Microsoft®
Visual C#®

CHUYÊN ĐỀ NGÔN NGỮ LẬP TRÌNH 1

Người biên soạn: Hồ Quang Thái (MSCB: 2299)
BM. Công Nghệ Phần Mềm, Khoa CNTT&TT
Email: hqthai@cit.ctu.edu.vn
Số tín chỉ: 2 (20 LT + 20TH)

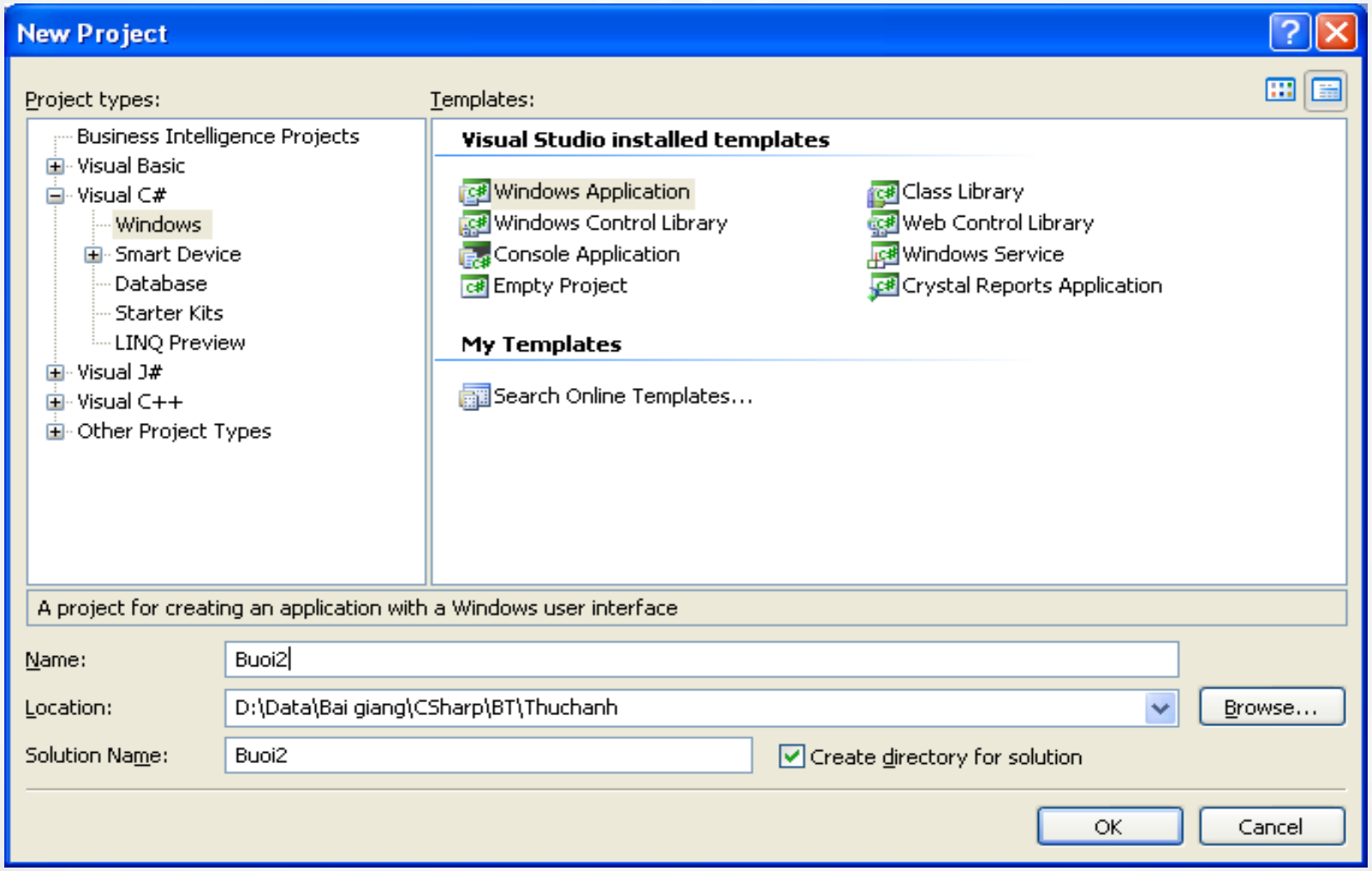
CHƯƠNG 4

WINDOWS FORMS

Nội dung

- *Ứng dụng Windows Forms*
- *Form*
- *Controls*
- *Sự hợp lệ của dữ liệu*
- *Ứng dụng MDI*

Kiểu ứng dụng phát triển bởi Visual Studio



Windows Forms (WinForms)

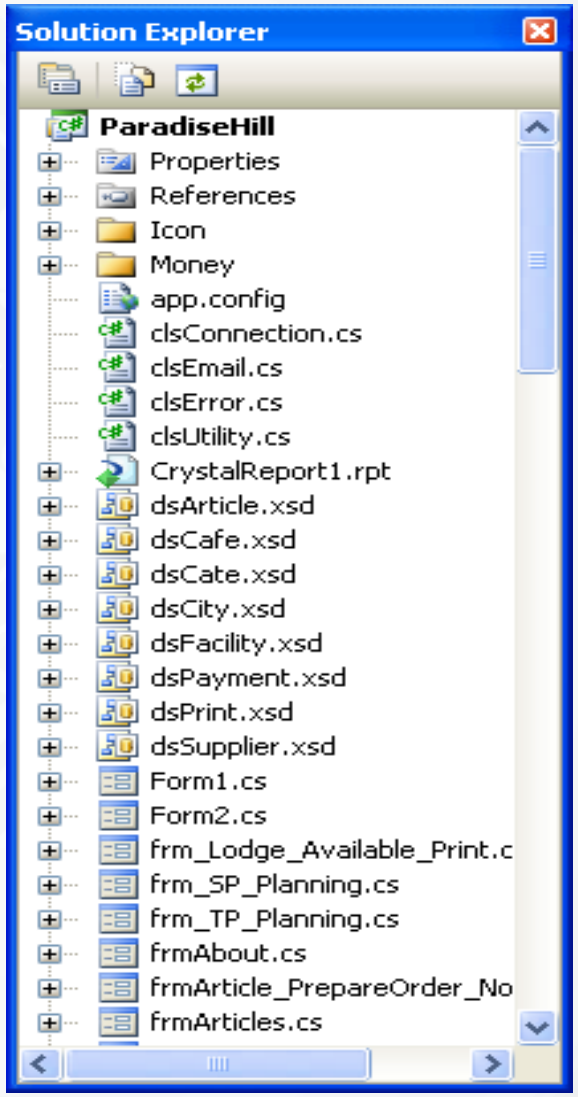
- **WinForms** là thư viện cho phép nhà phát triển tạo ứng dụng chạy trên nền Windows.
- Các thư viện này được định nghĩa trong không gian tên ***System.Windows.Forms***.
- Trong cửa số **New Project**, chọn **Templates Windows Application** để tạo ứng dụng **WinForms**

WebForms

- **WebForms** là thư viện lớp cho phép tạo ứng dụng chạy trên nền Web.
- Các thư viện này được định nghĩa trong không gian tên **System.Web**.

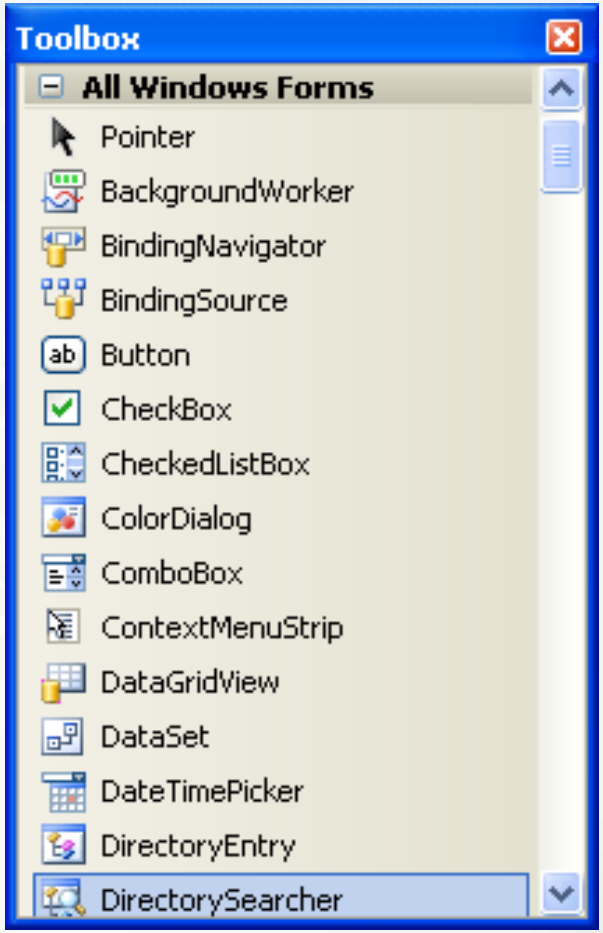
Cửa sổ Solution Explorer

- Cửa sổ Solution Explorer cho biết các dự án hiện hành cùng với cấu trúc phân cấp của các tập tin trong từng dự án.



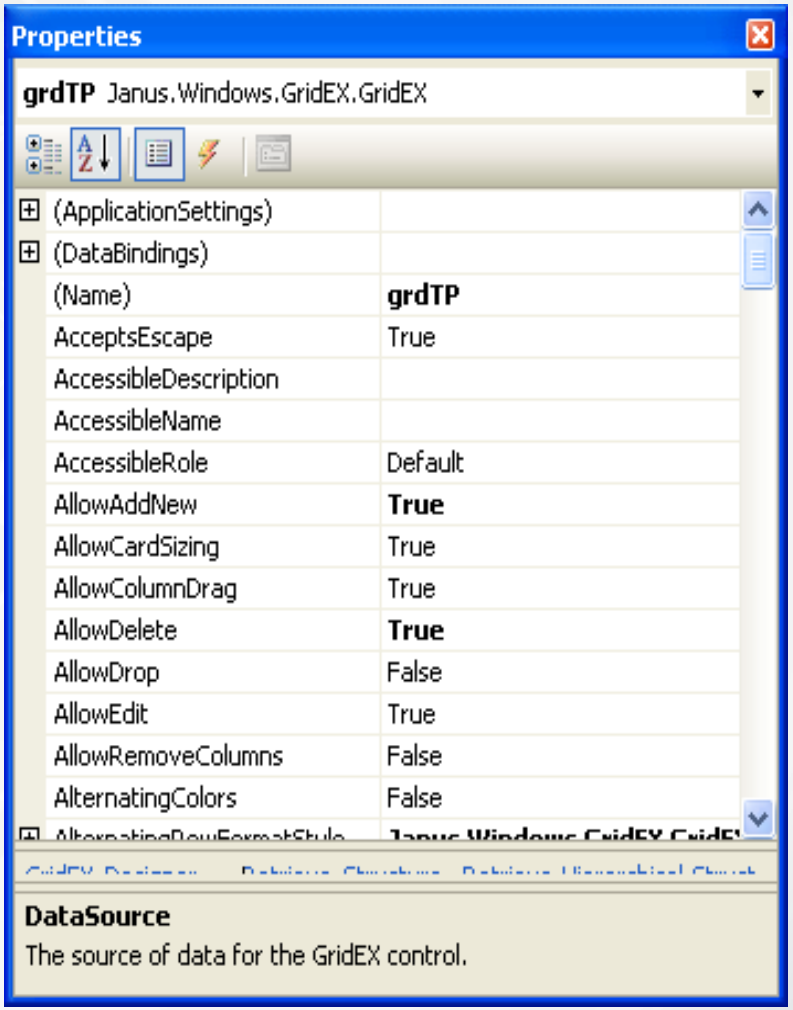
Hộp công cụ ToolBox

- Cửa sổ này chứa đựng các control WinForms, WebForms, ActiveX etc dùng để tạo giao diện của trình ứng dụng.



Cửa sổ thuộc tính Properties

- Cửa sổ thuộc tính cho phép xem và thiết lập giá trị các thuộc tính của form, controls, etc.

