

Chương 2 Các điều kiện cơ bản

Nội dung

Giới thiệu Form Label

TextBox Button

Các thuộc tính chung

Các sự kiện chung

Điều khiển sự kiện bàn phím

Điều khiển sự kiện chuột

Các điều khiển cơ bản khác

Giới thiệu

Một số điều kiện cơ bản

Form: Đối tượng cửa sổ của chương trình chứa các đối tượng khác.

Label: Đối tượng dùng để hiển thị văn bản và hình ảnh (người dùng không sửa được).

TextBox: Đối tượng dùng để hiển thị và nhập dữ liệu từ bàn phím.

Button: Là nút ấn cho phép Click nó để thực hiện một chức năng.

CheckBox: Đối tượng cho phép chọn hoặc không chọn.

ListBox: Đối tượng cho phép xem và chọn dữ liệu từ các dòng.

ComboBox: Đối tượng cho phép chọn dữ liệu từ các dòng.

GroupBox: Đối tượng chứa các đối tượng khác.

Panel: Đối tượng chứa các đối tượng khác.

Form

Dùng để tạo giao diện cho chương trình

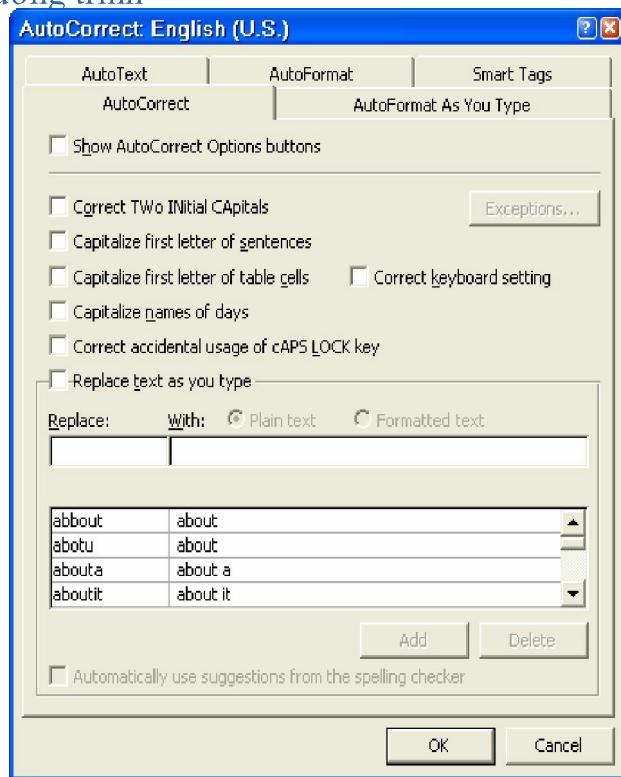
Thêm một Form mới

Chọn Project

->Add Windows Form

Chọn Windows Form

->gõ tên Form -> Add



Các điều khiển của Form

Là các thành phần đồ họa như Label, TextBox,...

Mỗi điều khiển tạo ra các đối tượng cùng lớp Các đối tượng có các thuộc tính, các sự kiện và các phương thức riêng.

Properties: Các thuộc tính mô tả đối tượng.

Methods: Các phương thức thực hiện các chức năng của đối tượng.

Events: Các sự kiện sinh ra bởi sự chuyển động của bàn phím và con chuột, chi tiết do người lập trình viết.

Các thuộc tính thường dùng

AcceptButton: Nút được click khi ấn phím *Enter*

CancelButton: Nút được click khi ấn phím *Esc*

BackgroundImage: Ảnh nền của Form

Font: Font hiển thị của Form và Font ngầm định của các đối tượng của Form.

FormBorderStyle: Kiểu đường viền của Form

None: Form không có đường viền

Fix...: Cố định kích thước khi chạy

Form Sizeable: Có thể thay đổi kích thước Form

ForeColor: Màu chữ của Form và màu chữ của các đối tượng của Form.

Text: Dòng văn bản hiển thị trên tiêu đề Form

MaximizeBox: Có/không nút phóng to

MinimizeBox: Có/không nút thu nhỏ

StartPosition: Vị trí bắt đầu khi chạy Form

CenterScreen: Nằm giữa màn hình

WindowState: Xác định trạng thái ban đầu Form

Close: Đóng Form và giải phóng các tài nguyên. Form đã đóng ko thể mở lại.

Hide: Ẩn Form và không giải phóng tài nguyên của Form.

Show: Hiển thị một Form đã ẩn.

Load: Xảy ra khi chạy Form (ngầm định khi nháy đúp chuột trong chế độ thiết kế).

FormClosing: Xảy ra khi đóng Form.

Ví dụ đặt các thuộc tính của Form

- ✓ Tạo một Form mới
- ✓ Gõ dòng tiêu đề của Form
- ✓ Đặt Form ở chế độ FixDialog
- ✓ Cắt nút phóng to
- ✓ Cắt nút thu nhỏ
- ✓ Cho Form nằm giữa màn hình
- ✓ Chèn một ảnh làm nền của Form
- ✓ Chạy Form

Label

Đối tượng hiển thị văn bản kết hợp hình ảnh, Ko sửa được văn bản hiển thị
Các thuộc tính thường dùng

- AutoSize:** Tự thay đổi kích thước của đối tượng
- Fonts:** Font chữ của đối tượng
- Label ForeColor:** Màu chữ của đối tượng
- Image:** Ảnh của đối tượng
- Text:** Văn bản xuất hiện trên đối tượng.
- TextAlign:** Lề của văn bản.

TextBox

Đối tượng dùng để nhập dữ liệu từ bàn phím

Các thuộc tính thường dùng

- Enabled:** Có/không cho phép thao tác đối tượng
- Multiline:** Có/không cho phép nhập dữ liệu nhiều dòng (mặc định là Ko)
- PasswordChar:** Nhập ký tự làm mật khẩu
- ReadOnly:** Có/không cho phép sửa dữ liệu của đối tượng (mặc định là có)
- Text:** Văn bản nhập (hiển thị) của đối tượng.

Các sự kiện thường dùng

- TextChanged:** Xảy ra khi nhập hoặc xoá các ký tự (ngầm định khi nháy đúp chuột trong chế độ thiết kế)
- KeyDown:** Xảy ra khi án một phím bất kỳ trên đối tượng.
- KeyUp:** Xảy ra khi thả một phím án trên đối tượng.
- Chú ý:** Dữ liệu nhập vào TextBox là văn bản do đó nếu thực hiện các phép toán số học, logic thì cần chuyển sang kiểu số.

Button

Đối tượng nút ấn cho phép thực hiện một chức năng

Có thể hiển thị hình ảnh kết hợp với văn bản

Các thuộc tính thường dùng

- Text:** Văn bản hiển thị trên đối tượng
- Image:** Hình ảnh hiển thị trên đối tượng

Các sự kiện thường dùng

- Click:** Xảy ra khi nhấn con trỏ chuột hoặc gõ Enter trên đối tượng (ngầm định khi nháy đúp chuột trong chế độ thiết kế).

Ví dụ

Giải phương trình bậc nhất

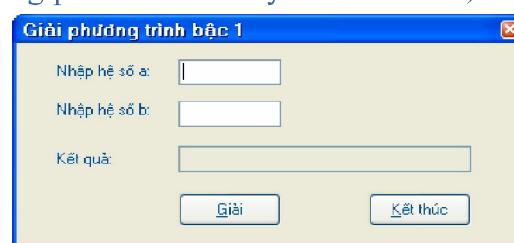
Nhập hệ số a, hệ số b Chọn nút <Giải>

Kiểm tra dữ liệu nhập? Nếu không phải là số? Chuyển thành số a, b ? Nếu
a = 0

Nếu b = 0 -> Vô số nghiệm

Nếu b != 0 -> Vô nghiệm

Nếu a != 0 -> x = -b/a



Chuyển dữ liệu từ xâu ký tự sang số

Chuyển không kiểm tra dữ liệu

Sử dụng hàm Convert: <num> = Convert.ToDouble(str)
double a = Convert.ToDouble("123.45");

Chuyển có kiểm tra dữ liệu

Sử dụng hàm TryParse của kiểu dữ liệu:
bool IsNumber = double.TryParse(str, out num)
IsNumber = true nếu đổi được
IsNumber = false nếu không đổi được

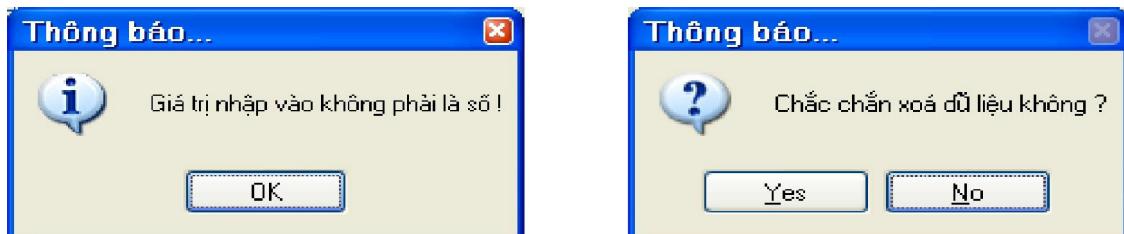
Chuyển dữ liệu từ số sang xâu

Sử dụng hàm Convert <str> = Convert.ToString(num)
string st = Convert.ToString(123);
Sử dụng hàm ToString() của đối tượng <obj>.ToString()
int a = 2; string s = a.ToString();

Hàm Message

Dùng để hiển thị một thông báo

Dùng để xác nhận một hành động



```
//hien thi thong bao - dang ham 7 co 4 tham so
MessageBox.Show("Giá trị nhập vào không phải là số !", "Thông
báo...", MessageBoxButtons.OK,
MessageBoxIcon.Information);
//Xac nhan hanh dong - dang ham 7 hoac dang ham 9
DialogResult myDialog;
myDialog = MessageBox.Show("Chắc chắn xoá dữ liệu không ?", "Thông
báo...", MessageBoxButtons.YesNo,
MessageBoxIcon.Question);
if (myDialog == DialogResult.No) return;
```

Các thuộc tính chung

Các thuộc tính thường dùng

BackColor: Màu nền của đối tượng.

