# Chương 1 Mở Đầu

Các thành phần cơ bản của .NET:

* User application
* .NET framework
* Hệ điều hành OS
* Device drivers
* Hareware component

# Các khái niệm

## Biến

### Khai báo biến

* + - Cú pháp

<Kiểu dữ liệu> <Tên biến> = <Giá trị khởi tạo>

* + - Ví dụ :

String name

Double math = 0

Double chemistry = 0

### Biến toàn cục

Biến toàn cục là biến được khai báo bên ngoài tất cả các hàm, thủ tục và được hiểu bên trong tất cả các hàm, thủ tục.

Để khai báo biến toàn cục cho một đơn thể ta chỉ cần khai báo biến đó ở đầu đơn thể. Khi đó biến toàn cục của đơn thể sẽ được hiểu trong tất cả các hàm của đơn thể nếu không trùng tên với các biến địa phương.

### Biến cục bộ

Biến cục bộ là biến được hiểu bên trong một phạm vi nào đó của chương trình. Ra khỏi phạm vi này, nó không còn được biết đến nữa.

## Các kiểu dữ liệu cơ sở

|  |  |  |
| --- | --- | --- |
| **Kiểu** | **Số byte** | **Mô tả** |
| byte | 1 | Số nguyên dương không dấu từ 0-255 |
| char | 2 | Ký tự Unicode |
| bool | 1 | Giá trị logic true/ false |
| sbyte | 1 | Số nguyên có dấu ( từ -128 đến 127) |
| short | 2 | Số nguyên có dấu giá trị từ -32768 đến 32767. |
| ushort | 2 | Số nguyên không dấu 0 – 65.535 |
| int | 4 | Số nguyên có dấu –2.147.483.647 và 2.147.483.647 |
| uint | 4 | Số nguyên không dấu 0 – 4.294.967.295 |
| float | 4 | Ký tự UnicodeKiểu dấu chấm động, giá trị xấp xỉ từ 3,4E-38 đến 3,4E+38, với 7 chữ số có nghĩa.. |
| double | 8 | Kiểu dấu chấm động có độ chính xác gấp đôi, giá trị xấp xỉ từ 1,7E-308 đến 1,7E+308, với 15,16 chữ số có nghĩa |
| decimal | 8 | Có độ chính xác đến 28 con số và giá trị thập phân, được dùng trong tính toán tài chính, kiểu này đòi hỏi phải có hậu tố “m” hay “M” theo sau giá trị |
| long | 8 | Kiểu số nguyên có dấu có giá trị trong khoảng : -9.223.370.036.854.775.808 đến 9.223.372.036.854.775.807 |
| ulong | 8 | Số nguyên không dấu từ 0 đến 0xffffffffffffffff |

## Cấu trúc chương trình

Cấu trúc của một chương trình đơn giản nhất là có duy nhất một module và trong module này sẽ chứa thủ tục main và các hàm, các thủ tục cần thiết khác.

* + - Ví dụ :

static class Declaration

{

static string Name;

static int Age;

static double Salary;

public static void Main()

{

Input();

Output();

}

public static void Input()

{

Console.Write("Name : ");

Name = Console.ReadLine();

Console.Write("Age : ");

Age = int.Parse(Console.ReadLine());

Console.Write("Salary : ");

Salary = double.Parse(Console.ReadLine());

}

public static void Output()

{

Console.WriteLine("Name : " + Name);

Console.WriteLine("Age : " + Age);

Console.WriteLine("Salary : " + Salary);

}

}

## Tham số, Hàm và Thủ tục

Tham số là các tên, đinh danh, ký hiệu thuộc phần dịnh nghĩa của hàm, thủ tục. Các hàm, thủ tục nhận thông tin đầu vào từ tham số để thực hiện các xử lý. Thông thường các tham số được gán giá trị cụ thể trước khi hàm được thực thi.

Có 2 loại tham số :

Tham biến : Có thể thay đổi về giá trị sau lời gọi hàm, bắt đầu với từ khóa ref, out, params.

Tham trị : Không thay đổi giá trị sau lời gọi hàm, không có tham số.

* + - Ví dụ :

public void getFullSalary(double Salary, int Hours, ref double FullSalary)

{

FullSalary = Hours \* Salary;

}

## Kiểu cấu trúc

Kiểu cấu trúc là một phương pháp dùng để tích hợp các kiểu dữ liệu đơn thành kiểu dữ liệu phức để mô tả thông tin của một đối tượng hay một khái niệm trong thế giới thực.

* + - Ví dụ :

static class Struct

{

public struct Emloyee

{

public string sName;

public int iAge;

public double dSalary;

}

public static void Input(ref Emloyee x)

{

Console.Write("Name : ");

x.sName = Console.ReadLine();

Console.Write("Age : ");

x.iAge = int.Parse(Console.ReadLine());

Console.Write("Salary : ");

x.dSalary = double.Parse(Console.ReadLine());

}

public static void Output(Emloyee x)

{

Console.WriteLine("Name : " + x.sName);

Console.WriteLine("Age : " + x.iAge);

Console.WriteLine("Salary : " + x.dSalary);

}

public static void Main()

{

Emloyee x = new Emloyee();

Input(ref x);

Output(x);

}

}

static class Struct

{

public static void Main()

{

Emloyee x;

Input(out x);

Output(x);

}

public struct Emloyee

{

public string sName;

public int iAge;

public double dSalary;

}

public static void Input(out Emloyee x)

{

Console.Write("Name : ");

x.sName = Console.ReadLine();

Console.Write("Age : ");

x.iAge = int.Parse(Console.ReadLine());

