MỤC LỤC

Trang

Tiền tố một số điều khiển cơ bản

Bắt buộc đặt tên cho tất cả các điều khiển khi tham gia xử lý trong chương trình. Cách đặt tên điều khiển được quy ước với phần tiền tố như sau:

Điều khiển	Tiền tố	Ví dụ
Label	lbl	lblHelpMessage
TextBox	txt	txtLastName
Button	btn	btnExit
CheckBox	chk	chkReadOnly
CheckListBox	chklst	chklsTuyChon
RadioButton	rad	radNam
ComboBox	cbo	cboChon
List box	lst	lsPolicyCodes
PictureBox	pic	picVGA
GroupBox	grp	grpPhai
ListView	lw	lwHeadings
ProgressBar	prg	prgLoadFile
ImageList	imglst	imglsAllIcons
Image	img	imgIcon
DateTimePicker	dtp	dtpPublished
Timer	tm	tmAlarm
TreeView	tv	tvOrganization
Form	frm	frmEntry
Menu	mnu	mnuFileOpen
Dialog	dlg	dlgFileOpen
StatusBar	stt	sttDateTime
Toolbar	tb	tbActions

LỜI MỞ ĐẦU

Nhiều lập trình viên đã từng làm quen với lối lập trình truyền thống trên môi trường DOS, lập trình theo hướng lệnh, tuần tự theo chỉ định. Với lối lập trình này, các lập trình viên gặp rất nhiều khó khăn vì nó chỉ hỗ trợ đơn xử lý (nó chiến toàn bộ tài nguyên CPU) và phải sử dụng các thư viện Multimedia riêng biệt, đặc biệt là phần giao diện thiết kế không hấp dẫn với người sử dụng. Tuy nhiên, với các phương pháp lập trình trên đều đòi hỏi lập trình viên phải nhớ rất nhiều câu lệnh với mỗi lệnh có một cú pháp và tác dụng riêng, khi viết chương trình phải tự kết nối các lệnh để có một chương trình giải quyết từng bài toán riêng biệt.

Trong xu hướng tin học phát triển mạnh mẽ như hiện nay và số người sử dụng máy tính tăng lên rất nhanh. Máy tính được sử dụng trong hầu hết các lĩnh vực của đời sống nên đòi hỏi các ứng dụng cũng như ngôn ngữ lập trình phải đơn giản, dễ sử dụng và mang tính đại chúng cao. Chính vì vậy ngôn ngữ lập trình theo hướng sự kiện ra đời. Đặc điểm của các ngôn ngữ lập trình theo hướng sự kiện là dễ sử dụng, dễ triển khai các ứng dụng một cách nhanh chóng. Từ đó phương pháp lập trình Windows Form ra đời và được phát triển trên nền tảng hệ điều hành Windows.

Đặc điểm nổi bật của phương pháp lập trình Windwos Form là :

- ✓ Cho phép xây dựng chương trình theo hướng sự kiện(Event-Driven Programming), nghĩa là một chương trình ứng dụng được viết theo kiểu này đáp ứng dựa theo tình huống xảy ra lúc thực hiện chương trình. Tình huống này bao gồm người sử dụng ấn một phím tương ứng, chọn lựa một nút lệnh hoặc gọi một lệnh từ một ứng dụng khác chạy song song cùng lúc.

- ✓ Người lập trình trực tiếp tạo ra các khung giao diện (interface), ứng dụng thông qua các thao tác trên màn hình dựa vào các đối tượng (ojbect) như hộp hội thoại hoặc nút điều khiển (control button), những đối tượng này mang các thuộc tính (properties) riêng biệt như: màu sắc, Font chữ… mà ta chỉ cần chọn lựa trên một danh sách cho sẵn.

- ✓ Khi dùng ngôn ngữ để lập trình Windows Form người lập trình rất ít khi phải tự viết các lệnh, tổ chức chương trình... một cách rắc rối mà chỉ cần khai báo việc gì cần làm gì khi một tình huống xuất hiện.

- ✓ Máy tính sẽ dựa vào phần thiết kế và khai báo của lập trình viên để tự động tạo lập code chương trình.

Như vậy với kỹ thuật lập trình Windows Form, lập trình viên giống như một nhà thiết kế, tổ chức để tạo ra các biểu mẫu, đề nghị các công việc cần thực hiện và máy tính sẽ

dựa vào đó để xây dựng chương trình. Hiện nay các ngôn ngữ lập trình, hệ quản trị cơ sở dữ liệu theo hướng này: Visual Basic, Visual Foxpro, Visual C, Delphi, C Sharp...

Với những thuận lợi như đã nêu trên, tôi cũng xin phép được biên soạn quyển giáo trình này nhằm giúp các sinh viên nắm bắt được những kỹ thuật và ngôn ngữ lập trình theo hướng sự kiện trên môi trường hệ điều hành Windows, cụ thể là sử dụng bộ phần mềm Visual Studio.Net với ngôn ngữ C# để có thể tạo ra các sản phẩm ưng ý và đỡ vất vả hơn.

Mặc dù rất cố gắng trong việc biên soạn cuốn giáo trình này, nhưng do đây là bộ phần mềm mới, ngôn ngữ và kỹ thuật mới, nên trong phạm vi giáo trình này chỉ đề cập đến những kỹ thuật còn rất khiêm tốn, mong các đồng nghiệp và sinh viên góp ý để tôi có thể hoàn thiện hơn ở những lần biên soạn và hiệu chỉnh giáo trình kế tiếp.

Tôi cũng chân thành cảm ơn các thầy Dương Quang Thiện, Ngô Phương Lan, Nguyễn Anh Tài, Nguyễn Hữu Khang, … đã biên soạn các tài liệu rất hữu ích mà tôi đã tham khảo để biên soạn giáo trình này.

Xin chân thành cảm ơn!

Chương 1 : GIỚI THIỆU WINDOWS FORMS

1.1. Cài đặt và sử dụng Visual Studio .NET

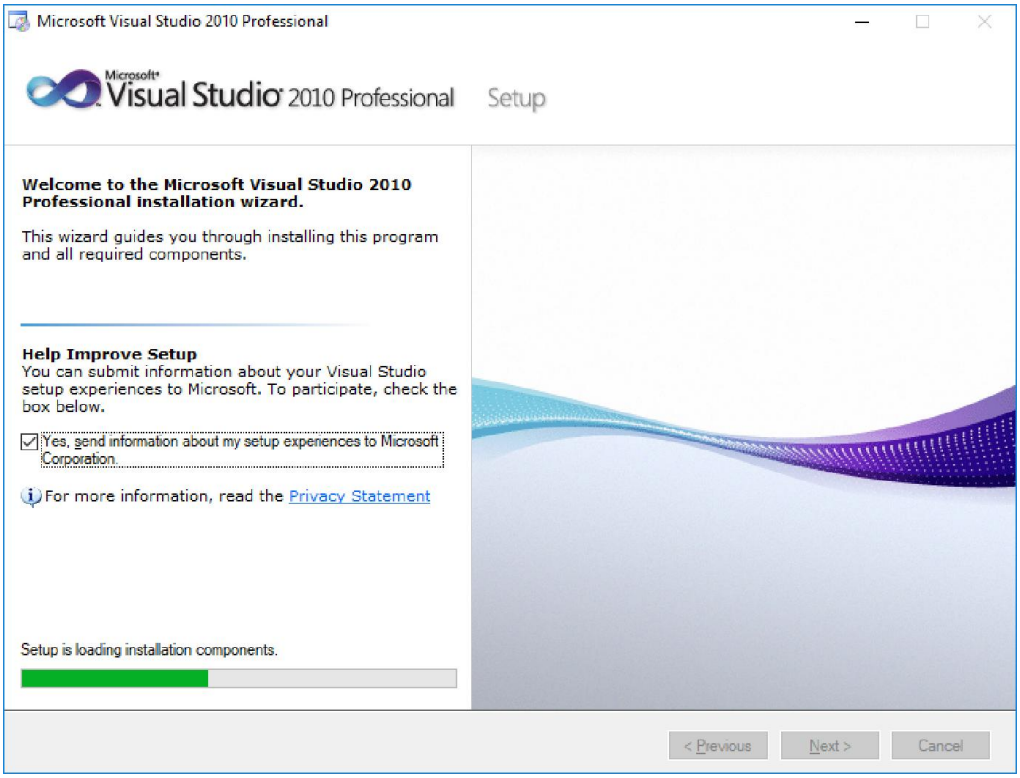
1.1.1.Yêu cầu hệ thống

- Bộ vi xử lý CPU 1,6 GHz hoặc nhanh hơn.
- RAM: 1 GB đối với x86, 2 GB RAM đối với x64; thêm 512 MB nếu chạy trong máy ảo.
- Ổ cứng trống 3GB
- Ổ đĩa cứng: 5400 RPM
- Card video DirectX 9 với độ phân giải màn hình hiển thị 1024 x 768 hoặc cao hơn
- Ổ DVD-ROM

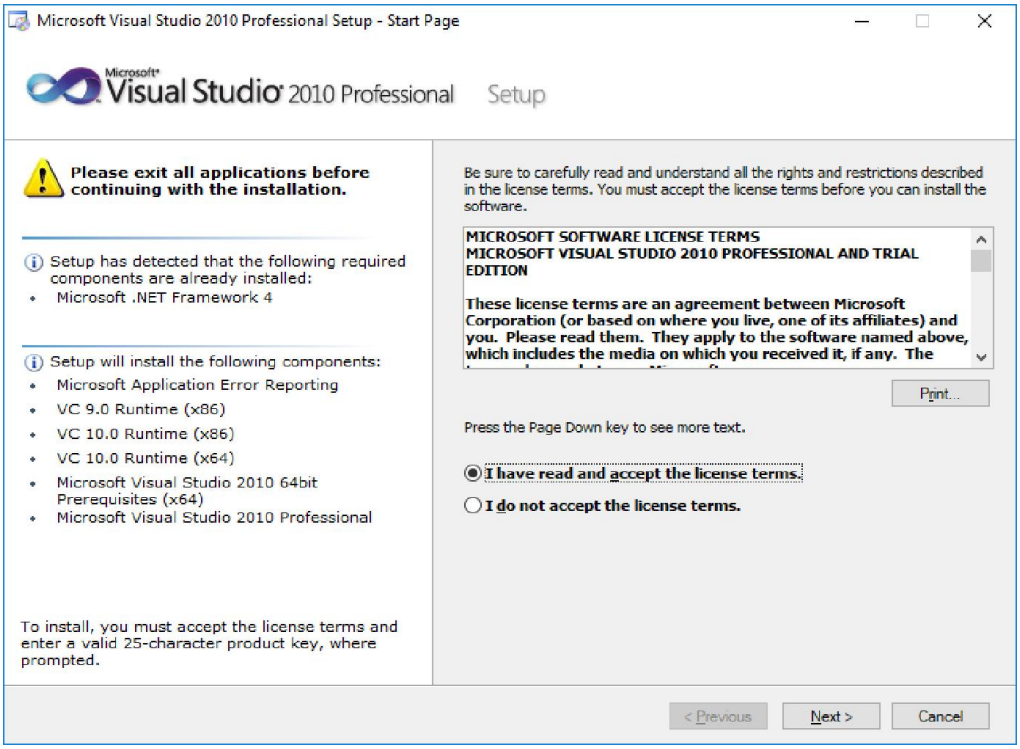
1.1.2. Cài đặt

Bước 1: Click đúp vào file setup trong thư mục chứa bộ cài đặt (trong ổ cứng hoặc đĩa DVD chương trình). Màn hình lựa chọn cài đặt hiển thị, chọn Install Microsoft Visual Studio 2010.

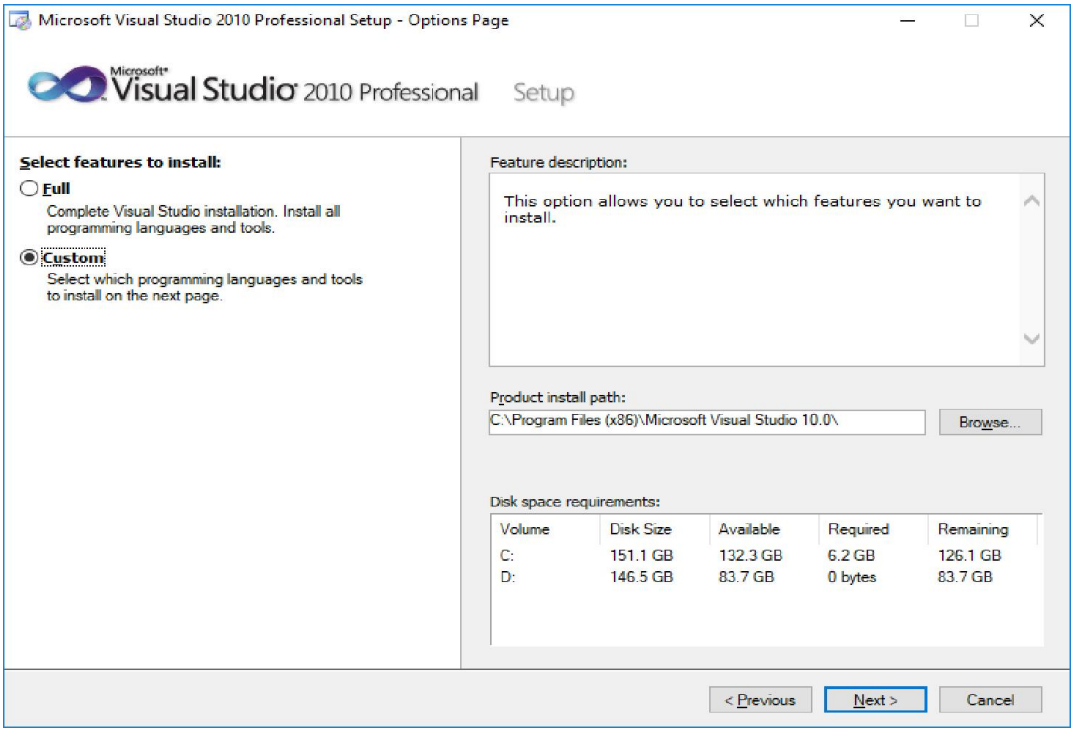
Bước 2: Chương trình sẽ giải nén bộ cài đặt để sẵn sàng cho việc cài đặt. Bạn sẽ nhìn thấy màn hình lựa chọn như bên dưới. Nhấn Next để tiếp tục



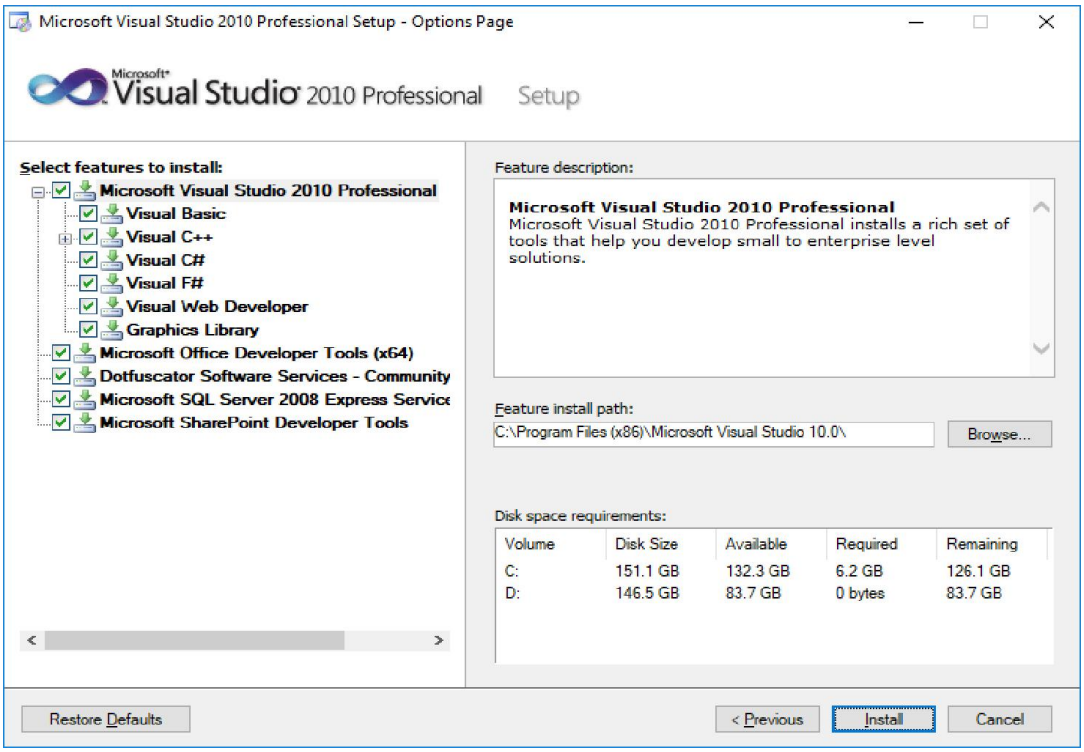
Bước 3: Chọn I have read and accept the license terms và nhấn Next



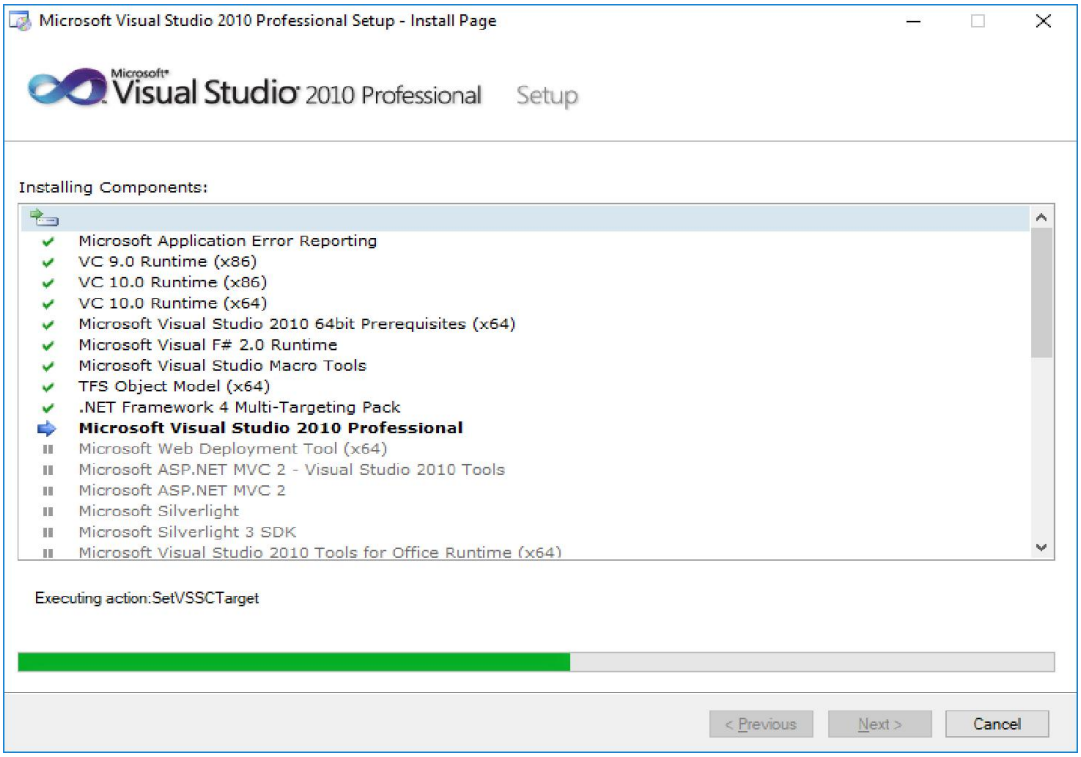
Bước 4: Chọn Custom, sau đó chọn Next để tiếp tục.



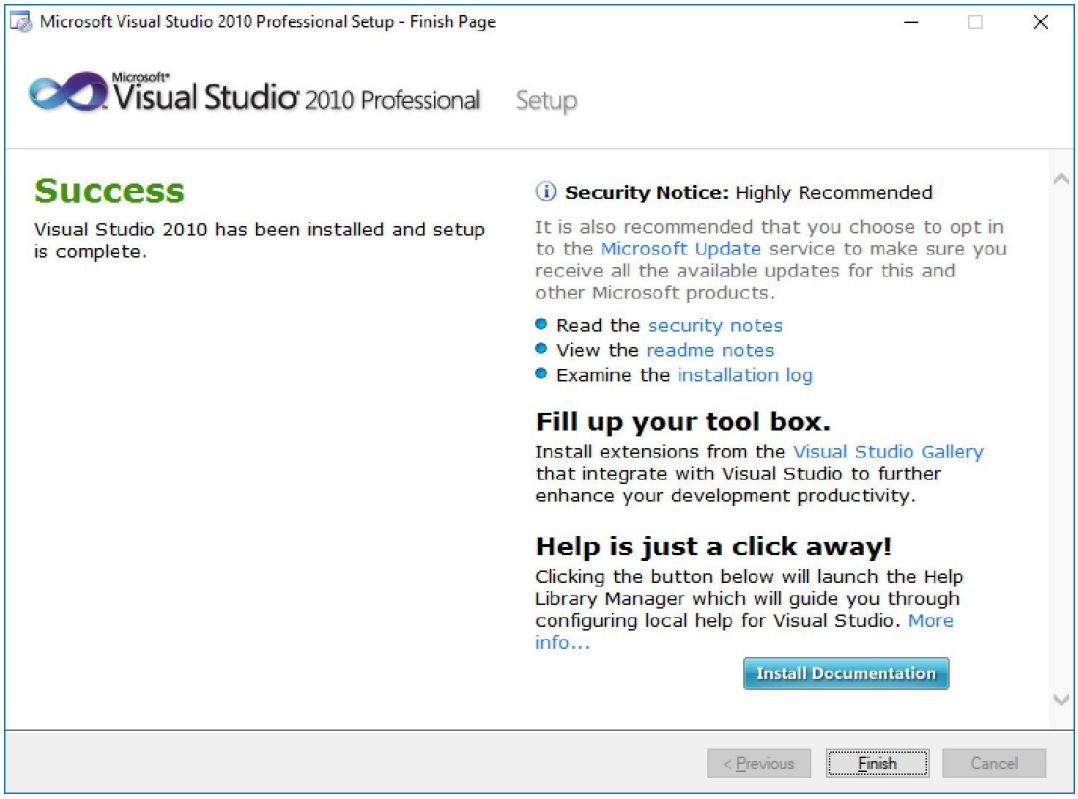
Bước 5: Lựa chọn ngôn ngữ cài đặt bằng cách chọn các ô checkbox bên trái. Sau đó chọn nơi sẽ cài đặt vào, mặc định ở ổ C:\Program Files\……Chọn Install để sang bước tiếp theo.



Bước 6: Chờ quá trình cài đặt diễn ra



Bước 7: Nhấn Finish để hoàn tất quá trình cài đặt



1.2. Tổng quan .Net Framework

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

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

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

1.3. Cấu trúc .Net Framework

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

- CLR là thành phần chính của .NET Framework, quản lý mã (code) có thể thực thi của chương trình, quản lý các tiến trình, quản lý tiểu trình (Threading), quản lý bộ nhớ, cung cấp dịch vụ để biên dịch, tích hợp và tác vụ truy cập từ xa (Remoting).
- FCL bao gồm tất cả các dịch vụ như giao tiếp người sử dụng, điều khiển, truy cập dữ liệu, XML, Threading, bảo mật.

Tóm lại, CLR được xem như máy ảo .NET (.NET Virtual Machine), nó có thể kiểm soát, nạp và thực thi chương trình .NET. Trong khi đó, FCL cung cấp các lớp, giao tiếp và các kiểu giá trị, phương thức truy cập và chức năng chính của hệ thống như: Microsoft.Csharp, Microsoft.Jscript, Microsoft.VisualBasic, Microsoft.Vsa, Microsoft.Win32, System (cùng với các không gian tên con của không gian tên System).

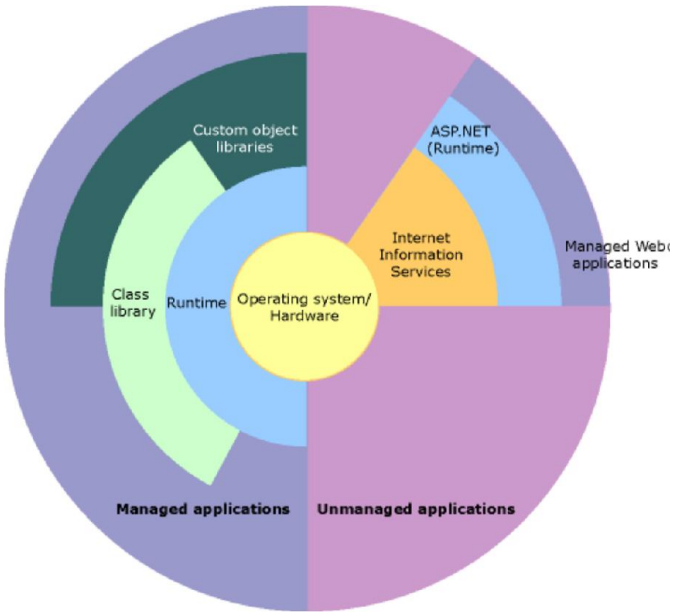
- Microsoft.Csharp : cung cấp các lớp hỗ trợ biên dịch và phát sinh mã khi sử dụng ngôn ngữ lập trình C#.
- Microsoft.Jscript : cung cấp các lớp hỗ trợ biên dịch và phát sinh mã khi sử dụng ngôn ngữ lập trình J#.
- Microsoft.VisualBasic : cung cấp các lớp hỗ trợ biên dịch và phát sinh mã khi sử dụng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

- Microsoft.Vsa : cung cấp các gia tiếp cho phép tích hợp với các kịch bản của .NET Framework vào ứng dụng khi biên dịch hay thực thi.
- Microsoft.Win32: cung cấp hai lớp giao tiếp trực tiếp với tài nguyên của hệ điều hành và System Registry.
- System: bao gồm các lớp cơ sở dùng để định nghĩa giá trị, tham chiếu, biên cố, giao tiếp, thuộc tính và kiểm soát ngoại lệ. Ngoài ra, một số lớp khác cung cấp các dịch vụ chuyển đổi kiểu dữ liệu, tham số, tính toán, xử lý và truy cập từ xa.

Trong đó, Code bao gồm hai loại :

- Manage Code: bao gồm những chương trình được tạo ra từ các ngôn ngữ lập trình có hỗ trợ .NET, chẳng hạn, sử dụng ngôn ngữ lập trình C# để phát triển chương trình ứng dụng A, sau đó biên dịch chúng ra tập tin thi hành (.EXE), tập tin .EXE này đuợc gọi là Manage Code trong môi trường .NET.
- Unmanage Code : là những chương trình được tạo ra từ các ngôn ngữ lập trình ngoài .NET. Ví dụ, sử dụng ngôn ngữ lập trình Visual Basic 6.0 để khai báo lớp (Class) có tên là B, rồi biên dịch chúng ra tập tin thư viên (.DLL), tập tin .DLL được gọi là Unmanage Code khi tham chiếu chúng trong môi trường .NET

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



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

Những đặc điểm chính của .NET Framework

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

- **CLR :** CLR là môi trường thi hành, nơi cung cấp dịch vụ để thực thi, quản lý bộ nhớ, tiểu trình cho các ứng dụng hỗ trợ bởi .NET
    - Quản lý quá trình thực thi: để quản lý quá trình thực thi của trình, CLR thực hiện qua các bước sau: chọn chương trình biên dịch tương ứng với ngôn ngữ lập trình, biên dịch ứng dụng sang tập tin MSIL (trình bày chi tiết trong phần biên dịch và thực thi ứng dụng), biên dịch từ mã định dạng MSIL sang mã máy bằng trình JIT (Just-In-Time) rồi sau đó CLR cung cấp cơ sở hạ tầng để thi hành chương trình.
    - Quản lý bộ nhớ: tự quản lý bộ nhớ là một trong những dịch vụ mà CLR cung cấp trong quá trình thực thi chương trình. Trình thu gom (Garbage Collector) quản lý bộ nhớ đã cấp cho một tiến trình rồi sau đó tự động thu lại khi chương trình kết thúc (chúng ta sẽ tìm hiểu chi tiết về Garbage Collector trong cuốn sách " lập trình hướng đối tượng" sắp phát hành).
- **FCL:** Bao gồm các thư viện lớp cơ sở cho phép bạn sử dụng để thực hiện mọi tác vụ liên quan đến giao diện, Internet, cơ sở dữ liệu, hệ điều hành,…
- **Common Type System (CTS):** CTS đưa ra các quy tắc cho phép bạn khai báo, sử dụng và quản lý kiểu dữ liệu trong quá trình thi hành. Ngoài ra, CTS còn cung cấp các tiêu chuẩn cho phép phát hành tương tác giữa các ngôn ngữ lập trình với nhau. Tóm lại, CTS thực hiện các chức năng chính sau:
    - Thiết lập khung cho phép tương tác giữa các ngôn ngữ, mã an toàn(safe code), tối ưu hóa xử lý.
    - Cung cấp mô hình hướng đối tượng nhằm hỗ trợ quá trình cài đặt đa ngôn ngữ trong ứng dụng.
    - Định nghĩa các quy tắc mà ngôn ngữ lập trình phải tuân theo và hỗ trợ tính chuyển đổi và bảo đảm đối tượng được tạo ra từ ngôn ngữ này có thể tương tác với ngôn ngữ khác.
- **Metadata and Self-Describnbing Components (MSDC):** trong những phiên bản trước đây, ứng dụng được tạo ra bởi một ngôn ngữ lập trình nào đó được biên dịch ra tập tin .EXE hay .DLL và khó khăn khi sử dụng chúng với một ứng dụng được viết trong một ngôn ngữ lập trình khác. Ví dụ COM là một điển hình. Tuy nhiên, .NET framework cung cấp giải pháp chuyển đổi cho phép khai báo thông tin cho mọi module và Assembly (có thể là .EXE hay .DLL). Những thông tin này được gọi là siêu dữ liệu và sự mô tả.
- **Cross Language Interoperability (CLI):** CLR là hỗ trợ tiến trình trao đổi và sử dụng giữa các ngôn ngữ với nhau.Tuy nhiên, hỗ trợ này không bảo đảm mã do bạn viết có thể dùng được bởi lập trình viên sử dụng ngôn ngữ lập trình khác.
- **Assemblies:** là tập hợp các kiểu dữ liệu và tài nguyên được đóng gói dạng từng đơn vị chức năng. Ngoài ra, assemblies chính là các đơn vị chủ yếu dùng để triển khai, điều khiển phiên bản, thành phần sử dụng lại, chẳng hạn như các tập tin .EXE hay .DLL.

- **Applicatin Domains:** miền ứng dụng cho CLR quản lý nhằm cách ly nhiều ứng dụng đang thi hành trên cùng một máy tính cụ thể:
    - Mỗi ứng dụng sẽ được nạp vào tiến trình(Process) tách biệt mà không ảnh hưởng đến ứng dụng khác. Với kỹ thuật kiểu mã an toàn Application Domains bảo đảm đoạn mã đang chạy trong miền ứng dụng độc lập với tiến trình của ứng dụng khác trên cùng một máy.
    - Khi tạm dừng từng thành phần thì sẽ không dừng toàn bộ tiến trình. Đối với trường hợp này Application Domains cho phép bạn loại bỏ đoạn mã đang chạy trong ứng dụng đơn.
    - Application Domains cho phép bạn cấu hình, định vị, cấp quyền hay hạn chế quyền sử dụng tài nguyên đang thi hành.
    - Ngoài ra, sự cách ly này cho phép CLR ngăn cấm truy cậptruwcj tiếp giữa các đối tượng của những ứng dụng khác nhau.
- **Runtime Hosts:** là trung tâm thi hành cho phép nạp ứng dụng vào tiến trình, CLR hỗ trợ cho phép nhiều loại ứng dụng khác nhau cùng chạy trong một tiến trình.
    - Mỗi loại ứng dụng thì cần đoạn mã để khởi động được gọi là Runtime hosts.
    - Runtime Hosts nạp kênh thi hành vào tiến trình và tạo ra Application Domains rồi thi hành ứng dụng vào trong miền ứng dụng đó.

1.4. Tạo ứng dụng đầu tiên

1.4.1. Các bước cơ bản viết chương trình:

Để viết một chương trình, ta thực hiện các bước cơ bản sau.