BackgroundImage: Ảnh nền của đối tượng

Cursor: Kiểu con trỏ chuột khi đưa con trỏ chuột vào đối tượng

Enabled: Có/không cho phép thao tác với đối tượng

Font: Font chữ của đối tượng

ForeColor: Màu chữ của đối tượng

TabIndex: Thứ tự ấn phím Tab để chuyển con trỏ đến đối tượng, Chọn biểu tượng

Text: Dòng văn bản hiển thị trên đối tượng

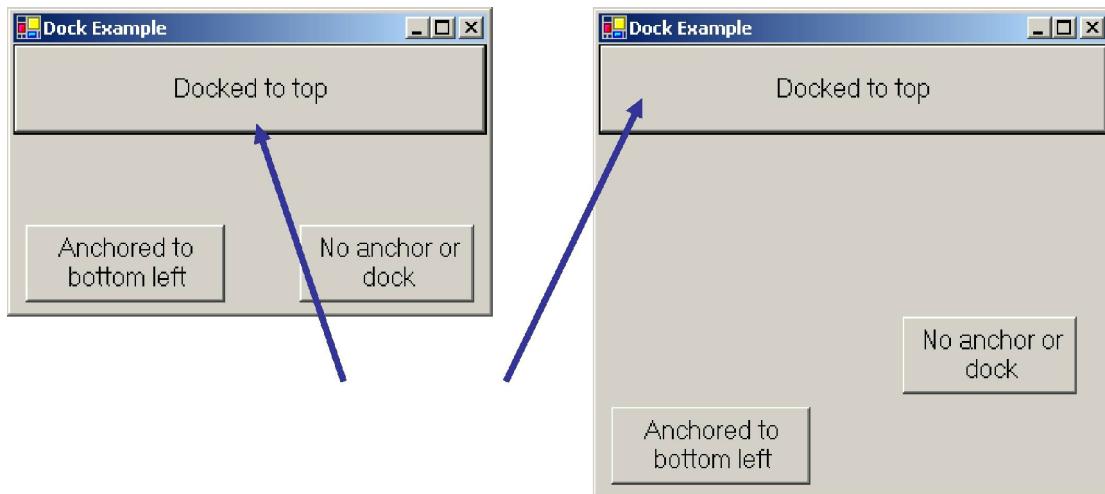
TextAlign: Lề của dòng văn bản hiển thị trên đối tượng

Visible: Ẩn/hiện đối tượng

Anchor: Neo đối tượng so với các cạnh của đối tượng chứa

Dock: Cố định đối tượng trong đối tượng chứa **Location:** Vị trí của đối tượng so với đối tượng chứa

Ví dụ về thuộc tính **Anchor** và **Dock**



Focus: Chuyển con trỏ đến đối tượng

Hide: Ẩn đối tượng

Show: Hiển thị đối tượng ẩn

Các sự kiện chung

Sự kiện của Form

KeyDown

KeyUp

KeyPress

MouseDown

MouseUp

Các sự kiện thường dùng

KeyDown: Xảy ra khi một phím được ấn trên đối tượng.

KeyUp: Xảy ra khi một phím được thả trên đối tượng

KeyEventArgs: Tham số cho sự kiện **KeyDown** và **KeyUp**.

KeyPress: Xảy ra khi ấn và thả một phím trên đối tượng.

KeyPressEventArgs: Tham số cho sự kiện **KeyPress**

KeyPressEventArgs: Là tham số của sự kiện KeyPress

KeyChar: Trả về ký tự của phím được ấn

Alt: Có/không phím Alt đã được ấn

Control: Có/không phím *Control* đã được ấn.

KeyEventArgs: Là tham số của các sự kiện KeyDown và KeyUp.

Shift: Có hay không phím *Shift* đã được ấn.

KeyCode: Trả về phím được ấn.

KeyValue: Trả về mã của phím được ấn

Ví dụ

Xây dựng Form cho phép gõ phím Enter hoặc các phím mũi tên để di chuyển con trỏ giữa các TextBox

```
private void textBox2_KeyDown(object sender, KeyEventArgs e)
{
    if ((e.KeyValue == 13) || (e.KeyValue == 40))
        textBox3.Focus();
    if (e.KeyValue == 38) textBox1.Focus();
}
```

Các sự kiện chuột

Các sự kiện thường dùng

MouseEnter: Xảy ra khi đưa con trỏ chuột vào vùng của đối tượng.

MouseLeave: Xảy ra khi đưa con trỏ chuột ra khỏi vùng của đối tượng.

MouseDown: Xảy ra khi ấn nút chuột trong khi con trỏ chuột đang nằm trong vùng của đối tượng.

MouseUp: Xảy ra khi thả nút chuột trong khi con trỏ chuột đang nằm trong vùng của đối tượng.

MouseMove: Xảy ra khi di chuyển con trỏ chuột trong vùng của đối tượng.

MouseEventArg: Là tham số của các sự kiện MouseUp, MouseDown, và MouseMove

Button: Nút chuột đã ấn (**left**, **right**, **middle** or **none**).

Clicks: Số lần nút chuột được ấn.

X: Toạ độ tương đối *x* của con trỏ chuột.

Y: Toạ độ tương đối *y* của con trỏ chuột.

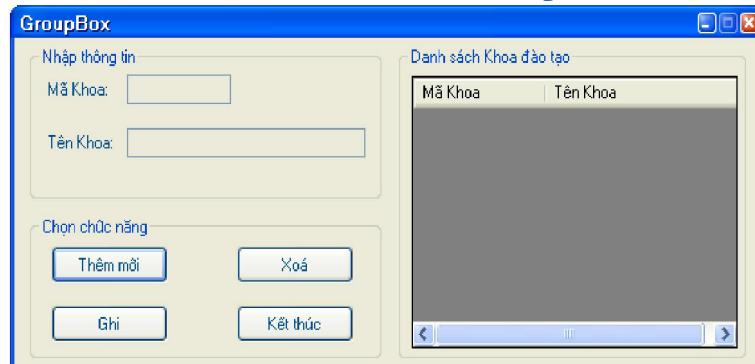
Bài tập: Viết chương trình hiển thị toạ độ

Viết CT nhập vào 3 cạnh của 1 tam giác, tính diện tích và chu vi.

GroupBox

Đối tượng dùng để chứa các đối tượng khác Mỗi đối tượng có tiêu đề
Thuộc tính thường dùng

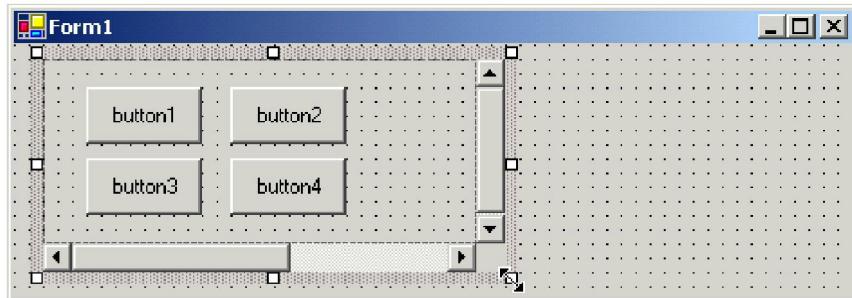
Text: Văn bản hiển thị trên tiêu đề **GroupBox**



Panel

Đối tượng dùng để chứa các đối tượng khác Mỗi đối tượng không tiêu đề
Thuộc tính thường dùng

BorderStyle: Đường viền của đối tượng (ngầm định là **None**)



CheckBox

Đối tượng cho phép chọn/không chọn giá trị, cho phép chọn đồng thời nhiều
đối tượng

Các thuộc tính thường dùng

Checked: Có/không đối tượng được chọn

Text: Văn bản hiển thị trên đối tượng

Các sự kiện thường dùng

CheckedChanged: Xảy ra khi chọn/không chọn đối tượng (ngầm định khi nháy
đúp chuột trong chế độ thiết kế)

RadioButton

Cho phép chọn/ko chọn giá trị, cho
phép chọn 1 đối tượng ở một thời điểm.

Để chọn nhiều đối tượng phải đặt các
điều khiển trong GroupBox hoặc Panel

Các thuộc tính thường dùng

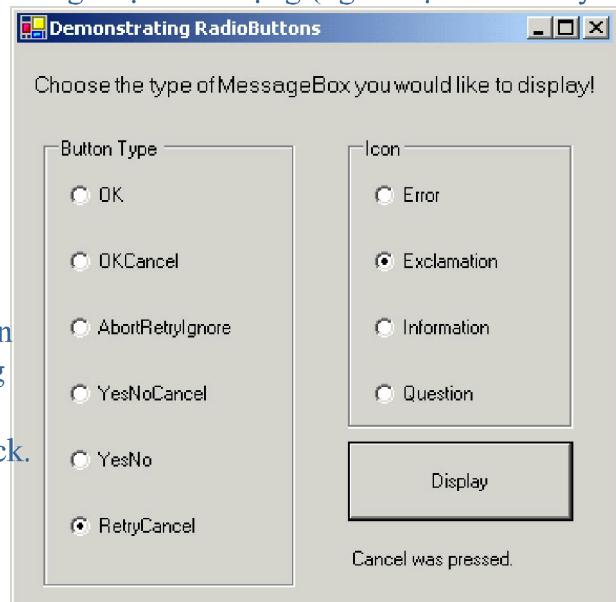
Checked: Có/ko đối tượng được chọn

Text: Văn bản hiển thị trên đối tượng

Các sự kiện thường dùng

Click: Xảy ra khi đối tượng được click.