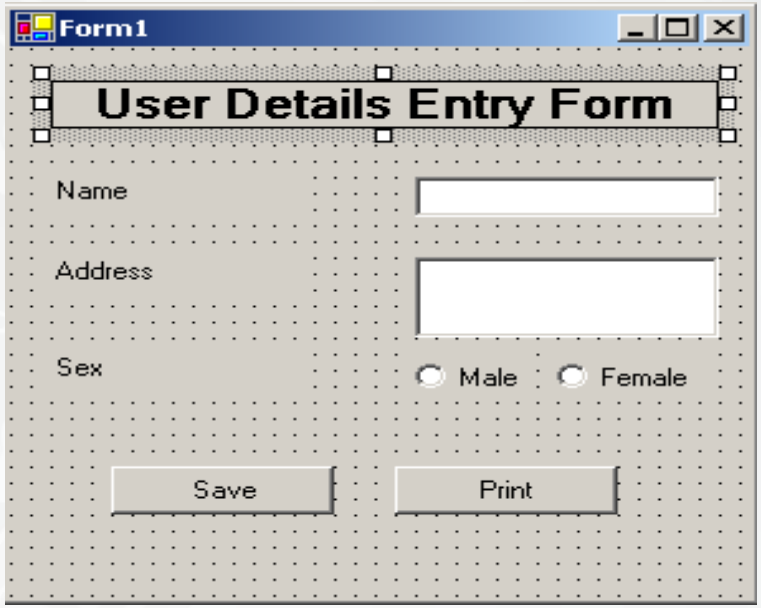


Ứng dụng WinForms

Application

User Interface

Program

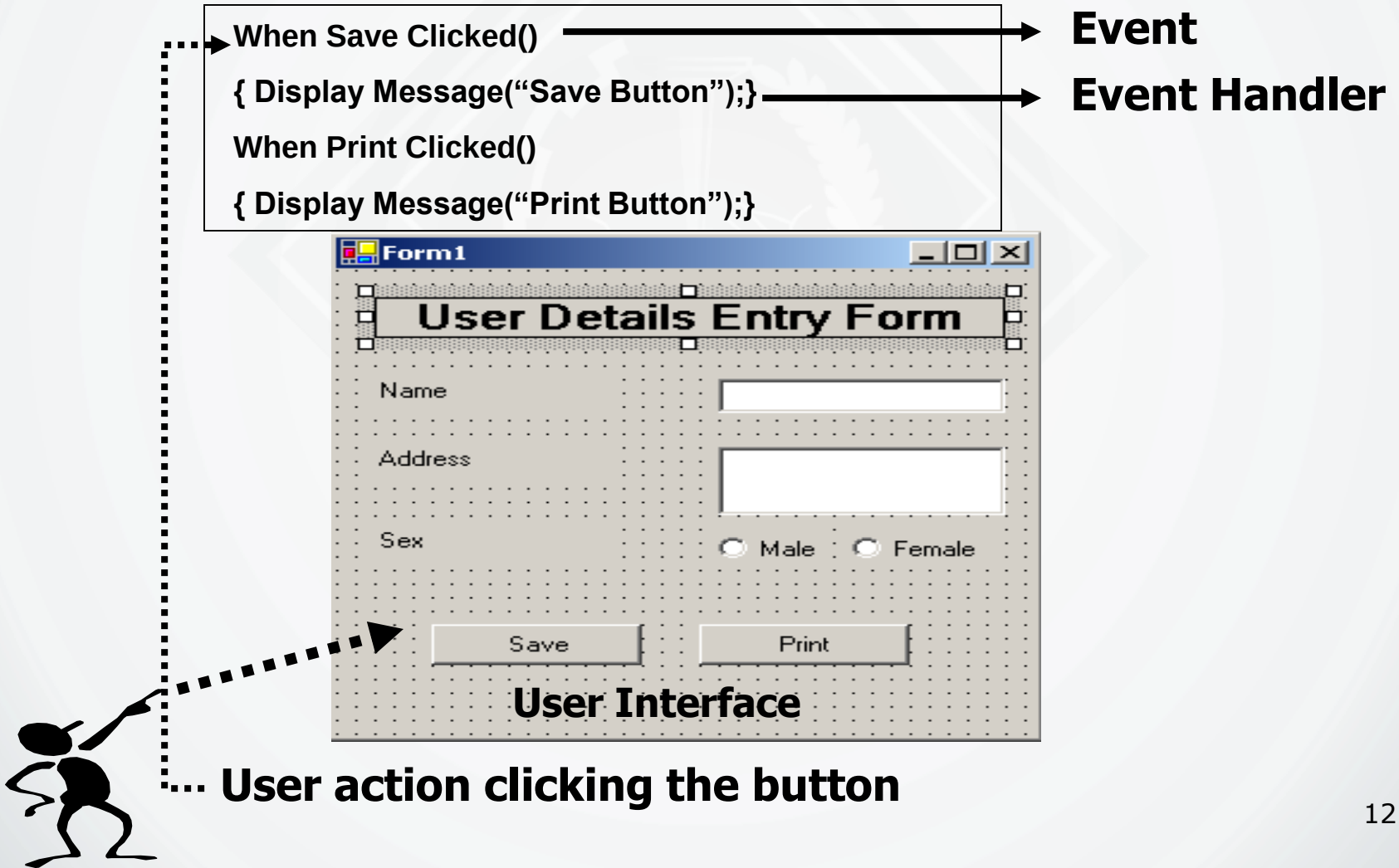



```
private void button1_Click(object sender, System.EventArgs e)
{
    MessageBox.Show("Save");
}
```

Các tập tin của ứng dụng WinForms

*.sln	Tập tin Solution (văn bản)
*.cs	Tập tin chứa định nghĩa về form & mã lệnh bên trong đó
*.resx	Tập tin tài nguyên
*.csproj	Tập tin dự án (tập tin XML)
…	Các tập tin khác

Sự kiện và xử lý sự kiện



Form

- **Form**: cửa sổ được lập trình để hiển thị dữ liệu & nhận thông tin phía người dùng.
- Một form là 1 lớp trong ứng dụng **WinForms**

Form - Thuộc tính

- Một số thuộc tính phổ biến:
  - *BackColor*: màu nền.
  - *ForeColor*: màu chữ.
  - *Text:* Tiêu đề form.
  - *Font*: Font chữ của form.
  - *Name*: tên lớp của form
- Thuộc tính có thể thay đổi nhờ mã lệnh hoặc cửa sổ thuộc tính

Form - Thuộc tính

- Thay đổi thuộc tính của form bằng mã lệnh:

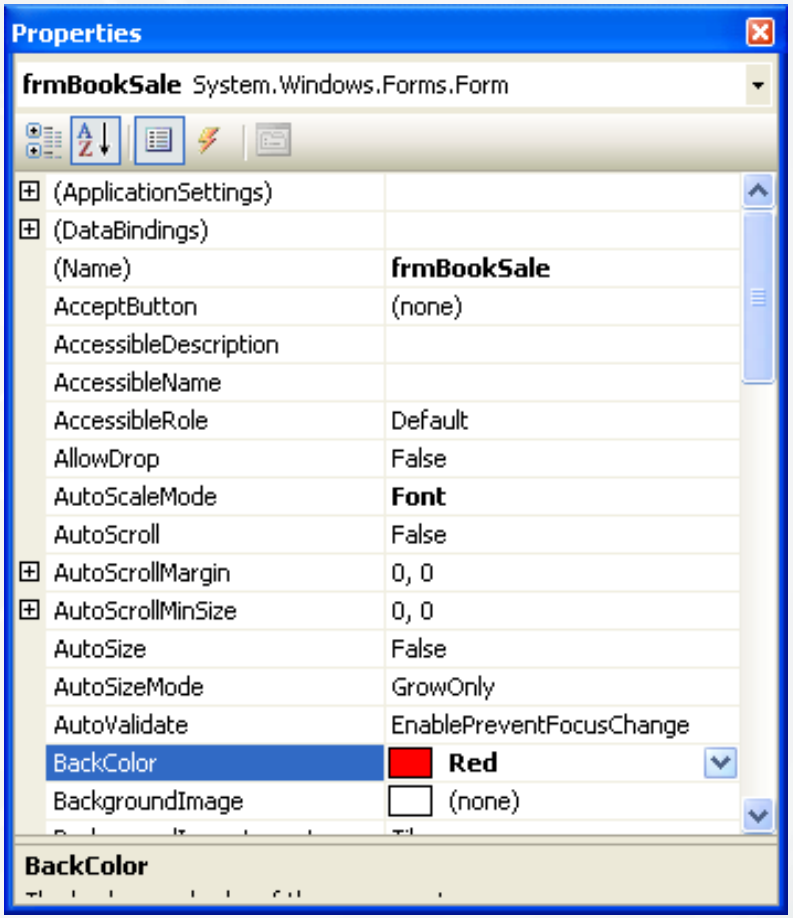
```
<Tên đối tượng form>.<Tên thuộc tính> = <Giá trị>;
```

- Thí dụ:

```
this.BackColor = Color.Red;
```

Form - Thuộc tính

- Thay đổi thuộc tính của form bằng cửa sổ thuộc tính:



Form - Phương thức

- Một số phương thức phổ biến:
  - *Show()*: thể hiện form lên màn hình.
  - *ShowDialog()*: thể hiện form lên màn hình dạng Dialog.
  - *Activate()* : form được kích hoạt (active)
  - *Hide()*: ẩn form đi.
  - *Close()*: đóng lại form và giải phóng tài nguyên.

Form - Phương thức

- Gọi thực thi phương thức:

  <Tên đối tượng form>.<Tên phương thức> (Tham số)

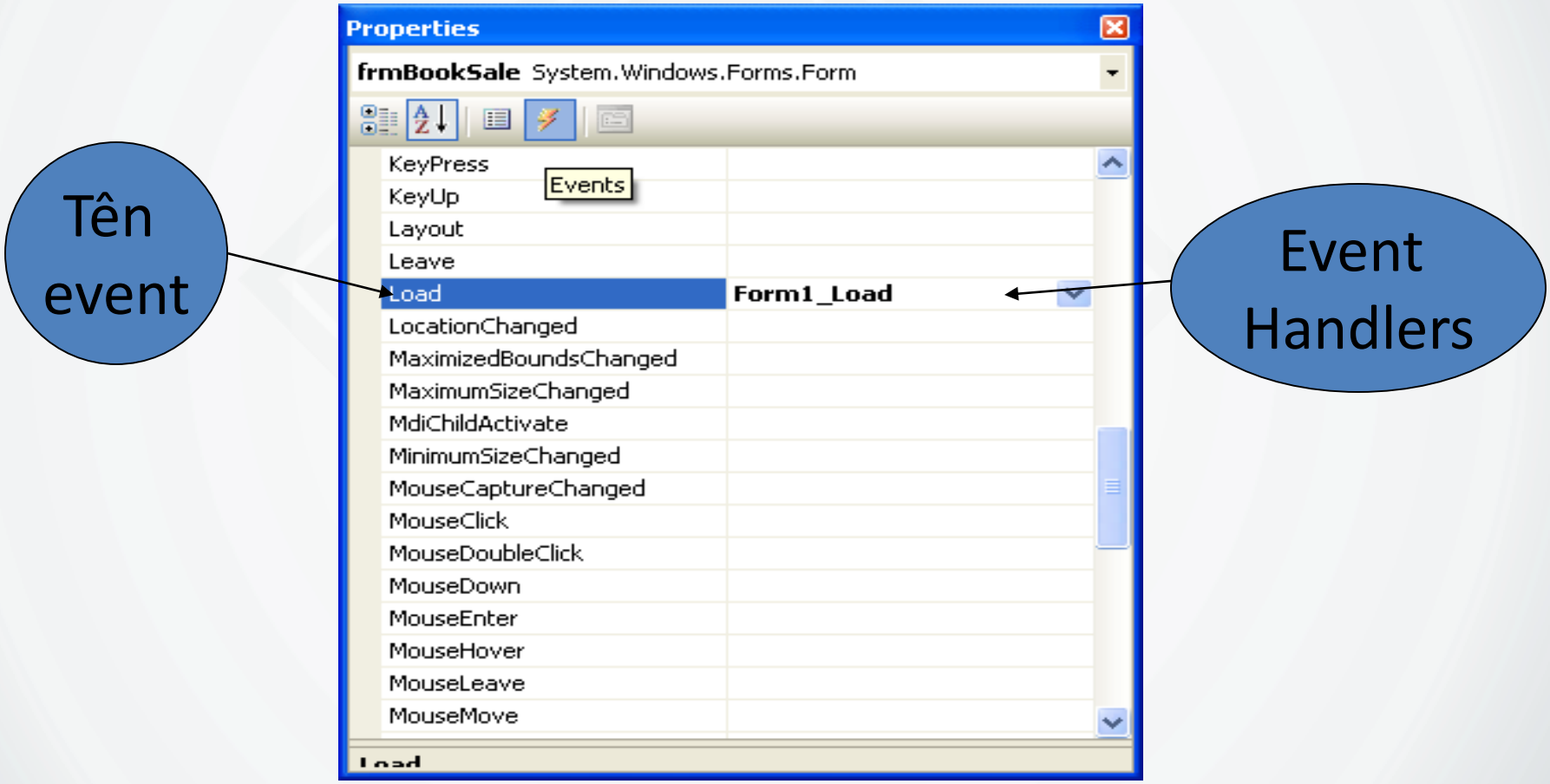
- **Thí dụ**:

```
DialogForm myForm = new DialogForm();
myForm.Show(); //Thể hiện Form trạng thái bình thường
myForm.ShowDialog(); //Thể hiện Form dạng Dialog
myForm.Activate();
myForm.Hide();
myForm.Close();
```

Form - Sự kiện

- Một số sự kiện:
  - *Load* : Xảy ra khi thể hiện đầu tiên của Form được Load lên màn hình.
  - *Activated* : Xảy ra khi Form được kích hoạt.
  - *Deactivated*: Xảy ra khi Form mất kích hoạt.
  - *FormClosing*: Xảy ra khi Form đang trong quá trình đóng lại.
- Lưu ý:
  - Sự kiện *Load* được sử dụng để khởi tạo giá trị của các thành phần trong Form.
  - Sự kiện *FormClosing* & *FormClosed* thường được sử dụng để thu hồi tài nguyên được cấp phát cho Form.