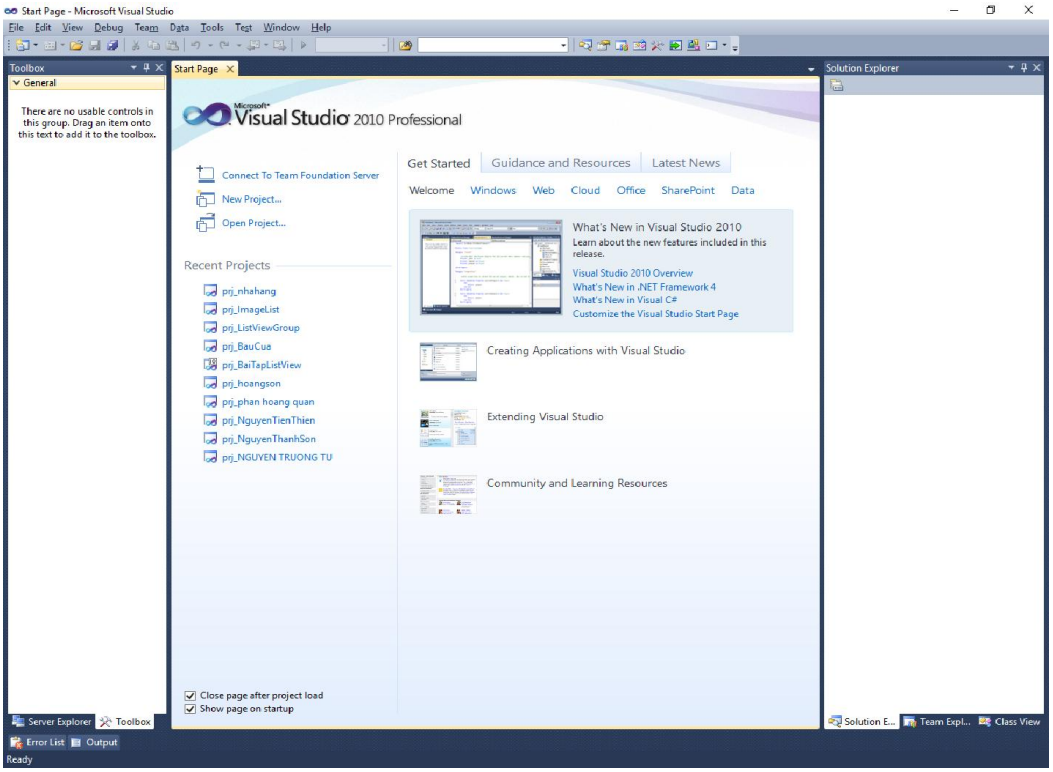
1. Thu thập thông tin theo yêu cầu của ứng dụng.
2. Thiết kế giao diện cho phù hợp với yêu cầu.
3. Cài đặt các thuộc tính điều khiển thiết kế trên giao diện.
4. Dự đoán các ràng buộc(tình huống/sự kiện) dữ liệu nhập xuất trên giao diện.
5. Xây dựng các hàm xử lý tình huống, sự kiện.
6. Biên dịch, chạy thử.
7. Xử lý các tình huống gây lỗi trong chương trình (bẫy lỗi, giải quyết sự cố).
8. Cải tiến, nâng cấp để hoàn chỉnh chương trình.

1.4.2. Tạo Project mẫu:

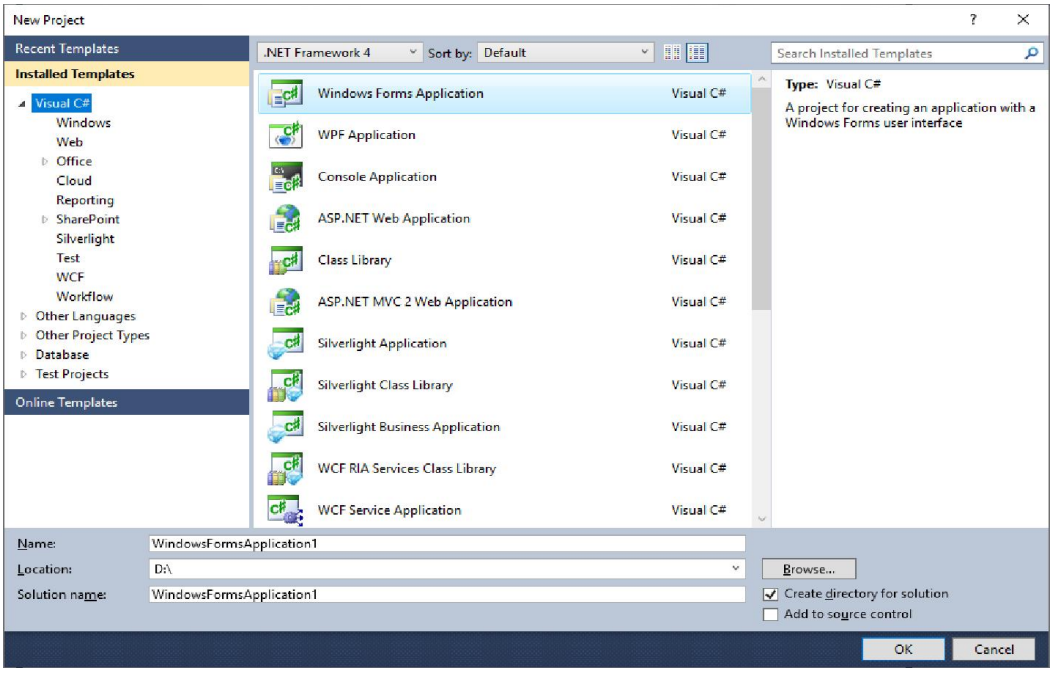
Tạo một ứng dụng có tên "**Hello**".

Ý nghĩa: Khi người dùng nhập vào họ tên và xác định giới tính và nhấn nút "Chào" thì sẽ xuất hiện lời chào theo họ tên dựa vào giới tính, nếu là nam thì lời chào là "Ông xxx" ngược lại, lời chào là "Bà xxx". Nếu chưa nhập họ và tên thì xuất thông báo nhắc nhở nhập họ và tên.

Khởi chạy chương trình Visual studio 2010 từ trình đơn Start xuất hiện màn hình giao diện sau



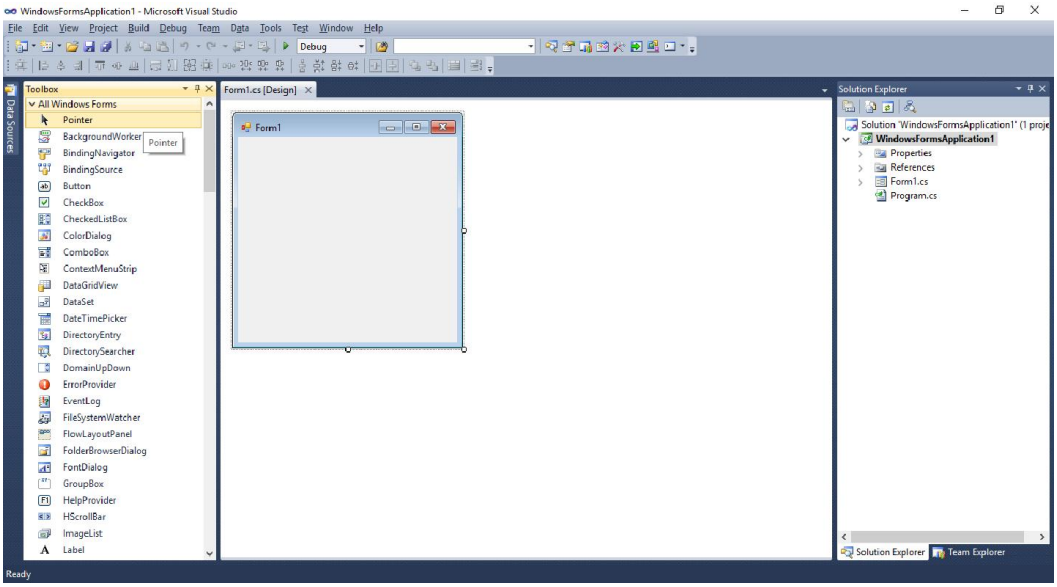
Từ giao diện chính chương trình, ta vào File/New/Project xuất hiện hộp thoại



- Trên hộp thoại chọn Visual C# ở cửa sổ bên trái
- Ở cửa sổ phải chọn Windows Form Application

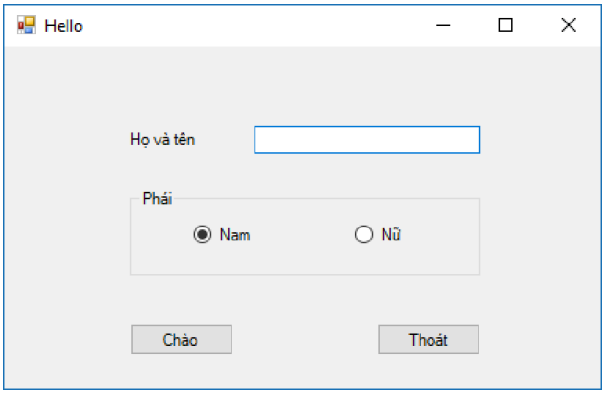
- Ở mục Name: nhập tên project chương trình ứng dụng cần xây dựng
- Ở mục Location: chọn đường dẫn tạo tên project ứng dụng ở mục Name
- Chọn Ok.

Sau khi một Project được khởi tạo theo mẫu ứng dụng Windows Form, thì một màn hình được xuất hiện được gọi là IDE Main Windows (cửa sổ môi trường phát triển giao diện), bao gồm một Form mới có tên mặc định là Form1



Lúc này mới bắt tay thiết kế giao diện chương trình. Theo phân tích ví dụ ở trên.

Thiết kế giao diện chương trình như sau:



Gồm có:

1. Một TextBox cho phép nhập họ tên đối tượng dữ liệu.
2. Một nhóm chức năng (Group box) gồm hai nút chức năng (Radio Button) để xác định giới tính của đối tượng dữ liệu.
3. Một nút lệnh (Button) để thực hiện lời chào nếu là một đối tượng dữ liệu hợp lệ.
4. Một nút lệnh (Button) khác dùng để kết thúc ứng dụng.

Thiết đặt các Properties:

Đặt tên cho các đối tượng như sau:

Điều khiển	Thuộc tính
Với TextBox:	Name: txtHoTen cho TextBox. MaxLength: 100
Với Groupbox:	Name: grpGioiTinh
Với nút chức năng thứ nhất:	Name: radNam. Checked: True (mặc định giới tính là "Nam").
Với nút chức năng thứ hai:	Name: radNu. Checked: False.
Với Button Chào	Name: btnChao. Text: Chào
Với Button Thoát	Name: btnThoat. Text: Thoát

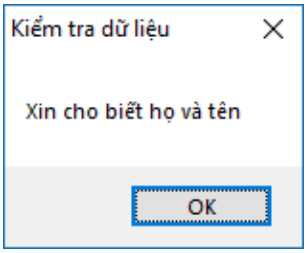
Run Project và nâng cấp Project:

Sau khi đặt tên cho các đối tượng trên giao diện, nhấn phím F5 (hay nhấn nút  hoặc thự hiện lệnh Debug/ Start Debudding) để chạy thử ứng dụng.

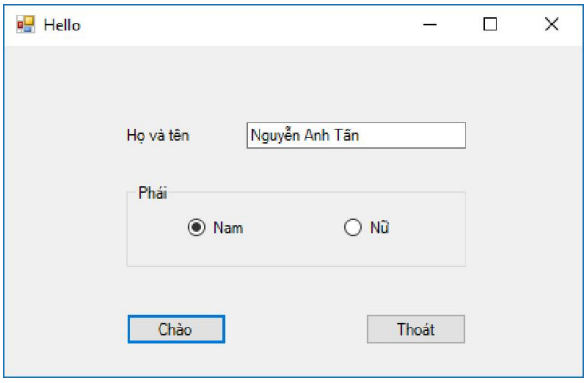
Sau đó, để nâng cấp Project, viết mã lệnh (Code) xử lý cho tình huống (sự kiện) cho nút chào như sau:

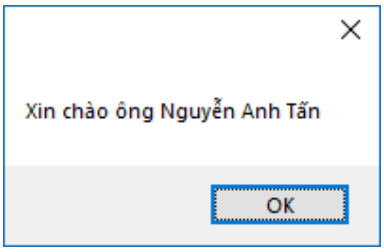
```
private void btnChao_Click(object sender, EventArgs e) {
    if (txtHoTen.TextLenght == 0)
            MessageBox.Show("Xin cho biết họ và tên", "Kiểm tra dữ liệu");
    else {
            if (radNam.Checked == true)
                    MessageBox.Show("Xin chào ông " + txtHoTen.Text);
            else     MessageBox.Show("Xin chào bà " + txtHoTen.Text);
            }
}
```

Ý nghĩa: Nếu chưa nhập họ và tên, sau đó nhấn nút "**Chào**" sẽ xuất hiện thông báo như sau:

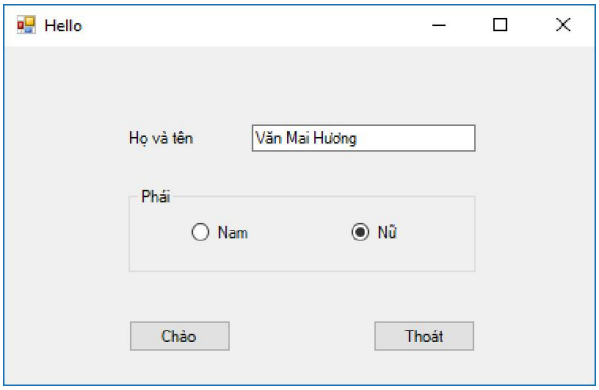


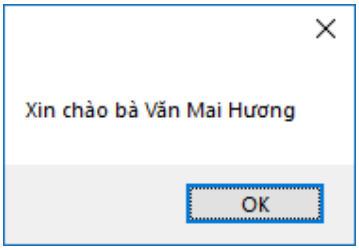
Nếu nhập đầy đủ họ và tên và giới tính được chọn là **Nam**, sau đó nhấn nút "**Chào**" sẽ xuất hiện thông báo:





Nếu nhập đầy đủ họ và tên và giới tính được chọn là **Nữ**, sau đó nhấn nút "**Chào**" sẽ xuất hiện thông báo:





Nếu nhấn nút "**Thoát**" sẽ kết thúc chương trình theo lệnh sau:

```
private void btnThoat_Click(object sender, EventArgs e){
        Application.Exit();
}
```

Trở về màn hình thiết kế, Click chọn thanh tiêu đề của Form và quy định các Properties như sau:

AcceptButton: **btnChao** (để khi nhấn Enter sẽ thực hiện mã lệnh đã quy định trong tình huống (sự kiện) **btnChao_Click).**

CancelButton: btnThoat(để khi nhấn nút ESC sẽ thực hiện mã lệnh đã quy định trong tình huống(sự kiện) **btnThoat_Click**).

1.5. Giới thiệu Windows Forms

Windows Forms (WinForms) là thư viện lớp đồ họa (GUI) được bao gồm như một phần của Microsoft .NET Framework hoặc Mono Framework, cung cấp nền tảng để viết các ứng dụng phong phú cho máy tính để bàn, máy tính xách tay và máy tính bảng.

Windows Forms cung cấp quyền truy cập vào Giao diện người dùng Windows gốc Điều khiển chung bằng cách gói API Windows hiện có trong mã được quản lý. Với sự trợ giúp của Windows Forms, .NET Framework cung cấp sự trừu tượng hóa toàn diện hơn API Win32 so với Visual Basic hoặc MFC đã làm.

Windows Forms tương tự như thư viện Microsoft Foundation Class (MFC) trong việc phát triển các ứng dụng khách. Nó cung cấp một trình bao bọc bao gồm một tập hợp các lớp C ++ để phát triển các ứng dụng Windows. Tuy nhiên, nó không cung cấp khung ứng dụng mặc định như MFC. Mọi điều khiển trong ứng dụng Windows Forms là một thể hiện cụ thể của một lớp.

1.6. Ứng dụng Windows Forms

Ứng dụng Windows Forms là một ứng dụng hướng sự kiện được hỗ trợ bởi .NET Framework của Microsoft. Không giống như một chương trình hàng loạt, nó dành phần lớn thời gian của nó chỉ đơn giản là chờ người dùng làm gì đó, chẳng hạn như điền vào hộp văn bản hoặc nhấp vào nút.

Với những ưu việt của mỗi loại ứng dụng trên nền Windows, ta có thể triển khai ứng dụng quản lý trong mạng cục bộ bằng ứng dụng web Form hay ứng dụng internet bằng Windows Forms.

Tuy nhiên, việc đi đến quyết định chọn loại ứng dụng nào để xây dựng ứng dụng phục vụ mục đích quản lý việc kinh doanh của mình dựa trên nhiều cơ sở khoa học và các yếu tố khách quan hay chủ quan.

Vậy Windows Forms sử dụng cho mọi loại ứng dụng phục vụ quản lý. Thông thường sử dụng nhiều nhất: quản lý tài chính, nhân sự, sản xuất, quản lý doanh nghiệp,…

1.7. Không gian tên(namespace)

Namespace trong C# là kỹ thuật phân hoạch không gian các định danh, các kiểu dữ liệu thành những vùng dễ quản lý hơn, nhằm tránh sự xung đột giữa việc sử dụng các thư viện khác nhau từ các nhà cung cấp.

Ví dụ khi bạn tạo một lớp trong một namespace nào đó, bạn không cần phải kiểm tra xem nó có bị trùng tên với một lớp nào đó trong namespace khác không. Việc sử dụng namespace trong khi lập trình là một thói quen tốt, bởi vì công việc này chính là cách lưu các mã nguồn để sử dụng về sau. Ngoài thư viện namespace do Microsoft .NET và các hãng thứ ba cung cấp, ta có thể tạo riêng cho mình các namespace.

Định nghĩa namespace. Để tạo một namespace sử dụng cú pháp sau:

```
namespace namespace {
    //Định nghĩa lớp A, lớp B…
    .....
}
```

Ví dụ:

```
using System;
namespace MyLib{
    public class Tester{
        public static void Main() {
            for (int i =0; i < 10; i++)
                Console.WriteLine( “i: {0}”, i);
        }
    }
}
```

Có thể định nghĩa nhiều namespace lồng nhau như sau:

```
using System;
namespace MyLib{
    namespace Demo{
        public class Tester{
            public static void Main() {
                for (int i =0; i < 10; i++)
                    Console.WriteLine( “i: {0}”, i);
```

```
                    }
                }
            }
        }
```

Sử dụng namespace

C# đưa ra từ khóa **using** để khai báo sử dụng các định danh, kiểu dữ liệu định nghĩa từ namespace trong chương trình:

using *namespace*;

Thay vì sử dụng lệnh using, có thể sử dụng dấu chấm truy cập namespace. Ví dụ để truy cập lớp Tester trong ví dụ trên dùng cú pháp sau:

MyLib.Demo.Tester

Ví dụ:

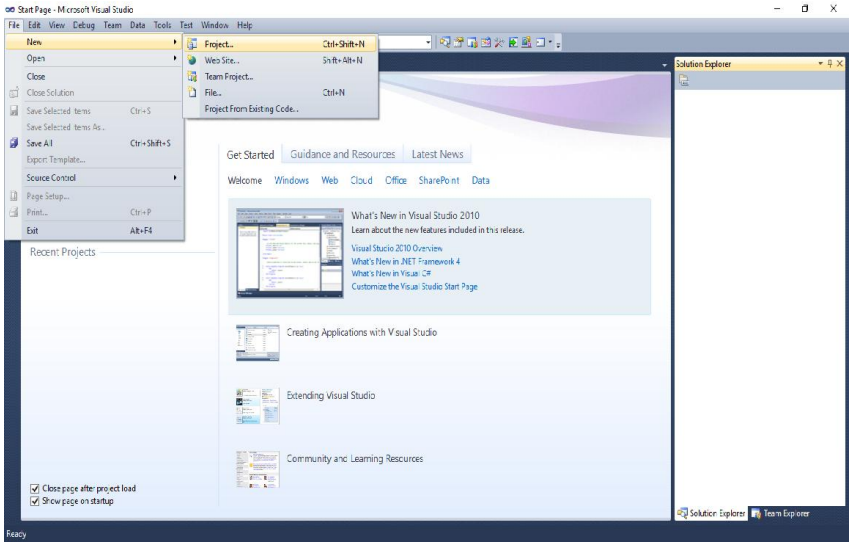
```
using System;
namespace MyLib{
    namespace Demo1{
        class Example1{
            public static void Show1() {
               Console.WriteLine("Lop Example1");
            }
        }
    }
    namespace Demo2{
        public class Tester{
            public static void Main() {
                Demo1.Example1.Show1();
                Demo1.Example2.Show2();
            }
        }
    }
}
```

Lớp Example2 có cùng namespace MyLib.Demo1 với lớp Example1 nhưng hai khai báo không cùng một tập tin

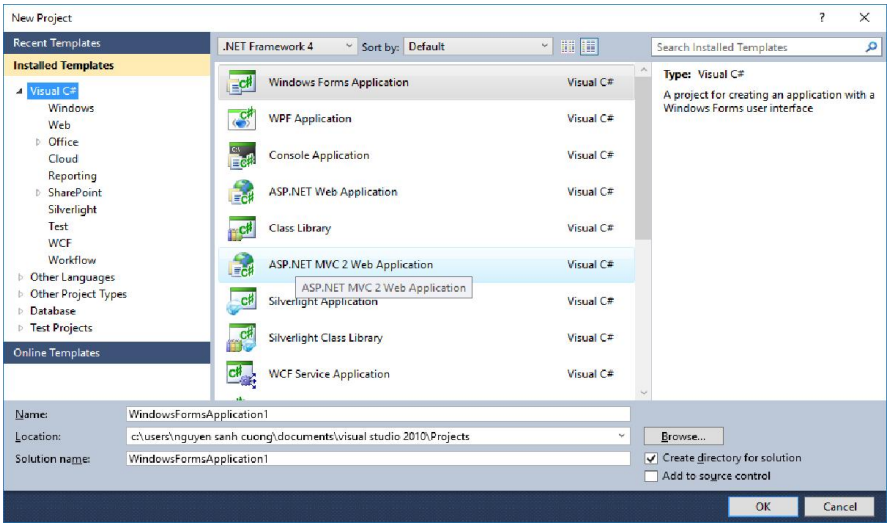
```
namespace MyLib.Demo1{
        class Example2{
                public static void Show2(){
                        Console.WriteLine("Lop Example2");
                }
        }
}
```

1.8. Trình đơn Project

Từ nàm hình giao diện chính của chương trình, ta vào File/New/Project..



Xuất hiệp hộp thoại New Project có hình như sau



Khi tạo mới một project theo một trong các cách như trên thì sẽ xuất hiện một hộp thoại được gọi là New Project Dialog, cho phép ta lựa chọn ngôn ngữ các mẫu cho ứng dụng.

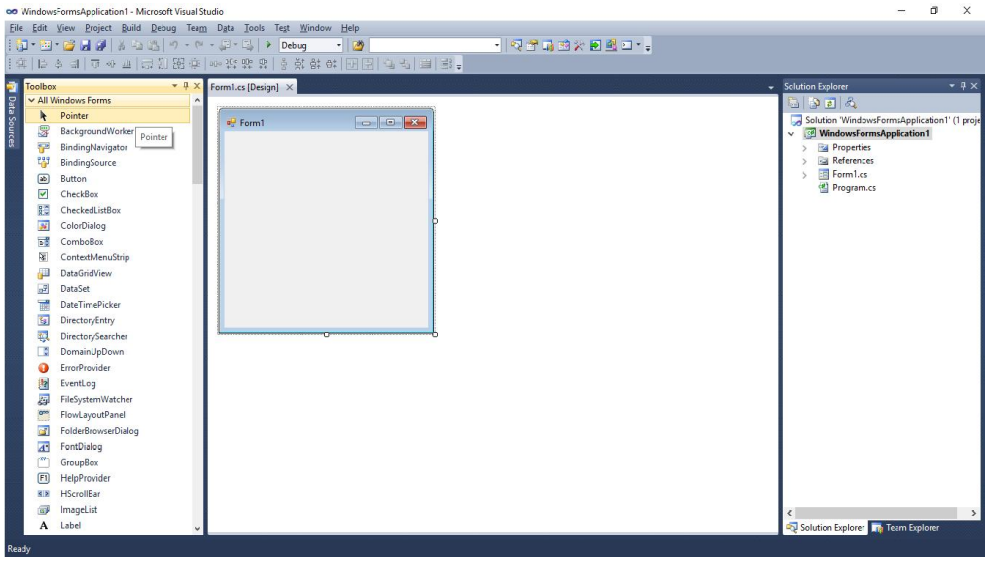
Trong đó ta có các ngôn ngữ sau:

1. Visual Basic (.Net)
2. C# (C Sharp)
3. Visual J#.
4. Visual C++.

Và các mẫu ứng dụng như sau:

1. Windows Application: ứng dụng Windows Form (giao diện cửa sổ).
2. Windows Control Library: Ứng dụng Windows Control Library mẫu để tạo ra các điều khiển tùy chỉnh để sử dụng trên Windows Forms.
3. Console Application: ứng dụng lập trình hướng lệnh (thực thi tuần tự các lệnh) và chạy trên môi trường DOS.
4. Empty Project: tạo một dự án rỗng dùng để nhúng những dự án có sẵn.
5. Class Library: mẫu tạo lớp đối tượng.
6. Web Control Library: ứng dụng mẫu để tạo các điều khiển sử dụng trên ứng dụng Web Form.
7. Windows Service: mẫu tạo các trình phục vụ cho Windows.
8. Crystal Reports Application: mẫu tạo các báo biểu.

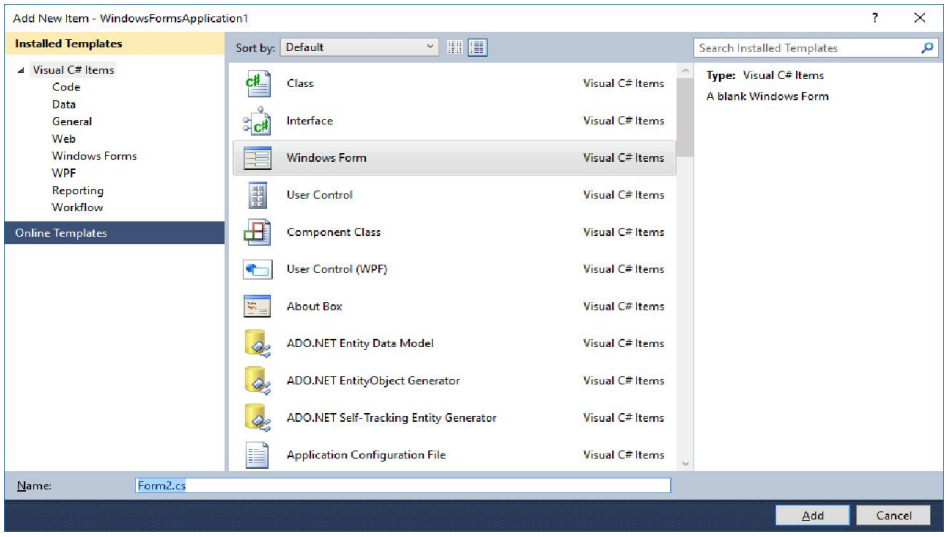
Sau khi một Project được khởi tạo theo mẫu ứng dụng Windows Form, thì một màn hình được xuất hiện được gọi là IDE Main Windows (cửa sổ môi trường phát triển giao diện), bao gồm một Form mới có tên mặc định là Form1



1. 8.1 Thực đơn Add Windows Form:

Thêm một Form mới vào project(Trường hợp này chương trình có nhiều Form)

- Vào Project/Add Windows Form

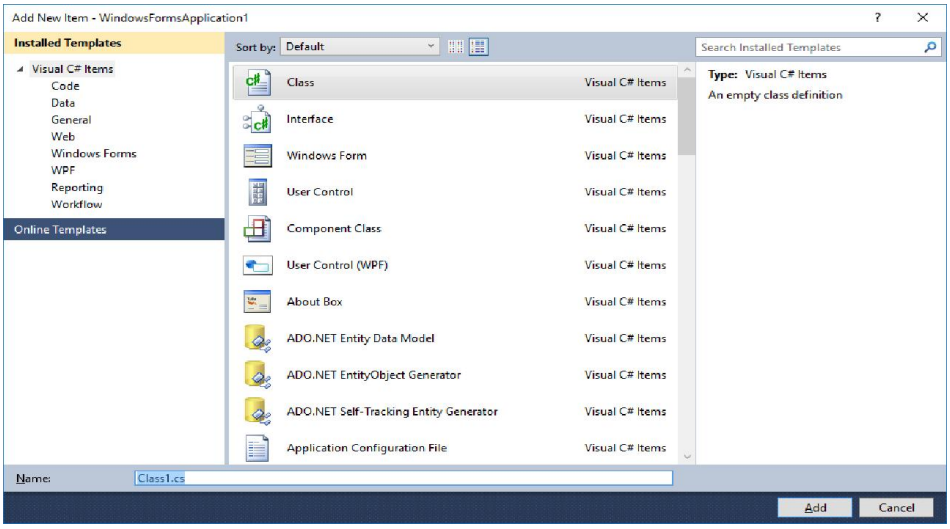


- Ở cửa sổ giữa chọn Windows Form
- Ở mục Name: Nhập tên Form cần tạo thêm trong ứng dụng
- Chọn nút Add

1.8.2 Thực đơn Add Class:

Thêm Class vào chương trình, trong trường hợp chương trình ứng dụng xây dựng theo hướng đối tượng.

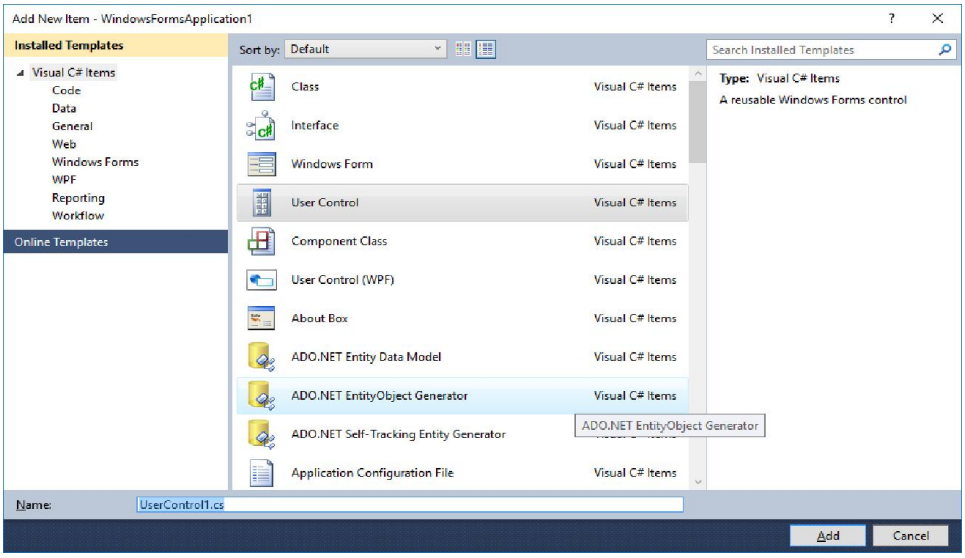
- Vào Project/Add Class



- Ở cửa sổ giữa chọn Class
- Ở mục Name: Nhập tên Class cần tạo trong ứng dụng.
- Nhấn nút Add.

1.8.3 Thực đơn Add User Control

- Vào Project/Add Use Control

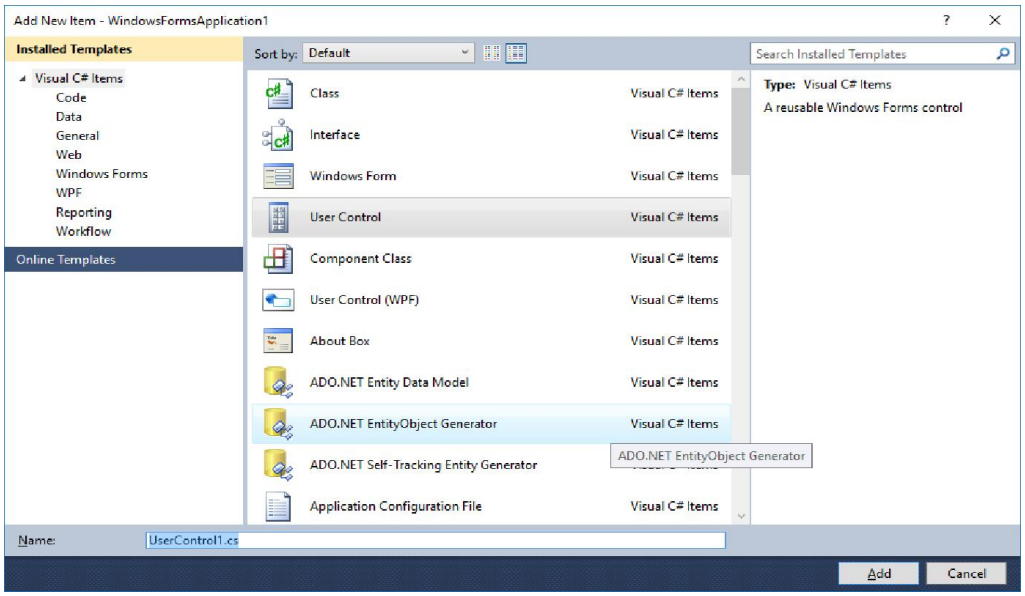


- Ở mục cửa sổ giữa chọn Use Control
- Ở mục Name nhập tên Use Contol cần tạo
- Chọn nút Add.

