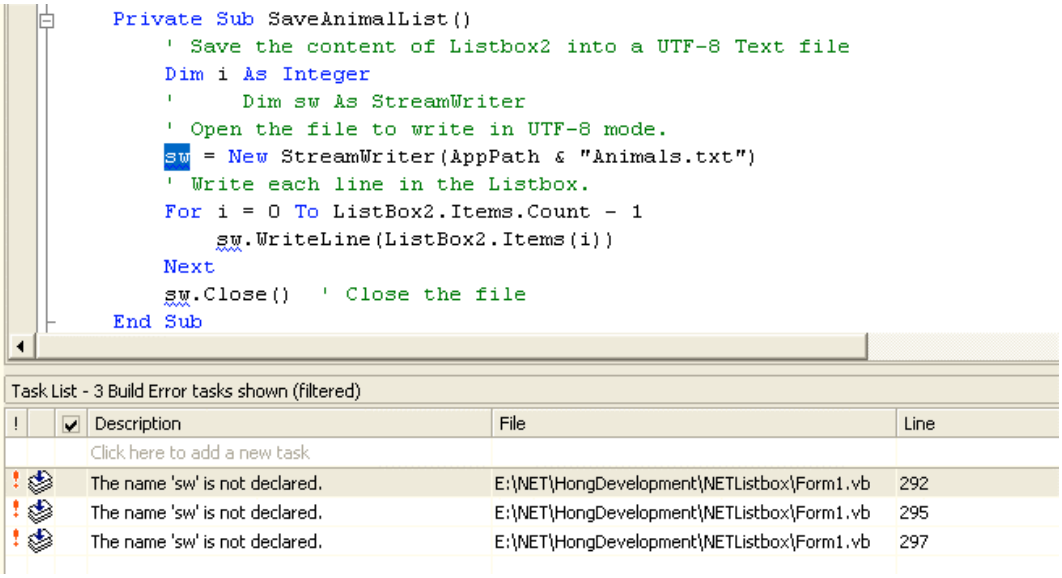


Các Windows bên phải có thể float, nằm chồng lên nhau thành những Tabs trong một Window set hay nằm cạnh nhau vertically tiled. Nắm Tittle bar để dời nguyên một Window set đi. Nắm Window Tab của một Window set để kéo chỉ một Window ra. Bạn có thể để chồng hai

Windows lại với nhau bằng cách nắm Tittle bar của một Window để chồng lên một Window khác. Thử nhích đi, nhích lại, trước khi buông Window ra để làm quen với kết quả.

Phía dưới có **Task List Window** để bạn giữ sổ sách về diễn tiến của dự án và quản lý cả đội. Khi bạn dùng menu command **Build | Build** để compile program, nếu có errors chúng sẽ được hiển thị trong Task List Window. Double click lên một hàng error để mang cursor đến chỗ gây ra error ấy trong code window.

Trong hình dưới đây, ta cố ý comment out hàng **Dim sw As StreamWriter**:



Ngoài ra, để Debug bây giờ bạn có **Output Window** để in ra các messages mà trong VB6 bạn dùng Immediate Window. Thí dụ trong VB6 bạn viết:

Debug.Print "Count=" & CStr(Count)

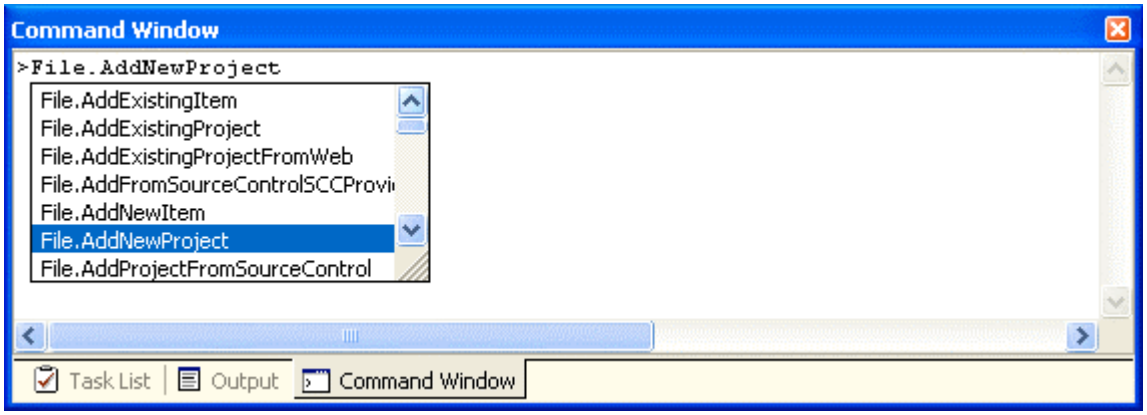
để in ra trong Immediate Window, thì trong VB.NET bạn có thể viết:

Console.WriteLine("Count= {0}", Count)

để in ra trong Output Window.

Dĩ nhiên bạn vẫn có thể tiếp tục dùng Immediate Window trong công tác Debug như trước đây trong VB6.

Chưa hết, VS.NET còn cho bạn **Command Window** để ta có thể enter những VS.NET commands để manipulate IDE, xử lý macros, .v.v Để hiển thị Window này bạn dùng menu command **View | Other Windows | Command Window**. Để tiện hơn, bạn drag title bar của nó để chồng lên tab bar của Task List và Output windows.



Nếu ta enter một command như:

File.AddNewProject

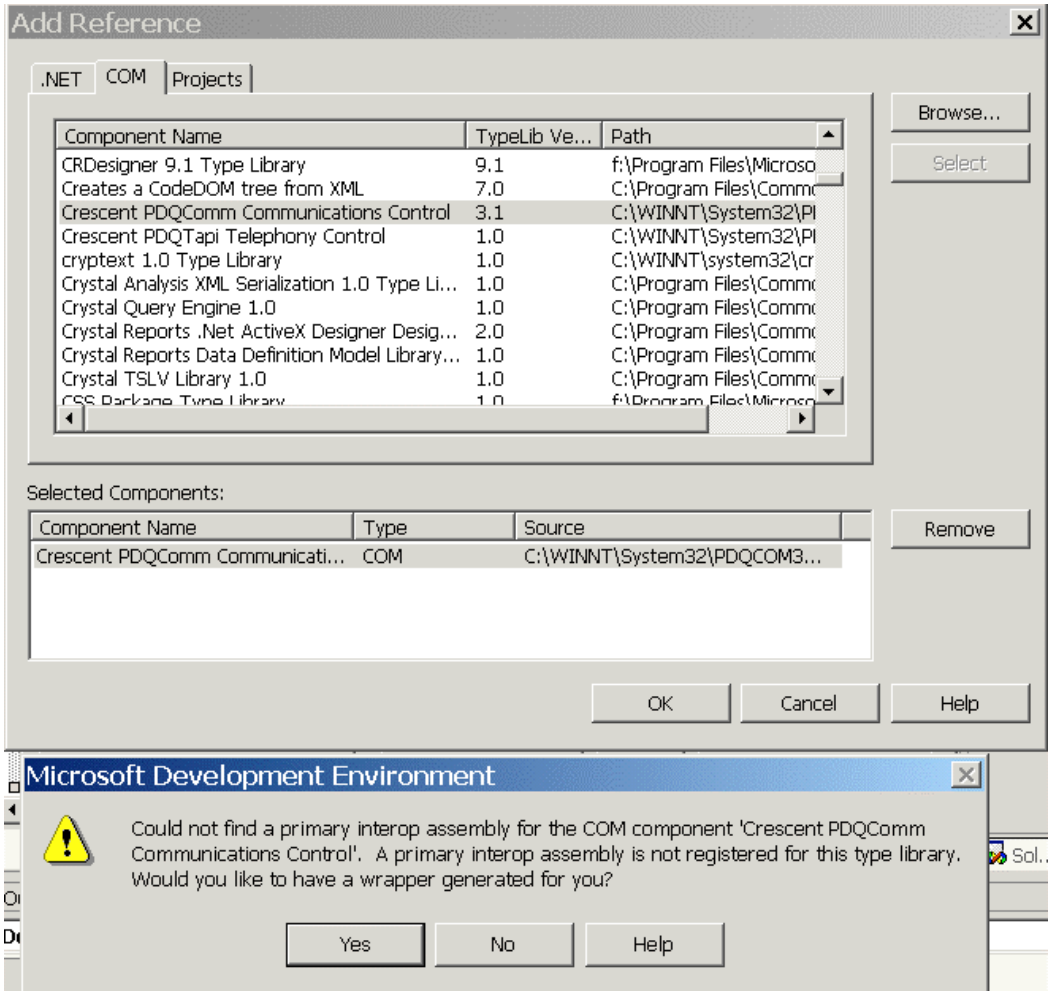
IDE sẽ hiển thị **Add New Project** dialog.

VS.NET hỗ trợ **Macro** để bạn có thể **record** và **playback** một chuỗi công tác.

Phương tiện **Integrated Debugging** cho ta **Debug Menu, Call Stack Window, Breakpoints Window** và **Watch and Value Display Windows**.

Thường thường bạn sẽ Add Reference các **.NET** components. Nhưng bạn cũng có thể dùng **ActiveX** (có sẵn trong VB6) trong .NET application bằng cách Add Reference **COM** (click Tab COM trên Add Reference Dialog). .NET sẽ gói ActiveX thành một NET component (click **Yes** trả

lời câu hỏi "Would you like to have a wrapper generated for you?"). Ngoài ra dùng Add Reference **Projects** để refer đến DLL của các User developed DLL.

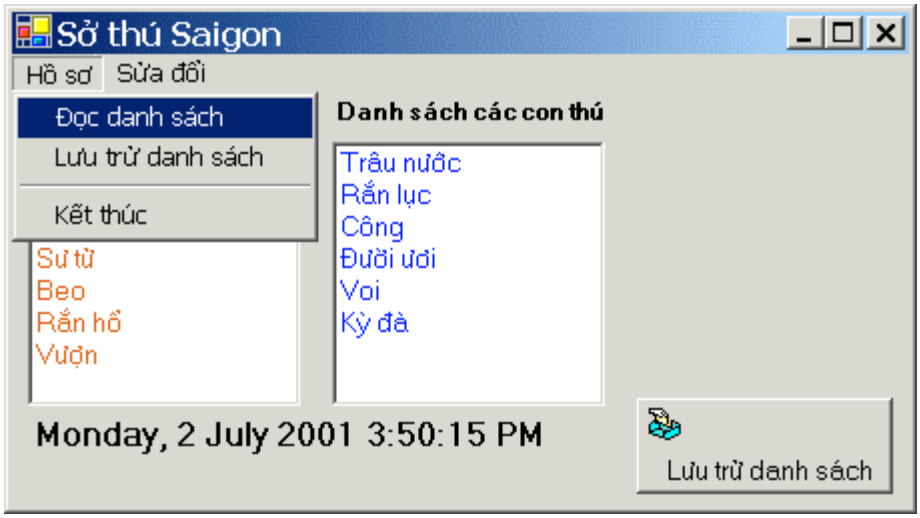


Demo Program

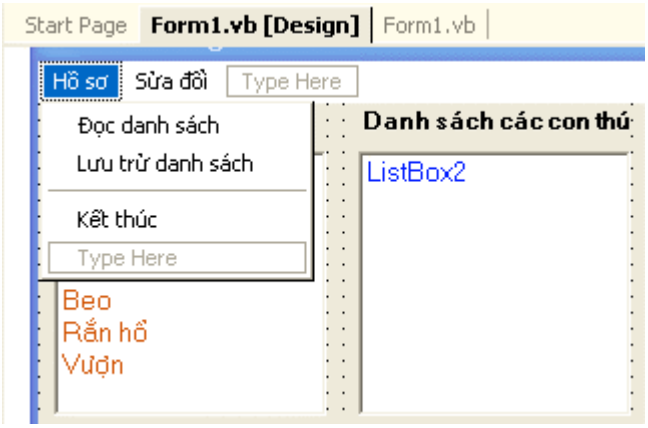
Trong chương trình biểu diễn này, ta dùng giao diện hầu như hoàn toàn bằng chữ Việt. Ta có thể đánh chữ Việt (**Unicode**) cho Title Bar, Menu, TextBox, ListBox .v.v..

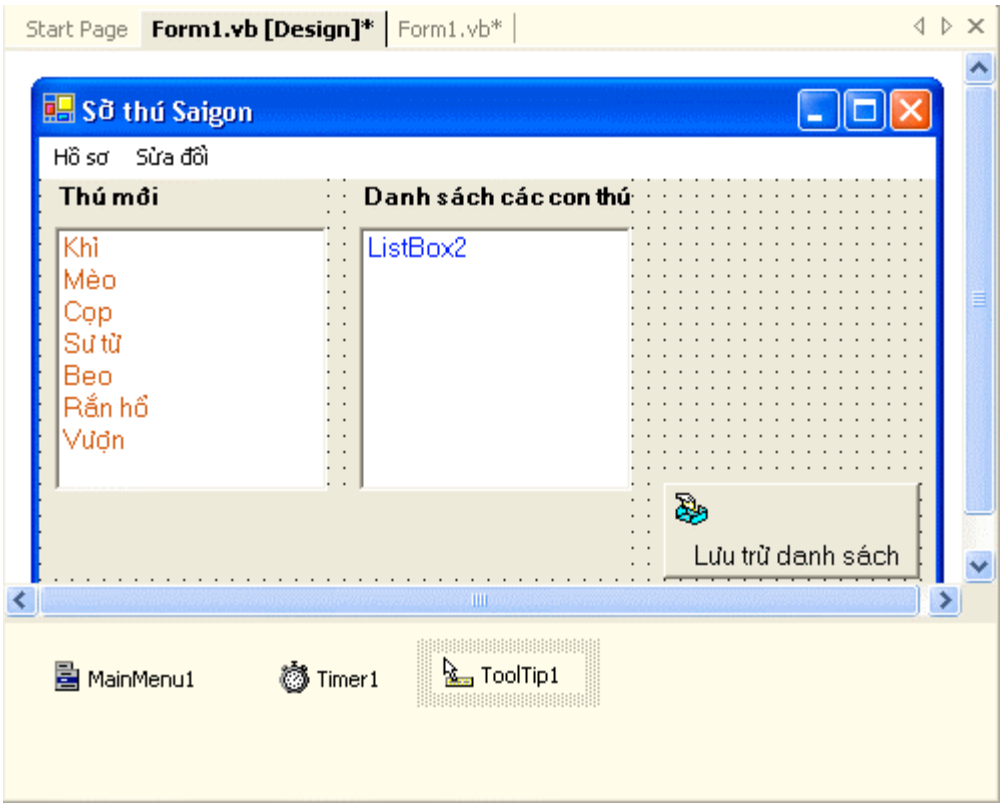
Nếu bạn cần một key input software cho chữ Việt hổ trợ Unicode thì download **VPS, Vietkey** hay **Unikey**.
Các documents như Form, XML .v.v. trong .NET project đều được Saved với **UTF-8 encoding.**

Trong program **Sở thú Saigon** này ta có hai Listboxes. Khi chạy, bạn có thể drag tên các loại thú từ Listbox1 (bên trái) để drop vào Listbox2 (bên phải). Phía dưới là một Label dùng để hiển thị ngày giờ. Có một button **Lưu trử danh sách** để ta save data trong Listbox2 vào file **animal.txt** trong subFolder **bin**. Ngoài ra bạn cũng có thể dùng Main menu Item **Đọc danh sách** để Load data từ file animal.txt vào Listbox2.

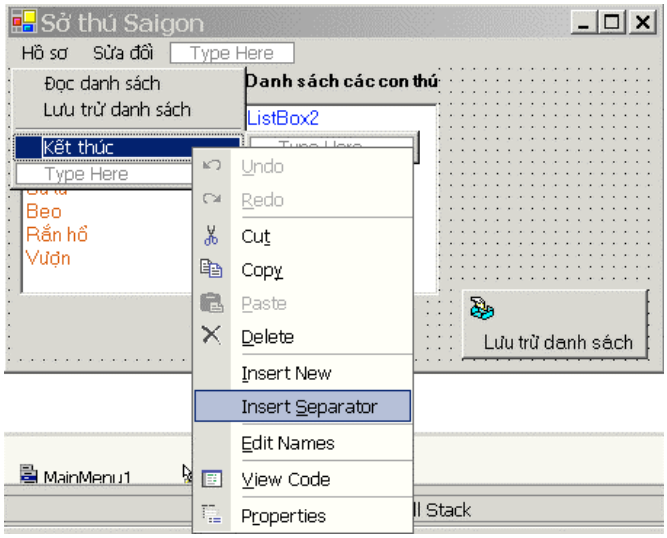


Những components không cần phải hiện ra lúc runtime như **Timer, Menu, Tooltip** .v.v. nằm trong một Component Tray (mâm) riêng. Muốn Edit MainMenu, click lên MainMenu1 icon rồi đánh trực tiếp vào MainMenu. Thêm các menuitems mới bằng cách đánh thẳng vào các chỗ có chữ **Type Here**. Lưu ý các Tabs bên trên Editing Area khi bạn mở nhiều forms.

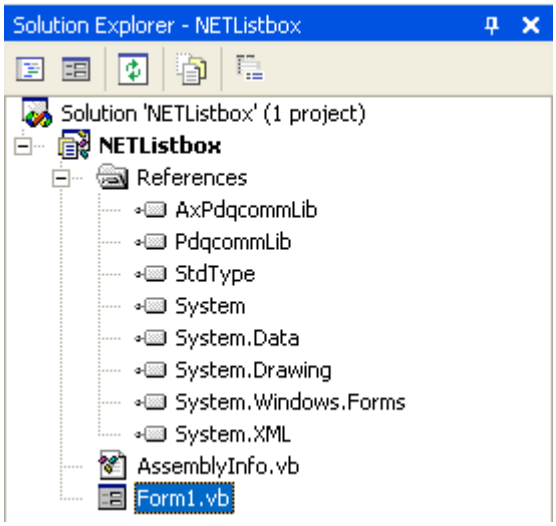




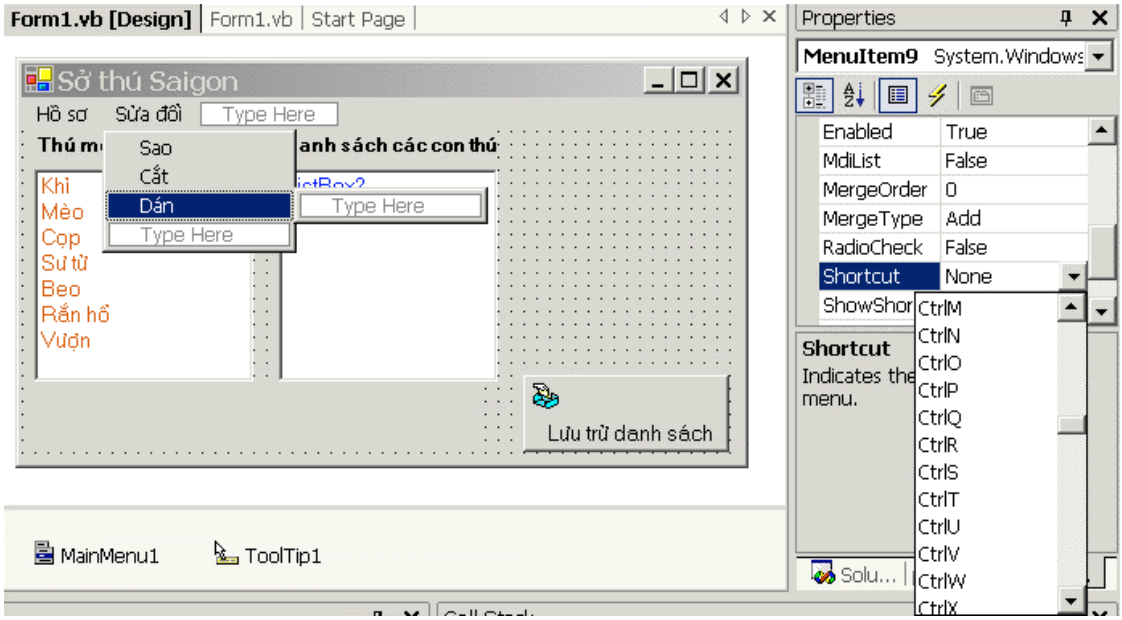
Edit MainMenu, insert một **Separator**



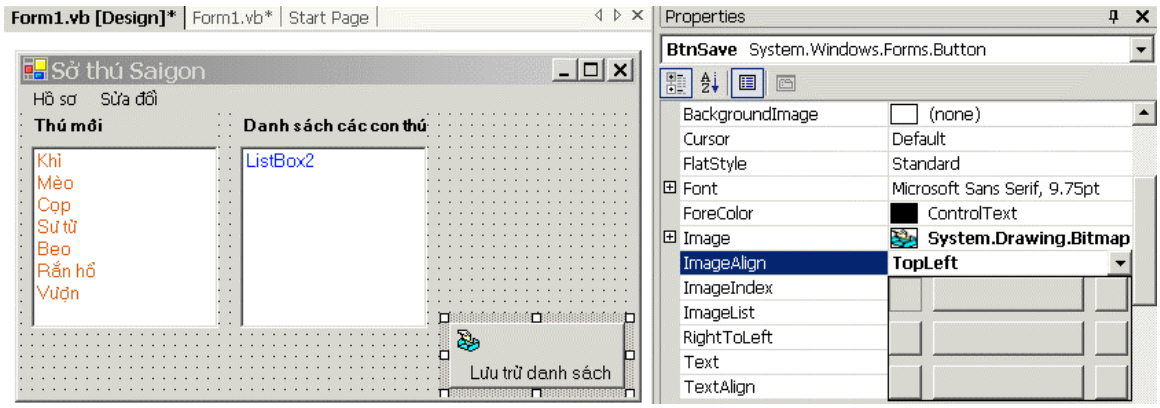
Chú ý danh sách các **References** được liệt kê trong **Solution Explorer**.



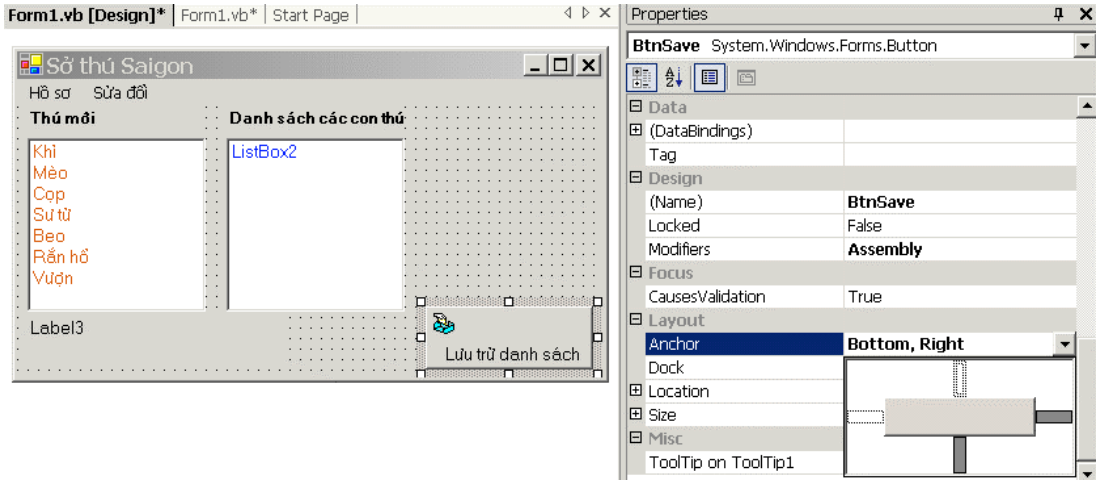
Edit Shortcut cho một menu item.



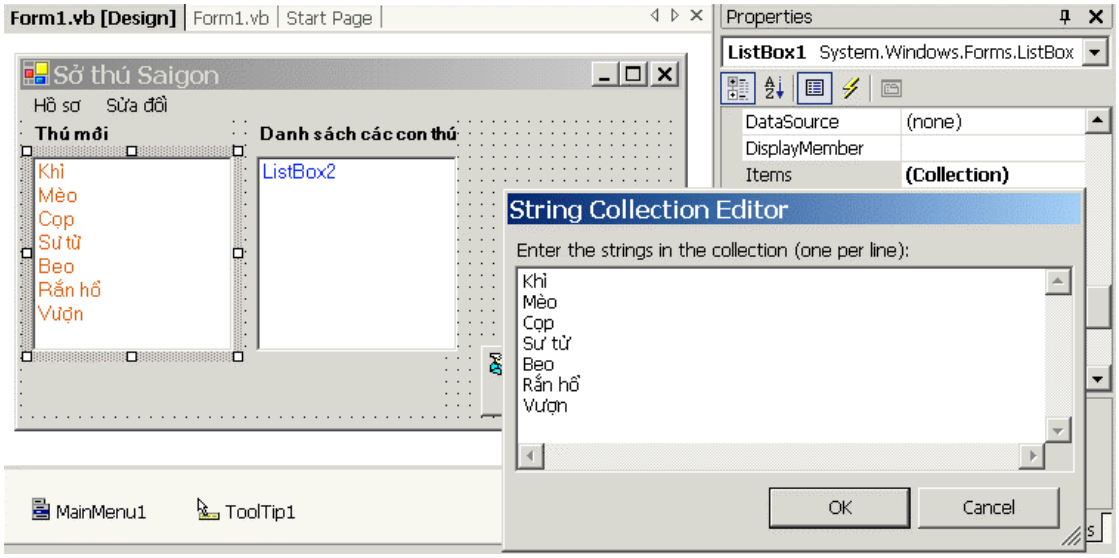
Button bây giờ chẳng những có thể chứa hình mà còn cho bạn chọn vị trí của hình trong button bằng **ImageAlign** nữa.



Bạn có thể **Anchor** một Button để nó dính vào một góc của form để khi form resizes thì Button chạy theo góc ấy của form. Ở đây ta click hai thanh **Dưới** và **Phải** cho chúng trở nên màu đen để chọn Anchor **Bottom** và **Right**.



Trong lúc thiết kế ta có thể edit các List items của một Listbox bằng cách mở property **Items Collection** ra và Edit vào một trang.



Walk Through Code

Trong .NET, các classes được chia ra thành nhiều nhóm như System.IO, System.XML, System.Data, System.Drawing ..v.v.. Ngoài ra trong mỗi nhóm lại còn chia thành những nhóm con, cháu như System.Windows.Forms, System.Windows.Collections, System.Windows.Diagnostics, .v.v.. Mặc dầu một khi đã **Project | Add Reference** các .NET components ấy ta có thể dùng chúng trong program

nhưng vẫn phải biên một tên dài như **System.IO.StreamReader** để tránh lẫn lộn. Để có thể viết tên class gọn hơn ta dùng **Imports** như **Imports System.IO**, sau đó ta chỉ cần viết **StreamReader** là đủ. Công việc Imports này được gọi là importing **Namespace** (của System.IO).

Tương tự như thế, để có thể tiếp tục dùng các Functions **Left, Right, Mid** của VB6 trong .NET ta có thể thêm câu **Imports VB6 = Microsoft.VisualBasic** ở đầu chương trình. Sau đó ta có thể viết:

AppPath = VB6.Left(AppPath, Pos)

Nguyên program chúng ta tại đây là **Public Class Form1**. Form1 thừa kế standard form class của .NET Framework nên ta declare:

Inherits System.Windows.Forms.Form

```
Imports System.IO

Imports VB6 = Microsoft.VisualBasic

Public Class Form1

Inherits System.Windows.Forms.Form

Dim AppPath As String
Private Sub MenuItem4_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles MenuItem4.Click

  End  ' Terminate the program

End Sub
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load

  ' Obtain the folder where this program EXE resides and initialise tooltip
```

```
    Dim AppPath As String

    Dim Pos As Integer

    ' Fetch full pathname of the EXE file

    AppPath = System.Reflection.Assembly.GetExecutingAssembly.Location

    ' Locate the last slash in the pathname string

    Pos = InStrRev(AppPath, "/")

    ' Extract the part up to the backslash

    AppPath = VB6.Left(AppPath, Pos)

    ' Initialise the tooltip for Listbox1

    ToolTip1.SetToolTip(ListBox1, "Xin nắm kéo tên một con thú qua Listbox bên phải")
End Sub
```

Hãy xem cách viết một Event Handler như:

```
Private Sub MenuItem4_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles MenuItem4.Click
```

Khác với VB6, nó có thêm các chữ Handles MenuItem4.Click ở phía cuối để nói handling event Click của MenuItem4. Mọi Event Handler đều được passed cho hai parameters: ByVal sender As System.Object và ByVal e As System.EventArgs. Parameter thứ nhất, sender, là Object chủ động chuyện RaiseEvent, còn e là Event có chứa nhiều dữ kiện khác nhau tùy theo tình huống. Khi edit code bạn có thể nhờ Intellisense của IDE giúp đỡ cho biết parameter e chứa những dữ kiện gì.

Trong .NET, vấn đề handling event không phải là một điều bí hiểm như trong VB6. Khi một control có thể RaiseEvent thì chẳng những một, mà

nhiều controls khác đều có thể Đăng ký (Register) để được Thông báo (Notified) khi Event ấy xãy ra.

Control RaiseEvent được gọi là Publisher (Nhà Xuất Bản), các controls muốn handle event được gọi là Subcribers (những Người Đặt Mua dài hạn) . Dĩ nhiên cách handle event của mỗi control đều khác nhau, dầu rằng được passed cho cùng hai parameters. Các EventHandlers này được gọi là Delegates (những Nhà Đại Diện lãnh trách nhiệm giải quyết một sự cố).

Do đó, ta có thể dùng vỏn vẹn một EventHandler để handle nhiều Event khác nhau, xuất phát từ nhiều Objects. Nói cho đơn giản ra, sau khi ta đã đăng ký một hay nhiều Delegates (tức là EventHandler Subs), thì khi Event xảy ra, các EventHandler Subs ấy sẽ được xử lý. Nếu bạn vẫn còn thấy khó hiểu thì hãy đọc thí dụ này. Tưởng tượng bạn làm biếng nấu ăn nên đặt nhà hàng giao cơm mỗi ngày đến tận nhà. Có hai cách để bạn nhận "gà-mên" cơm:

1. Người giao cơm sẽ để "gà-mên" cơm trước nhà, cạnh bên hộp thư. Khi đi làm về, bạn sẽ mang nó vô nhà. Trong trường hợp này bạn xử lý công việc khi nào tiện, tức là lúc về đến nhà. Cách này có điểm bất lợi là hôm nào bạn đi làm về trể thì cơm có thể bị thiêu vì trời nóng.

Bạn đưa chìa khóa nhà cho người giao cơm giữ. Khi giao cơm, người ấy sẽ tự động mở cửa vô nhà để "gà-mên" thẳng vào trong tủ lạnh. Trong cách này **Event GiaoCơm** được handled bằng EventHandler **Sub OpenDoorPutIntoFridge**, do người giao cơm xử lý, chớ không phải chính bạn.

Như thế, là Subscriber (người ăn cơm tháng giao tận nhà) bạn register EventHandler "Sub OpenDoorPutIntoFridge" với người giao cơm qua

việc đưa chìa khóa. Khi Event "GiaoCơm" xãy ra, người giao cơm tự động executes Sub OpenDoorPutIntoFridge. Kỹ thuật giao **AddressOf** Sub cho một Object khác để nó execute khi cần còn có tên là **CallBack**.

Do đó, ngay cả trong lúc runtime (không phải khi design), để Register EventHandler **Sub MenuItem4_Click** với system để handle Event Click của MenuItem4 ta có thể execute code:

AddHandler MenuItem4.Click, AddressOf MenuItem4_Click

Lưu ý cách ta dùng control ToolTip1 để register Tooltip Text với Listbox1. Ta có thể dùng chỉ một control ToopTip1 để register nhiều Tooltip Texts với những controls khác nhau như TextBox, ComboBox .v.v..

Bạn có thể thay thế hàng:

AppPath = VB6.Left(AppPath, Pos)

bằng

AppPath = AppPath.SubString(0,Pos)

Trong VB6, Visual components của một form được chứa dưới dạng Text diễn tả các controls rất dễ đọc ở ngay đầu form file, nhưng nó không phải là VB6 code.

Trong .NET, Visual components của một form được chứa dưới dạng code thật sự. Tức là, nếu không có VS.NET ta có thể dùng Notepad viết code như thế và sau khi compile, nó vẫn chạy y hệt như trong trường hợp ta dùng VS.NET. Điểm này giống như trong Java, ta có thể viết code bằng Notepad và dùng Command line để compile và link code file với các components khác.

Thí dụ như khi ta viết một VB.NET program đơn giản để chạy trên trong DOS Console, ta có thể compile nó như sau:

vbc /t:exe /r:system.dll mysource.vb

vbc là VisualBasic Compiler, **/t:** có nghĩa **target** tức là EXE để chạy trong DOS console. **/r:** có nghĩa **reference** đến DLL.

Nếu muốn chạy trong Windows, ta dùng:

vbc /t:winexe /r:system.dll /r:system.windows.forms.dll mysource.vb

Bình thường generated code được dấu trong **Region** để khỏi choán chỗ, hay khêu gợi chúng ta sửa đổi.

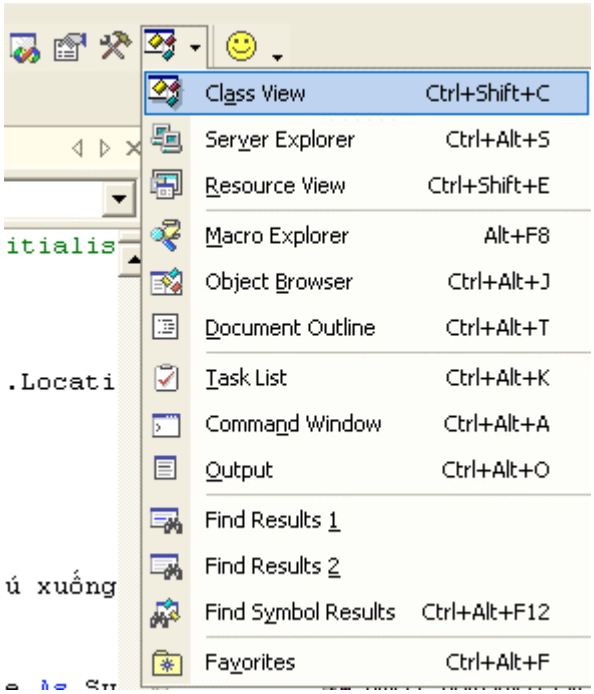
Click dấu + bên lề trái để mở một **Region** hay **Sub/Function**. Click dấu - để đóng lại.

Đôi khi ta cũng có thể Edit generated code, nhưng bạn nhớ backup code trước, để rủi form không thể hiển thị vì bị error, chỉ cho ta một trang giấy trắng, thì ta còn có đường restore.

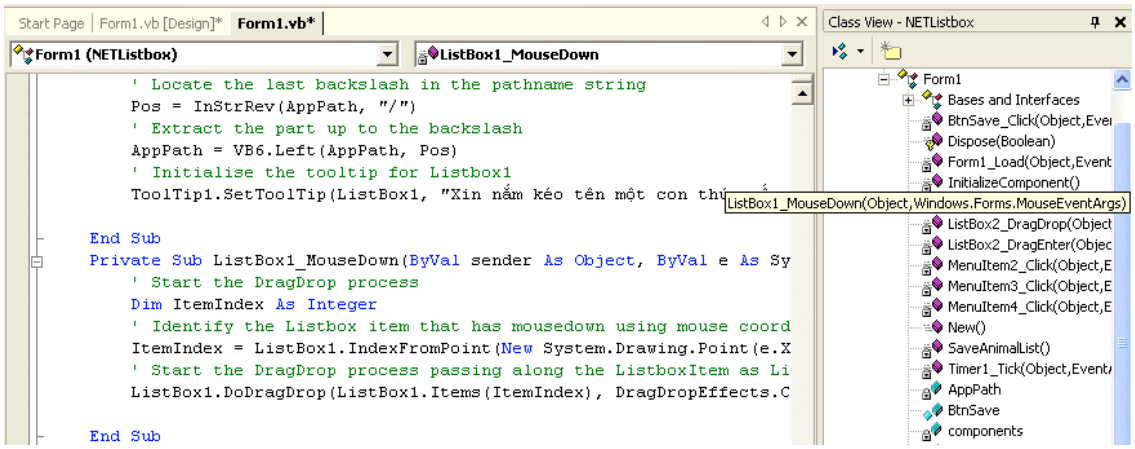
```
#Region " Windows Form Designer generated code "

    Public Sub New()
        MyBase.New()

        'This call is required by the Windows Form Designer.
        InitializeComponent()

        'Add any initialization after the InitializeComponent() call

    End Sub

    'Form overrides dispose to clean up the component list.
    Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)
        If disposing Then
            If Not (components Is Nothing) Then
                components.Dispose()
            End If
        End If
        MyBase.Dispose(disposing)
    End Sub
    Friend WithEvents ListBox1 As System.Windows.Forms.ListBox
    Friend WithEvents Label1 As System.Windows.Forms.Label
    Friend WithEvents ListBox2 As System.Windows.Forms.ListBox
    Friend WithEvents MainMenu1 As System.Windows.Forms.MainMenu
    Friend WithEvents MenuItem1 As System.Windows.Forms.MenuItem
```

Mỗi class đều có ít nhất một **Sub New**, gọi là **Constructor** (giống như **Class_Initialize** của VB6 class) và **Sub Dispose**, gọi là **Destructor**. Đó là hai Sub dùng để tạo ra và phá hủy Object. Vì Form1 thừa kế từ Standard Form nên trong Sub New **trước hết** phải gọi constructor **MyBase.New()** của cha nó, và trong Sub Dispose **sau hết** phải gọi destructor **MyBase.Dispose** của cha nó.

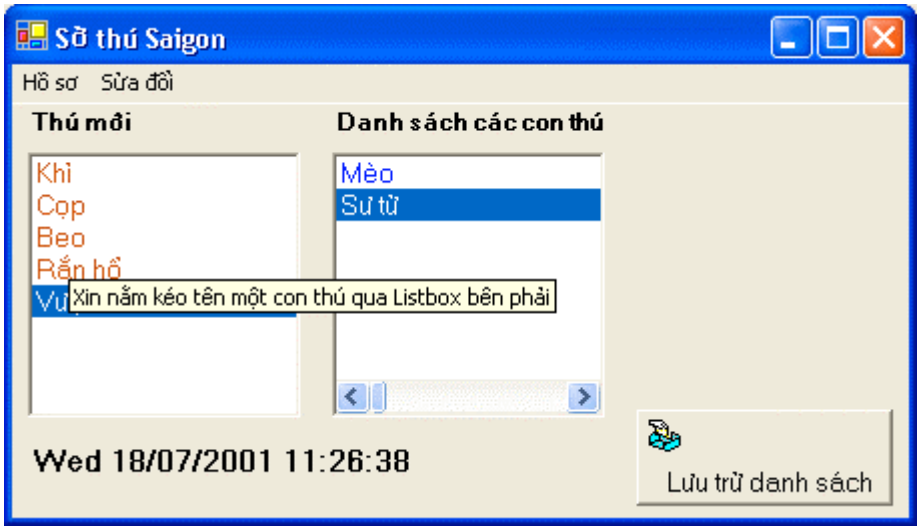
Mở Class View Window



Bạn có thể Navigate trong Code qua Class View. DoubleClick lên tên của Object hay Sub/Function trong Class View để mang cursor đến code của nó trong trang Edit.



Bi□u di□n DragDrop



Code của DragDrop, lưu ý ta phải viết thêm **Sub ListBox2_DragEnter** để handle Event DragEnter.

```
Private Sub ListBox1_MouseDown(ByVal sender As Object, ByVal e As
System.Windows.Forms.MouseEventArgs) Handles
ListBox1.MouseDown

  ' Start the DragDrop process

  Dim ItemIndex As Integer
```

```
    ' Identify the Listbox item that has mousedown using mouse coordinates
    ItemIndex = ListBox1.IndexFromPoint(New System.Drawing.Point(e.X, e.Y))
    ' Start the DragDrop process passing along the ListboxItem as
ListBox1.Items(ItemIndex)
    ListBox1.DoDragDrop(ListBox1.Items(ItemIndex),
DragDropEffects.Copy Or DragDropEffects.Move)
End Sub

Private Sub ListBox2_DragEnter(ByVal sender As Object, ByVal e As
System.Windows.Forms.DragEventArgs) Handles ListBox2.DragEnter
    ' Apply the copy effect
    ' AND remember to set the property Allow Drop of Listbox2 to TRUE
    If (e.Data.GetDataPresent(DataFormats.Text)) Then
        e.Effect = DragDropEffects.Copy
    Else
        e.Effect = DragDropEffects.None
    End If
End Sub

Private Sub ListBox2_DragDrop(ByVal sender As Object, ByVal e As
System.Windows.Forms.DragEventArgs) Handles ListBox2.DragDrop
    Dim LItem As String
```

```
    ' Obtain the Source ListItem String
    LItem = e.Data.GetData(DataFormats.Text).ToString
    ' Add it to Listbox2
    ListBox2.Items.Add(LItem)
    ' Remove the Item from Listbox1
    ListBox1.Items.RemoveAt(ListBox1.FindString(LItem))
End Sub
```

Khi Load data vào Listbox ta dùng **StreamReader** để Open một File as Input.