Xử lý sự kiện Form



Xử lý sự kiện Form

```
private void Form1_Load(object sender, EventArgs e)
{
    MessageBox.Show("Form Load");
}
```

Thí dụ xử lý sự kiện Form

- Ngăn thao tác đóng form của người dùng:
  - Xử lý sự kiện FormClosing, đặt thuộc tính Cancel của đối tượng FormClosingEventArgs là true.

```
private void frmBookSale_FormClosing(object sender, FormClosingEventArgs e)
{
    e.Cancel=true;
}
```

Sự liên kết giữa các form trong ứng dụng

- Tạo một đối tượng Form
- Thể hiện Form lên màn hình nhờ phương thức Show()

```
<Lớp form> <Tên đối tượng> = new <Lớp form>();
<Tên đối tượng>.Show();

Thí dụ:
ImageForm NewForm = new ImageForm();
NewForm.Show();
```

Controls

- **Controls**: các thành phần có sẵn để tạo giao diện tương tác với người dùng.
- Trong WinForms, lớp Control là lớp cơ sở cho các controls khác.

Thuộc tính của controls

- Một số thuộc tính phổ biến:

Thuộc tính	Ý nghĩa
BackColor	Màu nền
ForeColor	Màu chữ
Name	Tên của control (tên đối tượng)
Enabled	Control được sử dụng hay không?
Visible	Control có hiển thị hay không?
Text	Chuỗi xuất hiện trên control

Truy cập thành viên của controls

- Truy cập thành viên của 1 control bằng mã lệnh:

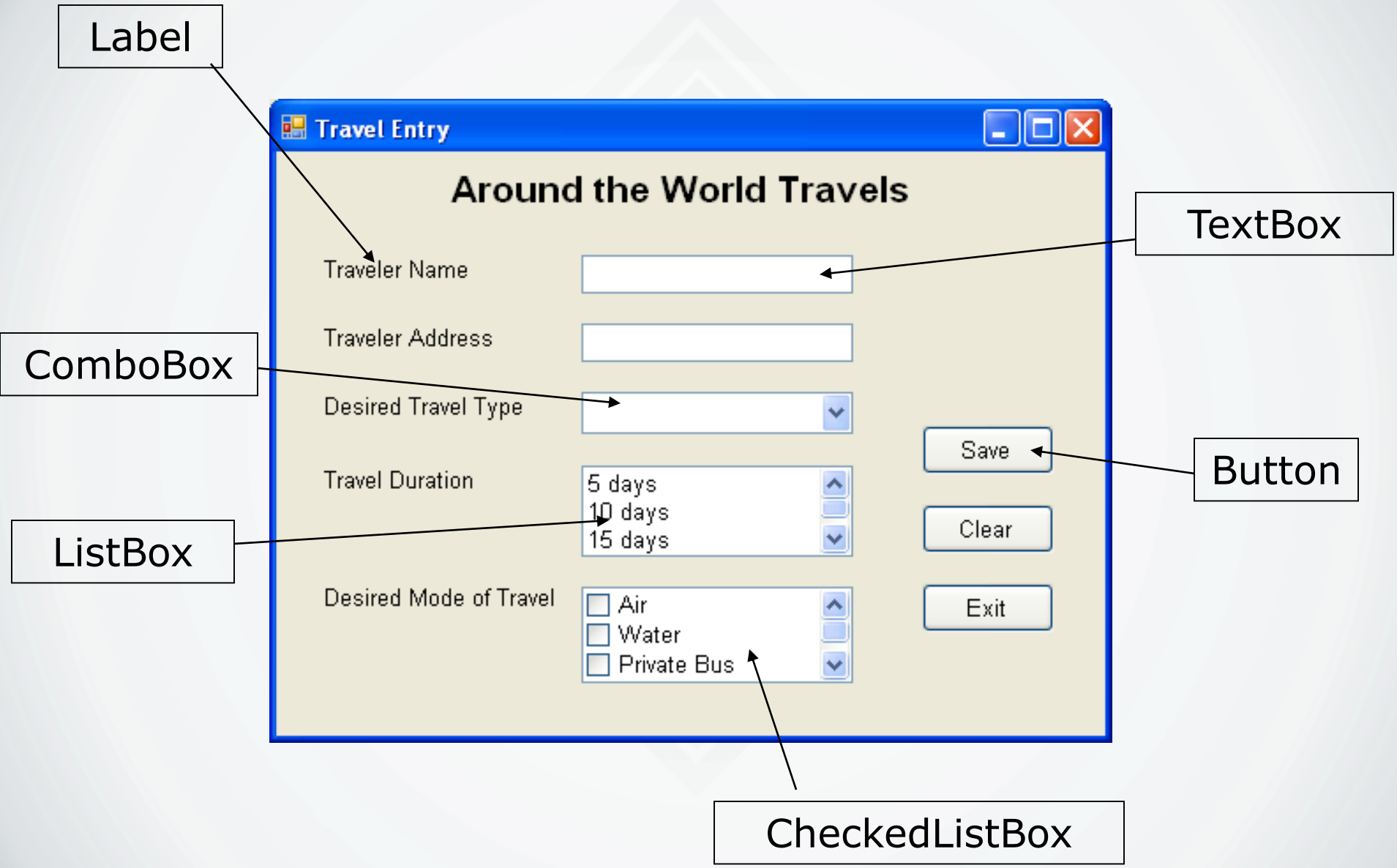
```
<Tên control>.<Tên thành viên>
```

- Thí dụ:

```
button1.Enabled = true;
textBox1.Clear();
```

Controls trên form

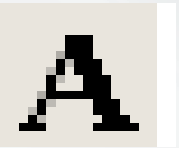
- Khi controls được đặt trên form, controls là dữ liệu thành viên của form đó.
  - Tên dữ liệu thành viên chính là tên controls


Label
Travel Entry
Around the World Travels
TextBox
Traveler Name
Traveler Address
ComboBox
Desired Travel Type
Save
Button
Travel Duration
5 days
10 days
15 days
ListBox
Clear
Desired Mode of Travel
Air
Water
Private Bus
Exit
CheckedListBox

Controls trên form

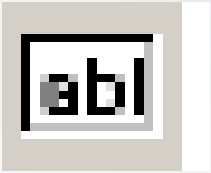
```
partial class frmTravelEntry
{
    /// <summary>
    /// Required designer variable.
    /// </summary>
    private System.ComponentModel.IContainer components = null;

    /// <summary>
    /// Clean up any resources being used.
    /// </summary>
    /// <param name="disposing">true if managed resources should be dis
    protected override void Dispose(bool disposing)...

    Windows Form Designer generated code

    private System.Windows.Forms.Label label1;
    private System.Windows.Forms.Label label2;
    private System.Windows.Forms.TextBox txtName;
    private System.Windows.Forms.TextBox txtAdd;
    private System.Windows.Forms.Label label3;
    private System.Windows.Forms.Label label4;
    private System.Windows.Forms.ComboBox cboTravelType;
    private System.Windows.Forms.Label label5;
    private System.Windows.Forms.ListBox lstDuration;
    private System.Windows.Forms.Label label6;
    private System.Windows.Forms.CheckedListBox lstMode;
    private System.Windows.Forms.Button btnSave;
    private System.Windows.Forms.Button btnClear;
    private System.Windows.Forms.Button btnExit;
}
```

Label



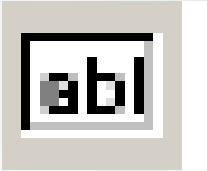
- Dùng để hiển thị dữ liệu cho người dùng.
- Một đối tượng thuộc lớp Label được tạo ra khi 1 **Label** được thêm vào form.

TextBox



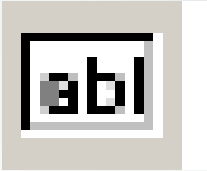
- Dùng để nhận/hiển thị dữ liệu về phía người dùng.
- Một đối tượng thuộc lớp **TextBox** được tạo ra khi 1 **TextBox** được thêm vào form
- **Thuộc tính**
  - *Multiline*: TextBox có nhận nhiều dòng văn bản hay không? (Mặc định giá trị này là **false**).
  - *CharacterCasing*: tự động chuyển đổi dữ liệu nhập của TextBox thành dạng tương ứng (Normal, Upper, Lower).
  - *PasswordChar*: Ký tự hiển thị thay thế khi nhập.
  - *Font*: Font chữ của TextBox

TextBox



- **Phương thức**
  - *Clear():* Xóa trắng TextBox
  - *Focus():* TextBox nhận được tiêu điểm (input focus), sẵn sàng nhận dữ liệu từ bàn phím.
- **Sự kiện**
  - *KeyPress*: Xảy ra khi người dùng nhấn/thả 1 phím có mã Ascii từ bàn phím.
  - *KeyDown*: Xảy ra khi người dùng nhấn 1 phím từ bàn phím
  - *KeyUp*: Xảy ra khi người dùng thả 1 phím từ bàn phím

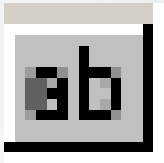
TextBox



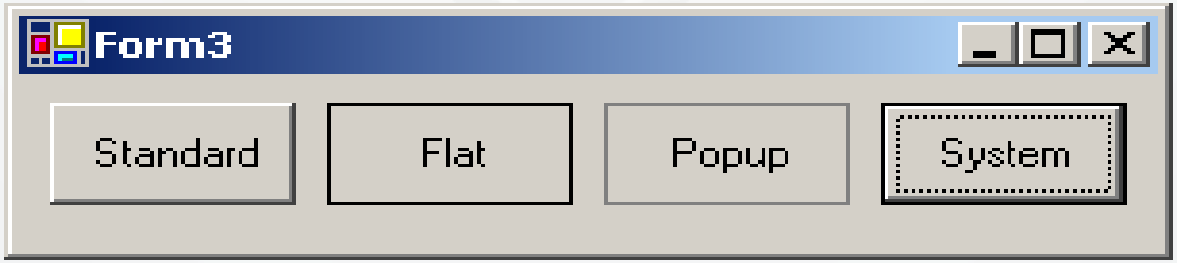
- **Sự kiện**
  - *Leave*: Xảy ra khi tiêu điểm (input focus) vừa thoát khỏi TextBox.
  - *Validating*: Xảy ra trước khi tiêu điểm thoát khỏi TextBox. Có thể dùng để kiểm tra sự hợp lệ của dữ liệu nhập
    - Thí dụ:

```
private void txtName_Validating(object sender, CancelEventArgs e)
{
    if(txtName.Text=="")//Nếu ô nhập là rỗng
        e.Cancel=true;  //  thì không thể di chuyển khỏi ô nhập
}
```

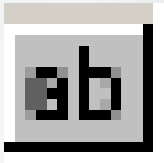
Button



- Dùng để xác nhận hành động hay 1 thao tác của người dùng.
- Một đối tượng **Button** được tạo ra khi 1 **Button** được thêm vào form.
- **Thuộc tính**
  - *Image*: Hình ảnh trên Button.
  - *FlatStyle*: Hình dạng của Button (`Flat`, `Popup`, `Standard`, and `System`.)



Button



- **Phương thức**
  - *Focus()*: Button nhận được tiêu điểm
- **Sự kiện**
  - *Click*: Xảy ra khi button được click chuột vào

Các ListBoxes

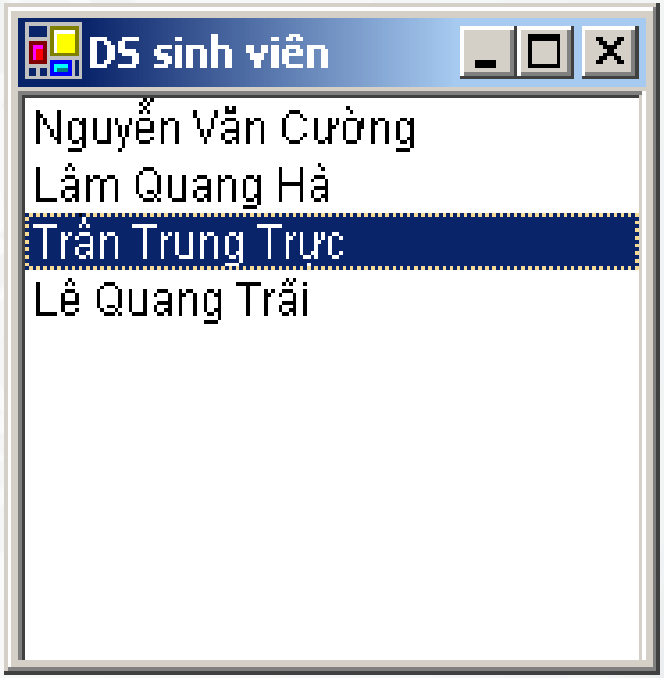
- Sử dụng để hiển thị một danh sách cho người dùng lựa chọn.
- WinForms hỗ trợ 2 loại list box:
  - ListBox 
  - CheckedListBox 
- Sự khác biệt chính là những phần tử của CheckedListBox được hiển thị có dấu check trước đó.