CheckedChanged: Xảy ra khi chọn/
không chọn đối tượng (ngầm định khi nháy
đúp chuột trong chế độ thiết kế).



PictureBox

Đối tượng dùng để hiển thị hình ảnh (Bitmap, GIF, JPEG, Metafile, Icon)

Các thuộc tính thường dùng

Image: Ảnh hiển thị trong PictureBox.

SizeMode: Chế độ hiển thị ảnh

Normal:Đặt ảnh ở góc trên bên trái của đối tượng

CenterImage: Đặt ảnh ở giữa đối tượng

StretchImage: Thay đổi kích thước ảnh đúng với kích thước đối tượng in **PictureBox**.

AutoSize: Thay đổi kích thước đối tượng **PictureBox** đúng với kích thước ảnh.

Chương 3 Các điều kiện nâng cao

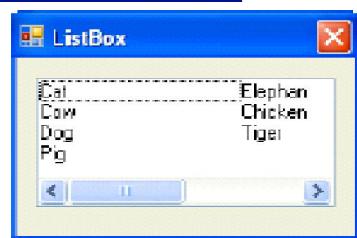
Nội dung

- ListBox
- CheckedListBox
- ComboBox
- TabControl
- Menu
- Toolbar
- MDI Windows
- TreeView
- ListView



ListBox

Cho phép xem và chọn các dòng dữ liệu



Các thuộc tính thường dùng

Items: Mảng các dòng trong ListBox.

Items[0] = "Cat"

Items[1] = "Mouse"

MultiColumn: Có/không chia ListBox thành nhiều cột.

SelectedIndex: Trả về dòng hiện thời được chọn

Nếu chọn nhiều dòng thì trả về 1 giá trị tùy ý của các dòng được chọn.

Nếu không chọn thì trả về giá trị -1.

SelectedIndices: Trả về một mảng các chỉ số của các dòng được chọn.

SelectedItem: Trả về giá trị dòng được chọn.

SelectedItems: Trả về một mảng giá trị các dòng được chọn.

Sorted: Có/Không sắp xếp dữ liệu trong ListBox. Ngầm định là **False**.

SelectionMode: Xác định số lượng dòng được chọn của ListBox.

one: Một dòng

Multi: Nhiều dòng

GetSelected(index): Trả về True nếu dòng *Index* được chọn, ngược lại trả về false.

Add: Thêm một dòng vào ListBox

```
listBox1.Items.Add("Cat");  
listBox1.Items.Add("Mouse");
```

RemoveAt(row): Xoá dòng ở vị trí row

```
listBox1.Items.RemoveAt(row);
```

Clear: Xoá tất cả các dòng

```
listBox1.Items.Clear();
```

Sự kiện thường dùng

SelectedIndexChanged: Xảy ra khi chọn một dòng. Ngầm định khi nháy đúp ở ché độ thiết kế.

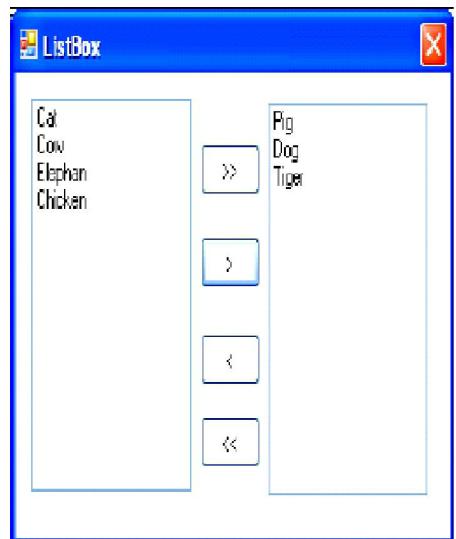
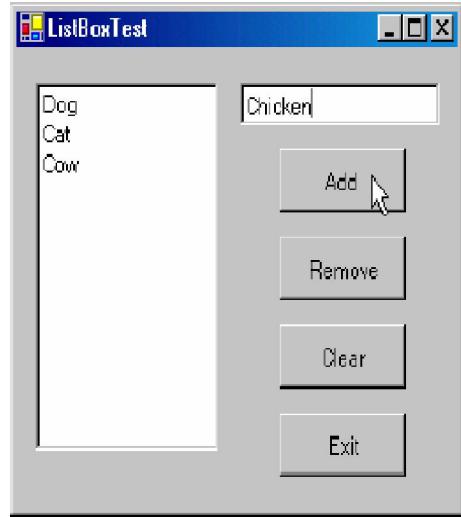
Ví dụ

Xây dựng Form

```
private void btnAdd_Click(object sender, EventArgs e)
{
    listBox1.Items.Add(txtInput.Text);
    txtInput.Clear();
}
private void btnRemove_Click(object sender, EventArgs e)
{
    int row=listBox1.SelectedIndex;
    if (row != -1)
        listBox1.Items.RemoveAt(row);
}
private void btnClear_Click(object sender, EventArgs e)
{
    listBox1.Items.Clear();
}
```

Bài tập

Xây dựng Form cho phép di chuyển các dòng giữa 2 ListBox



CheckedListBox

Là sự mở rộng của ListBox bằng cách thêm CheckBox ở phía bên trái mỗi dòng
-> Có thể chọn các dòng

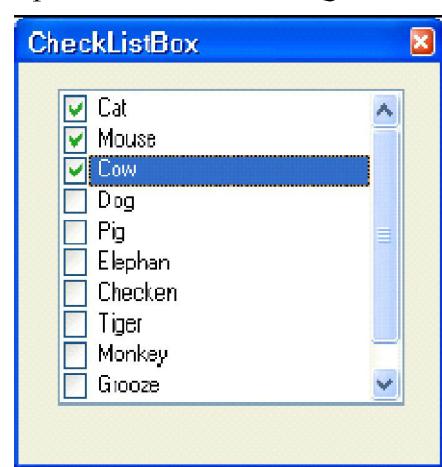
Các thuộc tính thường dùng

CheckedItems: Mảng các giá trị của dòng được đánh dấu Check.

CheckedIndices: Mảng các chỉ số dòng được đánh dấu Check.

Phương thức thường dùng

GetItemChecked(index): Trả về true nếu dòng được chọn.



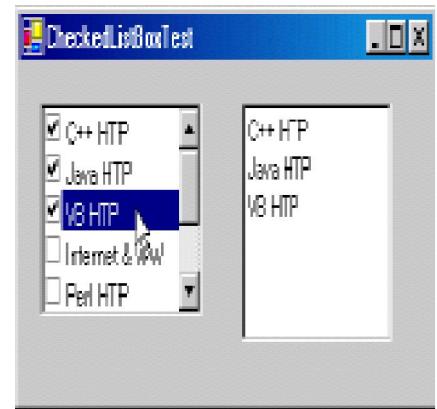
Sự kiện thường dùng

ItemCheck: Xảy ra khi dòng được checked hoặc unchecked.

Ví dụ

Xây dựng Form khi đánh dấu check thì dòng được đưa sang ListBox bên phải, khi bỏ dấu Check thì xoá dòng trong ListBox bên phải

```
private void myCheckedListBox_Load(object sender, EventArgs e)
{
    checkedListBox1.Items.Add("C++ HTP");
}
private void checkedListBox1_ItemCheck(object sender, ItemCheckEventArgs e)
{
    string item = checkedListBox1.SelectedItem.ToString();
    if (e.NewValue == CheckState.Checked)
    {
        listBox1.Items.Add(item);
    }
    else
    {
        listBox1.Items.Remove(item);
    }
}
```

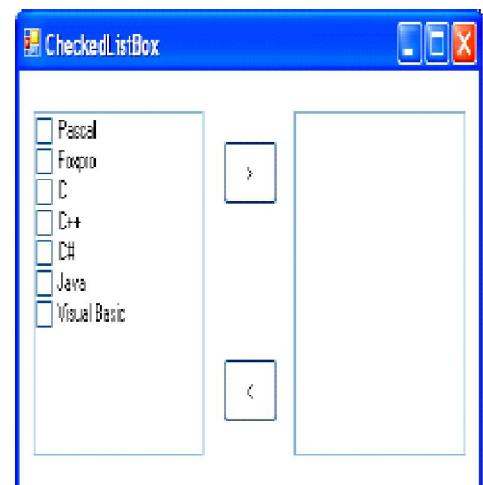


Bài tập

Xây dựng form cho phép đánh dấu và chuyển các dòng giữa 2 CheckedListBox

Sự kiện cho nút Add của CheckedListBox

```
private void btnAdd_Click(object sender, EventArgs e)
{
    for (int i = 0; i < checkedListBox1.Items.Count - 1; i++)
    {
        if (checkedListBox1.GetItemChecked(i) == true)
        {
            // Adds Item i to checkedListBox2
            string item = checkedListBox1.Items[i].ToString();
            checkedListBox2.Items.Add(item);
            // Removes Item i at checkedListBox1
            checkedListBox1.Items.RemoveAt(i);
        }
    }
}
```



ComboBox

Là sự kết hợp của **TextBox** và **ListBox**

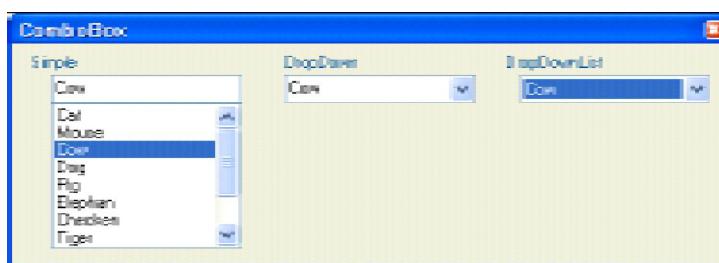
Các thuộc tính thường dùng

DropDownStyle: Xác định kiểu của ComboBox.