Console.Write("Salary : ");

x.dSalary = double.Parse(Console.ReadLine());

}

public static void Output(Emloyee x)

{

Console.WriteLine("Name : " + x.sName);

Console.WriteLine("Age : " + x.iAge);

Console.WriteLine("Salary : " + x.dSalary);

}

}

## Cấu trúc điều khiển rẽ nhánh

### Cấu trúc điều khiển rẽ nhánh If

* + - Cú pháp:

if (<điều kiện>)

{

//câu lệnh thực thi nếu điều kiện thỏa

}

* + - Ví dụ

if (2>3)

Console.write(“2 lớn hơn 3”);

* + - * 2 lớn hơn 3

### Cấu trúc điều khiển rẽ nhánh If…Esle

* + - Cú pháp :

if (<điều kiện>)

{

//câu lệnh thực thi nếu điều kiện thỏa

}

esle

{

//câu lệnh thực thi nếu không thỏa điều kiện

}

* + - Ví dụ :

if (score > 9.0)

Console.write("Excellent");

else

Console.write(“Not”);

* + - * Score = 9.2
      * Excellent

### Cấu trúc điều khiển rẽ nhánh If…Esle

if (< Điều kiện> )

< Lệnh >

elseIf (< Điều kiện>)

< Lệnh >

else

< Lệnh >

* + - Ví dụ :

if (score >= 9.0)

Console.WriteLine("Excellent")

elseIf (score >= 8.0)

Console.WriteLine("Good")

elseIf (score >= 7.0)

Console.WriteLine("Fair")

else

Console.WriteLine("Bad")

* + - * Score = 7.2
      * Fair

### Cấu trúc điều khiển rẽ nhánh Switch Case

* + - Cú pháp :

switch (<điều kiện>)

{

case <giá trị>: //câu lệnh

break;

case <giá trị>: //câu lệnh

break;

......

default: // câu lệnh

}

* + Điều kiện: tham số cần xét.
  + Giá trị: có thể là số, chuỗi hay kí tự.
    - Ví dụ :

switch (thang)

{

case 1:

Console.write(“Tháng gieng”);

break;

case 2:

Console.write(“Tháng hai”);

break;

case 3:

Console.write(“Tháng ba”);

break;

case 4:

Console.write(“Tháng tư”);

break;

case 5:

Console.write(“Tháng nam”);

break;

case 6:

Console.write(“Tháng sau”);

break;

case 7:

Console.write(“Tháng bay”);

break;

case 8:

Console.write(“Tháng tam”);

break;

case 9:

Console.write(“Tháng chin”);

break;

case 10:

Console.write(“Tháng muoi”);

break;

case 11:

Console.write(“Tháng muoi mot”);

break;

case 12:

Console.write(“Tháng muoi hai”);

break;

default: Console.write (“Thang ko hop le”);

}

## Cấu trúc điều khiển lặp

### Cấu trúc điều khiển For

* + - Cú pháp :

for (<phần khởi tạo>;<điều kiện>;<bước lặp>)

{

//lệnh thực thi

}

* + Phần khởi tạo: chứa giá trị biến khởi tạo của vòng lặp. Có thể có nhiều giá trị khởi tạo, mỗi giá trị cách nhau bởi dấu phẩy.
  + Điều kiện: chứa điều kiện lặp lại của vòng lặp.
  + Bước lặp: chứa biểu thức tăng bước lặp sau mỗi vòng lặp.
    - Ví dụ :

for (int i = 0; i <= 100 ; i+=2)

{

Console.WriteLine("Count : " & i)

}

### Cấu trúc điều khiển Foreach

* + - Cú pháp :

foreach ( <kiểu tập hợp> <tên biến truy cập> in < tên tập hợp>)

{

//lệnh thực thi

}

* + - Ví dụ :

int[] a = {1,4,7};

Console.write(“Cac phan tu cua a la :”);

foreach ( int i in a)

{

Console.write(“%d ”,i);

}

* + - * Cac phan tu cua a la : 1 4 7

### Cấu trúc điều khiển While

* + - Cú pháp :

while ( <điều kiện lặp>)

{

//lệnh thực thi

}

* + - Ví dụ :

int i = 0;

Console.write(“Cac so nho hon 5: “);

while ( i < 5 )

{

Console.write(“%d ”,i);

i++;

}

* + - * Cac so nho hon 5 : 0 1 2 3 4

### Cấu trúc điều khiển Do…While

* + - Cú pháp :

do

{

//lệnh thực thi

}while (<điều kiện lặp>) ;

* + - Ví dụ :

int i = 0;

Console.write(“Cac so nho hon 5: “);

do

{

Console.write(“%d ”,i);

i++;

} while ( i < 5 ) ;

* + - * Kết quả: Cac so nho hon 5 : 0 1 2 3 4

### Câu lệnh nhảy

* + - goto
      * Nhảy đến một nhãn đã đặt sẳn
      * Cú pháp : goto <tên nhãn>

vd:

int a = 1;

tiep: Console.write (“Hello word”);

if ( a < 2 )

goto tiep;

* + - break : thoát khỏi vòng lặp.

vd:

for (int i = 0; i < 5; i++)

{

if (i == 3)

break;

Console.write(“%d ”,i);

}

* + - * 0 1 2
    - continue : bỏ qua các câu lệnh sau conitnue và quay lại đầu của vòng lặp

vd:

for (int i = 0; i < 5; i++)

{

if (i == 3)

continue;

Console.write (“%d ”, i);

}

* + - * 0 1 2 4

## Các Toán Tử

### Toán tử số học

Là dạng phép tính giản đơn cộng, trừ, nhân, chia trong số học. Ngoài ra còn có phép chia lấy dư (%).