ListBox

- Thuộc tính
  - *Items*: tập các phần tử trong ListBox.
  - *SelectedIndex*: xác định chỉ số của mục đang hiện thời được chọn trong ListBox, phần tử đầu tiên có chỉ số 0.
  - *SelectedItems*: tập các mục đang được chọn.
  - *SelectedValue*: giá trị của mục đang được chọn.
- Phương thức
  - *ClearSelected*: xóa hết các lựa chọn trên ListBox.
  - *SetSelected(int, bool):* chọn hoặc không chọn 1 mục trong danh sách.
  - *GetSelected(int)*: trả về 1 giá trị bool chỉ ra phần tử nhất định có được lựa chọn hay không?

ListBox

- **Sự kiện**
  - *SelectedIndexChanged*: xảy ra khi giá trị của thuộc tính SelectedIndex thay đổi.
  - *SelectedValueChanged*: xảy ra khi giá trị của thuộc tính SelectedValue thay đổi.
- Thí dụ: Sử dụng thuộc tính đối tượng Items để thêm các mục vào ListBox. Xử lý sự kiện Load của Form để viết mã lệnh xử lý.



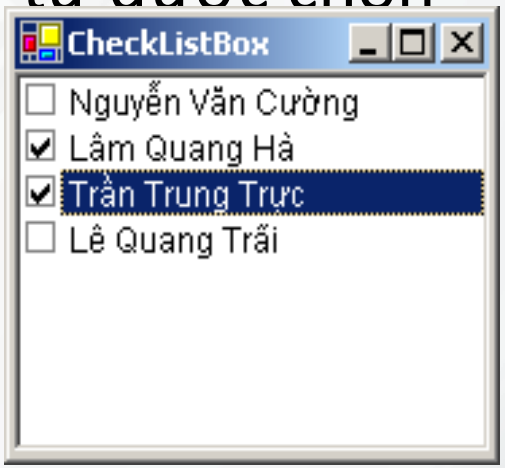
ListBox

```
private void frmTravelEntry_Load(object sender, EventArgs e)
{
    lstName.Items.Add("Nguyễn Văn Cường");
    lstName.Items.Add("Lâm Quang Hà");
    lstName.Items.Add("Trần Trung Trực");
    lstName.Items.Add("Lê Quang Trãi");
}
```

- Ngoài ra thuộc tính đối tượng Items còn có một số thuộc tính & phương thức khác: *Count, Clear, Insert, RemoveAt, Item*…

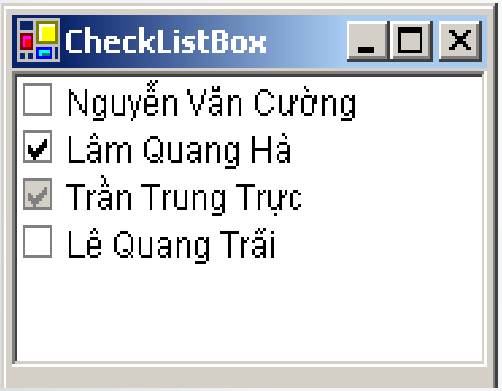
CheckedListBox

- **CheckedListBox** cũng hỗ trợ các thành viên giống như ListBox; bên cạnh đó control này còn có những đặc điểm riêng.
- **Thuộc tính**
  - *CheckedIndies*: tập các chỉ số của các phần tử được lựa chọn (checked).
  - *CheckedItems*: tập hợp các phần tử được chọn (checked).
- Thí dụ: Hình bên
  - CheckedIndies={1,2}
  - CheckedItems={Lâm Quang Hà, Trần Trung Trực}



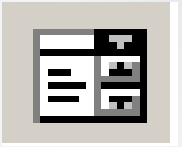
CheckedListBox

- **Phương thức**
  - *SetItemChecked*: thiết lập mục có chỉ số xác định được chọn (checked).
  - *GetItemChecked*: kiểm tra mục có chỉ số xác định có checked hay không?
  - *SetItemCheckState*: xác lập trạng thái checked của 1 mục xác định.
- Thí dụ: Sử dụng phương thức SetItemCheckState




```
lstName.SetItemCheckState(1,CheckState.Checked);
lstName.SetItemCheckState(2,CheckState.Indeterminate);
lstName.SetItemCheckState(3,CheckState.Unchecked);
```

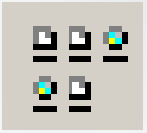
ComboBox



- Là control kết hợp giữa ***TextBox*** và ***ListBox***
- Một đối tượng **ComboBox** được tạo ra khi 1 **ComboBox** được đặt lên form
- **Thuộc tính**
  - *DropDownStyle*: kiểu của ComboBox; gồm có Simple, DropDown, DropDownList.

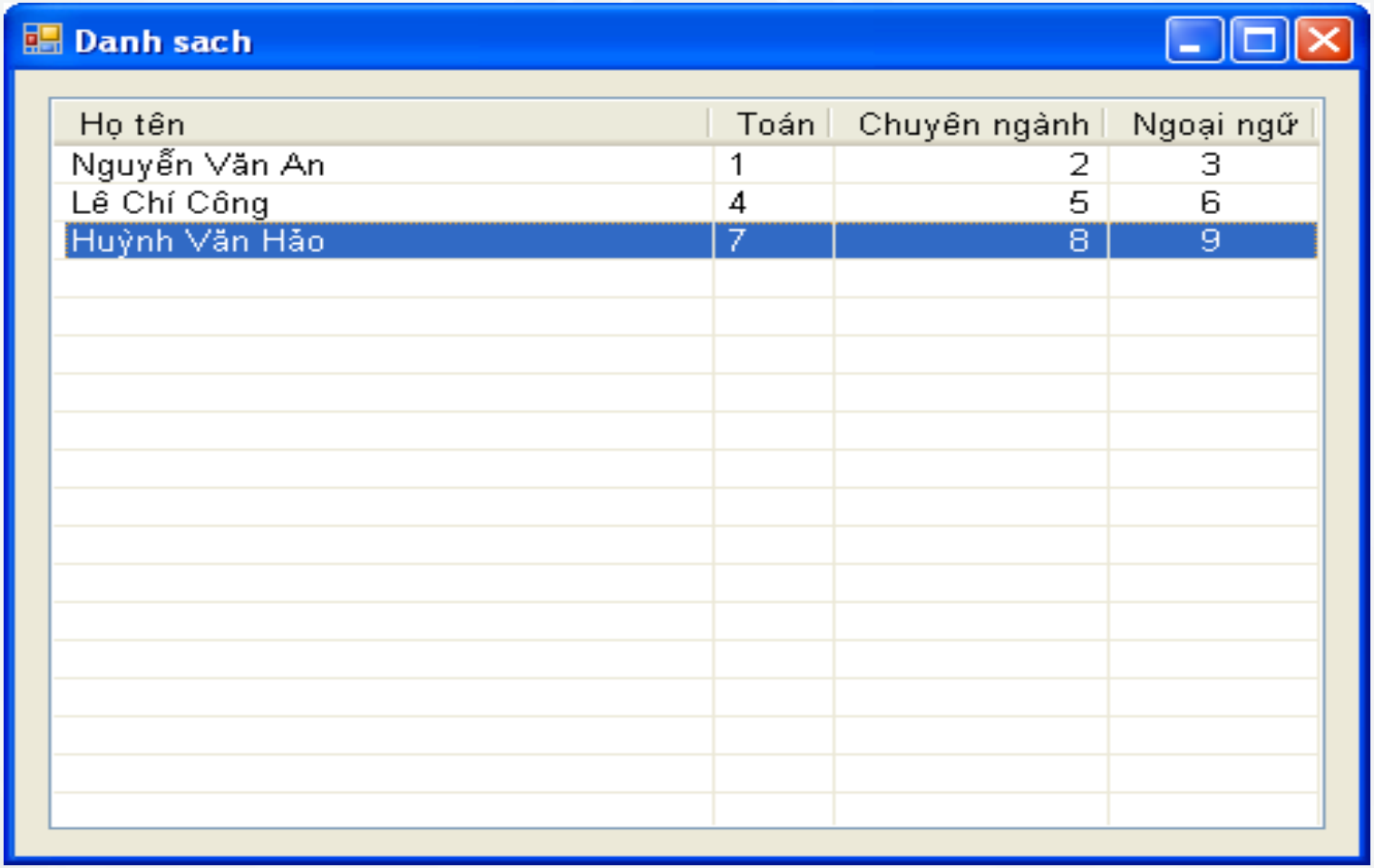
Simple DropDown DropDownList

ListView



- Cho phép hiển thị một danh sách các phần tử ở dạng thức đặc biệt. ListView có 4 kiểu sau:
  - *Text only*: mặc định.
  - Text with small icon.
  - Text with large icon.
  - Detail view: các phần tử hiển thị nhiều cột.
- ListView kế thừa các tính năng của CheckedListBox.
- **Thuộc tính**
  - *CheckBoxes*: mỗi phần tử trong ds có 1 checkbox đứng trước.
  - *Items*: tập các phần tử trong ds.
  - *MultiSelect*: ds có nhiều lựa chọn không?

ListView


Danh sach
Họ tên
Toán
Chuyên ngành
Ngoại ngữ
Nguyễn Văn An
1
2
3
Lê Chí Công
4
5
6
Huỳnh Văn Hảo
7
8
9

```
// Hiển thị chi tiết
lsvName.View = View.Details;
//Sửa giá trị của Text
lsvName.LabelEdit = true;
lsvName.FullRowSelect = true;
// Hiển thị lưới
lsvName.GridLines = true;
ListViewItem item1 = new ListViewItem("Nguyễn Văn An", 0);
item1.SubItems.Add("1");
item1.SubItems.Add("2");
item1.SubItems.Add("3");

ListViewItem item2 = new ListViewItem("Lê Chí Công", 1);
item2.SubItems.Add("4");
item2.SubItems.Add("5");
item2.SubItems.Add("6");

ListViewItem item3 = new ListViewItem("Huỳnh Văn Hảo", 2);
item3.SubItems.Add("7");
item3.SubItems.Add("8");
item3.SubItems.Add("9");

// Tạo các cột cho các mục
lsvName.Columns.Add("Họ tên", -2, HorizontalAlignment.Left);
lsvName.Columns.Add("Toán", -2, HorizontalAlignment.Left);
lsvName.Columns.Add("Chuyên ngành", -2, HorizontalAlignment.Right);
lsvName.Columns.Add("Ngoại ngữ", -2, HorizontalAlignment.Center);
// Thêm các mục vào ListView
lsvName.Items.AddRange(new ListViewItem[] {item1, item2, item3});
```

TreeView

- Cho phép hiển thị dữ liệu ở dạng phân cấp.
- **TreeView** hỗ trợ hầu hết các thành viên của ListView.
- **Thuộc tính**

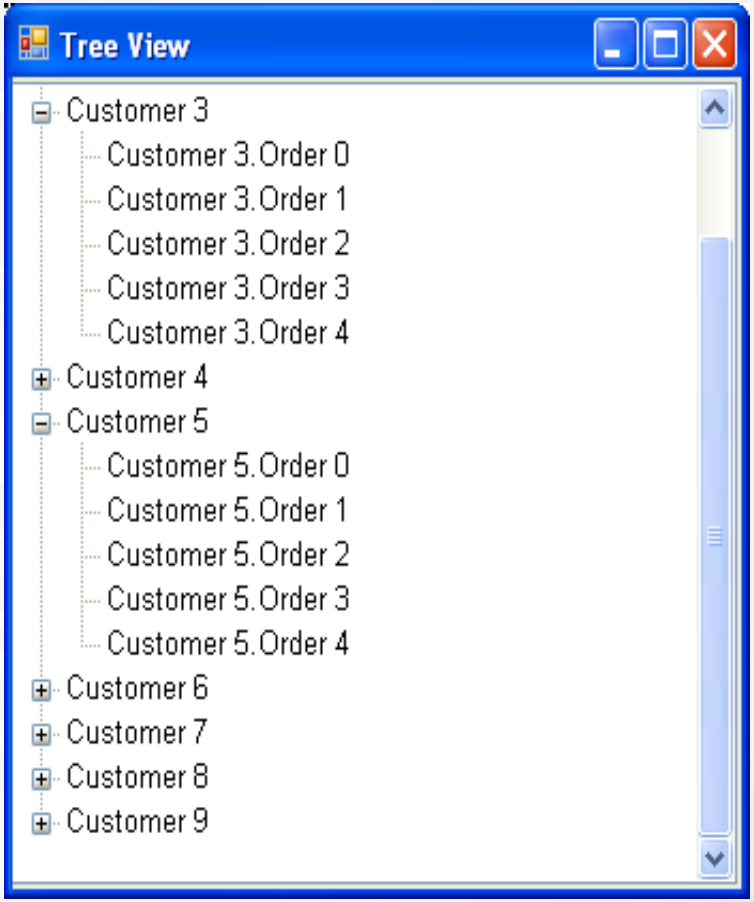
  - *ImageList*: ds các ảnh được hiển thị ở mỗi nút.
  - *Nodes*: ds các nút.
  - *SeletedNode*: nút hiện thời được lựa chọn.