Simple: Chọn hoặc gõ giá trị

DropDown(ngầm định): Chọn hoặc gõ giá trị

DropDownList: Chỉ cho phép chọn giá trị.



Items: Mảng các dòng trong ComboBox

SelectedIndex: Chỉ số dòng được chọn. Nếu không chọn có giá trị -1.

SelectedItem: Giá trị dòng được chọn.

Sorted: Có/Không sắp xếp dữ liệu trong ComboBox. Ngầm định là `false`.

Sự kiện thường dùng

SelectedIndexChanged: Xảy ra khi chọn 1 dòng.

Các phương thức thường dùng

Add: Thêm một dòng vào ComboBox

`comboBox1.Items.Add("Cat");`

`comboBox1.Items.Add("Mouse");`

RemoveAt(row): Xoá dòng ở vị trí row

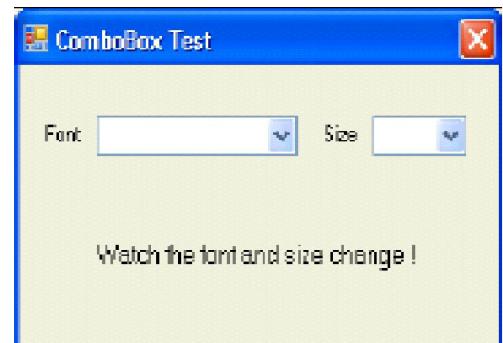
`comboBox1.Items.RemoveAt(row);`

Clear: Xoá tất cả các dòng trong ComboBox

`comboBox1.Items.Clear();`

Ví dụ

Xây dựng Form để lấy Font của hệ thống



```
private void comboBox_Load(object sender, EventArgs e)
{
    FontFamily[] ff = FontFamily.Families;
    for(int i = 0; i < ff.Length; i++)
        cboFont.Items.Add(ff[i].Name);
    for(int i = 8; i <= 72; i++)
        cboSize.Items.Add(i);
}
```

```

        cboSize.Items.Add(i);
    }
    private void cboFont_SelectedIndexChanged(object sender, EventArgs e)
    {
        if(cboFont.Text.Trim() == "") return;
        lbDisplay.Font = new Font(cboFont.Text, lbDisplay.Font.Size);
    }
    private void cboSize_SelectedIndexChanged(object sender, EventArgs e)
    {
        if(cboSize.Text.Trim() == "") return;
        lbDisplay.Font
        =newFont(lbDisplay.Font.Name,(float)Convert.ToDouble(cboSize.Text));
    }

```

Menu

Dùng để nhóm các lệnh cùng nhau

Menu có thể chứa

Menu ngang

Menu dọc

Menu con

Các biểu tượng

Các phím nóng

Các đường phân cách

....

Xây dựng menu

Kéo biểu tượng ToolStrip vào Form

Gõ các dòng cho menu

Đặt tên cho các dòng của menu

Chọn cửa sổ Properties và đặt
thuộc tính Name

Tên menu đặt bằng tiền tố mnu (ví dụ: mnuFile, mnuEdit)

Chèn hình ảnh cho các dòng của menu

Nháy chuột phải và chọn **Set Image**

Chọn **Local Resource -> Import** ->chọn hình ảnh

Đặt phím nóng cho các dòng của menu

Chọn cửa sổ Properties và đặt thuộc tính **ShortCutKey**

Các thuộc tính thường dùng

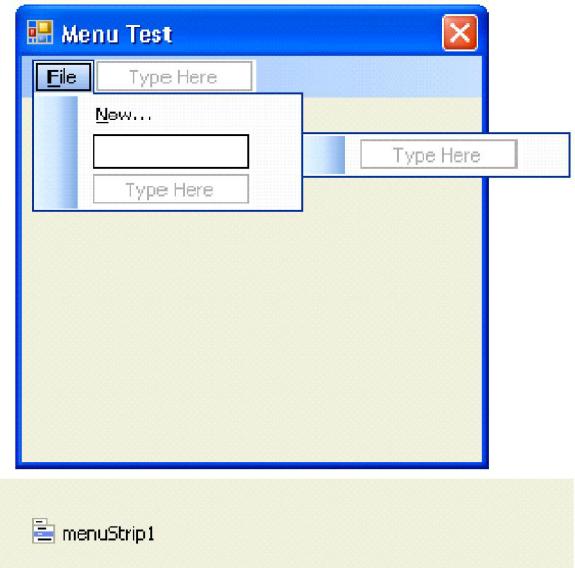
Name: Tên menu được dùng trong mã lệnh.

Checked: Có/Không dòng menu xuất hiện checked. Ngầm định là **false**.

ShortCutKey: Đặt phím nóng cho menu

ShowShortcut: Có/Không phím nóng hiển thị trên dòng menu. Ngầm định là **true**.

Text: Xuất hiện trên dòng menu.



Sự kiện thường dùng

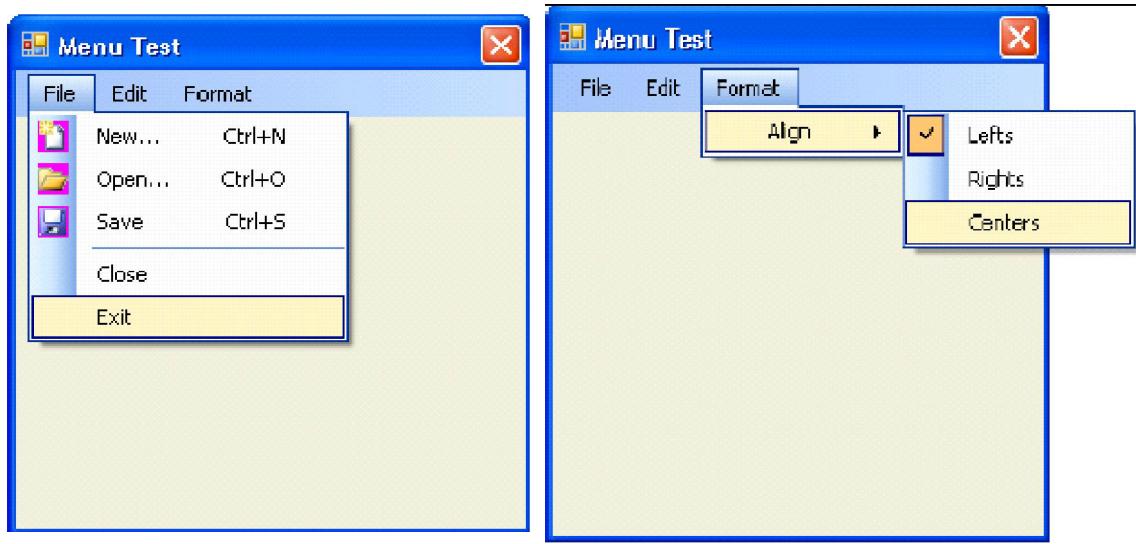
Click: Xảy ra khi một dòng của menu được click chuột hoặc ấn phím nóng. Ngầm định khi nháy đúp chuột trong chế độ thiết kế.

Viết lệnh cho dòng của menu gọi một Form

```
<Tên lớp> <Tên form> = new <Tên lớp>;
<Tên Form>.Show();
<Ten Form>.ShowDialog();
```

Ví dụ:

Thiết kế menu



Menu ngữ cảnh

Menu hiển thị khi nháy chuột phải trên đối tượng của Form hoặc trên Form.

Thiết kế menu ngữ cảnh

Kéo điều khiển **ContextMenu** vào form

Thiết kế các dòng menu ngữ cảnh giống như thiết kế menu

Hiển thị menu ngữ cảnh

Chọn đối tượng hoặc Form

Đặt thuộc tính **ContextMenu** của đối tượng được chọn là tên của menu ngữ cảnh.

ToolBar

ToolBar dùng để chứa các biểu tượng của các chức năng thường được sử dụng trong chương trình.

Xây dựng ToolBar

Kéo điều khiển **ToolStrip** vào Form

Nháy chuột vào biểu tượng phải và chọn đối tượng tạo ToolBar

Button: Nút ấn

DropDownButton: Nút sổ xuống
Separator: Đường phân cách

Xây dựng ToolBar

Đặt tên cho các nút của ToolBar
Chọn cửa sổ **Properties** và đặt thuộc tính **Name**

Tên **ToolBar** đặt bằng tiền tố tb (ví dụ: tbNew, tbOpen)

Chèn hình ảnh cho đối tượng của ToolBar

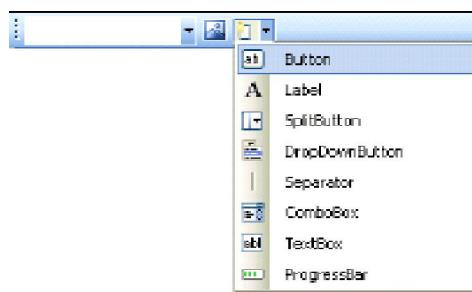
Nháy chuột phải và chọn **Set Image**

Chọn **Local Resource -> Import ->** chọn hình ảnh

Viết mã lệnh

Gọi từ menu: <tên menu>.PerformClick()

Gọi trực tiếp đối tượng



MDI Windows

Một ứng dụng MDI cho phép người dùng thao tác với nhiều cửa sổ ở một thời điểm.
SDI và MDI Forms

Ví dụ MDI Parent và MDI Child

Thuộc tính MDI Parent của Form

IsMdiContainer: Có/Không một Form là form MDI Parent. Ngầm định là False.