1.8.4 Thực đơn Add New Item

Chức năng thực đơn này dùng để thêm một Form vào chương trình, sử dụng nó khi chương trình có nhiều Form xử lý các chức năng khác nhau. Form mới thêm vào hoàn toàn mới chưa thiết kế. Lúc này để sử dụng qua lại giữa các Form ta dùng các phương thức Hide, Close, Show hay ShowDialog mà đã khảo sát ở trên.

- Vào Project/Add New Item



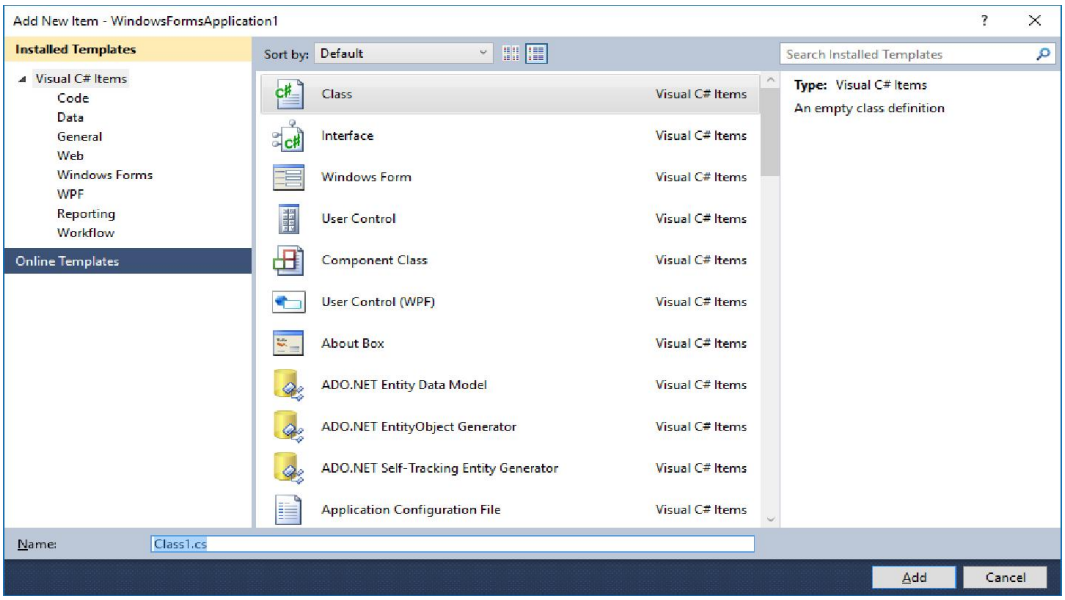
- Ở thực đơn này tuỳ ngữ cảnh chọn mục cần thêm vào ở cửa sổ giữa

- Ở mục Name nhập tên cần tạo

Thực đơn Add Exsiting Item

Chức năng thực đơn này dùng để thêm một Form đã có sẵn từ trước của project khác vào project của chương trình hiện tại. Form này đã thiết kế của project trước đó. Lúc này để sử dụng qua lại giữa các Form ta dùng các phương thức Hide, Close, Show hay ShowDialog mà đã khảo sát ở trên.

- Vào Project/Add Exsiting Item

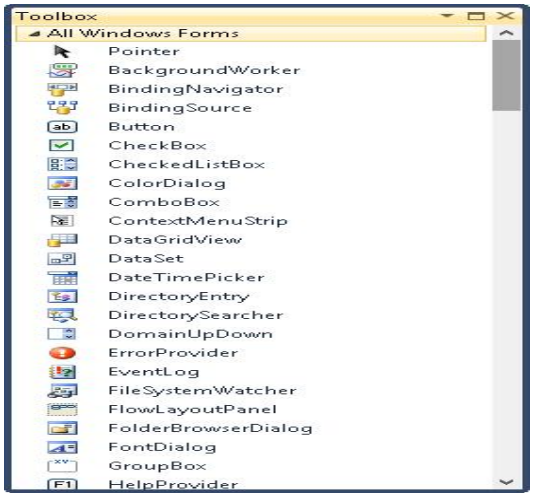


- Ở mục Look in chọn đường dẫn chứa tập tin cần thêm vào
- Chọn tập tin cần thêm vào ở trong cửa sổ Look in
- Chọn nút Add.

1.9. Thanh công cụ

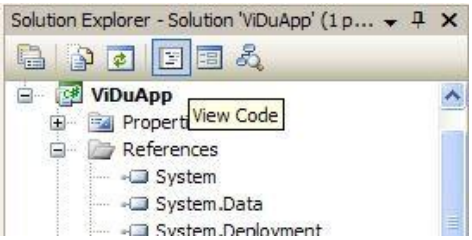
Để thiết kế giao diện cho ứng dụng, Visual Studio.Net cung cấp cho chúng ta một Toolbox liệt kê các điều khiển (Conttrol) bên trái màn hình.

Điểm tiện lợi của Visual Studio.Net là các điều khiển trên Toolbox sẽ được tự động giấu đi cho đến khi ta đưa trỏ chuột lên bên trên nó, khi ấy Toolbox sẽ liệt kê tất cả các điều khiển cho phép ta chọn để thiết kế. Ta cũng có thể "neo" cho Toolbox luôn hiển thị như hình bên dưới bằng cách (Khi Toolbox được "neo", thì Form Designer sẽ được đẩy sang phải)

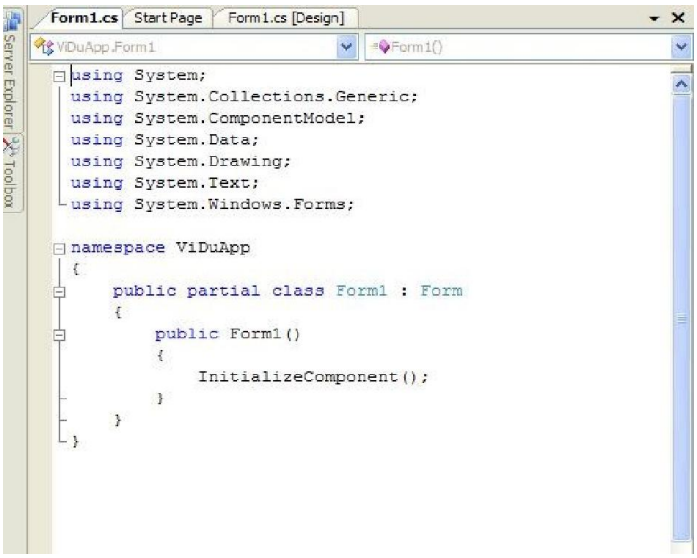


1.10. Định dạng mã C#

Là cửa sổ dùng để viết các lệnh thực thi của chương trình. Để mở cửa sổ này, ta nhấn vào nút View Code trên đầu cửa sổ Solution.



Khi đó cửa sổ viết mã lệnh được mở như sau:



Chương 2 : FORM VÀ CÁC LOẠI FORM

2.1. Các loại Forms.

Form chính là cửa sổ của một màn hình ứng dụng. Nó chứa đựng các dữ liệu, control trên đó và là cửa sổ giao tiếp giữa người sử dụng (user) và máy tính.

Khi viết một chương trình ứng dụng, thường ta không chỉ dùng một Form mà trong qúa trình chương trình làm việc nó còn phải thể hiện thêm các Form khác, nhất là dạng các Dialog để nhập dữ liệu hay trả lời, xác nhận một cái gì đó. Chương trình sử dụng bao nhiêu Form, bạn phải thiết kế bấy nhiêu Form. Trên Windows có hai dạng Form khác nhau:

- Dạng SDI (Single Document Interface): đây là dạng thông thường nhất, tất cả các Form mà ta đã tạo từ trước đến giờ đều là các SDI Window. Đối với các cửa sổ này, bên trong chỉ có các điều khiển (Control) mà thôi.
- Dạng MDI (Multi Document Interface): đây là dạng cửa sổ mà bên trong nó lại chứa nhiều cửa sổ SDI khác. Trên các SDI Window đó cũng có thể có nhiều Control khác nhau. Ta cũng đã làm việc với nhiều chương trình dạng MDI này, ví dụ: Winword, Excel,… Chẳng hạn Winword, ta thấy mỗi khi khởi động ứng dụng Winword, nó cũng tạo ra một cửa sổ mới nằm bên trong cửa sổ chính của ứng dụng, cửa sổ chính của ứng dụng Winword chính là một MDI Window.

Đặc điểm của MDI:

Trong một chương trình của C# có thể có nhiều cửa sổ MDI.

1. Cửa sổ MDI thường là cửa sổ chính của chương trình.
2. Các cửa sổ con bên trong MDI có thể được sắp xếp lại theo một trong hai dạng Casscade và Tile.
3. Các cửa sổ con SDI bên trong MDI nằm ở bên trong vùng làm việc cửa sổ cha của MDI và không thể kéo ra khỏi MDI.
4. Khi một cửa sổ con bên trong MDI được cực đại (Maximize), kích thước của nó sẽ bằng kích thước vùng làm việc của MDI (chứ không phải bằng kích thước của màn hình) và sẽ không có tiêu đề (tiêu đề của nó lúc này là chung với tiêu đề của MDI).
5. Khi một cửa sổ con bên trong MDI được cực tiểu (Minimize) thành một Icon, thì Icon của nó sẽ nằm bên trong MDI chứ không nằm trên Taskbar.
6. Khi di chuyển MDI, các cửa sổ con bên trong cũng sẽ di chuyển theo.

2.2 Các thuộc tính của Forms(Properties)

Trong Visual C#, ngoài một số thuộc tính tương tự như cửa sổ trong VB6, Access Form còn cung cấp một số thuộc tính khác để ấn định dáng vóc và phong cách của giao diện. Bảng sau liệt kê một số thuộc tính mới và thay đổi của Form.

Thuộc tính	Mô tả
AcceptButton	Nút lệnh trên Form sẽ được gọi tình huống/sự kiện Click khi phím Enter được nhấn.
AutoScale	Giá trị True/False quy định Form và cá điều khiển trên nó có định lại kích thước cho phù hợp khi Font của Form thay đổi hay không.
CancelButton	Nút lệnh trên Form sẽ được gọi tình huống/sự kiện Click khi phím Escape được nhấn.
DesktopBounds	Đối tượng Rectangle xác định vị trí và kích thước trên Desktop của Windows.
DesktopLocation	Đối tượng Point xác định vị trí trên Desktop của Windows.
DialogResult	Giá trị trả về của Form khi mở Form bằng phương thức ShowDialog (Modal). Chúng ta gán trị cho thuộc tính này bằng một giá trị từ Enum DialogResult tương tự giá trị trả về của MessageBox: DialogResult.Abort DialogResult.Cancel DialogResult.Ignore DialogResult.No DialogResult.None DialogResult.Ok DialogResult.Retry DialogResult.Yes Nếu trên Form mở chế độ Modal có các nút lệnh được gán trị ở thuộc tính DialogResult, khi nhấn vào nút nào, giá trị DialogResult của Form sẽ lấy trị DialogResult của nút nhấn. Nếu người dùng nhấn nút X (Close) trên thanh tiêu đề của Form, Form sẽ ẩn đi và giá trị DialogResult.Cancel sẽ được gán cho DialogResult của Form.

	Phương thức Close không tự động được gọi khi nhấn nút X cũng như khi gán trị cho DialogResult. Vì thế, nếu ứng dụng không cần Form này nữa cần phải gọi phương thức Dispose để giải phóng vùng nhớ.
IsMdiChild	Trả về giá trị True/False cho biết Form có phải là Form con của ứng dụng MDI không.
IsMdiContainer	Giá trị True/False cho biết Form có phải là Form chứa các Form con của ứng dụng MDI không.
Location	Quy định tọa độ góc trái trên của Form khi xuất hiện (trên màn hình hay trên đối tượng chứa nó)
MdiChildren	Trả về mảng các Form con dạng mở của Form chính trong ứng dụng MDI.
Modal	Trả về giá trị True/False cho biết Form có đang mở ở chế độ Modal hay không. Một Form mở ở chế độ Modal có nghĩa là: muốn thao tác đến một đối tượng khác ngoài Form thì phải đóng Form này lại mới thao tác được.
Opacity	Một giá trị %, quy định độ trong suốt của Form (nếu dưới 100%)
OwnedForms	Trả về mảng các Form thuộc về Form hiện hành. Nếu một FormA thuộc về FormB, nó sẽ bị thu nhỏ và đóng khi FormB bị thu nhỏ và đóng lại. Các Form được sở hữu nàykhông bao giờ hiển thị phía sau Form sở hữu chúng.
Owner	Form sở hữu hiện hành.
TopLevel	Giá trị True/False cho biết Form có phải là Top level form. Top-level Form phải là một Form không phải Form con hay là Form con của Desktop Window. Các Top-level Form thường được dùng làm Form chính trong ứng dụng.
TopMost	Giá trị True/False cho biết Form phải được hiển thị như Topmost Form. Top-most Form là Form luôn nằm trên các Form khác ngay cả khi không hiện hành (Always on top)

Text	Tiêu đề của Form
StartPosition	Quy định vị trí khi biểu mẫu xuất hiện, gồm các trị sau: **Manual**: Biểu mẫu được canh theo vị trí khai báo bởi thuộc tính location. **Centerscreen:** biểu mẫu được canh giữa màn hình. ta có thể dùng lệnh trong chương trình như sau: ***Startposition = Formstartposition.Centerscreen*** **Windowsdefaultlocation:** biểu mẫu được hiển thị theo vị trí mặc nhiên của windows. nói cách khác, sẽ không biết chính xác biểu mẫu sẽ được canh về đâu. **Windowsdefaultbound:** Biểu mẫu được hiển thị theo vị trí mặc nhiên của windows với một kích thước mặc nhiên (thuộc tính size bị bỏ qua). **Centerparent:** Nếu biểu mẫu được hiển thị theo kiểu Modal thì nó sẽ được canh giữa tương đối với biểu mẫu hiển thị nó. Nếu biểu mẫu này không là biểu mẫu con thì đặt để này giống giống như WindowsDefaultLocation.

2.3. Các phương thức của Forms(Methods).

Phương thức	Mô tả
AddOwnedForm	Thêm một Form vào tập hợp các Form thuộc sở hữu của Form hiện hành. **Cú pháp:** AddOwnedForm(<Tên Form>)
SetBounds	Định biên cho form trên đối tượng chứa. **Cú Pháp:** SetBounds(<Trái>, <Đỉnh>, <Rộng>, <Cao>) <Trái>, <Đỉnh>, <Rộng>, <Cao> là các trị kiểu Integer xác định hoành độ, tung độ, chiều rộng và chiều cao của Form.
SetDesktopBounds	Định biên cho Form trên Desktop, cách sử dụng như SetBounds.

SetDesktopLocation	Định vị cho Form trên Desktop, cách sử dụng như SetBounds nhưng chỉ có <Trái> và đỉnh.
LayoutMdi	Sắp xếp các Form con theo cách truyền vào **Cú pháp:** LayoutMdi(<Kiểu sắp xếp>) Kiểu sắp xếp gồm các trị sau: **ArrangeIcons:** sắp xếp các biểu tượng của Form con trong vùng hiển thị của Form. **Cascade:** sắp xếp các Form con trong vùng theo kiểu lợp ngói. **TileHorizontal:** sắp xếp các Form con trong vùng theo kiểu lát gạch theo chiều ngang. **TileVertical:** sắp xếp các Form con trong vùng theo kiểu lát gạch theo chiều dọc.
ShowDialog	Mở biểu mẫu dạng Modal. Khi Form không thuộc về Form khác, dùng cú pháp: **ShowDialog( )** Khi Form thuộc về Form khác, dùng cú pháp: **ShowDialog(<Tên Form sở hữu>)** Phương thức trả về giá trị DialogResult (xem thuộc tính DialogResult ở trên). Ta dùng phương thức này để hiển thị Form dạng hộp thoại trong ứng dụng. Sau khi được gọi, các dòng lệnh sau lệnh gọi sẽ không được thực hiện cho đến khi Form được đóng lại.
Show	Hiển thị Form. Khi gọi phương thức này, Form hiển thị và chương trình thực hiện tiếp các dòng lệnh tiếp theo.
Hide	Ẩn Form
Close	Đóng Form hiện hành

2.4. Các sự kiện của Forms(Events)

Trong lập trình visual, điều quan trọng nhất là sử lý các sự kiện. Khi lập trình, thường ta chỉ thực hiện các thao tác kéo thả là ta có thể tạo được một giao diện hoàn

chỉnh. Tuy nhiên, để giao diện đó hoạt động được theo đúng yêu cầu của ta thì ta buộc phải lập trình cho các sự kiện của từng hay nhiều control trên form đó.

Sự kiện	Mô tả
Load	Xử lý cho quá trình Form được load lên. Trường hợp này thông thường gọi là mặc định Form khi thực hiện phương thức Show hay ShowDialog
FormClosing	Xử lý cho quá trình khi đóng Form hay thoát chương trình
Click	Xử lý khi chuột được click lên trên form

Để tạo 1 hàm xử lý sự kiện (event function), cách tốt nhất là bạn bấm double click chuột lên đối tượng mà bạn muốn tạo hàm để xử lý sự kiện đó.

Tuy nhiên, cách này thì VS .Net nó chỉ phát sinh 1 hàm xử lý sự kiện mặc định của nó (với form thi hàm mặc định xử lý sự kiện là Form_Load.

Do vậy, bạn hãy mở cửa sổ properties của đối tượng đó, chọn tab các events, nó sẽ liệt kê các events mà đối tượng có, sau đó bạn double click vào sự kiện mà bạn muốn, và một hàm xử lý sự kiện được tạo ở bên phần source code và bạn chỉ việc viết thêm code để xử lý sự kiện này mà không cần quan tâm đến bằng cách nào mà hệ thống kiểm tra và xử lý nó.

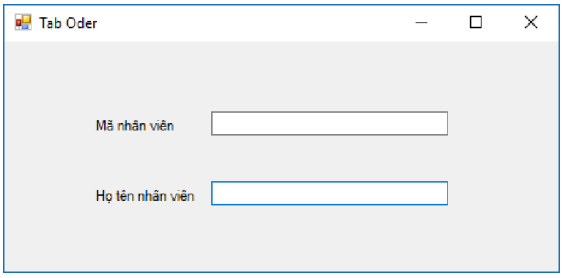
Chương 3 : CÁC ĐIỀU KHIỂN THƯỜNG DÙNG

3.1. Điều khiển Label.

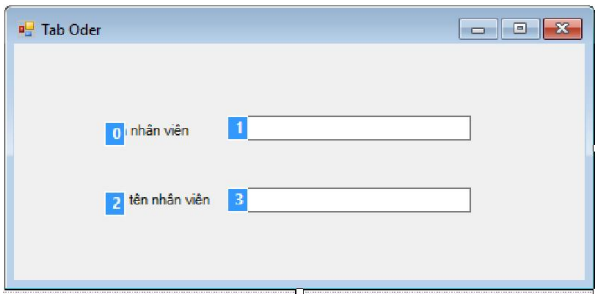
Label là thành phần đơn giản nhất và cũng là một trong những thành phần quan trọng nhất và thường xuyên được sử dụng nhất trong lập trình Winform. Label chỉ để dùng trình bày một chuỗi văn bản thông thường nhằm mục đính mô tả thêm thông tin cho các đối tượng khác. Ta cũng có thể dùng Label để làm công cụ đưa kết quả ra màn hình dưới dạng một chuỗi.

Các thuộc tính cơ bản của một label và thường được dùng tới như: Font, Text, TextAlign, ForeColor, Visible.
Ngoài ra Label còn có một đặc biệt gọi là sự kiện Click. Thông thường một label sẽ được đặt tên là: lbl…, Ví dụ: lblDangNhap, lblUser,…

Label không thể nhận Focus (tiêu điểm). Tuy nhiên, ta có thể dùng Label để di chuyển nhanh đến một điều khiển bằng phím nóng thông qua TabOrder.



Hiển thị TabOder của tất cả các điều khiển trên Form thiết kế giúp người dung có thể sửa lại theo thứ tự TabOder theo cách của mình khi nhấn phím Tab. Từ menu bar vào View/Tab Oder, nếu cần thay đổi tabIndex cho điều khiển nào đó thì di chuyển đến vị trí đó thay đổi.



Ta có thể hiệu chỉnh thông tin hiển thị trên Label thông qua thuộc tính Text. Thuộc tính Text là thuộc tính mặc định của Label bằng cách

lbl_Tên.Text=”Chuỗi hiển thị mới”;

Các thuộc tính(Properties):

Thuộc tính	Mô tả
Name	Tên tham chiếu của Label. Thuộc tính này có chung trên các loại điều khiển.
AutoSize	Cho phép thay đổi độ rộng Label **False:** cho phép thay đổi **True:** không cho phép thay đổi
BackColor	Màu nền
BorderStyle	Kiểu viền, gồm : **None:** không viền **Fixsingle:** viền đơn mảnh **Fix3D:** viền nổi
Enabled	Gồm hai giá trị: **True:** cho điều khiển có hiệu lực **False:** vô hiệu hóa điều khiển. Thuộc tính này có chung trên các loại điều khiển.
Font	Phông chữ
ForeColor	Màu chữ
Image	Quy định hình ảnh hiển thị trên Label
ImageAlign	Quy định vị trí của hình ảnh sẽ hiển thị trên Label.
ImageList	Điều khiển chứa các hình hiển thị trên Label
ImageIndex	Chỉ số của hình sẽ hiển thị trên Label
Text	Chuỗi văn bản hiển thị trên điều khiển. Thuộc tính này có chung trên các loại điều khiển.
TextAlign	Quy định vị trí của chuỗi văn bản sẽ hiển thị trên Label.

Visible	Gồm hai giá trị: **True:** cho hiển thị điều khiển **False:** ẩn điều khiển. Thuộc tính này có chung trên các loại điều khiển.

Các sự kiện (Events):

Phương thức	Mô tả
Click	Được kích hoạt khi nhấn chuột lên điều khiển.
DoubleClisk	Được kích hoạt khi nhấn đôi chuột lên điều khiển.
TextChanged	Được kích hoạt khi chuỗi của điều khiển bị thay đổi.

3.2. Điều khiển TextBox.

Là điều khiển được dùng để hiển thị hoặc nhập dữ liệu, nó là một trong những điều khiển thường dùng nhất trong Windows. TextBox trong C# có thể chứa đến 2 tỷ ký tự.

Một điểm mới trong điều khiển TexBox là xử lý riêng rẽ từng dòng văn bản có trong TextBox thông qua thuộc tính Lines. Thuộc tính Lines là một mảng chuỗi. Mỗi phần tử trong mảng giữ giá trị của một dòng văn bản. Lines(0) chứa dữ liệu của dòng đầu tiên, Lines(1) chứa dữ liệu của dòng thứ hai, … số dòng văn bản trong điều khiển TextBox được nhận thông qua Lines.Length.

Cuối cùng, thuộc tính cho phép ta chọn khối văn bản trong TexBox như SelStart, SelLength và SelText đã được đổi tên mới: SelectionStart, SelectionLength, SelectedText. Thuộc tính Alignment dùng để canh lề văn bản trong TextBox được đổi thành TextAlign. Phương thức AppendText cho phép thêm nội dung văn bản vào điều khiển với tốc độ thi hành nhanh hơn phép gán:

TextBox1.Text = TextBox1.Text + Newline

TextBox là điều khiển dùng để nhập liệu vô cùng linh hoạt, nó thường được sử dụng để nhập liệu một hoặc nhiều dòng văn bản như số, mật mã hoặc những ký tự văn bản. Trừ những ứng dụng đồ họa, TexBox không thể thiếu đối với các ứng dụng trong Windows.

Các thuộc tính(Properties):

Thuộc tính	Mô tả
Name	Tên tham chiếu của TextBox. Thuộc tính này có chung trên các loại điều khiển.
MultiLine	Cho phép TextBox hiển thị một hay nhiều dòng văn bản, gồm hai giá trị: **True:** cho phép hiển thị nhiều dòng **False:** chỉ hiển thị một dòng. Mặc định chỉ hiển thị một dòng (False).
ScrollBar	Cho phép gắn thêm thanh cuộn vào TextBox nếu văn bản thể hiện trong TextBox vượt quá kích thước của điều khiển, gồm các lựa chọn: **None:** không có thanh cuộn. **Horizontal:** thanh cuộn ngang. **Vertical:** thanh cuộn dọc. **Both:** cả hai. TextBox có thuộc tính MultiLine = False sẽ không có thanh cuộn cho dù nội dung văn bản thể hiện bên trong có vượt quá kích thước của điều khiển.
WordWrap	Gồm hai giá trị: **True:** cho phép nội dung của văn bản trong TextBox tự động xuống dòng khi kích thước ngang của điều khiển không đủ để hiển thị phần nội dung. **False:** không cho phép tự động xuống dòng dù kích thước ngang của điều khiển không đủ để hiển thị phần nội dung của văn bản. Mặc định là **True**. Khi thuộc tính **WordWrap = True** thì TextBox sẽ không có thanh cuộn ngang ngay cả khi thuộc tính ScrollBar được quy định là Horizotal.
AcceptsReturn	Gồm hai giá trị: **True:** cho phép xuống dòng mới trong điều khiển khi nhấn phím Enter.

	False: không cho phép xuống dòng mới trong điều khiển khi nhấn phím Enter. Chỉ có tác dụng với **MultiLine = True.** Mặc định thuộc tính AcceptsReturn = False, trong trường hợp này muốn xuống dòng trong TextBox ta dùng tổ hợp phím Ctrl + Enter.
AcceptsTab	Thông thường phím Tab được dùng để chuyển Focus (tiêu điểm) từ điều khiển này sang điều khiển khác. Khi quy định thuộc tính này là ta quy định lại cách sử dụng phím Tab bên trong Điều khiển TextBox được chỉ định. Thuộc tính này gồm hai giá trị: **True:** cho phép dùng phím Tab để tạo ký tự Tab trong điều khiển. Trong trường hợp này nếu muốn chuyển Focus sang điều khiển khác ta phải dùng tổ hợp phím Ctrl + Tab **False:** khi nhấn phím Tab sẽ chuyển Focus sang điều khiển khác.
CanUndo	Gồm hai giá trị: **True:** cho phép sử dụng phương thức Undo để phục hồi lại các thay đổi vừa mới xảy ra trong TextBox. **False:** không cho phép sử dụng phương thức Undo.
CharacterCasing	Chuyển đổi các ký tự của nội dung trong TextBox, gồm ba giá trị: CharacterCasing.Lower: Thành các ký tự thường CharacterCasing.Normal: Không thay đổi CharacterCasing.Upper: Thành các ký tự HOA
Dock	Là thuộc tính mới nhằm neo điều khiển tại một vị trí cố định. Mỗi điều khiển đều có một số thứ tự ở trong điều khiển chứa nó, gọi là Z-order , Z-order của mỗi điều khiển là duy nhất đối với điều khiển chứa nó. Khi hai điều khiển nằm trong cùng một
MaxLength	Quy định số ký tự tối đa được nhận trong TextBox

Text	Dùng Để Gán Giá Trị Cho Textbox. Trong Lúc Thiết Kế: Nếu Thuộc Tính Multiline = False Thì Nhập Giá Trị Vào Thuộc Tính Text. Ngược lại, nhập giá trị vào thuộc tính lines thông qua hộp thoại string collection editor. Thuộc tính Text của TextBox là một đối tượng thuộc lớp String, do đó nó cũng có các phương thức của đối tượng String. **Ví Dụ:** int Chieudai ; Chieudai = TextBox1.TextLength(); Để xóa nội dung trong TextBox ta có thể dùng phép gán: TextBox1.Text = “”; Hoặc dùng phương thức Clear: TextBox1.Clear();
ReadOnly	Gồm hai giá trị: **True:** không được hiệu chỉnh nội dung trong TextBox. **False:** được phép hiệu chỉnh nội dung trong TextBox.
Locked	Khác với C#, thuộc tính này không cho người dùng di chuyển điều khiển TextBox trong khi thiết kế (nếu gán trị True). Ngược lại, cho phép người dùng di chuyển điều khiển TextBox trong quá trình thiết kế.
Lines	Hỗ trợ cho thuộc tính Text, cho phép nhập nội dung vào TextBox khi thiết kế nếu thuộc tính Multiline = True. Thuộc tính này mang tính chỉ đọc, nghĩa là không thể hiệu chỉnh nội dung của TextBox khi thực thi.
PasswordChar	Quy định ký tự đặc biệt để thay thế các ký tự hiển thị trong TextBox, được dùng để nhập “Mật mã”.
HideSelection	Gồm hai giá trị: **True:** khối văn bản được chọn trong TextBox sẽ không được tô đậm màu khi Focus nằm ở điều khiển khác. **False:** khối văn bản được chọn trong TextBox sẽ luôn được tô đậm màu khi Focus nằm ở điều khiển khác.

Các phương thức(Methods):

Phương thức	Mô tả
SelectedText	Trả về phần nội dung được lựa chọn trong TextBox.
SelecttionStart	Trả về vị trí đầu tiên của ký tự được chọn trong TextBox.
SelectionLenght	Trả về chiều dài của chuỗi được chọn trong TextBox.
Select	Chọn khối văn bản trong TextBox từ vị trí bắt đầu với chiều dài là số ký tự được chỉ định. **Cú pháp:** TênTextBox.Select(VịtríBắtđầu, SốKýtựmuốnchọn)
SelectAll	Chọn khối văn bản trong ô TextBox **Cú pháp:** TênTextBox.SelectAll();
Undo	Phục hồi lại các thay đổi xảy ra gần đây nhất trên TextBox
ClearUndo	Vô hiệu hóa chức năng Redo.
Clear	Xoá dữ liệu trong TextBox
Focus	Di chuyển con trỏ nhập đến ô TextBox
TextLength	Cho biết chiều dài của nội dung trong TextBox (đây là phương thức kế thừa từ lớp String)
IndexOf	Vì TextBox là một kiểu String, nên có các phương thức String. Tìm chuỗi con có trong chuỗi. Trả về 2 giá trị -1: Không tìm thấy Khác -1: Tìm thấy chuỗi con **Ví dụ:** txt_Tên.Text.IndexOf(Chuỗi con tìm);
Contains	Tìm chuỗi con có trong chuỗi. Giá trị trả về **True:** Tìm thấy **False:** Không tìm thấy **Ví dụ:** txt_Tên.Text.Contains(Chuỗi con tìm) ;

Các sự kiện(Events):

Sự kiện	Mô tả
MouseClick	Xảy ra khi người dùng nhấp chuột trên điều khiển.
MouseDoubleCLick	Xảy ra khi người dùng nhấp đúp chuột trên điều khiển.
MultilineChanged	Xảy ra khi thuộc tính Multiline thay đổi giá trị từ true sang false hay ngược lại.
TextChange	Xảy ra khi nhập hay xoá trong điều khiển.
EnabledChange	Xảy ra khi thuộc tính Enabled thay đổi trị từ true sang false hay ngược lại.
ReadOnlyChange	Xảy ra khi thuộc tính ReadOnly thay đổi trị từ true sang false hay ngược lại.
VisibleChange	Xảy ra khi thuộc tính Visible thay đổi trị từ true sang false hay ngược lại.
KeyPress	Xảy ra khi một phím được nhấn.
KeyDown	Tương tự như KeyPress, nhưng xử lý với cú pháp lệnh hơi khác.
KeyUp	Tương tự như KeyPress, nhưng xử lý với cú pháp lệnh hơi khác.

- *Để lập trình các tình huống/sự kiện KeyDown, KeyUp ta cần biết một số mã phím (KeyCode) cần xử lý như: F1 là 112 hoặc Keys.F1, F2 là 113 hoặc Keys.F2, ..., Keys.Back (BackSpace) là 8.*

- *Để kiểm tra trạng thái của các phím điều khiển như Shift, Ctrl, Alt ta sử dụng các thuộc tính Shift, Control, Alternate của đối tượng e (tham số của tình huống/sự kiện) như: e.Shift, e.Ctrl, e.Atl.*

Mặc dù Windows hỗ trợ người dùng bẫy tình huống/sự kiện với các nút chức năng như F1, F2 … Tuy vậy ta nên tránh dùng những phím chức năng mà Windows đang sử dụng.