TreeView

- **Phương thức**
  - *GetNodeAt*: truy xuất 1 nút ở vị trí xác định trong TreeView.
  - *GetNodeCount*: tổng số nút.
- **Sự kiện**
  - *BeforeSelect*, *AfterSelect*: xảy ra trước/sau khi 1 nút được chọn.
  - *BeforeCollapse*, *AfterCollapse*: xảy ra trước/sau khi thu hẹp 1 nút.
  - *BeforeExpand*, *AfterExpand*: xảy ra trước/sau khi 1 mở rộng 1 nút.

TreeView

- Hiển thị 10 khách hàng, mỗi khách hàng có 5 đơn đặt hàng.
- Thiết kế các lớp Khách hàng & Đặt hàng.
- Tạo TreeView như hình

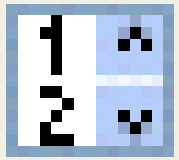


TreeView

```
class clsCustomer
{
    public string CustomerName;
    public ArrayList CustomerOrder;

    public clsCustomer(string Name)
    {
        CustomerName = Name;
        CustomerOrder = new ArrayList();
    }
}

class clsOrder
{
    public string OrderID;
    public clsOrder(string id)
    {
        OrderID=id;
    }
}
```

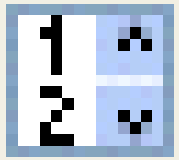
```
private void frmTreeView_Load(object sender, EventArgs e)
{
    ArrayList customerArray = new ArrayList();
    // Thêm khách hàng
    for(int x = 0; x<=9;x++)
        customerArray.Add(new clsCustomer("Customer " + x.ToString()));

    foreach(clsCustomer customer1 in customerArray)
        for(int y = 0; y<=4; y++)
            customer1.CustomerOrder.Add(new clsOrder("Order " + y.ToString()));


    // Xóa TreeView
    trvCusomer.Nodes.Clear();
    // Tạo các nút cha là các khách hàng

    foreach(clsCustomer customer1 in customerArray)
    {
        trvCusomer.Nodes.Add(new TreeNode(customer1.CustomerName));
        // Các nút con là các đơn đặt hàng

        foreach(clsOrder order1 in customer1.CustomerOrder)
            trvCusomer.Nodes[customerArray.IndexOf(customer1)].Nodes.Add
                    (new TreeNode(customer1.CustomerName + "."  + order1.OrderID));
    }
}
```

NumericUpDown



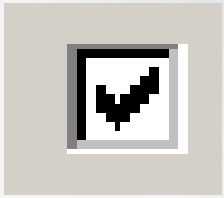
- Cho phép hiển thị một số có thể tăng hay giảm đến một giá trị xác định.
- Một đối tượng NumericUpDown được tạo ra khi 1 NumericUpDown được đặt lên form.
- **Thuộc tính**
  - *Increment*: Giá trị số mỗi lần NumericUpDown tăng hoặc giảm.
  - *Maximum, Minimum*: giá trị lớn nhất (nhỏ nhất) của NumericUpDown.
  - *Value*: giá trị hiện hành của NumericUpDown

NumericUpDown



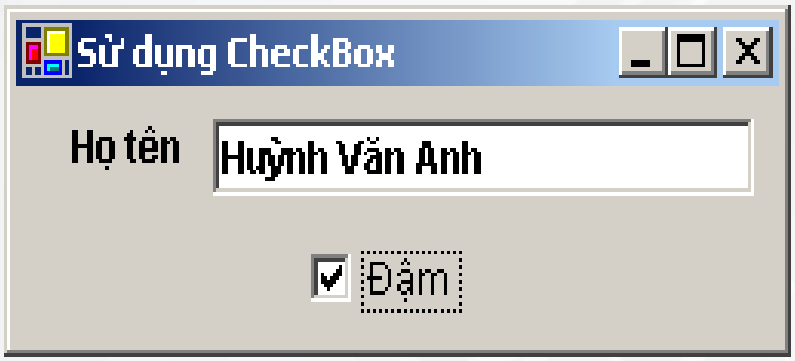
- **Phương thức**
  - *DownButton, UpButton*: giảm (tăng) giá trị hiện hành của NumericUpDown 1 lượng bằng Increment.
- **Sự kiện**
  - *ValueChanged*: xảy ra khi NumericUpDown thay đổi giá trị

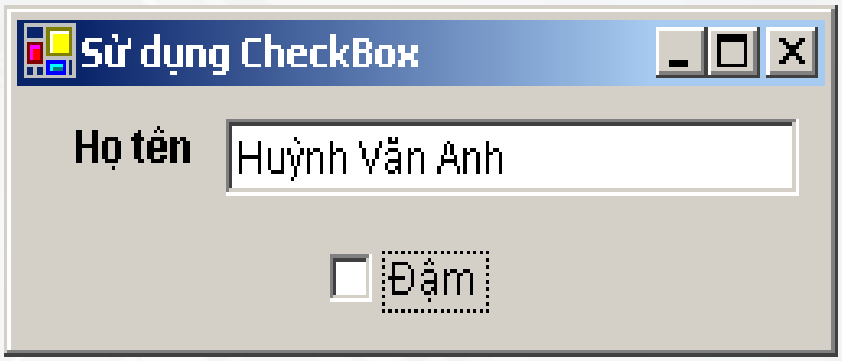
CheckedBox



- Được sử dụng để nhận thông tin dạng Yes/No.
- Một đối tượng CheckedBox được tạo ra khi 1 CheckedBox được đặt lên form.
- **Thuộc tính**
  - *Checked*: Xác định control đang ở trạng thái nào (true/false).
- **Sự kiện**
  - *CheckedChanged*: Xảy ra khi giá trị của thuộc tính Checked bị thay đổi.

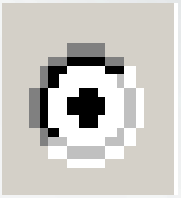
CheckedBox





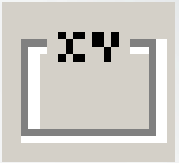

```
private void chkDam_CheckedChanged(object sender, EventArgs e)
{
    if(chkDam.Checked)
        txtName.Font = new Font("Arial",10.0F,FontStyle.Bold);
    else
        txtName.Font = new Font("Arial",10.0F,FontStyle.Regular);
}
```

RadioButton

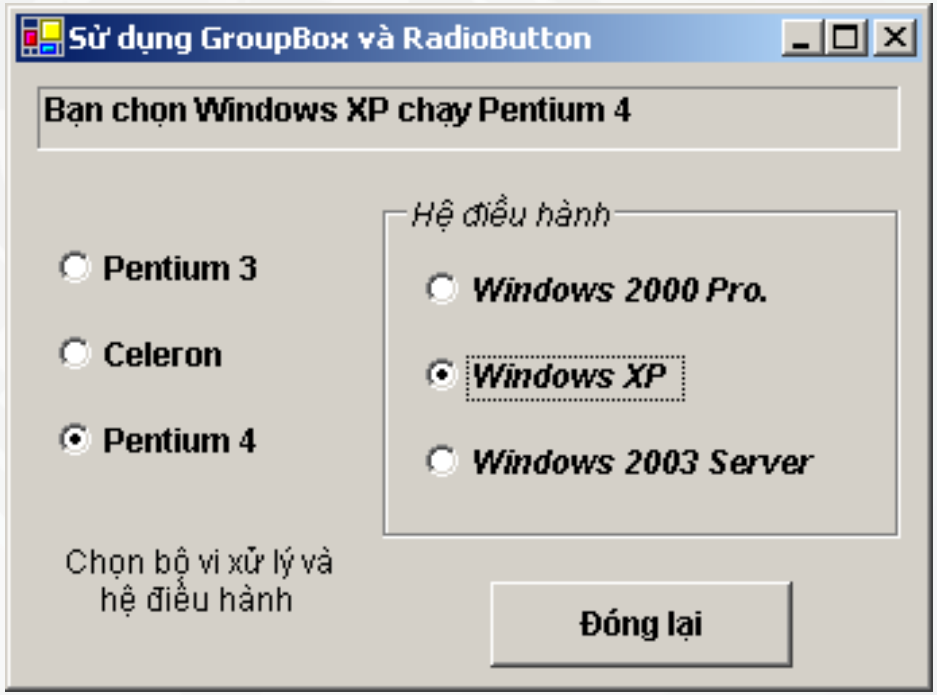


- Là control cho phép nhận dữ liệu Yes/No, nhưng các **RadioButton** cùng 1 nhóm thì không thể chọn đồng thời.
- Một đối tượng **RadioButton** được tạo ra khi 1 **RadioButton** được đặt lên form.
- Các thuộc tính, phương thức, sự kiện tương tự **CheckedBox**.

GroupBox



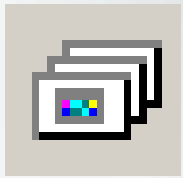
- Là control dùng để nhóm các controls khác.
- **Thí dụ**:



PictureBox

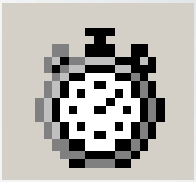
- Dùng để hiển thị hình ảnh
- Một đối tượng **PictureBox** được tạo ra khi 1 **PictureBox** được đặt lên form.
- **Thuộc tính**
  - *Image*: Xác định đối tượng hình ảnh cần hiển thị.
  - *SizeMode*: xác định cách thức ảnh được hiển thị (*AutoSize, CenterImage, Normal, StretchImage*)

ImageList



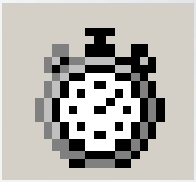
- Dùng để lưu các hình ảnh và cho phép chúng hiển thị ở các controls khác nhau.
- Một đối tượng **ImageList** được tạo ra khi một **ImageList** được đặt lên form.
- **Thuộc tính**
  - *Images*: tập hợp các ảnh của ImageList.
  - *ImageSize*: kích thước của các ảnh có trong ImageList.

Timer

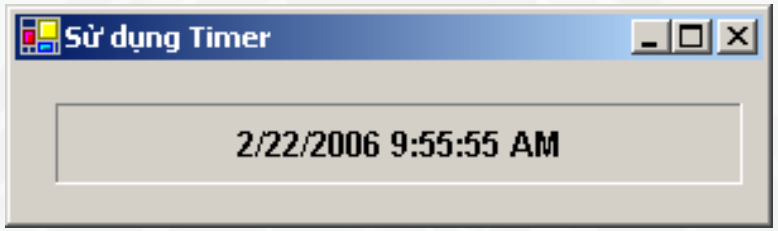


- Là control đáp ứng lại sự trôi đi của thời gian.
- **Thuộc tính**
  - *Enabled*: cho phép control thời gian thực thi.
  - *Interval*: Khoảng thời gian giữa 2 sự kiện.
- **Phương thức**
  - *Start*: khởi động control thời gian.
  - *Stop*: ngưng control thời gian.
- **Sự kiện**
  - *Tick*: Xảy ra sau 1 khoảng thời gian được chỉ định trong Interval

Timer



- **Thí dụ**: Đồng hồ số:



- Thuộc tính: Interval = 1000
  Enabled = true
- Sự kiện Tick:

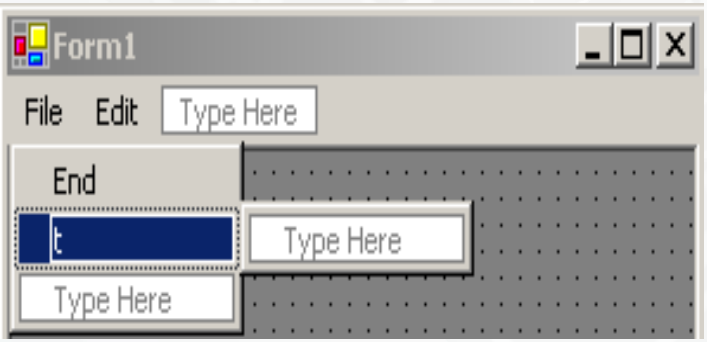
```
private void tmrDem_Tick(object sender, EventArgs e)
{
    lblTG.Text = DateTime.Now.ToString();
}
```