ActiveMdiChild: Trả về Form Child đang được kích hoạt.

Thuộc tính MDI Child

IsMdiChild: Có/Không Form là một MDI child (thuộc tính read-only).

MdiParent: Chỉ ra một MDI parent của Form

<Tên form>.MdiParent = this

Phương thức thường dùng

LayoutMdi: Xác định kiểu hiển thị của Form con trong MDI Form.

ArrangeIcons: Sắp xếp các biểu tượng dưới MDI

Cascade: Sắp xếp các cửa sổ chồng nhau

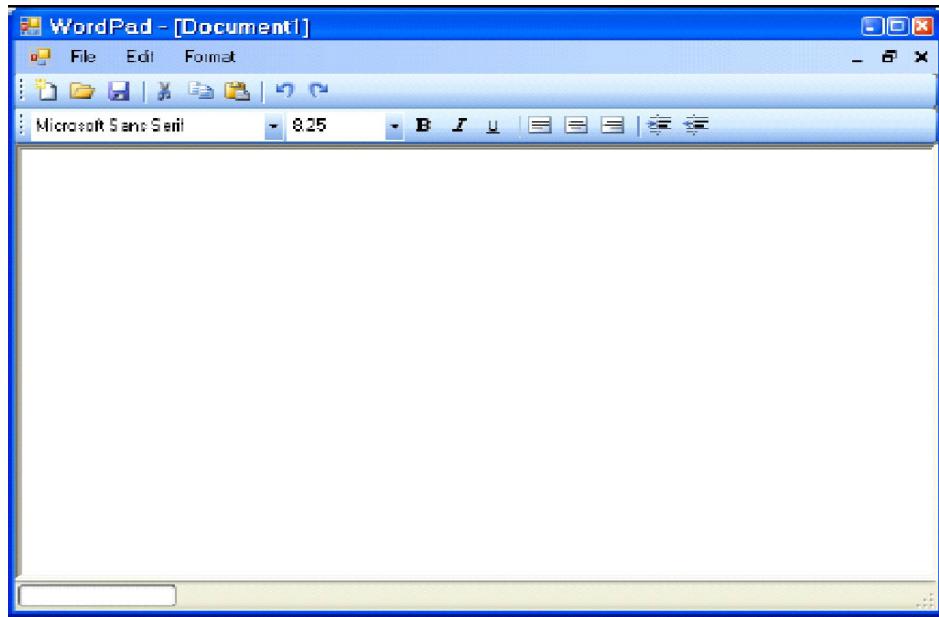
TileHorizontal: Sắp xếp cửa sổ theo chiều ngang

TileVertical: Sắp xếp cửa sổ theo chiều dọc

Các kiểu sắp xếp

Bài tập

Xây dựng một hệ soạn thảo văn bản theo dạng MDI Form.



TreeView

Hiển thị thông tin theo các nút
(node)

Các nút cha có các nút con

Nút đầu tiên gọi là nút gốc

Ấn dấu [+] để mở nút

Ấn dấu [-] để thu gọn nút

Mỗi nút có hình ảnh kèm theo

Các thuộc tính thường dùng

Checkboxes: Có/không xuất hiện các checkbox trên các node. Mặc định là **False**.

Checked: Có/không một **Node** được check (thuộc tính **Checkboxes** phải được đặt là **True**)

ImageList: Chỉ ra danh sách ảnh hiển thị trên các node.

ImageList là một mảng các đối tượng ảnh.

Tạo danh sách ảnh **ImageList** bằng cách kéo điều kiển vào Form, nháy chuột phải và chọn Choose Image để thêm các ảnh vào **ImageList**.

Nodes: Mảng các **TreeNodes** trong TreeView.

Nodes.Add: Bổ sung một node vào cây.

Nodes.Clear: Xoá toàn bộ các node trên cây.

Nodes.Remove: Xoá một node trên cây và các node con của nó.

SelectedNode: Node hiện thời được chọn

FullPath: Chỉ ra đường dẫn đến node bắt đầu từ node gốc.

SelectedImageIndex: Chỉ ra chỉ số ảnh được hiển thị trên node khi node được chọn.

ImageIndex: Chỉ ra chỉ số ảnh được hiển thị trên node khi node không được chọn (deselected).

Text: Text hiển thị của Node.

FirstNode: Node con đầu tiên của node.

LastNode: Node con cuối cùng của node.

PrevNode: Node con trước node con hiện thời.

NextNode: Node con tiếp theo node hiện thời.

Các phương thức thường dùng

Collapse: Thu nhỏ các node con của node.

Expand: Mở rộng các node con của node.

ExpandAll: Mở rộng tất cả các node con.

GetNodeCount: Trả về số lượng node con.

Các sự kiện thường dùng

AfterSelect: Xảy ra khi một node được chọn (ngầm định khi nháy đúp chuột ở chế độ thiết kế).

BeforeExpand: Xảy ra khi mở rộng một node

Ví dụ

Đặt tên các đối tượng

treeView1

txtInput

comboBox1

btnAddRoot

btnAddChild

btnDelete

Khai báo trong lớp:

private TreeNode currentNode;

Thiết lập thuộc tính *ImageCollection* cho đối tượng *imageList1*

Các sự kiện:

```
private void AddTreeView_Load(object sender, EventArgs e)
```

```
{
```

```
    treeView1.ImageList = imageList1;
```

```
    comboBox1.Items.Add("Image1");
```

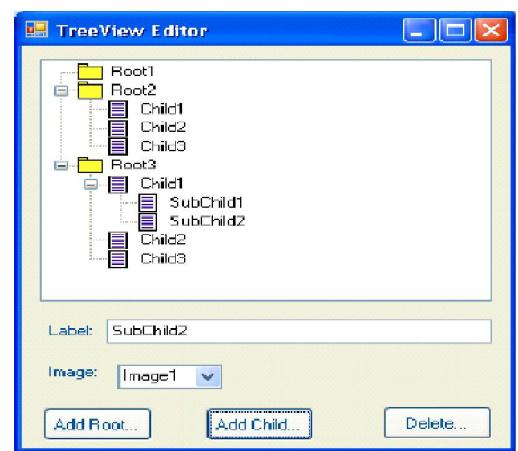
```
    comboBox1.Items.Add("Image2");
```

```
    comboBox1.SelectedIndex = 0;
```

```
}
```

```
private void btnAddRoot_Click(object sender, EventArgs e)
```

```
{
```



```

        if (txtInput.Text.Trim() == "") return;
        TreeNode childNode = new TreeNode();
        childNode.Text = txtInput.Text;
        childNode.ImageIndex = comboBox1.SelectedIndex;
        treeView1.Nodes.Add(childNode);
    }
    private void treeView1_AfterSelect(object sender, TreeViewEventArgs e)
    {
        currentNode = e.Node;
    }
    private void btnAddChild_Click(object sender, EventArgs e)
    {
        if (txtInput.Text.Trim() == "") return;
        TreeNode childNode = new TreeNode();
        childNode.Text = txtInput.Text;
        childNode.ImageIndex = comboBox1.SelectedIndex;
        currentNode.Nodes.Add(childNode);
        currentNode.ExpandAll();
    }
    private void btnDelete_Click(object sender, EventArgs e)
    {
        currentNode.Remove();
    }
}

```

ListView

Dùng để hiển thị dữ liệu theo các dòng và các cột

Có thể chọn một hoặc nhiều dòng

Có thể hiển thị các biểu tượng theo các dòng

Ví dụ ListView hiển thị danh sách thư mục TP và các tệp

Các thuộc tính thường dùng

Checkboxes: Có/ko các checkbox trên các dòng dữ liệu (ngầm định là **False**)

Columns: Các cột hiển thị trong chế độ **Details**.

FullRowSelect: chỉ định rằng mọi SubItems có được highlighted cùng với Item khi

được chọn hay không.

GridLines: Hiển thị lưới (chỉ hiển thị trong chế độ Details).

Items: Mảng các dòng (**ListViewItems**) trong ListView.

LargeImageList: Danh sách ảnh (**ImageList**) hiển thị trên ListView.

SmallImageList: Danh sách ảnh (**ImageList**) hiển thị trên ListView.

MultiSelect: Có/Không cho phép chọn nhiều dòng (ngầm định là **True**).

SelectedItems: Mảng các dòng được chọn.

View: Kiểu hiện thị của ListView

Icons: Hiển thị danh sách theo các biểu tượng

List: Hiển thị danh sách theo một cột

Details: Hiển thị ListView theo danh sách nhiều cột

Các phương thức thường dùng

Add: Thêm một dòng vào ListView

Clear: Xóa tất cả các dòng của ListView

Remove: Xóa một dòng trong ListView

RemoveAt(index): Xóa một dòng ở vị trí index

Sự kiện thường dùng

ItemSelectionChanged: Xảy ra khi chọn một dòng.

Ví dụ

Thiết kế form nhập, sửa, xoá dữ liệu

Đặt tên các đối tượng

txtId, txtFirstName, txtLastName, txtAddress, btnNew, btnEdit, btnDelete, btnSave, btnCancel.