#### Toán tử số học cơ bản

|  |  |  |
| --- | --- | --- |
| **Toán tử** | **Ý nghĩa** | **Ví dụ** |
| **+** | Toán tử cộng | a = a +2; |
| **-** | Toán tử trừ | a= a - 2; |
| **\*** | Toán tử nhân | a\*=2; |
| **/** | Toán tử chia | a/=2; |
| **%** | Toán tử lấy phần dư sau khi chia. |  |

#### Toán tử số học viết tắt

|  |  |  |
| --- | --- | --- |
| **Toán tử** | **Ý nghĩa** | **Ví dụ** |
| **=** | Toán tử gán | a = a +2; |
| **+=** | Toán tử cộng thêm giá trị toan hạng bên phải | a+=2; |
| **-=** | Toán tử trừ đi giá trị toán hạng bên phải | a-=2; |
| **\*=** | Toán tử nhân thêm giá trị toán hạng bên phải | a\*=2; |
| **/=** | Toán tử chia đi giá trị toán hạng bên phải | a/=2; |
| **%=** | Toán tử lấy phần dư sau khi chia bớt toạn hạng bên phải. |  |

### Toán tử gán

Trong các ngôn ngữ lập trình toán tử gán được dùng để gán giá trị của biến này cho biến khác. Trong ngôn ngữ lập trình hướng đối tượng toán tử gán còn được dùng để gán thành phần dữ liệu của đối tượng này cho đối tượng khác.

### Toán tử so sánh

|  |  |  |
| --- | --- | --- |
| **Toán tử** | **Ý nghĩa** | **Ví dụ** |
| **==** | So sánh bằng. | (2==2) 🡺 True  (2==3) 🡺 False |
| **!=** | Không bằng | (2!=3) 🡺 True  (2!=2) 🡺 False |
| **>** | Lớn hơn | (3>2) 🡺 True  (2>4) 🡺 False |
| **>=** | Lớn hơn hay bằng | (2>=2) 🡺 True  (2>=3) 🡺 False |
| **<** | Nhỏ hơn | (2<3) 🡺 True  (2<1) 🡺 False |
| **<=** | Nhỏ hơn hay bằng | (2<=2) 🡺 True  (2<=0) 🡺 False |

### Toán tử luận lý

|  |  |  |
| --- | --- | --- |
| **Toán tử** | **Ý nghĩa** | **Ví dụ** |
| **&&** | Toán tử và | (2==2 && 3<13) 🡺 True  (2==1 && 3>=3) 🡺 False |
| **||** | Toán tử hoặc | (2==2 || 3!=3) 🡺 True  (2!=2 || 2>3) 🡺 False |
| **!** | Phủ định | (2>3) 🡪 False 🡺 !(2>1) 🡪 True  (2>1) 🡪 True 🡺 !(2>1) 🡪 False |

# Chương 2 Lập Trình Hướng Đơn Thể Trong VB.NET

## Thủ tục

Hàm (function), nguyên nghĩa tiếng Anh có nghĩa là chức năng. Trong lập trình, ta có thể hiểu hàm là một đoạn chương trình được xây dựng để thực hiện một chức năng nào đó. Đoạn chương trình này chỉ cần phải viết duy nhất một lần, và có thể được sử dụng nhiều lần trong toàn bộ chương trình. Một hàm sẽ được xác định bởi tên hàm và các tham số đầu vào liên quan đến hàm đó. Thông thường, hàm sẽ trả về một kết quả nào đó.

* + - Cú pháp :

void <tên hàm> (<kiểu dữ liệu> tham số)

{

//lệnh thực thi

}

Gọi hàm : <tên hàm> (<danh sách tham số>)

* + - Ví dụ :

void Input(string Name, int Age,double Salary)

{

sName = Name

iAge = Age

dSalaryPerHour = Salary

Console.WriteLine("Name : " & sName)

Console.WriteLine("Age : " & iAge)

Console.WriteLine("Salary : " & dSalaryPerHour)

}

Input(“A”, 5 , 2000);

* + - * sName : A
      * Age : 5
      * Salary : 2000

## Hàm

Một thủ tục là dãy các câu lệnh để thực thi một công việc, một chức năng đặc thù nào đó. Nói cách khác thủ tục là một nhóm các câu lệnh được đưa vào một khối và được đặt tên. Giống như một hàm các thủ tục chỉ được viết 1 lần và có thể được sử dụng nhiều lần. Lưu ý, thủ tục không có giá trị trả về.

* + - Cú pháp :

<kiểu dữ liệu> tên\_hàm (<kiểu dữ liệu> tham số)

{

//lệnh thực thi

}

* + - Ví dụ :

int tinhtong(int a, int b)

{

int tong = 0;

tong = a + b;

return tong;

}

int tong = tinhtong(3,4);

* + - * tong = 8

## Đơn thể

Đơn thể là một tập hợp các thủ tục và hàm dùng để thực thi một số chức năng nào đó của chương trình. Có 2 loại đơn thể :

* Đơn thể hướng chức năng.
* Đơn thể hướng dữ liệu.