Menu

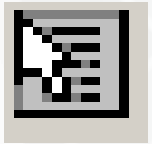
Types of Menu → ContextMenuStrip

MenuStrip

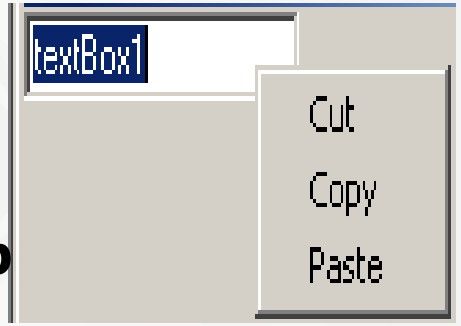
Trong hộp công cụ



Trong ứng dụng MDI khi thiết kế



Trong hộp công cụ

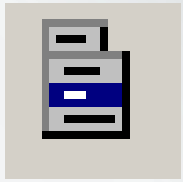


Lúc thực thi ứng dụng

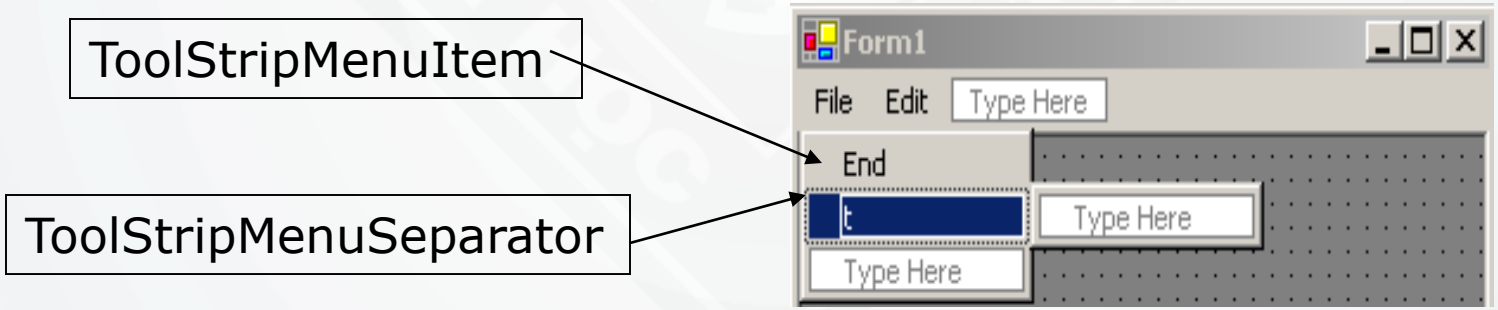
Menu

- Menu là một loại control trong đó người dùng có thể lựa chọn các mục để thực thi từ một danh sách cho trước.
- Phân loại
  - ***Menu thả xuống*** (MenuStrip): là dạng menu thông dụng nhất.
  - ***Menu bật ra*** (ContextMenuStrip): thường hiển thị khi ta ấn nút phải chuột.

MenuStrip

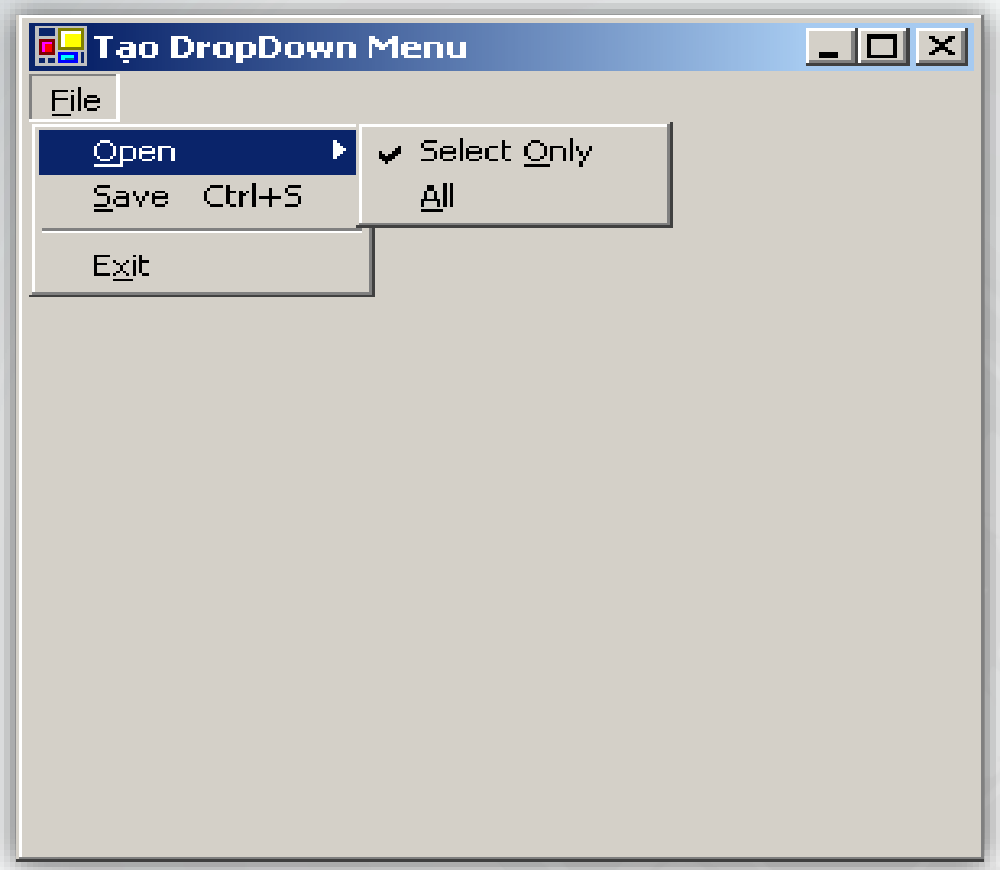


- MenuStrip là lớp cho phép tạo các menu.
  - Các mục con của 1 đối tượng **MenuStrip** là ***ToolStripMenuItem*** hoặc ***ToolStripMenuSeparator***



ToolStripMenuItem

- Thuộc tính
  - *Checked*: là **True** nếu có dấu ✓ trước mục menu con.
  - *Image*: biểu tượng ảnh xuất hiện trước mỗi mục menu con.
  - *ShortcutKeys*: xác lập phím tắt
- Sự kiện
  - *Click*: Xảy ra khi người dùng Click chọn mục menu.




```
private void mnuSave_Click(object sender, EventArgs e)
{
    MessageBox.Show("Ban chon File/Save", "Thong bao");
}

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

ContextMenuStrip

- Là control cho phép người dùng truy cập các mục trên menu nhờ thao tác nhấp chuột phải.
- Sau khi **ContextMenuStrip** được tạo ra, chúng sẽ được gắn với một control khác nhờ vào thuộc tính **ContextMenuStrip** của control đó.
- Cách thức tạo **ContextMenuStrip** tương tự cách thức tạo **MenuStrip**

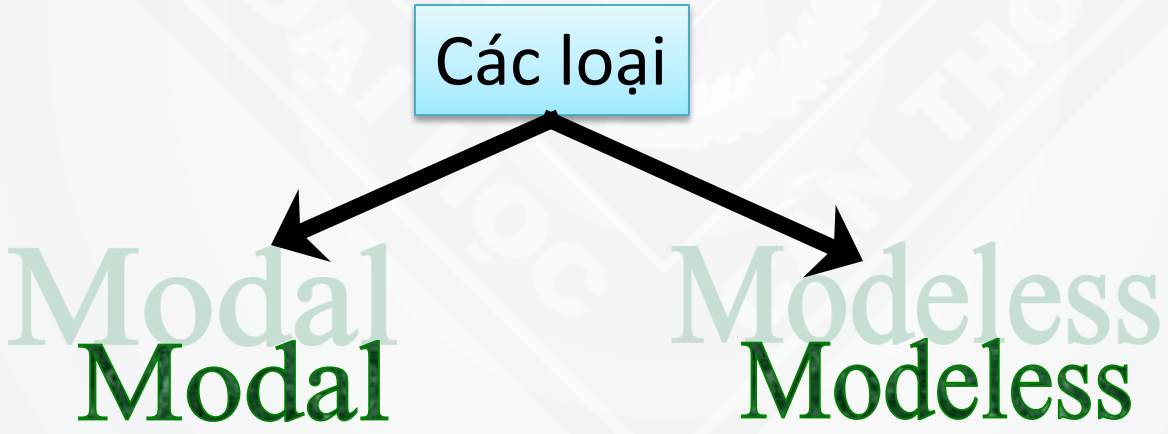
Một số controls khác

- DateTimePicker
- LinkedLabel
- TrackBar
- Panel
- TabControl
- RichTextBox
- ProgressBar
- ToolTip
- ToolBar
- StatusBar

Hộp thoại

Sử dụng

- **Hiển thị thông điệp đến người dùng**
- **Nhận thông tin từ phía người dùng**



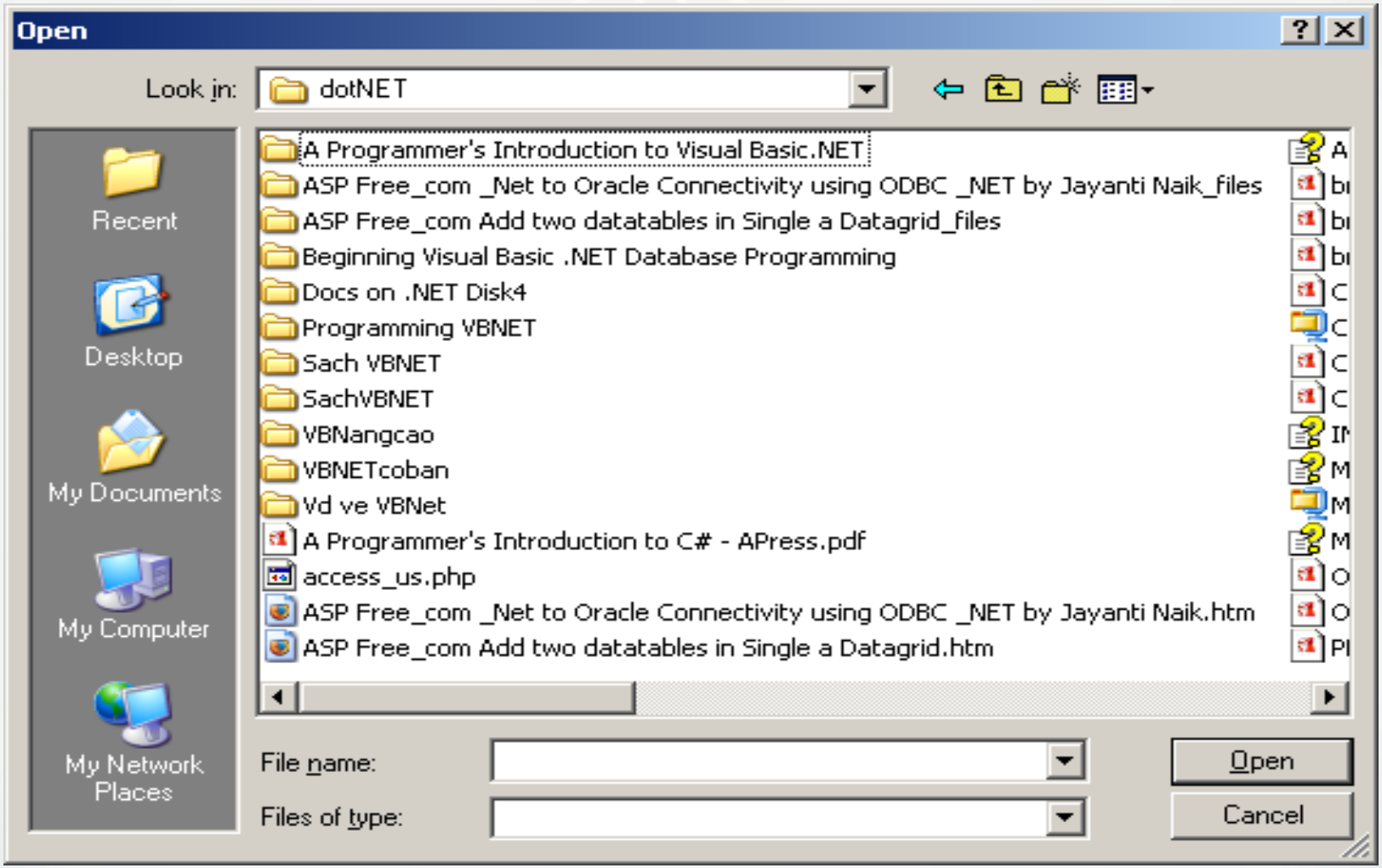
Các loại hộp thoại

Custom dialog boxes

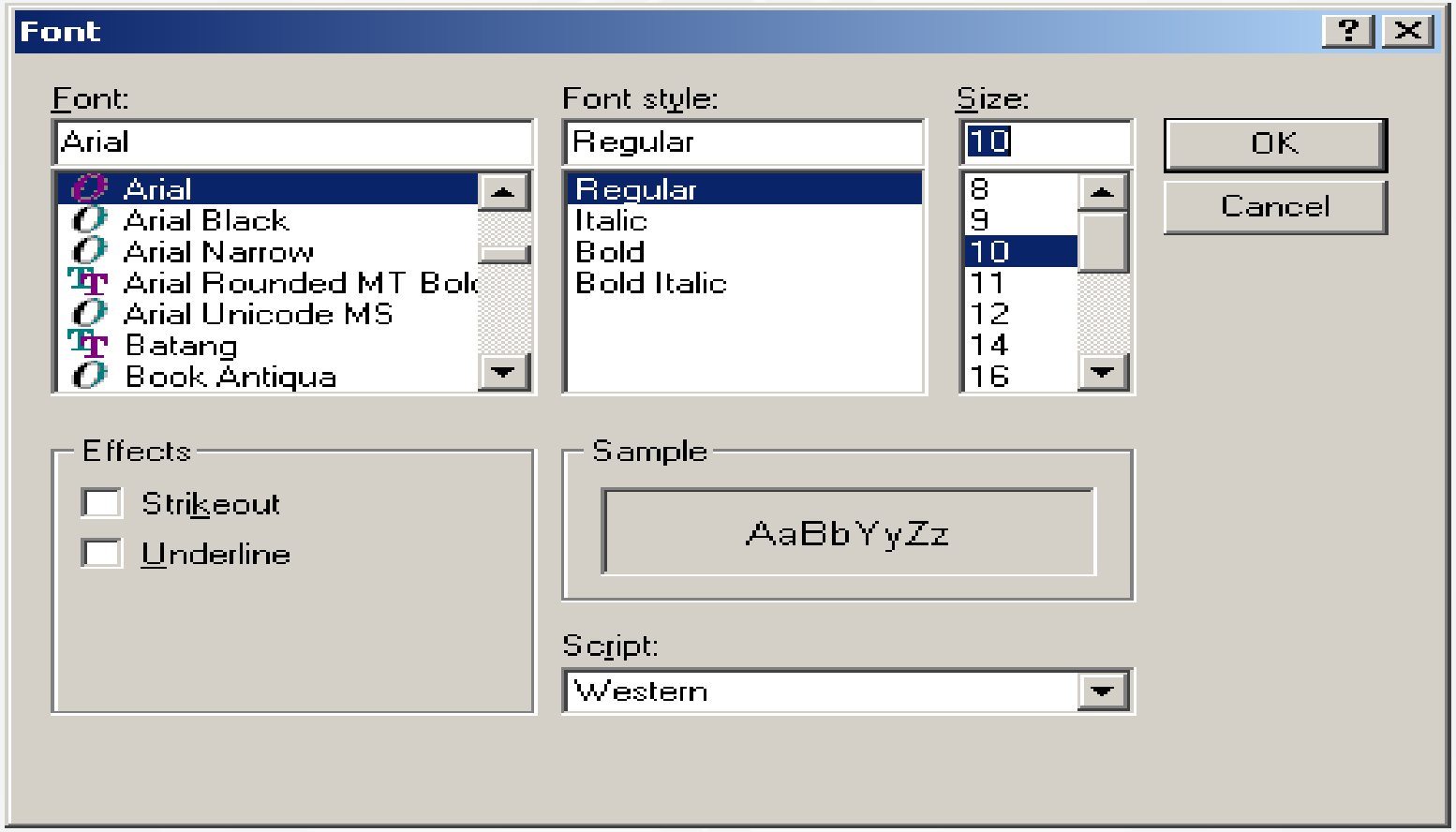
Common dialog boxes

- OpenFileDialog
- PageSetUpDialog
- FontDialog
- ColorDialog
- SaveFileDialog
- PrintPreviewDialog
- PrintDialog

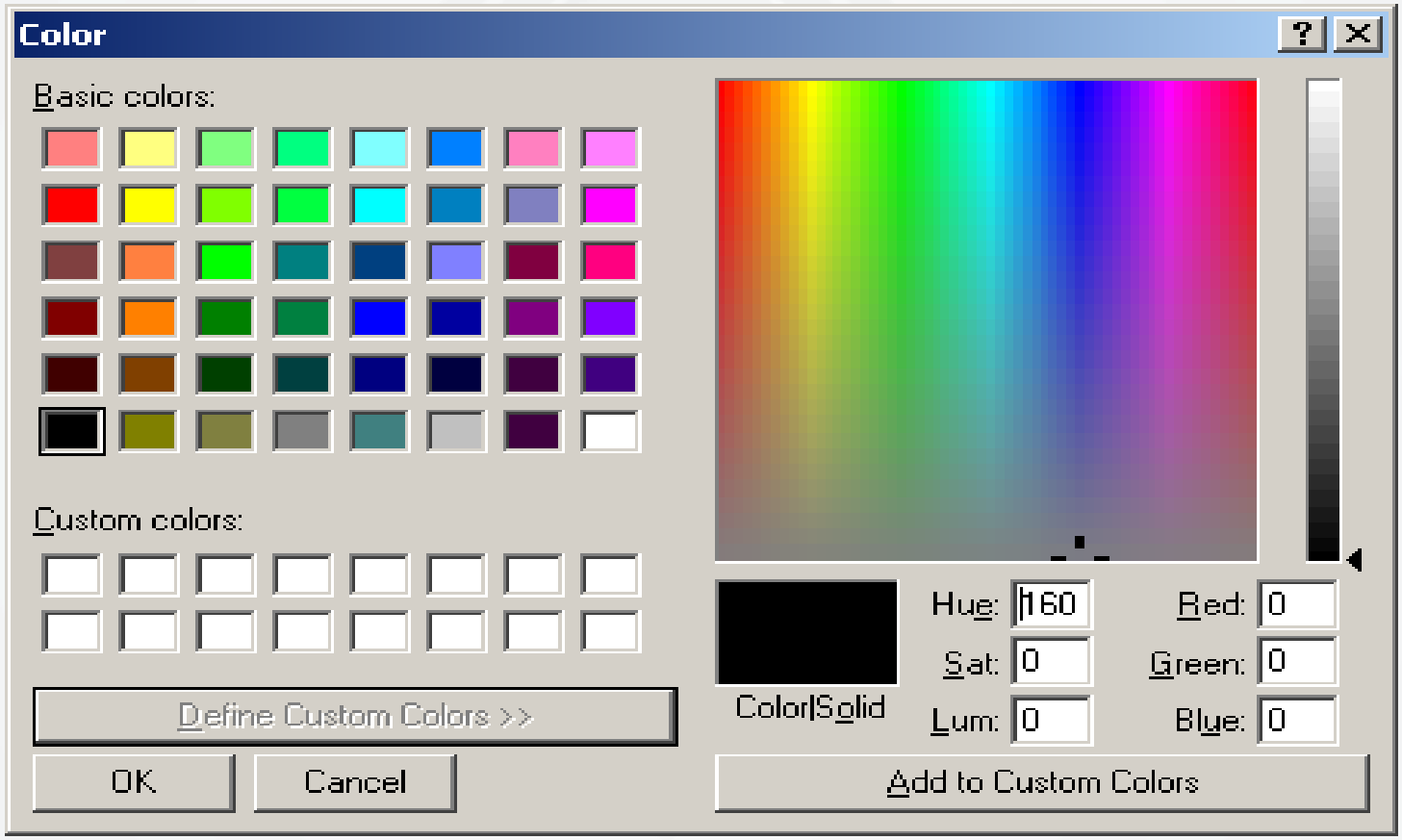
OpenFileDialog



FontDialog


Font
Font:
Arial
Arial
Arial Black
Arial Narrow
Arial Rounded MT Bol
Arial Unicode MS
Batang
Book Antiqua
Font style:
Regular
Regular
Italic
Bold
Bold Italic
Size:
10
8
9
10
11
12
14
16
OK
Cancel
Effects
Strikeout
Underline
Sample
AaBbYyZz
Script:
Western

ColorDialog



Hộp thông điệp - MessageBox

- Là loại hộp thoại cho phép hiển thị các thông điệp.

Đoạn mã hiển thị hộp thông điệp:

```
MessageBox.Show("[Message]");
```

- Phương thức tĩnh Show() của lớp MessageBox có 12 kiểu khác nhau (nạp chồng)

Hộp thông điệp - MessageBox

Overload
public static DialogResult Show(string);
public static DialogResult Show(IWin32Window, string);
public static DialogResult Show(string, string);
public static DialogResult Show(IWin32Window, string, string);
public static DialogResult Show(string, string, MessageBoxButtons);
public static DialogResult Show(IWin32Window, string string, MessageBoxButtons);
public static DialogResult Show(string, string, MessageBoxButtons, MessageBoxIcon);

Hộp thông điệp - MessageBox

Overload
public static DialogResult Show(IWin32Window, string, string, string, MessageBoxButtons, MessageBoxIcon);
public static dialogResultShow(string, string, MessageBoxButtons, MessageBoxIcon, MessageBoxDefaultButton);
public static DialogResult Show(IWin32Window, string, string, MessageBoxButtons, MessageBoxIcon, MessageBoxDefaultButton);
public static DialogResult Show(string, string, MessageBoxButtons, MessageBoxIcon, MessageBoxDefaultButton, MessageBoxOptions);
public static DialogResult Show(IWin32Window, string, string, MessageBoxIcon, MessageBoxDefaultButton, MessageBoxOptions);

Hộp thông điệp - MessageBox

- Sinh viên tự tìm hiểu:
  - *IWin32Window*
  - *MessageBoxButtons*
  - *MessageBoxIcon*
  - *MessageBoxDefaultButton*
  - *MessageBoxOptions*

Sự hợp lệ của dữ liệu

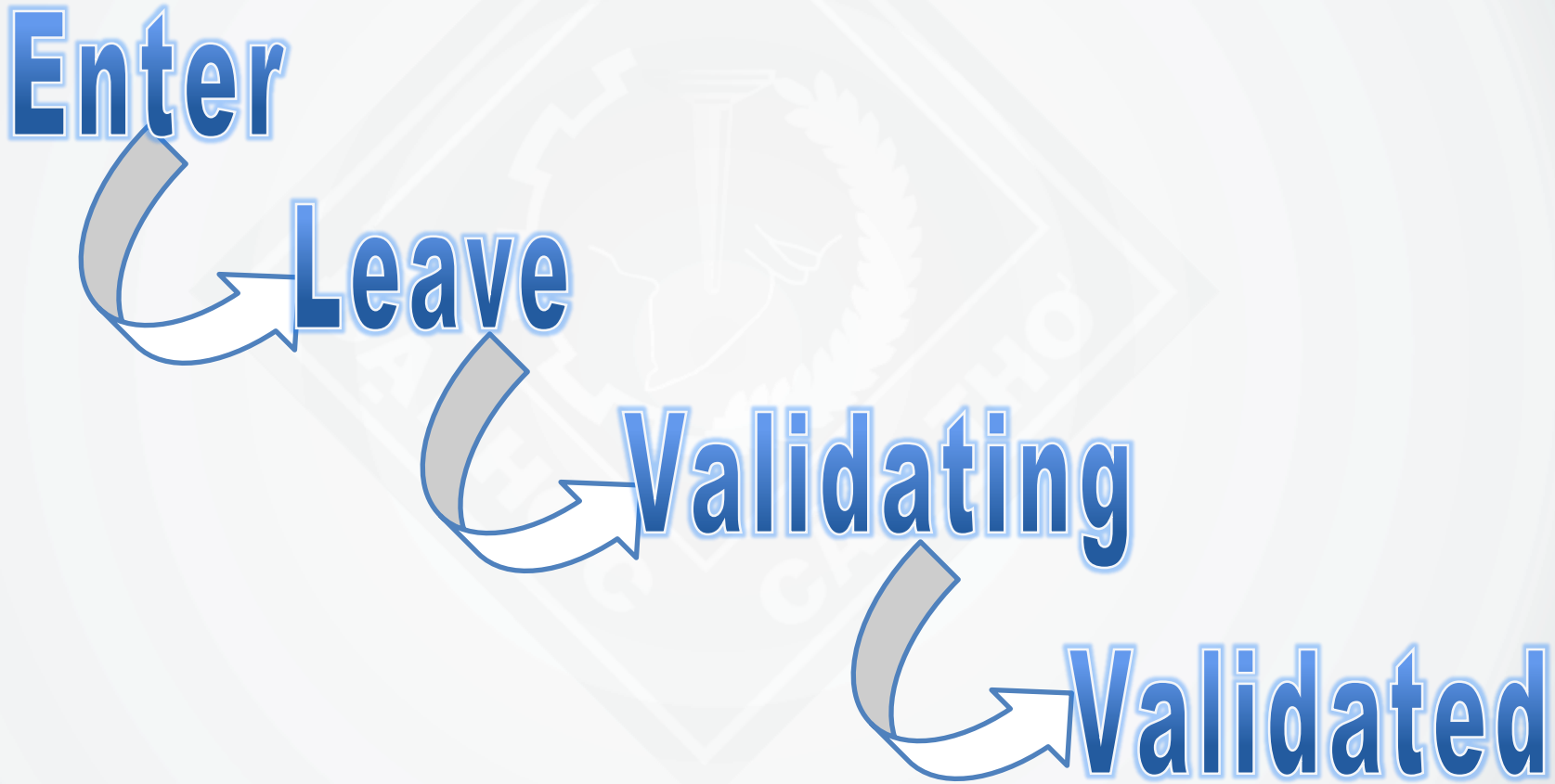
Một số sự kiện được dùng:

- Enter
- Leave
- Validating
- Validated

Các loại kiểm tra:

- ✓ Sự hợp lệ ở mức form
- ✓ Sự hợp lệ ở mức trường

Thứ tự các sự kiện



Sự hợp lệ mức trường

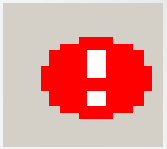
- Kiểm tra sự hợp lệ ở từng trường.
- Sử dụng các sự kiện *Enter, Leave, Validating, Validated*.
- Một số sự kiện bàn phím: *KeyPress, KeyDown, KeyUp*
- Sử dụng một số thuộc tính của control lúc thiết kế: *MaxLength, Locked, PasswordChar*.

Một số phương thức của lớp Char

Sự hợp lệ ở mức form

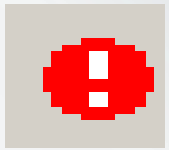
- Kiểm tra sự hợp lệ của tất cả các trường trên form ở 1 thời điểm nào đó.
- Cho phép kích hoạt hay không kích hoạt 1 thành phần nào đó.

ErrorProvider Control



- Sử dụng để hiển thị icon lỗi khi người dùng thao tác lỗi trên một control nào đó trên form.
- **Thuộc tính**
  - *BlinkRate*: xác định tốc độ nhấp nháy của icon lỗi.
  - *BlinkStyle*: xác định cách thức mà icon nhấp nháy trên control (*AlwaysBlink, BlinkDifferentError & NeverBlink*).
  - *Icon*: đường dẫn chỉ đến icon lỗi.
- **Phương thức**
  - *GetError*: nhận về chuỗi thông báo lỗi.
  - *SetError*: xác lập chuỗi thông báo lỗi trên một control xác định.

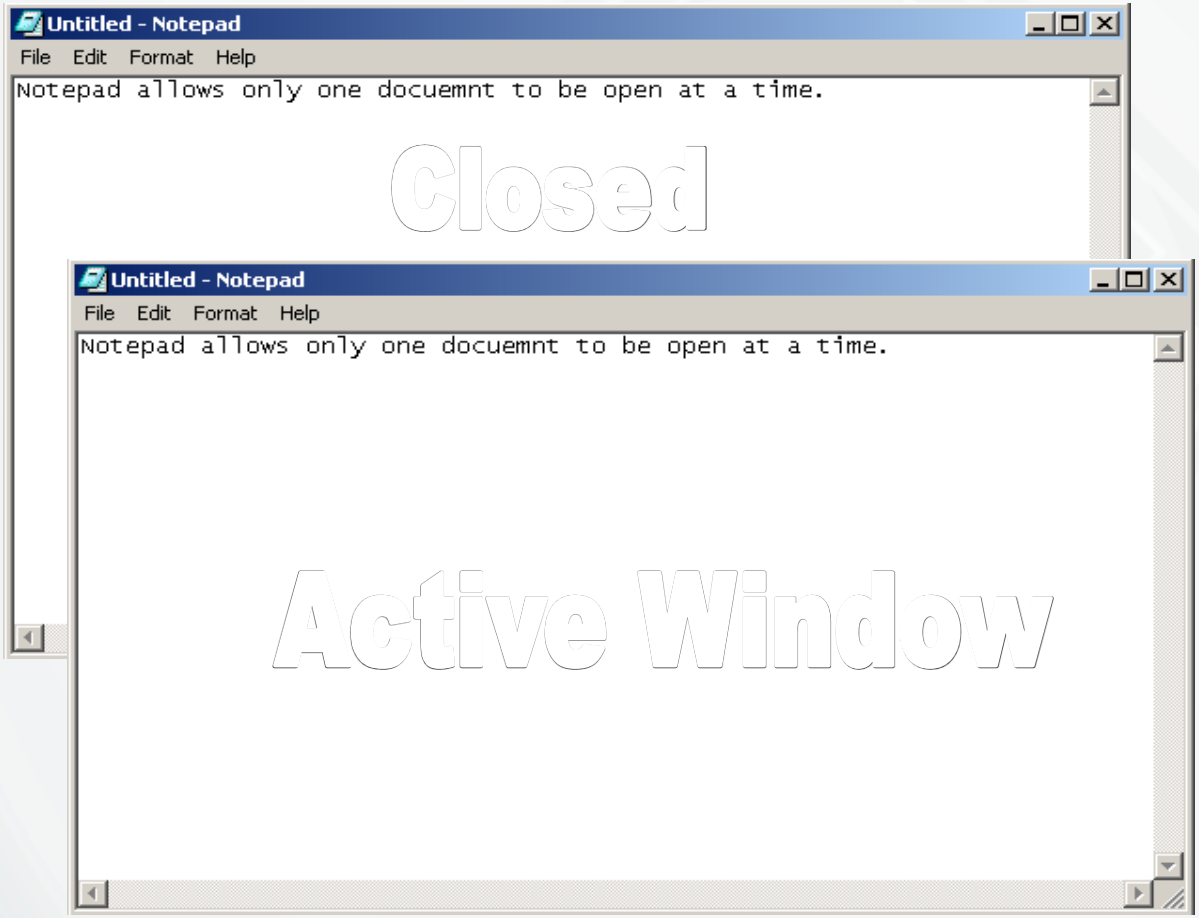
ErrorProvider Control



- Xử lý ngoại lệ khi người dùng nhập không phải là số.