Ví dụ:

Khi nhập dữ liệu vào ô TextBox1, kết thúc nhập ta nhấn phím Enter thì di chuyển con trỏ nhập đến ô TextBox2. Ta quy định sự kiện KeyDown hay KeyUp như sau:

```
private void TextBox1_KeyDown(object sender, KeyPressEventArgs e){
        if (e.KeyCode==Keys.Enter) TextBox2.Focus();
}
```

Hay

```
private void TextBox1_KeyUp(object sender, KeyPressEventArgs e){
        if (e.KeyCode==Keys.Enter) TextBox2.Focus();
}
```

Ví dụ:

Ràng buộc dữ liệu nhập vào một TextBox phải là một con số, ta quy định tại tình huống/sự kiện KeyPress như sau:

```
private void TextBox1_KeyPress(object sender, KeyPressEventArgs e {
        if (char.IsLetter(e.KeyChar) || char.IsWhiteSpace(e.KeyChar)
        || char.IsSymbol(e.KeyChar) ||  char.IsPunctuation(e.KeyChar))

                e.Handled = true;
}
```

Ví dụ:

Kiểm tra dữ liệu nhập vào một TextBox phải là một chuỗi ký tự, ta quy định tại tình huống/sự kiện KeyPress như sau:

```
private void TextBox1_KeyPress(object sender, KeyPressEventArgs {
        if (char.IsNumber(e.KeyChar) || char.IsSymbol(e.KeyChar)
                || char.IsPunctuation(e.KeyChar))
                e.Handled = true;
}
```

3.3. Điều khiển Button.

Button cũng là một phần không thể thiếu trong quá trình Design Form. Nhiệm vụ chính của Button là dùng chuột nhấn vào nó để thao tác, cho phép thực thi một hành động.

Thuộc tính thường dùng nhất của button là Text, chính là hiển thị chuỗi trên bề mặt button. Sự kiện của button quan trọng nhất đó là Click, nó sẽ được kích hoạt khi click vào button , khai báo mặc định khi người dùng double click vào button trong màn hình Design View của Form. Thông thường một button sẽ được đặt tên là: btn… , Ví dụ: btnXoa, btnReset,…

Các thuộc tính(Properties):

Thuộc tính	Mô tả
BackColor	Màu nền của Button
BackgroundImage	Ảnh nền của Button
Cursor	Hình con trỏ chuột khi đi qua Button
Enabled	Khoá/Mở khoá nút Button
FlatStyle	Chọn kiểu nút Button
Font	Chọn Phông chữ cho Button
ForeColor	Chọn màu chữ trên Button
Text	Nội dung hiển thị trên Button
Name	Đặt tên cho Button
TextAlign	Căn lề chữ trên Button

- Ngoài ra cũng có thể chỉnh các thuộc tính bằng code:

Ví dụ: Chỉnh text của Button:

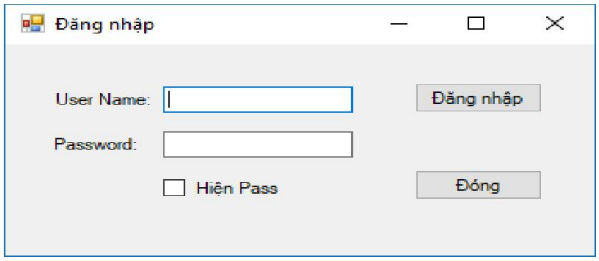
```
Button1.Text = "Thay đổi text";
Button1.Enabled = false;...
```

Các sự kiện(events):

Sự kiện	Mô tả
Click	Khi Click vào Button

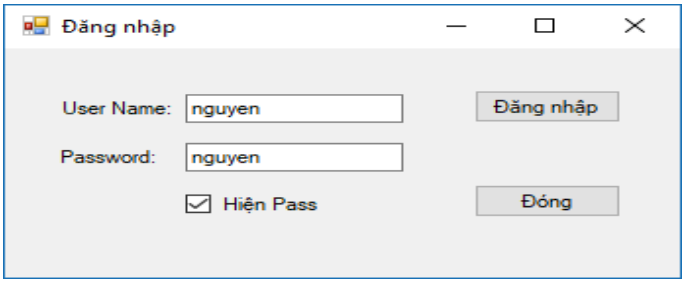
Ví dụ:

Xét chương trình đăng nhập có giao diện thiết kế như sau

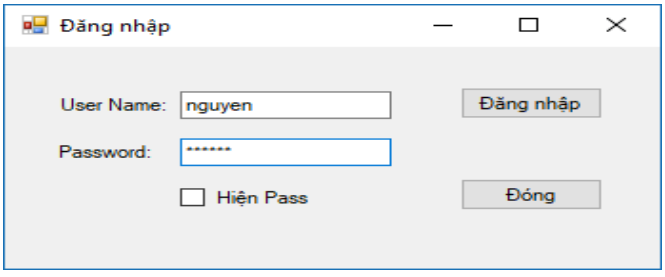


Gồm có:

- Hai TextBox dùng cho nhập User Name và Password.
- Hai nút nhấn Button có chức năng Đăng nhập và Đóng.
- Một CheckBox dùng để hiển thị Password khi được chọn trong ô CheckBox.
- Nếu dấu CheckBox được chọn thì ô Password hiển thị đúng Password.



- Nếu nhập User Name và Password giống nhau và khác trống thì xuất thông báo ”Đăng nhập thành công”.
- Ngược lại thì xuất thông báo "Đăng nhập sai.\nBạn nhập User Name hay Password không đúng yêu cầu".



Thiết đặt các Properties:

Đặt tên cho các đối tượng như sau:

Điều khiển	Thuộc tính
Với TextBox User Name	Name: txtUserName. Text: để trống
Với TextBox Password	Name: txtPass. Text: để trống
Với Button Đăng nhập	Name: btndangNhap. Text: Đăng nhập
Với Buuton Đóng	Name: btnDong Text: Đóng
Với CheckBox Hiện pass	Name: chkPas Text: Hiện Pass

Các xử lý sự kiện như sau

```
private void btnDong_Click(object sender, EventArgs e)      {
        Close();
}
```

```
private void btnDangNhap_Click(object sender, EventArgs e)    {
  if (txtUser.TextLength ==txtPass.TextLength && txtUser.TextLength>= 4)  {
      MessageBox.Show("Đăng nhập thành công","Thông báo");
      return;
    }
        MessageBox.Show("Đăng nhập sai.
              \nBạn nhập User Name hay Password không đúng yêu cầu",
              "Thông báo");
        txtUser.Clear();
        txtPass.Clear();
        txtUser.Focus();
 }
```

```
private void chkPass_CheckedChanged(object sender, EventArgs e)     {
        if (chkPass.Checked) txtPass.PasswordChar = (char)0;
        else txtPass.PasswordChar = '*';
}
```

3.4. Điều khiển ComboBox.

ComboBox là hộp danh sách lựa chọn thả xuống nó có thể cho phép nhập giá trị trực tiếp hay chọn giá trị từ danh sách đã có sẵn, nó chiếm ít không gian hơn. Mỗi dòng trong ComboBox là một giá trị. Chỉ số dòng được đánh theo thứ tự từ trên xuống bắt đầu là 0.

Các thuộc tính(Properties):

Thuộc tính	Mô tả
Name	Tên tham chiếu của ComboBox. Thuộc tính này có chung trên các loại điều khiển.
DataSource	Nhập nguồn dữ liệu của điều khiển. **Ví dụ:**string[] data={"Điện tử","Cơ khí","Điện lạnh"}; TênComboBox.DataSource=data;
Sorted	Gồm Hai Giá Trị: **True:** Sắp xếp các mục chọn trong danh sách, ngay cả các mục mới thêm vào. **False:** Không sắp xếp các mục chọn trong danh sách.
MaxDropDownItems	Quy định số mục chọn hiển thị trong Drop-Down ListBox (nếu số mục chọn trong danh sách có nhiều hơn số mục chọn quy định thì sẽ có thanh trượt dọc để xem)
Items	Tập hợp chứa các mục chọn của điều khiển. Ta có thể thêm mục chọn vào thời điểm thiết kế thông qua cửa sổ String Collection Editor bằng cách nhấn vào nút bên cạnh. Hoặc thông qua lệnh khi thi hành.

DropDownStyle	Quy định kiểu cho điều khiển ComboBox, bao gồm ba lựa chọn: **DropDo**wn: đây là giá trị mặc định. Điều khiển bao gồm một TextBox và một Drop-Down ListBox (hộp danh sách đổ xuống: sẽ tự động thu lại sau khi người sử dụng chọn một mục chọn). Người sử dụng có thể chọn một mục trong danh sách hay nhập một mục mới trong TextBox **DropDownList**: tương tự như DropDown, cũng cho phép người sử dụng chọn một mục trong danh sách, nhưng không cho phép nhập liệu trong TextBox. **Simple:** điều khiển bao gồm một TextBox và một ListBox (không Drop-Down). Người sử dụng có thể chọn một mục trong danh sách hay nhập một mục mới trong TexBox.
DropDownWidth	Mặc định, thuộc tính này bằng thuộc tính Width của điều khiển ComboBox. Tuy nhiên, ta có thể quy định kích thước ngang của Drop-Down ListBox thông qua thuộc tính này.

Các phương thức(Methods):

Phương thức	Mô tả
DataSource	Thêm vào ComboBox tập hợp các mục chọn **Ví dụ:** String[] Nuoc={"Cà phê", "Sting", "Pepsi"}; cbo_Tên.DataSource=Nuoc;
Add	Thêm vào ComboBox một mục chọn
AddRange	Thêm vào ComboBox tập hợp các mục chọn **Ví dụ:** String[] Nuoc={"Cà phê", "Sting", "Pepsi"}; cbo_Tên.Items.AddRange(Nuoc);
Remove	Xoá mục chọn có giá trị trong ComboBox **Ví dụ:** Cú pháp xoá 1 dòng trong ComboBox

	TênComboBox.Items.Remove(TênComboBox.SelectedItem);
RemoveAt	Xoá mục chọn có chỉ số trong ComboBox Ví dụ: Cú pháp xoá 1 dòng trong ComboBox TênComboBox.Items.Remove(TênComboBox.SelectedIndex);
Clear	Xoá dữ liệu trong ComboBox **Ví dụ:** Cú pháp các dòng trong ComboBox TênComboBox.Items.Clear();
Focus	Di chuyển con trỏ nhập đến ô ComboBox
SelectedIndex	Trả về chỉ số của mục hiện đang được chọn trong ComboBox

Các sự kiện cơ bản (events):

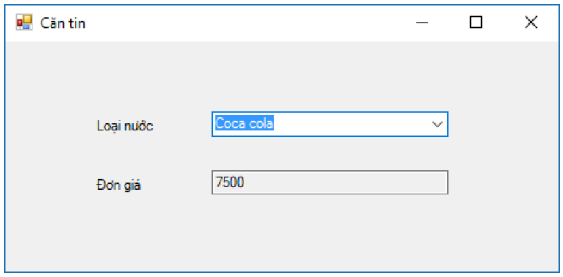
Sự kiện	Mô tả
SelectedIndexChanged	Xảy ra khi chọn một mục trong ComboBox

Ví dụ:

Giả sử ta có các loại nước và đơn giá tương ứng sau

Các loại nước	Đơn giá
Pepsi Cola	7000
Coca Cola	7500
7Up	6000
Sting	8000
Number One	6500

Khi thực thi chương trình, mặc định chương trình như hình

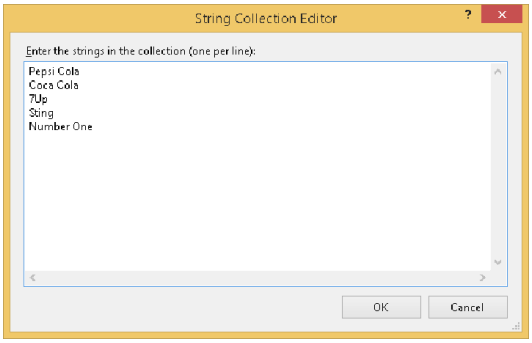


Gồm có:

1. Một ComboBox cho phép chọn loại nước.
2. Một Label xuất ra đơn giá sau khi chọn loại nước ở trên ComboBox.

Thiết đặt các Properties:

Đặt tên cho các đối tượng như sau:

Điều khiển	Thuộc tính
Với Label đơn giá:	Name: lblDonGia. Text: để trống
Với ComboBox:	Name: cboLoaiNuoc cho ComboBox Items nhấp chọn dấu "…"  Nhập các loại nước ở hộp thoại này

Đầu tiên ta khai báo mảng đơn giá kiểu số thực

float[] dongia = {7000,7500,6000,8000,6500};

các giá trị mảng đơn giá tương ứng theo thứ tự các loại nước ở giả thiết trên.

Viết sự kiện Load của Form1

```
private void Form1_Load(object sender, EventArgs e){
        cboLoaiNuoc.SelectedIndex = 1;
}
```

Khi chọn 1 dòng trong ComboBox loại nước thì đơn giá của loại nước tương ứng xuất hiện ở ô đơn giá. Viết sự kiện SelectedIndexChanged cho ComboBox loại nước

```
private void cboLoaiNuoc_SelectedIndexChanged(object sender, EventArgse){
        lblDonGia.Text=dongia[cboLoaiNuoc.SelectedIndex].ToString();
}
```

3.5. Điều khiển ListBox.

ListBox trong C# có khá nhiều thay đổi so với VB6, hay Access, nó không còn thuộc tính List. Thay vào đó, để xử lý từng mục chọn trên danh sách, ta xử lý thông qua tập hợp Items. Mục đầu tiên của điều khiển là Items[0], mục thứ hai là Items[1], … Tổng số mục lựa chọn trong danh sách nhận được thông qua thuộc tính Count của tập hợp Items (Items.Count).

Cách xử lý trong trường hợp nhiều lựa chọn cũng được cải tiến. Nếu điều khiển chỉ cho phép một lựa chọn (Single Selection), thuộc tính SelectedIndex và SelectedItem sẽ trả về chỉ số và giá trị của mục được chọn. Nếu điều khiển cho phép nhiều lựa chọn (Multi Selection), ta sẽ sử dụng tập hợp SelectedIndices và SelectedItems để xử lý những chỉ số giá trị của mục được chọn chứ không phải duyệt hết toàn bộ các mục có trong Listbox.

Ví dụ:

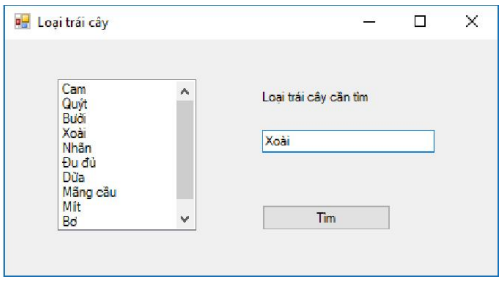
```
for (int i = 0; i <= ListBox1.SelectedIndices.Count -1; i++) {
        MessageBox.Show(ListBox1.SelectedItems[i]);
}
```

Lưu ý: *Ta có thể thay thế* ***SelectedIndices*** *bằng* ***SelectedItems***

Một điểm nổi bật của ListBox trong C# xây dựng sẵn phương thức tìm kiếm cho người sử dụng. Ta có thể sử dụng phương thức FindString và FindStringExact để xác định vị trí của một mục (Item) trong ListBox. Vị trí dòng đầu tiên trong ListBox được đánh chỉ số là 0.

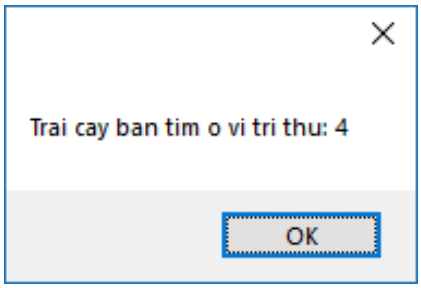
Ví dụ:

Ta có giao diện gồm một ListBox liệt kê các loại trái cây, và một TextBox cho phép nhập tên trái cây cần tìm. Ta xây dựng đoạn chương trình cho tình huống Click của nút tìm như sau:




```
private void btnThiHanh_Click(object sender, EventArgs e) {
        int vt;
        vt = System.Convert.ToInt16(listBox1.FindString(txtTim.Text))+1;
        MessageBox.Show("Trai cay ban tim o vi tri thu: " + vt.ToString());
}
```

Với FindString ta được kết quả như sau:

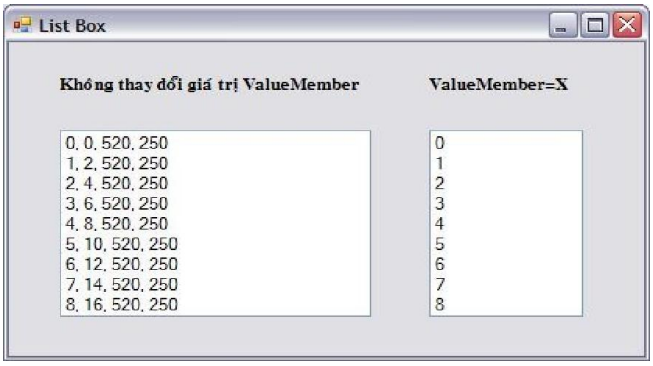


Cuối cùng, trong C#, ListBox có thêm hai phương thức mới nhằm tăng tính năng hiển thị của ListBox (và cả ComboBox) trong khi thêm nhiều mục lựa chọn vào điều khiển. Khi thêm nhiều mục chọn (Item) vào điều khiển cùng một lúc, trước hết ta gọi phương thức BeginUpdate(), sau đó Add hoặc Insert các mục lựa chọn vào danh sách và kết thúc bằng phương thức EndUpdate. Kỹ thuật này tránh trường hợp điều khiển phải vẽ lại nhiều lần khi một mục mới được thêm vào và chỉ vẽ lại khi gọi phương thức EndUpdate.

Các thuộc tính(Propreties):

Thuộc tính	Mô tả
Name	Tên tham chiếu của ListBox. Thuộc tính này có chung trên các loại điều khiển.
FLatStyle	Kiểu đường viền của điều khiển.
DataSource	Nhập nguồn dữ liệu của điều khiển. Ví dụ: string[] data={"Điện tử","Cơ khí","Điện lạnh"}; TênListBox.DataSource=data;
DisplayMember	Tên của Field (Trường) dữ liệu tương ứng với chuỗi trình bày trên điều khiển.
Items	Tập hợp chứa các mục chọn của điều khiển. Ta có thể thêm mục chọn vào thời điểm thiết kế thông qua cửa sổ String Collection Editor bằng cách nhấn vào nút bên cạnh. Hoặc thông qua lệnh khi thi hành.
MaxDropDownItems	Số phần tử (dòng) lớn nhất có thể liệt kê trong điều khiển (mặc định là 8).
DisplayMember	Tên của Field tương ứng với chuỗi trình bày trong ListBox.
ColumnWidth	Quy định chiều rộng của mỗi cột, ngăn cách nhau bởi dấu ";"
Sorted	Gồm Hai Giá Trị: • True: Sắp Xếp Các Mục Chọn Trong Danh Sách, Ngay Cả Các Mục Mới Thêm Vào. • False: Không Sắp Xếp Các Mục Chọn Trong Danh Sách.
MultiColumn	Gồm hai giá trị: True: cho phép ListBox hiển thị các mục chọn trên nhiều cột. False: ListBox hiển thị các mục chọn chỉ trên một cột.

Text	Giá trị của mục đang được chọn trong ListBox. Trong trường hợp có nhiều mục được chọn, nó sẽ trả về giá trị của mục được chọn đầu tiên theo thứ tự hiển thị trong ListBox.
IntegralHeight	Cho phép hay không cho phép ListBox hiển thị một phần của mục chọn trong kích thước hiển thị. Thuộc tính này gồm hai giá trị: True: ListBox sẽ tự động thay đổi chiều cao để không hiển thị một phần của mục chọn. False: giữ nguyên kích thước, không thay đổi chiều cao. Mặc định mang giá trị True.  IntegralHeight = True  IntegralHeight = False
ValueMember	Tập hợp các đối tượng được lưu trữ trong listbox, trong đa số trường hợp, người ta dùng listbox để lưu trữ các dữ liệu kiểu chuỗi. Ta có thể hiển thị bất kỳ thuộc tính nào của mục chọn bằng cách thay đổi giá trị của thuộc tính ValueMember của ListBox = tên thuộc tính của mục cần hiển thị. Ví dụ sau minh họa cách sử dụng thuộc tính ValueMember: ListBox có tên lstA không thay đổi giá trị thuộc tính ValueMember, nó sẽ hiển thị thông tin về đối tượng Rectangle. Trong khi đó ListBox có tên lstValMem thay đổi giá trị của thuộc tính ValueMember = X, do đó nó hiển thị thông tin giá trị tọa độ X (của Rectangle) trong lstValMem ***private void FrmSo2_Load****(object sender, EventArgs e) {* *int i;* *for (i = 0; i<=8; i++) {* *Rectangle r= new Rectangle(i, 2 * i, 520, 250);* *lstA.Items.Add(r);* *}* *lstValMem.ValueMember = "X";* *for (i = 0; i<=8;i++) {*

	*Rectangle rec = new Rectangle(i, 2 * i, 520, 250);* *lstValMem.Items.Add(rec);* *}* *}* 
SelectionMode	Quy định cách thức chọn các mục chọn trong danh sách. SelectionMode chỉ được phép thay đổi trong lúc thiết kế, vào lúc chương trình thực thi thì thuộc tính này mang tính chỉ đọc. Gồm các giá trị sau: **None**: không cho phép chọn lựa một mục nào cả. **One:** chỉ được chọn một mục trong danh sách (mặc định). § MultiSimple: cho phép chọn nhiều mục (đơn giản), bằng cách nấn chuột hay phím SpaceBar. **MultiExtended:** cho phép chọn nhiều mục chọn (mở rộng). Nhấn Shfit + nhấn chuột (hoặc phím mũi tên lên, xuống) để chọn một hoặc nhiều mục chọn trong danh sách. Nhấn Ctrl + nhấn chuột để chọn hay bỏ chọn một mục trong danh sách.

Các phương thức(Menthods):

Phương thức	Mô tả
DataSource	Thêm một chuỗi hay một tập đối tượng vào danh sách. **Ví dụ:** String[] Nuoc={“Cà phê”, “Sting”, “Pepsi”}; lst_Tên.DataSource=Nuoc;
Add	Thêm một mục chọn mới vào danh sách (chuỗi hay đối tượng).

	Sử dụng phương thức Items.Add theo cú pháp: TênListBox.Items.Add(<Giá trị>)
AddRange	Thêm một chuỗi hay một tập đối tượng vào danh sách. **Ví dụ:** String[] Nuoc={"Cà phê", "Sting", "Pepsi"}; lst_Tên.Items.AddRange(Nuoc);
Clear	Xóa tất cả các mục chọn. Sử dụng cú pháp: TênListBox.Items.Clear
Insert	Thêm một mục chọn mới vào danh sách tại một vị trí. Sử dụng phương thức Items.Insert theo cú pháp: TênListBox.Items.Insert(<Vị trí>, (<Giá trị>)
CopyTo	Chép các giá trị của các mục chọn trong ListBox từ vị trí chỉ định và lưu chúng vào một mảng. Sử dụng theo cú pháp: TênListBox.Items.CopyTo(<Tên mảng>, <vị trí bắt đầu>)
Count	Trả về số mục (Item) có trong danh sách.
SelectedItem	Trả về giá trị của mục hiện đang được chọn trong ListBox
SelectedIndex	Trả về chỉ số của mục hiện đang được chọn trong ListBox (tính từ 0)
SelectedItems	Tập hợp giá trị các mục đang được chọn trong ListBox
SelectedIndices	Tập hợp các chỉ số của các mục đang được chọn trong ListBox.
Remove	Xóa một mục chọn ra khỏi danh sách trong ListBox. **Cú pháp**: TênListBox.Items.Remove(<Giá trị>); Nếu có nhiều giá trị giống nhau trong danh sách thì chỉ có giá trị đầu tiên bị xóa. Trong trường hợp các mục chọn là đối tượng, ta truyền vào một biến tham chiếu đến mục chọn cần xóa.

RemoveAt	Xóa một mục chọn tại vị trí chỉ định ra khỏi danh sách. **Cú pháp:** TênListBox.Items.RemoveAt(<Vị trí>);
Contains	Kiểm tra xem giá trị (mục chọn) này đã có trong tập hợp các mục chọn hay chưa. Gồm hai giá trị: **True:** đã có mục chọn trong danh danh sách. **False:** chưa có mục chọn trong danh sách.
FindString	Tìm tương đối (gần đúng) một chuỗi trong danh sách. Nếu tìm thấy, phương thức sẽ trả về vị trí tìm thấy đầu tiên (tính từ 0), ngược lại sẽ trả về giá trị -1. FindString có thể tìm từ một vị trí cụ thể trên danh sách với tham số <Vị trí>. **Cú pháp:** TênListBox.FindString(<Chuỗi> [, <Vị trí>])
FindStringExact	Tương tự như FindString, nhưng tìm chính xác giá trị của một chuỗi trong danh sách.

Ví dụ: Hàm con xử lý xoá các dòng được chọn trong ListBox

```
private void LstXoa(ListBox lst){
        //Xoá tất cả các dòng  được chọn trong ListBox
        while(lst.SelectdIndices.Count>0)
                lst.Items.RemoveAt(lst.SelectedIndices[0]);
    lst.SelectedIndices.Clear();//Xoá tập hợp danh sách chỉ số
}
```

Khi sử dụng hàm LstXoa. Ta chỉ gọi hàm LstXoa(TênListBox).

Ví dụ: Hàm con xử lý di chuyển dòng được chọn trong ListBox

//Di chuyển lùi dòng

```
private void LstLuiDong(ListBox lst, int  vt){
  if(lst.SelectedIndex!=-1) {
        if(vt>0)//nếu dòng i không phải là dòng đầu
        {
              vt--;
              lst.SelectIndex=vt;
        }
  }
}
```

//Di chuyển tăng dòng

```
private void LstTangDong(ListBox lst, int  vt){
      if(lst.SelectedIndex!=-1) {
        if(vt<lst.Items.Count-1)//dòng vt không phải là dòng đầu
        {
            vt++;
            lst.SelectIndex=vt;
        }
      }
}
```

//Di chuyển về dòng đầu

```
private void LstDongDau(ListBox lst){
      if(lst.SelectedIndex!=-1)
            lst.SelectedIndex=0;
}
```

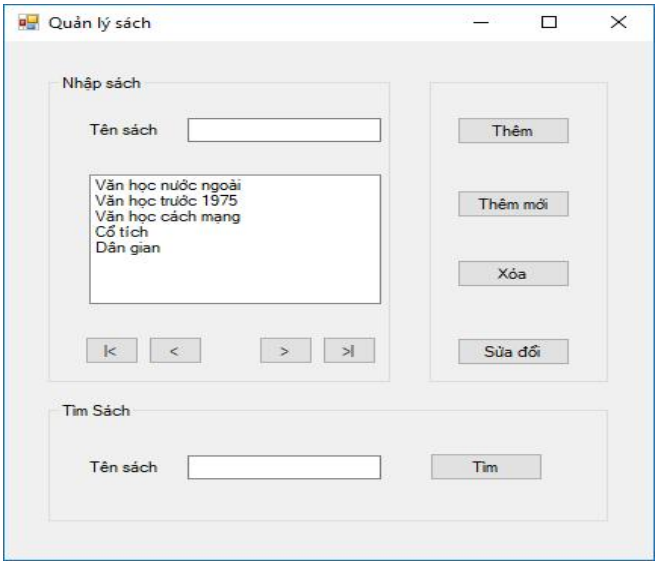
//Di chuyển về dòng cuối

```
private void LstDongCuoi(ListBox lst){
        if(lst.SelectedIndex!=-1)
                lst.SelectedIndex=lst.Items.Count-1;
}
```

Các sự kiện(Events):

Sự kiện	Mô tả
MouseClick	Xảy ra khi người dùng nhấn trái chuột lên ListBox
MouseDoubleClick	Xảy ra khi người dùng nhấp đúp chuột lên ListBox
SelectedIndexChanged	Xảy ra khi chỉ mục của mục chọn được thay đổi (chọn mục khác).
SelectedValueChanged	Xảy ra khi giá trị của mục chọn được thay đổi (chọn mục khác).
EnableChanged	Xảy ra khi thuộc tính Enabled thay đổi từ True sang False và ngược lại.
DisplayMemberChanged	Xảy ra khi tên cột khai báo trong thuộc tính DisplayMember thay đổi.
ValueMemberChanged	Xảy ra khi tên cột khai báo trong ValueMember thay đổi.
VisibleChanged	Xảy ra khi thuộc tính Visible thay đổi giá trị từ True sang False hay ngược lại.

Ví dụ: Xét chương trình có giao diện sau:



Gồm có:

1. Một TextBox cho phép nhập tên sách.
2. Một ListBox chứa danh sách các tên sách vừa nhập..
3. Các nút lệnh (Button) để thực hiện các yêu cầu tương ứng cho tiêu đề nút nhấn.
4. Một TextBox cho phép nhập tên sách tìm
5. Một menu lệnh để thoát ứng dụng.

Thiết đặt các Properties:

Đặt tên cho các đối tượng như sau:

Các điều khiển	Thuộc tính
Với TextBox tên sách:	Name: txtTenSach cho TextBox. MaxLength: 100
Với ListBox dùng để lưu danh sách các tên sách :	Name: lstTenSach
Với TextBox tìm tên sách	Name: txtTimTenSach MaxLength: 100

Với Label hiển thị dòng	Name: lblDuyet Text: để trống
Với nút di chuyển về đầu	Name: btnVeDau Text: |<
Với nút di chuyển lùi	Name: btnLui Text: <
Với nút di chuyển tiến	Name: btnTien Text: >
Với nút di chuyển về cuối	Name: btnVeCuoi Text: >|
Với nút thêm	Name: btnThem Text: Thêm
Với nút thêm mới	Name: btnThemMoi Text: Thêm mới
Với nút xóa	Name: btnXoa Text: Xóa
Với nút sửa đổi	Name: btnSuaDoi Text: Sửa đổi
Với nút tìm kiếm	Name: btnTimKiem Text: Tìm kiếm

Cài đặt các sự kiện các điều khiển

Sự kiện nút lệnh **Thêm mới**

```
private void btnThemMoi_Click(object sender, EventArgs e) {
        txtTenSach.Clear();
        txtTensach.Focus();
}
```

Sự kiện nút lệnh **Xóa**

```
private void btnXoa_Click(object sender, EventArgs e) {
        if(lstTenSach.SelectedIndex !=-1){
                lstTenSach.Items.Remove(lstTenSach.SelectedIndex);
                btnThemMoi_Click(sender, e);
                reutrn;
        }
        MessageBox("Bạn chưa chọn dòng xóa","Thông báo");
}
```

Sự kiện nút lệnh **Thêm**

```
        private void btnThem_Click(object sender, EventArgs e) {
        if(txtTenSach.TextLength==0)
        {
                MessageBox.Show("Chưa nhập tên sách","Thông báo");
                txtTenSach.Focus();
                return;
        }
        if(lstTenSach.FinStringExact(txtTenSach.Text)==-1)
        {
                lstTenSach.Items.Add(txtTenSach.Text);
                btnThemMoi_Click(sender, e);
                reutrn;
        }
        MessageBox("Trùng tên sách","Thông báo");
        txtTenSach.Focus();
        txtTensach.SelectAll();
}
```

Sự kiện nút lệnh **Sửa đổi**

```
private void btnSuaDoi_Click(object sender, EventArgs e) {
        if(lstTenSach.SelectedIndex!=-1){
                if(txtTenSach.TextLength==0){
                        MessageBox.Show("Bạn đã xóa  tên sách","Thông báo");
                        txtTenSach.Focus();
                        return;
                }
                lstTenSach.Items[lstTenSach.SelectedIndex]=txtTenSach.Text;
                txtTenSach.Clear();
                reutrn;
        }
        MessageBox("Bạn chưa chọn dòng dữ liệu cần sửa","Thông báo");
}
```

Sự kiện nút **Tìm kiếm**

```
private void btnTimKiem_Click(object sender, EventArgs e) {
        if(txtTimTenSach.TextLength==0){
                MessageBox.Show("Chưa nhập tên sách cần tìm","Thông báo");
                txtTimTenSach.Focus();
                return;
        }
        if(lstTenSach.FinStringExact(txtTimTenSach.Text)!=-1){
                MessageBox("Tìm thấy tên sách cần tìm","Thông báo");
                reutrn;
        }
        MessageBox("Không tìm thấy tên sách. Vui lòng nhập tên khác",
                        "Thông báo");
        txtTimTenSach.Focus();
        txtTimTensach.SelectAll();
}
```

Sự kiện **SelectedIndexChanged** của ListBox dang sách

```
private void lstTenSach_SelectedIndexChanged(object sender, EventArgs e) {
        if(lstTenSach.SelectedIndex!=-1){
            txtTenSach.Text=lstTenSach.SelectedItem.ToString();
        }
}
```

3.6. Điều khiển CheckBox.