Đặt thuộc tính cho listView1

Columns: Thêm 4 cột

FullRowSelect: true

GridLines: true

MultiSelect: false

View: Details

Viết Code cho các sự kiện

```
private bool modeNew;  
private int row; //Ham dat dieu khien trang thai cac textbox va cac nut  
private void SetControls(bool edit)  
{  
    txtId.Enabled = false;  
    txtFirstName.Enabled = edit;  
    txtLastName.Enabled = edit;  
    txtAddress.Enabled = edit;  
    btnNew.Enabled = !edit;  
    btnEdit.Enabled = !edit;  
    btnDelete.Enabled = !edit;  
    btnSave.Enabled = edit;  
    btnCancel.Enabled = edit;  
}  
private void frmListView_Load(object sender, EventArgs e)  
{
```

```

        SetControls(false);
    }
    private void btnNew_Click(object sender, EventArgs e)
    {
        modeNew = true;
        SetControls(true);
        row = listView1.Items.Count;
        txtId.Text = Convert.ToString(row + 1);
        txtFirstName.Clear();
        txtLastName.Clear();
        txtAddress.Clear();
        txtFirstName.Focus();
    }
    private void btnEdit_Click(object sender, EventArgs e)
    {
        modeNew = false;
        SetControls(true);
        txtFirstName.Focus();
    }
    private void btnSave_Click(object sender, EventArgs e)
    {
        if(modeNew)
        {
            listView1.Items.Add(txtId.Text);
            listView1.Items[row].SubItems.Add(txtFirstName.Text);
            listView1.Items[row].SubItems.Add(txtLastName.Text);
            listView1.Items[row].SubItems.Add(txtAddress.Text);
        }
        else
        {
            listView1.Items[row].SubItems[1].Text = txtFirstName.Text;
            listView1.Items[row].SubItems[2].Text = txtLastName.Text;
            listView1.Items[row].SubItems[3].Text = txtAddress.Text;
        }
        SetControls(false);
    }
    private void btnCancel_Click(object sender, EventArgs e)
    {
        SetControls(false);
    }
    private void btnDelete_Click(object sender, EventArgs e)
    {
        try{

```

```

        listView1.Items.RemoveAt(row);
    }catch(Exception){}
}
private void listView1_SelectedIndexChanged(
object sender,ListViewItemSelectionChangedEventArgs)
{
    row = e.ItemIndex;
    txtId.Text = listView1.Items[row].SubItems[0].Text;
    txtFirstName.Text = listView1.Items[row].SubItems[1].Text;
    txtLastName.Text = listView1.Items[row].SubItems[2].Text;
    txtAddress.Text = listView1.Items[row].SubItems[3].Text;
}
private void btnClose_Click(object sender, EventArgs)
{
    this.Close();
}

```

Bài tập tổng hợp

Xây dựng
 chương trình
 Explore hiển
 thị thông tin về
 tệp

Mã nguồn gợi ý:

```

private void frmExplore_Load(object sender, EventArgs)
{//Lay o dia cua may
    DriveInfo[] drives = DriveInfo.GetDrives();
    for(int i = 0; i < drives.Length; i++)
    {
        TreeNodemymyNode = newTreeNode();
        myNode.Text = drives[i].Name;
        myNode.ImageIndex = 0;
    }
}

```

```

        myNode.SelectedImageIndex = 0;
        treeView1.Nodes.Add(myNode);
        LoadTreeView(myNode.Text, myNode);
    }
}
public void LoadTreeView(string dirValue, TreeNode parentNode)
{
    string[] dirArray = Directory.GetDirectories(dirValue);
    if(dirArray.Length != 0)
    {
        foreach(string directory in dirArray)
        {
            DirectoryInfo curDirectory = new DirectoryInfo(directory);
            TreeNode myNode = new TreeNode(curDirectory.Name);
            parentNode.Nodes.Add(myNode);
            // recursively populate every subdirectory
            LoadTreeView(directory, myNode);
        }
    }
}
public void LoadFilesInDirectory(string curDirectory)
{
    listView1.Items.Clear();
    DirectoryInfo newDirectory = new DirectoryInfo(curDirectory);
    // put files and directories into arrays
    DirectoryInfo[] dirArray = newDirectory.GetDirectories();
    FileInfo[] fileArray = newDirectory.GetFiles();
    foreach(DirectoryInfo dir in dirArray)
    {
        ListViewItem newItem = listView1.Items.Add(dir.Name);
        newItem.SubItems.Add("");
        newItem.SubItems.Add("Folder");
        newItem.SubItems.Add(dir.LastWriteTime.ToString());
        newItem.ImageIndex = 0;
    }
    foreach(FileInfo file in fileArray)
    {
        ListViewItem newItem = listView1.Items.Add(file.Name);
        newItem.SubItems.Add(file.Length.ToString());
        newItem.SubItems.Add("File");
        newItem.SubItems.Add(file.LastWriteTime.ToString());
        newItem.ImageIndex = 1;
    }
}

```

```
        }
        private void treeView1_AfterSelect(object sender, TreeViewEventArgs e)
        {
            string currentDirectory = e.Node.FullPath;
            LoadFilesInDirectory(currentDirectory);
        }
        private void mnuList_Click(object sender, EventArgs e)
        {
            listView1.View = View.List;
        }
        private void mnuDetails_Click(object sender, EventArgs e)
        {
            listView1.View = View.Details;
        }
        private void tbUp_Click(object sender, EventArgs e)
        {
            string currentDirectory; currentDirectory = treeView1.SelectedNode.Parent.FullPath;
            LoadFilesInDirectory(currentDirectory);
        }
```

TabControl

Tạo ra các cửa sổ Tab

Mỗi cửa sổ Tab gọi là một **TabPage**

✓ **TabPage**s có thể chứa các điều khiển

Thêm và xoá các TabPage

Bài Tập

Chương 4. Xử lý lỗi

Nội dung

- Đặt vấn đề
- Xử lý lỗi
- Lệnh try ... catch
- Ví dụ

Giới thiệu

Một lỗi ngoại lệ (exception) là lỗi không mong đợi xảy ra khi chương trình thực hiện.
Lỗi ngoại lệ xảy ra do

- Các lỗi do lập trình không tốt
- Các lệnh gọi thư viện
- Tài nguyên không đủ khi thực hiện

.NET Framework xây dựng lớp **Exception** cho phép sử dụng để xử lý các lỗi ngoại lệ

Lớp Exception

Lớp Exception là lớp cơ sở cho phép các lớp khác có thể kế thừa

- OleDbException
- SqlException

Một số thuộc tính

- Message:** Lý do xảy ra lỗi
- StackTrace:** Nơi xảy ra lỗi
- Vị trí dòng lệnh
- Thủ tục

Lệnh try ... catch

Dùng để xử lý lỗi ngoại lệ

```
try{  
    1. Các lệnh có thể xảy ra lỗi  
}  
catch (Exception){  
    2. Hiển thị lỗi  
}  
finally{  
    3. (Tùy chọn) mã lệnh luôn được thực hiện.  
}
```

Ví dụ:

Ví dụ về phép chia

Mã code:

```
private void btnDivide_Click(object sender, EventArgs e)
{
    textBox3.Clear();
    try{
        int a = Convert.ToInt32(textBox1.Text);
        int b = Convert.ToInt32(textBox2.Text);
        int c = a / b;
        textBox3.Text = c.ToString();
    }
    catch(Exception ex){
        MessageBox.Show(ex.Message);
    }
}
```

Chương 5. ADO.NET

Nội dung

- Các khái niệm
- Các đối tượng của ADO.NET
- Xây dựng lớp truy nhập dữ liệu

I. Các khái niệm

ADO.NET là công nghệ truy nhập dữ liệu có cấu trúc

Cung cấp giao diện hướng đối tượng hợp nhất (Uniform object oriented) cho các dữ liệu khác nhau

- Cơ sở dữ liệu quan hệ
- XML
- Các dữ liệu khác

Được thiết kế cho các ứng dụng phân tán và Web

ADO.NET= ActiveX Data Objects

Các đối tượng ADO.NET chứa trong không gian tên **System.Data**.

Các đối tượng ADO.NET chia thành 2 loại

- **Connected:** Các đối tượng truyền thông trực tiếp với cơ sở dữ liệu.
- **Disconnected:** Các đối tượng không truyền thông trực tiếp với cơ sở dữ liệu.

Các đối tượng

DataSet: Một tập DataTable trong bộ nhớ

DataTable: Một bảng dữ liệu trong bộ nhớ

DataRow: Một bản ghi trong DataTable

DataColumn: Một cột dữ liệu trong DataTable

DataRelation: Đặt quan hệ của 2 DataTable

DataViewManager: Tạo Views của DataSet

DataTable

Có thể ánh xạ một bảng vật lý với DataTable

Một DataTable là một mảng 2 chiều gồm các dòng và các cột

Một số thuộc tính

- **Columns:** Các cột dữ liệu của DataTable
 - + Count: Số cột trong DataTable
- **Rows:** Các dòng dữ liệu của DataTable
 - + Count: Số dòng trong DataTable

Có thể tạo một DataTable trong bộ nhớ

```
DataTable myTable = new DataTable();
myTable.Columns.Add("MaKhoa", typeof(string));
myTable.Columns.Add("TenKhoa", typeof(string));
```

ADO.NET tổ chức thành mô hình đối tượng

```
System.Data
System.Data.OleDb
System.Data.Common
System.Data.SqlClient
System.Data.SqlTypes
```

System.Data: Các lớp của ADO.NET

System.Data.OleDb: Các lớp làm việc với dữ liệu OLEDB

System.Data.SqlClient: Các lớp làm việc với cơ sở dữ liệu SQL Server

II. Các đối tượng

ADO.NET Data Providers

Là các lớp truy nhập dữ liệu nguồn: MicrosoftSQL Server™2000, SQL Server 7, Oracle, Microsoft Access