Khi Save data của Listbox vào một Text file ta dùng **StreamWriter** để Open một File as Output (hay Append nếu ta cho thêm Option Append=True):

```
Private Sub MenuItem2_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles MenuItem2.Click
    ' Read the list of animals from a text file into Listbox2
    Dim sr As StreamReader
    Dim Pos As Integer
    Dim TStr As String
    ListBox2.Items.Clear()  ' Clear Listbox2
    ' Use a StreamReader to open the UTF-8 file to read.
    sr = New StreamReader(AppPath & "animals.txt")
    ' Read each line in the file.
    ' When the end of the file is reached, return the value "-1".
    Dim x As String
```

```
  While sr.Peek <> -1
    x = sr.ReadLine()  ' Read a line
    ListBox2.Items.Add(x)  ' Add it to Listbox2
  End While
  sr.Close()  ' Close the file
End Sub

Private Sub SaveAnimalList()
  ' Save the content of Listbox2 into a UTF-8 Text file
  Dim i As Integer
  Dim sw As StreamWriter
  ' Open the file to write in UTF-8 mode, using a StreamWriter.
  sw = New StreamWriter(AppPath & "Animals.txt")
  ' Write each line in the Listbox.
  For i = 0 To ListBox2.Items.Count - 1
    sw.WriteLine(ListBox2.Items(i))
  Next
  sw.Close()  ' Close the file
End Sub
```

Ta hiển thị ngày và giờ bằng cách dùng **Timer1** và Shared Function **DateTime.Now** formated bằng hai Functions có sẵn **ToLongDateString** và **ToLongTimeString**.

```
Private Sub Timer1_Tick(ByVal sender As Object, ByVal e As System.EventArgs) Handles
Timer1.Tick
  ' Display Date and Time every half a second
  Label3.Text = DateTime.Now.ToLongDateString & " " & DateTime.Now.ToLongTimeString
End Sub
```

Bạn cũng có thể hiển thị ngày giờ trong format khác bằng cách viết:

Label3.Text = DateTime.Now.ToString("ddd dd/MM/yyyy hh:mm:ss")

để có: WED 18/07/2001 09:16:42

Để ý trong Format string ta dùng **MM** cho Month và **mm** cho Minute.

Bài 3
Những khác biệt giữa VB.NET với VB6

VB.NET, còn gọi là VB7, chẳng qua là C# viết theo lối Visual Basic. Nay VB7 đã hoàn toàn là **Object Oriented**, tức là cho ta dùng lại (**reuse**) classes/forms theo cách thừa kế thật thoải mái, nên nó khác VB6 nhiều lắm.

Dầu vậy, đối với VB6 programmers học VB.NET không khó. Lý do là VB.NET không cho thêm nhiều từ mới (**reserved words**). Nói chung các ý niệm mới trong VB.NET đều dễ lĩnh hội, nhất là khi đem ra áp dụng cách thực tế. Đó là nhờ Microsoft vẫn giữ nguyên tắc **dấu và làm sẵn** (của VB6) những gì rắc rối phía sau sân khấu, để ta có thể tập trung vào việc tìm kiếm một giải pháp, thay vì quá bận tâm vào cách thức làm một việc gì. Chính nguyên tắc ấy đã giúp Microsoft chiêu mộ được 3 triệu VB6 programmers trên khắp thế giới. VB.NET cống hiến cho VB programmers một công cụ rất hữu hiệu để dùng cho mọi hoàn cảnh, từ database, desktop, distributed, internet cho đến real-time hay mobile (pocket PC).

Những ưu điểm (features) của VB.NET đến từ chức năng của **.NET Framework**. Nó mang đến phương tiện lập trình cho mạng cách Object Oriented như XML, Remoting, Streaming, Serialisation, Threading .v.v... Những thứ này tuy lạ nhưng không khó học, ngược lại sẽ tiết kiệm rất nhiều thì giờ.

Mặc dầu VB6 là một ngôn ngữ lập trình trưởng thành và hiệu năng, chắc chắn hãy còn tồn tại trong nhiều năm nữa, nhưng học thêm VB.NET là một đầu tư tương đối ít tốn kém và đảm bảo huê lợi gấp bao nhiêu lần trong hàng thập niên tới.

Trong bài này ta sẽ bàn về những điểm khác nhau giữa VB6 và VB.NET từ quan điểm ngôn ngữ lập trình. Trong một bài khác ta sẽ bàn về những chức năng Đối tượng (Object Oriented) của VB.NET.

Namespaces

Namespaces là một cách đặt tên để giúp sắp đặt các Classes ta dùng trong program một cách thứ tự hầu dễ tìm kiếm chúng. Tất cả code trong

.NET, viết bằng VB.NET, C# hay ngôn ngữ nào khác, đều được chứa trong một namespace.

Điểm này cũng áp dụng cho code trong .NET system class libraries. Chẳng hạn, các classes của WinForms đều nằm trong System.Windows.Forms namespace. Và các classes dùng cho collections như Queue, Stack, Hashtable .v.v.. đều nằm trong System.Collections namespace.

Tất cả code ta viết trong program của mình cũng đều nằm trong các namespaces.

Trước đây trong VB6, mỗi khi nhắc đến một Class trong một COM tên **CompName** ta viết **CompName.classname** (còn gọi là PROGID) , tức là cũng dùng một dạng namespace.

Tuy nhiên phương pháp này có một vài giới hạn:

- Địa chỉ của class bị buộc cứng vào component đang chứa nó.
- Những classes không nằm trong một COM component thì không có "namespace".
- Cách gọi tên PROGID chỉ có một bậc thôi, không có bậc con, bậc cháu.
- Tên của Component luôn luôn có hiệu lực trên khắp cả computer.

Namespaces trong .NET khắc phục được mọi giới hạn nói trên trong VB6.

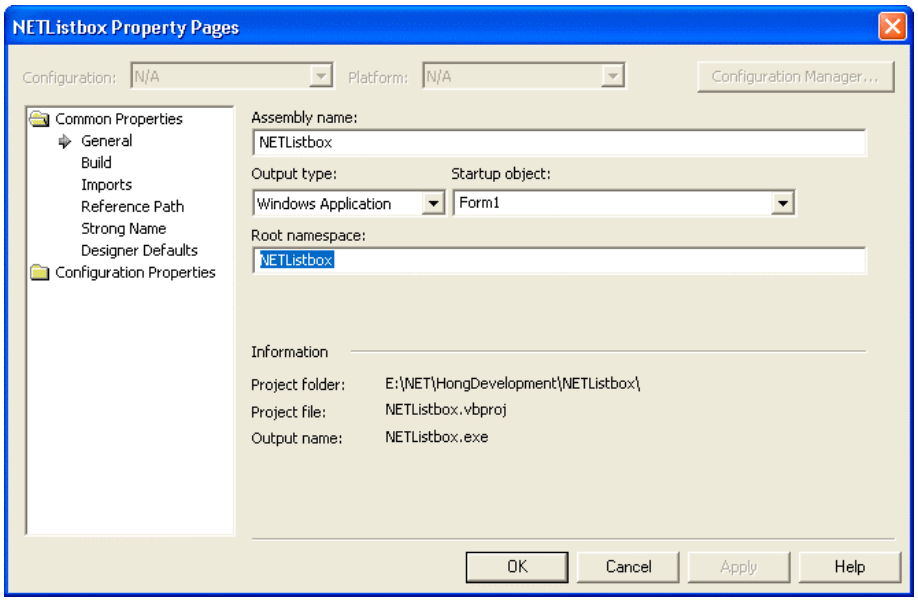
Nhiều **assemblies** có thể nằm trong cùng một namespace, nghĩa là classes tuyên bố trong các components khác nhau có thể có chung một namespace. Điều này cũng áp dụng xuyên qua các ngôn ngữ, giúp cho một class viết trong VB.NET có thể nằm trong cùng một namespace với một class viết trong C#, chẳng hạn.

Hơn nữa, trong một assembly có thể có nhiều namespaces, dù rằng thông

thường ta chỉ dùng một namespace duy nhất cho tất cả các classes trong ấy.

Nhớ là một assembly trong .NET thì đại khái tương đương với một COM component. Tất cả code trong .NET đều nằm trong những assemblies.

By default, tên của project được dùng làm namespace. Nếu bạn right click lên project name **NETListbox** trong Solution Explorer của program Demo, rồi chọn **Properties** trong popup menu, IDE sẽ hiển thị Property Pages dialog như dưới đây:



Bạn thấy **Root namespace** của project là **NETListbox**. Bạn có thể thay đổi tên namespace ấy nếu bạn muốn.

Namespaces có thể được phân chia thứ bậc giống như Folders trong một File Directory. Nó sẽ giúp user sắp đặt các classes theo đúng nhóm cho trong sáng và dễ đọc. Thí dụ bạn đang viết một program cho một hảng sản xuất, bạn sẽ dùng namespace **NhàSảnXuất** ở root level. Bên trong namespace ấy bạn sẽ tạo thêm các nhánh của chương trình như:

- **NhàSảnXuất.TồnKho**
- **NhàSảnXuất.SảnPhẩm**
- **NhàSảnXuất.KếToán.ChiPhí**
- **NhàSảnXuất.KếToán.ThuNhập**

Như thế ta đã định nghĩa một base namespace tên **NhàSảnXuất**, với những namespaces con, cháu bên trong, mỗi namespace có chứa classes, modules, enums, structures và các namespaces khác.

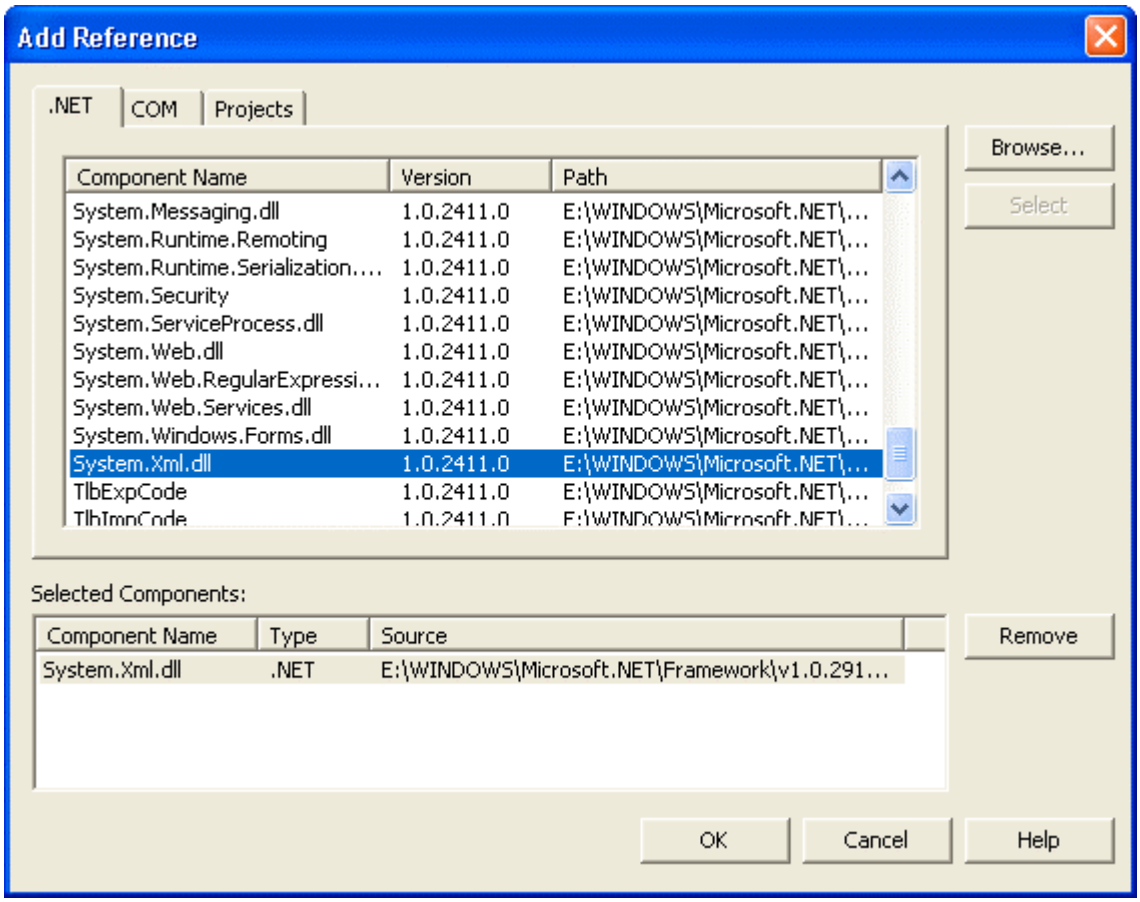
Mỗi namespace chứa những phần của code thích hợp cho nó trong program nói chung. Trong File Directory, ta có thể có hai files dù mang cùng tên nhưng nằm trong hai folders khác nhau. Giống như vậy, trong .NET ta có thể có hai classes có cùng một tên nhưng nằm trong hai namespaces khác nhau. Đó là vì khi ta viết tên của một class với cả namespace của nó thì có thể phân biệt với một class khác với cùng tên.

Local và Global Namespaces

Khác với COM components với "namespace" của chúng áp dụng cho khắp cả computer, namespaces của .NET thông thường là **Local**, chỉ có application program của nó thấy mà thôi. .NET cũng hổ trợ **Global** namespace, nhưng phải được ký tên (digitally signed) và đăng ký với .NET runtime để chứa nó trong global assembly cache. Công việc làm một namespace Global rắc rối như thế để giảm thiểu trường hợp ta trở về tình trạng **DLL hell** trước đây.

Dùng Namespaces

Ta có thể dùng namespaces bằng cách nói thẳng ra (explicitly) với nguyên tên (Direct Addressing) hay hàm ý (implicitly) với Import keyword. Nhưng điều tiên quyết là ta phải reference cái assembly chứa namespace mà ta muốn dùng. Ta thực hiện việc ấy với Menu command **Project | Add References**. Khi **Add References** dialog hiện ra, chọn Tab **.NET** cho standard .NET components hay Tab **Projects** cho DLL của một .NET project khác , highlight DLL bạn muốn rồi click **Select** button, đoạn click **OK**.



Chẳng hạn ta muốn read và write từ stdio (cái console input/output stream). Cái namespace ta cần sẽ là **System.Console**. Trong cách **Direct Addressing** ta sẽ code như sau để viết hàng chữ "Chào thế giới":

```
System.Console.WriteLine ("Hello world!")
```

Nếu ta dùng **Import keyword** bằng cách nhét vào câu **Imports System.Console** ở đầu code module, ta có thể code gọn hơn:

```
WriteLine ("Hello world!")
```

Dưới đây là một số namespaces thông dụng:

Namespace	**Chức năng**	**Classes điển hình**
System.IO	Đọc/Viết files và các data streams khác	FileStream, Path, StreamReader, StreamWriter
System.Drawing	Đồ họa	Bitmap, Brush, Pen Color, Font, Graphics
System.Data	Quản lý data	DataSet, DataTable, DataRow, SQLConnection, ADOConnection

System.Collection	Tạo và quản lý các loại collections	ArrayList, BitArray, Queue, Stack, HashTable
System.Math	Tính toán	Sqrt, Cos, Log, Min
System.Diagnostics	Debug	Debug, Trace
System.XML	Làm việc với XML, Document Object Model	XMLDocument, XMLElement, XMLReader, XMLWriter
System.Security	Cho phép kiểm soát an ninh	Cryptography, Permission, Policy

Aliasing Namespaces (dùng bí danh)

Khi hai namespaces trùng tên, ta phải dùng nguyên tên (kể cả gốc tích) để phân biệt chúng. Điển hình là khi ta dùng những namespaces liên hệ đến VB6 như **Microsoft.Visualbasic**. Thay vì code:

```
Microsoft.Visualbasic.Left ( InputString,6)
```

ta tuyên bố:

```
Imports VB6= Microsoft.Visualbasic
```

Sau đó ta có thể code:

```
VB6.Left ( InputString,6)
```

Dùng Namespaces keyword

Trong thí dụ về program cú Root Namespace là NhàSảnXuất như núi trờn, nếu ta muốn đặt ra một namespace con là TồnKho, ta phải dựng NameSpace keyword trong code như sau:

```
' Root Namespace là NhàSảnXuất

Namespace TồnKho


  Class PhòngLạnh

   ' Code cho Phòng Lạnh

  End Class


End Namespace
```

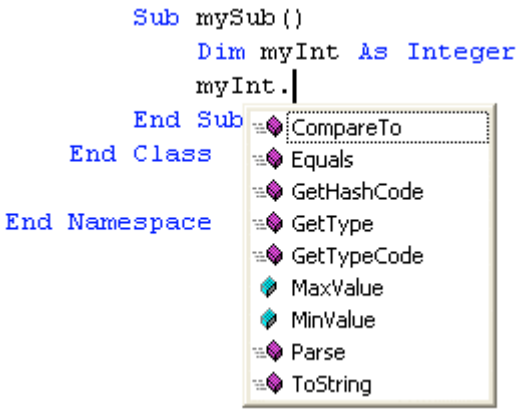
Bây giờ muốn nói đến class PhòngLạnh bên trong namespace TồnKho ta sẽ code như sau:

NhàSảnXuất.TồnKho.PhòngLạnh

Thay đổi trong Data Types

Tất cả đều là Object

Một thay đổi lớn cho Data Type của VB.NET, là những variables dùng Data Type địa phương như Integer, Single, Boolean,.v.v.. đều là những Objects. Chúng đều được derived (xuất phát) từ Class căn bản nhất tên **Object** trong VB.NET. Nếu bạn thử dùng Intellisense để xem có bao nhiêu Functions/Properties một Object loại Integer có, bạn sẽ thấy như dưới đây:



Trong .NET, Integer có bốn loại: **Byte** (8 bits, không có dấu, tức là từ 0 đến 255), **Short** (16 bits, có dấu cộng trừ, tức là từ -32768 đến 32767), **Integer** (32 bits, có dấu) và **Long** (64 bits, có dấu). Như vậy Integer bây giờ tương đương với Long trong VB6, và Long bây giờ lớn gấp đôi trong VB6.

Floating-Point Division (Chia số nổi)

Việc chia số nổi (Single, Double) trong VB.NET được làm theo đúng tiêu chuẩn của IEEE. Do đó nếu ta viết code như sau:

```
Dim dValueA As Double

Dim dValueB As Double

dValueA = 1

dValueB = 0
```

```
Console.WriteLine(dValueA / dValueB)
```

Trong VB6 ta biết mình sẽ gặp **Division by Zero** error, nhưng ở đây program sẽ viết trong Output Window chữ **Infinity** (vô cực). Tương tự như vậy, nếu ta viết code:

```
Dim dValueA As Double

Dim dValueB As Double

dValueA = 0

dValueB = 0

Console.WriteLine(dValueA / dValueB)
```

Kết quả sẽ là chữ **NaN** (Not a Number) hiển thị trong Output Window.

Thay thế Currency bằng Decimal

VB.NET dùng Decimal data type với 128 bits để thay thế Currency data type trong VB6. Nó có thể biểu diễn một số tới 28 digits nằm bên phải dấu chấm để cho thật chính xác. Hể càng nhiều digits nằm bên phải dấu chấm thì tầm trị số của Decimal càng nhỏ hơn.

Char Type

VB.NET có cả **Byte** lẫn **Char** data type. Byte được dùng cho một số nhỏ 0-255, có thể chứa một ASCII character trong dạng con số.

Char được dùng để chứa một Unicode (16 bit) character. Char là một character của String.

String Type

Nhìn lướt qua, String trong VB.NET không có vẻ khác VB6 bao nhiêu. Nhưng trừ khi ta muốn tiếp tục dùng các Functions như InStr, Left, Mid and Right trong VB6, ta nên xem String là một object và dùng những Properties/Functions của nó trong VB.NET cho tiện hơn. Sau này ta sẽ học thêm về String của VB.NET trong một bài riêng.

Ý niệm fixed-length (có chiều dài nhất định) String trong VB6 không còn dùng nữa. Do đó ta không thể declare:

```
Dim myString As String * 25
```

Object thay thế Variant

Một trong những data types linh động, hiệu năng và nguy hiểm trong VB6 là **Variant**. Một variable thuộc data type Variant có thể chứa gần như thứ gì cũng được (trừ fixed-length string), nó tự động thích nghi bên trong để chứa trị số mới. Cái giá phải trả cho sự linh động ấy là program chạy chậm và dễ có bugs tạo ra bởi sự biến đổi từ data loại này qua loại khác không theo dự tính của ta.

VB.NET thay thế Variant bằng **Object**. Vì trên phương diện kỹ thuật tất cả data types trong .NET đều là Object nên, giống như Variant, Object có thể chứa đủ thứ.

Nói chung, dầu Object giống như Variant, nhưng trong .NET ta phải nói rõ ra (explicitly) mình muốn làm gì. Ta thử xem một thí dụ code trong VB6 như sau:

```
Private Sub Button1_Click()
  Dim X1 As Variant
  Dim X2 As Variant
  X1 = "24.7"
  X2 = 5
  Debug.Print X1 + X2   ' Cộng hai số với operator +
  Debug.Print X1 & X2   ' Ghép hai strings lại với operator &
End Sub
```

Kết quả hiển thị trong Immediate Window là :

29.7

24.75

Trong VB.NET, ta phải code cho rõ ràng hơn như sau để có cùng kết quả như trên hiển thị trong Output Window:

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
Button1.Click
```

```
  Dim X1 As Object
  Dim X2 As Object
  X1 = "24.7"
  X2 = 5
  Console.WriteLine(CSng(X1) + CInt(X2))
  Console.WriteLine(CStr(X1) & CStr(X2))
End Sub
```

CType Statement

Trong VB.NET có **Option Strict** by default. Nó bắt ta phải thận trọng trong cách dùng data types. Vì Object có thể chứa bất cứ thứ gì, khi ta muốn dùng nó như một loại data type hay class nào, ta phải đổi Object ra thứ ấy bằng **CType**, thí dụ:

```
Class Product
  Public Description As String
End Class

Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
Button1.Click
  Dim X As Object
  X = New Product()
  ' Treat X like an actual product
  CType(X, Product).Description = "Soft Drink"
  Console.WriteLine(CType(X, Product).Description)
End Sub
```

Mặc dù X được instantiated như một Product, nó vẫn được xem như một Object variable. Do đó mỗi khi muốn dùng nó như một Product ta phải nhờ đến CType. Từ chuyên môn trong programming gọi đó là **Type Casting**.

Thay đổi trong cách tuyên bố Variables

Tuyên bố nhiều Variables

Trong VB6 ta có thể Declare nhiều variables trên cùng một hàng như:

```
Dim i, j, k As Integer
```

Kết quả là chỉ có k là Integer, còn i và j là Variant (có thể đó là điều bạn không ngờ). Trong VB.NET thì cả ba i,j và k đều là Integer, và như thế hợp lý hơn.

Tuyên bố trị số khởi đầu

Trong VB6, sau khi declare variable ta thường cho nó một trị số khởi đầu như:

```
Dim X As Integer

X = 12
```

Bây giờ trong VB.NET ta có thể gọp chung hai statements trên lại như sau:

```
Dim X As Integer = 12
```

Tuyên bố Constants

Khi tuyên bố Constants trong VB.NET ta phải khai rõ Data type của nó là String, Integer, Boolean ..v.v.:

```
Public Const myConstantString As String = "happy"

Public Const maxStudent As Integer = 30
```

Dim As New

Trong VB6 ta được khuyên *không nên* code:

```
Dim X As New Customer
```

vì VB6 không instantiate một Object Customer cho đến khi X được dùng đến - chuyện này rất nguy hiểm vì có thể tạo ra bug mà ta không ngờ.

Trong VB.NET ta có thể yên tâm code:

```
Dim X As New Customer()
```

vì statement nói trên lập tức tạo ra một Object Customer.

Tuyên bố Variable trong Scope của Block

Trong thí dụ dưới đây, variable X được declared trong một IF ..THEN...END IF block. Khi execution ra khỏi IF block ấy, X sẽ bị hủy diệt.

Do đó, VB.NET sẽ than phiền là X **undefined** vì nó không thấy X bên ngoài IF block. Luật này cũng áp dụng cho những Blocks khác như DO...LOOP, WHILE...END WHILE, FOR...NEXT, .v.v..

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
Button1.Click

  Dim A As Integer = 5

  Dim B As Integer = 5

  If A = B Then

    Dim X As Integer  ' X is declared in this IF block

    X = 12

  End If

  A = X    ' X has been destroyed, so it is undefined here

End Sub
```

Có lẽ bạn hỏi Declare Variable trong FOR...LOOP có lợi gì, tại sao ta không Declare một lần duy nhất ở đầu?

Thứ nhất là Block giới hạn scope (phạm vi hoạt động) của một variable để nó không đụng chạm ai dễ gây nên bug, thứ hai là trường hợp điển hình ta sẽ cần feature này là trong một FOR...LOOP, cứ mỗi iteration ta muốn instantiate một Object mới. Khi ấy ta cần Declare một Object variable, instantiate Object , rồi chứa nó vào một collection chẳng hạn.

Truy cập Variable/Class/Structure

Trong VB.NET ta có thể quyết định giới hạn việc truy cập một Variable, Class, Structure .v.v. bằng cách dùng các keywords sau:

Loại truy cập	**Thí dụ**	**Chú thích**
Public	**Public Class ClassForEverybody**	Cho phép ở đâu cũng dùng nó được. Ta chỉ có thể dùng Public ở mức độ Module, Namespace hay File. Tức là ta không thể dùng Public trong một Sub/Function.

Protected	**Protected Class ClassForMyHeirs**	Cho phép các classes con, cháu được dùng. Ta chỉ có thể dùng Protected ở mức độ Class.
Friend	**Friend StringForThisProject As String**	Cho phép code trong cùng một Project được dùng.
Private	**Private NumberForMeOnly As Integer**	Cho phép code trong cùng module, class, hay structure được dùng. Lưu ý là **Dim** coi như tương đương với **Private**, do đó ta nên dùng Private cho dễ đọc.

Ngoài ra, nhớ là nếu container (Object chứa) của một Variable/Class/Structure là Private thì dù ta có tuyên bố một Variable/Class/Structure nằm bên trong container là Public ta cũng không thấy nó từ bên ngoài.

Thay đổi trong Array

Array index từ 0

Trong VB.NET không có **Option Base** và mọi Array đều có index bắt đầu từ 0. Khi bạn tuyên bố một array như:

```
Dim myArray(10) As Integer
```

Kết quả là một array có 11 elements và index từ 0 đến 10. UBound của array này là 10 và LBound của tất cả arrays trong VB.NET đều là 0.

Tuyên bố Array với những trị số khởi đầu

Bạn có thể tuyên bố Array với những trị số khởi đầu như sau:

```
Dim myArray() As Integer = { 1, 5, 8, 16 }  ' Note the curly brackets
```

Statement làm hai chuyện: quyết định size của array và cho các elements trị số khởi đầu. Để dùng feature này, bạn không được nói rõ size của array, mà để cho program tự tính.

ReDim Preserve

Trong VB.NET bạn cũng có thể tiếp tục dùng **Preserve keyword** để giữ nguyên trị số của các elements trong một array khi bạn **ReDim** nó. Tuy

nhiên có một giới hạn cho array với hơn một dimension - bạn chỉ có thể resize dimention cuối (bên phải) , nên những hàng code sau đây hợp lệ:

```
Dim myArray(,) As String

ReDim myArray(5, 5)

ReDim Preserve myArray(5, 8)
```

Thay đổi trong User-Defined Type

Ý niệm **User-Defined Type (UDT)** rất tiện cho ta gom các mảnh data liên hệ lại thành một data type có cấu trúc. Trong VB6 ta dùng nó như sau:

```
Public Type UStudent
  FullName As String
  Age As Integer
End Type
```

VB.NET cũng giữ y đặc tính của UDT nhưng thay đổi chữ **Type** thành **Structure**:

```
Public Structure UStudent
  Public FullName As String
  Public Age As Integer
End Structure
```

Lưu ý các Structure Members (như FullName , Age) cần phải được Declared với keyword Dim, Public, Private hay Friend, nhưng không thể dùng Protected vì Structure không thể Inherit từ một Structure khác. Sở dĩ, có dùng Private là vì bên trong Structure có thể có Property, Sub/Function .v.v..

Thay đổi trong Collections

VB6 hổ trợ Collection và sau này Windows Scripting Host Library cho ta collection kiểu Dictionary. VB.NET cho ta một thành phần collection rất hùng hậu trong Namespace **System.Collections**. Vì Collection là một trong những công cụ rất thông dụng và hiệu năng trong VB.NET nên ta sẽ có một bài dành riêng cho collection sau này.

Dưới đây là danh sách các collections ta sẽ dùng thường xuyên:

Collection	Chức năng
ArrayList	Dynamic Array tự động lớn lên khi elements được bỏ vào.
BitArray	Array chứa trị số Boolean (True/False).
HashTable	Collection chứa những cặp key-value data, cho ta dùng làm tự điển.
Queue	Chứa một FIFO (First In, First Out) structure. Element có thể là bất cứ Object loại nào.
Stack	Chứa một LIFO (Last In, First Out) structure.
SortedList	Chứa một danh sách những cặp key-value data được sắp theo thứ tự.

Arithmetic Operators mới

VB.NET cho ta thêm cách viết Arithmetic Operator mới mà C programmers rất thích từ lâu nay.

```
X += 4       tương đương với  X = X + 4

Mess &= " text" tương đương với  Mess = Mess & " text"
```

Arithmetic Operation	Trong VB6	Cách viết tắc mới
Cộng	X = X +5	X += 5
Trừ	X = X - 10	X -= 10
Nhân	X = X * 7	X *= 7
Chia	X = X / 19	X /= 19
Chia Integer	X = X \ 13	X \= 13
Lũy thừa	X = X ^ 3	X ^= 3
Ghép Strings	X = X & "more text"	X &= "more text"

Ta vẫn có thể tiếp tục dùng cách viết trong VB6, nhưng bây giờ có thêm một cách viết gọn hơn.

Short Circuit trong IF..THEN Statement

Trong VB6, nếu ta viết:

```
Dim myInt As Integer

myInt = 0

If (myInt <> 0) And (17 \ myInt < 5 ) Then
```

Thì sẽ bị Division by Zero error, vì mặc dầu phần **(myInt <>0)** là **False**, nhưng VB6 vẫn tiếp tục tính phần

(17 \ myInt < 5), và tạo ra error vì 17 chia cho một số 0. Trong vài ngôn ngữ lập trình khác, khi **(myInt <>0)** là **False** thì nó không tính thêm nữa, tức là nó nói rằng khi một phần của **AND** là False thì nhất định kết quả của Logical Statement trong IF phải là False. Đặc tính này gọi là **Short-Circuit (đi tắc)**.

Nếu ta dùng code nói trên trong VB.NET, nó vẫn cho Division by Zero error giống như VB6. Tuy nhiên, nếu ta muốn dùng đặc tính Short-Circuit thì ta chỉ cần thay thế chữ **And** bằng **AndAlso** như sau:

```
Dim myInt As Integer

myInt = 0

If (myInt <> 0) AndAlso (17 \ myInt < 5 ) Then
```

Short-Circuit cũng áp dụng cho Logical OR khi ta thay thế chữ **Or** bằng **OrElse** để nói rằng khi phần đầu của **OR** là True thì nhất định kết quả của Logical Statement trong IF phải là True.

Không còn Set statement cho Object

Trong VB6 ta có thể viết:

```
Set x = New Product

Set w = x
```

Trong VB.NET sẽ được viết lại như sau:

```
x = New Product()

w = x
```

Bây giờ ta không cần phải nhớ dùng chữ **Set** khi nói đến Object.

Thay đổi trong cách viết Property routines

Dùng một Property duy nhất

Nếu trong VB6 ta viết:

```
Private mdescription as String

Public Property Let Description (Value As String)

  mdescription = Value

End Property


Public Property Get Description() As String

  Description = mdescription

End Property
```

Trong VB.NET **Let** và **Get** được hợp lại trong một Property routine duy nhất và ta lại dùng chữ **Set** thay cho chữ **Let** (mặc dầu chữ Set không còn dùng cho Object như mới nói ở trên) như sau:

```
Private mdescription As String

Public Property Description() As String

  Set  (ByVal Value As String)

    mdescription = Value

  End Set

  Get

    Description = mdescription

  End Get

End Property
```

ReadOnly và WriteOnly property

Bây giờ nếu Property là ReadOnly ta sẽ viết:

```
Public ReadOnly Property Age() As Integer
  Get
    Age = 3
  End Get
End Property
```

hay WriteOnly ta sẽ viết:

```
Private _data As Integer
Public WriteOnly Property Data() As Integer
  Set (ByVal Value As Integer)
    _data = Value
  End Set
End Property
```

Default Properties

Ta dùng **Default keyword** để tạo ra Default Property như sau:

```
Default Public Property Item(ByVal Index As Integer) As String
```

VB.NET bắt buộc ta phải ít nhất một parameter cho Default Property.

Dùng Reserved Word làm Procedure Name

Trong VB.NET ta có thể dùng Reserved Word làm Procedure Name bằng cách để nó giữa ngoặc vuông. Giả tỉ ta muốn dùng chữ **Compare** làm tên một Function, ta sẽ viết như sau:

```
Public Function [Compare] (ByVal v1 As Integer, ByVal v2 As Integer) As Boolean
```

Structured Error Handling

TRY...CATCH...FINALLY

VB.NET cho ta Structure TRY...CATCH...FINALLY...END TRY để xử lý error. Thí dụ như trong bài toán chia dưới đây, nếu bị *Division by 0 error* thì ta sẽ cho kết quả bằng 0. Dù có error hay không, program vẫn hiển thị kết quả trong Output Window qua statement **Console.WriteLine(result)** trong phần **Finally**:

```
Try
  result = a / b  ' if this section has error jump to Catch section
Catch
  ' only get here if an error occurs between Try and Catch
  result = 0
Finally
  ' This section is optional, but is always executed whether there is an error or not
  Console.WriteLine( result)
End Try
```

Nếu ta không code gì ở phần Catch thì có nghĩa là chúng ta có Handle Error nhưng lại không làm gì hết, do đó Program sẽ không té. Ngược lại, nếu ta không dùng Try..Catch, thì program sẽ té.

Nếu muốn nhảy ra khỏi Try Structure bất cứ lúc nào ta có thể dùng **Exit Try**,

Những cách CATCH error

Ta có thể dùng **Catch** giống như **Select Case** để có một cách xử lý cho mỗi error:

```
Try
  ' Main code goes here
Catch When Err.Number=5
  ' handle Error 5
Catch
  ' handle other errors
End Try
```

Ta có thể Catch Error Exception data trong một variable để dùng nó như sau:

```
Catch e as Exception
  MessageBox.Show (e.ToString)
```

Hai cách code ở trên có thể được gợp lại thành:

```
Catch e As Exception When Err.Number = 5
```

Thay đổi trong cách viết Sub/Function

Dùng dấu ngoặc khi gọi Procedure

Trong VB6, nếu không dùng keyword **Call** ta không dùng dấu ngoặc khi gọi Sub. Trong VB.NET ta luôn luôn dùng cặp dấu ngoặc, ngay cả khi không có parameter. Thí dụ:

```
ProcessData()

x = New Customer()
```

ByVal là Default cho mọi Parameters

Trong VB6, **ByRef** là default cho các parameters passed vô Sub/Function. Tức là, Sub/Function có thể vô tình làm thay thổi trị số nguyên thủy của parameter variables.

Trong VB.NET, **ByVal** là default cho các parameters passed vô Sub/Function. Do đó, nó sẽ tránh lỗi lầm nói trên.

Optional Parameter cần có trị số Default

Trong VB6 ta có thể dùng **IsMissing** để biết xem **Optional parameter** có hiện diện không. VB.NET đã bỏ IsMissing và bắt buộc ta phải cung cấp trị số Default cho Optional parameter trong phần procedure declaration giống như sau đây :

```
Public Sub VerifyInput (Optional ByVal InputData As String="")
```

trong thí dụ này ta cho Default value của Optional parameter InputData là Empty string.

Return Statement

Hãy xem một thí dụ dùng Function để return một Customer Object trong VB6:

```
Public Function GetCustomer (ByVal CustID As Long) As Customer
  Dim objCust As Customer
  Set objCust = New Customer
  objCust.Load CustID
```

```
  Set GetCustomer = objCust
End Function
```

Trong VB.NET ta có thể dùng Return Statement để Return kết quả của một Function thay vì dùng chính tên của Function.

```
Public Function GetCustomer (ByVal CustID As Long) As Customer
  Dim objCust As New Customer(CustID)
  Return objCust
End Function
```

Delegate

Delegate là một cách giúp ta pass một procedure như một parameter vào trong một method. Ý niệm này được gọi là **Function Pointer** hay **Callback**. Một trường hợp cổ điển ta dùng Delegate là cung cấp một dataArray để sort với một Function để so sánh mỗi hai items trong array.

Trong VB.NET ta dùng **AddressOf** operator để pass một procedure. Ta declare một Delegate bằng cách nói nó là một procedure dưới dạng nào, có bao nhiêu parameters, mỗi parameter thuộc loại data type nào. Thí dụ:

```
Delegate Function IsGreater (ByVal v1 As Integer, ByVal v2 As Integer) as Boolean
```

Khi viết code ta cứ yên tâm sẽ được cung cấp một Function có dạng ấy và đại khái code như sau:

```
Public Sub DoSort (ByRef DataArray() As Integer, Greater As IsGreater)
  Dim outer As Integer
  Dim inner As Integer
  Dim temp As Integer
  For outer = 0 To UBound(DataArray)
    For inner = outer + 1 To UBound(DataArray)
      If GreaterThan.Invoke( DataArray(outer), DataArray(inner)) Then
        temp = DataArray(outer)
        DataArray(outer) = DataArray(inner)
        DataArray(inner) = temp
```

```
        End If
    Next
  Next
End Sub
```

Để ý cách dùng **Method Invoke** để gọi một Delegate. Bây giờ ta chỉ cần cung cấp Delegate routine mà ta đã hứa:

```
Public Function myIsGreater (ByVal v1 As Integer, ByVal v2 As Integer) as Boolean
  Return ( v1 > v2)
End Function
```

Tiếp theo đây là cách ta dùng Delegate nói trên:

```
Dim myData() As Integer = { 2, 5, 8, 13, 26}

DoSort (myData, AddressOf myIsGreater)
```

Khi một Subscriber registers với một Publisher một routine để Handle một loại Event, ta cũng dùng delegate như sau:

```
AddHandler Button4.Click, AddressOf Button4_Click
```

Khi một Event Click xãy ra ở Button4, system sẽ execute Sub Button4_Click.

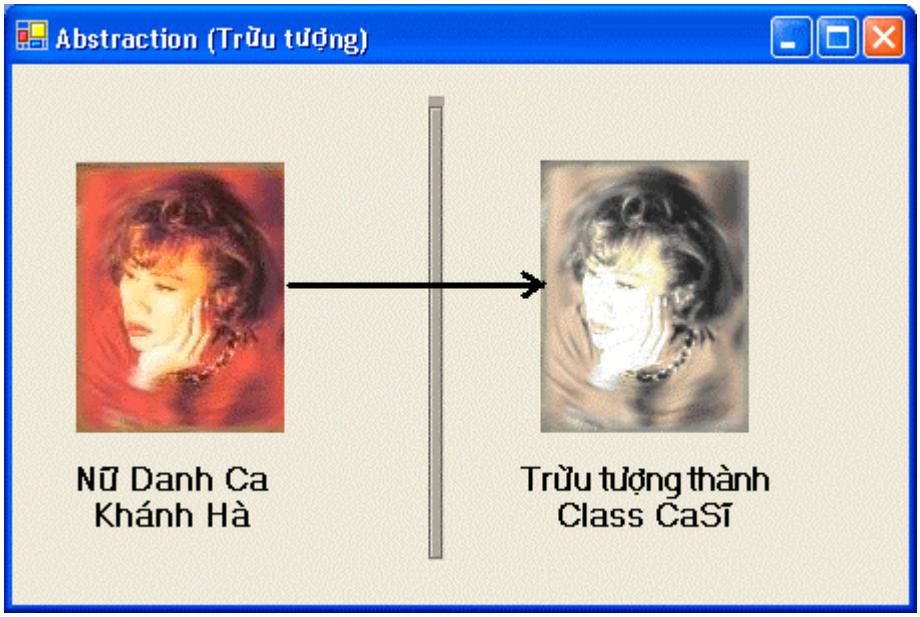
Bài 4
Những chức năng Đối Tượng mới của VB.NET (phần I)

VB.NET khắc phục những giới hạn về **Đối Tượng (Object-Oriented)** của VB6 và mang đến cho ta một ngôn ngữ lập trình hoàn toàn Object-Oriented (OO). Gần như mọi thứ trong VB.NET đều liên hệ với Object. Nếu bạn còn mới với lập trình theo hướng đối tượng (Object Oriented Programming) thì phần giải thích sau đây sẽ giúp bạn làm quen với nó.