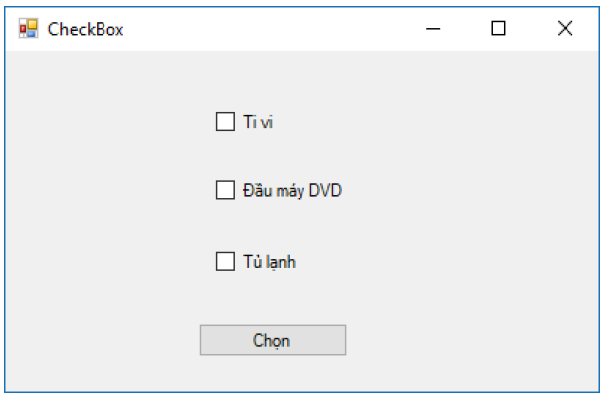
Là điều khiển dùng để quy định chọn lựa một mục gì đó, khi nhấn điều khiển ta được một trong hai giá trị sau:

true: khi điều khiển được chọn (có dấu kiểm bên trong)

false: khi bỏ chọn (điều khiển rỗng).

Khi đặt nhiều điều khiển CheckBox lên Form, ta có thể chọn hoặc bỏ chọn một hay nhiều mục chọn vì chúng mang tính độc lập với nhau.

Ví dụ:



Các thuộc tính(Properties):

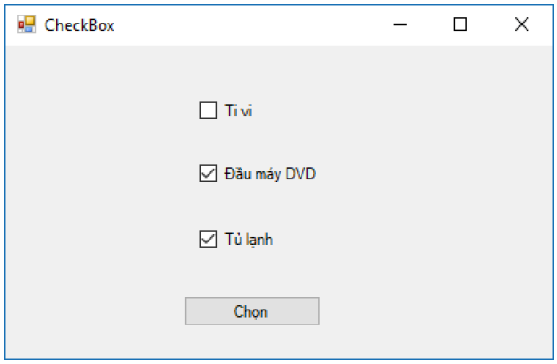
Thuộc tính	Mô tả
Name	Tên của điều khiển.
FlatStyle	Kiểu đường viền của điều khiển.

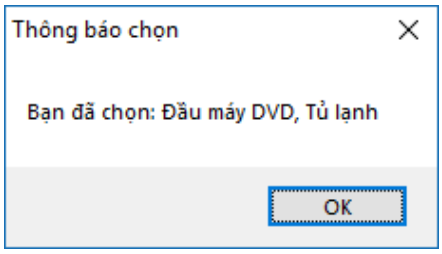
Appearance	Hình dạng của điều khiển.
Text	Chuỗi trình bày (diễn giải của tùy chọn)
CheckAlign	Quy định vị trí của hộp chọn so với nhãn đi cùng
Checked	Gồm hai giá trị: True: đang được chọn False: không được chọn
ThreeState	Cho phép điều khiển có hay không 3 trạng thái được chọn trong thuộc tính CheckState
CheckState	Chỉ trạng thái điều khiển. Gồm ba giá trị: Unchecked: điều khiển đang được chọn Checked: đang đươc chọn Indeterminate: vô định (không ro, mập mờ)

Để truy cập thuộc tính của điều khiển CheckBox, ta có thể khai báo đoạn chương trình trong tình huống Click của nút "Chọn" như sau:

```
private void btnChon_Click(object sender, EventArgs e)
{
        string choose="";
        foreach (Control chk in this.Controls) {
                if (chk is CheckBox) {
                        if (((CheckBox)chk).Checked == true) {
                                choose += chk.Text + ", ";
                        }
                }
        }
        choose = choose.Substring(0, choose.Length - 2);
        MessageBox.Show("Ban da chon: " + choose);
}
```

Khi thi hành chương trình, nếu người sử dụng chọn các điều khiển trên Form và nhấn nút "Chon" thì sẽ có kết quả như sau:





Các sự kiện (Event):

Sự kiện	Mô tả
CheckedChanged	Được kích hoạt khi có sự thay đổi giữa chọn và bỏ chọn
Click	Được kích hoạt khi nhấn chọn lên đối tượng

Chú ý: Trường hợp nếu các ô CheckBox nằm trong GroupBox thì sự kiện nút chọn có thay đổi một chút.

```
private void btnChon_Click(object sender, EventArgs e) {
        string choose="";
        foreach (Control grp in this.Controls) {
            if(grp is GroupBox){
                foreach(Control chk in grp.Controls){
                    if (chk is CheckBox) {
                        if (((CheckBox)chk).Checked == true) {
                            choose += chk.Text + ", ";
```

```
                    }
                }
            }
        }
    }
    choose = choose.Substring(0, choose.Length - 2);
    MessageBox.Show("Ban da chon: " + choose);
}
```

3.7. Điều khiển CheckedListBox.

Tương tự như ListBox, CheckedListBox cũng dùng để hiển thị một danh sách, nhưng trước mỗi mục chọn có thêm hộp kiểm để xác định mục chọn có đang được chọn hay không. Mỗi mục chọn trong CheckedListBox cũng có ba trạng thái tương tự như CheckBox là: Checked, UnChecked và Inderterminate.

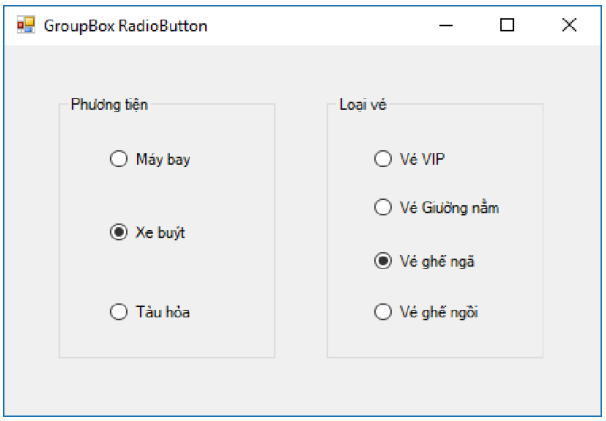
Lưu ý: ta phải gán trạng thái Indeterminate bằng mã lệnh bởi vì điều khiển không cung cấp kỹ thuật để chọn trạng thái này trong chế độ thiết kế.

Các thuộc tính và phương thức cũng tương tự như ListBox.

3.8. Điều khiển RadioButton.

Là điều khiển dùng để chứa và phân nhóm các điều khiển RadioButton, khi các điều khiển RadioButton được đặt trong cùng một GroupBox (được gọi là một nhóm nút chức năng), các điều khiển này sẽ mang tính loại trừ nhau, nghĩa là khi chọn một mục này thì các mục khác sẽ bị loại bỏ.

Ví dụ:



Các thuộc tính(Properties):

Thuộc tính	Mô tả
Name	Tên của điều khiển.
Enabled	Quy định tính hiệu lực của điều khiển.
FlatStyle	Quy định cách thể hiện(định dạng) của điều khiển.
Text	Chuỗi trình bày (diễn giải của điều khiển)
CheckAlign	Quy định vị trí của hộp chọn so với nhãn đi cùng
Checked	Gồm hai giá trị: True: đang được chọn False: không được chọn

Các phương thức(Method):

Phương thức	Mô tả
EnabledChanged	Được kích hoạt khi tính hiệu lực của điều khiển được thay đổi
Enter	Được kích hoạt khi điều khiển được Focus.
MouseHover	Được kích hoạt khi rê chuột trên điều khiển.

Các sự kiện (Event):

Sự kiện	Mô tả
CheckedChanged	Được kích hoạt khi có sự thay đổi giữa chọn và bỏ chọn
Click	Được kích hoạt khi nhấn chọn lên đối tượng

3.9. Điều khiển PictureBox.

Là điều khiển dùng để hiển thị một hình ảnh nào đó lên Form.

Các thuộc tính(Properties):

Thuộc tính	Mô tả
Image	Quy định hình ảnh được hiển thị. Khi thiết kế, ta có thể chỉ định hình ảnh hiển thị bằng cách nhấn vào nút  bên cạnh. Trong quá trình thực thi, muốn gán lại hình ảnh cho đối tượng, ta có thể dùng phương thức Image.FromFile của lớp System.Drawing theo cú pháp sau: **System.Drawing.Image.FromFile(<DuongDan>)** Với <DuongDan> là chuỗi đường dẫn chỉ đến tập tin hình ảnh muốn hiển thị.
BorderStyle	Quy định kiểu đường viền của điều khiển.
SizeMode	Quy định cách thức hiển thị của hình ảnh bên trong điều khiển, gồm các giá trị sau: 1. Normal: Hình ảnh hiển thị theo kích thước gốc của nó. 2. StrectchImage: Hình ảnh hiển thị căng ra theo kích thước của PictureBox. 3. AutoSize: Kích thước của PictureBox sẽ co, giản cho vừa với kích thước gốc của hình ảnh. 4. CenterImage: Hiển thị hình ảnh vảo giữa PictureBox.

Khi thao tác với với các đối tượng, để tạo màu cho các đối tượng khi chương trình thực thi ta có thể dùng phương thức đồ họa Color.FromArgb của lớp Color tạo màu từ ba màu Red, Green, Blue như sau:

Color.FromArgb(RedValue, GreenValue, BlueValue)

Các phương thức(Methods): không có.

Các sự kiện(Events): không có.

Ví dụ: Sử dụng PictureBox

Viết chương trình đổi năm dương lịch sang năm âm lịch có kèm hình ảnh 12 con giáp tương ứng.

Năm âm lịch dựa vào Can Chi để tính. Để đổi năm dương lịch sang năm âm lịch dựa vào bảng sau.

Bảng tính Can

NămDL%10	0	1	2	3	4	5	6	7	8	9
Can	Canh	Tân	Nhâm	Quý	Giáp	Ất	Bính	Đinh	Mậu	Kỷ

Bảng tính Chi

NămDL%12	0	1	2	3	4	5	6	7	8	9	10	11
Chi	Thân	Dậu	Tuất	Hợi	Tý	Sửu	Dần	Mão	Thìn	Tỵ	Ngọ	Mùi

Chẳng hạn: nhập vào ô năm dương lịch là 2019. Sau đó nhấn nút View thì năm âm lịch xuất hiện trong ô năm âm lịch là Kỷ Hợi.

Dựa vào bảng tính Can và Chi ở trên ta tạo các mảng Can và chi như sau

```
string[] Can = {"Canh", "Tân", "Nhâm", "Quý", "Giáp", "Ất",
                "Bính", "Đinh", "Mậu", "Kỷ"};
string[] Chi = {"Thân", "Dậu", "Tuất", "Hợi", "Tý", "Sửu",
                "Dần", "Mão", "Thìn", "Tỵ", "Ngọ", "Mùi"};
```

Các hình ảnh của các chi tương ứng được Add vào Resources trong thuộc tính Solution của chương trình. Ta có thể khai báo đoạn chương trình trong tình huống Click của nút "View" như sau:

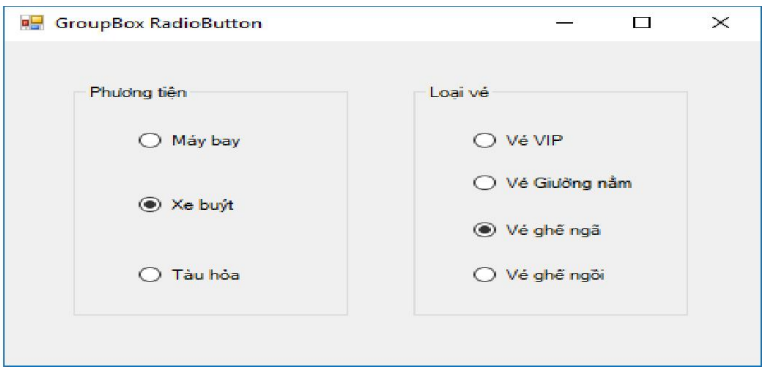
```
private void btnView_Click(object sender, EventArgs e){
    int namDL=int.Parse(txtNamDL.Text);
    txtNamAL.Text=Can[namDL%10]+ " " +
                        Chi[namDL%12];
switch(namDl%12)
{
    case 0:
        picHinh.Image=Properties.Resources.Than;
        break;
  case 1:
        picHinh.Image=Properties.Resources.Dau;
        break;
 case 2:
        picHinh.Image=Properties.Resources.Tuat;
        break;
 case 3:
        picHinh.Image=Properties.Resources.Hoi;
        break;
 case 4:
        picHinh.Image=Properties.Resources.Ty;
        break;
 case 5:
        picHinh.Image=Properties.Resources.Suu;
        break;
case 6:
        picHinh.Image=Properties.Resources.Dan;
        break;
case 7:
        picHinh.Image=Properties.Resources.Mao;
        break;
```

```
            case 8:
                        picHinh.Image=Properties.Resources.Thin;
                        break;
            case 9:
                        picHinh.Image=Properties.Resources.Ti;
                        break;
           case 10:
                        picHinh.Image=Properties.Resources.Ngo;
                        break;
            default:
                        picHinh.Image=Properties.Resources.Mui;
                        break;
           }
}
```

3.10. Điều khiển GroupBox.

Là điều khiển dùng để chứa các điều khiển có cùng mục đích giống nhau hoặc phân nhóm các điều khiển RadioButton. Khi các điều khiển RadioButton được đặt trong cùng một GroupBox (được gọi là một nhóm nút chức năng), các điều khiển này sẽ mang tính loại trừ nhau, nghĩa là khi chọn một mục này thì các mục khác sẽ bị loại bỏ.

Ví dụ:



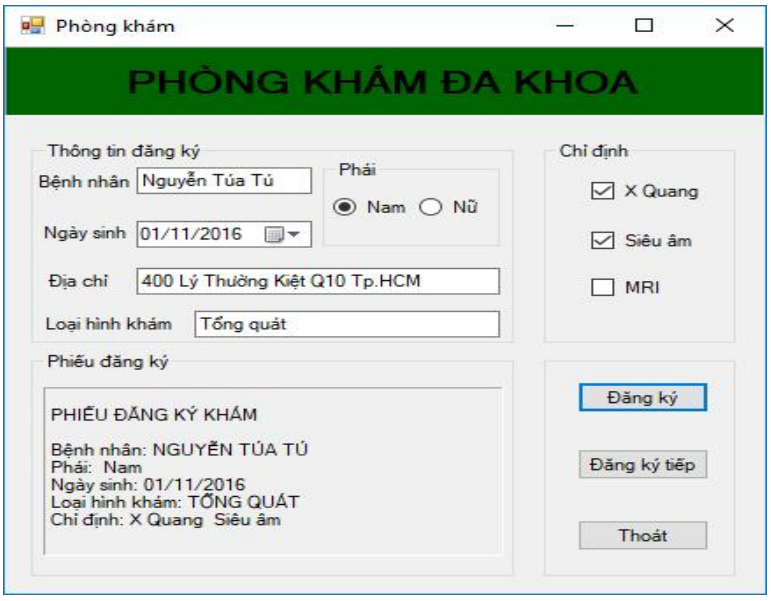
Các thuộc tính(Properties):

Thuộc tính	Mô tả
Name	Tên của điều khiển.
Enabled	Quy định tính hiệu lực của điều khiển.
FlatStyle	Quy định cách thể hiện(định dạng) của điều khiển.
Text	Chuỗi trình bày (diễn giải của điều khiển)
CheckAlign	Quy định vị trí của hộp chọn so với nhãn đi cùng

Các phương thức(Methods):

Phương thức	Mô tả
EnabledChanged	Được kích hoạt khi tính hiệu lực của điều khiển được thay đổi
Enter	Được kích hoạt khi điều khiển được Focus.
MouseHover	Được kích hoạt khi rê chuột trên điều khiển.

Ví dụ: Thiết kế và xây dựng chương trình sau



Gồm có:

1. Ba Textbox nhập liệu tên, địa chỉ, loại hình khám.
2. Một DateTimePicker dùng để chọn ngày tháng năm sinh
3. Ba CheckBox dùng để lựa chọn chỉ định
4. Một Label xuất ra phiếu đăng ký
5. Ba nút nhấn Button để thực hiện chức năng xử lý.
6. Bốn GroupBox dùng để nhóm các lệnh.

Thiết đặt các Properties:

Đặt tên cho các đối tượng như sau:

Điều khiển	Thuộc tính
Với Label phiếu khám	Name: lblPhiếu. Text: để trống AutoSize: False
Với TextBox bệnh nhân	Name: txtBenhNhan. Text: để trống TextLenght: 100
Với TextBox địa chỉ	Name: txtDiaChi. Text: để trống TextLenght: 100
Với TextBox loại hình khám	Name: txtLoaiHinh Text: để trống TextLenght: 100
Với DateTimePicker	Name: dtpNgaySinh CustomFormat: Custom Format:dd/MM/yyyy
Ba CheckBox dùng để chọn chỉ định	Name: chkXQuang, chkSieuAm, chkMRI Text: X Quang, Siêu âm, MRI Checked: False
Ba nút nhấn Button	Name: btnDangKy, btnDangKyTiep, btnThoat Text: Đăng ký, Đăng ký tiếp, Thoát

Đầu tiên ta khai báo mảng đơn giá kiểu số thực

string chidinh;

Các sự kiện nút nhấn Button

```
private void btnDangKyTiep_Click(object sender, EventArgs e)       {
        radNam.Checked = true;
        txtBenhNhan.Clear();
        txtDiaChi.Clear();
        txtLoaiHinh.Clear();
        dtpNgaySinh.Value = DateTime.Now;
        lblPhieu.Text = "";

        chkMRI.Checked = false;
        chkSieuAm.Checked = false;
        chkXQuang.Checked = false;

        txtBenhNhan.Focus();
}
```

```
private void btnDangKy_Click(object sender, EventArgs e)     {
   if (txtBenhNhan.TextLength == 0)        {
        MessageBox.Show("Vui lòng nhập tên bệnh nhân.","Thông báo");
              txtBenhNhan.Focus();
               return;
         }
         if (txtDiaChi.TextLength == 0)       {
              MessageBox.Show("Vui lòng nhập thông tin địa chỉ.",
                                                          "Thông báo");
              txtDiaChi.Focus();
```

```
                return;
            }
            if (txtLoaiHinh.TextLength == 0)      {
                MessageBox.Show("Vui lòng nhập loại hình đăng ký khám.",
                                                        "Thông báo");
                txtLoaiHinh.Focus();
                return;
            }
            chidinh=(chkXQuang.Checked?"X Quang " :"")+
                (chkSieuAm.Checked?" Siêu âm ":"")+
                (chkMRI.Checked?" MRI ":"");
            lblPhieu.Text = "\nPHIẾU ĐĂNG KÝ KHÁM\n" +
                "\nBệnh nhân: " + txtBenhNhan.Text.ToUpper() +
                "\nPhái: " + (radNam.Checked ? " Nam" : " nữ") +
                "\nNgày sinh: " + dtpNgaySinh.Value.ToString("dd/MM/yyyy")
+
            "\nLoại hình khám: " + txtLoaiHinh.Text.ToUpper() +
            "\nChỉ định: " + chidinh;
    }
```

```
    private void btnThoat_Click(object sender, EventArgs e)      {
        Application.Exit();
    }
```

```
    private void Form1_Load(object sender, EventArgs e)      {
        btnDangKyTiep_Click(sender,e);
    }
```

```
        private void Form1_FormClosing(object sender, FormClosingEventArgs e)
{
                DialogResult tb = MessageBox.Show("Bạn thoát hả?","Cảnh báo",

MessageBoxButtons.YesNo,MessageBoxIcon.Warning);
                if (tb == DialogResult.No) e.Cancel = true;
    }
```

Chương 4 : CÁC ĐIỀU KHIỂN ĐẶC BIỆT

4.1. Điều khiển ProgressBar.

Điều khiển ProgressBar dùng để trình bày thời lượng đã thực hiện bằng 3 loại:

1. Blocks: tô từng khối màu xanh từ trái qua phải.
2. Continuous: tô màu xanh từ trái qua phải.
3. Marquee: cuộn từng khối màu xanh từ trái qua phải.

Các thuộc tính(Properties):

Thuộc tính	Mô tả
Maximum	Giới hạn trên của điều khiển.
Minimum	Giới hạn dưới của điều khiển.
Value	Giá trị hiện hành đang xử lý của điều khiển.
Style	Kiểu thể hiện của điều khiển khi thực thi.
Step	Bước nhảy của điều khiển.

Ví dụ:

Như hình bên dưới

(Hình ảnh minh họa sử dụng ProgressBar)

Ví dụ tạo một thanh tiến trình trên form, sau 5 giây sẽ đóng Form (khi thanh tiến trình đã chạy hết), ta thực hiện các bước sau:

Các điều khiển trên Form gồm có

- Thanh ProgressBar
- Điều khiển Timer

Các thuộc tính(Properties):

Thuộc tính	Mô tả
Với ProgressBar	Name: prgThanhChay Dock: Bottom Step: 10 Value:0
Với Timer	Name: tmDongHo Enable: True Interval: 250

Nhấp đôi chuột vào đối tượng Timer để kích hoạt tình huống Tick và quy định mã lệnh như sau:

```
private void tmDongHo_Tick(object sender, EventArgs e) {
        if (prgThanhChay.Value == 100)  {
                tmDongHo.Stop();
                this.Close();
                return;
        }
        prgThanhChay.Value += 5;
}
```

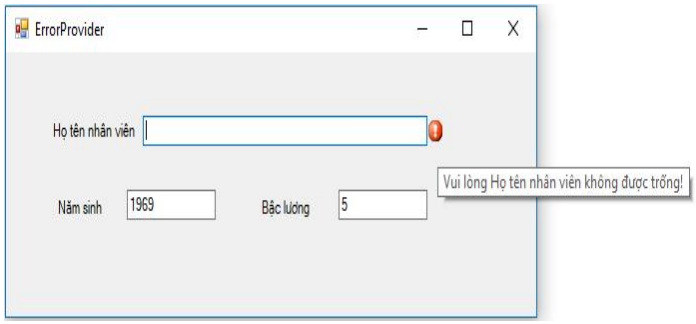
Biên dịch và chạy thử

4.2. Điều khiển ErrorProvider.

Là điều khiển cho phép thông báo một lỗi xảy ra khi nhập dữ liệu vào TextBox có hợp lệ hay không hợp lệ. Để sử dụng điều khiển này, ta gán một câu thông báo lỗi trong thuộc tính Error của điều khiển ErrorProvider ngay tại vị trí của TextBox tương ứng.

Khi có lỗi xảy ra, một dấu hiệu báo lỗi sẽ xuất hiện ngay bên cạnh TextBox đang nhập dữ liệu không hợp lệ. Nếu biết lỗi cụ thể ta đưa trỏ chuột đến dấu hiệu sẽ xuất hiện một thông báo lỗi dưới hình thức một ToolTip.

Ví dụ: như hình dưới



Trường hợp này cài đặt sự kiện Validating cho TextBox Họ tên nhân viên như sau

```
private void txtHoTen_Validating(object sender, CancelEventArgs e)  {
 if (string.IsNullOrEmpty(txtHoTen.Text)  {
        e.Cancel = true;
        txtHoTen.Focus();
       errorProvider1.SetError(txtHoTen,
                        "Vui lòng Họ tên nhân viên không được trống!");
 }
 else  {
    e.Cancel = false;
    errorProvider1.SetError(txtHoTen, null);
 }
}
```

Hoàn toàn tương tự ta có thể cài đặt sự kiện Validating cho hai ô nhập năm sinh và bậc lương.

4.3. Điều khiển ListView.

Điều khiển ListView dùng để trình bày các mục theo dạng danh sách gồm nhiều dòng và nhiều cột với khung hiển thị (View) khác nhau.

Các thuộc tính(Properties):

Thuộc tính	Mô tả
AllowColumnReorder	Cho phép sắp xếp cột trên điều khiển ListView ở chế độ thực thi hay không (gồm hai giá trị True và False). Giá trị mặc định là False: không cho phép.
CheckBoxes	Cho xuất hiện hộp kiểm (CheckBox) bên cạnh từng mục chọn trên điều khiển ListView hay không (True / False).
Columns	Tạo số cột (có Header) và tiêu đề của điều khiển.
FullRowSelect	Gồm hai giá trị: True: cho phép tô màu ứng với dòng của mục được chọn. False: không cho phép. Chỉ tô màu ở ô đầu tiên của dòng. Mặc định là False.
Group	Khai báo nhóm dùng để phân loại các mục sau khi được trình bày trên điều khiển.
GridLines	Tạo các ô lưới trên ListView. Gồm hai giá trị: True: cho phép tạo ô lưới False: không cho phép tạo ô lưới Mặc định là False
LabelEdit	Gồm hai giá trị: True: Cho phép người sử dụng thay đổi chuỗi của các mục chọn. False: Không cho phép. Mặc định là False.
LabelWrap	Cho phép chuỗi của các mục chọn sẽ tự động xuống dòng hay không khi chiều dài không đủ để trình bày (True/False)
LargeImageList	Hiển thị hình ảnh lớn khi sử dụng kèm đối tượng ImageList. Cáchình ảnh chứa danh sách các ImageList được đánh số chỉ mục từ 0 đến n-1 (n là số lượng hình) được sử dụng cho trường hợp thuộc tính View là LargeIcon.

SmallImageList	Hiển thị hình ảnh nhỏ khi sử dụng kèm đối tượng ImageList. Cáchình ảnh chứa danh sách các ImageList được đánh số chỉ mục từ 0 đến n-1 (n là số lượng hình) được sử dụng cho trường hợp thuộc tính View là SmallIcon.
Sorting	Quy định kiểu sắp xếp.
View	Quy định chế độ trình bày tương ứng trên điều khiển, gồm: List, Details, LargeIcon, SmallIcon và Tilte.

Các phương thức(Menthods):

Phương thức	Mô tả
Add	Thêm một mục chọn mới vào danh sách (chuỗi hay đối tượng). Sử dụng phương thức Items.Add theo cú pháp: **Cách 1:** //Khai báo biến item kiểu ListViewItem ListViewItem item; //Khởi tạo item và đưa dữ liệu vào từng SubItems theo thứ tự từ trái sang phải item=new ListViewItem(<Giá trị SubItems[0]>); item.SubItems.Add(<Giá trị SubItems[1]>); … Item.SubItems.Add(<Giá trị SubItems[i]>); … //Đưa item vào ListView TênListView.Items.Add(item); **Cách 2:** //Tạo mảng các SubItems string[] data = {string1, string2, …}; //Khai báo và tạo một Item từ mảng SubItems trên ListViewItem item = new ListViewItem(data); //Đưa Item vừa tạo vào ListView

	TênListView.Items.Add(item);
Clear	Xóa tất cả các mục trong điều khiển ListView. Sử dụng phương thức Items.Clear theo cú pháp: TênListView.Items.Clear();
SelectedItems	Trả về danh sách giá trị các mục được chọn.
CheckedItems	Trả về danh sách các mục được đánh dấu kiểm.
Count	Trả về số mục trong ListView
ArrangeIcon	Sắp xếp các mục hiển thị trong ListView theo giá trị. Gồm các giá trị sau: 
Remove	Xóa một mục chọn ra khỏi danh sách trong ListView. Cú pháp: TênListView.Items.Remove(<Giá trị>) Nếu có nhiều giá trị giống nhau trong danh sách thì chỉ có giá trị đầu tiên bị xóa. Trong trường hợp các mục chọn là đối tượng, ta truyền vào một biến tham chiếu đến mục chọn cần xóa.
RemoveAt	Xóa một mục chọn tại vị trí chỉ định ra khỏi danh sách ListView. Cú pháp: TênListView.Items.RemoveAt(<Vị trí>)
FocusedItem	Dòng đang đươc chọn trong ListView
SubItems[i]	Tập hợp các cột của một mục chọn (dòng) trong ListView. Chỉ số cột i tính từ 0 (là cột đầu tiên)
Focused	Chuyển dòng chọn Ví dụ: chuyển dòng chọn đến dòng có chỉ số i TênListView.Items[i].Focuse=true;

Selected	Chọn dòng Ví dụ: chọn dòng có chỉ số i TênListView.Items[i].Selected=true;

Ví dụ: Các hàm con xử lý xóa, tìm kiếm trong ListView.

Hàm con xử lý xoá các dòng được chọn trong ListView

```
private void LwXoa(ListView lw){
        if(lw.SelectedIndices.Count>0){
                DialogResult Tb=MessageBox.Show("Bạn xoá dữ liệu",
                    MessageBoxButtons.YesNo,MessageBoxIcon.Waning);
                if(Tb==DialogResult.Yes) {
                        while(lw.SelectedIndices.Count>0)
                                lw.Items.RemoveAt(lw.SelectedIndices[0]);
                }
                 lw.SelectedIndices.Clear();
                return;
        }
        MessageBox.Show("Bạn chưa chọn dòng xoá", "Thông báo");
}
```

Khi sử dụng hàm LwXoa. Ta chỉ gọi hàm LwXoa(TênListView).

Hàm con xử lý xoá các dòng được chọn trong ListView

```
private int LwTimKiem(ListView lw,int iSub, string sChuoiTim){
        for(int i=0; i<lw.Items.Count;i++)
                if(lw.Items[i].Subitems[iSub].Text==sChuoiTim)
                        return i;
        return -1;
}
```

Khi sử dụng hàm LwTimKiem.

Ta chỉ gọi hàm LwTimKiem(TênListView,iSub,ChuoiTim).

Ví dụ: Hàm con xử lý di chuyển dòng được chọn trong ListView

//Di chuyển lùi dòng

```
private void LwLuiDong(ListView lw){
        lw.Focus();
        int i=lw.FocusedItem.Index;//lấy chỉ số dòng đang chọn

        if(i>0){ //nếu dòng i không phải là dòng đầu
                i--;
                lw.Items[i].Focused=true;
                lw.Items[i].Selected=true;
        }
}
```

//Di chuyển tăng dòng

```
private void LwTangDong(ListView lw){
        lw.Focus();
        int i=lw.FocusedItem.Index;//lấy chỉ số dòng đang chọn
        if(i<lw.Items.Count-1)//dòng i không phải là dòng cuối
        {
                i++;
                lw.Items[i].Focused=true;
                lw.Items[i].Selected=true;
        }
}
```

//Di chuyển về dòng đầu

```
private void LwDongDau(ListView lw){
        lw.Focus();
        lw.Items[0].Focused=true;
        lw.Items[0].Selected=true;
}
```

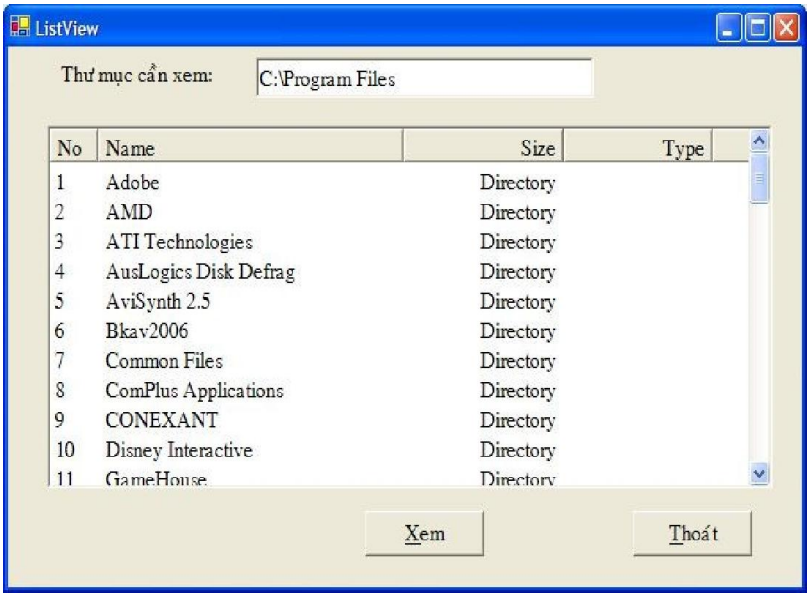
//Di chuyển về dòng cuối

```
private void LwDongCuoi(ListView lw){
        lw.Focus();
        lw.Items[lw.Items.Count-1].Focused=true;
        lw.Items[lw.Items.Count-1].Selected=true;
}
```

Các sự kiện(Events):

Sự kiện	Mô tả
SelectedIndexChanged	Khi người sử dụng thay đổi mục chọn trong điều khiển.
ItemActivate	Khi chọn một mục trong điều khiển.
ItemChecked	Khi nhấn chọn vào hộp kiểm của mỗi mục trong điều khiển.