## Namespace

Namespace là hệ thống các đơn thể

# Chương 3 : Phương Pháp Lập Trình Hướng Đối Tượng Trong C#

## Lớp đối tượng

Đối tượng là một trị có thể được tạo ra, lưu giữ và sử dụng. Trong C# tất cả các biến đều là đối tượng. Các biến kiểu số, kiểu chuỗi … đều là đối tượng. Mỗi một đối tượng đều có các biến thành viên để lưu giữ dữ liệu và có các phương thức (hàm) để tác động lên biến thành viên. Mỗi đối tượng thuộc về một lớp đối tương nào đó. Các đối tượng có cùng lớp thì có cùng các biến thành viên và phương thức.

### Lớp:

Hiểu một cách đơn giản là sự tích hợp giữa hai thành phần: thành phần dữ liệu và thành phần xử lý.

### Cú pháp khai báo

[attribute][bổ từ truy xuất] *class* **định danh** [:lớp cơ sở]

{

//thân lớp

}

Ví dụ: Khai báo một lớp

public class Tester

{

public static int Main( )

{

...

}

}

### Ghi chú

Khi khai báo một lớp ta định nghĩa các đặc tính chung của tất cả các đối tượng của lớp và các hành vi của chúng

### Đối tượng

### Khái niệm

Đối tượng là một sự thể hiện của lớp. Một lớp có thể có nhiều sự thể hiện khác nhau.

### Cú pháp khai báo

*{ objectclass | Object } VariableName = new { objectclass | Object }*

Ví dụ minh họa:

HocSinh a = new HocSinh();

HocSinh x = new HocSinh();

HocSinh y = new HocSinh();

Trong ví dụ trên, ta nói: a, x và y là 3 đối tượng thuộc lớp CHocSinh. Nói cách khác, trong ví dụ này lớp CHocSinh có 3 thể hiện.

### Ghi chú

* Có thể có nhiều đối tượng cùng thuộc về một lớp.

Từ khóa để tham khảo khái niệm đối tượng trong MSDN là “object variable, declaring”

### Phương thức

### Khái niệm

Phương thức là khả năng mà một đối tượng thuộc về lớp có thể thực hiện.

### Cú pháp

// phương thức public

public void DisplayCurrentTime( )

{

Console.WriteLine( "stub for DisplayCurrentTime" );

}

## Chương trình minh họa

***Chương trình 1:*** Viết chương trình nhập họ tên, điểm toán, điểm văn của một học sinh. Tính điểm trung bình và xuất kết quả.

* **Lớp học sinh**

Public Class CHocSinh

{

string HoTen;

int Toan;

private int Van;

private double DiemTrungBinh;

Public void Nhap()

{

Console.Write("Nhap ho ten:");

HoTen = Console.ReadLine();

Console.Write("Nhap toan:");

Toan = Console.ReadLine();

Console.Write("Nhap van:");

Van = Console.ReadLine();

}

Public void XuLy()

{

DiemTrungBinh = (Toan + Van) / 2;

}

Public void Xuat()

{ Console.WriteLine("Ho ten :" + HoTen);

Console.WriteLine("Toan :" + Toan);

Console.WriteLine("Van:" + Van);

Console.WriteLine("Diem Trung Binh:" + DiemTrungBinh);

}

}

* **Hàm main()**

Void main()

{

ChocSinh hs = new ChocSinh();

hs.Nhap();

hs.XuLy();

hs.Xuat();

}

***Chương trình 2:*** Viết chương trình nhập vào một phân số. Rút gọn phân số đó và xuất ra kết quả.

* **Lớp phân số**

namespace ConsoleApplication1

{

public class CPhanSo

{

private int Tu;

private int Mau;

public CPhanSo()

{

Tu = 0;

Mau = 1;

}

public void Nhap()

{

Console.Write("Nhap tu: ");

Tu = Convert.ToInt32(Console.ReadLine());

Console.Write("Nhap mau: ");

Mau = Convert.ToInt32(Console.ReadLine());

}

public void Xuat()

{

Console.WriteLine(Tu + "/" + Mau);

}

public void RutGon()

{

int a = Math.Abs(Tu);

int b = Math.Abs(Mau);

while (a \* b != 0)

{

if (a > b) a = a - b;

else b = b - a;

}

Tu = Tu / (a + b);

Mau = Mau / (a + b);

}

}

* **Hàm main()**

static void Main(string[] args)

{

CPhanSo ps = new CPhanSo();

ps.Nhap();

ps.RutGon();

ps.Xuat();

}

***Chương trình 3:*** Viết chương trình nhập tọa độ 2 điểm trong mặt phẳng Oxy. Tính khoảng cách giữa chúng và xuất ra kết quả.

class CDiem

{

private double x;

private double y;

public CDiem()

{

x = 0;

y = 0;

}

public void Nhap()

{

Console.Write("Nhap x: ");

x = Convert.ToInt32(Console.ReadLine());

Console.Write("Nhap y: ");

y = Convert.ToInt32(Console.ReadLine());

}

public void Xuat()

{

Console.Write("(" + x + "," + y + ")");

}

public double KhoangCach(CDiem P)

{

return Math.Sqrt((x - P.x) \* (x - P.x) + (y - P.y) \* (y - P.y));

}

}

* **Hàm main()**

static void Main(string[] args)

{

CDiem A = new CDiem();

CDiem B = new CDiem();

double kc

A.Nhap();

B.Nhap();

kc = A.KhoangCach(B);

A.Xuat();

B.Xuat();

Console.WriteLine("Khoang cach giua hai diem la:" + kc);

}

## Bài tập

Xây dựng các chương trình dưới đây bằng phương pháp lập trình hướng đối tượng với ngôn ngữ VB.NET

1. Viết chương trình nhập vào một phân số. Hãy cho biết phân số đó là âm, dương hay bằng 0.
2. Viết chương trình nhập tạo độ hai điểm trong không gian. Tính khoảng cách giữa chúng và xuất kết quả.
3. Viết chương trình nhập vào 2 phân số. Tìm phân số lớn hơn và xuất kết quả.
4. Viết chương trình nhập vào 2 số phức. Tính tổng, hiệu và tích, xuất kết quả.
5. Viết chương trình nhập tọa độ 3 đỉnh A, B, C của 1 tam giác trong mặt phẳng Oxy. Tính chu vi tam giác và xuất kết quả.
6. Viết chương trình nhập vào một ngày. Tìm ngày kế tiếp và xuất kết quả.
7. Viết chương trình nhập vào một ngày. Tìm ngày hôm qua và xuất kết quả.