Classes và Objects, nguyên tắc Abstraction

Theo phương pháp đối tượng, program được thiết kế để một phần code đại diện cho một vật tương đương ngoài đời. Nó được gọi là **Class**.

Khi lập trình VB6 ta đã dùng những controls từ Toolbox như Textbox, Label, Listbox ..v.v.. **Textbox** là Class của các Objects **Text1, Text2**. Cũng như **Label1, Label2** là những Objects tạo ra từ Class **Label**. Ta hay dùng hai từ **Class** và **Object** lẫn lộn nhau. Điều đó không quan trọng, miễn là ta biết rằng Class là một ý niệm **Trừu tượng (Abstraction)**, còn Object là một vật thực hữu. Giống như **Class CaSĩ** là một ý niệm trừu tượng, còn **Object KhánhHà** của Class CaSĩ là một người bằng da, bằng thịt với tiếng hát được nhiều người ngưỡng mộ.



Ta nói Object là một **Instance** của Class, và ta **instantiate** Class để có một Object.

Thường thường khi ta phân tích một vấn đề để thiết kế chương trình thì các **Danh từ (Nouns)** là những Classes. Giả dụ ta phân tích hoạt động của một Nhà Kho (warehouse). Ta có phòng chứa, ngăn tủ, bãi nhận hàng, xe nâng hàng, nhân viên ..v.v., mỗi thứ đều có thể là một Object nên ta sẽ thiết kế một Class cho nó.

Fields, Properties, Methods và Events, nguyên tắc Encapsulation

Class CaSĩ diễn tả CaSĩ là người như thế nào. Như **SốBàiHát** là một Public Variable của Class, được gọi là **Field** có thể được đọc/viết trực tiếp. Còn **Kiểu tóc** (dài, ngắn, màu đen, có sọc nâu ...), **Giọng hát** (cao, trầm, ..). là những **Properties**. Chúng cũng giống như Field nhưng được implemented (thi hành) bằng cách dùng procedures **Property Get** và **Property Set**. Property Set có thể được coded để kiểm soát nếu "Kiểu tóc" không thích hợp thì sẽ bị loại bỏ. Ngược lại, nếu "Kiểu tóc" thích hợp và được áp dụng thì ta sẽ thấy kết quả ngay là CaSĩ lại đẹp thêm ra. Thường thường Fields và Properties là các **Danh từ (Nouns)**.

Một CaSĩ có khả năng **ĐơnCa**, **KýTênLưuNiệm**, **TrìnhDiễn**. Ta gọi đó là những **Methods** mà ta implemented bằng **Subs** và **Functions** (thí dụ như **Function KýTênLưuNiệm** sẽ return một chữ ký). Thường thường Methods là những **Động từ (Verbs)**

Đối với code bên trong Class thì Property giống như một Method còn đối với **Client** (tức là program đang dùng Class) thì Property giống như Field.

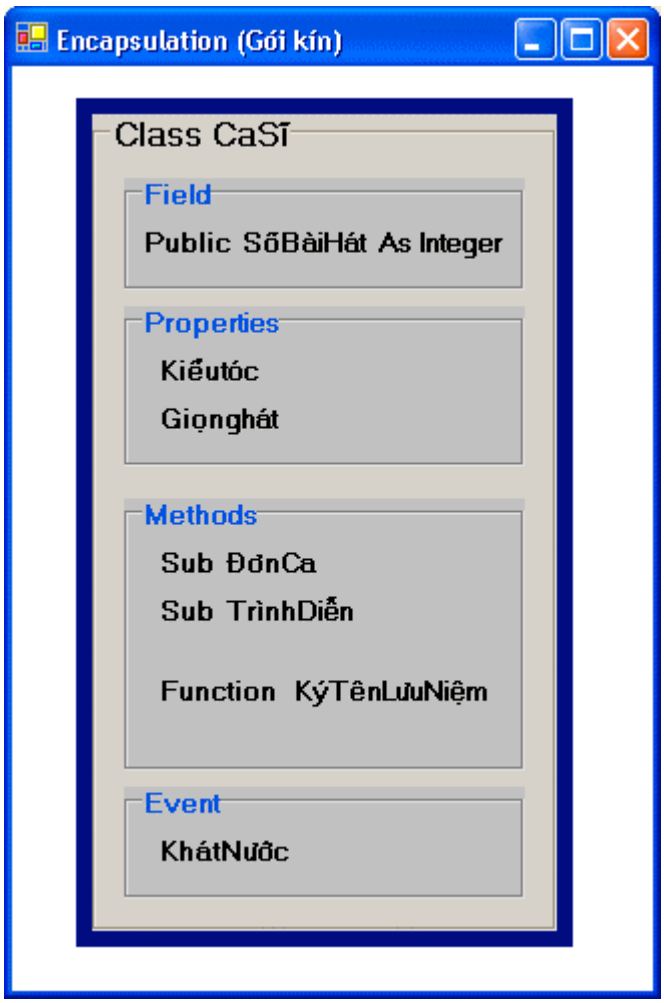
Đôi khi, nếu trình diễn lâu, CaSĩ cần một ly nước. CaSĩ sẽ Raise **Event Khát Nước** để nhân viên trong hậu trường phục vụ.

Ta gọi chung Fields, Properties, Methods và Events là những **Class Members (Các Thành viên của Class)**

Có một ngoại lệ về sự khác biệt giữ Class và Member, đó là khi ta dùng các **Shared Class Members** của một Class thì ta không nhất thiết phải instantiate một Object. Ta có thể dùng thẳng tên của Class như một Object.
Cái lợi điểm của Object Oriented Programming là ta có thể gói tất cả những đặc điểm, khả năng của một Class vào trong một **Unit of Code** (Đơn vị mã) tự túc. Khi chúng ta lịch sự yêu cầu thì CaSĩ **ĐơnCa**. Ta biết

CaSĩ ca thì thu hút lòng người, nhưng ta không cần biết làm sao CaSĩ đạt đến trình độ như vậy. Đó không phải là chuyện để chúng ta quan tâm.

Đối với ta Class CaSĩ là một **Black Box**, ta không biết và không cần biết chuyện gì xãy ra bên trong. Nếu sau này CaSĩ thay đổi kỹ thuật đơn ca để hát dễ và hay hơn, điều đó không ảnh hưởng gì đến chúng ta. Đặc tính OO ấy gọi là **Encapsulation (Gói kín)**.



Cách ta lập trình với Class chỉ khác cách ta lập trình trước đây một chút thôi. Nếu trước đây ta phải tự làm, thì bây giờ ta instantiate một Object của Class chuyên trị những chuyện ta muốn làm, rồi bảo nó làm cho ta. So với ngoài đời, thí dụ bạn có mở một tiệm photocopy. Sau một năm bạn tự trông coi, công chuyện làm ăn ổn định và có kết quả tốt. Bạn muốn mở thêm một tiệm photocopy nữa ở chỗ khác. Trước khi đi lo chỗ khác bạn huấn luyện nghề photocopy cho một người làm công trung thành, rồi giao cho người ấy làm quản lý để thay thế bạn. Người đó là môt Object của **Class QuảnLýTiệmPhotoCopy**.

Trở lại cách lập trình, những công việc bạn làm hằng ngày trong tiệm photocopy là những **Methods**. Tất cả đồ đạc, sổ sách của tiệm là những **Properties**. Bạn đã sắp đặt mỗi tuần phải gọi người lại quét dọn tiệm, mỗi tháng phải bảo trì các máy photocopiers, đó là những **Events**. Bây giờ bạn gói tất cả những thứ ấy lại thành **Class QuảnLýTiệmPhotoCopy**. Lần đầu bạn instantiate Class QuảnLýTiệmPhotoCopy làm thành **ChúTưThông**, người sẽ thay thế bạn làm quản lý tiệm photocopy đầu tiên. Khi bạn muốn mở thêm tiệm thứ ba, bạn sẽ instantiate Class QuảnLýTiệmPhotoCopy một lần nữa làm thành **DìSáuHương** , người sẽ thay thế bạn làm quản lý tiệm photocopy thứ nhì.

Khi đã phân chia trách nhiệm các phần code thành những Class, bạn có thể tập trung tư tưởng vào từng Class một, không cần phải cố nhớ mọi thứ trong đầu khi giải quyết chuyện gì. Vì code của Class nào chỉ làm việc và ảnh hưởng trong phạm vi hoạt động của nó, không đụng chạm đến ai khác. Nếu có gì trục trặc, thường thường ta có thể xác định đó là lỗi của Class nào tương đối dễ dàng.

Có một câu hỏi đùa rằng theo phương pháp OO thì: "Thay một bóng đèn cần bao nhiêu programmers?". Đáp: "Không cần programmer nào hết, bạn bảo đèn tự thay bóng của nó." (Lời đáp khác: "Không cần programmer nào hết, Microsoft đã đổi tiêu chuẩn ra bóng đêm.")

Do đó, nếu trước kia bạn lập trình để tự mình lo liệu công chuyện thì bây giờ hãy giao cho các Objects tự lo cho chúng. Tức là trước đây, nếu bạn là chủ điền mỗi năm bạn phải đi góp lúa ruộng, thì bây giờ bạn bảo các tá điền phải tự đem nộp lúa vào trong kho cho bạn. Sướng không? Chỉ ở trong thế giới lập trình OO, ta mới có thể mơ mộng như vậy.

Inheritance (Thừa Kế)

Nguyên tắc Encapsulation nói trên cho phép ta dùng nhiều Objects của một hay nhiều Classes một cách an toàn, tức là không sợ Methods của các Objects giẫm chân lên nhau.

Giả sử ta muốn dùng lại một Class để làm một Class mới, đặc biệt hơn, thí dụ như ta muốn làm nên một **Class CaSĩ** từ **Class NghệSĩ**. Cách làm

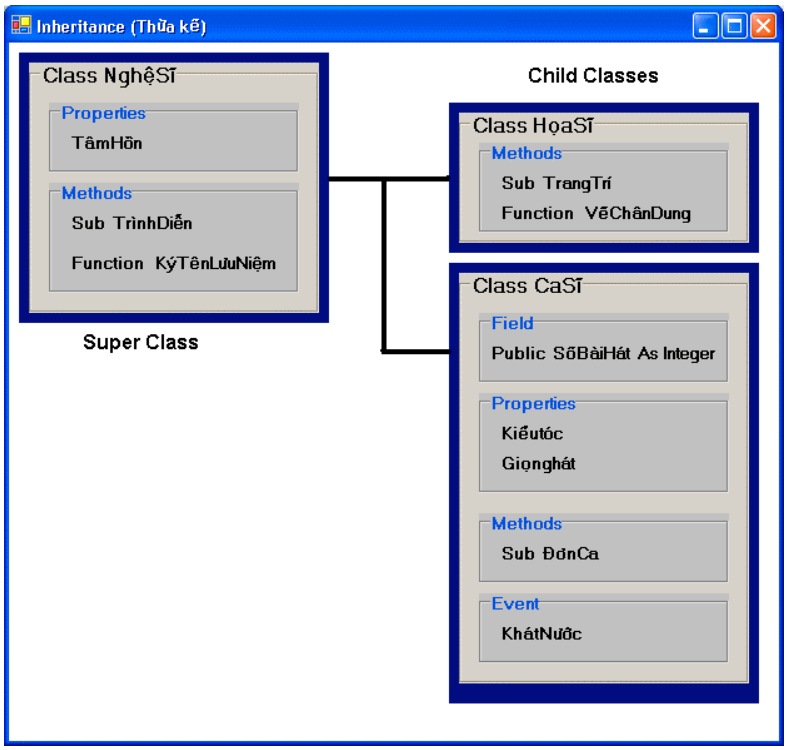
ấy gọi là **Inheritance (Thừa kế)**. Công việc thừa kế này được thực hiện qua một quá trình gọi là **Subclassing**.

Ở đây ta dùng lại Class NghệSĩ mà hoàn toàn không đụng đến **Source Code (Nguồn Mã)** của Class NghệSĩ. Nguyên tắc ấy gọi là **Reusability (Dùng lại)**. Lưu ý là nếu ta dùng lại Source code mà có sửa đổi một chút trong Source Code thì không thể gọi là Reuse được vì có thể việc sửa đổi Source Code đó sẽ gây ra bugs mới. Ta phải chỉ cần Inherit từ Object Code của một Class cũng được thì mới thật sự là Reuse.

Ta dùng Inheritance để cho thêm các Class Members, tức là thêm đặc tính và chức năng. Thí dụ NghệSĩ thì có **Property TâmHồn** (NhạyCảm (Sentitive) , ThơMộng (Romantic),...), và **Methods KýTênLưuNiệm, TrìnhDiễn**. Class CaSĩ sẽ giữ y các đặc tính và chức năng ấy và thêm **Sub ĐơnCa**, **Function HátNhạcYêuCầu**, .v.v..

Tương tự như vậy, ta cũng có thể thừa kế từ Class NghệSĩ để tạo ra **Class HọaSĩ**. Class HọaSĩ sẽ giữ y các đặc tính và chức năng của Class NghệSĩ nhưng thêm **Function VẽChânDung**, **Sub TrangTrí**.

Trong thí dụ nói trên, người ta gọi Class NghệSĩ là **Parent Class, Super Class** hay **Base Class**. Còn Class CaSĩ và Class HọaSĩ được gọi là **Child Class** hay **SubClass**.



Nếu ta lại Inherit Class CaSĩ để tạo ra **Class CaSĩTânNhạc** và **Class CaSĩCổNhạc** thì trong trường hợp này CaSĩ là Parent Class và CaSĩTânNhạc với CaSĩCổNhạc là Child Classes.

Mỗi Casĩ **là** một NghệSĩ nên ta có mối liên hệ "**IS (Là)**" giữa hai classes này. Nó khác với mối liên hệ "**HAS (Có)**". Thí dụ nếu trong Class CaSĩ **có** một Object thuộc **Class ĐầuBếp**, thì một CaSĩ có thể cho ta một bữa ăn ngon nhưng không hẳn cho chính CaSĩ nấu. Nó giống như ngoài đời CaSĩ KhánhHà mướn một đầu bếp để đãi khách. Ta sẽ nói Class CaSĩ có mối liên hệ **HAS (Có)** với Class ĐầuBếp trong trường hợp này, chớ không phải Class CaSĩ **IS (Là)** một Class ĐầuBếp.

Trong .NET ta chỉ có **Single (Đơn) Inheritance**, tức là một Class không thể Inherit từ hai hay ba Classes khác. Giống như nói Con thừa kế từ Cha và Cha thừa kế từ ÔngNội, không có nhắc gì đến Mẹ hay BàNội. Một Child Class chỉ có một Parent Class, ngược lại, một Parent Class có thể có nhiều Child Classes.

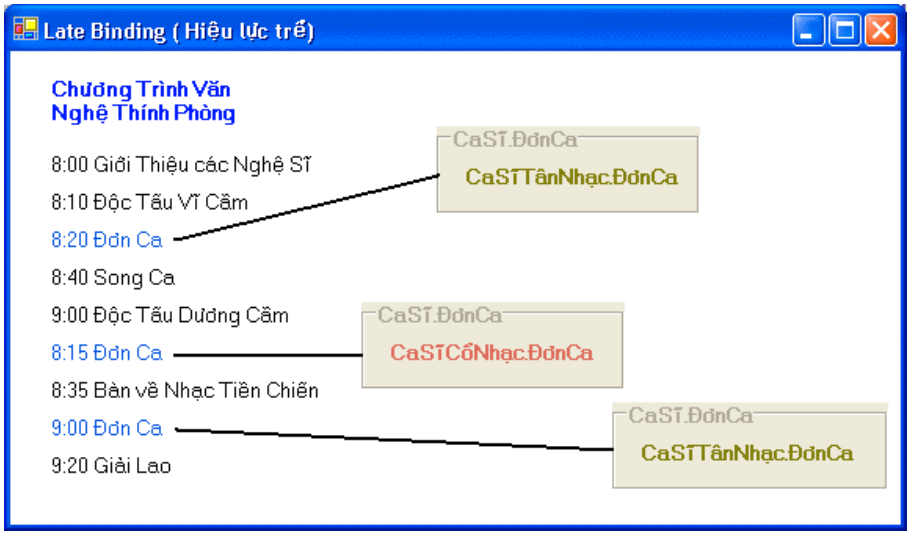
Polymorphism (Đa dạng)

Polymorphism là khả năng dùng Class Members trùng tên của Objects thuộc về các Classes khác nhau. Thí dụ Objects **KháchHàng** và **NhânViên** đều có **Property Name**. Nếu ta có thể lập trình để dùng Name

mà không cần nói rõ nó thuộc về Object KháchHàng hay NhânViên thì đó là Polymorphism.

Polymorphism thể hiện dưới nhiều hình thức:

1. **Late Binding (Hiệu lực trễ)**: Có nghĩa là đợi đến giờ chót, khi execution, thì code mới biết nó đang làm việc với loại Object nào. Chữ binding nói đến "hiệu lực", late binding là có hiệu lực trễ. Điều này được thực hiện bằng cách hứa hẹn một Object thuộc Parent Class để trong lúc runtime ta có thể giao cho code một Object thuộc Child Class. Thí dụ ta hứa với khán giả sẽ có một CaSĩ trình diễn, lúc mở màn ta có thể cung cấp một CaSĩTânNhạc hay một CaSĩCổNhạc.



2. **Overloading (Quá tải, đã có rồi mà còn cho thêm)** : Overloading cho phép ta viết trong cùng một Class nhiều versions khác nhau của Property hay Method. Chúng được phân biệt nhờ dùng parameters khác data type hay con số parameters khác nhau. Thí dụ một version của Sub được passed cho một Integer Parameter, một version khác được passed cho một String Parameter, một version khác lại được passed cho hai parameters. Khi ta gọi một Method của Class, nó sẽ dựa vào data type của parameters ta pass và số parameters ta pass để execute đúng version của Method.

   Một thí dụ về Overloading ngoài đời là khi ta yêu cầu CaSĩ đơn ca ta được phép đề nghị CaSĩ hát theo

Karaoke, hay được Ban Nhạc Sống phụ họa, hay thêm cả một nhóm ca sĩ khác phụ họa .v.v..

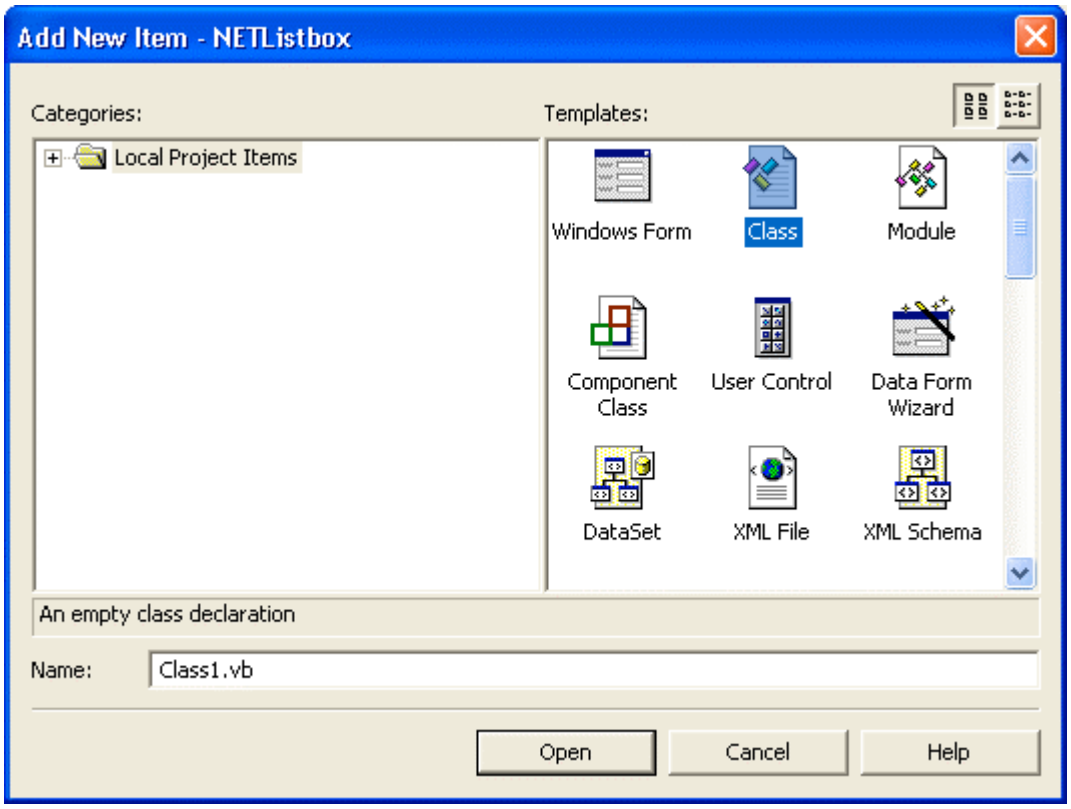
3. **Overriding (Lấn quyền)** : Overriding áp dụng cho Child Class đối với Parent Class. Trong Child Class ta cung cấp một Method cùng tên, cùng số parameters và cùng parameter data type với một Method trong Parent Class (ở đây không nhất thiết phải là Cha, có thể là ÔngNội hay nhiều đời trước) để dùng nó thay thế cho Parent Class Method. Ta nói Child Class thay đổi behaviour (tánh tình, cách xử sự) của Parent Class. Đại khái giống như **cụ LữLiên** trước đây **Hát** nhạc hài hước, bây giờ **cô KhánhHà** thừa kế từ cụ nhưng override **Method Hát** của cụ và cô implement một **Method Hát** mới dùng cho nhạc trử tình.

   Lúc runtime, nếu một Object không có implementation của một Method thì CLR (Common Language Runtime) sẽ dùng Method của Parent Class của nó. Trong thí dụ trên vì cô KhánhHà có một implementation cho method Hát nên system sẽ dùng method đó, thay vì dùng method Hát của cụ LữLiên.

Dùng OO trong VB.NET

Tạo một Class mới

Bạn tạo một Class mới trong VB.NET IDE bằng cách dùng Menu Command **Project | Add Class**. Dialog **Add New Item** sẽ hiện ra, chọn **Class** trong số hình các Icons nằm trong khung bên phải của Dialog.



Source code của Class mới này sẽ được chứa trong một VB source file với extension **vb**. Trong VB.NET tất cả mọi VB source files đều có extension **.vb**. System sẽ nhận diện ra loại VB file (form, class, module,.v.v..) nhờ đọc content của file, chớ không dựa vào file extension.

Nếu bạn muốn đặt tên cho Class mới này là TheClass chẳng hạn, thì bạn có thể sửa tên nó trong Dialog. Khi bạn click button **Open** một file mới sẽ được cho thêm vào trong Project và nó chứa hai hàng code sau:

```
Public Class TheClass

End Class
```

Bài 5
Những chức năng Đối Tượng mới của VB.NET (phần II)

Dùng OO trong VB.NET

Tạo một Class mới

Class Keyword

Trong một .vb file ta có thể viết nhiều Classes, code của mỗi Class nằm trong một **Class** ... **End Class** block. Thí dụ:

```
Public Class TheClass

  Public Sub Greeting()

    MessageBox.Show("Hello world", MsgBoxStyle.Information, "TheClass")

  End Sub

End Class
```

MessageBox.Show và MsgBoxStyle.Information trong VB.NET thay thế MsgBox và vbInformation trong VB6.

Classes và Namespaces

Nhắc lại là .NET dùng **Namespace** để sắp đặt các Classes cho thứ tự theo nhóm, loại. Namespaces được declared với một Block Structure giống như sau:

```
Namespace Vovisoft

  Public Class TheClass

    Public Sub Greeting()

      MessageBox.Show("Hello world", MsgBoxStyle.Information, "TheClass")

    End Sub

  End Class

End Namespace
```

Muốn nói đến bất cứ Class, Structure, hay thứ gì được declared bên trong một **Namespace...End Namespace** block ta phải dùng tên Namespace trước. Thí dụ:

```
Private myObject As Vovisoft.TheClass
```

Một source file có thể chứa nhiều Namespaces, và bên trong mỗi Namespace lại có thể có nhiều Classes.

Ngoài ra, Classes thuộc về cùng một Namespace có thể nằm trong nhiều files khác nhau trong một VB.NET project.

Thí dụ ta có một source file với code như sau:

```
Namespace Vovisoft

  Public Class TheClass

  ' Code

  End Class

End Namespace
```

Và một source file khác trong cùng project với code:

```
Namespace Vovisoft

  Public Class TheOtherClass

  ' Code

  End Class

End Namespace
```

Vậy thì trong Namespace **Vovisoft** ta có hai Classes **TheClass** và **TheOtherClass**.

Nhớ là, by default, **Root Namespace** của một VB.NET project là tên của project ấy. Khi ta dùng Namespace block structure là chúng ta đang thêm một tầng tên vào Root Namespace. Do đó, trong thí dụ trên nếu tên project là **MyProject** thì, từ bên ngoài project ấy, ta có thể declare một variable như sau:

```
Private myObject As MyProject.Vovisoft.TheClass
```

Tạo ra Methods

Methods trong VB.NET có hai thứ: **Sub** và **Function**. Function thì phải **return** một kết quả. By default, parameters của Method là **ByVal** chớ không phải **ByRef**. Tức là nếu muốn parameter nào ByRef thì phải nhớ khai ra rõ ràng.

Nhắc lại là khi một variable được passed vào trong một method bằng **ByVal** thì system cho method đó một copy (bản sao) của variable, do đó, trị số của variable không bị thay đổi bởi công tác của method. Ngược lại, nếu một variable được passed vào trong một method bằng **ByRef** thì method dùng chính variable đó, do đó, trị số của variable có thể bị thay đổi bởi công tác của method.

Ta có thể giới hạn việc sử dụng một method bằng cách áp đặt một **Access Modifier** (sửa đổi quyền truy nhập) hay còn gọi là **Scoping keyword** (phạm vi hoạt động):

- **Private** - chỉ cho phép code trong cùng Class được gọi.
- **Friend** - chỉ cho phép code trong cùng project/component được gọi.
- **Public** - cho phép ai gọi cũng được.
- **Protected** - cho phép code trong subclasses (classes con, cháu) được gọi.
- **Protected Friend** - cho phép code trong cùng project/component hay code trong subclasses được gọi.

Tạo ra Properties

Trong VB.NET ta chỉ dùng một routine duy nhất cho mỗi Property, với hai chữ **Get** và **Set** như sau (không còn dùng chữ **Let** của VB6 nữa):

```
Private mdescription As String

Public Property Description() As String

  Set  (ByVal Value As String)

    mdescription = Value
```

```
  End Set
  Get
    Description = mdescription
  End Get
End Property
```

ReadOnly và WriteOnly property

Bây giờ nếu Property là ReadOnly ta sẽ lấy phần Set ra và viết:

```
Public ReadOnly Property Age() As Integer
  Get
    Age = 3
  End Get
End Property
```

hay WriteOnly ta sẽ ấy phần Get ra và viết:

```
Private _data As Integer
Public WriteOnly Property Data() As Integer
  Set (ByVal Value As Integer)
    _data = Value
  End Set
End Property
```

Default Properties

Default Property là property của Object mà program dùng khi ta chỉ cho tên của Object và không nói rõ property nào. Thí dụ trong VB6 khi ta code:

```
TextBox1 = "The house of rising sun"
```

VB6 hiểu rằng ta muốn dùng Default Property **text** của Textbox1 nên code ấy tương đương với:

```
TextBox1.text = "The house of rising sun"
```

Trong VB6 khi ta dùng keyword **Set** với tên của Object, thí dụ như:

```
  Dim myTextBox As Textbox
```

```
Set myTextBox = TextBox1
```

program sẽ hiểu là ta muốn nói đến chính Object myTextBox . Nếu không thì nó biết ta muốn nói đến Object Default Property mà làm biếng code cho rõ ra.

Trong VB.NET Default Property phải là một **Property array**. Một Property array là một property được **Indexed** (nói đến từng Item bằng con số Index) giống như một array. Lý do chính của sự bắt buộc này là để khỏi lẫn lộn giữa hai trường hợp ta nói đến ***Default property*** *của một Object* hay ***chính Object ấy***, vì trong VB.NET ta không còn dùng **Set** keyword cho Object assignment nữa (ta chỉ còn dùng keyword Set trong Property mà thôi).

Bây giờ hễ muốn nói đến Default Property của Object thì phải dùng Index. Thí dụ để nói đến chính Object, ta code:

```
myValue = myObject
```

để nói đến Default Property Item 3 của Object, ta code:

```
myValue = myObject(3)
```

Sự thay đổi từ VB6 này có nghĩa là một property array procedure phải nhận một parameter. Thí dụ:

```
Private theData(100) As String

Default Public Property Data(ByVal Index As Integer) As String
  Get
    Data = theData(Index)
  End Get
  Set(ByVal Value As String)
    theData(Index) = Value
  End Set
End Property
```

Từ nay ta không thể code:

```
TextBox1 = "Good morning!"
```

như trong VB6 được nữa, mà phải code:

```
TextBox1.text = "Good morning!"
```

Vì Property Text không còn là Default Property của TextBox.

Overloading methods

Một trong những chức năng đa diện (Polymorphism) hùng mạnh nhất của VB.NET là **overload** (quá tải, có rồi mà còn cho thêm) một method. Overloading có nghĩa là ta có thể dùng cùng một tên cho nhiều methods - miễn là chúng có danh sách các parameters khác nhau, hoặc là parameter dùng data type khác nhau (td: method này dùng Integer, method kia dùng String), hoặc là số parameters khác nhau (td: method này có 2 parameters, method kia có 3 parameters).

Overloading không thể được thực hiện chỉ bằng cách thay đổi ***data type của Return value của Function***. Phải có parameter list khác nhau mới được.

Dưới đây là thí dụ ta dùng Overloading để code hai Functions tìm data, một cái cho String, một cái cho Integer:

```
Public Function FindData(ByVal Name As String) As ArrayList
  ' find data and return result
End Function

Friend Function FindData(ByVal Age As Integer) As ArrayList
  ' find data and return result
End Function
```

Để ý là ta có thể cho mỗi overloading Function một phạm vi hoạt động (Scope on implementation) khác nhau. Trong thí dụ trên ta dùng Access Modifier **Public** cho Function đầu và **Friend** cho Function sau.

Object Lifecycle

Object Lifecycle (cuộc đời của Object) được dùng để nói đến khi nào Object bắt đầu hiện hữu và khi nào nó không còn nữa. Sở dĩ ta cần biết rõ cuộc đời của một Object bắt đầu và chấm dứt lúc nào là để tránh dùng nó khi nó không hiện hữu, tức là chưa ra đời hay đã khuất bóng rồi.

New method

Trong VB6, khi một Object thành hình thì **Sub Class_Initialize** được executed. Tương đương như vậy, trong VB.NET ta có **Sub New()**, gọi là **Constructor**. VB.NET bảo đảm Sub New() sẽ được CLR gọi khi Object được instantiated và nó chạy trước bất cứ code nào trong Object.

Nếu Sub Class_Initialize của một Class Object trong VB6 không nhận parameter thì Sub New() trong VB.NET chẳng những có nhận parameters mà còn cho phép ta nhiều cách để gọi nó. Sự khác biệt trong Constructors của VB6 và VB.NET rất quan trọng.

Tưởng tượng ta có một **Khuôn làm bánh bông lan**; khuôn là Class còn những bánh làm ra từ khuôn sẽ là các Objects bánh bông lan. Nếu ta muốn làm một cái bánh bông lan với một lớp sô-cô-la trên mặt thì công tác sẽ gồm có hai bước:

1. Dùng khuôn (Class) nướng một cái Object bánh bông lan (dùng **Sub Class_Initialize**)
2. Đổ lên mặt bánh một lớp sô-cô-la (dùng class **Public Sub ThoaSôcôla**)

Đến đây, mọi chuyên tương đối ổn thỏa. Bây giờ, nếu khách hàng muốn một cái bánh bông lan dùng trứng vịt thay vì trứng gà thì ta chịu thua thôi, vì không có cách nào bảo Sub Class_Initialize dùng trứng vịt thay vì trứng gà ngay trong lúc đang tạo dựng ra Object bánh bông lan.

Sub New() trong VB.NET có thể nhận parameters nên nó có thể nhận chỉ thị để dùng trứng vịt ngay trong lúc nướng cái Object bánh bông lan.

Cái dạng đơn giản nhất của Sub New() mà ta có thể dùng là không pass parameter nào cả (trong trường hợp này thì giống như Sub Class_Initialize của VB6). Ta code Sub New() trong Class như sau:

```
Public Class BanhBongLan

  Public Sub New()

    ' Code to initialise object here

  End Sub

End Class
```

Ta instantiate một Object bánh bông lan như sau:

```
Dim myBanhBongLan As New BanhBongLan()
```

Để cho Users có sự lựa chọn khi instantiate Object, ta có thể code thêm những Sub New khác, mỗi Sub dùng một danh sách parameter khác nhau. Thí dụ:

```
Public Class BanhBongLan

  Public Sub New()

    ' Code to initialise object here

  End Sub


  Public Sub New(ByVal LoaiTrung As String)

    Select Case LoaiTrung

    Case "Vit"

      ' Code for TrứngVịt here

    Case "Ga"

      ' Code for TrứngGà here

    End Select

  End Sub


End Class
```

Dùng cùng một tên method để implement nhiều methods khác nhau được gọi là **overload**. Đó là một trường hợp đa dạng (polymorphism) của OO programming. Trong thí dụ trên nếu **TrứngVịt** và **TrứngGà** là hai loại Data Types khác nhau thì ta cũng có thể dùng:

```
  Sub New (ByVal TrứngVịt As TrứngVịtDataType)
```

để instantiate bánh TrứngVịt và

```
  Sub New (ByVal TrứngGà As TrứngGàDataType)
```

để instantiate bánh TrứngGà.

Như thế ta khỏi bận tâm với **Select Case LoaiTrung** khi chỉ dùng một Sub New duy nhất với 1 parameter.

Trong VisualStudio.NET, khi ta dùng tên của một overloaded method, **IntelliSense** sẽ hiển thị để hướng dẫn ta đánh vào parameter list khác nhau tùy theo method ta chọn.

Termination

Trong VB6 một Object sẽ bị huỷ diệt khi cái reference (chỗ dùng đến Object) cuối cùng bị lấy đi. Tức là khi không có code nào khác dùng Object nữa thì Object sẽ bị tự động huỷ diệt. System giữ một counter để đếm số clients đang dùng Object. Cách này hay ở chỗ khi counter trở thành 0 thì Object bị huỷ diệt ngay. Ta nói nó có **deterministic finalization**, nghĩa là ta biết rõ ràng khi nào Object biến mất.

Tuy nhiên, nếu ta có hai Object dùng lẫn nhau (gọi là **circular references**), thì ngay cả đến lúc chúng không còn hoạt động nữa, chúng vẫn hiện hữu mãi trong bộ nhớ vì cái Reference counter của cả hai Objects không bao giờ trở thành 0. Nếu trường hợp này xảy ra thường lần lần system không còn memory nữa, ta gọi đó là **memory leak (bị rỉ bộ nhớ)** .

.NET dùng phương pháp khác để quản lý chuyện này. Cứ mỗi chốc, một program sẽ chạy để kiểm xem có Object nào không còn reference nữa để huỷ diệt. Ta gọi đó là **Garbage Collection (nhặt rác)**. Ngay cả trường hợp hai Objects có circular references nhưng nếu không có code nào khác reference một trong hai Objects thì chúng cũng sẽ được huỷ diệt. Có điều, công tác nhặt rác chạy in the background (phía sau hậu trường) với ưu tiên thấp, khi CPU rảnh rang, nên ta không biết chắc một Object sẽ bị hủy diệt đến bao giờ mới thật sự biến mất. Ta nói nó có **nondeterministic finalization**.

Ta có thể ép CLR nhặt rác lập tức bằng code:

```
System.GC.Collect()
```

Tuy nhiên, ta chỉ làm việc ấy khi kẹt quá thôi. Tốt hơn, ta duyệt lại design của mình để cho phép các Objects hết xài có thể ngồi chơi trong bộ nhớ chờ đến lúc được hủy diệt.

Dùng Dispose Method

Nếu ta có một Object dùng nhiều tài nguyên (resources) như bộ nhớ, database connection, file handle,.v.v. và ta cần phải thả các tài nguyên ra ngay sau khi Object không còn hoạt động nữa, ta cần implement một **Interface** tên **IDisposable** với **Implements keyword** như sau:

```
Public Class TheClass

Implements IDisposable
```

Bạn phải viết code cho Sub Dispose giống như sau:

```
Private Sub Dispose() Implements IDisposable.Dispose

 ' Viết clean up code ở đây để thả các tài nguyên ra

End Sub
```

Sau đó bạn vẫn phải viết code cho Client để nó gọi Dispose Method trong IDisposable interface. Bạn cần phải dùng **CType** để cast Object Class khi gọi Dispose.

```
Dim objObject As New TheClass()

CType (objObject, IDisposable).Dispose()
```

Để lấy đi Reference đến một Object (gọi là **Dereference Object**) bạn có thể dùng:

```
myObject = Nothing
```

Để ý là ta không có dùng keyword Set như trong VB6. Nhớ là sau khi statement trên được executed thì myObject không biến mất ngay nhưng nó đợi Garbage Collector đến giải quyết.

Thừa kế

Thừa kế (Inheritance) là khả năng của một Class đạt được **interface** (giao diện) và **behaviours** (tánh tình) của một Class có sẵn. Cái quá trình để làm nên việc ấy được gọi là **Subclassing**. Khi ta tạo ra một Class mới thừa kế cả interface lẫn behaviours từ một Class có sẵn là chúng ta đã tạo ra một **subclass** của Class nguyên thủy. Người ta nói đó là một mối liên hệ **is-a** (là một), ý nói Class mới **là một** loại Class nguyên thủy.

Ta phân biệt mối liên hệ **is-a** với mối liên hệ **has-a** (có một). Trong mối

liên hệ has-a, Object chủ có thể làm chủ một hay nhiều Objects tớ, nhưng Object tớ là một loại có thể hoàn toàn khác với Object chủ.

Để biểu diễn đặc tính Inheritance ta hãy xét trường hợp một công ty cung cấp **Sản phẩm** và **Dịch vụ**. Ta có thể code một Class cho Sản phẩm (**ProductLine**) và một Class cho Dịch vụ (**ServiceLine**) , riêng rẽ nhau. Nhưng vì thấy chúng có nhiều điểm tương đồng nên ta sẽ code một Class gọi là Món hàng (**LineItem**), rồi inherit từ LineItem ra ProductLine và ServiceLine.

LineItem có các properties ID, Item, Price (giá) và Quantity (số lượng). Nó cũng có một Public Function để cho Amount (số tiền).

```
Public Class LineItem

  Private mintID As Integer

  Private mstrItem As String

  Private msngPrice As Single

  Private mintQuantity As Integer


  Public Property ID() As Integer

    Get

      Return mintID

    End Get

    Set (ByVal Value As Integer)

      mintID = Value

    End Set

  End Property


  Public Property Item() As String

    Get

      Return mstrItem

    End Get

    Set (ByVal Value As String)
```

```
        mstrItem = Value
    End Set
End Property

Public Property Price() As Single
    Get
        Return msngPrice
    End Get
    Set (ByVal Value As Single)
        msngPrice = Value
    End Set
End Property

Public Property Quantity() As Integer
    Get
        Return mintQuantity
    End Get
    Set (ByVal Value As Integer)
        mintQuantity = Value
    End Set
End Property

Public Function Amount() As Single
    Return mintQuantity * msngPrice
End Function

End Class
```

Để tạo Class ProductLine từ Class LineItem ta phải dùng **Inherits** keyword. Mỗi Object ProductLine **là một** Object LineItem với ProductID và Description. **ProductID** của ProductLine được pass vào Sub New lúc

instantiate Object ProductLine. Còn Description là một ReadOnly property của ProductLine. Ta có thể code Class ProductLine như sau:

```
Public Class ProductLine
  Inherits LineItem

  Private mstrDescription As String

  Public ReadOnly Property Description() As String
    Get
      Return mstrDescription
    End Get
  End Property

  Public Sub New(ByVal ProductID As String)
    Item = ProductID
    mstrDescription = "No description yet" ' Default description
   ' Viết code ở đây để đọc chi tiết của Product từ Database
   ' trong đó có thể có Description của Product
  End Sub
End Class
```

Statement **Inherits LineItem** khiến ProductLine thừa kế mọi interface và behaviours của LineItem. Do đó ta có thể code một **Sub BtnProduct_Click** để hiển thị chi tiết của ProductLine trong một Listbox như sau:

```
Protected Sub BtnProduct_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles BtnProduct.Click
  Dim pl As ProductLine
  pl = New ProductLine("P1234")
  ListBox1.Items.Add("ProductItem:" & pl.Item)
  ListBox1.Items.Add("Description: $" & pl.Description)
```

```
End Sub
```

Trong code bên trên ta dùng cả property Item của Class LineItem lẫn property Description của Class ProductLine. Cả hai đều là property của ProductLine vì nó là một SubClass của LineItem.