Một ví dụ về ListView: tạo một Form để hiển thị các thư mục con và tập tin của thư mục chỉ định như sau:




```
private void btnXem_Click(object sender, EventArgs e)
{
    try
    { int i = 0;
        this.listView1.Clear();
        // Khai báo và sử dụng đối tượng thư mục
        DirectoryInfo dir;
        dir = new DirectoryInfo(@txtThuMuc.Text + "\\");
        // Khai báo số cột
        this.listView1.Columns.Add("No", 30,
        HorizontalAlignment.Center);
        this.listView1.Columns.Add("Name", 120,
        HorizontalAlignment.Left); this.listView1.Columns.Add("Size",
        50, HorizontalAlignment.Right);
        this.listView1.Columns.Add("Type", 120,
        HorizontalAlignment.Right);
        // Cho phép chọn hàng
        this.listView1.FullRowSelect = true;
        // Chọn chế độ trình bày
        this.listView1.View = View.Details;
        // Khai báo một mục (dòng) ListView
```

```
        ListViewItem item;
        // Duyệt qua từng thư mục foreach(DirectoryInfo
        SubDir in dir.GetDirectories("*.*"))
        {
            i += 1;
          // Khởi tạo đối tượng ListView
            item = new ListViewItem(i.ToString());
          // Trình bày tên thư mục
            item.SubItems.Add(SubDir.Name);
          // Trình bày thuộc tính của thư mục
            item.SubItems.Add(SubDir.Attributes.ToString());
            // Thêm đối tượng ListViewItem vào điều khiển ListView1
            this.listView1.Items.Add(item); }
        // Duyệt qua từng tập tin
        foreach(FileInfo f in dir.GetFiles("*.*"))
        {
            i += 1;
            // Khởi tạo đối tượng ListView item = new
            ListViewItem(i.ToString());
            // Trình bày tên tập tin
            item.SubItems.Add(f.Name);
            // Trình bày thuộc tính kích thước tập tin
            item.SubItems.Add(f.Length.ToString());
// Trình bày thuộc tính của tập tin
item.SubItems.Add(f.Attributes.ToString());
          // Thêm đối tượng ListViewItem vào điều khiển ListView1
this.listView1.Items.Add(item); }
    }
    catch (DirectoryNotFoundException )  //Nếu thư mục không hợp lệ
    {
          MessageBox.Show("Thu muc nay khong ton tai"); return;
    }
}
```

4.4. Điều khiển TreeView.

Điều khiển TreeView dùng để hiển thị danh sách theo dạng hình cây. Các mục chọn có phân cấp theo từng nút (TreeNode). TreeView là thành phần phức tạp được Microsoft đầu tư công phu nhất. TreeView có thể biểu diễn được hầu như mọi cấu trúc dữ liệu một cách trực quan, từ hệ thống tập tin /thư mục, hệ thống quản trị cơ sở dữ liệu, trình diễn XML, quản lý các thư mục dự án… TreeView quản lý các mục ở dạng nút (TreeNode), trong đó mỗi đối tượng TreeNode lại giữ danh sách các TreeNode con khác.

Các thuộc tính(Properties):

Thuộc tính	Mô tả
CheckBoxes	Xuất hiện các CheckBox bên cạnh từng Node trên điều khiển.
Nodes	Khai báo các Node (có Header) của điều khiển.
FullRowSelect	Gồm hai giá trị: True: cho phép tô màu ứng với hàng của mục được chọn. False: không cho phép. Mặc định là False.
ShowLine	Gồm hai giá trị: True: Cho phép đường viền ứng với từng Node (mặc định). False: Không cho đường viền ứng với từng Node.
LabelEdit	Gồm hai giá trị: True: Cho phép người sử dụng thay đổi chuỗi cua các mục chọn. False: Không cho phép. Mặc định là False.
ShowPlusMinus	Cho xuất hiện dấu + hay – trên mỗi Node hay không (True/False)
ShowRootLine	Cho hiển thị Node gốc hay không (True/False)
ImageList	Đối tượng ImageList chứa danh sách các Image theo số chỉ mục từ 0 đến n-1 (n là số lượng hình) được sử dụng cho trường hợp thuộc tính View là LargeIcon.

ImageIndex	Chỉ mục Image theo số chỉ mục từ 0 đến n-1 ứng với Node.
SelectedImageIndex	Chỉ mục Image theo số chỉ mục từ 0 đến n-1 ứng với Node được chọn.

Các phương thức(Methods):

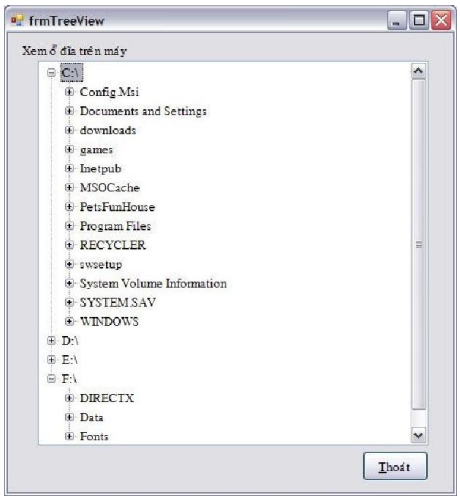
Phương thức	Mô tả
CollapseAll	Thu gọn tất cả các Node trên điều khiển.
ExpandAll	Mở rộng tất cả các Node trên điều khiển.

Các sự kiện(Events):

Sự kiện	Mô tả
AfterCheck	Xảy ra sau khi người sử dụng Check vào CheckBox trên điều khiển.
AfterCollapse	Xảy ra sau khi người sử dụng thu gọn tất cả Node trên điều khiển.
AfterExpand	Xảy ra sau khi người sử dụng mở rộng tất cả Node trên điều khiển.
AfterSelect	Xảy ra sau khi người sử dụng nhấn chọn trên Node sủa điều khiển.
BeforeCheck	Xảy ra trước khi người sử dụng Check vào CheckBox trên điều khiển.
BeforeCollapse	Xảy ra trước khi người sử dụng thu gọn tất cả Node trên điều khiển.
BeforeExpand	Xảy ra trước khi người sử dụng mở rộng tất cả Node trên điều khiển.
BeforeSelect	Xảy ra trước khi người sử dụng nhấn chọn trên Node sủa điều khiển.
Click	Xảy ra khi người sử dụng nhấn chọn một mục (Node) trên điều khiển.

DoubleClick	Xảy ra khi người sử dụng nhấp đôi trên một mục (Node) trên điều khiển.

Một ví dụ về TreeView, cho hiển thị các ổ đĩa có trên máy:




```
using System.IO;  //Trước dòng khai báo Class
 private void GetDisk() {
        // Duyệt qua từng tên ổ đĩa
        foreach( string d in irectory.GetLogicalDrives()){
            // Thêm Node vào điều khiển TreeView
            TreeNode n   = treeView1.Nodes.Add(d);
            n.Tag = d;
            n.Nodes.Add("Nut con ...");
         }
}

private void frmTreeView_Load(object sender, EventArgs e) {
        GetDisk();
 }
private void button1_Click(object sender, EventArgs e) {
        Application.Exit();
```

```
}
private void treeView1_BeforeExpand(object sender,
TreeViewCancelEventArgs e)  {
        try
        {
          if (e.Node.Nodes[0].Text == "Nut con ...")
          {
            e.Node.Nodes.Clear();
            foreach(string FullPath in
            Directory.GetDirectories(e.Node.Tag.ToString()))
            {
              int idx = FullPath.LastIndexOf("\\");
              TreeNode childNode = e.Node.Nodes.Add(FullPath.Substring(idx
              + 1)); childNode.Tag = FullPath;
              childNode.Nodes.Add("Nut con
          ..."); } }
    if ((e.Node.Nodes.Count > 0) && (e.Node.Nodes[0].Text == "Nut con ..."))
            e.Cancel = true;
        }
        catch (Exception ex) { }
      }
```

4.5. Điều khiển DateTimePicker.

Là điều khiển cho phép người sử dụng chọn giá trị thời gian và trình bày giá trị này với định dạng cho trước.

Ví dụ sau sử dụng dụng điều khiển DateTimePicker:

Các thuộc tính(Properties):

Thuộc tính	Mô tả
CustomFormat	Kiểu định dạng do người sử dụng quy định.
Format	Chọn định dạng theo mẫu có sẵn.
MaxDate	Quy định ngày lớn nhất mà điều khiển có thể thể hiện (31/12/9998 là ngày lớn nhất cho phép người sử dụng chọn trên điều khiển).
MinDate	Quy định ngày nhỏ nhất mà điều khiển có thể thể hiện.
Check	Gồm hai giá trị True /False: **True:** CheckBox sẽ được chọn nếu ShowCheckBox là True. **False:** CheckBox sẽ không được chọn
ShowCheckBox	Nếu chọn True thì điều khiển CheckBox sẽ xuất hiện bên cạnh giá trị, ngược lại sẽ không xuất hiện.
ShowUpDown	Nếu giá trị là False, cho phép người sử dụng chọn giá trị thời gian theo hình thức DropDown (như hình trên), ngược lại cho chọn thời gian theo từng thành phần ngày, tháng năm.
Value	Gán hay lấy giá trị thời gian trên điều khiển.

4.6. Điều khiển Timer.

Là đối tượng dùng để xử lý các sự kiện thời gian. Khi ta muốn trong chương trình, qua một khoảng thời gian sẽ tự động làm một điều gì đó, ta sẽ đặt đối tượng Timer lên Form. Timer không phải là một điều khiển, nó không xuất hiện trên Form như một cửa sổ. Khi được đặt lên Form thì cứ sau một khoảng thời gian nhất định do ta quy định, Timer sẽ phát ra một sự kiện thời gian cho Form. Ta sẽ viết lệnh để quy định mỗi lần sự kiện thời gian xảy ra Form sẽ làm gì.

Các thuộc tính(Properties):

Thuộc tính	Mô tả
Enabled	Gồm hai giá trị: **True**: cho phép Timer phát ra sự kiện thời gian trên Form. **False:** không cho phép Timer phát ra sự kiện thời gian trên Form.
Interval	Là một giá trị số dùng để quy định sau bao nhiêu lâu thì phát ra sự kiện thời gian. Đơn vị tính bằng 1/1.000 giây (mili giây). Ta có thể đặt giá trị cho nó khi thiết kế hay khi thực thi. Khi giá trị của Interval = 0 nghĩa là không cho phép Timer hoạt động.

Để quy định xử lý sự kiện thời gian cho Timer, ta quy định tại tình huống/sự kiện Tick của đối tượng.

Các phương thức(Menthods):

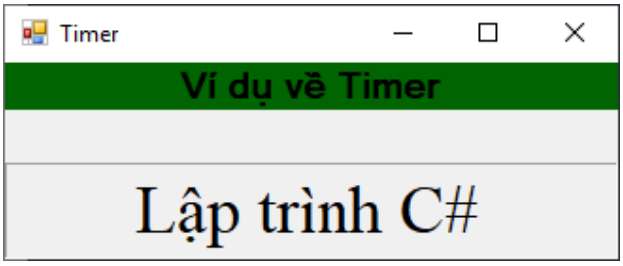
Phương thức	Mô tả
Stop	Tắt xử lý thời gian cho Timer.
Start	Bật xử lý thời gian cho Timer.

Các sự kiện(Events):

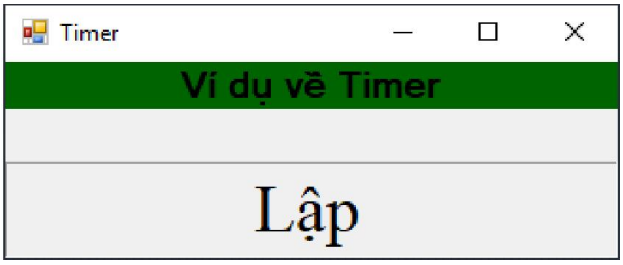
Sự kiện	Mô tả
Tick	Xảy ra khi bật(Start) xử lý thời gian cho Timer.

Ví dụ:

Thiết kế và viết chương trình xử lý chạy chữ "Lập trình C#". Khi thực hiện chương trình chuỗi chữ xuất hiện từ phải sang trái cho đến khi hết chuỗi.



Khi thực hiện chương trình xử lý như hình bên dưới



Thiết kế giao diện gồm có

- Một Label dùng làm tiêu đề
- Một Label dùng để xuất chuỗi như yêu cầu
- Một Timer dùng để bật đồng hồ hệ thống và xử lý chuỗi xuất

Thiết kế và cài đặt thuộc tính

Điều khiển	Mô tả
Với Label tiêu đề	Text: Ví dụ về Timer
Với Label xuất chuỗi	Name: lblXuat AutoSize: False Dock: Bottom Text: để trống TextAlign: MiddleCenter
Với Timer	Name: tmDongHo Enable: True Interval: 200

Clich đúp vào điều khiển Timer, xử lý sự kiện DongHo_Tick như sau

Khai báo các biến sử dụng

```
string tieude = "Lập trình C#";
int i = 0;
```

Sự kiện xử lý của DongHo

```
private void tmDongHo_Tick(object sender, EventArgs e)   {
        if (i == tieude.Length)   {
                tmDongHo.Stop();
                return;
        }
        lblXuat.Text+=tieude[i];
        i++;
}
```

4.7. Điều khiển TabControl.

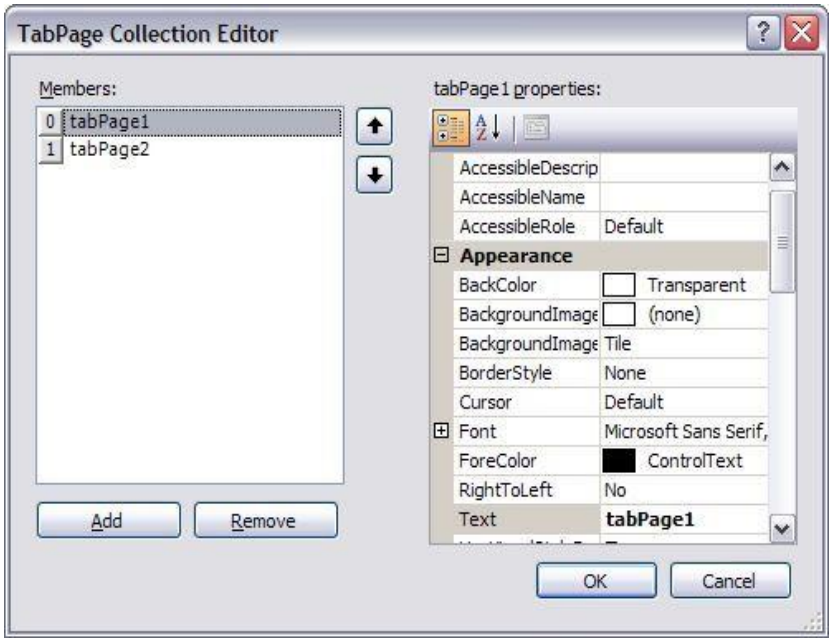
Là điều khiển chứa danh sách các đối tượng TabPage (Thẻ/Trang) cho phép thêm các điều khiển khác vào từng TabPage. Có dạng như sau:

Các thuộc tính(Properties):

Thuộc tính	Mô tả
Alignment	Vị trí xuất hiện của Tab(ngăn chọn) trên điều khiển TabControl ứng với Top, Bottom, Left, Right
ImageList	Chứa danh sách hình ảnh cho phép chọn hình ảnh ứng với từng tab
Appearance	Hình thức xuất hiện của Tab như Normal, Buttons, FlatButtons
Multiline	True sắp xếp danh sách Tab trên nhiều hàng

Mặc định, khi mới được tạo, TabControl sẽ có hai TabPage, để thêm mới một TabPage (hay xóa bỏ) ta có thể thực hiện một trong các cách sau:

Nhấn vào nút ... tại thuộc tính TabPages (Collection) trong cửa sổ Properties, sẽ xuất hiện hộp thoại sau:



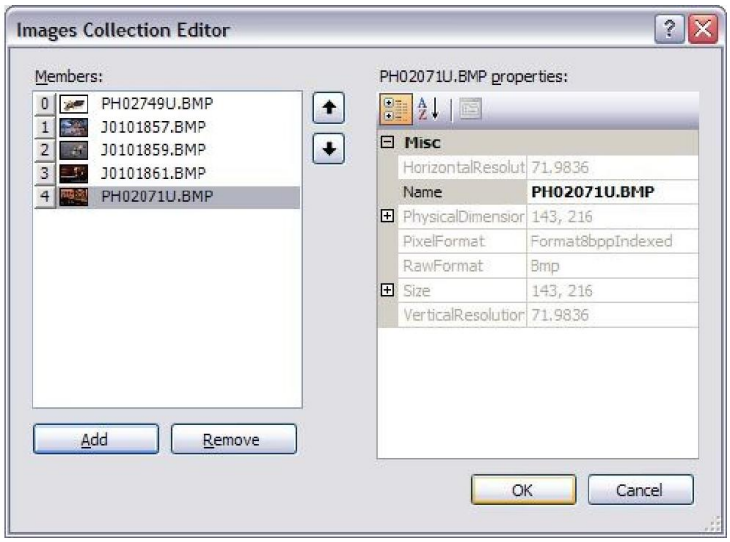
- Nhấn nút để them mới và quy định các đặt để trong cửa sổ bên cạnh.
- Nhấn nút Remove để xóa bỏ TabPage đang được chọn.
- Nhấn OK để chấp nhận (hoặc Cancel để hủy thao tác).

Chương 5 : ĐIỀU KHIỂN XÂY DỰNG MENU

5.1. Điều khiển ImageList.

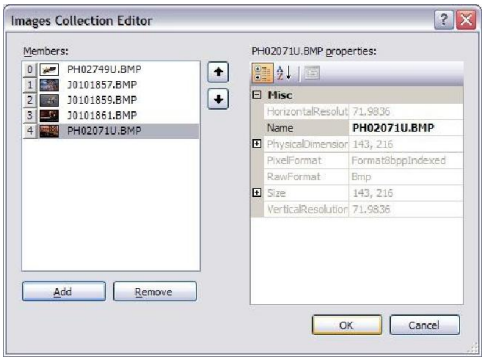
ImageList dùng để lưu trữ danh sách hình ảnh để có thể quản lý dễ dàng hơn, nói đơn giản hơn ImageList đưa hình ảnh vào trong mảng và chúng ta có thể thao tác trên hình ảnh thông qua chỉ số, tất cả sẽ cùng 1 tên chung chứ không phải là từng hình ảnh rời rạc. ImageList không phải là Control hiển thị trực tiếp nên không thể thấy trên giao diện Form, thay vào đó nó sẽ nằm ở thanh công cụ bên dưới của VS 2008.

Để thêm hình ảnh vào ImageList có thể thao tác trong giao diện thiết kế hoặc cũng có thể thêm bằng Code thực thị. Trong giao diện thiết kế ta chỉ cần nhấp vào Control ImageList để hiển thị hộp thoại quản lý hình ảnh, tại đây có thể thêm, bớt hình ảnh. Lưu ý: Hình ảnh trong ImageList chỉ giới hạn ở kích **256*256** pixel. Có thể sử dụng hình ảnh màu 32bit. Nếu muốn add hình ảnh vào ImageList bằng Code thực thi thì cần phải đưa đường dẫn trực tiếp của hình ảnh vào phương thức add() của ImageList(các hình sẽ có số chỉ mục Index bắt đầu từ 0).



Các thuộc tính(Properties):

Thuộc tính	Mô tả
ColorDepth	Chế độ màu hiển thị ảnh Gồm các giá trị: Depth4bit: chế độ ảnh 4bit . Depth8bit: chế độ ảnh 8bit Depth16bit: chế độ ảnh 16bit

	Depth24bit: chế độ ảnh 24bit Depth32bit: chế độ ảnh 32bit
Image	Tập hợp ảnh trong điều khiển ImageList. Để đưa ảnh vào ImageList ta chọn mục (Collection) 
ImageSize	Kích thước ảnh hiển thị theo dạng Pointer theo dạng dài và rộng. Mặc định là 16,16

Ví dụ: Sử dụng PictureBox

Viết chương trình đổi năm dương lịch sang năm âm lịch có kèm hình ảnh 12 con giáp tương ứng.

Năm âm lịch dựa vào Can Chi để tính. Để đổi năm dương lịch sang năm âm lịch dựa vào bảng sau.

Bảng tính Can

NămDL%10	0	1	2	3	4	5	6	7	8	9
Can	Canh	Tân	Nhâm	Quý	Giáp	Ất	Bính	Đinh	Mậu	Kỷ

Bảng tính Chi

NămDL%12	0	1	2	3	4	5	6	7	8	9	10	11
Chi	Thân	Dậu	Tuất	Hợi	Tý	Sửu	Dần	Mão	Thìn	Tỵ	Ngọ	Mùi

Chẳng hạn: nhập vào ô năm dương lịch là 2019. Sau đó nhấn nút View thì năm âm lịch xuất hiện trong ô năm âm lịch là Kỷ Hợi.

Dựa vào bảng tính Can và Chi ở trên ta tạo các mảng Can và chi như sau

```
string[] Can = {"Canh", "Tân", "Nhâm", "Quý", "Giáp", "Ất",
                "Bính", "Đinh", "Mậu", "Kỷ"};
string[] Chi = {"Thân", "Dậu", "Tuất", "Hợi", "Tý", "Sửu",
                "Dần", "Mão", "Thìn", "Tỵ", "Ngọ", "Mùi"};
```

Các hình ảnh của các chi tương ứng được Add vào điều khiển ImageList theo thứ tự như khai báo chi ở trên. Ta có thể khai báo đoạn chương trình trong tình huống Click của nút "View" như sau:

```
private void btnView_Click(object sender, EventArgs e){
        int namDL=int.Parse(txtNamDL.Text);
        txtNamAL.Text=Can[namDL%10]+ " " +
                        Chi[namDL%12];
        picHinh.Image=Imglst.Images[namDL%12];
}
```

5.2. Điều khiển MenuStrip.

Khi chương trình có nhiều Form, ngoài việc thiết kế trên các Form, quy định Form chính, ta cần phải tạo sự liên kết giữa Form chính và các Form cần thiết khác phục vụ cho các chức năng của chương trình. Do vậy, việc tạo một trình đơn (trình đơn của chương trình - MenuBar) trên Form chính để kết nối giữa các Form là cần thiết.

Khi tạo bất kỳ loại Menu hệ thống nào cho ứng dụng, ta phải xác định được những chức năng gì cần cho hệ thống Menu đó và ta muốn tổ chức những chức năng này như thế nào. Visual C# cho phép chúng ta tạo nhanh một hệ thống MenuBar và PopUp dễ dàng thông qua điều khiển MenuStrip.

Các thuộc tính(Properties):

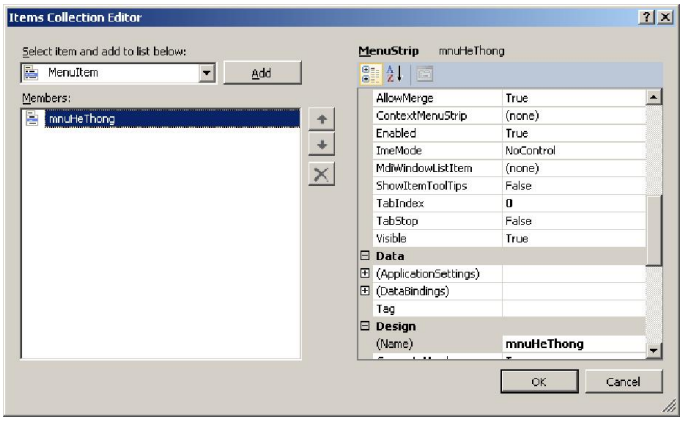
Thuộc tính	Mô tả
Name	Tên thuộc tính điều khiển MenuStrip
Text	Tên hiển thị Menu ngang
Items	Tạo Menu ngang và PopUp. Để tạo Menu ta chọn ...

Các sự kiện(Events):

Sự kiện	Mô tả
Click	Xảy ra khi Click vào 1 dòng trong Menu hay nhấn tổ hợp phím tắt..

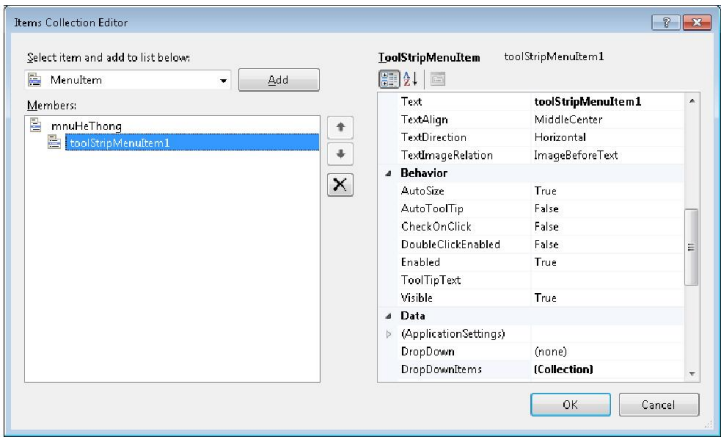
Đầu tiên ta tạo hệ thống Menu Bar với các thao tác sau

- Tạo Menu Bar
  - Vào Properties của MenuStrip chọn Items Click vào ...
  - Xuất hộp thoại như sau



Hình 1

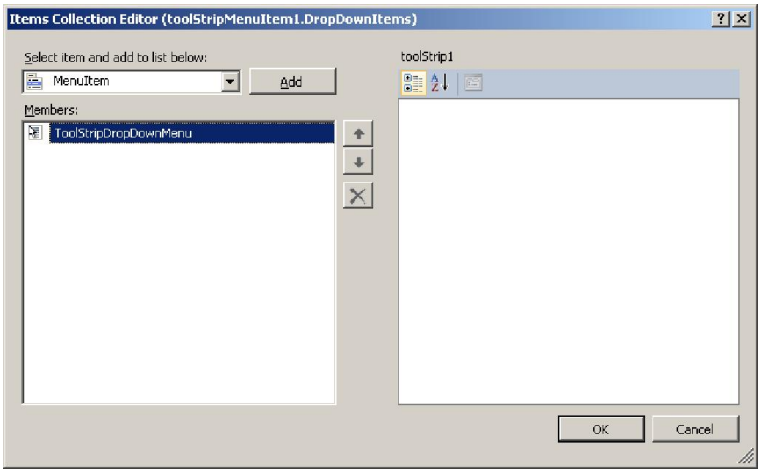
- Chọn nút Add để thêm vào chức năng nhóm lệnh



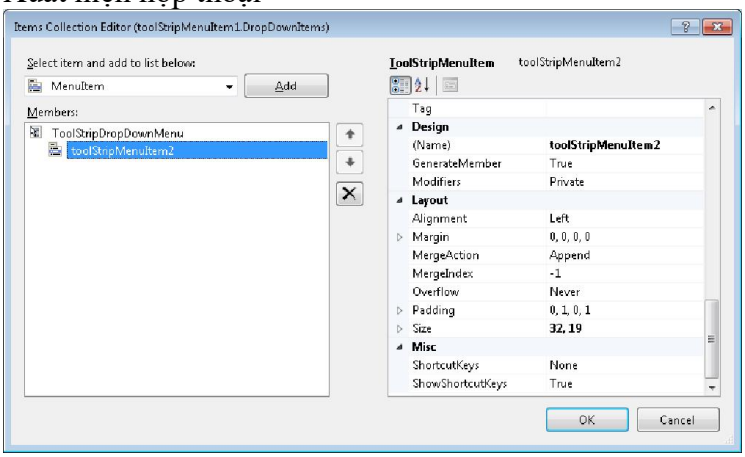
Hình 2

- Đặt các thuộc tính của nhóm lệnh toolStripMenuItem

➢ Tạo Menu PopUp cho từng mục trên Menu Bar.
- Trên hộp thoại Hình 1 chọn mục nhóm lệnh trên Thanh Menu Bar vừa tạo
- Ở cửa sổ bên phải chọn DropDownItems chọn 
- Xuất hiện hộp thoại



- Click vào nút Add thêm các mục lệnh PopUp trong nhóm lệnh trên Menu Bar
- Xuất hiện hộp thoại



- Đặt các thuộc tính của nhóm lệnh toolStripMenuItem

Các thuộc tính của ToolStripMenuItem:

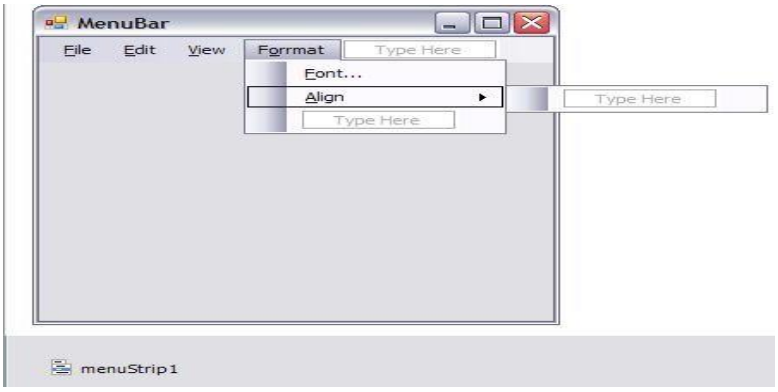
Thuộc tính	Mô tả
Checked	Gồm hai giá trị: True: cho phép hiển thị dấu trước Menu Item. False: không cho phép dấu trước Menu Item.

DefaultItem	Gồm hai giá trị: **True:** Quy định mục chọn mặc định của Menu hoặc Menu con. **False:** Không quy định mục chọn mặc định của Menu hoặc Menu con. Giá trị mặc định của thuộc tính này là False. **Khi giá trị của thuộc tính này = True, mục chọn này sẽ được in đậm.**
Enabled	Cho phép người sử dụng xử lý hay vô hiệu hóa mục chọn. Giá trị mặc định của thuộc tính này = True, cho phép người sử dụng xử lý mục chọn. Ngược lại, nếu = False sẽ vô hiệu hoá mục chọn.
DisplayStyle	Quy định cách thể hiện các mục chọn, gồm các giá trị sau: **None:** không thể hiện gì cả. **Text:** chỉ thể hiện chuỗi văn bản. **Image:** chỉ thể hiện hình ảnh quy định. **ImageAndText:** thể hiện hình ảnh và chuỗi văn bản.
ShortcutKeys	Quy định phím tắt của mục chọn
ShowShortcutKeys	Quy định phím tắt của mục chọn có được phép hiển thị hay không (True/False). Giá trị mặc định là True.
Text	Quy định nội dung hiển thị trong mục chọn. Giá trị mặc định là chuỗi rỗng.
Image	Quy định hình ảnh hiển thị trên mục chọn.
TextAlign	Canh lề cho chuỗi văn bản của mục chọn.
TextDirection	Quy định hướng văn bản mục chọn.

Sau khi đã tạo MenuStrip, để gán vào Form nào, ta quy định tại thuộc tính Menu của Form đó.

Ví dụ:

Để tiến hành tạo một hệ thống Menu cho chương trình như hình bên dưới.

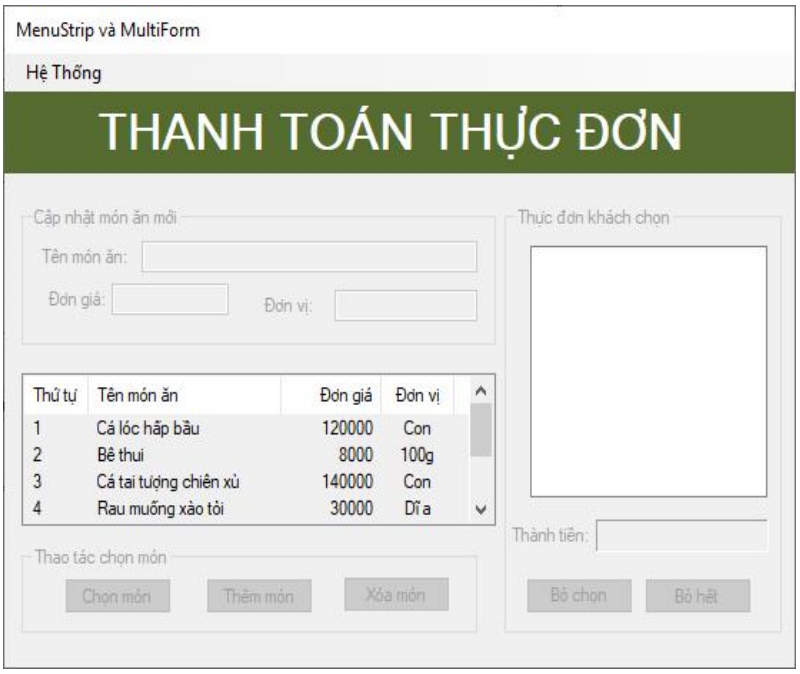


Cách làm theo thứ tự hướng dẫn ở trên

2. <u>Viết code:</u>

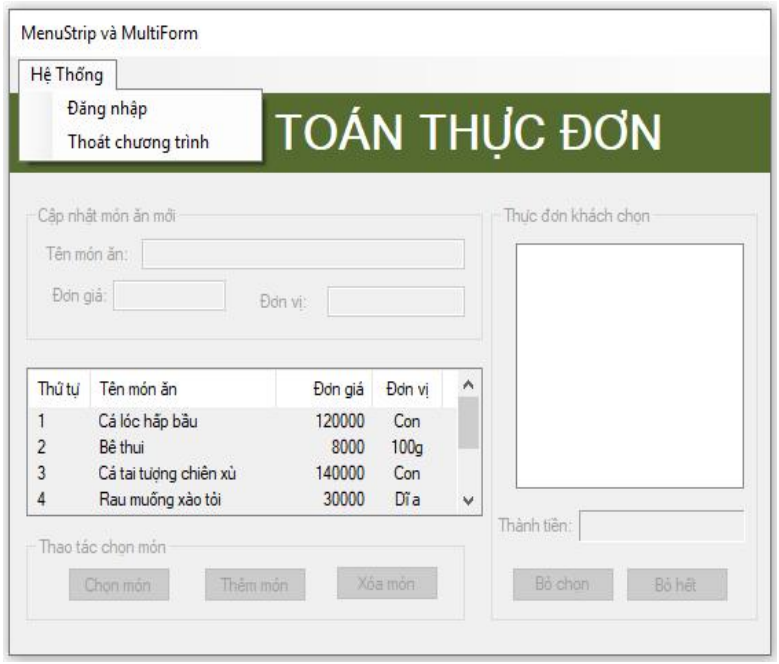
Sau đó quy định lệnh xử lý cho các mục chọn bằng cách nhấp đúp chuột lên mục chọn để mở cửa sổ viết lệnh.