## Phương Thức Thiết Lập Constructors

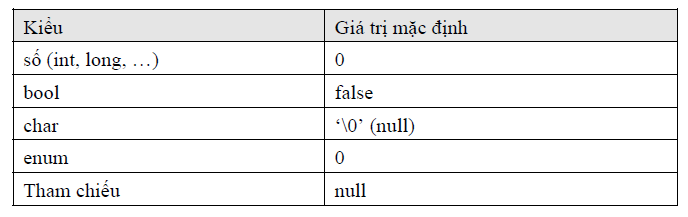
### Mục đích

Các phương thức thiết lập của một lớp có nhiệm vụ thiết lập thông tin ban đầu cho các đối tượng thuộc về lớp ngay khi đối tượng được khai báo. Là phương thức đầu tiên được triệu gọi và chỉ gọi một lần khi khởi tạo đối tượng, nó nhằm thiết lập các tham số đầu tiên cho đối tượng. Tên hàm trùng tên lớp; còn các mặt khác như phương thức bình thường

### Đặc Điểm

* Phương thức thiết lập của lớp được định nghĩa thông qua toán tử new.
* Không có giá trị trả về.
* Được tự động gọi thực hiện ngay khi đối tượng được khai báo.
* Có thể có nhiều phương thức thiết lập trong một lớp.
* Trong một quá trình sống của đối tượng thì chỉ có 1 lần duy nhất phương thức thiết lập được gọi thực hiện, đó là khi đối tượng được khai báo.
* Nếu lớp không định nghĩa hàm khởi tạo, trình biên dịch tự động tạo một hàm khởi tạo

mặc định. Khi đó các biến thành viên sẽ được khởi tạo theo các giá trị mặc định:



### Phân Loại

Để đơn giản ta có thể chia các phương thức thiết lập của một lớp thành 3 nhớm như sau:

* Phương thức thiết lập mặc định.
* Phương thức thiết lập sao chép.
* Phương thức thiết lập nhận tham số đầu vào.

Về mặt nguyên tắc có bao nhiêu phương thức khởi tạo thì có bấy nhiêu phương thức thiết lập, phương thức khởi tạo mặc định thì tương ứng với phương thức thiết lập mặc định, phương thức khởi tạo dựa vào 1 đối tượng khác tương ứng 1 phương thức thiết lập sao chép, các phương thức khởi tạo còn lại tương ứng với phương thức thiết lập nhận tham số đầu vào.

## Chương trình minh họa

*Chương trình 1*: Hãy xác định và cài đặt các phương thức thiết lập cho lớp phân số.

Lớp phân số có 2 thuộc tính: tử số và mẫu số.

* Phương thức thiết lập mặc định: tử số được mặc định là 0. Mẫu số mặc định là 1.
* Phương thức thiết lập khi biết tử số: tử số được gán giá trị tương ứng với giá trị của đối số đầu vào, mẫu số mặc định là 1.
* Phương thức thiết lập khi biết đầy đủ thông tin: tử số và mẫu số được gán giá trị tương ứng với giá trị của các đối số đầu vào.

Phương thức thiết lập sao chép: nhận tham số đầu vào là một đối tượng thuộc lớp phân số và tạo ra một đối tượng phân số mới giống hoàn toàn đối tượng phân số đầu vào.

public class PhanSo

{

//Các thuộc tính

private int Tu;

private int Mau;

// Phương thức thiêt lập mặc định

public PhanSo()

{

Tu = 0;

Mau = 1;

}

//‘ Phương thức thiêt lập khi biêt tử số

public PhanSo(int t)

{

Tu = t;

Mau = 1;

}

//‘ Phương thức thiết lập khi biết đẩy đủ thông tin

public PhanSo(int t, int m)

{

Tu = t;

Mau = m;

}

//‘ Phương thức thiết lập sao chép

public PhanSo(PhanSo ps)

{

Tu = ps.Tu;

Mau = ps.Mau;

}

public void Nhap()

{

Console.Write("Nhap tu: ");

Tu = Convert.ToInt32(Console.ReadLine());

Console.Write("Nhap mau: ");

Mau = Convert.ToInt32(Console.ReadLine());

}

public void Xuat()

{

Console.WriteLine(Tu + "/" + Mau);

}

}

* **Hàm main()**

static void Main(string[] args)

{

PhanSo a = new PhanSo();

a.Xuat();

PhanSo b = new PhanSo(1);

b.Xuat();

PhanSo c = new PhanSo(1, 2);

c.Xuat();

PhanSo d = new PhanSo(c);

d.Xuat();

}

## Phương Thức Hủy

### Mục đích

C# cũng cung cấp bộ thu dọn rác tự động nó sẽ ngầm hủy các biến khi không dùng. Tuy nhiên trong một số trường hợp ta cũng cần hủy tường minh, khi đó chỉ việc cài đặt phương thức Finalize(), phương thức này sẽ được gọi bởi bộ thu dọn rác. Ta không cần phải gọi phương thức này.