Thiết lập kết nối giữa **DataSets** và dữ liệu nguồn

Có 2 thư viện ADO.NET Data Providers

System.Data.OleDb: Dùng truy nhập cơ sở dữ liệu OLE

System.Data.SqlClient: Truy nhập SQL Server

<u>SqlClient</u>	<u>OleDb</u>
SqlCommand	OleDbCommand
SqlConnection	OleDbConnection
SqlDataReader	OleDbDataReader
SqlDataAdapter	OleDbDataAdapter

Đối tượng Connection

Biểu diễn kết nối tới cơ sở dữ liệu

```
//Kết nối tới cơ sở dữ liệu MS Access
string conStr = "Provider=Microsoft.Jet.OLEDB.4.0;" + "Data
```

```
Source=<DataName>;
```

```
OleDbConnection myConnection = new OleDbConnection(conStr);
```

```

myConnection.Open();

//Kết nối với cơ sở dữ liệu SQL Server
String conStr = "DataSource = <Computer Name>; " + "Persist Security Info = true; "
               + "Initial Catalog = <DataName>; " + "User Id
               =name; Password=psw; "
               + "Connect Timeout = <seconds> ";
SqlConnection myConnection = new SqlConnection(conStr);
myConnection.Open();

```

Đối tượng DataAdapter

Dùng để lấy dữ liệu từ dữ liệu nguồn vào DataSet
 Dùng để cập nhật dữ liệu từ DataSet vào dữ liệu nguồn
 OleDbDataAdapter làm việc với CSDL MS Access
 SqlDataAdapter làm việc với dữ liệu SQL Server

Ví dụ phương thức Fill lấy dữ liệu vào DataTable:

```

String conStr = "Data Source = may01;" + "Initial Catalog = QLSV; " + "Persist
Security Info = true; " + "User Id =sa; Password=sa; Connect Timeout =50
";
//Kết nối với cơ sở dữ liệu
SqlConnection myConnection = new SqlConnection(conStr);
myConnection.Open();
string sqlStr= "SELECT * FROM tblKhoaDaoTao";
SqlDataAdapter myDataAdapter= new SqlDataAdapter(sqlStr,myConnection);
DataSet myDataSet= new DataSet();
myDataAdapter.Fill(myDataSet,"tblKhoaDaoTao");
DataTable myTable=myDataSet.Tables["tblKhoaDaoTao"];

```

Ví dụ xây dựng form hiển thị dữ liệu

Mã code:

```
private string conStr = "Data Source = (local); " + "Initial Catalog = QLSinhVien;"  
+ "persist security info = true; " + "User Id=sa; Password=sa; Connect Timeout =50";  
private SqlConnection myConnection;  
private SqlDataAdapter myDataAdapter;  
private DataSet myDataSet;  
private DataTable myTable;  
private int pos = 0;  
private void frmDataTable_Load(object sender, EventArgs e)  
{  
    myConnection = new SqlConnection(conStr);  
    myConnection.Open();  
    string SqlStr = "SELECT * FROM tblKhoaDaoTao",  
    myDataAdapter = new SqlDataAdapter(SqlStr, conStr);  
    myDataSet = new DataSet();  
    myDataAdapter.Fill(myDataSet, "tblKhoaDaoTao");  
    myTable = myDataSet.Tables["tblKhoaDaoTao"];  
    btnFirst.PerformClick();  
}  
private void btnFirst_Click(object sender, EventArgs e)  
{  
    if(myTable.Rows.Count == 0) return;  
    pos = 0;  
    txtMaKhoa.Text = myTable.Rows[pos]["MaKhoa"].ToString();  
    txtTenKhoa.Text = myTable.Rows[pos]["TenKhoa"].ToString();  
}  
private void btnPrevious_Click(object sender, EventArgs e)  
{  
    if(myTable.Rows.Count == 0) return;  
    pos--;  
    if(pos < 0) pos = 0;  
    txtMaKhoa.Text = myTable.Rows[pos]["MaKhoa"].ToString();  
    txtTenKhoa.Text = myTable.Rows[pos]["TenKhoa"].ToString();  
}  
private void btnNext_Click(object sender, EventArgs e)  
{  
    if(myTable.Rows.Count == 0) return;  
    pos++;  
    if(pos > myTable.Rows.Count -1) pos = myTable.Rows.Count -1;  
    txtMaKhoa.Text = myTable.Rows[pos]["MaKhoa"].ToString();  
    txtTenKhoa.Text = myTable.Rows[pos]["TenKhoa"].ToString();
```

```

        }
        private void btnLast_Click(object sender, EventArgs e)
        {
            if(myTable.Rows.Count == 0) return;
            pos = myTable.Rows.Count - 1;
            txtMaKhoa.Text = myTable.Rows[pos]["MaKhoa"].ToString();
            txtTenKhoa.Text = myTable.Rows[pos]["TenKhoa"].ToString();
        }
    
```

Đối tượng DataGridView

- Dùng để hiển thị dữ liệu từ 1 DataTable
- Cách thực hiện
 - ✓ Thêm đối tượng DataGridView vào Form
 - ✓ Nháy chuột phải và chọn **Add column** hoặc **Editcolumns**
 - ✓ Lần lượt chọn **Add** để thêm các cột
 - ✓ Mỗi cột cần khai báo các thuộc tính
 - **Name:** Tên cột dùng trong mã lệnh
 - **Header text:** Tiêu đề hiển thị của cột
 - **DataPropertyName:** Tên cột dữ liệu của DataTable.
 - **DataSource:** Tên DataTable cần hiển thị lên lưới
 - **AutoGenerateColumns:** Tự động lấy các cột nếu bằng true, ngược lại lấy đúng số cột đã khai báo.
 - **dataGridView1.AutoGenerateColumns = false;**
 - **dataGridView1.DataSource = myTable;**
 - **AllowUserToAddRows:** Cho/không thêm dòng trên lưới
 - **AllowUserToDeleteRows:** Cho/không xoá dòng trên lưới
 - ✓ Sự kiện thường dùng
 - **RowEnter:** Xảy ra khi con trỏ đưa vào một dòng
 - **e.RowIndex:** Dòng hiện thời
 - **e.ColumnIndex:** Cột hiện thời

Ví dụ:

Hiển thị dữ liệu trong bảng **tblKhoaDaoTao** lên lưới, khi chuyển con trỏ trên lưới dữ liệu hiển thị lên TextBox.

```

private string conStr = "Data Source = (local); " + "Initial Catalog = QLSinhVien;" +
+ "persist security info = true; " + "User Id=sa; Password=sa; Connect Timeout =50";
private SqlDataAdapter myDataAdapter;
private DataSet myDataSet;
private DataTable myTable;
private void frmDataGridView_Load(object sender, EventArgs e)
{
    string SqlStr = "SELECT * FROM tblKhoaDaoTao";
    myDataAdapter = new SqlDataAdapter(SqlStr, conStr);
    myDataSet = new DataSet();
    myDataAdapter.Fill(myDataSet, "tblKhoaDaoTao");
    myTable = myDataSet.Tables["tblKhoaDaoTao"]; //Chuyen len luoi
    dataGridView1.DataSource = myTable;
    dataGridView1.AutoGenerateColumns = false;
}
private void dataGridView1_RowEnter(object sender, DataGridViewCellEventArgs e)
{
    try
    {
        int row = e.RowIndex;
        txtMaKhoa.Text = myTable.Rows[row]["MaKhoa"].ToString();
        txtTenKhoa.Text = myTable.Rows[row]["TenKhoa"].ToString();
    }
    catch(Exception e){}
}

```

Đối tượng SqlCommand

- Dùng để thực hiện câu lệnh SQL: **Insert; Update; Delete**
- Khai báo biến

private string conStr = “Data Source = ...;”;

- ```

private SqlConnection myConnection;
private SqlCommand myCommand;

```
- Mở kết nối
 

```
myConnection = new SqlConnection(conStr);
myConnection.Open();
```
  - Thực hiện câu lệnh SQL
 

```
myCommand = new SqlCommand(sqlStr,myConnection);
myCommand.ExecuteNonQuery();
```
  - ❖ **Chú ý:** sqlStr là câu lệnh SQL
  - Thiết kế Form cho phép nhập, xoá bảng dữ liệu tblKhoaDaoTao

```

private string conStr = "Data Source = (local); " + "Initial Catalog = QLSinhVien; "
+ "persist security info = true;" + "User Id=sa; Password=sa; Connect Timeout =50";
private SqlDataAdapter myDataAdapter;
private SqlCommand myCommand;
private SqlConnection myConnection;
private DataSet myDataSet;
private DataTable myTable;
public frmSqlCommand()
{
 InitializeComponent();
}
private void SetControls(bool edit)
{
 txtMaKhoa.Enabled = edit;
 txtTenKhoa.Enabled = edit;
 btnAdd.Enabled = !edit;
 btnSave.Enabled = edit;
}

```

```

 }
 private void Display()
 {
 string SqlStr = "SELECT * FROM tblKhoaDaoTao";
 myDataAdapter = new SqlDataAdapter(SqlStr, myConnection);
 myDataSet = new DataSet();
 myDataAdapter.Fill(myDataSet, "tblKhoaDaoTao");
 myTable = myDataSet.Tables["tblKhoaDaoTao"];//Chuyen len luoi
 dataGridView1.DataSource = myTable;
 dataGridView1.AutoGenerateColumns = false;
 }
 private void frmSqlCommand_Load(object sender, EventArgs e)
 {//Mo ket noi
 myConnection = new SqlConnection(conStr);
 myConnection.Open();
 Display();
 SetControls(false);
 }
 private void dataGridView1_RowEnter(object sender, DataGridViewCellEventArgs e)
 {
 try{
 int row = e.RowIndex;
 txtMaKhoa.Text = myTable.Rows[row]["MaKhoa"].ToString();
 txtTenKhoa.Text = myTable.Rows[row]["TenKhoa"].ToString();
 }
 catch(Exception){}
 }
 private void btnAdd_Click(object sender, EventArgs e)
 {
 txtMaKhoa.Clear();
 txtTenKhoa.Clear();
 SetControls(true);
 txtMaKhoa.Focus();
 }
 private void btnSave_Click(object sender, EventArgs e)
 {
 string sSql = "Insert Into tblKhoaDaoTao (MaKhoa, TenKhoa)+"+"Values (N"++
 txtMaKhoa.Text + ",N" + txtTenKhoa.Text + ")";
 myCommand = new SqlCommand(sSql, myConnection);
 myCommand.ExecuteNonQuery();
 Display();
 SetControls(false);
 }
}