<u>Ví dụ:</u> Thiết kế và viết chương trình xử lý ứng dụng có giao diện sau

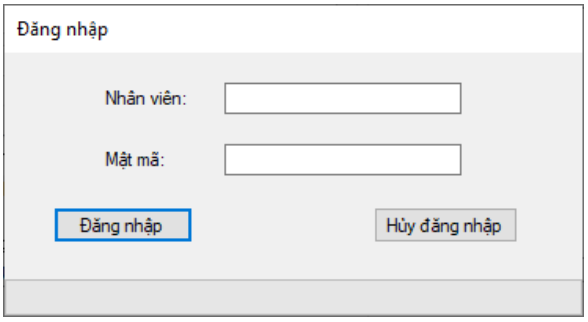


Khi thực hiện chương trình các Group trên chương trình bị khoá. Các group này được mở khoá khi đăng nhập thành công.

Trong giao diện có hệ thống Menu PopUp gồm có 2 chức năng là: Đăng nhập và Thoát chương trình có giao diện như sau.



- Nếu chọn chức năng Đăng nhập thì xuất hiện Form đăng nhập có giao diện



  - Nếu đăng nhập thành công thì ở khoá các Group cho phép thay đổi các dữ liệu, đồng thời chức năng Đăng nhập thay bằng chức năng Đăng xuất.
  - Nếu chọn Đăng xuất thì các Group khoá lại như ban đầu.

- Nếu chọn chức năng Thoát chương trình thì thoát hẳn chương trình.

Các đối tượng trên Form đăng nhập gồm có

- ✓ Hai TextBox dùng nhập tên và mật mã của nhân viên
- ✓ Hai nút nhấn Button thực hiện Các chức năng.

- ✓ Thanh ProgressBar
- ✓ Timer dùng để bật thời gian cho thanh ProgressBar

Thiết đặt các Properties:

Đặt tên cho các đối tượng trên Form đăng nhập như sau:

Các điều khiển	Thuộc tính
Với TextBox nhân viên	Name: txtten MaxLength: 100
Với TextBox mật mã	Name: txtma MaxLength: 100
Với Timer	Name: tmDongHo Enable: False
Với thanh ProgressBar	Name: prgchay Dock: Bottom
Với Button huỷ đăng nhập	Name: btnhuy Text: Huỷ đăng nhập
Với Button đăng nhập	Name: btndangnhap Text: Đăng nhập

Các sự kiện trên Form Đăng nhập

```
private void btnhuy_Click(object sender, EventArgs e)      {
        txtma.Clear();
        txtten.Clear();
        this.Close();
}
```

```
private void btndangnhap_Click(object sender, EventArgs e)      {
    if (txtten.TextLength ==txtma.TextLength && txtma.Text !="")    {
        tmDongHo.Start();
```

```
            s = txtten.Text.ToUpper();
            return;
        }
        MessageBox.Show("Đăng nhập sai!","Thông báo");
        txtma.Clear();
        txtten.Clear();
        txtten.Focus();
    }
```

```
    private void tmDongHo_Tick(object sender, EventArgs e)     {
            if (prgchay.Value == 100)      {
                tmDongHo.Stop();
                this.Close();
                return;
            }
            prgchay.Value += 1;
    }
```

Các sự kiện trên thanh Menu hệ thống

```
    private void mnuthoat_Click(object sender, EventArgs e)     {
            if (mnudangnhap.Text == "Đăng xuất")   {
                MessageBox.Show("Vui lòng đăng xuất trước khi thoát");
                return;
            }
            Application.Exit();
    }
```

```
    private void mnudangnhap_Click(object sender, EventArgs e)    {
        if (mnudangnhap.Text == "Đăng nhập")    {
            Form fr=new frmletan();
            fr.ShowDialog();
```

```
            if(fr.Controls["txtten"].Text==fr.Controls["txtma"].Text
                && fr.Controls["txtma"].Text!="")    {
                    mnudangnhap.Text = "Đăng xuất";
                    grpchonmon.Enabled = true;
                    grpmonmoi.Enabled = true;
                    grpthucdon.Enabled = true;
                    lwmon.Enabled = true;
            }
        }
        else
        {
            grpchonmon.Enabled = false;
            grpmonmoi.Enabled = false;
            grpthucdon.Enabled = false;
            lwmon.Enabled = false;
            mnudangnhap.Text = "Đăng nhập";
        }
    }
```

Các sự kiện trên Form MenuStrip và MultiForm

Khai báo biến

```
float tong = 0;
```

```
    private void frmMenuStrip_Load(object sender, EventArgs e)    {
            grpchonmon.Enabled = false;
            grpmonmoi.Enabled = false;
            grpthucdon.Enabled = false;
            lwmon.Enabled = false;
    }
```

```
private void btnbohet_Click(object sender, EventArgs e)     {
        lstThucDon.Items.Clear();
        lblthanhtien.Text = "";
        tong = 0;
}
```

```
private void btnthoat_Click(object sender, EventArgs e)   {
        Application.Exit();
}
```

```
private void danhso()       {
        for (int i = 0; i < lwmon.Items.Count; i++)
                lwmon.Items[i].SubItems[0].Text = (i + 1).ToString();
}
```

```
private void btnxoa_Click(object sender, EventArgs e)       {
        if (lwmon.SelectedIndices.Count>0)      {
                DialogResult tb = MessageBox.Show("Bạn đồng ý loại món
                                ăn này...?","cảnh báo",
                                MessageBoxButtons.OKCancel,
                                MessageBoxIcon.Warning);
                if (tb == DialogResult.OK)   {
                        while(lwmon.SelectedIndices.Count>0)
                                lwmon.Items.RemoveAt(lwmon.SelectedIndices[0]);
                danhso();
        }
        lwmon.SelectedIndices.Clear();
    }
}
```

```
private void btnthem_Click(object sender, EventArgs e)      {
    if (txtgia.TextLength != 0 && txttenmon.TextLength != 0
                                && txtdonvi.TextLength!=0)      {
            ListViewItem dong = new ListViewItem((lwmon.Items.Count +
                                                        1).ToString());
            dong.SubItems.Add(txttenmon.Text);
            dong.SubItems.Add(txtgia.Text);
            dong.SubItems.Add(txtdonvi.Text);

            lwmon.Items.Add(dong);

            txtdonvi.Clear();
            txtgia.Clear();
            txttenmon.Clear();
             return;
    }
    MessageBox.Show("Bạn nhập chưa đầy đủ dữ liệu","Thông báo");

    if (txttenmon.TextLength == 0)    {
            txttenmon.Focus();
            return;
    }
    if (txtgia.TextLength == 0)  {
             txtgia.Focus();
            return;
    }
    if (txtdonvi.TextLength == 0)  {
            txtdonvi.Focus();
            return;
    }
}
```

```
private void txtgia_KeyPress(object sender, KeyPressEventArgs e)    {
        if (char.IsLetter(e.KeyChar) || char.IsPunctuation(e.KeyChar) ||
            char.IsSymbol(e.KeyChar) || Char.IsWhiteSpace(e.KeyChar))
                e.Handled = true;
}

private void txttenmon_KeyPress(object sender, KeyPressEventArgs e)  {
        if (char.IsNumber(e.KeyChar) || char.IsPunctuation(e.KeyChar) ||
                                            char.IsSymbol(e.KeyChar))
                e.Handled = true;
}

private void txtdonvi_KeyPress(object sender, KeyPressEventArgs e)   {
        if (char.IsWhiteSpace(e.KeyChar) || char.IsSymbol(e.KeyChar) ||
                                        char.IsPunctuation(e.KeyChar))
                e.Handled = true;
 }
```

```
private void btnchon_Click(object sender, EventArgs e)   {
   if (lwmon.FocusedItem != null)    {
        lstThucDon.Items.Add(lwmon.FocusedItem.SubItems[1].Text);
        tong += float.Parse(lwmon.FocusedItem.SubItems[2].Text);
        lblthanhtien.Text = tong.ToString();
   }
 }
```

```
private int tim(ListView lw, string s)       {
        for (int i = 0; i < lw.Items.Count; i++)
                if (lw.Items[i].SubItems[1].Text == s) return i;
        return -1;
}
```

```
private void btnbochon_Click(object sender, EventArgs e)   {
    if (lstThucDon.SelectedIndex != -1)     {
        int i = tim(lwmon, lstThucDon.
                                Items[lstThucDon.SelectedIndex].ToString());
        lstThucDon.Items.RemoveAt(lstThucDon.SelectedIndex);

          float tien=(float.Parse(lblthanhtien.Text) –
                      float.Parse(lwmon.Items[i].SubItems[2].Text));
          if (tien != 0) lblthanhtien.Text = tien.ToString();
                  else lblthanhtien.Text = "";
            }
  }
```

5.3. Điều khiển ContextMenuStrip.

Cũng như MenuStrip, ta chọn điều khiển ContextMenuStrip vẽ lên Form. Các thao tác thêm và thiết lập các MenuItemStrip cho trình đơn ngữ cảnh giống như làm việc với MenuStrip.

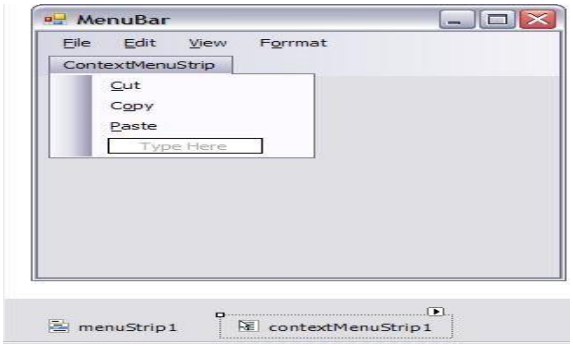
Ta có thể gán trình đơn ngữ cảnh cho một đối tượng trong qúa trình thiết kế hay thông qua lệnh khi thực thi. Trong quá trình thiết kế, chọn thuộc tính ContextMenu từ cửa sổ thuộc tính của đối tượng.

Ví dụ:

Ta muốn khi nhấn phải lên Form trong lúc thực thi sẽ xuất hiện một Menu ngữ cảnh, ta thực hiện các bước sau:

Bước 1: Chọn điều khiển ContextMenuStrip trên ToolBox vẽ lên Form

Bước 2: Tạo các mục chọn tương tự như MenuStrip (nhưng chỉ có các mục chọn theo chiều dọc).



Bước 3: Quy định lệnh xử lý cho các mục chọn bằng cách nhấp đúp lên mục chọn để mở cửa sổ viết lệnh.

Bước 4: Quy định tại thuộc tính ContextMenuStrip của Form là tên của trình đơn ngữ cảnh vừa tạo (là xong).

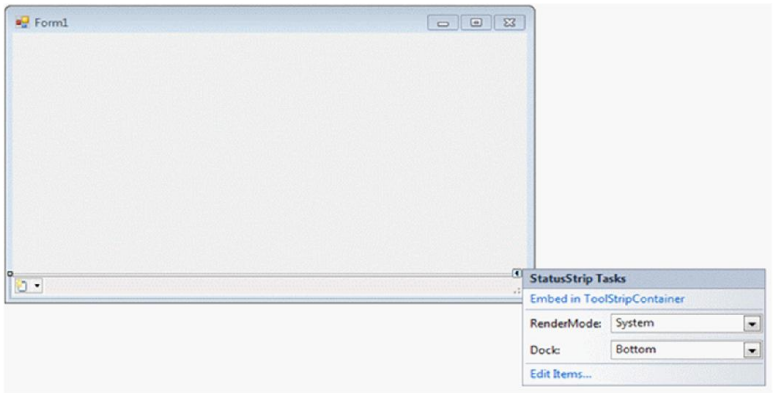
Sau khi thiết kế hệ thống Menu ngữ cảnh, ta phải viết lệnh xử lý cho các mục chọn bằng cách nhấp đúp chuột lên mục chọn để mở cửa sổ viết lệnh. Vì hệ thống ContextMenuStrip thông thường chứa các lệnh thường được sử dụng có trên MenuStrip, nên ta có thể quy định lệnh xử lý bằng cách gọi thực hiện các phương thức đã quy định trên các mục chọn của MenuStrip.

5.4. Điều khiển StatusStrip.

Điều khiển này dùng hiển thị thông tin về trạng thái của chương trình. Nó thường ở dưới cùng của một cửa sổ.

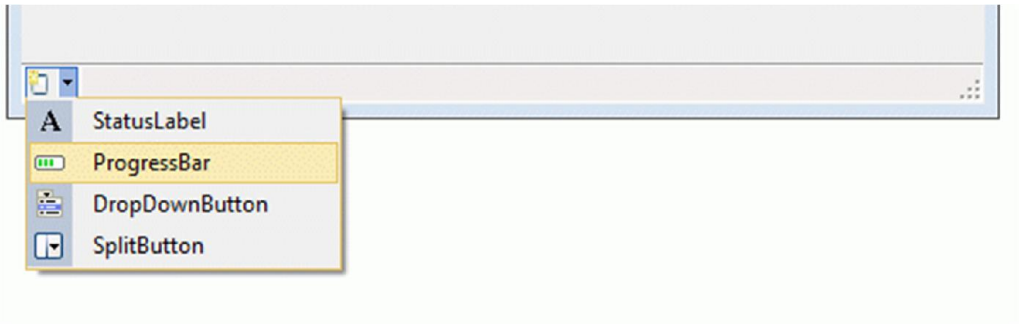
Ví dụ

Để xây dựng điều khiển StatusStrip tại thời điểm thiết kế, chỉ cần kéo và thả điều khiển StatusStrip từ Hộp công cụ vào Biểu mẫu. Điều khiển StatusStrip mặc định được neo ở dưới cùng của Biểu mẫu như trong Hình 1.



Hình 1

Khi thiết kế cũng cho phép bạn thêm một số điều khiển vào điều khiển StatusStrip bao gồm StatusLabel, ProgressBar, DropDownButton và SplitButton. Nếu bạn bấm vào biểu tượng thả xuống nhỏ, bạn sẽ thấy bốn trong số các điều khiển này trong danh sách như trong Hình 2. Ngay sau khi bạn chọn một trong các điều khiển này, nó sẽ được thêm vào điều khiển StatusStrip.



Hình 2

5.5. Điều khiển ToolStrip.

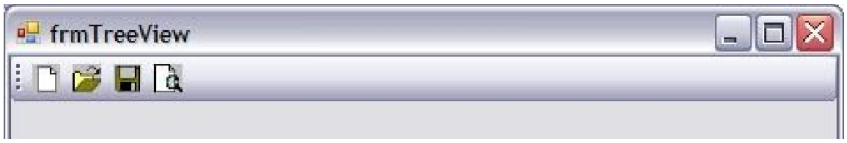
Thông thường, trong các ứng dụng, ngoài trình đơn (MenuBar) để thực hiện các chức năng của chương trình, ta thấy còn có thêm các thanh công cụ chứa các nút lệnh với hình ảnh gợi nhớ, trực quan cho ta thực hiện nhanh các chức năng của chương trình mà không cần phải chọn lệnh trên trình đơn.

Điều khiển ToolStrip là điều khiển dùng để chứa các điều khiển ToolStripItem như: ToolStripButton(nút lệnh), ToolStripComboBox(danh sách ComboBox),

ToolStripSplitButton(danh sách các nút), ToolStripLabel(nhãn), ToolStripSeperator(đường phân cách), …

Ví dụ:

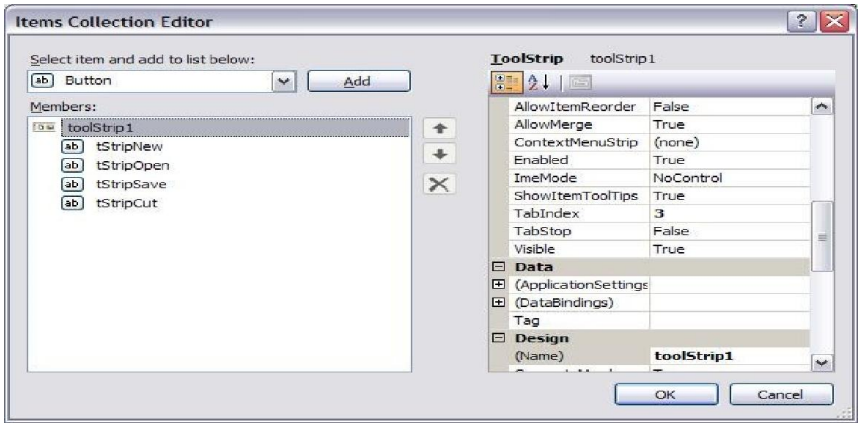
Để tạo một thanh công cụ như hình sau:



Ta thực hiện tuần tự các bước sau:

Bước 1: Chọn điều khiển ToolStrip trong ToolBox vẽ lên Form (giao diện chính của chương trình).

Bước 2: Nhấn vào nút ... tại thuộc tính Items (Collection), sẽ xuất hiện hộp thoại sau:



Bước 3: Chọn kiểu ToolStripItem trong hộp danh sách "Select item and add to list below".

Bước 4: Nhấn nút Add để thêm mục chọn vào ToolBar.

Bước 5: Quy định kiểu hiển thị tại thuộc tính DisplayState, chuỗi hiển thị tại thuộc tính Text, hình ảnh hiển thị tại thuộc tính Image…

Lặp lại các bước từ 3 đến 5 để thêm các mục chọn khác..

Bước 6: Dùng các nút và để sắp xếp lại thứ tự cho các mục chọn.

Bước 7: Nhấn OK để kết thúc.

Quy định lệnh thi hành cho các mục chọn trên thanh công cụ tương tự như quy định lệnh trên trình đơn.

Chương 6: ĐIỀU DIALOG VÀ MESSAGEBOX

Trong các ứng dụng hiện nay như Winword, Excel, Access,… để thao tác mở File, lưu File, định dạng Font chữ hay quy định màu sắc… các ứng dụng đã cung cấp cho ta các tiện ích như các hộp thoại, cửa sổ cho phép ta chọn đường dẫn lưu hay mở tập tin, chỉ định Font chữ hay chọn màu sắc và chỉ khi đóng các hộp thoại đó thì mới có thể tiếp tục thực hiện chức năng khác của chương trình. Các công cụ đó là những điều khiển có sẵn trong Windows và hiện tại Visual Studio cũng sẵn sàng cung cấp cho người viết ứng dụng những điều khiển như vậy, những điều khiển đó được gọi là các hộp thoại thông dụng hay thông thường (Common Dialog).

6.1. Điều khiển MessageBox.

MessageBox là hộp hội thoại thông báo cho người dùng khi muốn xác nhận một lệnh thực thi quan trọng nào đó hay hướng dẫn người dùng thao tác đúng khi nhập một dữ liệu. *MessageBox* là một lớp (class) nằm trong *System.Windows.Forms* có một phương thức **Show** để hiển thị thông báo. Có rất nhiều kiểu thông báo, bạn có thể điều chỉnh nội dung thông báo, tiêu đề, các nút OK-Cancel, biểu tượng, v.v...

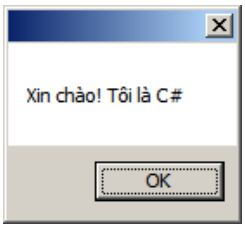
MessageBox.Show("Xin chào! Tôi là C#");

Đây là kiểu thông báo đơn giản nhất, chỉ có nội dung và nút OK, chưa bao gồm biểu tượng, tiêu đề, v.v..

Code cho sự kiện Click Button1

```
private void button1_Click(object sender, EventArgs e)  {
        MessageBox.Show("Xin chào! Tôi là C#");
}
```

Khi nhấn nút Button 1 thì hộp thoại xuất hiện là.



Để có tiêu đề ta thêm 1 tham số chuỗi truyền vào phương thức như sau:

MessageBox.Show("Xin chào! Tôi là C#","Thông báo");

Để cài đặt thêm tiêu đề thông báo cho hộp thoại, ta cũng them 1 tham số kiểu enum là ***MessageBoxButtons.<Loại nút>.***

Các loại nút có sẵn bao gồm: ***AbortRetryIgnore, OK, OKCancel, RetryCance, YesNo, YesNoCancel.***

Ví dụ:

Để có tiêu đề ta thêm 1 tham số chuỗi truyền vào phương thức như sau:

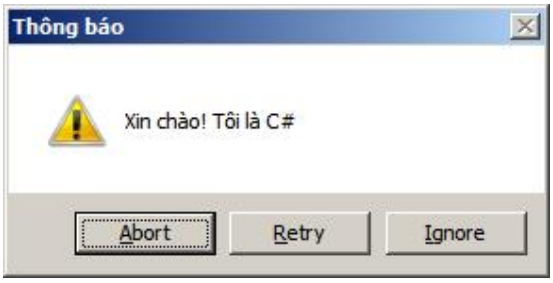
MessageBox.Show("Xin chào! Tôi là C#","Thông báo",
MessageBoxButton. AbortRetryIgnore);

Để xử lý các sự kiện khi nhấn vào các nút này mình sẽ hướng dẫn bên dưới. Để thêm vào icon ta thêm tham số kiểu enum là **MessageBoxIcon.<loại icon>**, có nhiều loại nhưng phổ biến là **Warning** (tam giác vàng có dấu chấm than), **Error** (hình tròn đỏ có chữ X), **Information** (hình tròn xanh lam có chữ i), **Question** (hình tròn lam có dấu chấm hỏi).

Ví dụ:

```
private void button1_Click(object sender, EventArgs e)    {
        MessageBox.Show("Xin chào! Tôi là C#", "Thông báo",
                        MessageBoxButtons.AbortRetryIgnore,
```

```
                                        MessageBoxIcon.Warning);
    }
```



Còn rất nhiều tùy chọn khác các bạn có thể tự khám phá. Bây giờ sẽ hướng dẫn các bạn xử lý sự kiện khi click vào một button trên MessageBox. Giả sử bạn có một Form, và bạn muốn khi người dùng nhấp vào nút Close trên thanh tiêu đề thì sẽ có thông báo hỏi người dùng có muốn thoát chương trình. Nếu người dùng chọn Yes, chương trình sẽ kết thúc, chọn No sẽ không tắt chương trình.

Ta xử lý sự kiện *FormClosing*, tức là một Form đang đóng lại (nháy vào nút X hoặc lệnh this.Close(),...). Bạn sẽ dùng một biến kiểu **DialogResult** để lưu lại kết quả trả về của phương thức *MessageBox.Show()*

```
DialogResult dlr = MessageBox.Show("Bạn muốn thoát chương trình?",
                                   "Thông báo",
                                    MessageBoxButtons.YesNo,
                                   MessageBoxIcon.Question);
```

Đến lúc này ta chỉ cần xét giá trị của biến *dlr* để rẽ nhánh thôi. Vì Form đang đóng nên nếu người dùng nhấn No thì sẽ hoãn hành động đóng lại, tham số sự kiện của *FormClosing* là **e** nên ta thực hiện như sau:

```
private void frmMessageBox_FormClosing(object sender, FormClosingEventArgs e) {
    DialogResult dlr = MessageBox.Show("Bạn muốn thoát chương trình?",
                                       "Thông báo",
                                       MessageBoxButtons.YesNo,
                                       MessageBoxIcon.Question);
```

```
        if (dlr == DialogResult.No) e.Cancel = true;
}
```

6.2. Điều khiển ColorDialog.

Là hộp thoại cho phép người sử dụng chọn màu sắc cho một đối tượng nào đó.

Các thuộc tính(Properties):

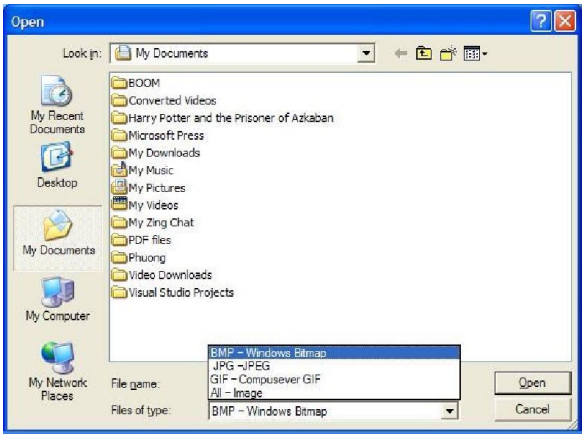
Thuộc tính	Mô tả
AllowFullOpen	Quy định hình thức mở hộp thoại màu. Gồm hai giá trị: **True:** Cho phép hiển thị các màu cho người sử dụng chọn lựa và cho hiệu lực chức năng Define Custom Color. **False:** Cho phép hiển thị các màu cho người sử dụng chọn lựa và vô hiệu hóa chức năng Define Custom Color.
Color	Màu do người sử dụng đã chọn và sẽ được trả về khi người sử dụng đóng hộp thoại.
CustomColors	Là mảng số nguyên đại diện cho các màu thường được chọn. Ví dụ minh họa hiển thị ba màu thường sử dụng trên hộp thoại: int Mau[]= {222663, 35453, 7888}; ColorDialog1.CustumColor = Mau;
SolidColorOnly	Là thuộc tính có giá trị luận lý (True/False). Nếu giá trị là True, nó sẽ giới hạn việc chọn màu của người sử dụng, chỉ cho phép người sử dụng chọn những màu tô đặc đối với những máy hiển thị 256 màu.

6.3. Điều khiển OpenFileDialog.

Là điều khiển cho phép ta chọn tập tin cần mở. Thông thường khi được hiển thị, điều khiển ít khi hiển thị tất cả các tập tin trong thư mục, thường thì những tập tin được hiển thị sẽ được giới hạn vào một số loại mà ứng dụng có thể nhận ra và xử lý chúng. Thuộc tính Filter cho phép chọn lọc và hiển thị một số loại tập tin trong hộp thoại thông qua phần mở rộng của chúng. Phần mở rộng mặc định của hộp thoại được thiết lập thông qua thuộc tính DefaultExtension. Hai thuộc tính Filter và DefaultExtension có thể được thiết lập ngay trong quá trình thiết kế hay trong khi chương trình thực thi.

Các thuộc tính(Properties):

Thuộc tính	Mô tả
CheckFileExist	Đây là thuộc tính có giá trị luận lý (True/False) sẽ cho phép hiển thị bảng thông báo nếu người dùng nhập vào một tập tin không tồn tại trong hệ thống.
CheckPathExist	Đây là thuộc tính có giá trị luận lý (True/False) sẽ cho phép hiển thị bảng thông báo nếu người dùng nhập vào một đường dẫn không tồn tại trong hệ thống.
AddExtension	Cho phép tự động thêm phần mở rộng vào tập tin nếu người dùng bỏ qua nó. Phần mở rộng thêm vào chính là giá trị của thuộc tính DefaultExtension.
DefaultExtension	Quy định phần mở rộng của tập tin sẽ được thêm vào. Thuộc tính này luôn được sử dụng khi giá trị của thuộc tính AddExtension = True. Giá trị bắt đầu của thuộc tính này là dấu chấm (“.”). ví dụ: “.Txt”
DereferenceLinks	Mặc định, thuộc tính này có giá trị True. Khi mở một ShortCut, giá trị FileName trả về là đường dẫn đến tập tin thi hành chương trình của ShortCut đó. Nếu muốn tham chiếu đến đường dẫn của ShortCut này, ta gán giá trị cho thuộc tính này là Fasle.
FileName	Thuộc tính trả về tên tập tin (cùng đường dẫn) của tập tin được chọn. Nếu người sử dụng nhấn phím Cancel, ta không nên truy xuất đến thuộc tính này.
FilterIndex	Quy định loại tập tin nào được hiển thị mặc định khi mở hộp thoại. Giá trị đầu tiên được tính là 1. Trong ví dụ trên nếu ta cho thuộc tính FilterIndex = 4 thì All – Image sẽ được chọn mặc định.

InitialDirectory	Quy định thư mục nào sẽ được mở khi hiển thị hộp thoại.
RestoreDirectory	Mỗi khi mở hộp thoại, thư mục hiện hành là thư mục được người sử dụng mở lần cuối trước đó sẽ được hiển thị hay không (True/False).
FileNames	Nếu hộp thoại cho phép chọn nhiều tập tin, thuộc tính này sẽ lưu giữ tất cả đường dẫn và tập tin được chọn.
ShowReadOnly	Cho phép hiển thị CheckBox Open as Read Only trong hộp thoại hay không (True/False). Giá trị mặc định là False.
Filter	Dùng để quy định các loại tập tin sẽ được hiển thị trong hộp thoại. Để hiển thị những tập tin văn bản, ta sẽ gán giá trị của thuộc tính là: "Tập tin văn bản|*.txt" **Ví dụ:** Hiển thị các tập tin hình ảnh. SaveFileDialog1.Filter = "BMP – Windows Bitmap|*.bmp| JPG –JPEG|*.jpg|GIF – Compusever GIF|*.gif|All – Image|*.BMP;*.GIF;*.JPG" (viết trên một dòng). 
ReadOnlyChecked	Quy định giá trị của CheckBox ReadOnlyChecked trong hộp thoại có được chọn hay không (True/False). Nên thiết lập giá trị của thuộc tính này = True nếu chỉ mở tập tin để đọc. Giá trị mặc định là False.