### Đặc điểm

- Phương thức hủy của lớp được định nghĩa thông qua toán tử Finalize.

- Không có giá trị trả về.

- Không có tham số đầu vào.

- Được tự động gọi thực hiện khi đối tượng hết phạm vi sử dụng.

- Phương thức hủy thuộc nhóm các phương thức xử lý.

- Có duy nhất một phương thức hủy trong 1 lớp mà thôi

## Kế thừa - Đa xạ

Phần trước đã trình bày cách xây dựng một chương trình bằng việc xây dựng các lớp đối tượng. Phần tiếp theo sẽ đi sâu vào mối quan hệ giữa những đối tượng trong thế giới thực và thể hiện những quan hệ đó trên máy tính. Quan hệ đặc biệt – tổng quát của thế giới thực được thể hiện thông qua khái niệm kế thừa trong hướng đối tượng. Còn khái niệm đa xạ thì cho phép các thể hiện của lớp có liên hệ với nhau được xử lý theo một cách tổng quát.

### Kế thừa

Trong thế giới thực, khi chúng ta nói rằng mèo là một loại động vật có vú, có nghĩa là mèo là một trường hợp đặc biệt của loại động vật có vú. Nó có tất cả các đặc tính của bất cứ động vật có vú nào (như sinh ra con, có sữa mẹ và có lông...). Tuy nhiên, mèo có thêm các đặc tính riêng được xác định trong họ nhà mèo mà các họ động vật có vú khác không có được. Chó cũng là loại động vật có vú, chó cũng có tất cả các thuộc tính của động vật có vú, và riêng nó còn có thêm các thuộc tính riêng xác định họ loài chó mà khác với các thuộc tính đặc biệt của loài khác ví dụ như mèo. Như vậy, chó và mèo là trường hợp đặc biệt của động vật có vú. Ngược lại động vật có vú là trường hợp tổng quát của các loài chó và mèo. Kế thừa là cách thể hiện chung nhất và tự nhiên nhất cho quan hệ đặc biệt – tổng quát.

**Động vật có vú**

**Động vật có vú**

**Chó**

**Mèo**

Chương trình minh họa:

using System;

public class Animal

{

private int Age;

private String Name;

public Animal( int Age, string Name)

{

this.Age = Age;

this.Name = Name;

}

public void Say()

{

Console.WriteLine(“I'm {0}, {1} years old”, Name, Age);

}

}

public class Cat: Animal

{

private string cWord;

public Cat(int Age, string Name, string word) : base(Age,Name)

{

cWord = word;

}

public new void Say()

{

base.Say();

Console.WriteLine(“ Cat say: {0}”, cWord);

}

}

public class Tester

{

public static void Main()

{

Animal a = new Animal(5, "Animal");

a.Say();

Cat c = new Cat(4,"Mimi", “Hello world!”);

C.Say();

}

}

//Ket qua:

// I'm Animal, 5 years old

// I'm Mimi, 4 years old

// Cat say: Hello world!

### Đa xạ (đa hình)

Tính đa hình (polymorphism), theo tiếng Anh từ này được kết hợp từ poly là nhiều và morph có nghĩa là form (hình thức). Do vậy, đa hình được hiểu như là khả năng sử dụng nhiều hình thức của một kiểu mà không cần phải quan tâm đến từng chi tiết.

Ví dụ trong thực tế, khi một tổng đài điện thoại gởi cho máy điện thoại của chúng ta một tín hiệu có cuộc gọi, tổng đài không quan tâm đến điện thoại của ta là loại nào. Có thể ta đang dùng một điện thoại cũ dùng motor để rung chuông, hay là một điện thoại điện tử phát ra tiếng nhạc số. Hoàn toàn các thông tin về điện thoại của ta không có ý nghĩa gì với tổng đài, tổng đài chỉ biết một kiểu cơ bản là điện thoại mà thôi và diện thoại này sẽ biết cách báo chuông. Còn việc báo chuông như thế nào thì tổng đài không quan tâm.

Tóm lại, tổng đài chỉ cần bảo điện thoại hãy làm điều gì đó để reng. Còn phần còn lại tức là cách thức riêng là tùy thuộc vào từng loại điện thoại. Đây chính là tính đa hình.

Chương trình minh họa

using System;

using System.Collections.Generic;

using System.Text;

namespace \_\_OOP\_polymorphism

{

    class Parent

    {

        public int methodA()

        {

            return methodB() \* methodC();

        }

        public virtual int methodB()

        {

            return 1;

        }

        public int methodC()

        {

            return 2;

        }

    }

    class Child : Parent

    {

        public override int methodB()

        {

            return 3;

        }

    }

    class Program

    {

        static void Main(string[] args)

        {

            Child objchild = new Child();

            Console.WriteLine("Result is " + objchild.methodA().ToString());

            Console.Read();

        }

    }

}

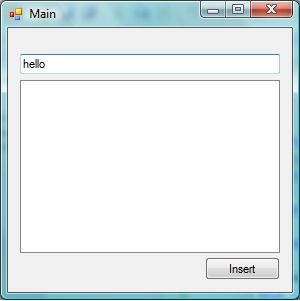
Kết quả chạy trương trình:  
6

Như bình thường của mô hình kế thừa, kết quả trả về khi gọi hàm methodA() từ đối tượng của lớp Child phải là “Result is 2”. Nhưng trong kết quả trên, kết quả là “Result is 6”. Kết quả này do hàm methodB() tại lớp Child đã override hàm methodB() tại lớp Parent.