Giống như vậy, một ServiceLine có thể có ghi ngày giờ cung cấp service. Ta code Class ServiceLine như sau:

```
Public Class ServiceLine
  Inherits LineItem

  Private mdtDateProvided As Date

  Public Sub New()
    ' Make 1 as default number of services of this kind for invoice
    Quantity = 1
  End Sub

  Public Property DateProvided() As Date
    Get
      Return mdtDateProvided
    End Get
    Set (ByVal Value As Date)
      mdtDateProvided = Value
    End Set
  End Property

End Class
```

Một lần nữa ta dùng Statement Inherits để nói rằng ServiceLine là một SubClass của LineItem. Ta thêm property **DateProvided** vào interface thừa kế từ Class LineItem.

Bài 6
Những chức năng Đối Tượng mới của VB.NET (phần III)

Dùng OO trong VB.NET

Ngăn cản Thừa kế

Bình thường (By default) class nào cũng có thể được dùng làm base class để từ đó ta thừa kế. Nhưng đôi khi ta không muốn cho ai thừa kế từ một Class nào đó, để làm việc ấy ta dùng keyword **NotInheritable** khi declare class:

```
Public NotInheritable Class KhôngCon

End Class
```

Khi ta đã dùng keyword NotInheritable rồi thì không class nào có thể dùng keyword **Inherits** để tạo một subclass từ class ấy.

Thừa kế và Phạm vi hoạt động

Khi ta dùng đặc tính thừa kế để tạo một SubClass thì class mới này có đủ mọi methods, properties và variables với Access Modifier **Public** hay **Friend** của SuperClass. Bất cứ thứ gì declared là **Private** trong SuperClass thì SubClass không thấy hay dùng được.

*Có một ngoại lệ là **New** method. Các Constructor methods cần phải được implemented (định nghĩa)*
lại trong mỗi SubClass. Một chốc nữa ta sẽ bàn vào chi tiết về điểm này.

Để làm sáng tỏ vấn đề SubClass có thể dùng Class Members nào của SuperClass, ta thử code lại **Function Amount** trong **LineItem** class bằng cách khiến nó gọi một Private Function tên **CalculateAmount** để tính ra Amount thay vì để nó tính trực tiếp như trước đây:

```
Public Function Amount() As Single
  Return CalculateAmount
End Function
```

```
Private Function CalculateAmount() As Single
  Return mintQuantity * msngPrice
End Function
```

Khi ta SubClass LineItem để tạo ra ServiceLine class, bất cứ Object ServiceLine nào cũng thừa kế Function Amount vì Function ấy được declared Public trong BaseClass LineItem. Ngược lại, vì Function CalculateAmount là Private nên cả ServiceLine class lẫn bất cứ client code nào dùng một LineItem Object đều không truy cập nó được.

Như thế, mặc dầu ta gọi Function Amount được, nhưng đến phiên nó gọi Private Function CalculateAmount thì có bị trở ngại không? Không sao cả. Vì Function Amount nằm trong cùng Class với Private Function CalculateAmount nên nó có thể gọi được, dù rằng ta gọi Function Amount từ ServiceLine hay client code.

Thí dụ trong client code ta có những hàng code như sau:

```
Protected Sub BtnShowAmount_Click(ByVal sender As System.Object,  _
                    ByVal e As System.EventArgs) Handles BtnShowAmount.Click
  Dim Service As ServiceLine
  Service = New ServiceLine()
  Service.Item = "Delivery"
  Service.Price = 50
  Service.DateProvided = Now
  MessageBox.Show (Service.Amount.ToString, "Amount", MessageBoxButtons.OK,
MessageBoxIcon.Information)
End Sub
```

Kết quả sẽ được hiển thị trong message box, cho thấy Function CalculateAmount được Function Amount gọi dùm cho client code dù rằng cả client code lẫn ServiceLine code đều không thể gọi trực tiếp được.

Điểm này nhắc tôi nhớ lại khi còn bé, có lần bà con trong vườn đem ra chợ cho ba má tôi cả thúng xoài thơm rất ngon. Bạn tôi ở lối xóm thấy

vậy biểu tôi lén lấy hai trái xoài để ăn vụn. Vì không phải là người nhà nên bạn tôi không thể lấy được xoài, bởi Access Modifier của thún xoài là Private trong nhà tôi. Nhưng vì tôi là Public, nên bạn tôi có thể nhờ tôi lấy dùm.

Protected Methods

Đôi khi **Public** hay **Private** thôi chưa đủ. Nếu ta declare thứ gì Private thì nó hoàn toàn giới hạn trong class, ngược lại nếu ta declare nó Public (hay Friend) thì nó có thể được dùng trong subclasses hay client code.

Tuy nhiên, có lúc ta muốn một class member chỉ có thể được dùng trong subclasses thôi, chớ không cho client code dùng. Trong trường hợp ấy ta dùng keyword **Protected**. Thí dụ:

```
Public Class FatherClass
  Protected DiSản As Single
End Class

Public Class SonClass
  Inherits FatherClass
  Public Function ChiaCủa() As Single
    Return Disản
  End Function
End Class
```

Ở đây ta có BaseClass **FatherClass** với **Protected Field Disản**. Không có client code nào có thể thấy Field DiSản được. Thế nhưng bất cứ SubClass nào của FatherClass cũng đều thừa kế và dùng được DiSản.

Trong thí dụ trên, một lần nữa SubClass có một Public method (ChiaCủa) có thể return một protected value - nhưng chính value ấy, DiSản, không trực tiếp cho phép client code dùng.

Overriding Methods

Chúng ta biết rằng đặc tính quan trọng của Inheritance là một SubClass chẳng những thừa kế behaviours của ParentClass mà còn có thể **override**

(lấn quyền) các behaviours ấy nữa. Chúng ta đã thấy một SubClass có thể **extend** (thêm ra) ParentClass bằng cách cho thêm các methods Public, Protected và Friend. Hơn nữa, khi dùng overriding, một SubClass có thể **alter** (sửa đổi) behaviours của các methods trong ParentClass.

Bình thường (By default), ta không thể override methods trong ParentClass trừ khi các methods ấy được declared với keyword **Overridable** trong ParentClass. Thí dụ:

```
Public Class ClassCha
  Public Overridable Sub ChàoHỏi()
    MessageBox.Show("Chào các cháu", "Class Cha")
  End Sub
End Class
```

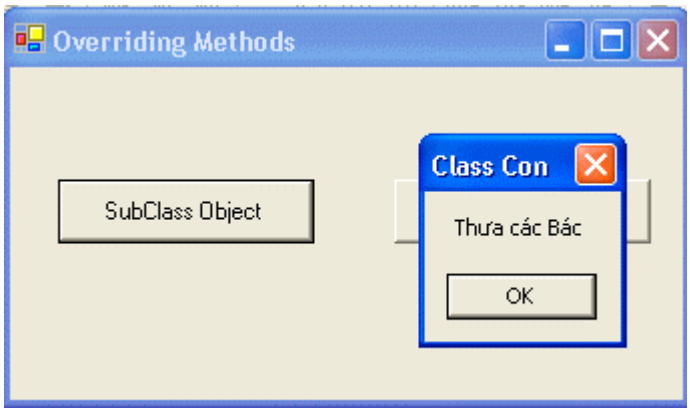
Tiếp theo, khi tạo một SubClass, nếu muốn ta có thể override behaviour của **Sub ChàoHỏi** bằng cách dùng keyword **Overrides** như sau:

```
Public Class ClassCon
  Inherits ClassCha
  Public Overrides Sub ChàoHỏi()
    MessageBox.Show("Thưa các Bác", "Class Con")
  End Sub
End Class
```

Bây giờ ta có thể viết client code như sau:

```
Private Sub BtnSubClassObject_Click(ByVal sender As System.Object,  _
                ByVal e As System.EventArgs) Handles BtnSubClassObject.Click
  Dim obj As New ClassCon()
  obj.ChàoHỏi()
End Sub
```

Khi ta click button BtnSubClassObject program sẽ hiển thị message dialog dưới đây:



Virtual Methods

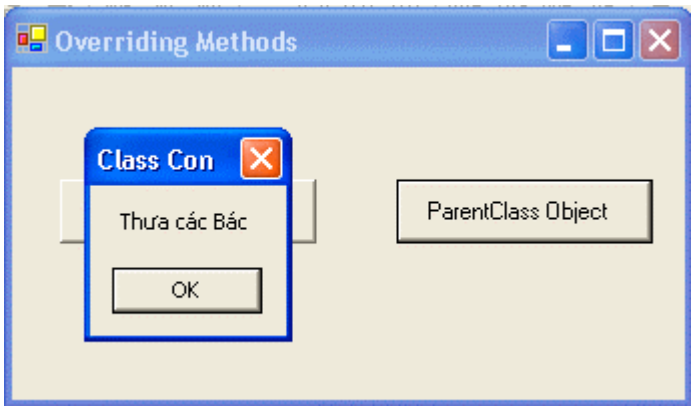
Tuy nhiên, hãy xem trường hợp ta code như sau:

```
Private Sub BtnParentClassObject_Click(ByVal sender As System.Object, _
                ByVal e As System.EventArgs) Handles BtnParentClassObject.Click
  Dim obj As ClassCha
  obj = New ClassCon()
  obj.ChàoHỏi()
End Sub
```

Trước hết, ở đây có vẻ kỳ kỳ, tại sao declare một variable loại ClassCha mà lại instantiate một object ClassCon. Chuyện đó hoàn toàn bình thường, vì ClassCon **là một** ClassCha. Tức là một variable loại ClassCha hay ClassCon đều có thể chứa, thật ra là **hold references to** (point to, chỉ tới), một instance của ClassCon.

Điểm này áp dụng tổng quát khi ta dùng Inheritance. Một variable loại SuperClass có thể hold reference to bất cứ SubClass Object nào thừa kế từ SuperClass ấy. Đó là một cách để ta implement tính đa dạng (polymorphism).

Đều có thể làm ta ngạc nhiên là khi ta click button BtnParentClassObject ta cũng thấy hiển thị message " **Thưa các Bác**".



Sao lạ vậy? Variable **obj** được declared là ClassCha tại sao message không phải là "**Chào các cháu**"? Lý do là **Sub ChàoHỏi** của **ClassCon** được gọi thay vì Sub ChàoHỏi của ClassCha. Ta nói Sub ChàoHỏi là **Virtual** method. Tất cả methods trong VB.NET đều là virtual.

Ý niệm virtual để nói rằng cái implementation của **con cháu trẻ nhất trong dòng họ** được dùng - không cần biết là variable có **data type** là class của thế hệ nào trong dòng họ. Tức là, nếu variable dùng trong client code hold references to ClassÔngNội, ClassCha, ClassCon hay ClassCháu thì method trong ClassCháu được gọi. Nếu trong ClassCháu không có implementation của method thì ta gọi method trong ClassCon, nếu không có thì gọi method trong ClassCha .v.v.. theo thứ tự từ bề dưới lên bề trên.

Keyword Me

Keyword **Me** được dùng khi ta muốn nói rõ (explicitly) rằng ta muốn dùng method của chính cái Class đang chứa code ấy, chớ không phải một implementation nào khác của method ấy.

Cũng có trường hợp ta phải dùng keyword Me để nói ta muốn dùng **class-level** variable chớ không phải **procedure-level** variable có cùng tên. Một procedure-level variable, tức là local variable của một method, có cùng tên với một class-level variable được gọi là **shadowed** variable. Thí dụ:

```
Public Class TheClass
  Private strName As String
  Public Sub DoSomething()
    Dim strName As String
```

```
    strName = "Quang"
  End Sub
End Class
```

Ở đây, variable **strName** được declared ở class-level và bên trong **Sub DoSomething**. Bên trong method ấy local variables (kể cả shadowed variables) sẽ được dùng vì chúng che đậy class-level variables trừ khi ta nói rõ rằng phải dùng variable của class-level bằng cách dùng keyword Me:

```
Public Class TheClass
  Private strName As String
  Public Sub DoSomething()
    Dim strName As String
    strName = "Quang" ' thay đổi value của local (shadowed) variable
    Me.strName = "Kim" ' thay đổi value của class-level variable
  End Sub
End Class
```

Keyword MyBase

Keyword Me rất tiện dụng khi ta muốn dùng Class members của chính Class chứa code. Tương tự như vậy, đôi khi ta muốn dùng Class method của BaseClass (cũng gọi là SuperClass), chớ không phải một implementation của method ấy trong SubClass. Nhớ là một virtual method luôn luôn gọi implementation của Class trẻ nhất.

Từ trong một SubClass, nếu muốn gọi một method của BaseClass ta dùng keyword **MyBase** như sau:

```
Public Class ClassCon
  Inherits ClassCha
  Public Overrides Sub ChàoHỏi()
    MessageBox.Show("Thưa các Bác", "Class Con")
    MyBase.ChàoHỏi()
  End Sub
```

```
End Class
```

Bây giờ nếu ta chạy Sub ChàoHỏi của ClassCon ta sẽ có hai messages, một cái từ ClassCon theo sau bởi một cái từ ClassCha.

MyBase chỉ nói đến BaseClass trực tiếp, tức là Class cha thôi chớ không nói đến Class ông nội. Không có cách nào để nói đến hơn một thế hệ.

Dầu vậy, keyword Mybase có thể được dùng cho bất cứ thứ gì đã được declared Public, Friend hay Protected trong ParentClass. Điều này kể luôn cả những thứ mà ParentClass thừa kế từ các thế hệ trước trong gia đình, tức là ClassÔngNội, ClassÔngCố .v.v..

Keyword MyClass

Vì lý do virtual method, ta sẽ gặp những trường hợp rắc rối như khi code của ParentClass lại chạy code của SubClasses.

Khi viết code của một class, từ method này ta thường gọi những methods khác nằm trong cùng class. Thí dụ như:

```
Public Class ClassCha
  Public Sub VôĐề()
    ChàoHỏi()
  End Sub

  Public Overridable Sub ChàoHỏi()
    MessageBox.Show("Chào các cháu", "Class Cha")
  End Sub
End Class
```

Trong trường hợp này, **VôĐề** gọi **Sub ChàoHỏi** để đón tiếp. Để ý là vì ChàoHỏi được declared Overridable nên rất có thể một SubClass sẽ implement method ChàoHỏi và lấn quyền nó. Thí dụ:

```
Public Class ClassCon
  Inherits ClassCha
  Public Overrides Sub ChàoHỏi()
    MessageBox.Show("Thưa các Bác", "Class Con")
```

```
  End Sub

End Class
```

Vì đặc tính virtual của ChàoHỏi nên ta tưởng ClassCha execute chính Sub ChàoHỏi của nó nhưng té ra nó lại execute code của ChàoHỏi trong ClassCon. Trong code dưới đây, một Object ClassCon gọi Sub VôĐề của ClassCha:

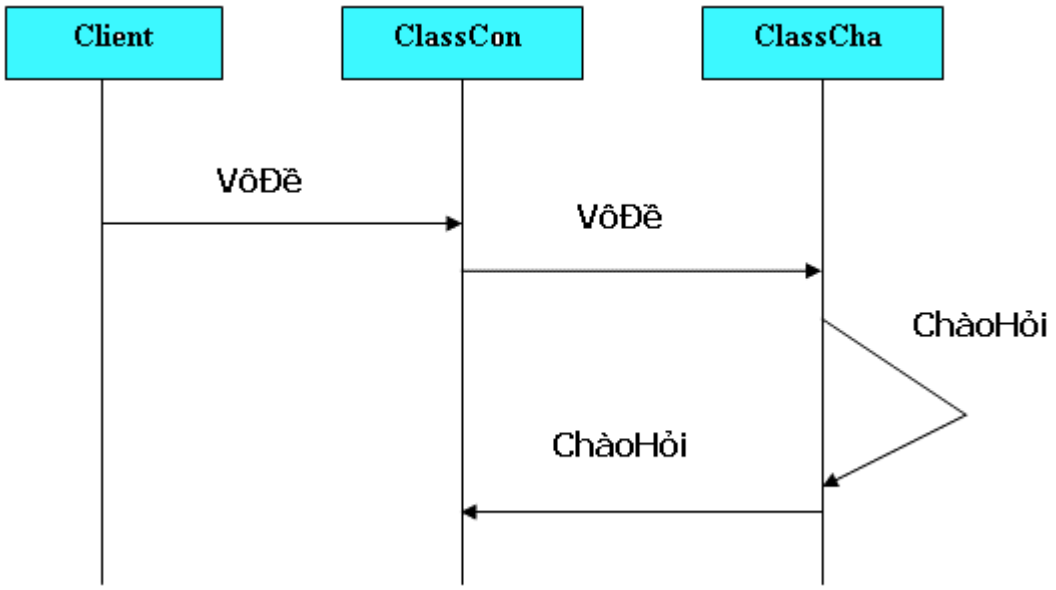
```
Private Sub BtnSubClassObject_Click(ByVal sender As System.Object, _
                    ByVal e As System.EventArgs) Handles BtnSubClassObject.Click
  Dim obj As New ClassCon()
  obj.VôĐề()
End Sub
```

Trong ClassCha, Sub VôĐề gọi ChàoHỏi của chính nó, tuy nhiên Sub ChàoHỏi ấy bị overridden bởi implementation của ChàoHỏi trong ClassCon. Do đó, program sẽ hiển thị message "**Thưa các Bác**".

Nếu ta không muốn như vậy, ta muốn VôĐề execute chính code của ChàoHỏi trong ClassCha thì phải dùng keyword **MyClass** như sau:

```
Public Class ClassCha
  Public Sub VôĐề()
    MyClass.ChàoHỏi()
  End Sub

  Public Overridable Sub ChàoHỏi()
    MessageBox.Show("Chào các cháu", "Class Cha")
  End Sub
End Class
```

Ở đây ta không thể dùng keyword **Me** vì VôĐề có gọi ChàoHỏi ở class-level trong ClassCha chớ không phải trong một SubClass, nhưng bị overridden. Hình dưới đây minh họa quá trình gọi VôĐề từ client code:



Sub VôĐề thật ra nằm trong ClassCha mà ClassCon thừa kế nên VôĐề được executed trong ClassCha và gọi Sub ChàoHỏi trong cùng class (ClassCha). Nhưng vì ClassCon có một implementation của Sub ChàoHỏi nên nó overrides ChàoHỏi của ClassCha.

Overridding Method New

Chúng ta đã thấy ta có thể override methods và dùng các keywords Me, MyBase và MyClass để gọi các overriden methods trong dây chuyền thừa kế. Tuy nhiên, đối với Constructor của class thì có những luật lệ đặc biệt dành riêng cho method **New**.

Những methods New không tự động di truyền từ BaseClass xuống SubClass. Mỗi SubClass phải có một implementation riêng cho Constructor dù rằng, nếu muốn, nó có thể gọi vào BaseClass với keyword MyBase:

```
Public Class ClassCon
  Inherits ClassCha
  Public Sub New()
    MyBase.New()
   ' để thêm các code khác để initialise tại đây
  End Sub
End Class
```

Khi gọi Constructor của BaseClass, ta phải gọi nó trước nhất - nếu không sẽ bị error. Tuy nhiên ta không cần gọi Constructor của BaseClass vì Constructor của BaseClass được gọi tự động.

Có một luật đặc biệt là nếu tất cả methods New trong BaseClass đều đòi hỏi parameters thì ta phải implement ít nhất một method New trong SubClass và ta phải đặt statement **MyBase.New** ngay phía đầu.

Dĩ nhiên là ta có thể Overload method New trong SubClass, nhưng ta phải tự lo liệu cách gọi một method New thích hợp trong BaseClass.

Tạo BaseClasses và Abstract Methods

Cho đến giờ ta đã bàn về virtual method với đặc tính override trong nguyên tắc thừa kế. Trong các thí dụ trước đây BaseClass được instantiated thành Object để làm chuyện này, chuyện kia. Nhưng đôi khi ta muốn tạo một BaseClass *chỉ để dùng* cho thừa kế mà thôi.

Keyword MustInherit (Phải được Thừa Kế)

Trở lại cái thí dụ về Inheritance với Class LineItem. Sở dĩ ta đặt ra Class LineItem là vì nó chứa những thứ chung cho cả hai classes ProductLine và ServiceLine. Chớ thật ra một Object của Class LineItem không chứa đủ mọi đặc tính để làm một việc gì thực tế. Nếu ta muốn nói rõ rằng Class LineItem chỉ được dùng để tạo những SubClasses bằng cách thừa kế từ nó, ta có thể declare như sau:

```
Public MustInherit Class LineItem
```

Tức là ta chỉ thêm keyword **MustInherit** thôi, chớ không thay đổi gì khác. Kết quả là từ nay Client code không thể instantiate một Object từ Class LineItem. Do đó dòng code sau sẽ bị syntax error:

```
Dim myObject As New LineItem()
```

Thay vào đó, nếu muốn dùng LineItem ta phải tạo SubClass từ nó.

Keyword MustOverride (Phải bị Lấn Quyền)

Tương tự với ý niệm Phải-được-thừa-kế trong Class, ta cũng có **MustOverride** cho một method. Có thể trong BaseClass ta khai báo một method, nhưng ta đòi hỏi method ấy phải có một implementation trong SubClass. Ta declare như sau:

```
Dim MustOverride Sub CalculatePrice
```

Để ý là ở đây không có thân thể của Sub CalculatePrice hay statement End Sub gì cả. Khi dùng MustOverride ta không được phép cung cấp một implementation cho method trong BaseClass. Một method như thế được gọi là **abstract method** hay **pure virtual function**, vì nó chỉ có phần khai báo chớ không có phần định nghĩa. Những abstract methods phải được overridden trong bất cứ SubClass nào của BaseClass thì mới dùng được. Nếu không, ta sẽ không có phần implementation của method đâu cả và khi compile sẽ gặp syntax error.

Abstract Base Classes

Nếu hợp cả hai ý niệm MustInherit và MustOverride lại ta sẽ tạo ra một **abstract base class**. Đây là một Class chỉ có khai báo chớ hoàn toàn không có implementation. Ta phải SubClass từ nó thì mới làm việc được, thí dụ như:

```
Public MustInherit Class ClassCha

  Public MustOverride Sub VôĐề()

  Public MustOverride Sub ChàoHỏi()

End Class
```

Kỹ thuật này rất thích hợp để ta code cái sườn hay bố cục của program ngay trong lúc thiết kế. Class nào thừa kế **ClassCha** thì phải implement cả **Sub VôĐề** lẫn **Sub ChàoHỏi**, nếu không sẽ bị syntax error.

Nhìn về một phương diện, abstract base class rất giống khai báo Interface. Nếu dùng Interface, chúng ta có thể khai báo như sau:

```
Public Interface ICha

  Sub VôĐề()

  Sub ChàoHỏi()

End Interface
```

Bất cứ class nào chịu implement interface **ICha** thì phải implement cả **Sub VôĐề** lẫn **Sub ChàoHỏi**, nếu không sẽ bị syntax error - do đó, ta thấy Interface rất giống một abstract base class.

Sự khác biệt chính giữa abstract base class với Interface là ở chỗ thừa kế. Khi ta tạo một class con bằng cách SubClass từ ClassCha, chính class con ấy lại cũng có thể được SubClassed. Mấy class cháu này sẽ tự động thừa kế VôĐề và ChàoHỏi từ class con.

Trong khi ấy nói về Interface, mỗi class phải tự implement ICha một cách độc lập và phải cung cấp hai Subs VôĐề và ChàoHỏi của chính nó. Vì thế, nếu ta không có ý định dùng lại code của các Subs khi ta tạo các classes mới thì ta có thể dùng interface. Ngược lại nếu ta muốn dùng lại code trong SubClass theo nguyên tắc thừa kế thì ta nên dùng abstract base class.

Bài 7
Những chức năng Đối Tượng mới của VB.NET (phần IV)

Dùng OO trong VB.NET

Shared class members (Các thành viên để dùng chung của class)

Mặc dù Object rất hiệu năng và hữu ích, có khi ta chỉ muốn truy cập các variables hay methods của một class để làm việc mà không cần phải instantiate một Object nào cả. Tức là y như trong quá khứ, khi viết VB6, ta dùng các variables hay methods của một BAS Module. Đại khái giống như thay vì ký giao kèo với một thầu (Object) để thực hiện một công trình, ta chỉ muốn mướn thợ hay chuyên viên làm việc gia công (gọi các methods) thôi.

Shared Methods

Trong VB.NET chẳng những một Class có các methods và properties thông thường như ta đã thấy - tức là những methods và properties của một Object ta có thể dùng ngay sau khi Object ấy thành hình qua quá trình instantiation - mà còn có các methods và properties ta có thể dùng mà không cần phải tạo ra một instance nào từ Class. Chúng được gọi là **shared methods**. (Trong các ngôn ngữ lập trình khác các methods này còn được gọi là **static methods** hay **class methods**).

Ta không thể truy cập một shared method qua một Object như method bình thường, nhưng phải dùng trực tiếp **tên của class**. Thí dụ sau đây sẽ minh họa điều này:

```
Public Class Math

  Shared Function Add( ByVal x As Single, ByVal y As Single) As Single

    Return x + y

  End Function

End Class
```

Sau khi định nghĩa **Class Math**, ta có thể dùng **Shared Function Add** mà không cần instantiate một Object thuộc class Math như sau:

```
Dim Result As Single

result = Math.Add(12.5, 36.8)
```

Để ý thay vì dùng một object variable ta dùng thẳng tên của class Math để truy cập method Add. Với một method bình thường thì làm như thế sẽ bị syntax error, nhưng trong trường hợp này thì không sao.

Ta cũng có thể **overload** shared methods, tức là có thể code nhiều shared methods với cùng một tên nhưng có những parameter lists khác nhau.

Phạm vi hoạt động bình thường (**Default Scope**) của shared methods là **Public**. Tuy nhiên ta có thể giới hạn việc truy cập chúng bằng cách dùng những Access Modifiers như **Friend, Protected** hay **Private**. Thật ra khi overloading một shared method ta có thể dùng những scopes khác nhau cho mỗi shared method.

Có một thí dụ về shared method từ .NET system class libraries. Để mở một text file theo mode input, điển hình ta dùng shared method trong **File** class như sau:

```
Dim inFile As StreamReader = File.OpenText("words.txt")

Dim strIn As String


strIn = inFile.ReadLine()
```

Ở đây không có object File nào được tạo ra. Method **OpenText** là một shared Function, nó mở input text file **words.txt** và cho ta một object loại StreamReader tên **inFile** để ta dùng sau đó.

Shared Variables

Đôi khi ta muốn tất cả objects của cùng một class đều dùng chung một variable. Ta có thể thực hiện việc ấy với **shared variables**.

Một shared variable được khai báo với keyword **shared** giống như shared method:

```
Public Class MyCounter

  Private Shared mintCount As Integer

End Class
```

Ta có thể cho shared variable một scope Public hay Private tùy ý, nhưng By Default, scope của shared variables là **Private**, khác với shared methods thì By Default là **Public**.

Điểm quan trọng của shared variables là chúng được dùng chung giữa mọi instances (objects) của cùng một class. Dưới đây là một thí dụ trong đó ta giữ cái counter có trị số tăng thêm 1 mỗi lần có một instance mới của **class MyCounter**. Bất cứ lúc nào ta cũng có thể biết có bao nhiêu objects đã được tạo ra bằng cách đọc **property Count**:

```
Public Class MyCounter

  Private Shared mintCount As Integer


  Public Sub New()

    mintCount += 1

  End Sub


  Public ReadOnly Property Count() As Integer

    Get

      Return mintCount

    End Get

  End Property

End Class
```

Như thế, nếu ta chạy client code dưới đây nó sẽ hiển thị kết quả là **3**:

```
Protected Sub Button1_Click( ByVal sender As Object, ByVal e As System.EventArgs) Handles
Button1.Click

  Dim obj As MyCounter


  obj = New MyCounter()

  obj = New MyCounter()

  obj = New MyCounter()

  MsgBox(obj.Count, MsgBoxStyle.Information, "Counter")
```

```
End Sub
```

Nếu ta chạy code thêm hai lần nữa, ta sẽ có **6** và **9**. Hể ta còn chạy chương trình thì cái counter còn làm việc. Khi ta chấm dứt chương trình thì cái counter sẽ biến mất.

Global values

Một cách dùng rất thông dụng khác của shared variable là xem nó như một loại **Global variable**. Khi dùng scope Public ta sẽ có một dạng tương đương với VB6 Global variable trong một BAS Module. Thí dụ như:

```
Public Class GlobalData

  Public Shared TotalCost As Single

End Class
```

Sau đó ta có thể dùng variable này khắp nơi trong client code:

```
GlobalData.TotalCost += 45.60
```

Events

Raising Event để xử lý trong một Project khác

VB.NET không hổ trợ **Events** từ đời cha đến đời con theo đúng nguyên tắc thừa kế. Nếu một BaseClass định nghĩa một Public Event thì ta chỉ có thể **raise** event ấy trong code của BaseClass thôi chớ không thể **raise** event ấy trong SubClass nào của BaseClass ấy.

Khác với methods, ta không thể **overload** một Event, tức là không thể dùng một tên cho hai Events có parameter list khác nhau.

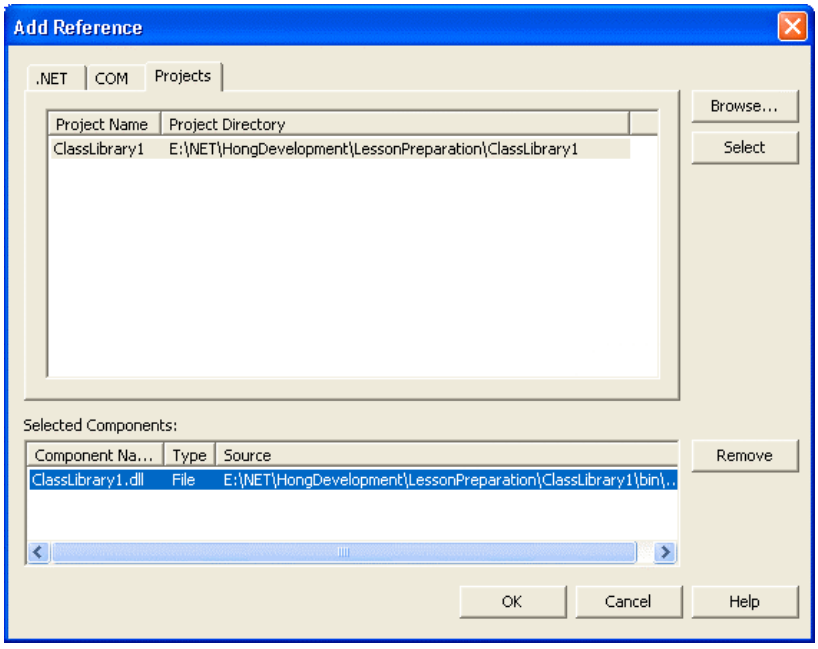
Ta có thể tạo một Class Library Project với một Class trong đó có raise một Event rồi tạo một project khác trong đó có code để đón nhận và xử lý Event ấy.

Để thử việc này bạn hãy tạo một Class Library Project mới với tên **ClassLibrary1** về viết những dòng code định nghĩa **Class Class1** với **Event TheEvent** và **Sub LàmViệc** để raise Event như sau:

```
Public Class Class1

  Public Event TheEvent()
```

```
  Public Sub LàmViệc()
    RaiseEvent TheEvent()
  End Sub
End Class
```

Kế đó bạn dùng Menu command **File | Add Project | New Project** để thêm một project mới với tên **EventClass**. Để có thể dùng Class1, bạn cần phải reference nó với Menu command **Project | Add Reference...**, chọn Tab **Projects** và click **Browse** để chọn **ClassLibrary1.DLL** từ subfolder **ClassLibrary1\bin** của solution như trong hình dưới đây:



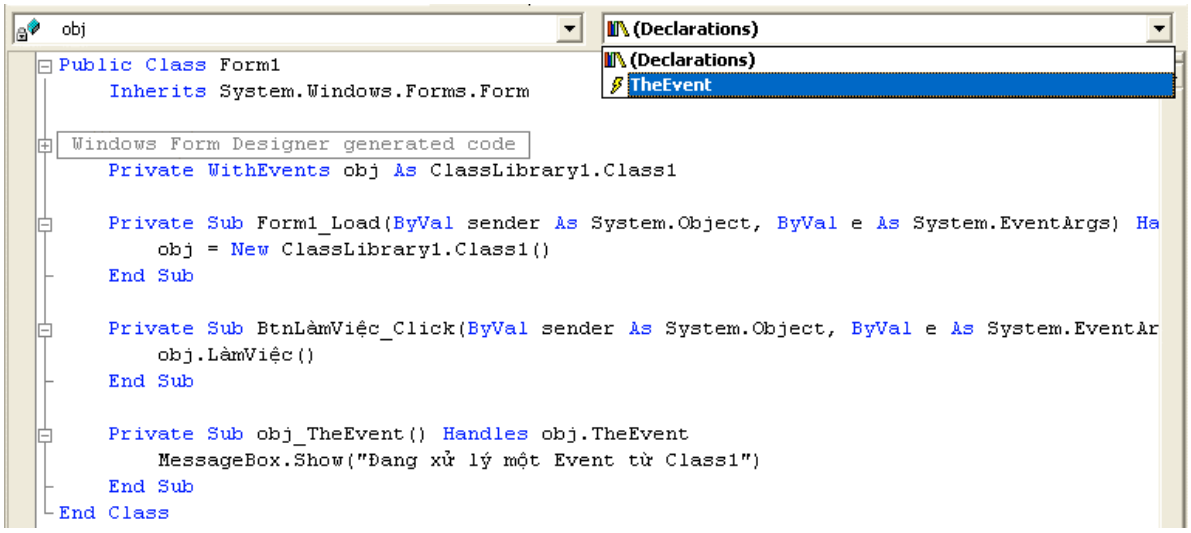
Một khi đã referenced ClassLibrary1 với Class1 trong ấy, bây giờ bạn có thể doubleclick lên Form1 để code như sau:

```
Private WithEvents obj As ClassLibrary1.Class1

Private Sub Form1_Load( ByVal sender As System.Object, _
                ByVal e As System.EventArgs) Handles MyBase.Load
  obj = New ClassLibrary1.Class1()
End Sub
```

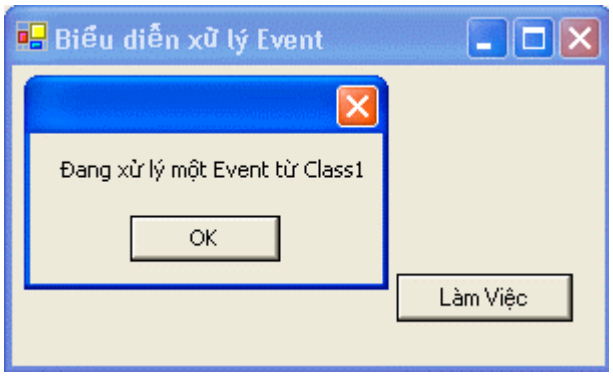
Nhớ là ta phải declare variable **obj** thuộc loại **ClassLibrary1.Class1** với **WithEvents**. Đặt một Button tên BtnLàmViệc và doubleclick lên nó để code như sau:

```
Private Sub BtnLàmViệc_Click( ByVal sender As System.Object, _
                              ByVal e As System.EventArgs) Handles BtnLàmViệc.Click
  obj.LàmViệc()
End Sub
```

Để xử lý Event của **obj** bạn chọn tên từ combobox phía trên bên trái, rồi chọn **TheEvent** từ combobox bên phải như trong hình dưới đây:



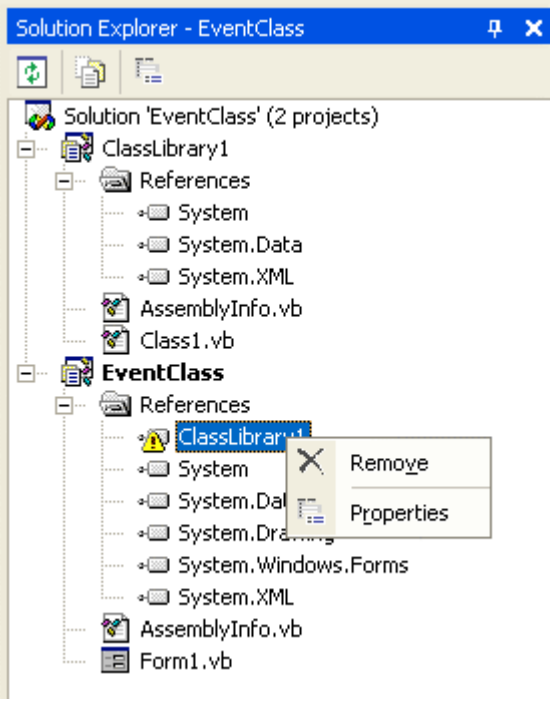
Ở đây ta handle Event bằng cách hiển thị một message đơn giản: **Đang xử lý một Event từ Class1**. Bây giờ bạn có thể chạy program. Khi bạn click Button BtnLàmViệc program sẽ hiển thị message để chứng minh rằng từ một Application ta có thể handle event trong Class của một Project khác.



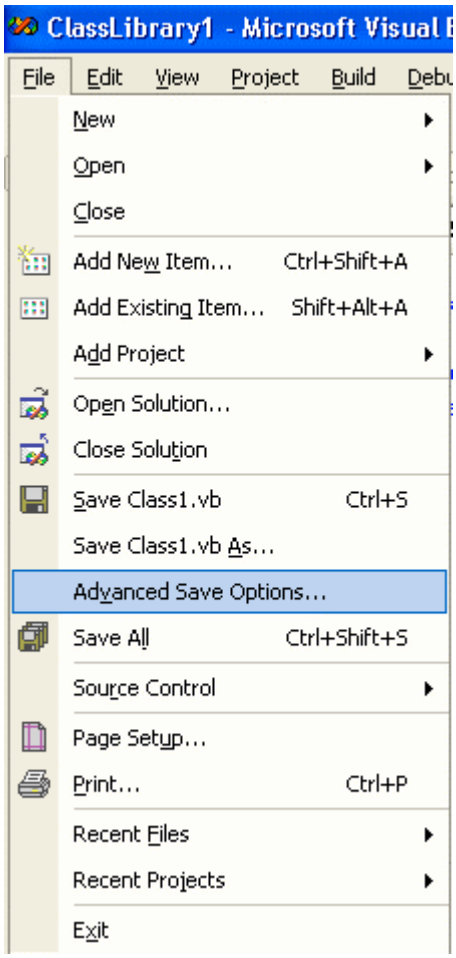
Ghi chú

Nếu sau khi Unzip source file và load project vào, bạn dùng IDE Menu command **Build | Rebuild Solution** để compile lại hết các modules nhưng gặp error về references thì hãy làm như sau:

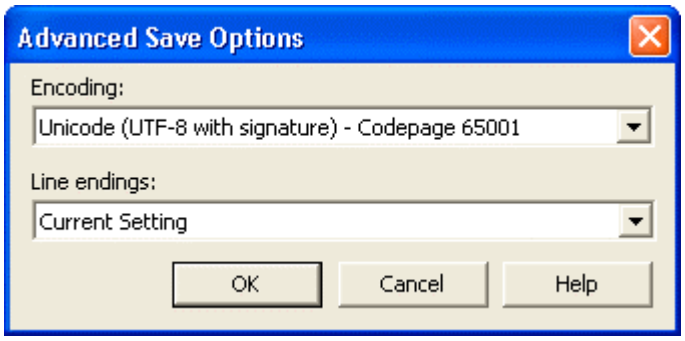
- Trong Solution Explorer click các tree nodes **references** để tìm các references có dấu chấm thang trong tam giác vàng và remove chúng.
- Dùng Menu command **Project | Add Reference...** để chọn ***.dll** lại từ một **\bin** subfolder.
- Rebuild Solution.