```
private void btnTinh_Click(object sender, EventArgs e)
{
    try
    {
        int N = Convert.ToInt32(txtN.Text);
        long K=1;
        for(int i=1;i<=N;i++)
            K *= i;

        lblTG.Text=K.ToString();
    }
    catch(Exception ex)
    {
        errTB.SetError(txtN,ex.Message);
    }
}
```

SDI - Single Document Interface

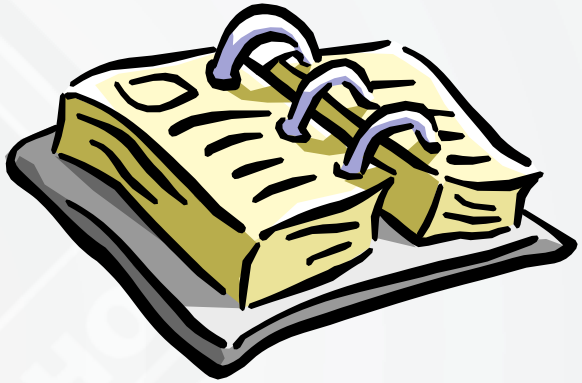


Ứng dụng SDI chỉ có một cửa sổ tại một thời điểm

SDI

- Không thể mở nhiều form tại thời điểm nào đó.
- Nếu muốn mở nhiều form thì ta cần tạo nhiều thể hiện của ứng dụng.

MDI - Multiple Document Interface

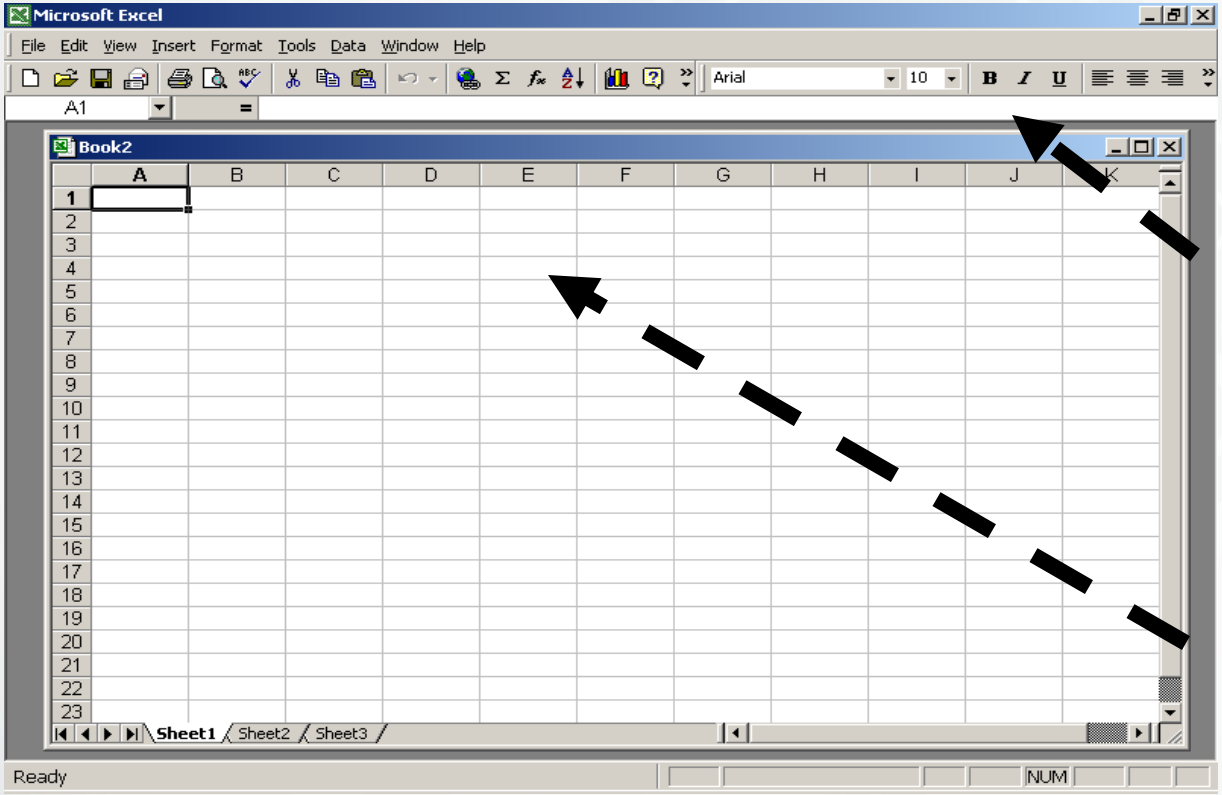


Nhiều tài liệu

Ứng dụng MDI nhóm nhiều tài liệu trong một đơn vị duy nhất.

- **Ứng dụng MDI có thể thao tác với nhiều cửa sổ**

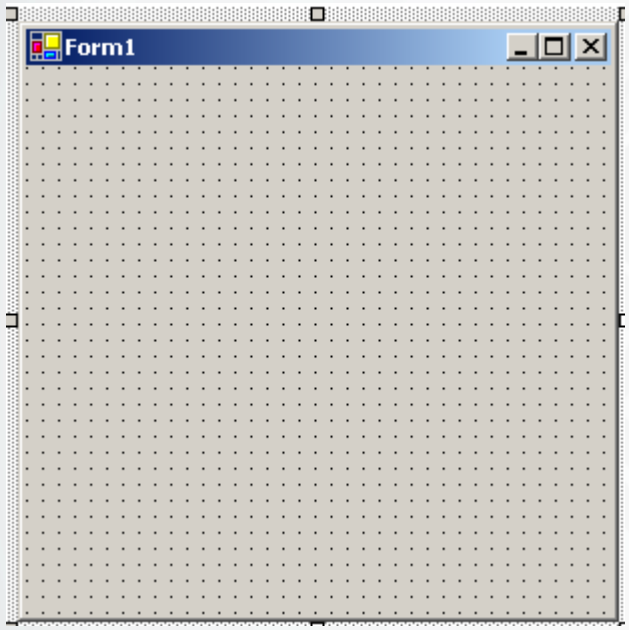
MDI



Cửa sổ cha

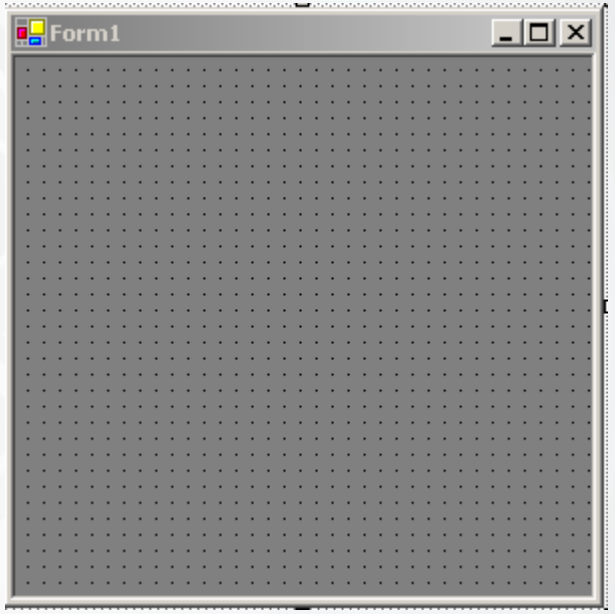
Cửa sổ con

Tạo MDI form



Property

IsMdiContainer = true



Form con của form MDI

- Sử dụng thuộc tính MdiParent:

```
<Đối tượng form con>.MdiParent =
                        <Đối tượng form cha>;
<Đối tượng form con>.Show();
```

- **Thí dụ**: Giả sử form hiện hành là 1 MDI form.

```
frmTravelEntry f = new frmTravelEntry();
f.MdiParent = this;
f.Show();
```

Đặc điểm của ứng dụng MDI

- Đặc điểm của form cha
  - Hiển thị khi ứng dụng khởi động
  - Là khung chứa cho các cửa sổ khác
  - Hiển thị menu của các form con
  - Chỉ có duy nhất một form cha
  - Có thể mở nhiều form con tại một thời điểm

Đặc điểm của ứng dụng MDI

- Form con không thể di chuyển khỏi vùng hiển thị của form cha.
- Đóng form cha thì tất cả các form con đều bị đóng lại

Sắp xếp cửa sổ con