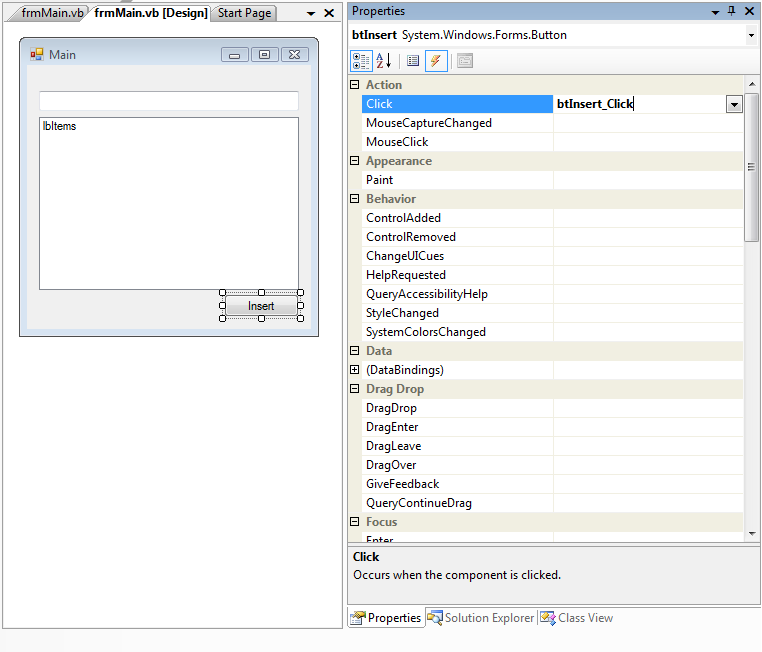
# Chương 7 Lập Trình Window Form với VB.Net

## Form

Cấu trúc một form đơn giản



## Button



Hàm xử lý sự kiện click của button

private void btnInsert\_Click(object sender, EventArgs e)

{

if(txtInput.Text != "")

if(lstbItems.Items.Contains(txtInput.Text))

MessageBox.Show("ListBox has already contained this item");

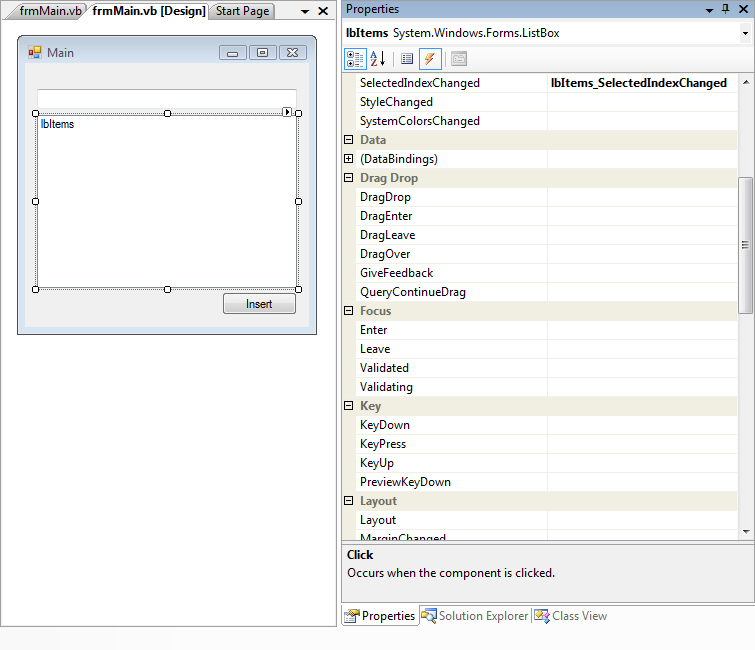
else

lstbItems.Items.Add(txtInput.Text);

else

MessageBox.Show("You must type item's name");

}



Hàm xử lý sự kiện thay đổi lựa chọn trên ListBox

private void lstbItems\_SelectedIndexChanged(object sender, EventArgs e)

{

if (lstbItems.SelectedIndex >= 0)

MessageBox.Show("You have selected item " + lstbItems.SelectedItem.ToString());

}

## TextBox, RichTextBox

## ListView, TreeView

# Chương 8 Cơ Sở Dữ Liệu với ADO.Net

## Kiến trúc ADO.Net

ADO.NET là một phần của .NET Framework, nó được xem là “bộ thư viện lớp” chịu trách nhiệm xử lý dữ liệu trong ngôn ngữ MS.NET. ADO.NET được thiết kế với dạng dữ liệu “ngắt kết nối”, nghĩa là chúng ta có thể lấy cả một cấu trúc phức tạp của dữ liệu từ database, sau đó ngắt kết nối với database rồi mới thực hiện các thao tác cần thiết. Đây là một sự tiến bộ về mặt thiết kế bởi vì thiết kế ADO trước đây luôn cần duy trì một kết nối trong quá trình thao tác dữ liệu.

Có thể coi ADO.NET là một thế hệ tiếp theo của ADO. ADO.NET kế thừa tất cả những ưu điểm của ADO, đồng thời với ý tưởng thiết kế hoàn toàn mới ADO.NET có một diện mạo khác hẳn so với tiền thân của nó. Một vài đặc điểm nổi bật của ADO.NET mà ADO không có như sau:

ADO.NET được thiết kế hoàn toàn dựa vào XML vì XML là chuẩn trao đổi dữ liệu tiến bộ và tốt nhất trên môi trường Internet hiện nay. ADO.NET được thiết kế hoàn toàn hướng đối tượng : đây là đặc điểm chi phối toàn bộ các sản phẩm Microsoft .NET.