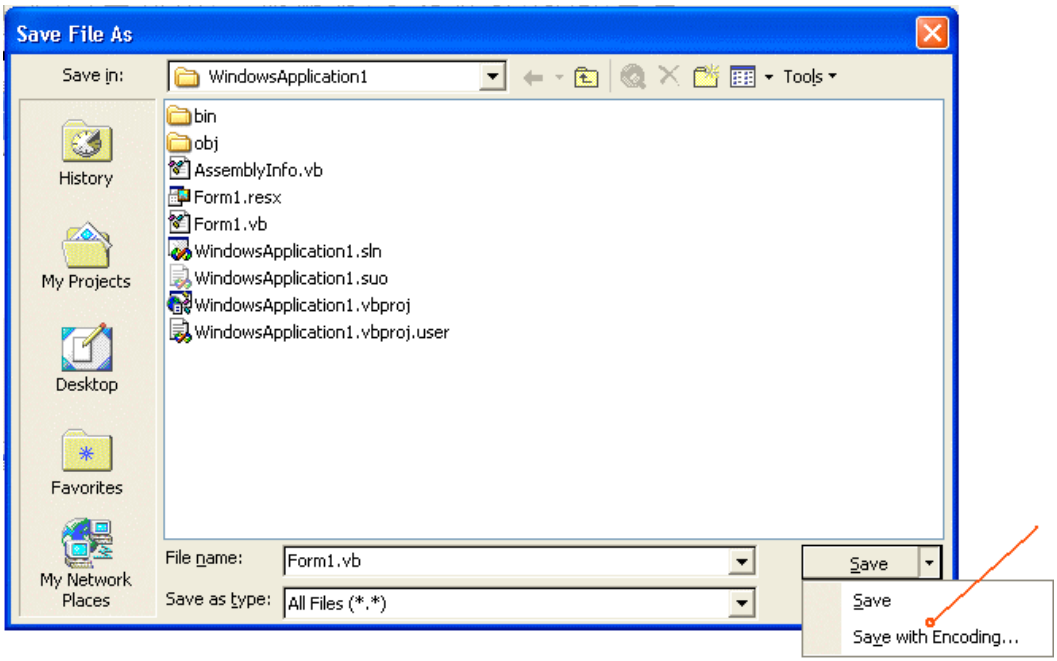
Nếu bạn dùng chữ Việt Unicode trong program thì nhớ set up **Advanced Save Option** với Menu command **File** như trong hình dưới đây:



Khi Dialog hiện ra, bạn chọn **Unicode (UTF-8)** cho **Encoding**:



Nếu bạn không thấy có menuItem Advanced Save Option trong Menu File thì cứ dùng menuItem **Save As...** rồi click lên combo box **Save** phía dưới, bên phải của Save File As Dialog rồi chọn **Save with Encoding...** như trong hình dưới đây:



Nếu bạn quên set up Advanced Save Option như trên, chữ Việt sẽ bị lưu trử dưới dạng ANSI nên một số sẽ mất dấu chữ Việt và thay vào đó bằng những dấu **?**.

Shared Events

Events có thể được declared là **Shared**. **Shared methods** chỉ có thể raise shared events, chúng không thể raise non-shared events. Thí dụ như:

```
Public Class NguồnEvent

  Shared Event EventDùngChoSharedMethods()


  Public Shared Sub DùngChung()

    RaiseEvent EventDùngChoSharedMethods()

  End Sub

End Class
```

Một shared event có thể được raised bởi cả shared methods lẫn non-shared methods:

```
Public Class NguồnEvent

  Public Event TheEvent()
```

```
Shared Event EventDùngChoSharedMethods()

Public Shared Sub DùngChung()
  RaiseEvent EventDùngChoSharedMethods()
End Sub

Public Sub LàmViệc()
  RaiseEvent TheEvent()
  RaiseEvent EventDùngChoSharedMethods()
End Sub
End Class
```

Nếu bạn tìm cách raise một non-shared event từ một shared method thì sẽ bị syntax error.

Early Binding hay Late Binding (Hiệu lực Sớm hay Trễ)

Early Binding có nghĩa là program biết rõ ngay từ đầu loại Object (thuộc Class nào) sẽ được dùng trong hoàn cảnh nào. Nó cho phép IntelliSense hiển thị cho ta thấy những class members nào ta có thể dùng và compiler kiểm xem những methods ta dùng có hiện hữu không. Early Binding code được compiled ra IL rất hiệu năng vì compiler biết rõ ràng data types của các parameters.

Ngược lại **Late Binding** có nghĩa là ta làm việc cách linh động với một Object lúc run-time, tức là program không biết trước Object ấy thuộc loại nào. Late Binding cho ta sự uyển chuyển chỉ làm sao Object cung cấp đúng method cần thiết là đủ. Do đó, ta không hưởng được sự sang trọng IntelliSense cung cấp và compiler không thể kiểm soát loại Object trước dùm cho ta được. Mặc dầu Late Binding code chạy chậm hơn nhưng nó cho ta sự tự do giống như khi làm việc ngoài đời, để đến giờ chót mới xác nhận.

By Default, mọi objects trong VB.NET đều là Late Bound. Visual Studio.NET IDE với **Option Strict Off** by default áp đặt luật đó. Nếu muốn áp đặt Early Binding ta cần phải nhét câu **Option Strict On** ở đầu một source file.

Dùng Object Type

Ta có Late Binding khi compiler không thể xác định loại Object ta đang gọi. Ta có thể thực hiện điều này bằng cách dùng **Object Type** để tuyên bố một cách mơ hồ rằng ta sẽ dùng một loại Object nào đó, vì một variable với Object type có thể hold-reference-to bất cứ một Object nào. Do đó, những dòng code sau đây có thể được dùng cho bất cứ Object nào mà Class của nó có implement **Sub CôngTácTôi** và không dùng parameter nào cả:

```
Option Strict Off


Module LateBind
  Public Sub LàmViệc( ByVal obj As Object)
    obj.CôngTácTôi()
  End Sub
End Module
```

Nếu **obj** passed vào **Sub LàmViệc** không có một **Sub CôngTácTôi** chẳng dùng parameter nào hết thì program sẽ bị error lúc run-time. Do đó, ta nên luôn luôn dùng một **Try Structure** để bắt cái error đó. Thí dụ như:

```
Option Strict Off


Module LateBind
  Public Sub LàmViệc( ByVal obj As Object)
    Try
      obj.CôngTácTôi()
    Catch e As Exception
     ' Code để xử lý trường hợp Object không thích hợp
      Console.WriteLine("Invalid Object passed to LàmViệc")
    End Try
  End Sub
End Module
```

Late Binding và Reflection

.NET framework hổ trợ một ý niệm gọi là **reflection**. Nó nói đến khả năng của program kiểm tra .NET code để biết trong code có những thứ gì. Ta dùng namespace **System.Reflection** để viết code làm chuyện ấy.

Với System.Reflection ta có thể viết code để khám phá những classes nằm trong một **assembly**, để biết mỗi class có những methods, properties và events nào. Tiếp theo đó, ta có thể dùng reflection để instantiate và dùng những objects từ các classes ấy. Cả quá trình này hoàn toàn linh động - giống hệt như Late Binding.

Thật ra, CLR (Common Language Runtime) dùng reflection để implement Late Binding dùm cho chúng ta. Thay vì bắt chúng ta phải tự dùng reflection để code Late Binding, .NET đã tử tế lo lắng chuyện ấy một cách tự động cho chúng ta.

Dùng Function CType

Dầu ta có dùng Late Binding hay không, nhiều khi rất tiện để ta pass reference đến một object nào đó, từ chỗ này đến chỗ khác, bằng cách dùng Data Type **Object** tổng quát - khi nào cần dùng nó thì ta đổi nó ra đúng loại Object trong hoàn cảnh. Ta thực hiện việc convert data type bằng cách dùng **Function CType**, điều đó cho phép ta nói trước Data Type Object sẽ được converted ra object của class nào để gọi một method theo cách **Early Bound**:

```
Module LateBind

  Public Sub LàmViệc( ByVal obj As Object)

     CType(obj, TheClass).CôngTácTôi()

  End Sub

End Module
```

Trong thí dụ trên dù rằng ta đang làm việc với variable thuộc type Object - trên nguyên tắc thì có vẽ là **Late Bound** - nhưng chúng ta đang dùng **Function CType** để convert **obj** ra một object thuộc class **TheClass**.

Kỹ thuật này được gọi là **casting** (đổ khuôn). Nếu ta xem **TheClass** như một cái khuôn, khi ta ép **obj** vào khuôn ấy thì giống như đổ khuôn để cho obj có dạng của TheClass.

Function CType rất hữu dụng khi ta làm việc với những objects có implement nhiều **interfaces**, vì ta có thể dùng cùng một object cho những

interfaces khác nhau. Giả dụ như ta có một object thuộc loại TheClass và nó cũng có implement một interface tên **MyInterface**, ta có thể dùng interface ấy trong code sau đây:

```
Dim obj As TheClass

obj = New TheClass

CType(obj, MyInterface).DoSomething()
```

Theo cách trên ta có thể gọi methods theo cách Early Bound trong nhiều interfaces của một object mà không cần phải declare một variable mới.

Thừa kế từ một ngôn ngữ khác

VB.NET code được compile ra IL (Intermediate Language) managed code, tức là code sẽ được CLR (Common Language Runtime) chạy trong .NET Framework. Mọi managed code, không cần biết được compiled từ ngôn ngữ nào đều có thể làm việc chung nhau, tức là ta có thể tạo một class trong ngôn ngữ này và dùng nó trong một ngôn ngữ khác, kể cả việc thừa kế.

Thật ra hầu như ta luôn luôn làm việc ấy khi viết VB.NET. Đó là vì phần lớn .NET system library được viết bằng C#, nhưng ta dùng hay thừa kế từ nó thường xuyên trong VB.NET.

Tạo một VB.NET BaseClass

Trong thí dụ về thừa kế từ một ngôn ngữ khác, trước hết ta thử tạo một Class Library Project trong VB.NET tên **vblib** và thêm vào đó một class đơn giản tên **Parent** giống như sau:

```
Public Class Parent

  Public Sub SayHello()

    MsgBox("Hello from Parent Class", MsgBoxStyle.Information, "Parent Class in VB.NET")

  End Sub

End Class
```

Ta sẽ dùng Parent làm BaseClass để thừa kế thành một SubClass trong C#.

Tạo một C# SubClass

Dùng **File | Add Project** để thêm một C# Class Library project mới và đặt tên nó là **cslib**. Reference vblib bằng cách dùng Menu command **Project | Add Reference...** và chọn Tab Projects, click Browse để tìm **vblib.dll** trong **vblib\bin** subfolder.

Lưu ý là ta vừa mới reference vblib.dll, cái assembly của Class Parent, chớ ta không đụng đến hay cần VB.NET source code của Class Parent. Trong C#, ta sẽ thừa kế Class Parent qua reference BaseClass trong vblib.dll assembly.

Bây giờ code C# như sau:

```
namespace cslib
{
 using System.Windows.Forms;
 using vblib;

 public class cSharpclass : Parent
 {
  public cSharpclass()
  {
    MessageBox.Show("Instantiating cSharpclass object, inheriting VB.NET Parent class", "CSharp Class");
  }
 }
}
```

Code C# bên trên có nhiều điểm tương đồng với VB.NET. Tuy nhiên vì C# đến từ ngôn ngữ lập trình C và C++ nên nó có syntax hơi khác một chút:

- Mọi statement trong C# phải chấm dứt bằng dấu **;** để đánh dấu cuối hàng
- Cặp dấu ngoặc cong queo **{ .. }** được dùng để đánh dấu đầu và cuối của một Statement Block thay vì dùng **End Sub**.

- Keyword **using** được dùng thay vì keyword **Imports** trong VB.NET
- C# thì **case sensitive**, tức là phân biệt chữ hoa, chữ thường - thí dụ **obj** thì khác với **Obj**.
- Constructor method mang cùng tên với class thay vì tên **New** như trong VB.NET.

Ta hãy thử đi qua các dòng code. Câu thứ nhất định nghĩa namespace cho source file. Trong C#, mọi namespace phải được tuyên bố rõ ràng (explicitly declared) trong mỗi code module.

```
namespace cslib
```

Kế đó là hai câu tuyên bố ta nhập khẩu System.Windows.Forms và vblib:

```
using System.Windows.Forms;

using vblib;
```

Câu kế đó tuyên bố cSharpclass thừa kế từ class Parent, để ý cách dùng dấu **:** thay vì keyword **Inherits**:

```
public class cSharpclass : Parent
```

Sau cùng là Constructor dùng chính tên của class:

```
  public cSharpclass()

  {

    MessageBox.Show("Instantiating cSharpclass object, inheriting VB.NET Parent class", "CSharp
Class");

  }
```

Để ý cách dùng **MessageBox.Show** giống hệt như trong VB.NET để hiển thị một message.

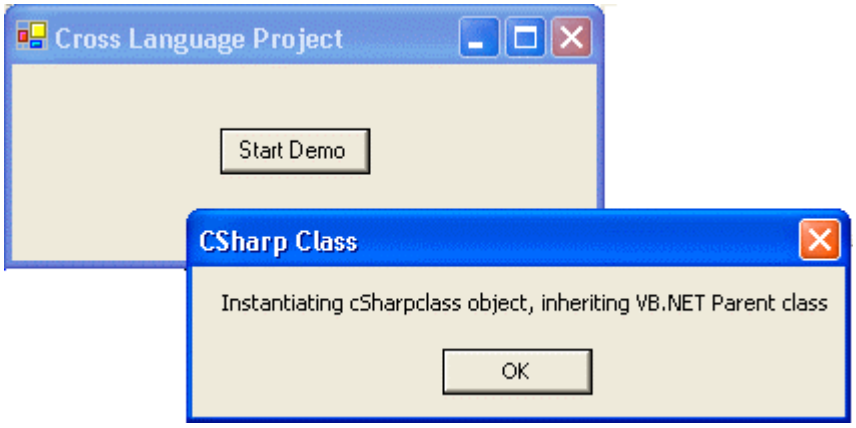
Tạo một program Client

Dùng menu command **File | Add Project** để thêm một VB.NET Windows Application project mới cho solution. Trong project mới này ta dùng menu command **Project | Add Reference...** để thêm references cho **cslib** và **vblib**. Right-click lên project trong Solution Explorer và chọn nó làm **Set As Startup Project** để project này chạy khi ta bấm *F5*.

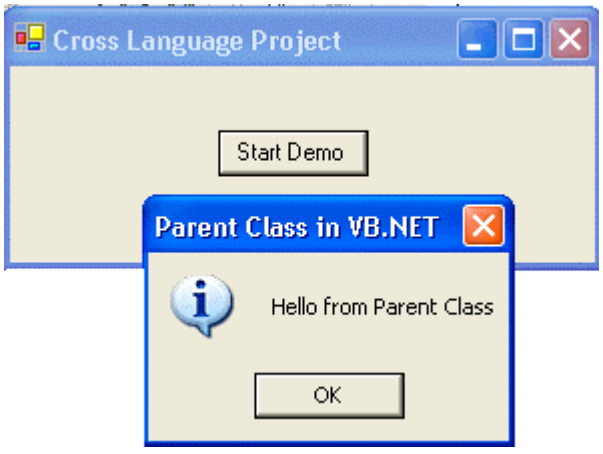
Bây giờ đặt một Button tên BtnStartDemo lên Form và viết code dưới đây để xử lý Event Click:

```
Private Sub BtnStartDemo_Click( ByVal sender As System.Object,  _
                                ByVal e As System.EventArgs) Handles BtnStartDemo.Click
  Dim objCS As New cslib.cSharpclass()
  objCS.SayHello()
End Sub
```

Khi ta chạy program và click button StartDemo ta sẽ thấy một dialog cho biết Constructor của cSharpclass đang được gọi để instantiate object **objCS**:



Tiếp theo đó một dialog thứ nhì hiển thị message từ Sub SayHello mà objCS thừa kế từ BaseClass Parent:



Thừa kế hình ảnh (Visual Inheritance)

Cho đến bây giờ ta đã bàn qua chức năng OO của ngôn ngữ VB.NET, phần lớn nhắm vào đặc tính thừa kế.

Vì các hình ảnh (Visual Components) trong VB.NET được implemented bằng ngôn ngữ lập trình chính quy chớ không phải dùng một cách thức khác biệt như trong VB6 (tin tức diễn tả các hình ảnh nằm ở phần đầu các ***.frm** files), nên VB.NET cũng hổ trợ **Thừa kế hình ảnh (Visual Inheritance)** cho Windows Forms một cách tự nhiên. Điều này có nghĩa là sau khi làm xong một Windows Form với những Textboxes, Labels, Listboxes ..v.v.. ta có thể thừa kế nó rồi để vô thêm các hình ảnh khác. Ta sẽ bàn vô chi tiết về chuyện này trong tương lai.

Ta cũng có thể thừa kế từ chính các hình ảnh. Thí dụ ta có thể thừa kế từ một Textbox để tạo ra một class Textbox mới, có thêm chức năng nhận keystrokes theo cách VNI và hiển thị chữ Việt Unicode.

Cùng một nguyên tắc thừa kế này của Windows Forms Controls cũng áp dụng cho **Web Forms Controls**, tức là ta có thể SubClass một Web Forms Control, cho thêm các chức năng mới và overriding một số chức năng có sẵn.

Bài 8
Những chức năng mới trong giao diện cửa sổ của VB.NET (phần I)

Xin nhắc lại là .NET Framework cho ta ba cách để user giao diện với chương trình áp dụng, đó là Windows Forms (có khi được gọi tắt là WinForms), Web Forms và Console applications. Lần lượt chúng ta sẽ học qua cả ba thứ này.

Ngoài ra, kể từ đầu tháng hai 2002, **thầy Vũ Năng Hiền** sẽ viết một loạt bài riêng về **ASP.NET**. ASP.NET là hậu thân của ASP (Active Server Pages), cái framework để ta lập trình trên Webserver. Microsoft dùng ASP để thay thế **cgi-Perl (Common Gateway Interface - Practical Extraction and Report Language)**, một ngôn ngữ lập trình rất thịnh hành trên các Unix-based Webserver. Sau này chính Microsoft mướn người ta port Perl qua WindowsNT.

Một chương trình ASP gồm có nhiều trang giống như trang Web (HTML) nhưng bên trong có những mảnh chương trình viết bằng VBScript hay JavaScript (thật ra script nào cũng được, kể cả PerlScript) nằm ở nhiều nơi. Các mảnh Script này có thể truy cập cơ sở dữ liệu để sửa đổi hay lấy dữ kiện ra để hiển thị tại chỗ ấy (nơi mảnh Script nằm trong trang ASP) để tạo trang Web kết quả.

Trong ASP.NET, tất cả các mảnh chương trình Script ấy được lấy ra riêng, để chung với nhau và được viết lại dưới dạng ngôn ngữ thuần túy VB.NET hay C#. Phần coding đó được gọi là **code behind** (code nằm phía sau) và rất giống như các Event Handling Sub ta viết trong VB.NET cho Windows Forms.

Trong tương lai, khi bàn đến Web Forms ta chỉ học tổng quát về ASP.NET và nhất là chỉ dùng VB.NET trong các chương trình đơn giản.

Sự quan trọng của Windows Forms ?

Windows Forms là cách hiển thị màn ảnh tối tân hơn Win32 bình thường. Kỹ thuật nằm phía sau Windows Forms trước đây được phát triển cho Windows Foundation Classes (WFC), để dùng trong Visual J++. Điều này cắt nghĩa sự già dặn và vững chải của một sản phẩm hãy còn ở tình trạng Beta.

Khi ta nghe nói đến .NET với những hứa hẹn về ứng dụng trên Internet như Web Forms và Web Services, rất dễ cho ta tưởng rằng Microsoft phải cung cấp Windows Forms là cực chẳng đã cho nó trọn vẹn món hàng. Thật ra, Windows Forms là một phần của các base classes của .NET Framework. Cái Namespace dùng cho nó là **System.Windows.Forms**, một Namespace chứa rất nhiều thứ đến đổi hầu như chúng ta sẽ không cần phải dùng trực tiếp các **Windows API về đồ hoạ (Graphics và Drawings)** như trong VB6 nữa.

Nhu cầu có những áp dụng phía khách (client-based application) phong phú (rich), linh động (flexible) và nhanh chóng (responsive) sẽ vẫn còn đó. Hiện nay, để tránh phí tổn về cài đặt (deployment) các chương trình, người ta bắt đầu có khuynh hướng đặt các chương trình chạy trên Webserver, rồi cho user sử dụng chúng qua WebBrowser. Ngoài công chúng thì dùng Internet, trong hãng xưởng thì dùng Intranet (Intranet là Internet chạy trên Local Area Network - mạng địa phương, không liên lạc gì với bên ngoài), tuy nhiên giao diện trên Web không phong phú hay nhanh như trên desktop và dĩ nhiên công tác lập trình đòi hỏi một thời gian phát triển lâu hơn.

Vì .NET Framework chứa đầy đủ mọi thư viện cần thiết cho chương trình, nên một khi đã cài đặt .NET Framework trên máy khách rồi ta chỉ cần **XCopy** đến đó những folders cần thiết có chứa các tệp (files) chương trình và dữ kiện là đủ. Thực hiện việc này trên mạng địa phương (Local Area Network) rất dễ và nhanh, thậm chí ta có thể tự động hóa công tác copy này.

Trong mô hình lập trình nhiều tầng (**multi-tier programming model**) mà ta gọi là **Windows DNA (Distributed Network Application)**, quá trình xử lý một công tác được chia ra làm nhiều giai đoạn như:

1. Kiểm chứng các con số user mới điền vào các forms tại máy khách (**user interface**)
2. Tính toán (**business logic**)
3. Truy cập cơ sở dữ liệu (**database access**)

Và mỗi giai đoạn nói trên có thể nằm trên một computer khác nhau. Nếu dùng Internet thì giai đoạn **1** nói trên sẽ chạy trong WebBrowser bằng trang Web có chứa JavaScript routines để kiểm chứng các con số user

mới đánh vào. Còn các giai đoạn kia có thể chạy trên WebServer. Dĩ nhiên giai đoạn **3** phải chạy trên WebServer, nơi chứa cơ sở dữ kiện.

.NET cho phép ta lập trình giai đoạn 1 để chạy trong Windows Forms. Còn các giai đoạn kia có thể để y nguyên.

Như thế, giả dụ như ta có một hệ thống đặt hàng, ta có thể cho các telephone operators dùng desktop (Winforms) application với một giao diện được tối ưu hóa, chạy thật nhanh để phục vụ những người đặt hàng bằng điện thoại. Trong khi đó khách hàng cũng có thể đặt hàng qua Internet WebBrowser như bình thường. Cả hai nhóm users này dù có giao diện khác nhau nhưng đều xài chung các tầng business logic và database access.

Đây là một ưu điểm rất quan trọng của .NET mà ít ai chú ý. Nếu thiết kế khéo, ta có thể lập trình để dùng chung hầu hết phần mềm trên desktop, distributed (phân tán), Internet và Mobile (Mobile phone, Pocket-PC).

Những điểm căn bản của Windows Forms ?

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

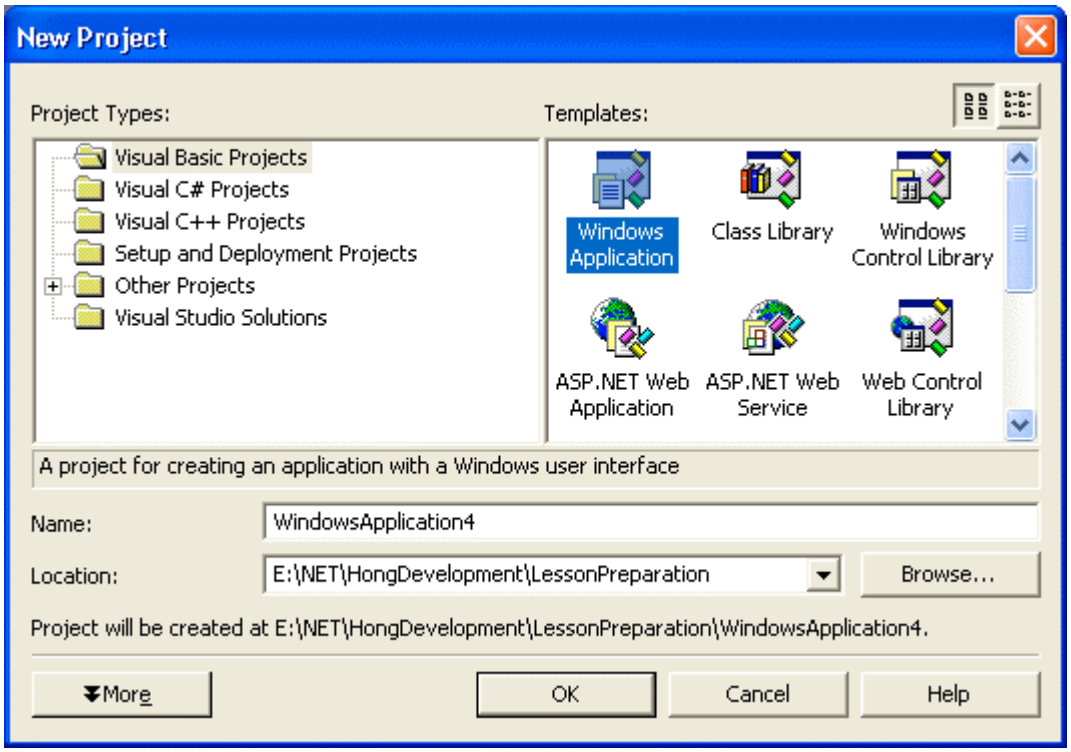
- Một Windows Form thật sự là một **class**. Trong .NET không có từ đặc biệt như "form module" để dùng cho nó.
- Vì một form là một class nên ta không thể load nó mà không nói thẳng thừng ra. Tức là trong VB6 nếu ta Show hay dùng đến một Form thì nó tự động được loaded. Chẳng những thế thôi, cái **class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

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

- Cái visual forms designer của VS.NET nhét rất nhiều code để instantiate form và đặt các controls vào form. Đó là code mà đáng lẽ ta phải tự viết nếu ta dùng notepad để lập trình. Phần code này thay thế cái phần nằm ở đầu tệp .frm của VB6 để diễn tả các visual components của form. Mỗi lần ta thêm bớt các controls hay thay thế các properties của controls trên form thì code generated cho form được thay đổi theo. Do đó bạn nên tránh sửa đổi code ấy, trừ khi biết chắc mình đang làm gì, hay là bạn làm một phiên bản trước khi thay đổi để nếu lỡ kẹt thì restore code cũ.

- Event được xử lý bằng cách linh động hơn. Các events chứa nhiều tin tức hơn. Một Event có thể được xử lý bởi nhiều controls cùng một lúc và mỗi control có một cách xử lý khác nhau. Ngược lại, nhiều Events khác nhau có thể được xử lý bằng một Event Handler duy nhất.

Bạn tạo một chương trình Windows Forms bằng cách dùng IDE menu command **File | New | Project..** để hiển thị giao thoại New Project và chọn **Template Windows Application**.

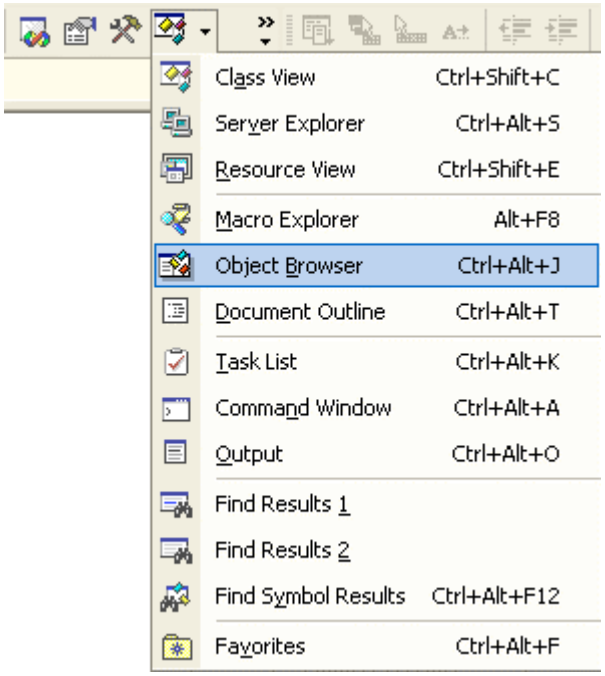


Trong thí dụ này, khi bạn click nút OK thì một subfolder tên (Name:) **WindowsApplication4** sẽ được tạo ra trong folder (Location:) **E:\NET\HongDevelopment\LessonPreparation** để chứa các tệp của Project. Sau này, khi bạn build, tức là compile chương trình, thì kết quả sẽ là một tệp **.exe** chứa trong folder

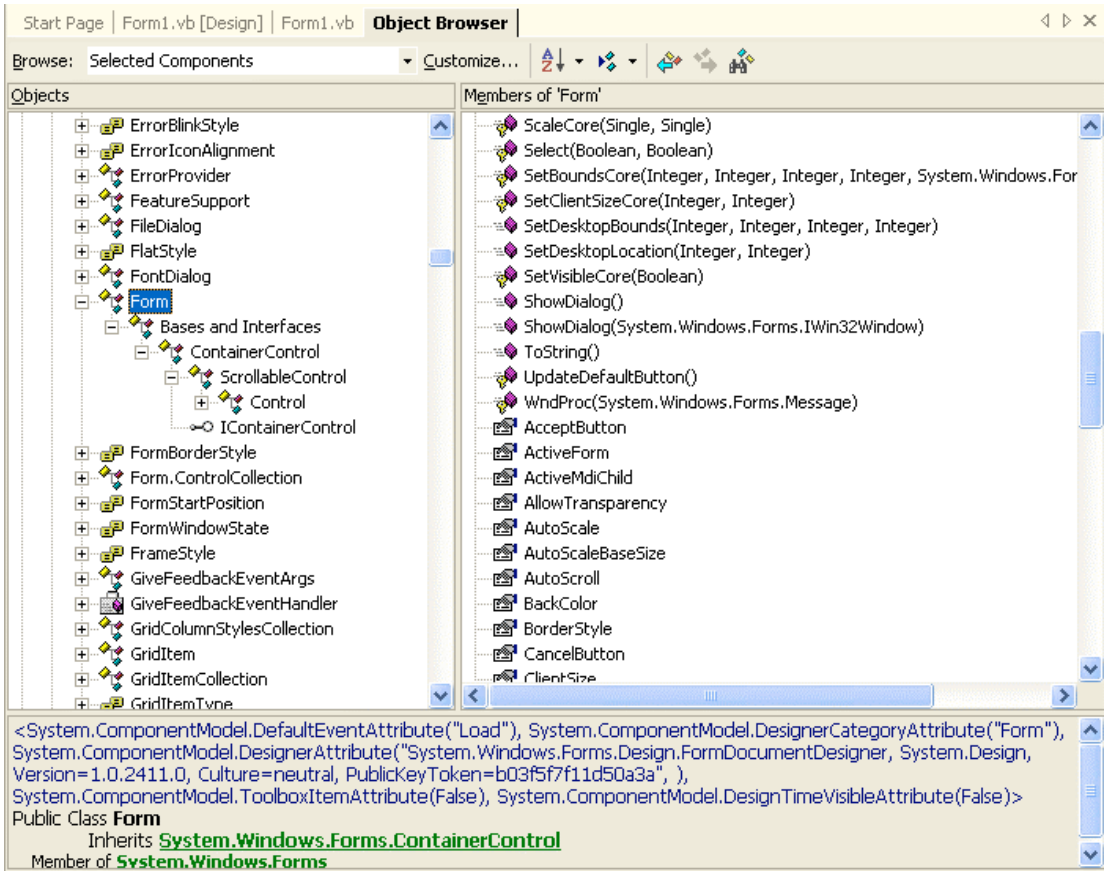
E:\NET\HongDevelopment\LessonPreparation\WindowsApplication 4\bin.

Dĩ nhiên trước khi click nút OK bạn có thể sửa **Name:** hay **Location:** tùy ý. Ngoài ra, vì bạn chọn Windows Application, nên project của bạn tự động có reference đến .NET component **System.Windows.Forms.dll**.

Để xem lướt qua namespace System.Windows.Forms, bạn hãy thử xúc tiến tạo cái project WindowApplication4 này. Kế đó bạn chạy **Object Browser** bằng cách click hình tam giác nhỏ của **Class View** icon rồi chọn Object Browser:



Trong Object Browser, expand cái System.Windows.Forms tree để xem những types được định nghĩa bên trong và các class members của Form:



Kiến trúc (Architecture) của Windows Forms ?

Nếu bạn xem gia phả của form, bạn sẽ thấy tổ phụ (đời thứ nhất) nó là **class Object**, còn form là con cháu đời thứ bảy. Dưới đây là cái cây của gia phả form và một ít chú thích:

Thứ bậc các classes	Chú thích
Object	Ông tổ trong .NET, superclass cao nhất từ đó sanh ra con cháu.
MarshalByRefObject	Cung cấp các code cần thiết để quản lý cuộc đời của objects.
Component	Cung cấp sự gầy dựng căn bản của IComponent interface và cho phép các chương trình khác nhau dùng chung một object
Control	Đây là base class của mọi component dùng để hiển thị. Nó hỗ trợ những khả năng liên hệ đến vóc dáng và công tác hiển thị từ Show, BringtoFront, Font, Color cho đến Dock, Anchor. Ngoài ra nó còn cung cấp các Events của keyboard, mouse và có **method WndProc** để cho ta truy cập các thông điệp của Windows.
ScrollableControl	Cung cấp chức năng tự động cuốn khi có chứa bên trong một control cần thêm chỗ để hiển thị.
ContainerControl	Cho phép một component chứa các controls khác.
Form	Cửa sổ chính của một chương trình.

Các chức năng mới của Windows Forms

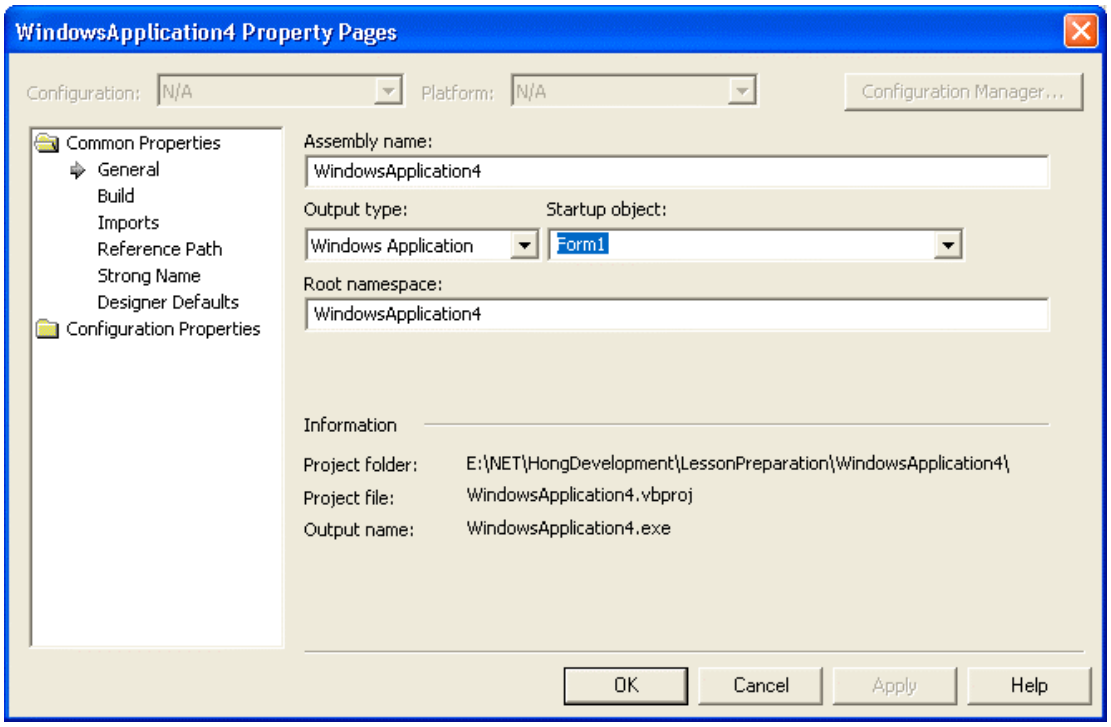
Những Controls tàn hình được chứa riêng

Một thay đổi rất tốt trong .NET từ VB6 là những controls không hiển thị lúc chạy thì khi thiết kế chúng được chứa trong một cái mâm riêng phía dưới. Thí dụ như trong hình dưới đây ta có Timer, Tooltip, Menus và các Dialogs được cho nằm trong một component Tray.

Muốn thay đổi properties của Control nào, ta chỉ cần chọn nó rồi right click và chọn **Properties**.

Chọn Startup Form

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



Vị trí ban đầu

Nhiều lúc ta muốn form hiện ra ngay giữa màn ảnh khi chương trình khởi động. VB.NET có thể làm việc ấy tự động nếu bạn set **property StartPosition** của nó thành **CenterScreen**. Các vị trí khởi đầu bạn có thể set được liệt kê dưới đây:

Trị số Vị trí khởi đầu	Kết quả
Manual	Hiển thị form ở vị trí theo giá trị của **property Location** của form
CenterScreen	Hiển thị form ở ngay giữa màn ảnh
CenterParent	Hiển thị form ở ngay giữa form chủ (owner) của nó
WindowsDefaultLocation	Hiển thị form ở vị trí default của cửa sổ
WindowsDefaultBounds	Hiển thị form ở vị trí default của cửa sổ, với kích thước default của cửa sổ

Borders của Form

Thay đổi **property FormBorderStyle** sẽ ảnh hưởng những gì user có thể thay đổi hay dùng về MaximizeBox, MinimizeBox, SizeGripStyle (mấy cái gạch chéo ở góc dưới phải của cửa sổ) và HelpButton.

Luôn luôn nằm trên hết

Một số chương trình có khả năng luôn luôn nằm trên hết, ngay cả khi nó không có focus. Để thực hiện điều này trong VB6 ta phải gọi API. Trong VB.NET, forms có một property mới tên là **TopMost**. Chỉ cần set TopMost của một form thành True thì nó luôn luôn nằm trên hết.

Owned Forms (Forms có chủ)

Khi một form có chủ, nó được minimized và closed theo form chủ của nó. Owned forms, đôi khi còn được gọi là forms nô lệ, luôn luôn nằm lên trên form chủ của nó. Dầu vậy, nó không cản trở form chủ nhận focus.

Ta dùng **method AddOwnedForm** của form chủ để cho thêm owned form vào collection of OwnedForms của nó như sau:

```
Private Sub Form1_Load( ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
MyBase.Load

  Dim myForm2 As New Form2()

  myForm2.Show()

  Me.AddOwnedForm(myForm2)

End Sub
```

Form chủ có thể truy cập collection của các forms nô lệ qua property OwnedForms. Dưới đây là code để loop qua các forms nô lệ của một form:

```
Private Sub BtnListOwnedForms_Click( ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles BtnListOwnedForms.Click

  Dim OwnedForm As Form

  For Each OwnedForm In Me.OwnedForms

    Console.Write(OwnedForm.Text)

  Next

End Sub
```

Form chủ có thể cắt bỏ (remove) một form nô lệ bằng cách dùng **method RemoveOwnedForm** như:

```
Me.RemoveOwnedForm(myForm2)
```

Khi một form không còn là nô lệ nữa, nó không hẳn bị unloaded, chỉ trở thành một form tự do (không còn liên hệ với form chủ nữa) thôi.

Chú ý sự khác biệt giữa form nô lệ và TopMost form là form nô lệ chỉ nằm trên form chủ nó, trong khi TopMost form nằm trên tất cả mọi forms khác. TopMost form cũng không bị minimized hay closed khi một form nào khác của chương trình bị minimized hay closed.

Không phải mọi controls đều bị khoá (locked)

Trong Vb6, ta có option **Lock Controls** trong Format menu. Khi ta chọn Option này cho một form, tất cả controls đều bị khóa, ngay cả những controls mới được để vào mặt form sau này.

Trong VB.NET, ta cũng có option Lock Control trong Format menu hay khi ta right click một nhóm controls đã được chọn trên form. Nhưng thao tác khóa này chỉ hiệu lực đối với các controls có sẵn trên form mà thôi. Một control mới được đặt lên form sau đó sẽ không bị khóa. Điều này cho phép ta khóa những controls đã được để đúng vị trí, rồi tiếp tục sắp đặt các controls mới mà không ngại vô tình làm di chuyển vị trí các controls cũ.