```

```

private void btnDelete_Click(object sender, EventArgs e)
{
 string sSql = "Delete From tblKhoaDaoTao " + "Where MaKhoa = N'" +
 txtMaKhoa.Text + "'";
 myCommand = new SqlCommand(sSql, myConnection);
 myCommand.ExecuteNonQuery();
 Display();
}

```

### **Đối tượng SqlCommandBuilder**

- Tự động thực hiện **Update, Insert, Delete**

- Khai báo các biến

```

private string conStr = "Data Source = ;";
private SqlConnection myConnection;
private SqlDataAdapter myDataAdapter;
private SqlCommandBuilder myCommandBuilder;
private DataSet myDataSet;
private DataTable myTable;
private string sqlStr;

```

- Tạo kết nối tới cơ sở dữ liệu

```

myConnection = new SqlConnection(conStr);
//Tạo một SqlDataAdapter
myDataAdapter = new SqlDataAdapter(sqlStr, myConnection);
//Tạo một SqlCommandBuilder
myCommandBuilder = new SqlCommandBuilder(myDataAdapter);
//Tạo một DataTable
myDataSet = new DataSet();
myDataAdapter.Fill(myDataSet, "....");

```

- ```

myTable = myDataSet.Tables[“....”];

```
- Xoá một dòng
- ```

myTable.Rows[pos].Delete(); //pos là dòng cần xoá
myDataAdapter.Update(myTable);
//Thêm một dòng
DataRow newRow = myTable.NewRow();
newRow[“MAKHOA”] = txtMakhoa.Text;
newRow[“TENKHOA”] = txtTen.Text;
myTable.Rows.Add(newRow);
myDataAdapter.Update(myTable);

```
- Sửa một dòng
- ```

DataRow editRow =myTable.Rows[pos];      //pos là dòng cần sửa
editRow[“MAKHOA”] = txtMakhoa.Text;
editRow[“TENKHOA”] = txtTen.Text;
myDataAdapter.Update(myTable);
//Loại bỏ sửa đổi dòng
myTable.RejectChanges();

```
- Thiết kế Form cho phép nhập, xoá bảng dữ liệu tblKhoaDaoTao

```

private string conStr = "Data Source = (local); " + "Initial Catalog = QLSinhVien; "
+ "persist security info = true;" + "User Id=sa; Password=sa; Connect Timeout =50";
private SqlDataAdapter myDataAdapter;
private SqlCommandBuilder myCommandBuilder;
private SqlConnection myConnection;
private DataSet myDataSet;
private DataTable myTable;
private void SetControls(bool edit)
{
    txtMaKhoa.Enabled = edit;

```

```

        txtTenKhoa.Enabled = edit;
        btnAdd.Enabled = !edit;
        btnSave.Enabled = edit;
    }
    private void Display()
    {
        string SqlStr = "SELECT * FROM tblKhoaDaoTao";
        myDataAdapter = new SqlDataAdapter(SqlStr, myConnection);
        myCommandBuilder = new SqlCommandBuilder(myDataAdapter);
        myDataSet = new DataSet();
        myDataAdapter.Fill(myDataSet, "tblKhoaDaoTao");
        myTable = myDataSet.Tables["tblKhoaDaoTao"];
        dataGridView1.DataSource = myTable;
        dataGridView1.AutoGenerateColumns = false;
    }
    private void frmCommandBuilder_Load(object sender, EventArgs e)
    {
        myConnection = new SqlConnection(conStr);
        myConnection.Open();
        Display();
        SetControls(false);
    }
    private void dataGridView1_RowEnter(object sender, DataGridViewCellEventArgs e)
    {
        try
        {
            int row = e.RowIndex;
            txtMaKhoa.Text = myTable.Rows[row]["MaKhoa"].ToString();
            txtTenKhoa.Text = myTable.Rows[row]["TenKhoa"].ToString();
        }
        catch (Exception){}
    }
    private void btnAdd_Click(object sender, EventArgs e)
    {
        txtMaKhoa.Clear();
        txtTenKhoa.Clear();
        SetControls(true);
        txtMaKhoa.Focus();
    }
    private void btnSave_Click(object sender, EventArgs e)
    {
        DataRow newRow = myTable.NewRow();
        newRow["MaKhoa"] = txtMaKhoa.Text;

```

```

newRow["TenKhoa"] = txtTenKhoa.Text;
myTable.Rows.Add(newRow);
myDataAdapter.Update(myTable);
SetControls(false);
}
private void btnDelete_Click(object sender, EventArgs e)
{
    int pos = dataGridView1.CurrentRow.Index;
    myTable.Rows[pos].Delete();
}
private void btnClose_Click(object sender, EventArgs e){this.Close();}
➤ Bài tập: Thiết kế form

```

➤ Đôi tượng ListBox và ComboBox

- ✓ **ListBoxes:** Cho phép người dùng xem và chọn các dòng dữ liệu từ danh sách
- ✓ **ComboBox:** Sự kết hợp của **TextBox** và **ListBox**
- ✓ **DataSource:** Nguồn dữ liệu, là một DataTable
- ✓ **DisplayMember:** Cột hiển thị trong ListBox
- ✓ **ValueMember:** Cột giá trị trả về khi chọn ListBox
- ✓ **SelectedIndex:** Dòng hiện thời được chọn
- ✓ **SelectedValue:** Giá trị được chọn trên ListBox

➤ Ví dụ hiển thị dữ liệu trong tblKhoaDaoTao

```

if(myTable.Rows.Count > 0)
{
    comboBox1.DataSource = myTable;
    comboBox1.DisplayMember = "TenKhoa";
    comboBox1.ValueMember = "MaKhoa";
}

```

```

        comboBox1.SelectedIndex = 0;
    }

```

Giá trị trả về khi chọn là: **comboBox1.SelectedValue**

III. Lớp truy nhập cơ sở dữ liệu

- Mọi form không phụ thuộc vào cơ sở dữ liệu
- Chỉ kết nối dữ liệu một lần khi chạy ứng dụng
- Mỗi form sử dụng các đối tượng lớp
 - ✓ Lấy dữ liệu vào DataTable
 - ✓ Cập nhật dữ liệu từ DataTabe vào cơ sở dữ liệu
 - ✓ Thực hiện các câu lệnh SQL thao tác với dữ liệu

➤ Mã nguồn:

```

private static SqlConnection myConnection;
private SqlDataAdapter myDataAdapter;
public DataServices(){}
~DataServices(){}
public bool OpenDB(string myComputer,string myDatabase,string myUser,string
myPass) {
    string strConnection = "Data Source="" + myComputer + ":" + "Initial
Catalog="" +      myDatabase + ";" +"Persist Security Info=True;" +"User ID="" +
myUser +      ";" + "Password="" +myPass + ";" +"Connect Timeout =120";
    if ((myConnection != null) && (myConnection.State == ConnectionState.Open))
        return true;
    try {   myConnection = new SqlConnection(strConnection);
        myConnection.Open();
    }
    catch (SqlException ex) {
        DisplayError(ex);
        myConnection = null;
        return false;
    }
    return true;
}
public void CloseDB() {
    if (myConnection != null) {
        if (myConnection.State == ConnectionState.Open) {
            try { myConnection.Close(); }
            catch (Exception){}
            myConnection = null;
        }
    }
}
private void DisplayError(SqlException ex){

```

```

string message;
switch (ex.Number) {
    case 17:
        message = "Server không đúng !";
        break;
    case 4060:
        message = "Cơ sở dữ liệu không đúng !";
        break;
    case 18456:
        message = "Không đúng tên truy nhập và mật khẩu !";
        break;
    case 547:
        message = "Vi phạm khoá ngoài !";
        break;
    case 2627:
    case 2601:
        message = "Trùng giá trị trường khoá";
        break;
    case 8152:
        message = "Dữ liệu nhập quá dài";
        break;
    default:
        message = ex.Message;
        break;
}
MessageBox.Show(message, "Error " + ex.Number.ToString());
}

public DataTable RunQuery(string QueryString, string TableName){
    DataSet myDataSet = new DataSet();
    myDataAdapter = new SqlDataAdapter();
    try{
        myDataAdapter.SelectCommand = new SqlCommand(QueryString,
myConnection);
        SqlCommandBuilder myCommandBuilder = new
                    SqlCommandBuilder(myDataAdapter);
        if (TableName.Trim().Length > 0)
            myDataAdapter.Fill(myDataSet, TableName);
        else    myDataAdapter.Fill(myDataSet);
    }
    catch (SqlException ex) {
        DisplayError(ex);
        return null;
    }
}

```

```
        return myDataSet.Tables[TableName];
    }
    public void Update(DataTable myDataTable) {
        try{ myDataAdapter.Update(myDataTable); }
        catch (SqlException ex){ DisplayError(ex); }
    }
    public void ExecuteNonQuery(string cmdText) {
        SqlCommand myCommand = new SqlCommand(cmdText, myConnection);
        try{   myCommand.ExecuteNonQuery(); }
        catch (SqlException ex){ DisplayError(ex); }
    }
}
```