MultiSelect	Cho phép người sử dụng chọn nhiều tập tin cùng lúc trong hộp thoại Open hay không (True/False). Mặc định là False, chỉ cho phép chọn một tập tin. Lưu ý: chỉ được phép chọn các tập tin trong cùng thư mục. Ví dụ xử lý các tập tin được chọn: … if (OpenFileDialog1.ShowDialog( ) == DialogResult.OK) { *// Duyệt từ tập tin đầu đến tập tin cuối được chọn* for (int i = 0; i <= OpenFileDialog1.FileNames.Getlength(0) -1; i++) { *// Xử lý tập tin được chọn* MessageBox.Show(OpenFileDialog1.FileNames[i]); } } …

Ví dụ:

Thiết kế và xử lý các sự kiện chương trình có giao diện sau

Thiết đặt các Properties:

Đặt tên cho các đối tượng như sau:

Các điều khiển	Thuộc tính
Với TextBox Mã NV	Name: txtMaNV MaxLength: 100
Với RadioButton Nam	Name: radNam Text: Nam
Với RadioButton Nữ	Name: radNu Text: Nữ
Với GroupBox	Dùng đển nhóm các chức năng
Với ImageList	Name: imglst ColorDepth: Depth32Bit ImageSize: 100,100 Images: chèn các ảnh cần thiết
Với ListView	Name: lwDanhSach FullRowSelect: True SmallImageList: Imglst View: Details
Với Label Chọn hình đại diện	Name: lblChonHinh Text: Chọn hình đại diện
Với Button Loại bỏ	Name: btnloaibo Text: Loại bỏ
Với Button Thêm	Name: btnthem Text: Thêm
Với Button Thêm mới	Name: btnthemmoi Text: Chơi tiếp
Với Menustrip Thoát	Name: mnuThoat Text: Thoát

Khai báo các biến

ListViewItem item;

```
int i = 0;
Boolean Thaydoi=false;
 Image anh;
```

Các hàm con hổ trợ xử lý

Hàm LoadAnh lại sau khi xoá dòng trong ListView

```
private void LoadAnh()  {
    for (int i = 0; i < lwdanhsach.Items.Count; i++)    {
        lwdanhsach.Items[i].ImageIndex = i;//danh lai chi so anh
    }
}
```

Hàm kiểm tra trùng mã nhân viên

```
private int KTma(ListView lw, string ma)    {
        for (int i = 0; i < lw.Items.Count; i++)
            if (lw.Items[i].SubItems[1].Text == ma) return i;
        return -1;
}
```

Sự kiện nút Thêm

```
private void btnthem_Click(object sender, EventArgs e)    {
    if (txtMaSo.TextLength == 0 || txtHoTen.TextLength == 0
        || txtNam.TextLength==0)    {
        MessageBox.Show("Thiếu dữ liệu");
        return;
    }
    if (KTma(lwdanhsach, txtma.Text) != -1)  {
        MessageBox.Show("Trùng mã");
        txtMaSo.Focus();
        txtMaSo.SelectAll();
        return;
```

```
        }
        if (DateTime.Now.Year - int.Parse(txtNam.Text) < 14)    {
            MessageBox.Show("Chưa đủ 14 tuổi", "Thông báo");
            txtNam.Focus();
            txtNam.SelectAll();
            return;
        }
        if (Thaydoi==false)       {
            MessageBox.Show("Vui lòng chọn ảnh mới", "Thông báo");
            return;
        }
        imglst.Images.Add(anh);
        i = lwdanhsach.Items.Count;
        item = new ListViewItem();
        item.SubItems.Add(txtMaSo.Text);
        item.SubItems.Add(txtHoTen.Text);
        item.SubItems.Add(txtNam.Text);
        item.SubItems.Add(radPhaiNam.Checked ? "Nam" : "Nữ");
         item.ImageIndex = i;
        lwdanhsach.Items.Add(item);
        btnthemmoi_Click(sender, e);
    }
```

Sự kiện khi chọn 1 dòng trong ListView

```
private void lwdanhsach_SelectedIndexChanged(object sender, EventArgs e)   {
        if (lwdanhsach.SelectedIndices.Count == 1)     {
            i = lwdanhsach.FocusedItem.Index;
            txtMaSo.Text = lwdanhsach.FocusedItem.SubItems[1].Text;
            txtHoTen.Text = lwdanhsach.FocusedItem.SubItems[2].Text;
            txtNam.Text = lwdanhsach.FocusedItem.SubItems[3].Text;
            radPhaiNam.Checked = lwdanhsach.FocusedItem.SubItems[4].Text
                                                    == "Nam" ? true : false;
```

```
            radPhaiNu.Checked = !radPhaiNam.Checked;
            pic_Hinh.Image =imglst.Images[i];
            return;
        }
        if (lwdanhsach.SelectedIndices.Count > 1)  btnthemmoi_Click(sender, e);
    }
```

Sự kiện nút Thêm mới

```
    private void btnthemmoi_Click(object sender, EventArgs e)      {
        txtMaSo.Clear();
        txtHoTen.Clear();
        radPhaiNam.Checked = true;
        txtNam.Text = DateTime.Now.Year.ToString();
        pic_Hinh.Image = Properties.Resources._6a8951b136f7dfa986e6;
        Thaydoi = false;
        txtMaSo.Focus();
    }
```

Sự kiện nút Loại bỏ

```
    private void btnloaibo_Click(object sender, EventArgs e)     {
        if (lwdanhsach.SelectedIndices.Count > 0)      {
            DialogResult tb = MessageBox.Show("Bạn xoá dữ liệu?","Cảnh báo",
                        MessageBoxButtons.YesNo,MessageBoxIcon.Warning);
            if (tb == DialogResult.Yes)  {
                while (lwdanhsach.SelectedIndices.Count > 0)  {
                    int i = lwdanhsach.SelectedIndices[0];
                    lwdanhsach.Items.RemoveAt(lwdanhsach.SelectedIndices[0]);
                    imglst.Images.RemoveAt(i);
                }
             lwdanhsach.SelectedIndices.Clear();
            LoadAnh();//danh lai chi so anh
```

```
            btnthemmoi_Click(sender,e);
            return;
           }
        }
        MessageBox.Show("Bạn chưa chọn dữ liệu xoá");
    }
```

Sự kiện Load Form

```
    private void Form1_Load(object sender, EventArgs e)    {
            btnthemmoi_Click(sender,e);
    }
```

Sự kiện FormClosing khi đóng Form

```
private void Form1_FormClosing(object sender, FormClosingEventArgs e)  {
    DialogResult tb = MessageBox.Show("Bạn thoát chương trình?",
                        "Cảnh báo",MessageBoxButtons.YesNo,
                        MessageBoxIcon.Warning);
    if (tb == DialogResult.No) e.Cancel = true;
}
```

Sự kiện Click Label Chọn hình đại diện

```
    private void lblChonHinh_Click(object sender, EventArgs e)   {
        OpenFileDialog fr = new OpenFileDialog();
        fr.InitialDirectory = @"D:\";
        fr.Filter = "Bmp|*.bmp|JPG|*.jpg|All File|*.*";
        fr.FilterIndex = 2;
        fr.RestoreDirectory = true;
        if (fr.ShowDialog() == DialogResult.OK)    {
            pichinh.Image = Image.FromFile(fr.FileName);
            anh = Image.FromFile(fr.FileName);
            Thaydoi = true;
```

```
    }
}
```

Các ràng buộc dữ liệu ô TextBox

```
private void txtHoTen_KeyPress(object sender, KeyPressEventArgs e) {
    if (char.IsNumber(e.KeyChar) || char.IsPunctuation(e.KeyChar)
            || char.IsSymbol(e.KeyChar))
        e.Handled = true;
}
private void txtMaSo_KeyPress(object sender, KeyPressEventArgs e)    {
    if (char.IsWhiteSpace(e.KeyChar) || char.IsPunctuation(e.KeyChar)
            || char.IsSymbol(e.KeyChar))
        e.Handled = true;
}
private void txtNam_KeyPress(object sender, KeyPressEventArgs e)   {
    if (char.IsLetter(e.KeyChar) || char.IsPunctuation(e.KeyChar)
            || char.IsSymbol(e.KeyChar) || char.IsWhiteSpace(e.KeyChar))
        e.Handled = true;
}
```

Sự kiện Clik vào MenuStrip Thoát

```
private void mnuThoat_Click(object sender, EventArgs e) {
    Application.Exit();
}
```

6.4. Điều khiển FontDialog.

Là điều khiển cho phép người sử dụng lựa chọn Font chữ, kiểu (in đậm, nghiêng …), kích cỡ chữ. Ngoài các chức năng cơ bản trên, nó còn có các tùy chọn cho phép chọn màu chữ, hơn thế nữa, nó còn cho phép chấp nhận các giá trị đang lựa chọn trên hộp thoại mà không cần đóng lại cửa sổ thông qua nút lệnh Apply.

Các thuộc tính(Properties):

Thuộc tính	Mô tả
AllowScriptChange	Gồm hai giá trị: **True:** Cho phép hiển thị Combo Script trong hộp thoại. **False:** Không cho phép hiển thị Combo Script trong hộp thoại.
AllowVectorFonts	Gồm hai giá trị: **True:** Cho phép hiển thị và lựa chọn các Vector (hướng) Font trong hộp thoại. **False:** Không cho hiển thị và lựa chọn các Vector Font trong hộp thoại.
AllowVeticalFonts	Gồm hai giá trị: **True:** Cho phép hiển thị các Font chữ được thể hiện theo chiều ngang và chiều dọc (đứng). **False:** Chỉ cho phép hiển thị các Font chữ theo chiều ngang.
Color	Thiết lập hay nhận giá trị màu của Font chữ trên hộp thoại.
Font	Cho phép thiết lập hay nhận các thiết lập của người sử dụng từ hộp thoại Font sau khi người sử dụng nhấn Apply hay OK
FontMustExist	Gồm hai giá trị: True: Chỉ cho phép người sử dụng chọn các Font đang có trong máy tính, nếu người sử dụng đánh vào một Font chữ không tồn tại thì sẽ xuất hiện thông báo lỗi.
	False: Cho phép người sử dụng nhập vào một Font chữ không tồn tại trong máy tính.
Maxsize	Giới hạn kích thước tối đa của Font chữ trong hộp thoại Font. Nếu người sử dụng nhập vào kích thước lớn hơn giới hạn này sẽ bị báo lỗi.

Minsize	Giới hạn kích thước tối thiểu của Font chữ trong hộp thoại Font. Nếu người sử dụng nhập vào kích thước nhỏ hơn giới hạn này sẽ bị báo lỗi.
ShowApply	Cho hiển thị nút lệnh Apply trong hộp thoại hay không (True/False). Mặc định là không hiển thị nút lệnh Apply.
ShowColor	Cho hiển thị ComboBox chọn màu cho Font chữ trong hộp thoại hay không (True/False). Mặc định là False.
ShowEffects	Cho hiển thị hay không hiển thị hai tùy chọn Strikeout và Underline và cả ComboBox chọn màu chữ cho dù thuộc tính ShowColor = True.

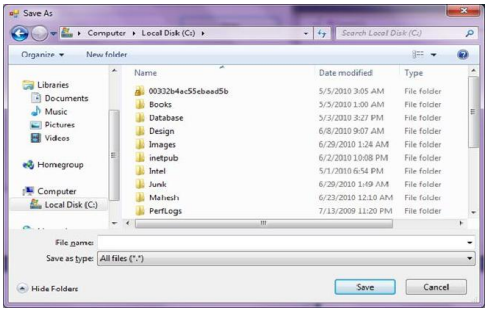
6.5. Điều khiển SaveFileDialog.

Là điều khiển cho phép chỉ định vị trí và lưu trữ tập tin như hộp thoại Save và Save As trong Winword hay Excel.

Các thuộc tính và phương thức cũng tương tự như OpenFileDialog ngoại trừ các thuộc tính: FileNames, MultiSelect, ReadOnlyChecked và ShowReadOnly.

Các thuộc tính(Properties):

Thuộc tính	Mô tả
CheckFileExist	Đây là thuộc tính có giá trị luận lý (True/False) sẽ cho phép hiển thị bảng thông báo nếu người dùng nhập vào một tập tin không tồn tại trong hệ thống.
CheckPathExist	Đây là thuộc tính có giá trị luận lý (True/False) sẽ cho phép hiển thị bảng thông báo nếu người dùng nhập vào một đường dẫn không tồn tại trong hệ thống.
AddExtension	Cho phép tự động thêm phần mở rộng vào tập tin nếu người dùng bỏ qua nó. Phần mở rộng thêm vào chính là giá trị của thuộc tính DefaultExtension.
DefaultExtension	Quy định phần mở rộng của tập tin sẽ được thêm vào. Thuộc tính này luôn được sử dụng khi giá trị của thuộc tính AddExtension = True. Giá trị bắt đầu của thuộc tính này là dấu chấm (“.”). ví dụ: “.Txt”

DereferenceLinks	Mặc định, thuộc tính này có giá trị True. Khi mở một ShortCut, giá trị FileName trả về là đường dẫn đến tập tin thi hành chương trình của ShortCut đó. Nếu muốn tham chiếu đến đường dẫn của ShortCut này, ta gán giá trị cho thuộc tính này là Fasle.
Filter	Dùng để quy định các loại tập tin sẽ được hiển thị trong hộp thoại. Để hiển thị những tập tin văn bản, ta sẽ gán giá trị của thuộc tính là: "Tập tin văn bản|*.txt" Ví dụ: Hiển thị tất cả các tập tin. OpenFileDialog1.Filter = "All File|(*.*)"; 
FileName	Thuộc tính trả về tên tập tin (cùng đường dẫn) của tập tin được chọn. Nếu người sử dụng nhấn phím Cancel, ta không nên truy xuất đến thuộc tính này.
FilterIndex	Quy định loại tập tin nào được hiển thị mặc định khi mở hộp thoại. Giá trị đầu tiên được tính là 1. Trong ví dụ trên nếu ta cho thuộc tính FilterIndex = 4 thì All – Image sẽ được chọn mặc định.
InitialDirectory	Quy định thư mục nào sẽ được mở khi hiển thị hộp thoại.
RestoreDirectory	Mỗi khi mở hộp thoại, thư mục hiện hành là thư mục được người sử dụng mở lần cuối trước đó sẽ được hiển thị hay không (True/False).

Chương 7: LÀM VIỆC VỚI HỆ THỐNG

7.1. Lớp SystemInformation.

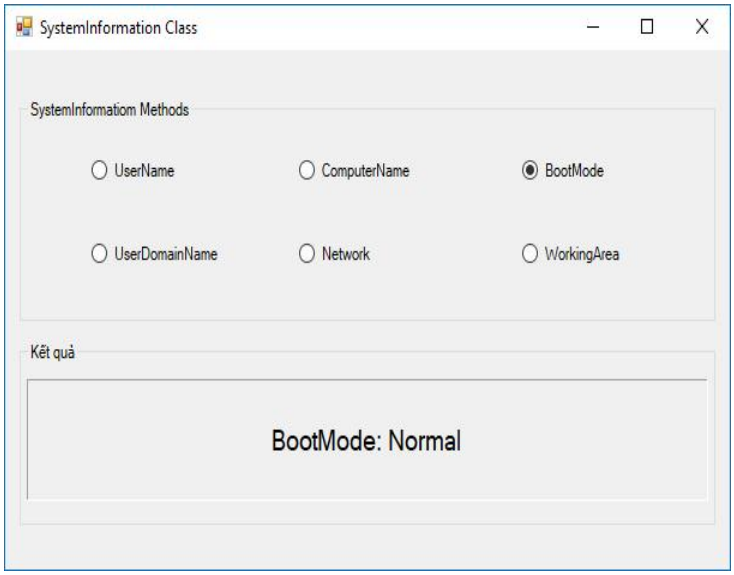
Lớp SystemInformatoin cung cấp nhiều thuộc tính nắm giữ về hệ thống. Tình trạng kết nối mạng, tên miền mà người dùng đăng nhập vào tài khoản, tên máy tính và cả chế độ khởi động hệ điều hành.

Các thuộc tính(Properties)

Thuộc tính	Mô tả
Username	Trả về tài khoản người sử dụng ứng dụng hiện thời khi đăng nhập vào hệ thống.
UserDomainName	Trả về tên miền mà người sử dụng đã đăng nhập bằng tài khoản hợp lệ.
Network	Cho biết tình trạng kết nối mạng. Gồm hai giá trị True: trạng thái được kết nối mạng False: trạng thái không kết nối mạng
ComputerName	Trả về giá trị là tên máy tính do mình đặt trong quá trình cài đặt máy tính.
BootMode	Chế độ khởi động hệ điều hành Windows. Gồm ba giá trị Normal FailSafe(Safe Mode) FailSafeWithNetwork(Safe Mode With Network)
PowerStatus	Cho phép truy cập vào các thông tin về nguồn điện sử dụng cho máy
WorkingArea	Giữ các thông tin về vùng làm việc trên màn hình

Ví dụ:

Chương trình được thiết kế giao diện xử lý các phương thức trong lớp SystemInformation khi người dùng chọn một trong các radio trong nhóm phương thức trên giao diện



Các sự kiện cho các phương thức sau

```
private void Form1_Load(object sender, EventArgs e)       {
            radBootMode.Checked = true;
}
```

```
private void radUserNam_CheckedChanged(object sender, EventArgs e) {
    lblketQua.Text = radUserNam.Text+": "+ SystemInformation.UserName;
}
```

```
private void radComputerName_CheckedChanged(object sender, EventArgs e){
            lblketQua.Text =radComputerName.Text + ": " +
                        SystemInformation.ComputerName;
}
```

```
private void radUserDomainName_CheckedChanged(object sender, EventArgs
e){
        lblketQua.Text = radUserDomainName.Text + ": " +
                    SystemInformation.UserDomainName;
}
```

```
private void radNetwork_CheckedChanged(object sender, EventArgs e)    {
        lblketQua.Text = radUserNam.Text + ": " +
       SystemInformation.UserName;
        if (!SystemInformation.Network)
                        lblketQua.Text += "\nNetwork: Đang ngắt kết nối";
        else    lblketQua.Text += "\nNetwork: Đã kết nối";
    }
```

```
private void radBootMode_CheckedChanged(object sender, EventArgs e)  {
        lblketQua.Text = radBootMode.Text + ": " +
                                SystemInformation.BootMode;
    }
```

```
private void radWorkingArea_CheckedChanged(object sender, EventArgs e)   {
                Size s = new Size();
                 s = SystemInformation.WorkingArea.Size;
                lblketQua.Text = "Độ rộng màn hình: " + s.Width.ToString() +
                                "\nĐộ cao màn hình: " + s.Height.ToString();
    }
```

7.2. Lớp SendKeys.

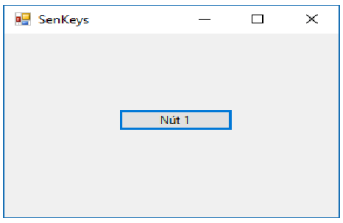
SendKeys cung cấp các phương thức để gửi chuỗi dữ liệu cùng tổ hợp phím đến một ứng dụng hiện hành. Lớp SendKeys cung cấp ba phương thức chính: Send, SendWait và Flush cho phép thao tác và xử lý bàn phím.

Các phương thức(Methods)

Phương thức	Mô tả
Send	Cho phép gởi một chuỗi dữ liệu và các tổ hợp phím đến ứng dụng hiện hành

SendWait	Cho phép gởi một chuỗi dữ liệu và các tổ hợp phím đến ứng dụng hiện hành và chờ cho đến khi tiến trình kết thúc.
Flush	Xử lý các thông điệp đang tồn tại ở hàng đợi.

Ví dụ: Giả sử thiết kế giao diện sau



Đoạn mã sau đây trình bày cách sử dụng phương thức Gửi.

- Trên Form SendKeys chứa nút có tên là Nút1. Đảm bảo các sự kiện nhấp được liên kết với các phương thức xử lý sự kiện của chúng trong ví dụ này. Thuộc tính Tab Index của nút điều khiển phải được đặt thành 0.
- Khi thực hiện chương trình, bấm đúp vào biểu mẫu để kích hoạt sự kiện nhấp chuột của nút.

// Click Button Nut 1 soạn đoạn mã.

```
private void Button1_Click(System.Object sender, System.EventArgs e){
        MessageBox.Show("Bạn Click vào Button Nut1!");
}
```

// Viết sự kiện DoubleClick của Form

```
private void Form1_DoubleClick(object sender, System.EventArgs e){
        SendKeys.Send("{ENTER}");
}
```

Định nghĩa một thông điệp đúng:

1. Các chuỗi ký tự được send một cách bình thường, ví dụ "Tuân", trong dấu ngoặc kép là tất cả nội dung một thông điệp bạn muốn gửi.
2. Một phím đặc biệt được đặt trong cặp ngoặc nhọn {}, ví dụ "**{F4}**" hoặc "**{ESC}**", đây là hai trong nhiều phím đặc biệt.
3. Một phím được nhấn khi đang có một phím khác được giữ(*hold*), ví dụ nhấn giữ phím **Alt** và nhấn tiếp **F4** thì thông điệp cần send là "**%{F4}**", trong đó % là phím Alt đang được nhấn giữ + F4. Xem bảng quy tắc sau:

Bảng mã của phím kết hợp.

Phím	Mã
SHIFT	+
CTRL	^
ALT	%

Để chỉ định rằng bất kỳ sự kết hợp nào của SHIFT, CTRL và ALT nên được giữ trong khi nhấn một số phím khác, hãy đóng mã cho các khóa đó trong ngoặc đơn.

Ví dụ:

Để chỉ định giữ phím Shift trong khi nhấn E và C, hãy sử dụng "+ (EC)". Để chỉ định giữ phím Shift trong khi nhấn E, tiếp theo là C không có Shift, sử dụng "+ EC".

Bảng mã của phím hành động.

Phím	Mã
BACKSPACE	{BACKSPACE}, {BS}, hoặc {BKSP}
BREAK	{BREAK}
CAPS LOCK	{CAPSLOCK}
DEL hoặc DELETE	{DELETE} hoặc {DEL}

DOWN ARROW	{DOWN}
END	{END}
ENTER	{ENTER} hoặc ~
ESC	{ESC}
HELP	{HELP}
HOME	{HOME}
INS or INSERT	{INSERT} hoặc {INS}
LEFT ARROW	{LEFT}
NUM LOCK	{NUMLOCK}
PAGE DOWN	{PGDN}
PAGE UP	{PGUP}
PRINT SCREEN	{PRTSC}
RIGHT ARROW	{RIGHT}
SCROLL LOCK	{SCROLLLOCK}
TAB	{TAB}
UP ARROW	{UP}
F1	{F1}
F2	{F2}
F3	{F3}
F4	{F4}
F5	{F5}
F6	{F6}

F7	{F7}
F8	{F8}
F9	{F9}
F10	{F10}
F11	{F11}
F12	{F12}
Keypad add	{ADD}
Keypad subtract	{SUBTRACT}
Keypad multiply	{MULTIPLY}
Keypad divide	{DIVIDE}

Cách hoạt động của các phương thức SendKeys này: bởi vì bạn không thể send keys mà không có đích đến, nếu bạn cứ thế Send keys mà không chủ đích thì keys sẽ được Send đến bất kỳ một Form/Window nào đó đang được Active.

Vậy để thực hiện Sendkeys từ một Application/Form/Window đến một Form/Window nào đó thì chúng ta cần set Active cho Form/Window đó trước khi Send, nhưng class **Sendkeys** không support việc set Active cho Form/Window ta cần thao tác Active cho cửa sổ nào đó cần chuyển dữ liệu.

7.3. Lớp Application.

Cung cấp các phương thức và thuộc tính tĩnh để quản lý một ứng dụng, chẳng hạn như các phương thức khởi động và dừng ứng dụng, để xử lý các thông báo Windows và các thuộc tính để lấy thông tin về một ứng dụng. Lớp này không thể được thừa kế.

Các thuộc tinh(Properties)

Thuộc tính	Mô tả
CommonAppDataPath	Trả về đường dẫn của ứng dụng

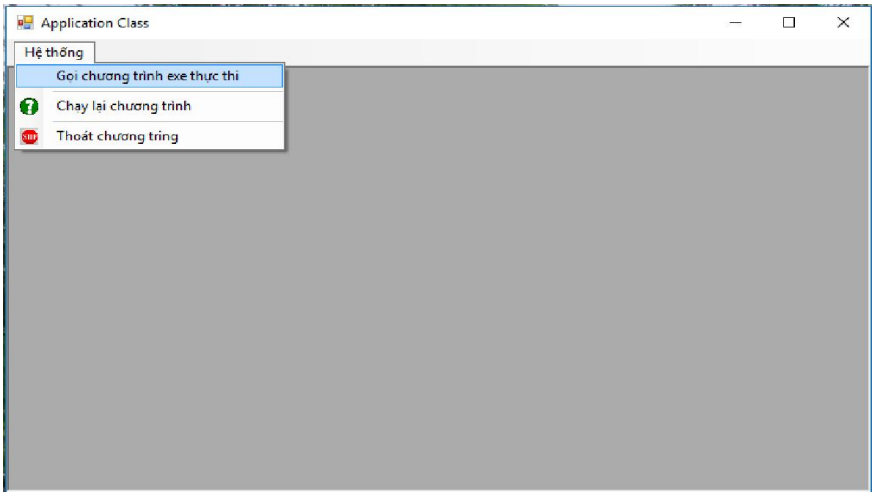
CompanyName	Trả về tên công ty
CurrentCulture	Gán hay trả về thuộc tính của tiến trình hiện tại
CurrentInputLanguage	Gán hay trả về ngôn ngữ của tiến trình hiện hành
ExecutablePath	Trả về đường dẫn của tập tin đang thực thi
ProductName	Trả về tên ứng dụng
productVersion	Trả về phiên bản của ứng dụng
StartupPath	Trả về dường dẫn tập tin đang thực thi
UserAppDataPath	Trả về đường dẫn ứng dụng, từ đó người sử dụng khác có thể truy cập được

Các phương thức(Methods)

Phương thức	Mô tả
Exit	Thông báo cho ứng dụng sẽ kết thúc, sau đó tất cả các cửa sổ ứng dụng sẽ đóng lại.
ExitThread	Thông báo cho tiến trình hiện tại sẽ kết thúc.
DoEvents	Xử lý các thông điệp đang còn trong hàng đợi.
EnableVisualStyles	Cho phép hiện thực kiểu của điều khiển theo kiểu của hệ điều hành mà ứng dụng đang chạy
Restart	Kết thúc và khởi động lại ứng dụng đang hoạt động
Run	Dùng để thực hiện khởi chạy đối tượng đầu tiên của ứng dụng.

Ví dụ:

Chương trình sau thể hiện các phương thức Restart và Exit



Các sự kiện tương ứng sau

```
private void cmdExit_Click(object sender, EventArgs e)       {
        DialogResult tb = MessageBox.Show("Bạn muốn thoát chương
                trình ứng dụng?", "Thông báo hỏi",
                MessageBoxButtons.YesNo,
                MessageBoxIcon.Question);
    if (tb == DialogResult.Yes) Application.Exit();
}
```

```
private void cmdRestart_Click(object sender, EventArgs e)       {
        DialogResult tb = MessageBox.Show("Bạn muốn chạy lại
                chương trình ứng dụng?","Cảnh báo",
            MessageBoxButtons.OKCancel,MessageBoxIcon.Warning);
        if( tb== DialogResult.OK) Application.Restart();
}
```

```
private void cmdGoiChuongTrinh_Click(object sender, EventArgs e)  {
    System.Diagnostics.Process.Start(Application.StartupPath +
                @"\BauCua.exe");
}
```

7.4. Lớp Clipboard.

Lớp Clipboard cung cấp các phương thức để lưu trữ và lấy dữ liệu nhiều loại định dạng khác nhau từ Clipboard hệ thống vào các ô nhập.

Khi làm việc đối tương Clipboard, nếu ta sử dụng một phương thức để lưu dữ liệu vào Clipboard thì sẽ có phương thức tương ứng để lấy dữ liệu từ Clipboard.

Ví dụ:

Để đưa 1 đoạn text vào clipboard sử dụng:

```
Clipboard.SetText("hanhtranglaptrinh.com");
```

Kiểm tra bằng cách kiếm cái ô text nào đó Ctrl + V hoặc click phải chuột chọn paste sẽ thấy kết quả.

Để lấy 1 đoạn text đang được lưu trong clipboard ra sử dụng:

```
string   text = Clipboard.GetText();
```

Kiểm tra bằng cách kiếm 1 đoạn văn bản nào đó bôi đen Click chuột phải chọn copy hoặc Ctrl + C sau đó chạy đoạn code trên và kiếm tra biến text.

Đối với ứng dụng windows (windows form, WPF, ...) thì ngoài text ra còn có thể đưa một số object khác (Image, Stream, ...) vào Clipboard. Ứng dụng silverlight chỉ hỗ trợ text.

Các phương thức(Methods)

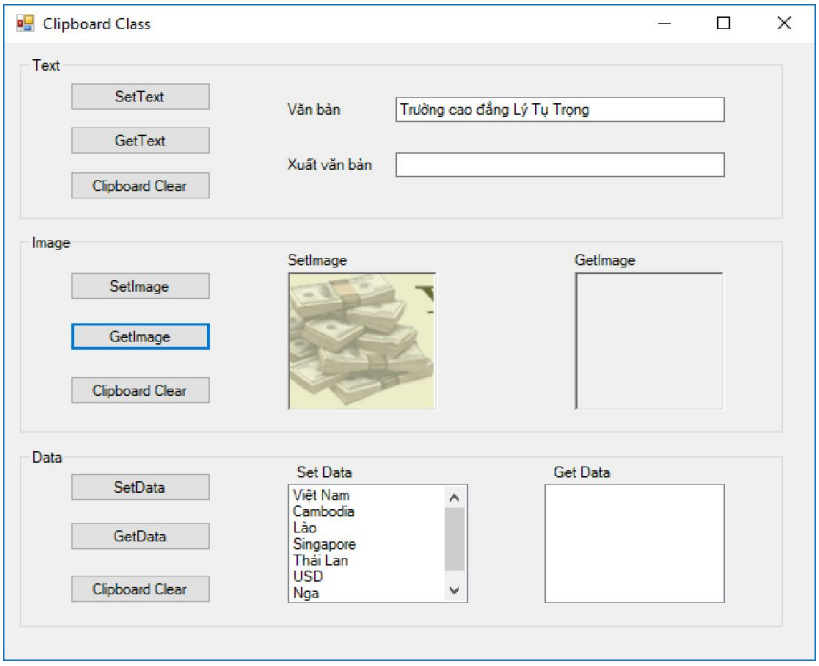
Phương thức	Mô tả
Clear	Xoá tất cả dữ liệu trong Clipboard
SetText	Lưu một chuỗi vào Clipboard
GetText	Lấy chuỗi đang có trong Clipboard
ContaintsText	Cho kết quả tìm kiếm chuỗi trong Clipboard. Gồm hai giá trị True: Tìm thấy chuỗi lưu trữ trong Clipboard False: Không tìm thấy chuỗi lưu trữ trong Clipboard
SetImage	Lưu một ảnh vào Clipboard
GetImage	Lấy ảnh đang có trong Clipboard
ContaintsImage	Cho kết quả tìm kiếm ảnh trong Clipboard.

	Gồm hai giá trị True: Tìm thấy ảnh lưu trữ trong Clipboard False: Không tìm thấy ảnh lưu trữ trong Clipboard
SetData	Lưu một đối tượng vào Clipboard
GetData	Lấy đối tượng đang có trong Clipboard
ContaintsData	Cho kết quả tìm kiếm đối tượng trong Clipboard. Gồm hai giá trị True: Tìm thấy đối tượng lưu trữ trong Clipboard False: Không tìm thấy đối tượng lưu trữ trong Clipboard

Ví dụ:

Xét chương trình có giao diện thiết sau.

- Khi nhấn nút SetText thì lưu đoạn văn bản có trong ô nhập văn bản vào Clipboard, nếu ô văn bản đã nhập văn bản.
- Khi nhấn nút GetText thì
  - Nếu trong Clipboard không tìm thấy văn bản thì xuất hộp thoại thông báo
  - Nếu có thì xuất đạo văn bản trong Clipboard ra ô xuất văn bản.
- Khi nhấn nút Clipboard Clear
  - Xóa tất cả trên ô văn bản, xuất văn bản và Clipboard.



Tương tự giống các phương thức Text, các phím chức năng Image hoàn toàn giống trên.

Các sự kiện trong nhóm Text trên chương trình

```
private void btnSetText_Click(object sender, EventArgs e) {
        if(txtSettext.TextLength!=0)  Clipboard.SetText(txtSettext.Text);
}
```

```
private void btnGetText_Click(object sender, EventArgs e)    {
if (Clipboard.ContainsText())
        txtGetText.Text = Clipboard.GetText();
else MessageBox.Show("Không tìm thấy đoạn văn trong Clipboard",
                        "Thông báo");
}
```

```
private void btnClearText_Click(object sender, EventArgs e)      {
        txtSettext.Clear();
        txtGetText.Clear();
        Clipboard.Clear();
}
```

Các sự kiện trong nhóm Image trên chương trình

```
private void btnSetImage_Click(object sender, EventArgs e)      {
        Clipboard.SetImage(pic1.Image);
}
```

```
private void btnGetImage_Click(object sender, EventArgs e)      {
        if (Clipboard.ContainsImage())
                pic2.Image = Clipboard.GetImage();
        else
                MessageBox.Show("Không có hình ảnh trong Clipboard",
                                        "Thông báo");
}
```

```
private void btnClearImage_Click(object sender, EventArgs e)      {
                Clipboard.Clear();
                pic1.Image = null;
        pic2.Image = null;
}
```

Các sự kiện trong nhóm Data trên chương trình

```
private void btnSetData_Click(object sender, EventArgs e) {
        if (lstSetData.SelectedIndex != -1)
                Clipboard.SetData(DataFormats.StringFormat,
                                                (lstSetData.SelectedItem));
}
```

```
private void btnGetData_Click(object sender, EventArgs e)      {
if (Clipboard.ContainsData(DataFormats.StringFormat))
```

```
                lstGetData.Items.Add((string)Clipboard.GetData(
                                        DataFormats.StringFormat));
    else
      MessageBox.Show("Không thấy Data trong Clipboard","Thông báo");
}
```

```
    private void btnClearData_Click(object sender, EventArgs e)      {
            Clipboard.Clear();
            lstGetData.Items.Clear();
           lstSetData.SelectedIndex = -1;
}
```

TÀI LIỆU THAM KHẢO

1. NET toàn tập của tác giả Dương Quang Thiện.
2. Lập trình ứng dụng C# của tác giả Nguyễn Anh Tài(Trung tâm Tin học trười đại học Khoa học Tự nhiên)
3. Lập trình Visual Studio .Net của tác giả Nguyễn Hữu Khang.
4. Lập trình .Net của tác giả Ngô Phương Lan
5. Những bài thực hành Visual Basic .Net căn bản của tác giả Đinh Xuân Lâm (VNGuide nhà xuất bản thống kê)
6. Một số trang Web Ebook.com, tailieu.com…