Độ đậm (Opacity) của Form

Có một property mới của form rất thú vị để dùng, dù rằng sự ích lợi hay mục đích của áp dụng không rõ ràng. Đó là ta có thể thay đổi độ đậm của một form. Ta có thể làm cho nó trong suốt khi set **property Opacity** của form bằng 0, hay cho nó mờ mờ như ma nếu trị số của Opacity ít hơn 1. Bạn hãy thử đánh code dưới đây vào một form cho Button1 chẳng hạn, rồi chạy chương trình và click Button1 ấy:

```
Private Sub Button1_Click( ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
Button1.Click

  Dim i As Double

  For i = 0 To 1 Step 0.01

  ' Opacity có trị số từ 0 (trong suốt)  đến 1 (đậm đặt)
```

```
    Me.Opacity = i
  Next
End Sub
```

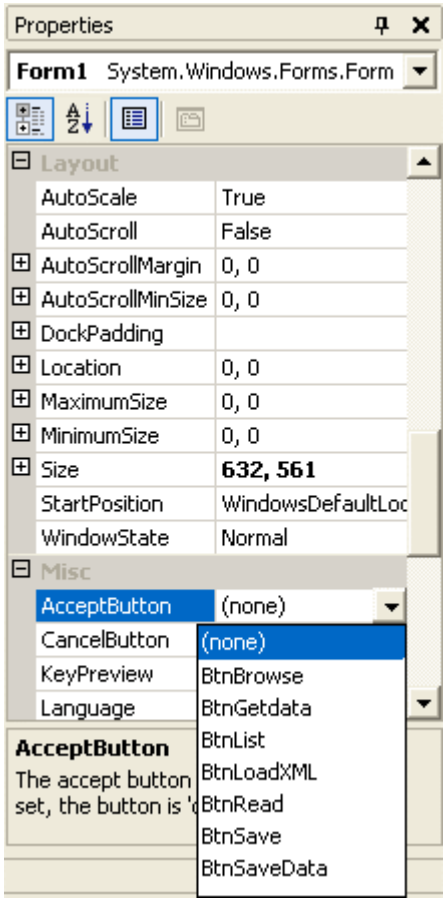
Form properties cho Cancel Button và Default Button

Trong VB6, ta có thể set một button để nó như được clicked khi thật ra user bấm phím **Esc**. Ta thực hiện điều này bằng cách set **property Cancel** của button ấy thành True. Nó được gọi là **Cancel button**.

Tương tự như thế, nếu ta set **property Default** của một button thành True, nó được gọi là **Default button**, khi user bấm phím **Enter** Default button coi như được clicked.

Trong VB.NET ta cũng có thể dùng các chức năng ấy, nhưng bây giờ ta không đá động gì đến property nào của các buttons, mà lại set các **properties CancelButton** và **AcceptButton** của chính form.

Khi ta click bên phải của property AcceptButton trong cửa sổ Properties thì danh sách các buttons có sẵn trên form được liệt kê ra để ta chọn như dưới đây:



Ngoài ra ta cũng có thể chọn các CancelButton và AcceptButton lúc đang chạy chương trình, nhất là khi ta muốn bổ nhiệm các công tác này cho những buttons khác vì form đang làm việc trong một trạng thái khác như trong code thí dụ dưới đây:

```
Me.CancelButton = BtnCancel2

Me.AcceptButton = BtnAccept2
```

Bài 9
Những chức năng mới trong giao diện cửa sổ của VB.NET (phần II)

Sự khác biệt trong các Hộp Giao Thoại (Dialog Boxes)

Trong VB6, các hộp giao thoại thật ra là những form bình thường nhưng được hiển thị với parameter vbModal, tức là trong Modal mode. Điều này khiến cho hộp giao thoại trở nên form tích cực (active form) duy nhất trong chương trình cho đến khi nó đi khuất.

Một hộp giao thoại cần một phương tiện để liên lạc với form gọi nó (calling form). Trong VB6, ta giải quyết vấn đề này bằng cách chế ra một property tạm gọi là **Action**. Ta dùng **Read-only property Action** như sau trong một hộp giao thoại có hai buttons, **OK** và **Cancel**:

```
' VB6 code used for Dialog Boxes

Public Enum dialogAction

  actionOK = 1

  actionCancel = 2

End Enum


Dim mAction As dialogAction


Public Property Get Action() As dialogAction

  Action = mAction

End Property


Private Sub cmdOK_Click()

  ' Get here when user click the OK button

  mAction = actionOK

  ' Hide the Dialog Box to return control to calling form

  Me.Hide

End Sub
```

```
Private Sub cmdCancel_Click()
  ' Get here when user click the Cancel button
  mAction = actionCancel
  ' Hide the Dialog Box to return control to calling form
  Me.Hide
End Sub
```

Chú ý ta dùng Enumerated type **dialogAction**. Nó có hai trị số: **actionOK** và **actionCancel**. Property Action thuộc loại enumerated type này. Khi user click một button, ta set trị số cho local variable **mAction** rồi **Hide** cái dialog box. Cái Giao thoại phải được dấu đi (hidden) nhưng không unloaded, vì cái calling form còn phải truy cập dialog box để đọc trị số của property Action để biết user vừa mới click button nào.

Giả dụ ta đặt tên cho hộp giao thoại đó là **frmDialog**. Để gọi một hộp giao thoại từ một form khác trong VB6 ta có thể code như sau:

```
Dim Dialog As frmDialog
Set Dialog = New frmDialog  ' Instantiate a Dialog Box
' Show dialog box in Modal mode
Dialog.Show vbModal
```

Nhưng bao nhiêu đó chỉ là hiển thị hộp giao thoại thôi. Sau khi hộp giao thoại đã Hide rồi ta còn phải truy cập nó để đọc trị số của property Action. Do đó ta cần phải viết thêm codes cho đầy đủ sau đây:

```
Dim Dialog As frmDialog
Set Dialog = New frmDialog  ' Instantiate a Dialog Box
' Show dialog box in Modal mode
Dialog.Show vbModal
' Get here after the dialog box has hidden, but still loaded
' Now process the Action
Select Case Dialog.Action
Case actionOK
```

```
  ' code goes here for normal processing
Case actionCancel
  ' code goes here for user canceling
End Select
Unload Dialog  ' Now we can unload the dialog box
```

Có hai sự thay đổi quan trọng trong VB.NET, đó là dùng **ShowDialog** và **DialogResult**.

ShowDialog thay vì Show vbModal

Argument vbModal không được hỗ trợ trong VB.NET. Thay vào đó, một form có thể dùng method **ShowDialog**. Dưới đây là sự so sánh của coding trong VB6 và VB.NET.

VB6 code:

```
Dim Dialog As frmDialog
Set Dialog = New frmDialog  ' Instantiate a Dialog Box
' Show dialog box in Modal mode
Dialog.Show vbModal
```

VB.NET code:

```
Dim Dialog As New frmDialog()
' Show dialog box in Modal mode
Dialog.ShowDialog
```

Để ý là trong VB.NET ở hàng code đầu ta có thể kết hợp hai chuyện khai báo và instantiate form mới trong một statement. Hàng code cuối cho thấy sự thay đổi từ Show vbModal qua ShowDialog.

DialogResult

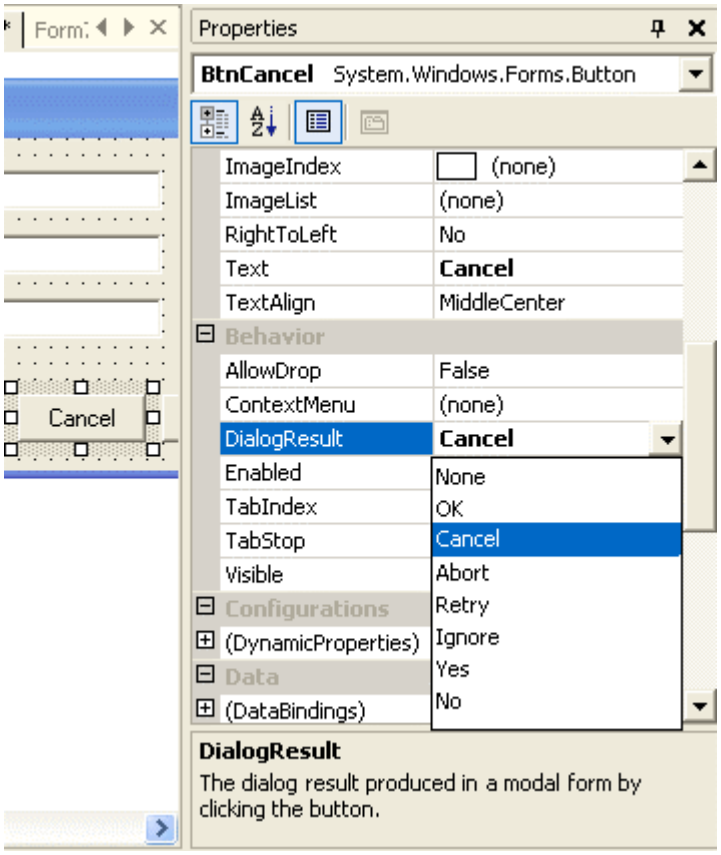
Trong VB.NET, khi một form khải thị bằng method ShowDialog, nó đã dự bị sẵn một property tên là **DialogResult** để calling form có thể truy cập.
DialogResult có thể mang một trong những trị số enumerated sau đây:

- DialogResult.Abort

- DialogResult.Cancel
- DialogResult.Ignore
- DialogResult.No
- DialogResult.None
- DialogResult.OK
- DialogResult.Retry
- DialogResult.Yes

Có điểm rất tiện là khi DialogResult được set cho một trị số thì dialog được dấu đi (hidden) một cách tự động.

Cách đơn giản nhất để set trị số cho DialogResult là assign một trị số cho **property DialogResult** của một **button**. Khi user click button ấy thì DialogResult của hộp giao thoại lấy trị số của property DialogResult của button và hộp giao thoại Hide.



Để biểu diễn ShowDialog trong VB.NET, kèm theo đây là mã nguồn của một thí dụ. Trong thí dụ này ta tạo một form tên frmDialog có hai button tên OK và Cancel. Ta set property DialogResult của button OK thành **OK**

và property DialogResult của button Cancel thành **Cancel**. Form frmDialog hoàn toàn không có một hàng code nào cả.

Form chính của chương trình, Form1, chỉ có một button tên **BtnShowDialog** với code cho Event Click như dưới đây:

```
Private Sub BtnShowDialog_Click( ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles BtnShowDialog.Click

 ' Declare and instantiate a Dialog Box

  Dim Dialog As New frmDialog()

 ' Show the Dialog Box in Modal mode

  Dialog.ShowDialog()

 ' get here after user has clicked a button and the Dialog box has hidden

 ' Process the DialogResult

  Select Case Dialog.DialogResult

  Case DialogResult.OK

    MsgBox("User clicked OK, se please go ahead")

  Case DialogResult.Cancel

    MsgBox("Sorry, but User clicked Cancel")

  End Select

  Dialog = Nothing ' Dispose the Dialog Box

End Sub
```

Bạn có thể chạy chương trình rồi click button ShowDialog. Khi Dialog box hiển thị, thử click một trong hai buttons trên ấy.

So sánh với VB6, ta thấy dùng Dialog Box trong VB.NET đơn giản và tự nhiên hơn.

Nếu không dùng Property DialogResult của một button trong Dialog Box để trả về kết quả DialogResult, ta cũng có thể dùng code trong Dialog form như sau:

```
Me.DialogResult = DialogResult.Retry
```

Hàng code trên set DialogResult của Dialog form thành DialogResult.Retry và kèm theo phản ứng phụ là Hide Dialog Box. Calling form sẽ truy cập được kết quả DialogResult.Retry này.

Sự khác biệt về sắp đặt vị trí cho Forms và Controls

VB.NET có những chức năng về positioning và layout tương tợ như trong VB6, nhưng cách thực thi hơi khác.

Property Location

Thay vào các properties **Left** và **Top** trong VB6, forms và controls trong VB.NET có property **Location**. Property Location nhận và trả về một structure tên **Point**, có tọa độ **X** và **Y** tương ứng với Left và Top mà ta dùng trước đây.

Structure point được dùng trong nhiều áp dụng về đồ họa trong **GDI+ (Graphic Devices Interface plus)** của .NET.

Thật ra, trong code ta vẫn còn dùng Top và Left được như xưa. Nhưng Top và Left không hiện ra trong cửa sổ Properties của forms hay controls. Để chỉ định một vị trí mới cho form, ta có thể code như sau:

```
Me.Location = (New Point(200, 100))
```

Property Size

Property Size trong VB.NET có cùng một ý niệm như property **Location**, có điều nó tương xứng với Width và Height. Property Size nhận và trả về một structure tên **Size**, có chiều cao và chiều rộng để áp dụng cùng một lúc thay vì tuần tự từng chiều. Giống như Left và Top, trong code ta vẫn còn dùng Width và Height được như xưa. Nhưng Width và Height không hiện ra trong cửa sổ Properties của forms hay controls. Để thay đổi Size của một form, ta có thể code như sau:

```
Me.Size = (New Size(300, 400))
```

ReSize nhiều controls

VS.NET cho ta một chức năng mới là Resize nhiều controls cùng một lúc. Trước hết bạn Select nhiều controls bằng cách drag một dây thun (rubber band) bao quanh chúng hay ấn nút **Ctrl** trong lúc click các

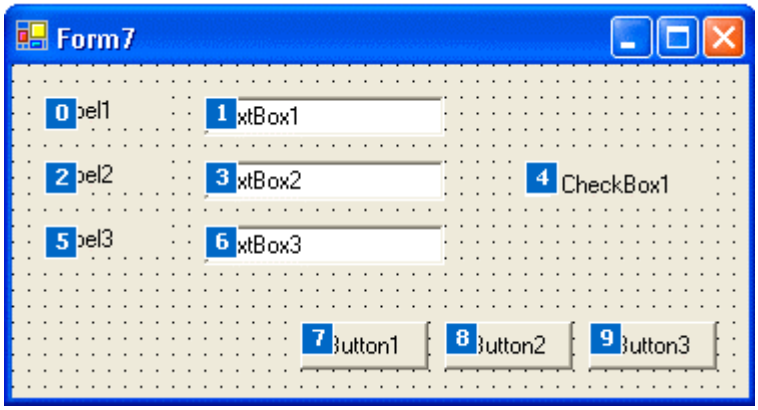
controls. Kế đó, tập trung việc resize vào một control, các controls kia cũng được resized theo.

Các features thiết kế chung cho các Controls

Tab Order của các Controls

Sắp đặt thứ tự trong Tab của các controls (**Tab Order**) trên một form đôi khi rất phiền phức trong VB6. VS.NET cho ta một feature rất tiện dụng để làm việc này. Để khởi động feature ấy, ta dùng IDE menu command **View | Tab Order**. Nó sẽ hiển thị một con số nhỏ ở góc trên trái của mỗi control, cho thấy trị số **Tab Index** của mỗi control. Bây giờ ta chỉ cần click lên từng control một theo thứ tự mà ta muốn.

Dưới đây là screenshot của một form sau khi user chỉ định Tab Order cho các controls. Muốn ra khỏi Tab Order mode, ta bấm menu command **View | Tab Order** một lần nữa.



Ghi chú: Trong VB.NET nhiều controls có thể có cùng một Tab Index. Trong trường hợp ấy, thứ tự về Tab của chúng được quyết định dựa vào **z-order**. Control có z-order cao nhất sẽ nhận focus trước nhất trong nhóm. Z-order của một control có thể được thay đổi bằng cách right click control rồi chọn **Bring to Front**.

Control Arrays

Khi nghe nói VB.NET không hỗ trợ Control Arrays chắc bạn buồn năm phút. Có hai lý do tại sao bạn cần Control Arrays:

1. Dùng cùng một Event handler (thí dụ như Sub BtnBrowse_Click) để xử lý Event từ nhiều Controls tương tự.

2. Để dynamically tạo thêm Controls trong form lúc đang chạy program (at runtime).

May thay, VB.NET cung cấp cho ta một phương tiện khác để khỏi phải thua thiệt. VB.NET cho phép ta linh động bổ nhiệm các methods để xử lý Events của các controls. Điểm thứ nhất bạn sẽ chú ý là bạn không thể dùng cùng một tên cho nhiều controls nữa. **Property Index** đã bị khai tử.

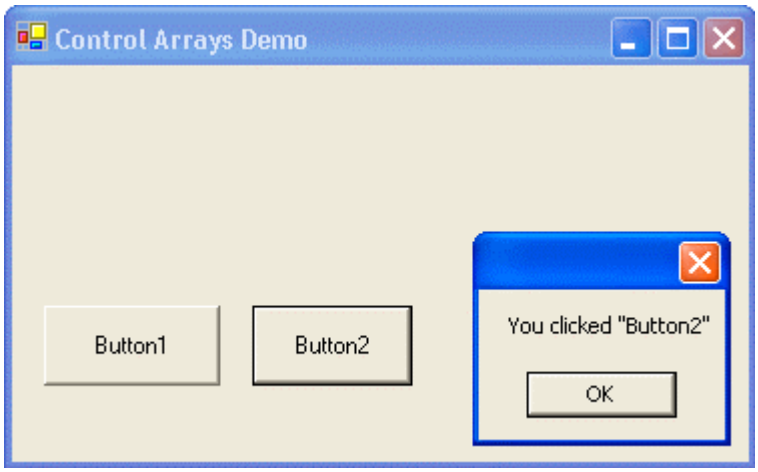
Trong VB.NET bạn có thể dùng một Event handler duy nhất để xử lý Events đến từ các controls tương tự. Trước đây ta dựa vào **Index** để biết Event phát xuất từ control nào. Bây giờ bạn dựa vào **parameter Sender**.

Để minh họa điểm này, ta sẽ viết một chương trình có hai buttons, Button1 và Button2, nằm trên form chính. Double click Button1 để viết code xử lý **Event Button1.Click**. Muốn dùng cùng một Event Sub này để xử lý luôn Event Click đến từ Button2, bạn chỉ cần thêm chữ **Button2.Click** vào cuối cái **Handles List** của **Sub Button1_Click**. Để cho có vẻ tổng quát ta rename Sub Button1_Click thành **Sub Button_Click**.

Bây giờ ta viết vài dòng code đơn giản để hiển thị cho biết Event Click đến từ Button nào:

```
' Note that we change the name of the Sub from Button1_Click to Button_Click to

' make it more general, since we're going to use this same Sub to handle Click

' Events originated from many different Buttons

' Also note that we add the word Button2.Click to the end of Sub Button_Click declaration

Private Sub Button_Click( ByVal sender As System.Object, ByVal e As System.EventArgs)  _

                Handles Button1.Click, Button2.Click

  Dim btnClicked As Button

  ' Type cast sender to Button

  btnClicked =  CType(sender, Button)

  ' Show what button was clicked

  MessageBox.Show("You clicked """ & btnClicked.Text & """")

End Sub
```

Thử chạy chương trình và click Button2, bạn sẽ thấy hình dưới đây:



Để biểu diễn chức năng quản lý Event Handling at runtime, ta sẽ đặt một button tên **BtnAddNewButton** vào form để nó dynamically add một button thứ ba tên **Button3**. Ta muốn button này cũng sẽ dùng Sub Button_Click để xử lý Event Click của nó. Vì không thể đánh thêm chữ Button3.Click vào cuối câu Sub Button_Click như trước đây ta đã làm với Button2.Click, nên at runtime ta sẽ dùng statement:

```
' Tell system to use Button_Click as Event Handler for the Event Button3.Click

AddHandler newButton.Click, AddressOf Me.Button_Click
```

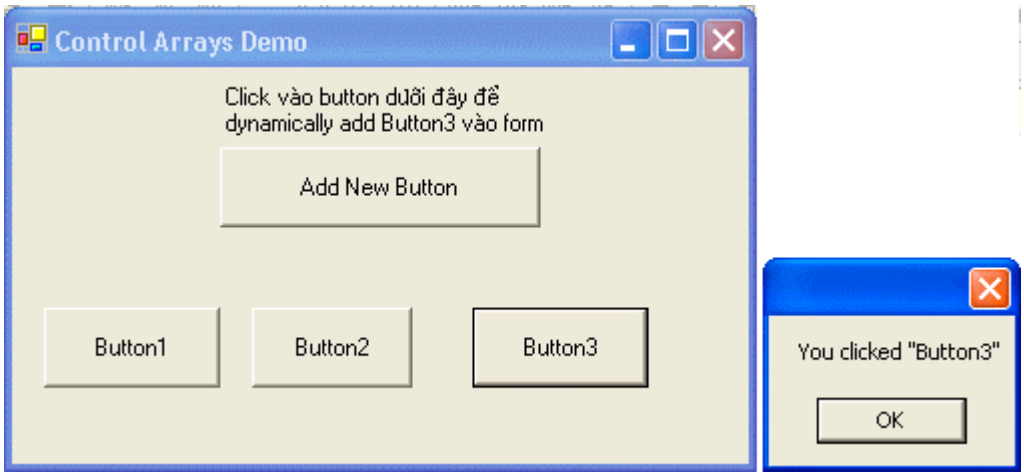
Mã nguồn đầy đủ của **Sub BtnAddNewButton_Click** được liệt kê dưới đây:

```
Private Sub BtnAddNewButton_Click( ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles BtnAddNewButton.Click

 ' Declare and instantiate a Button

  Dim newButton As New Button()

 ' Set it up on the form

  With newButton

    .Text = "Button3" ' Text of this new button

    .Location = New Point(230, 120) ' define its location on the form

    .Size = New Size(88, 40) ' define its size

  End With

 ' Add the new button to the form's collection of controls

  Me.Controls.Add(newButton)
```

```
' Tell system to use Button_Click as Event Handler for the Event Button3.Click

  AddHandler newButton.Click, AddressOf Me.Button_Click

End Sub
```

Khi user click BtnAddNewButton, Button3 với Size(88,40) sẽ được tạo ra và đặt ở Location(230,120) trên form. Kế nó chương trình thực hiện hai chuyện quan trọng: Add button mới này vào collection of controls của form và đăng ký (register) việc dùng Sub Button_Click làm Event Handler của Event Click của nó.

Làm xong mấy chuyện này rồi, bạn chạy chương trình, click AddNewButton để thêm Button3 vào form, kế đó click Button3, bạn sẽ thấy hình dưới đây:



Bài 10
Những chức năng mới trong giao diện cửa sổ của VB.NET (phần III)

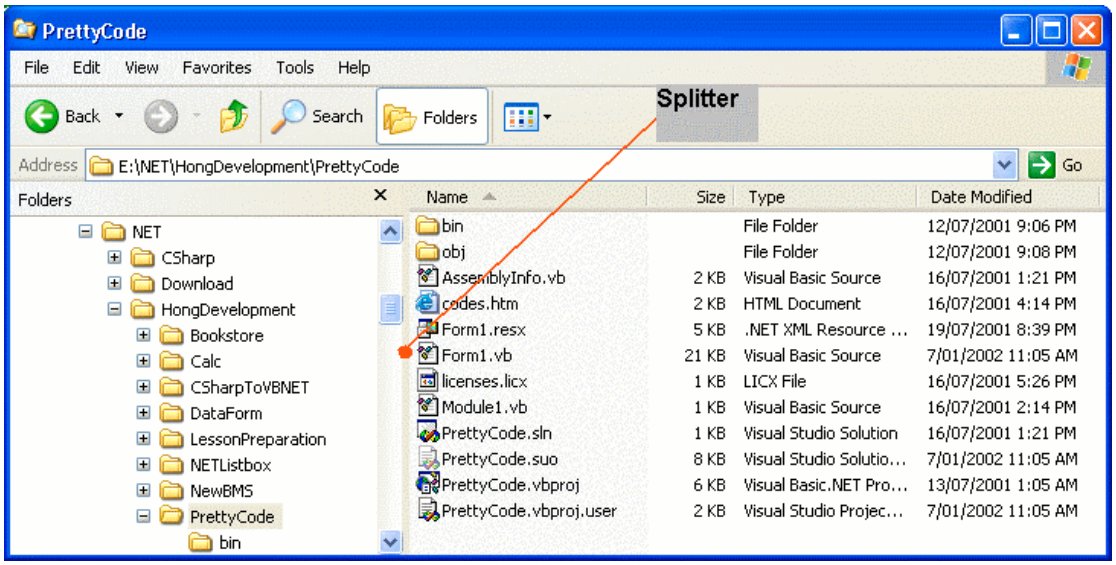
Tự động Resize và định chỗ (positioning)

Những chương trình ứng dụng chuyên nghiệp ta mua ngoài chợ để dùng thường thường có đặc tính resize các controls hay định vị trí của các controls trên form một cách tự động. Nếu bao giờ bạn đã thử thêm các chức năng ấy cho một chương trình áp dụng viết bằng VB6 của mình, bạn sẽ thông cảm rằng coi vậy chớ đó không phải là chuyện nhỏ.

Tưởng tượng là ta phải ghi nhớ vị trí và kích thước của mỗi control trên form để mỗi lần user resizes form thì ta phải theo đó resize và định vị trí của control. Trong lúc thiết kế ta phải cho user một phương tiện để chỉ định rằng họ muốn một control cư xử như thế nào khi form resize. Để chứa tin tức ấy hoặc ta dùng property Tag của control hoặc ta dùng registry. Chỉ việc đọc ra, viết vào để cập nhật hoá các tin tức cũng đủ mệt, chưa nói đến chuyện tính toán để resize và định vị trí của control. Do đó, nhiều khi làm biếng ta dùng đại một third party ActiveX để giúp ta làm các chuyện ấy.

.NET cho ta thêm các **properties Anchor và Dock** cho mỗi control. Ngoài ra .NET còn cung cấp control **Splitter** để cho phép ta nắm một thanh phân hai kéo qua, kéo lại hay kéo lên, kéo xuống tùy thích, để mở rộng thêm một bên trong khi bên kia bị thu hẹp.

Cái áp dụng của Splitter thông dụng nhất là trong Windows Explorer. Trong đó ta có hai phần: bên trái là một Treeview chứa cái cây của disk drives và file folders, bên phải là một Listview chứa icons hay chi tiết của các folder và files. Muốn xem Treeview nhiều hơn, ta nắm thanh phân hai ở giữa kéo qua bên phải một chút.

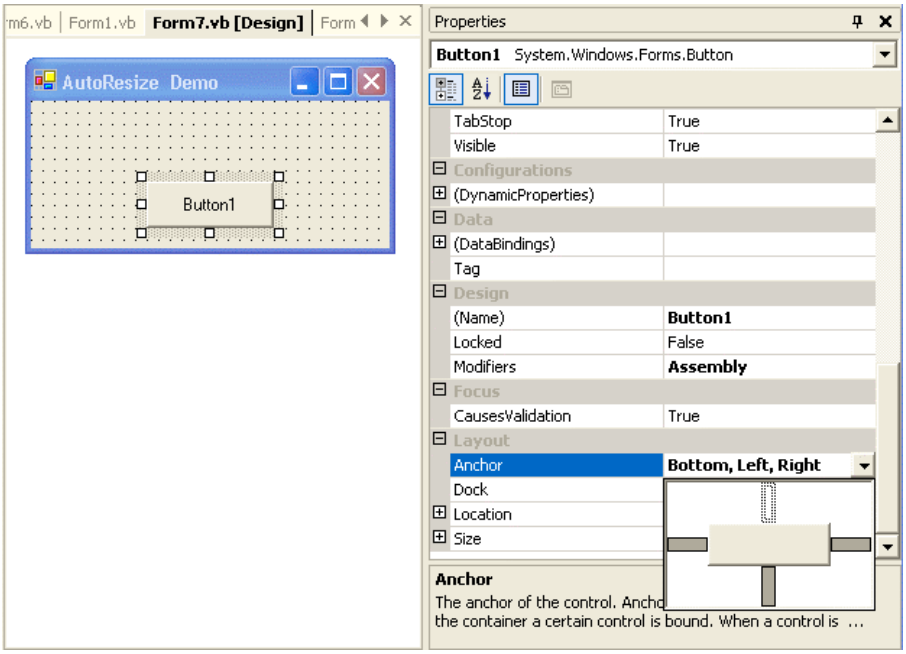


Anchoring (bỏ neo)

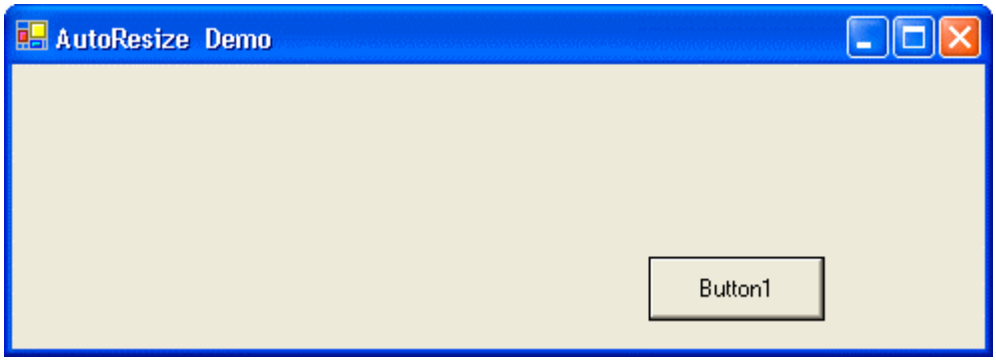
Khi con tàu bỏ neo là nó đỗ ở đó. Dù con nước chảy thế nào, con tàu vẫn nằm yên một chỗ vì nó đã được cột vào cái neo. Control trong .NET có property **Anchor** để ta chỉ định nó được buộc vào góc nào của form: **Left, Right, Bottom** hay **Top**.

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

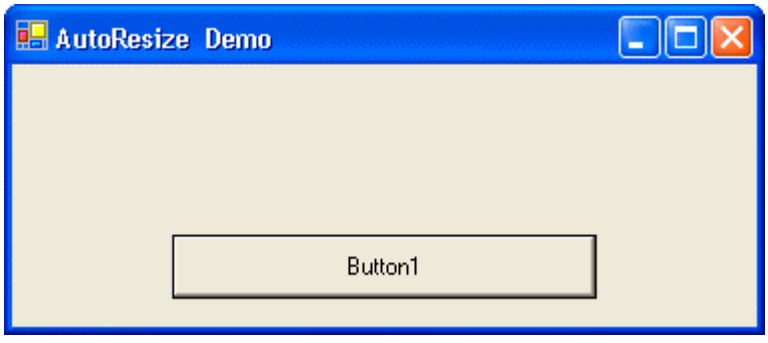
Thí dụ ta click vào thanh dưới và hai thanh hai bên, ta sẽ có **Bottom, Left, Right** như trong hình dưới đây:



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

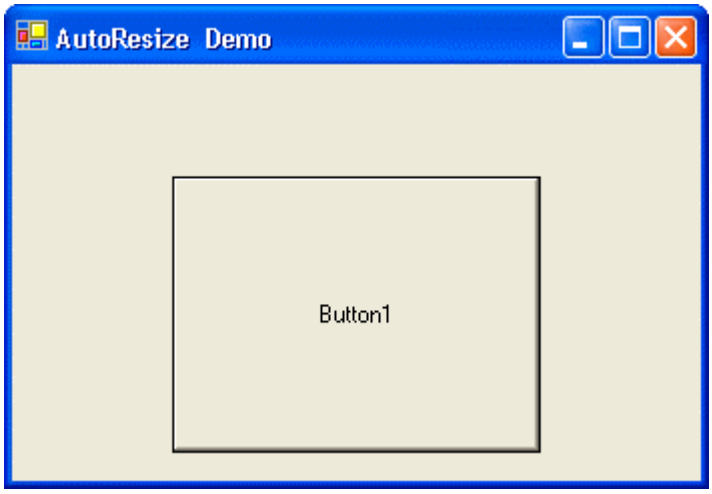


Nếu Button1 có Anchor là Left, Right, Bottom thì khi form resizes cho lớn ra, Button1 cứ giữ khoảng cách từ nó đến ba cạnh Left, Right, Bottom của form không đổi. Do đó nó phải nở rộng ra như trong hình dưới đây:



Nếu Button1 có Anchor là Top,Bottom,Left, Right thì khi form resizes, Button1 cứ giữ khoảng cách từ nó đến bốn cạnh Left, Right, Top, Bottom

của form không đổi. Do đó nó phải nở rộng hay thu nhỏ cả chiều cao lẫn chiều rộng như trong hình dưới đây:



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

Docking (gắn vào)

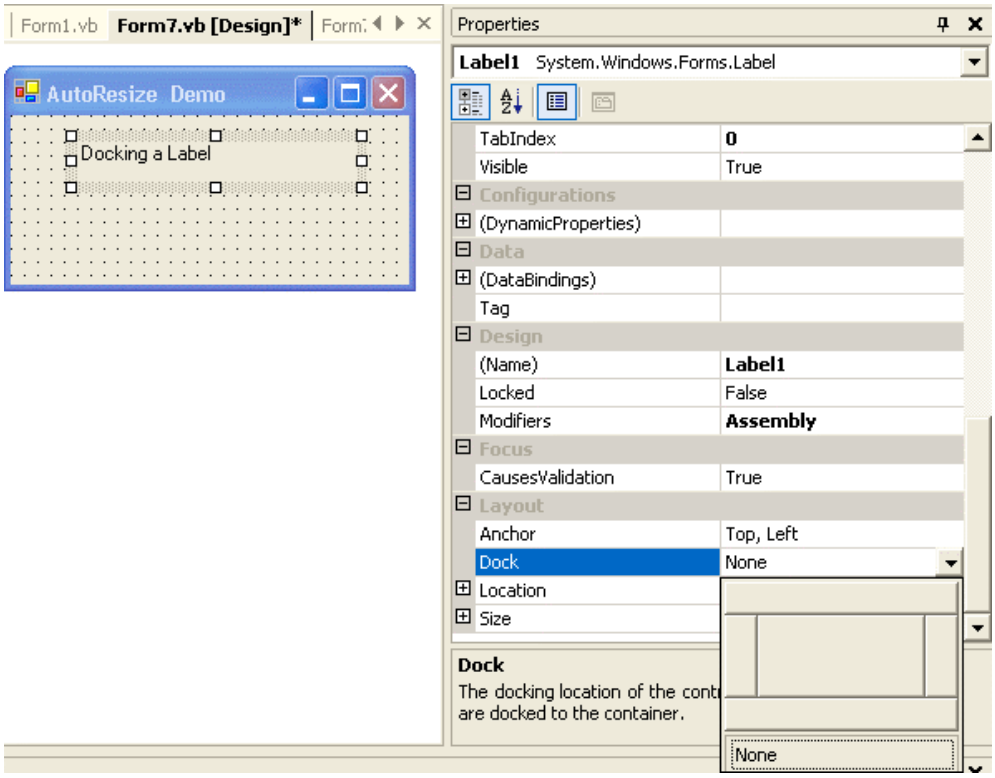
Khi ta **Dock** một control vào một cạnh của form có nghĩa là ta dán dính nó vào cạnh đó. Áp dụng ta thường thấy nhất của Docking là ToolBar và StatusBar. ToolBar thì dock vào phía trên của form, còm StatusBar thì dock vào phía dưới của một form. Chúng dãn ra chiếm từ trái qua phải của form, user không thể chỉ định chiều rộng của chúng. Khi form được resized thì ToolBar và StatusBar cũng dãn ra hay co vào theo chiều rộng của form.

Property Dock của control trong .NET cũng giống giống như **property Align** của control trong VB6 StatusBar.

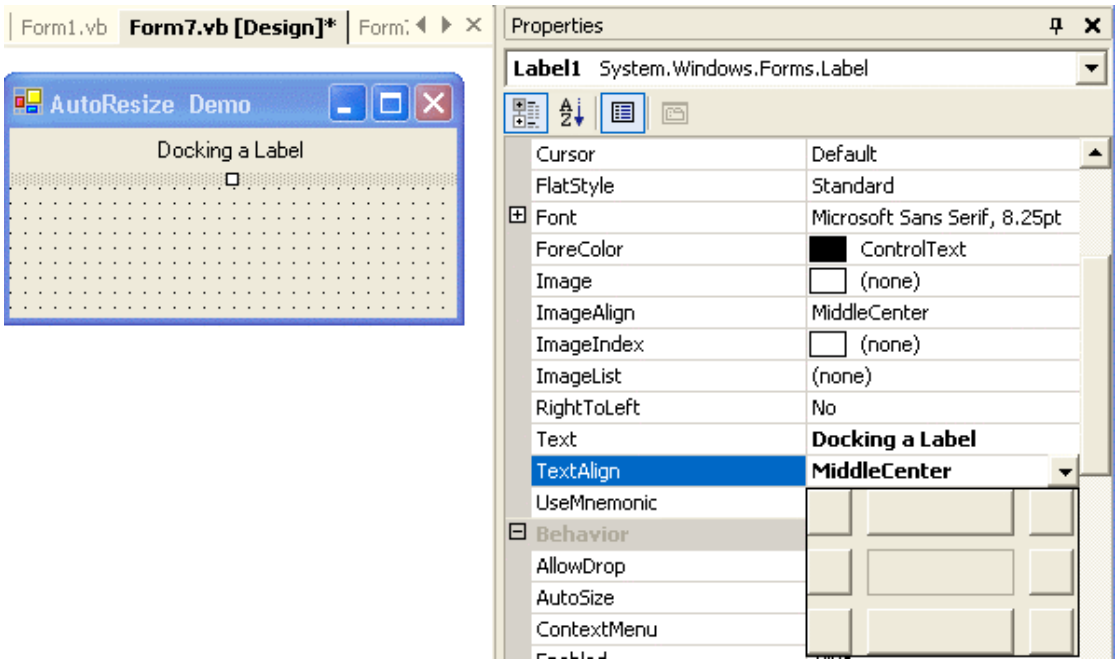
Ta chỉ có thể dán một control vào một trong bốn cạnh của form, chớ không có chuyện bắt cá hai, ba tay như trường hợp Anchor có thể neo vào Left, Right, Bottom cùng một lúc. Tuy nhiên, property Dock có trị số **Fill** để nói control chiếm hết bên trong phần còn lại của container của nó.

Trong lúc thiết kế, sau khi select cái control (thí dụ Label1), ta vào cửa sỗ Properties và click hình tam giác nhỏ bên phải property Dock. Một hình

vuông nhiều thanh màu xám sẽ hiện ra. Mỗi thanh tượng trưng cho một cạnh mà ta có thể chỉ định để dán control vào form (**Top, Bottom, Left** hay **Right**), cái hình vuông ở giữa tượng trưng cho trị số **Fill**, và thanh dưới chót có chữ **None** cho phép ta xóa không chọn trị số Dock nào cả.. Khi ta click một thanh, trị số Docking tương ứng sẽ hiển thị trong textbox area của combobox Dock.



Giả sử ta set **Property TextAlign** của Label là **MiddleCenter** bằng cách chọn cái thanh xám nằm ngay giữa trong số 9 thanh tượng trưng cho các vị trí của Text có thể nằm trong Label1 như trong hình dưới đây:

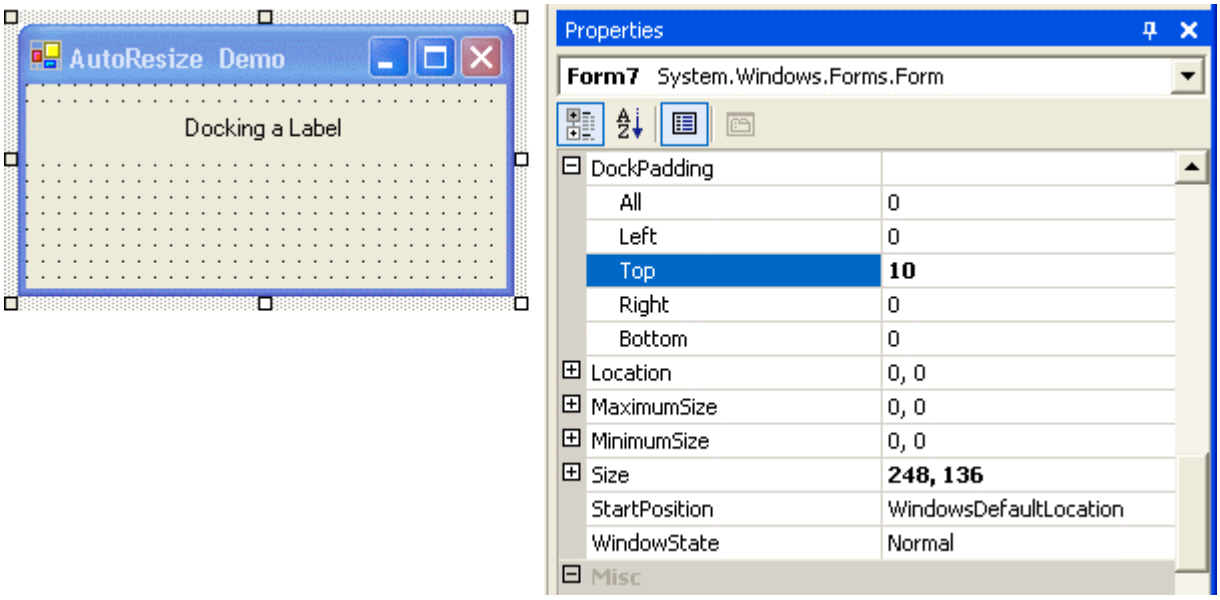


Khi chạy chương trình và resize form cho lớn ra, ta sẽ thấy Label1 dãn ra hai bên, nhưng không hề tăng bề cao, và Text của Label1 luôn luôn nằm ở giữa.

Nếu bạn tìm cách dock nhiều controls vào cùng một cạnh của form thì VB.NET phải quyết định control nào nằm sát cạnh ấy nhất. Qui ước về thứ tự là ngược lại với thứ tự trong z-order. Tức là trong z-order, control nào nằm dưới nhất thì lại được dock trước nhất vào cạnh của form. Do đó, nếu bạn dock hai controls vào một cạnh, và muốn cái control nằm xa cạnh được dock trước nhất (tức là sát cạnh nhất) thì right click control ấy và chọn **Send To Back**.

Nếu bạn muốn chừa một khoảng trống giữa control và cạnh của container thì set **Property DockPadding** của container. Tự trước đến giờ ta dùng form để đại diện container chứa controls. Thật ra container cũng có thể là một **Panel**. Bạn có thể set Property DockPadding của các cạnh của

container khác nhau bằng cách click dấu + bên trái chữ DockPadding trong cửa sổ Properties để mở ra các chi tiết như trong hình dưới đây:



Bạn có thể set tất cả Property DockPadding cùng một trị số bằng cách dùng **All** setting.

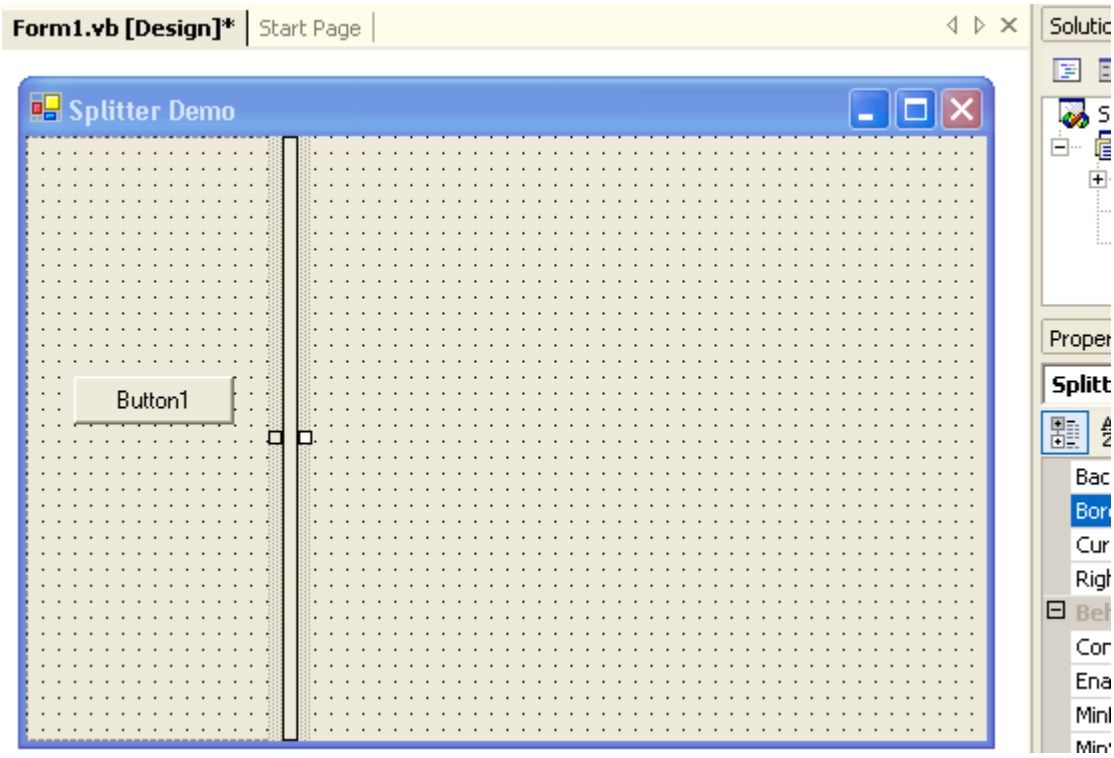
Control Splitter

Bây giờ bạn đã hiểu rõ các đặc tính, sự khác biệt và cách dùng hai properties Anchor và Dock của control, sau đây ta sẽ áp dụng kiến thức ấy vào việc thiết kế dùng Splitter trong một form. Nếu còn mới với Splitter bạn sẽ dễ bị bực mình khi dùng nó. Do đó, bạn hãy thử làm theo các bước sau đây:

1. Tạo một Application mới, đặt một Panel lên phía trái của form chính để nó chiếm bên trái của form bằng cách set property Dock của nó thành Left. Ta gọi Panel ấy là Panel1.

2. Đặt một Splitter lên form (nhớ tránh đặt nó lên Panel1 vì Panel cũng là một loại container nên có thể chứa Splitter được). Splitter sẽ tự động dock Left vào form tức là nằm bên phải Panel1. Chọn **property BoderStyle** của Splitter1 làm **FixedSingle** cho dễ thấy.

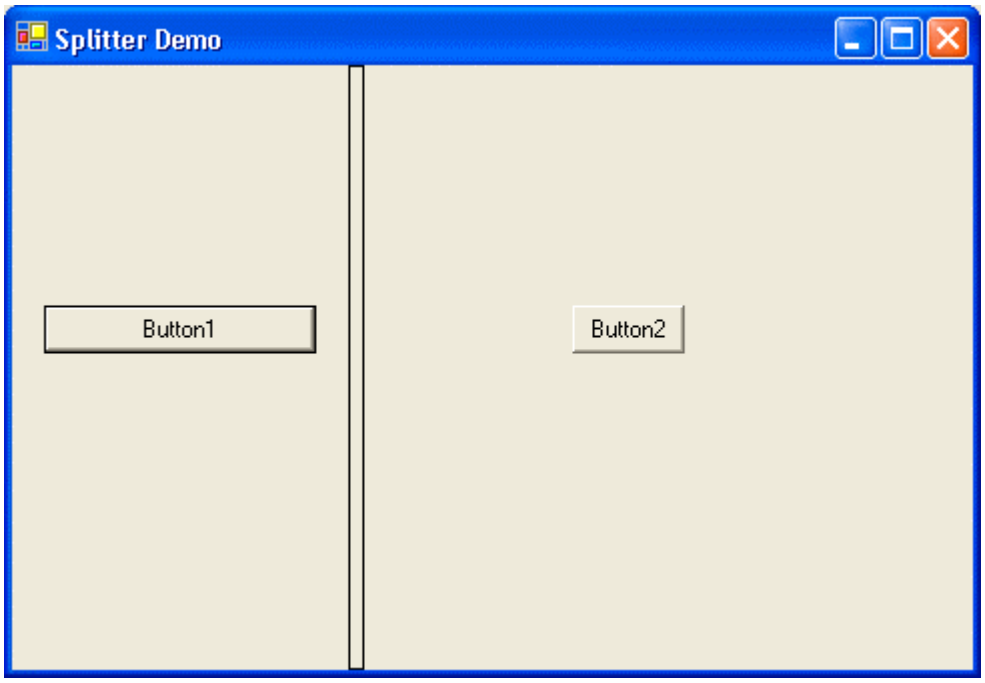
3. Đặt một button lên Panel1 và set property Anchor của nó thành Top, Left, Right. Bây giờ form sẽ giống như dưới đây:



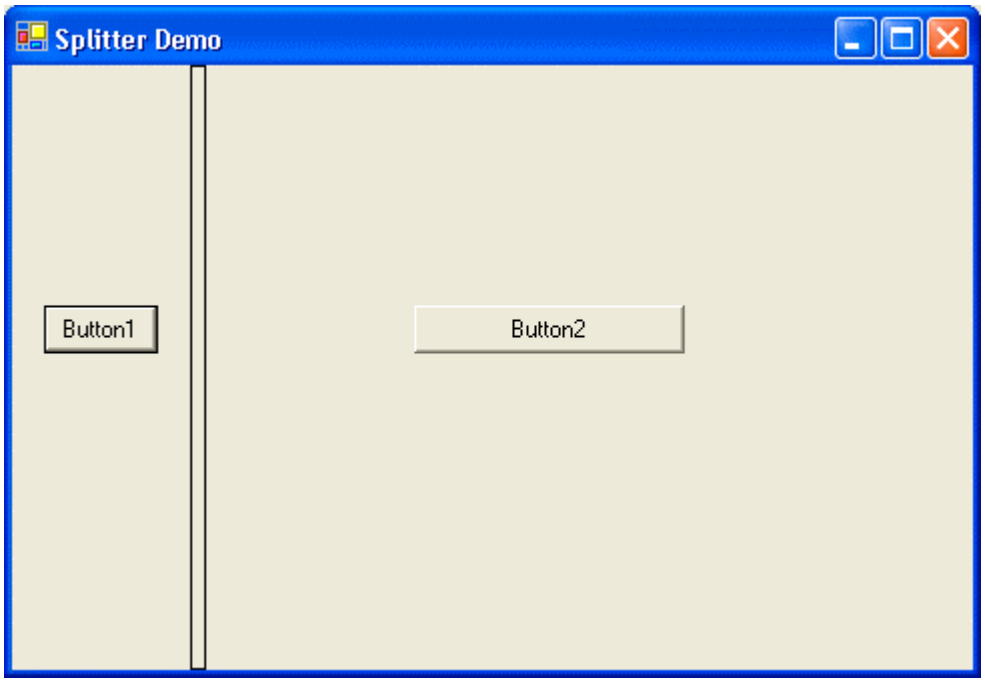
4. Kế đó, đặt một Panel lên bên phải của form, gọi là Panel2, và set property Dock nó thành Fill. Có nghĩa là ta muốn Panel2 chiếm hết phần còn lại bên phải của form.

5. Thêm vào trong Panel2 này một Button, gọi là Button2, và set property Anchor của nó thành Top, Left, Right.

Khi chạy chương trình, mỗi lần bạn nắm Splitter kéo qua phải thì Button1 dãn ra và Button2 co lại:



Ngược lại, nếu bạn nắm Splitter kéo qua trái thì Button1 co ra và Button2 dãn lại:



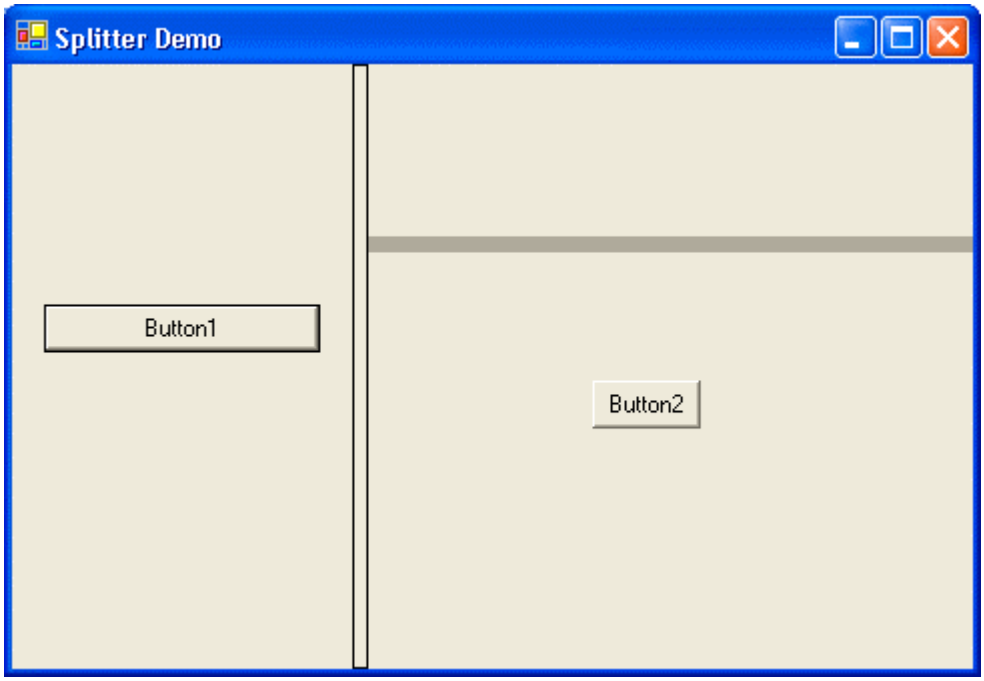
Trong thí dụ này ta để yên chiều rộng của Splitter, nhưng bình thường ta làm cho nó hẹp hơn. Nếu Splitter hẹp thì khó thấy, do đó bạn có thể cho nó một màu đỏ rực trong lúc thiết kế để dễ thấy. Khi thiết kế xong hết rồi, bạn đổi nó lại thành một màu dịu hơn.

Nếu bây giờ bạn muốn chia Panel2 thành hai phần, ngăn cách bởi một Horizontal Splitter thì sao? Ta cứ xem Panel2 như một form vậy, tức là cả

hai đều là containers, loại control có thể đựng nhiều controls, và lập lại các bước sau:

1. Đặt một Panel lên phía trên của Panel2, gọi nó là Panel3 và set property Dock của nó thành Top.
2. Đặt một Splitter lên Panel2 (nhớ tránh đặt nó lên Panel3), gọi nó là Splitter2 và set property Dock của nó cũng thành Top. Resize Splitter2 cho nó dẹp lại và đổi **property Backcolor** thành ra **ControlDark** cho dễ thấy.
3. Đặt một Panel lên phía dưới của Panel2, gọi nó là Panel4 và dời Button2 từ Panel2 qua Panel4 bằng cách Cut and Paste.
4. Set property Dock của Panel4 thành Fill.

Bây giờ hãy chạy chương trình và nắm kéo Splitter2 lên xuống.



Tóm lại, muốn dùng control Splitter trong một form hay panel ta đặt một PanelX với Docking Left hay Top lên trước, kế đó đặt một Splitter với cùng loại Docking với PanelX, rồi đặt PanelY với Docking Fill.

Bài 11
Những chức năng mới trong giao diện cửa sổ của VB.NET (phần IV)

Các control Providers

Trong Windows Forms có một gia đình controls mới mà ta chỉ có thể dùng khi chúng đi chung với các controls khác trên cùng một form. Chúng được gọi là **Provider Controls** và có đặc tính là khiến cho các property mới hiện ra trong các controls khác.

Provider Controls không hiển thị trên form lúc chạy program. Do đó chúng nằm riêng trong Component Tray lúc ta thiết kế. Hiện giờ có 3 Provider Controls : **HelpProvider, ToolTip** và **ErrorProvider**. Cả ba đều làm việc một cách tương tự nhau.

Controls HelpProvider và ToolTip

Trong VB6, các controls có **property HelpContextID** để ta chỉ định khi user bấm nút **F1** thì chương trình sẽ hiển thị Help ở đúng trang có trị số HelpContextID trong Help file. Còn **ToolTip** là một Textstring property của mỗi control. Ta chỉ cần dùng cửa sổ Properties để cho vào ToolTip text của một control là trong lúc chạy chương trình, khi nào ta để mouse cursor nằm lên control là chương trình sẽ hiển thị ToolTip text.

Hai thứ ấy không còn dùng trong Windows Forms nữa. Thay vào đó, ta phải đặt các Provider Controls lên form để thực hiện các công tác tương đương.

Control HelpProvider cho phép các controls khác chỉ định context sensitive help (trợ giúp trong tình huống đương thời) hiển thị khi user bấm nút **F1**. Khi một control HelpProvider (gọi là HelProvider1 by default) được thêm vào một form, thì mọi controls trên form đều sẽ có thêm các properties dưới đây, chúng sẽ hiển thị trong cửa sổ Properties sau khi ta chọn một control.

Property	Áp dụng

HelpString on HelpProvider1	Khi control được focus, user bấm nút F1 sẽ popup Tooltip HelpString cho control
HelpTopic on HelpProvider1	Cung cấp một Topic cho control để dùng trong Help file cho context-sensitive help. Control HelpProvider1 có một property để ta chỉ định dùng Help file nào
ShowHelp on HelpProvider1	Xác định là control HelpProvider có **Active** cho control này không

Một khi **property HelpString** đã được cho một Textstring thì trong lúc control nhận được focus, nếu user bấm nút F1 một Tooltip sẽ hiển thị Textstring ấy. HelpProvider có một property để dẫn đến một Help file, hoặc là HTMLHelp file, hoặc là Win32Help file, và trị số trong **property HelpTopic** sẽ chỉ dẫn đến topic ấy trong Help file.

Trong lúc chương trình chạy, ta cũng có thể thay đổi trị số HelpString của **Textbox1** như sau:

```
HelpProvider1.SetHelpString(Textbox1, "Một HelpString mới được dùng tại đây.")
```

Control ToolTip cũng hoạt động tương tự, nhưng đơn giản hơn. Nó chỉ cho thêm một property mới tên **ToolTip on ToolTip1** vào mỗi control, giả dụ tên của ToopTip provider là ToolTip1. Property này làm việc y hệt như ToolTipText trong VB6.

Trong lúc chương trình chạy, ta cũng có thể set cho property Tooltip của Textbox **txtName** một trị số Textstring như sau:

```
ToolTip1.SetToolTip(txtName, "Xin vui lòng đánh tên bạn vào đây")
```

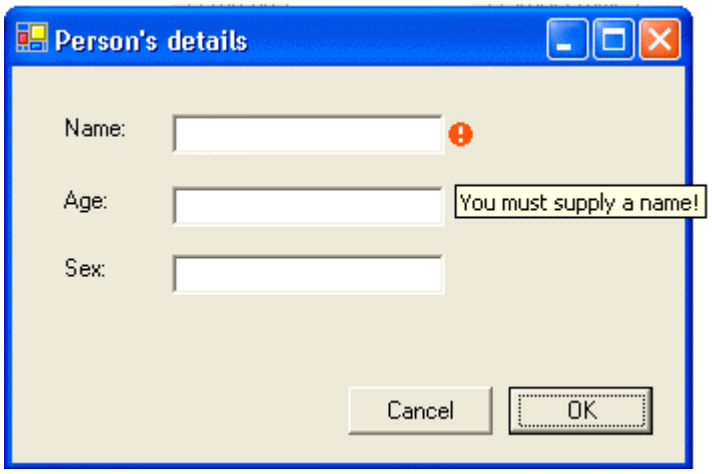
Control ErrorProvider

Thông thường sau khi user điền xong các dữ kiện vào một form thì sẽ click một button **OK** hay **Submit** chẳng hạn. Để tránh trường hợp cập nhật data của một record với những dữ kiện bất hợp lệ, ta thường kiểm tra lại dữ kiện nằm trong từng Textbox trên form và hiển thị một thông điệp để nhắc nhở và giải thích cho user khi có error. Nếu user lầm lỗi ở nhiều Textboxes thì có thể sẽ có nhiều thông điệp hiển thị lần lượt cái này tiếp theo cái kia, mỗi thông điệp liên hệ đến một Textbox có error. Cách ấy

cũng tạm được, nhưng có thể khiến cho user bực mình.

Control ErrorProvider cung cấp một cách đơn giản và thân thiện để cho user biết Textbox nào có dữ kiện bất hợp lệ. Control ErrorProvider cho các controls trên cùng form một property mới gọi là **Error on ErrorProvider1** (giả dụ là control ErrorProvider mang tên ErrorProvider1).

Trong lúc chương trình chạy, nếu kiểm thấy một Textbox có lỗi ta assign một TextString vào property Error on ErrorProvider1 của Textbox ấy. Lúc bấy giờ một icon đỏ hình dấu chấm than trắng sẽ hiển thị bên phải Textbox có Error. Nếu user để mouse cursor lên trên icon ấy thì chương trình sẽ hiển thị một Tooltip với trị số TextString của property Error on ErrorProvider1 giống như trong hình dưới đây:



Công việc assign một TextString vào property Error on ErrorProvider1 của một Textbox có thể được coded như sau:

```
Private Sub BtnOK_Click( ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
BtnOK.Click

 ' Set error if TextBox txtName is blank

  If txtName.Text = "" Then

   ' Assign error ToolTip message to Textbox txtName

    ErrorProvider1.SetError(txtName, "You must supply a name!")

  End If

End Sub
```

Trên đây ta dùng Event Click của button BtnOK để kiểm tra dữ kiện trong mọi Textbox. Có một Event của các controls mà ta cũng có thể dùng trong công tác kiểm tra dữ kiện của một TextBox. Đó là **Event Validating**. Để gây ra Event Validating ta cần phải dùng **property CauseValidation** của các controls. Thông thường, property CauseValidation của các controls được set thành True. TextBox txtName chỉ tạo ra Event Validating khi chính property CauseValidation của nó là True và khi focus được di chuyển đến một control khác có property CauseValidation là True.

Xin lưu ý là không nhất thiết Event Validating được tạo ra khi txtName mất focus. Khi txtName mất focus thì Textbox txtAge được focus (giả dụ txtAge có trị số TabOrder ngay sau txtName) , nhưng nếu property CauseValidation của txtAge không phải là True thì phải đợi đến khi focus đáp lên một control có property CauseValidation là True txtName mới gây ra Event Validating.

Ta có thể code cho **Sub txtName_Validating** như sau:

```
Private Sub txtName_Validating( ByVal sender As Object, ByVal e As
System.ComponentModel.CancelEventArgs)  _

                                    Handles txtName.Validating

 ' Set error if TextBox txtName is blank

  If txtName.Text = "" Then

    ' Assign error ToolTip message to Textbox txtName

     ErrorProvider1.SetError(txtName, "You must supply a name!")

  Else

    ' Clear the error ToolTip message for Textbox txtName and make error Icon invisible

     ErrorProvider1.SetError(txtName, "")

  End If

End Sub
```

Cái icon đỏ hình dấu chấm than trắng là **default icon** của ErrorProvider. Muốn dùng một icon khác ta chỉ cần assign icon ấy vào **property Icon** của ErrorProvider.

Menus

Mặc dầu **Menu Editor** của VB6 cung cấp đầy đủ các phương tiện để làm Menu và tương đối dễ dùng, VB.NET cho ta một giao diện càng thân thiện và tự nhiên hơn để thiết kế Menu.

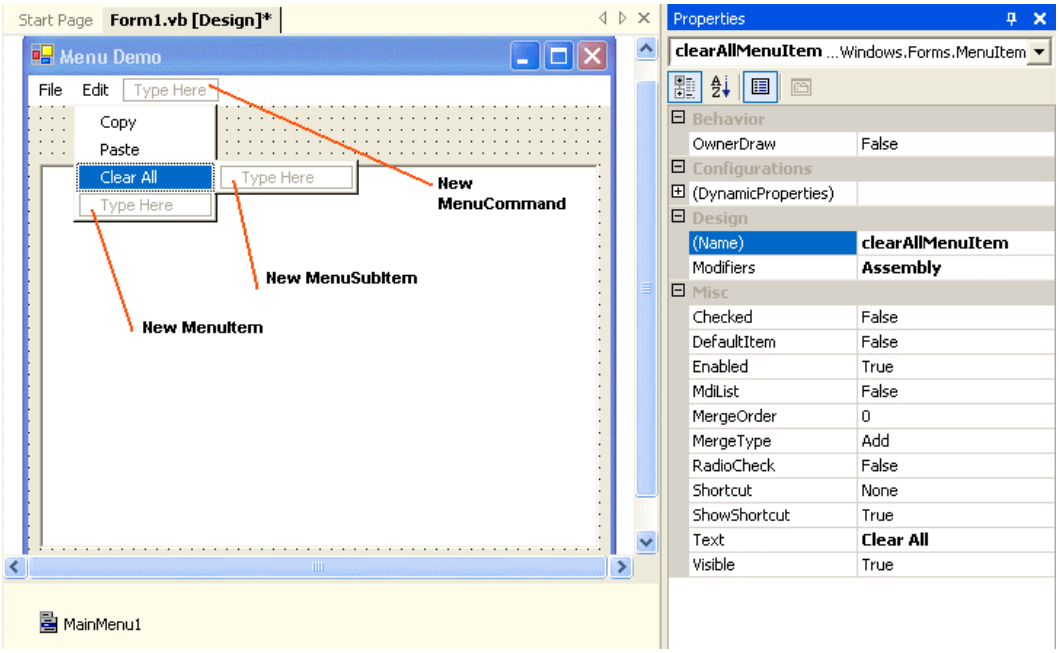
Menu được thêm vào form dưới dạng một control. Tuy Menu control nằm trong một mâm components phía dưới, nhưng trong lúc thiết kế, Menu hiện ra trong form y như lúc Runtime và bạn chỉ cần điền vào các menuitems cần thiết. Có hai loại menus: Main Menu (Menu dùng thông thường) và Context Menu (dùng cho Pop-Up).

Menus

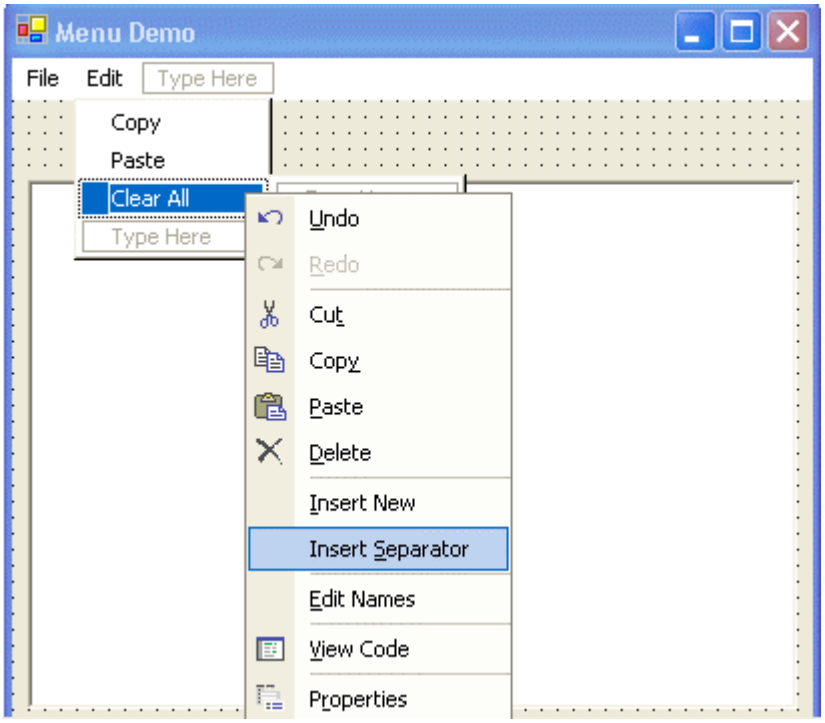
Main Menu là Menu căn bản mà bạn thấy nó dính vào cạnh trên của một form. Để dễ giải thích, ta sẽ dùng một thí dụ tạo ra một Editor thật đơn giản bằng VB.NET. Bạn hãy khởi động một Windows Application mới và thêm một Textbox vào trong form chính. Set **property MultiLine** của Textbox thành True để nó có thể hiển thị nhiều hàng, đồng thời Stretch (kéo dãn ra) cái Textbox cho lớn ra làm nơi ta có thể đánh vào một bài text.

Kế đó, thêm một Main Menu vào form. Cái menu Designer sẽ kích động và bạn chỉ cần đánh vào chi tiết các menuItems và dùng cửa sổ Properties để set các parameters.

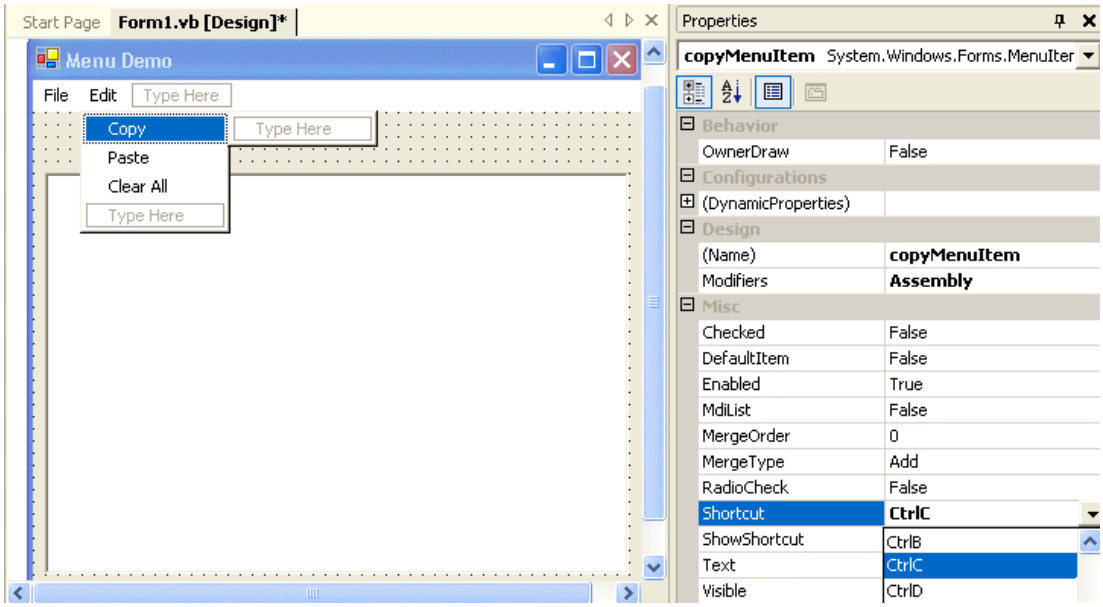
Khi nào bạn select control Main Menu trong mâm components là bạn có thể edit các MenuItems. Muốn làm việc với MenuItem nào thì select MenuItem đó. Những chỗ có chữ **Type Here** là đề nghị cho bạn đánh thêm vào một MenuItem (**Type Here** nằm phía dưới) , một MenuCommand mới (**Type Here** nằm bên phải một MenuCommand) hay một MenuSubItem (**Type Here** nằm bên phải một MenuItem).



Muốn insert một lằn ngang giữa MenuItem **Paste** và MenuItem **Clear All**, bạn select MenuItem Clear All rồi right click và chọn **Insert Separator** trong Pop-Up Menu.



Muốn chỉ định Shortcut cho một MenuItem, bạn select MenuItem ấy rồi vào cửa sổ Properties để chọn trị số cho **property Shortcut**. Tương tự như vậy cho **property Checked** để làm một checkmark hiện ra bên trái (phía trước) Text của MenuItem.

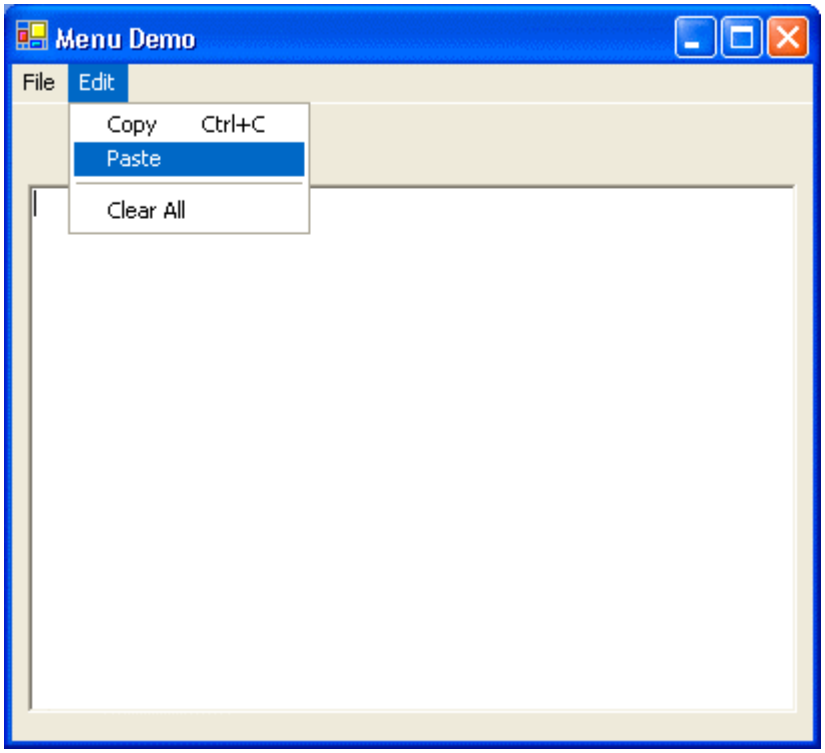


Thêm vào các dòng code sau đây cho chương trình. Khi doubleClick lên MenuItem **copyMenuItem** cửa sổ mã nguồn sẽ mở ra cho bạn đánh code cho **Private Sub copyMenuItem_Click**:

```
Private Sub copyMenuItem_Click( ByVal sender As System.Object, ByVal e As System.EventArgs)
                        Handles copyMenuItem.Click
 ' Copy the selected text to the Clipboard
   Textbox1.Copy()
End Sub

Private Sub pasteMenuItem_Click( ByVal sender As System.Object, ByVal e As System.EventArgs)
                        Handles pasteMenuItem.Click
 'Paste the Clipboard text into Textbox1
   Textbox1.Paste()
End Sub

Private Sub clearAllMenuItem_Click( ByVal sender As System.Object, ByVal e As System.EventArgs)
                        Handles clearAllMenuItem.Click
 ' Clear everything in Textbox1
   Textbox1.Text = ""
End Sub
```

```
Private Sub closeMenuItem_Click( ByVal sender As System.Object, ByVal e As System.EventArgs)
                    Handles closeMenuItem.Click
 ' Close the form
  Me.Close()
End Sub
```

Khi chạy chương trình, hình dưới đây sẽ hiển thị:



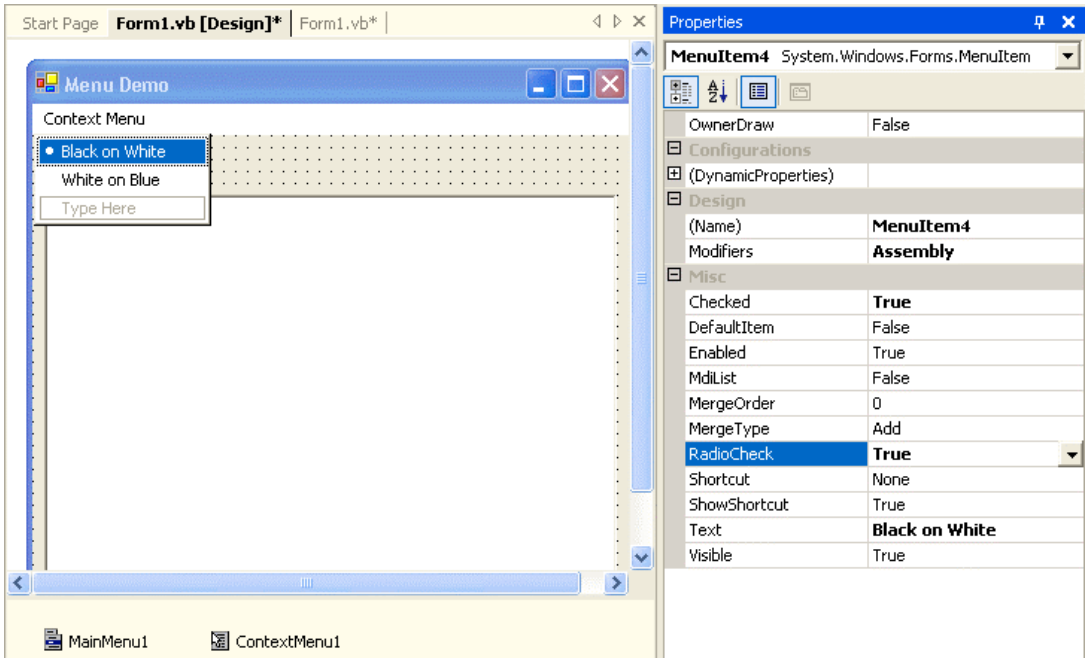
Context Menus

Ta dùng Context Menu để Pop-Up một Menu xứng hợp với tình huống đương thời của program khi user right click một control trên form. Trong VB6, Context Menu cũng là một MenuCommand thông thường nhưng ta thiết kế cho nó invisible, để chỉ khi nào ta muốn Pop-Up nó thì nó mới hiển thị.

Trong VB.NET, Context Menu là một control riêng, nhưng ta edit nó cũng giống như Main Menu. Khi đã thêm một control ContextMenu vào form rồi, mỗi lần ta select nó trong mâm components thì Context Menu hiện ra ở cạnh trên của form giống như Main Menu. Lúc Runtime, khi

user right click một control có Context Menu thì ContextMenu sẽ hiển thị ở vị trí đó.

Bạn hãy doubleClick control ContextMenu trong hộp đồ nghề để thêm một Context Menu vào trong form. Kế đó set up các MenuItem như sau:



Để hiển thị cái **Radio button** bên trái một MenuItem, bạn phải làm hai chuyện:

1. Set **property Checked** của MenuItem thành True để hiển thị một **checkmark** hay một hình tròn nhỏ (**Radio button**).

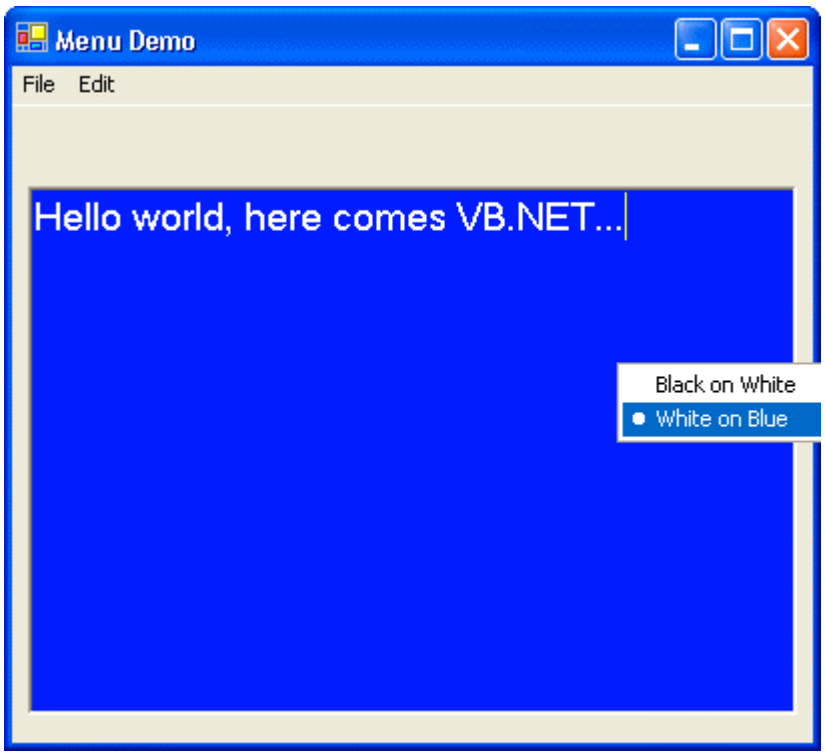
2. Set **property RadioCheck** của MenuItem thành True để khi nào nó hiển thị thì có dạng Radio button, thay vì một checkmark.

Nhớ là ta dùng **checkmark** khi muốn cho user chọn nhiều thứ cùng một lúc, và dùng **Radio button** khi muốn cho user chỉ chọn một nhiệm ý mà thôi, tức là **mutually exclusive**. Tuy nhiên, khác với khi edit một nhóm Radio buttons trong một container trên form, VB.NET không cản trở ta cho hai Radio buttons trong một menu cùng hiện ra. Do đó, bạn phải tự quản lý vấn đề mutually exclusive trong code của mình.

Để chỉ định ContextMenu1 Pop-up khi user right click Textbox1, bạn chỉ cần set property ContextMenu của Textbox1 thành ContextMenu1 (chọn

nó trong cái dropdown list của comboxbox của property ContextMenu trong cửa sổ Properties).

Khi bạn chạy chương trình và right click Textbox1, ContextMenu1 sẽ hiển thị như dưới đây:



Mã nguồn nằm phía sau các click events của hai MenuItems của ContextMenu1 được liệt kê dưới đây:

```
Private Sub blackOnWhiteMenuItem_Click( ByVal sender As System.Object,  _
                    ByVal e As System.EventArgs) Handles blackOnWhiteMenuItem.Click
 ' Change colors of Textbox1
   Textbox1.ForeColor = Color.Black
   Textbox1.BackColor = Color.White
 'Toggle the radio check
   blackOnWhiteMenuItem.Checked = True
   WhiteOnBlueMenuItem.Checked = False
End Sub

Private Sub WhiteOnBlueMenuItem_Click( ByVal sender As System.Object,  _
```

```
                ByVal e As System.EventArgs) Handles WhiteOnBlueMenuItem.Click
 ' Change colors of Textbox1
  Textbox1.ForeColor = Color.White
  Textbox1.BackColor = Color.Blue
 'Toggle the radio check
  blackOnWhiteMenuItem.Checked = False
  WhiteOnBlueMenuItem.Checked = True
End Sub
```

Để ý property Checked của hai MenuItems được coded để hễ cái này True thì cái kia phải False, tức là mutually exclusive. Và MenuItem nào có trị số Checked là True thì Radio button hiển thị phía trước nó.

Sửa đổi Menus lúc Runtime

Ta có thể sửa đổi Menu lúc Runtime, chẳng hạn như Context Menu thường có những dạng khác nhau tùy theo trạng thái của một control hay form.

Một thí dụ khác là hiển thị danh sách các files mà chương trình truy cập trong quá khứ. Thông thường ta chứa tên các files ấy trong Registry và khi cần sẽ đọc và load vào Menu.

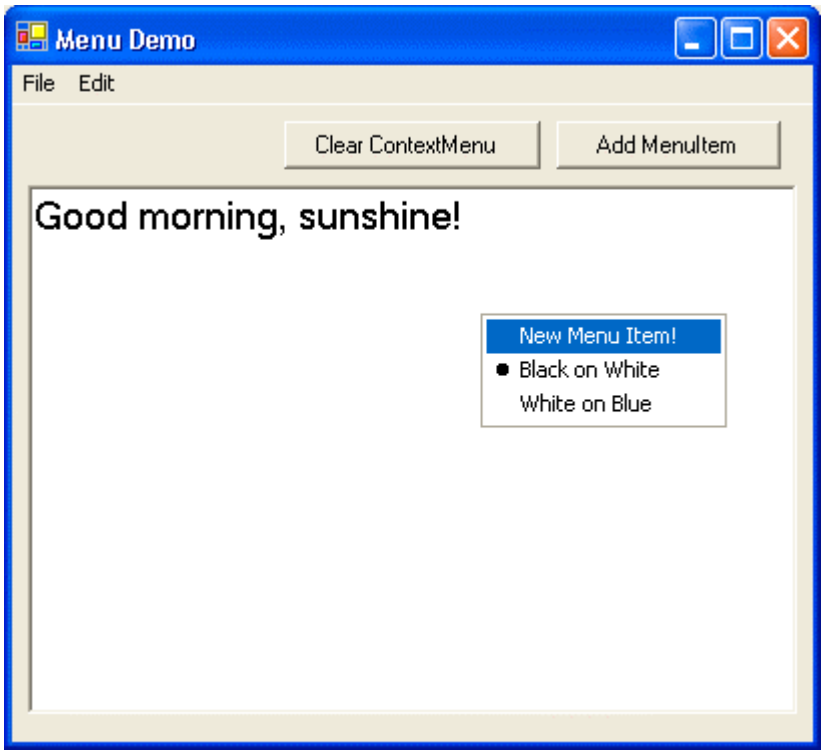
Dưới đây là code chỉ cách cho thêm một MenuItem vào trong một ContextMenu, và cách clear (xóa) mọi MenuItems. Ta biết rằng ContextMenu có một property là collection của những MenuItems. Do đó muốn thêm một MenuItem thì cần trải qua ba bước:

1. Instantiate một MenuItem.
2. Đăng ký Event Handler (ở đây là **AddressOf Sub NewMenuItem_Click**), mà chương trình sẽ dùng để xử lý Event Click của MenuItem ấy.
3. Thêm MenuItem ấy vào **collection MenuItems** của control ContextMenu.

Thêm vào form hai buttons đặt tên là **BtnAddMenuItem** và **BtnClearContextMenu**.

```
Private Sub BtnAddMenuItem_Click( ByVal sender As System.Object, ByVal e As System.EventArgs)
_
                                Handles BtnAddMenuItem.Click
 'Add a menu item at the top of ContextMenu1
  Dim AnewMenuItem As MenuItem ' Declare a MenuItem variable
 ' Create the new menu Item
  AnewMenuItem = New MenuItem("New Menu Item!")
 ' Register EventHandler for Event Click of this new Menu item
  AddHandler AnewMenuItem.Click, AddressOf Me.NewMenuItem_Click
 ' Add it to the collection MenuItems
  ContextMenu1.MenuItems.Add(0, AnewMenuItem)
End Sub

Private Sub NewMenuItem_Click( ByVal sender As System.Object, ByVal e As System.EventArgs)
  MessageBox.Show("You clicked new Menu Item!")
End Sub

Private Sub BtnClearContextMenu_Click( ByVal sender As System.Object, ByVal e As
System.EventArgs)  _
                                Handles BtnClearContextMenu.Click
 ' Remove all the menu items from ContextMenu1
  ContextMenu1.MenuItems.Clear()
End Sub
```

Sau khi bạn click nút **Add MenuItem**, lúc bạn right click Textbox1, Pop-up Menu sẽ có thêm một MenuItem như sau:



Thử click new Menu Item trong ContextMenu1, chương trình sẽ hiển thị thông điệp **You clicked new Menu Item!**. Bây giờ click nút **Clear ContextMenu** rồi right click Textbox1. ContextMenu1 đã bị cleared nên sẽ không hiển thị.

Duplicating Menus

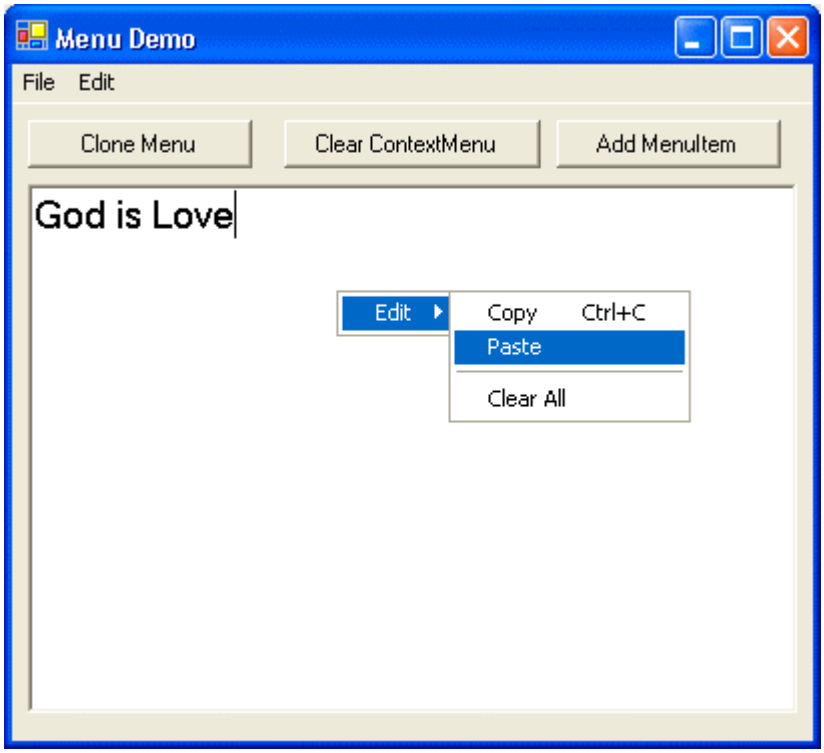
Một việc khác ta có thể làm trong lúc Runtime của chương trình là **cloning** (tạo object song sinh). Thí dụ, ta muốn dùng **Edit** menu của MainMenu1 làm ContextMenu (giống giống như trong VB6) cho Textbox1.

Để thực hiện việc này, ta dùng **method CloneMenu()**. Dưới đây là code ta dùng để thay thế ContextMenu1 trong chương trình bằng Edit menu của MainMenu1.

```
Private Sub BtnCloneMenu_Click( ByVal sender As System.Object, ByVal e As System.EventArgs) _
                                Handles BtnCloneMenu.Click
 ' Instantiate a new ContextMenu object
  Dim newContextMenu As New ContextMenu()
 ' Add a clone copy of EditMenu to this new ContextMenu's collection of MenuItems
  newContextMenu.MenuItems.Add(editMenuItem.CloneMenu)
```

```
' Assign this new Context Menu to Textbox1
  Textbox1.ContextMenu = newContextMenu
End Sub
```

Khởi động chương trình, click nút **Clone Menu**, rồi right click TextBox1, ContextMenu mới sẽ hiển thị như dưới đây:



Lưu ý: Vì **CloneMenu()** clone hoàn toàn Object editMenuItem, kể cả các Event Handlers của các SubMenuItems nên ta không cần phải làm thêm gì cả.

Muốn trở lại trạng thái cũ, tức là dùng ContextMenu1 cho Textbox1, ta chỉ cần reassign ContextMenu1 vào property ContextMenu của Textbox1 như sau:

```
Textbox1.ContextMenu = ContextMenu1
```

MDI Forms

Trong VB6 ta tạo một **MDI (Multiple Document Interface)** form bằng cách set property **MDIChild** của form ấy thành True. Một form như thế chỉ có thể được dùng làm child form, tức là nó cần một form MDI parent để hiển thị trong ấy. Ngoài ra, mỗi application chỉ có thể có một form

MDI parent duy nhất và chỉ trong lúc thiết kế ta mới có thể chỉ định đặc tính của một form là MDIChild. Một form không thể trở thành một MDIChild lúc Runtime.

Trong VB.NET, một form có thể trở thành một MDI child lúc Runtime bằng cách set **property MDIParent** của form ấy để nhắm vào một form MDI parent. Do đó, một form có thể vừa là MDIchild form, vừa là form bình thường tùy theo hoàn cảnh. Thật ra, ngược với VB6, ta không thể set property MDIParent lúc thiết kế, mà phải làm lúc Runtime.

Giống như VB6, trong VB.NET ta có thể hiển thị nhiều forms MDIChild trong một form MDI parent, khi parent form di chuyển thì mang theo các forms con. Khi hiển thị nhiều child forms, ta có thể dùng **property ActiveForm** để biết child form nào hiện thời là Active.

Ta thử khởi động một Windows Application mới. Đổi tên Form1 thành ParentForm và chỉ định nó làm MDI parent bằng cách set **property IsMDIContainer** của nó thành True. Kế đó thêm một form và đổi tên nó thành ChildForm. Dưới đây là code để thêm hai child forms vào ParentForm và hiển thị chúng:

```
' Declare child forms of type ChildForm
Private WithEvents FirstChild As ChildForm
Private WithEvents SecondChild As ChildForm

Private Sub ParentForm_Load( ByVal sender As System.Object, ByVal e As System.EventArgs)  _
                            Handles MyBase.Load
  ' Instantiate an Object of type Childform
   FirstChild = New ChildForm()
  ' Make this form the MDI Parent of FirstChild
   FirstChild.MdiParent = Me
   FirstChild.Text = "First Child Form" ' Set Title
  ' Show FirstChild
   FirstChild.Show()
```

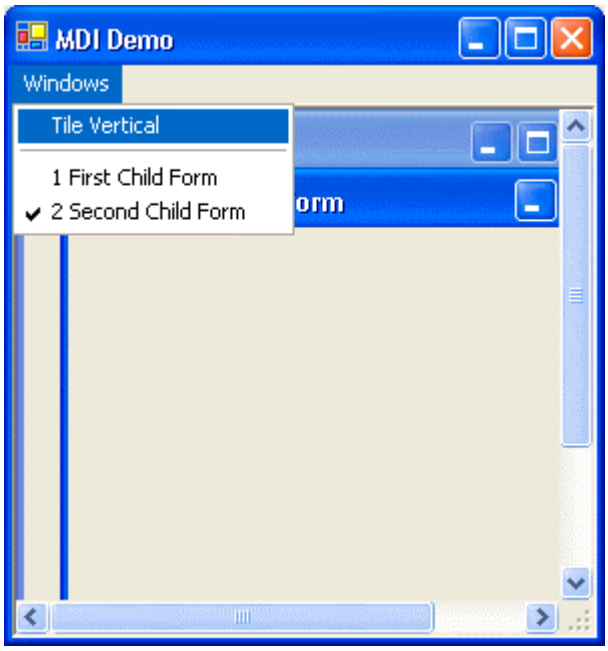
```
 ' Instantiate the second Object of type Childform
  SecondChild = New ChildForm()
 ' Make this form the MDI Parent of SecondChild
  SecondChild.MdiParent = Me
  SecondChild.Text = "Second Child Form" ' Set Title
 ' Show SecondChild
  SecondChild.Show()

End Sub
```

Để cung cấp một Menu hiển thị danh sách các forms MDIchild của ParentForm, ta thêm control MainMenu vào ParentForm. Kế đó, tạo một MenuItem tên **Windows** và set **property MDIList** của nó thành True. Property này sẽ khiến danh sách các forms child tự động hiển thị làm những menu items nằm phía dưới Menu Windows. Danh sách này tự động cập nhật khi một child form trở thành Active, được thêm vào, hay bị lấy ra.

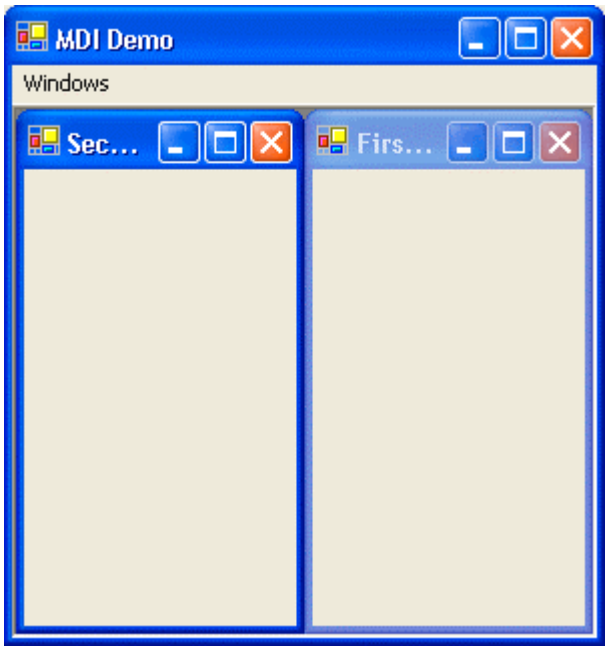
Parent MDI form có một method tên là **LayoutMDI** để tự động sắp đặt vị trí các forms child theo kiểu **Cascade** hay **Tile** layout. Thêm một MenuItem tên **Tile Vertical** và nhét mấy hàng code dưới đây vào form để xử lý Event click của nó:

```
Private Sub tileVerticalMenuItem_Click( ByVal sender As System.Object, ByVal e As
System.EventArgs)  _
                Handles tileVerticalMenuItem.Click
  Me.LayoutMdi(System.Windows.Forms.MdiLayout.TileVertical)
End Sub
```

Khởi động chương trình, by default hai forms childs được layout kiểu **Cascade**. Trong Menu Windows có hiển thị title của hai forms child và cho biết **Second Child Form** là **Active** form.



Nếu bạn click **Tile vertical**, hai forms child sẽ được layout kiểu **Tile** như dưới đây:



Bài 12
Những chức năng mới trong giao diện cửa sổ của VB.NET (phần V)

Toolbars

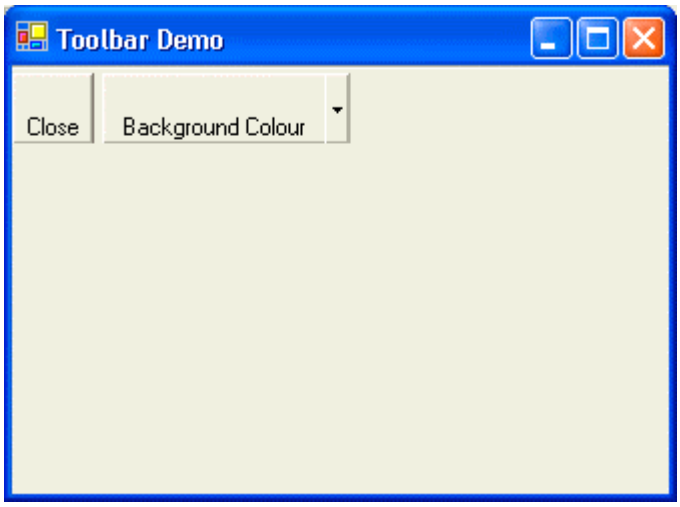
Toolbars trong .NET đã được nâng cấp bằng cách thêm chức năng cho các **ToolBarButtons** trong collection của những buttons ấy.

Để dùng thử Toolbar control, bạn hãy khởi động một Project mới và đặt một Toolbar vào form chính bằng cách doubleclick lên Toolbar icon trong Toolbox. Một Toolbar sẽ hiện ra nằm ngay dưới tiêu đề của form. Kế đó rightclick lên Toolbar ấy và chọn Properties để edit property Buttons Collection bằng cách click lên chữ **(Collection)** rồi click ba dấu chấm phía bên phải để hiển thị **ToolbarButton Collection Editor**.

Bạn hãy **Add** vào Toolbar ba buttons với những đặc tính sau:

- Đổi property **Text** của button thứ nhất (**ToolbarButton1**) ra **Close** vì ta muốn đóng chương trình khi user click lên button ấy. By default **Style** của ToolbarButton là **PushButton**.
- Đổi property **Style** của button thứ nhì (**ToolbarButton2**) ra **Separator** vì ta muốn dùng nó để tạo khoảng cách giữa button thứ nhất và button thứ ba.
- Đổi property **Text** của button thứ ba (**ToolbarButton3**) ra **Background Colour** và property **Style** ra **DropDownButton** vì ta muốn dùng nó như một Combobox.

Khi chạy thử chương trình ta sẽ thấy hình giống như dưới đây:



Bây giờ ta sẽ viết code để xử lý Event Click của Toolbar. Chỉ có một handler, **Sub ToolBar1_ButtonClick**, được dùng cho tất cả các buttons. Ta phân biệt Button nào dựa vào **Index** của nó, giống giống như một array of buttons trong VB6. Nếu user click button thứ nhất ta sẽ có **ToolBar1.Buttons.IndexOf(e.Button)** bằng **0**, lúc ấy ta sẽ Close form chính.

```
Private Sub ToolBar1_ButtonClick( ByVal sender As System.Object, ByVal e As
System.Windows.Forms.ToolBarButtonClickEventArgs) Handles ToolBar1.ButtonClick

  Select Case ToolBar1.Buttons.IndexOf(e.Button)

  Case 0 ' Close Button

    Me.Close()

  Case 1 ' Never happens because the Button is a Separator



  Case 2 '

    MessageBox.Show("You clicked the third button")

  End Select

End Sub
```

Nếu không muốn dùng ToolBar1.Buttons.IndexOf(e.Button), bạn cũng có thể so sánh Buttons với operator **Is** như sau:

```
If e.Button Is ToolBarButton1 Then

  Me.Close()

ElseIf e.Button Is ToolBarButton3 Then
```

```
  MessageBox.Show("You clicked the third button")

End If
```

Kế đó chúng ta cho đặt một ContextMenu tên **ContextMenu1** vào form và assign nó vào property **DropDownMenu** của button thứ ba như trong hình dưới đây:

Nếu không muốn assign ContextMenu1 vào button thứ ba trong lúc thiết kế, bạn có thể thực hiện việc ấy bằng code lúc form mới load như sau:

```
Private Sub frmToolbar_Load( ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.Load

  ToolBarButton3.DropDownMenu = ContextMenu1

End Sub
```

Bạn hãy edit hai menuItems cho ContextMenu1: một cái tên **mnuXám** với Text là **Xám** và cái kia tên **mnuTrắng** với Text là **Trắng**.

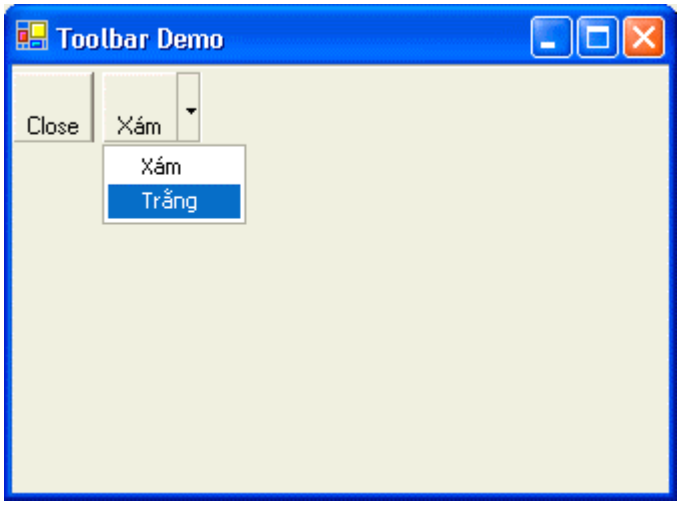
Khi chạy chương trình, nếu bạn click cái thanh có dấu tam giác đen nằm bên phải button thứ ba, ContextMenu1 sẽ hiện ra để bạn dùng. Nếu bạn click button thứ ba, chương trình cũng generate một Click Event nhưng hiện giờ ta không dùng nó, chỉ hiển thị một sứ điệp nhỏ để xác định là có Event Click ấy.

Như thế, ta thấy .NET ghép một ContextMenu vào một ToolbarButton để biến nó thành một DropDownMenu. Có điều sau khi user đã chọn một Item trong ContextMenu/DropDownMenu, Text của Item đó không được hiển thị giống như trong một ComboBox. Nếu bạn khó tính và muốn có chuyện đó thì phải tự làm lấy như cho thấy trong code dưới đây:

```
Private Sub frmToolbar_Load( ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.Load

  ToolBarButton3.DropDownMenu = ContextMenu1

  ToolBarButton3.Text = "Xám"

End Sub


Private Sub mnuXám_Click( ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
mnuXám.Click

  MessageBox.Show("Bạn chọn màu Xám")
```

```
  ToolBarButton3.Text = "Xám"

End Sub


Private Sub mnuTrắng_Click( ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles mnuTrắng.Click

  MessageBox.Show("Bạn chọn màu Trắng")

  ToolBarButton3.Text = "Trắng"

End Sub
```

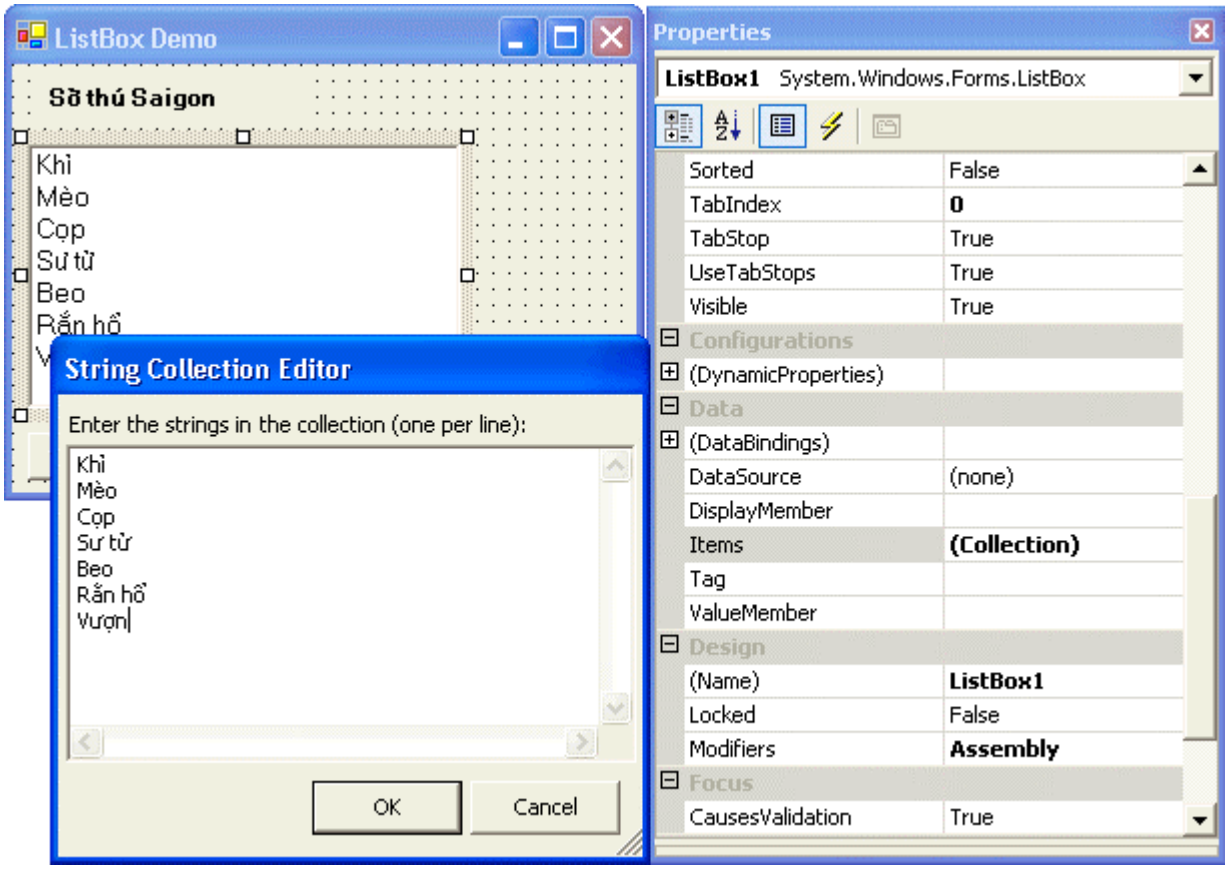
Khi chạy chương trình bạn sẽ thấy như sau:



ListBox

Items là một collection of Strings

Mới dùng đến, ta sẽ thấy .NET **ListBox** rất giống ListBox trong VB6. Tiện ở chỗ bây giờ ta có thể edit các string Items của ListBox trong một editor nho nhỏ sẽ hiện ra khi ta click vào chữ **(Collection)** của **property Items**:



Các Items được chứa trong một collection tên Items, do đó ta có thể làm việc với mọi chức năng của một collection như Add, Clear, Insert, Remove, RemoveAt, Count .v.v..

Thí dụ như ta cho thêm bốn Items vào Listbox1 lúc Form_Load như sau:

```
Private Sub frmListbox_Load( ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.Load

 ' Add individual items

  ListBox1.Items.Add("Kăng-gu-ru")

  ListBox1.Items.Add("Công")

 ' Add more than one items by instantiating an object with items list enclosed in curly brackets {}

  ListBox1.Items.AddRange(New Object() {"Đà điểu", "Gấu Panda"})

End Sub
```

Nếu trong khi chạy chương trình, bạn thêm nhiều Items vào ListBox và muốn tránh update display Listbox nhiều lần, bạn có thể kẹp code giữa hai statements **BeginUpdate** và **EndUpdate** như sau:

```
' Shutdown the painting of the ListBox as items are added.
```

```
ListBox1.BeginUpdate()
' Loop through and add 50 items to the ListBox.
Dim x As Integer
For x = 1 To 50
  ListBox1.Items.Add("Item " & x.ToString())
Next x
' Allow the ListBox to repaint and display the new items.
ListBox1.EndUpdate()
```

Giống như trong VB6, **property MultiColumn** hiển thị Items trong nhiều cột nếu được set thành True, **property SelectionMode** nếu bằng **MultiExtended** thì cho ta select nhiều Items cùng một lúc.

Tuy nhiên, các Items được chọn sẽ có mặt trong một collection chớ không phải có Selected(i)=True như trong VB6.

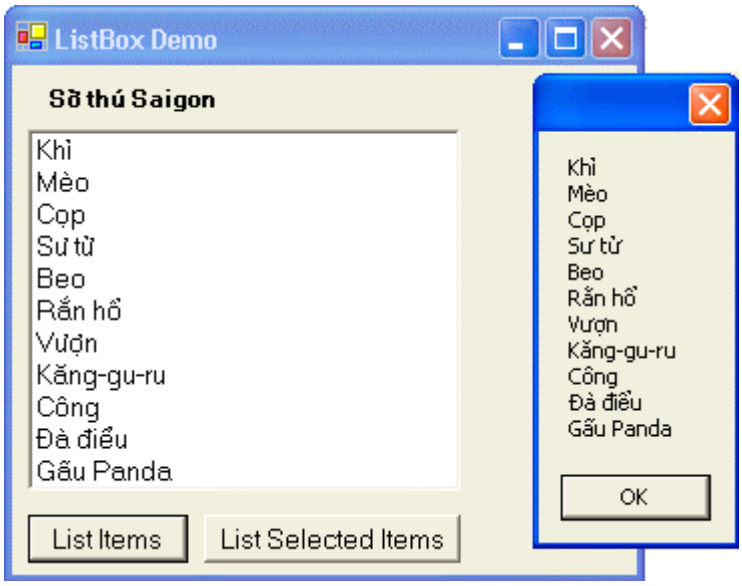
Muốn select một Item lúc run-time ta dùng code như sau:

```
' Select three items (2nd, fourth and sixth) from the ListBox.
ListBox1.SetSelected(1, True)  ' 1 is index of 2nd item
ListBox1.SetSelected(3, True)
ListBox1.SetSelected(5, True)
```

Trong thí dụ tại đây ta có ListBox1 với danh sách các con vật trong Sở Thú Saigon. Button **List Items** sẽ liệt kê danh sách này. Để ý cách ta hiển thị một Item với expression **Listbox1.Items(i).ToString**.

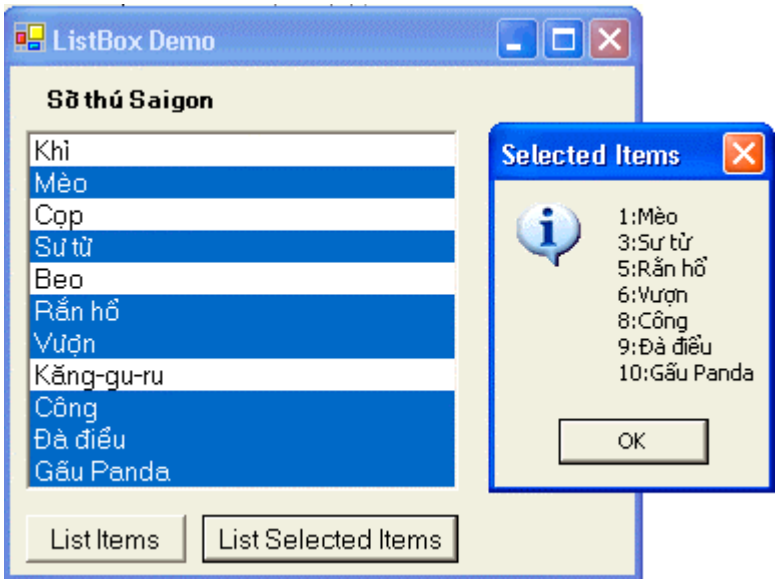
```
Private Sub BtnListItems_Click( ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles BtnListItems.Click
  Dim i As Integer
  Dim Mess As String
 ' make up the list of Items separated by CarriageReturn/LineFeed
  For i = 0 To ListBox1.Items.Count - 1
    Mess &= (ListBox1.Items(i).ToString) & vbCrLf
  Next
 ' Show the list
```

```
  MessageBox.Show(Mess)

End Sub
```



Sau khi set property SelectionMode của Listbox1 ra **MultiExtended**, code dưới đây sẽ liệt kê danh sách các items được chọn với index của chúng:

```
Private Sub BtnListSelectedItems_Click( ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles BtnListSelectedItems.Click

  Dim i As Integer

  Dim Mess As String

 ' make up the list of Selected Items separated by CarriageReturn/LineFeed

 ' Collection SelectedIndices contains the index of selecteditems

  For i = 0 To ListBox1.SelectedItems.Count - 1

    Mess &= (ListBox1.SelectedIndices(i).ToString) & ":" & (ListBox1.SelectedItems(i).ToString) &
vbCrLf

  Next

 ' Show the list

  MessageBox.Show(Mess, "Selected Items", MessageBoxButtons.OK, MessageBoxIcon.Information)

End Sub
```



Items là một Array of Objects

ListBox của .NET không hổ trợ ItemData như trong VB6. ItemData là một array chứa các con số tương ứng với những Items trong List array của ListBox trong VB6. Tức là mỗi ListBox Item trong Vb6 có thể được chỉ định trước một con số đại diện nó. Khi user select List(i), ta có thể lấy ra ItemData(i) của List Item ấy.

Thật ra Items của .NET Listbox cũng có thể là một Array of Objects, không nhất thiết phải là một collection of Strings như ta đã dùng.

Dưới đây là code ta định nghĩa một Class tên **LBItem**, đoạn dùng code thể Add một Array of Objects loại LBItem vào Listbox1:

```
Public Class LBItem
  Private mList As String
  Private mItemData As Integer
 ' List Item of Listbox
  Public Property List() As String
    Get
      Return mList
    End Get
    Set ( ByVal Value As String)
```

```
        mList = Value
      End Set
    End Property
  ' ItemData of Listbox
    Public Property ItemData() As Integer
      Get
        Return mItemData
      End Get
      Set ( ByVal Value As Integer)
        mItemData = Value
      End Set
    End Property
  ' Function to return a string representing this item for display
    Overrides Function ToString() As String
      Return mList
    End Function
End Class
```

Sau khi Add một Array of Objects vào ListBox1 ta phải chỉ định làm thế nào để hiển thị một Item. Thí dụ như dùng property List của LBItem như dưới đây:

```
' Indicate that Property List of LBItem will be used to display
ListBox1.DisplayMember = "List"
```

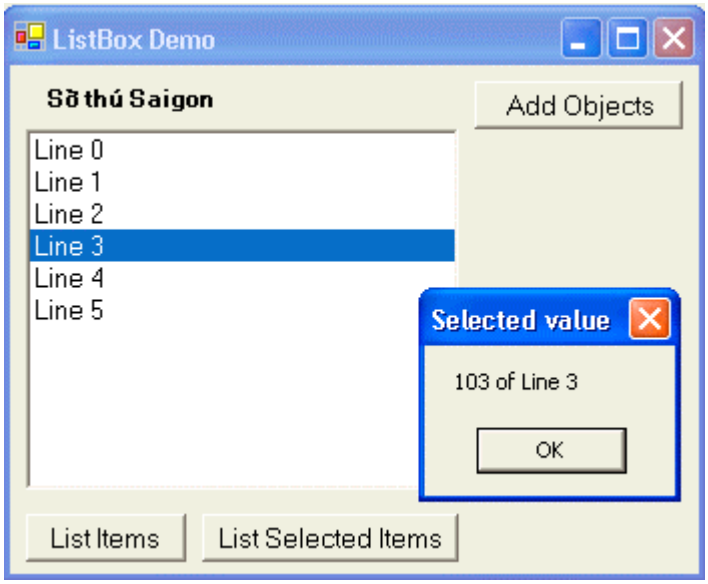
Nếu ta không chỉ định **DisplayMember**, tức là ListBox1.DisplayMember = "" thì ListBox1 sẽ dùng **Function ToString** của LBItem để hiển thị.

Ngoài ra, để trả về một value giống như ItemData của List Item ta chỉ định **ValueMember** như dưới đây:

```
Private Sub BtnAddOjects_Click( ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles BtnAddOjects.Click
  ' Clear all items in Listbox1
    ListBox1.Items.Clear()
```

```
  Dim Objs(5) As LBItem
 ' Create an array of 6 Objects of LBItem
  Dim i As Integer
  For i = 0 To 5
    Objs(i) = New LBItem()
    Objs(i).List = "Line " & i.ToString
    Objs(i).ItemData = i + 100
  Next
 ' Add the array of objects to Listbox1
  ListBox1.DataSource = Objs
 ' Indicate that Property List of LBItem will be used to display
  ListBox1.DisplayMember = "List"
 ' Indicate that Property ItemData of LBItem will be used to return a value
  ListBox1.ValueMember = "ItemData"
End Sub
```

Khi chạy chương trình này, sau khi click nút **Add Objects** để clear ListBox1 và Add 6 Objects mới, nếu bạn click hàng thứ 4 trong ListBox sẽ thấy hình dưới đây:



Code xử lý Event SelectedIndexChanged (tức là Event Click trước đây) của ListBox1 giống như dưới đây:

```
Private Sub ListBox1_SelectedIndexChanged( ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles ListBox1.SelectedIndexChanged

  Try

    If ListBox1.SelectedValue <> "" Then

      MessageBox.Show(ListBox1.SelectedValue & " of " & ListBox1.SelectedItem.ToString,
"Selected value")

    End If

  Catch ex As Exception

   ' Do nothing, ignore this error

  End Try

End Sub
```

Như thế ta đã implemented (thi hành) cho .NET ListBox một chức năng tương đương với ItemData của ListBox trong VB6.

.NET ListBox không hổ trợ Style Checkbox, nhưng ta có thể dùng **CheckedListBox**.

ComboBox

Vì ComboBox thừa kế từ ListBox nên tất cả những gì ta biết về ListBox đều áp dụng cho ComboBox. Đặc biệt bây giờ ComboBox có **property MaxDropDownItems** cho ta quyết định hiển thị bao nhiêu items khi danh sách được mở ra.

Kèm theo đây là một chương trình biểu diễn ComboBox trong đó ta dùng **Property ValueMember** của ComboBox để trả về một trị số đại diện Item. Data trong ComboBox1 được loaded từ một Access2000 database table bằng code sau đây:

```
Private Sub frmCombo_Load( ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.Load

  Dim ds As New DataSet () ' Instantiate a Dataset

 ' Instantiate an OleDbDataAdapter for Access2000 database Authors.mdb and return table Authors

  Dim myData As New OleDbDataAdapter("Select * from Authors",
"Provider=Microsoft.Jet.OLEDB.4.0;Data Source=..\Authors.mdb")

  myData.Fill(ds, "Authors") ' Load table Authors into Dataset
```

```
  With ComboBox1
    ' Bind Table Authors to ComboBox1
     .DataSource = ds.Tables("Authors")
    ' Make Property/Datafield FullName the DisplayMember of ComboBox1
     .DisplayMember = "FullName"
    ' Make Property/Datafield AuthorID the ValueMember of ComboBox1
     .ValueMember = "AuthorID"
  End With
End Sub
```

Chúng ta chỉ định record datafield **FullName** làm DisplayMember của ComboBox1 và datafield **AuthorID** làm ValueMember của ComboBox1.

Ta truy cập data của cơ sở dữ liệu bằng cách dùng một DataAdapter loại **OleDbDataAdapter** khi cho nó một SQL CommandText: **"Select * from Authors"** và một connection string, trong đó có cho biết database driver: **Microsoft.Jet.OLEDB.4.0** và tên của database **..\Authors.mdb**. File Authors.mdb nằm chung với mã nguồn của chương trình trong parent folder của folder **bin**, nơi chứa **ComboBox.exe**.

Kế đó ta dùng DataAdapter để bỏ table **Authors** vào dataset **ds**. Cách làm việc này tương tự như ADO (Active Data Object) trong VB6. Có điểm khác là Dataset có thể chứa nhiều tables (recordsets) và nó hoạt động như một cached disconnected database trong bộ nhớ. Kỹ thuật này có tên là ADO.NET và ta sẽ bàn thêm nhiều về nó trong tương lai.

Mỗi lần user select một item mới từ ComboBox1, chương trình sẽ hiển thị **AuthorId**, là ValueMember trong **Label1**.

```
Private Sub ComboBox1_SelectedIndexChanged( ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles ComboBox1.SelectedIndexChanged
  Try
    'Display the selected valueMember
     Label1.Text = ComboBox1.SelectedValue
  Catch
  End Try
```

```
End Sub
```

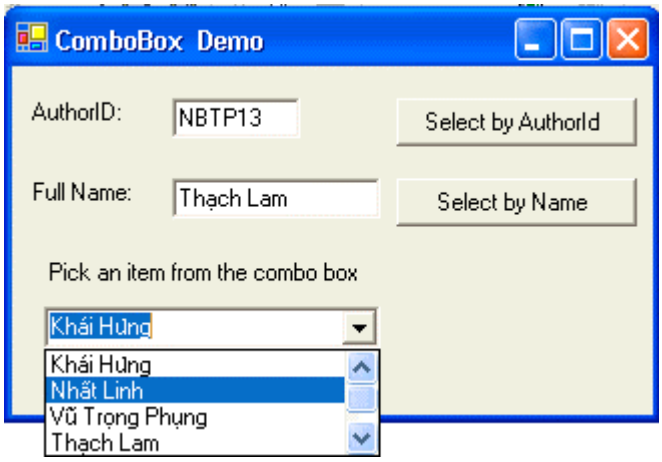
Ở đây có hai cách để ta select một ComboBox item bằng coding. Cách thứ nhất là cho biết AuthorId (ValueMember), user clicks button **Select by AuthorId** để thấy kết quả:

```
Private Sub BtnSelectbyAuthorId_Click_1( ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles BtnSelectbyAuthorId.Click

 'Use Try to ignore error if operation fails

  Try

   ' Select the ComboBox Item whose valueMember equal txtAuthorId.Text

    ComboBox1.SelectedValue = txtAuthorId.Text

  Catch

  End Try

End Sub
```

và cách thứ hai là cho biết FullName (DisplayMember), user clicks button **Select by Name** để thấy kết quả:

```
Private Sub BtnSelectByName_Click( ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles BtnSelectByName.Click

 'Use Try to ignore error if operation fails

  Try

   ' Select the ComboBox Item whose DisplayMember equal txtFullName.Text

   ' FindString returns the index of the found item

    ComboBox1.SelectedIndex = ComboBox1.FindString(txtFullName.Text)

  Catch

  End Try

End Sub
```

Khi chạy chương trình, bạn sẽ thấy hình như CÁCdưới đây. Trong hình ấy, MaxDropDownItems của ComboBox1 đã được set bằng 4.


ComboBox Demo
AuthorID:
NBTP13
Select by AuthorId
Full Name:
Thạch Lam
Select by Name
Pick an item from the combo box
Khái Hưng
Khái Hưng
Nhất Linh
Vũ Trọng Phụng
Thạch Lam