## DataSet

Là thành phần chính cho đặc trưng kết nối không liên tục (ngắt kết nối) của kiến trúc ADO.NET.

DataSet được thiết kế để thích ứng với bất kì nguồn dữ liệu nào.

Nhiệm vụ của DataSet là nhận dữ liệu về từ DataAdapter và xử lý nó.

Lưu trữ dữ liệu của DataBase trong bộ nhớ.

Mọi thao tác thay đổi dữ liệu được thực hiện trên DataSet, không làm ảnh hưởng đến DataBase.

Theo vết các thay đổi trên dữ liệu và có thể cập nhật dữ liệu ngược vào DataBase thông qua SqlDataAdapter.

Gồm các đối tượng : DataTable, DataRelationship, Constraint.

Việc sử dụng DataSet là một tiến bộ lớn của kiến trúc ADO.NET tuy nhiên với các ứng dụng Web việc sử dụng DataSet không được khuyến khích vì đối tượng DataSet được xem là quá lớn, nặng nề khó thích hợp cho đường truyền trên web vốn rất hạn chế.

### Các thuộc tính của DataSet

|  |  |  |
| --- | --- | --- |
| STT | Tên Thuộc Tính | Ý Nghĩa |
| 1 | DataSetName | Tên của dataset |
| 2 | Relations |  |
| 3 | Tables |  |

### Các phương thức chính của DataSet

|  |  |  |
| --- | --- | --- |
| STT | Tên Phương Thức | Ý Nghĩa |
| 1 | GetChange |  |
| 2 | RejectChanges |  |
| 3 | AcceptChanges |  |
| 4 | GetXML |  |
| 5 | ReadXML |  |
| 6 | WriteXML |  |

## SQL Server Data Provider

Các lớp chính của SQL Server Data Provider

|  |  |
| --- | --- |
| Tên Lớp | Ý Nghĩa |
| SqlCommand | Thực thi SQL queries, câu lệnh hoặc lưu trữ thủ tục |
| SqlConnection | Tạo kết nối tới SQL Server |
| SqlDataAdapter | Cầu nối trung gian giữa dataset và data source |
| SqlReader | Cung cấp một data stream tới kết quả |
| SqlError | Lưu trữ thông tin về lỗi và cảnh cáo (warning) |
| SqlException | Các ngoại lệ trong trường hợp SQL Server lỗi và cảnh báo |
| SqlParameter | Tham sô biên command |
| SqlTransaction | Transaction của SQL Server |

### Lớp đối tượng SqlConnection

Dùng để tạo kết nối đến các CSDL Sql Server

#### Các thuộc tính của SqlConnection

|  |  |  |
| --- | --- | --- |
| STT | Tên Thuộc Tính | Ý Nghĩa |
| 1 | ConnectionString | Chuỗi kết nối database |
| 2 | ConnectionTimeout | Thời gian chờ trước khi ngắt kết nối với database |
| 3 | Container | Icontainer chứa các Component |
| 4 | Database | Tên của database hiện tại sau khi kết nối |
| 5 | DataSource | Tên của server, tên file chứa dữ liệu |
| 6 | Provider | Tên của OLEDB provider |
| 7 | ServerVersion | Version của server |
| 8 | Site | Isite của Component |
| 9 | State | Trạng thái của liên kết |

#### Các phương thức chính của SqlConnection

|  |  |  |
| --- | --- | --- |
| STT | Tên Phương Thức | Ỳ Nghĩa |
| 1 | SqlConnection | Phương thức khởi tạo |
| 2 | Open | Mở kết nối tới database |
| 3 | Close | Đóng kết nối với database |
| 4 | Dispose | Hủy đối tượng |
| 5 | BeginTransaction | Bắt đầu 1 transaction |
| 6 | Commit | Kết thúc 1 transaction |

Cách tạo một kết nối:

Cách 1:

* Tạo chuỗi kết nối
* Tạo đối tượng kết nối SqlConnection, truyền tham số chuỗi kết nối vào.

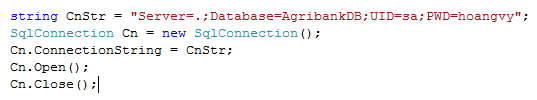
Ví dụ:

D:\i84\Research\Soan ADO.NET\Cn.png

Cách 2:

* Tạo chuỗi kết nối
* Tạo đối tượng kết nối SqlConnection, không truyền tham số
* Trỏ thuộc tính ConnectionString của đối tượng SqlConnection đến chuỗi kết nối.

Ví dụ:



Các tham số của SqlConnection

|  |  |
| --- | --- |
| Tham số | Mô tả |
| Data Source | Tên máy hoặc IP |
| Initial Catalog | Tên Database |
| Integrated Security | Sử dụng SSPI |
| User ID | Tên User kết nối |
| Password | Mật khẩu kết nối |

Lưu ý:

* Windows Authenticate: có nghĩa là sử dụng quyền hạn của Windows Account để truy cập SQl Server. Khi kết nối ở mode này, thì không cần truyền userID, password.
* Để kết nối bằng Window Authencation, trong chuỗi kết nối cho thêm thuộc tính: "Integrated Security=true" hoặc là: "Integrated Security=SSPI"
* Ví dụ:

Data Source=localhost;Initial Catalog=myDB;Integrated Security=SSPI;

* Sql server account: truyền thông tin account (username, password) vào chuỗi kết nối
* Ví dụ:

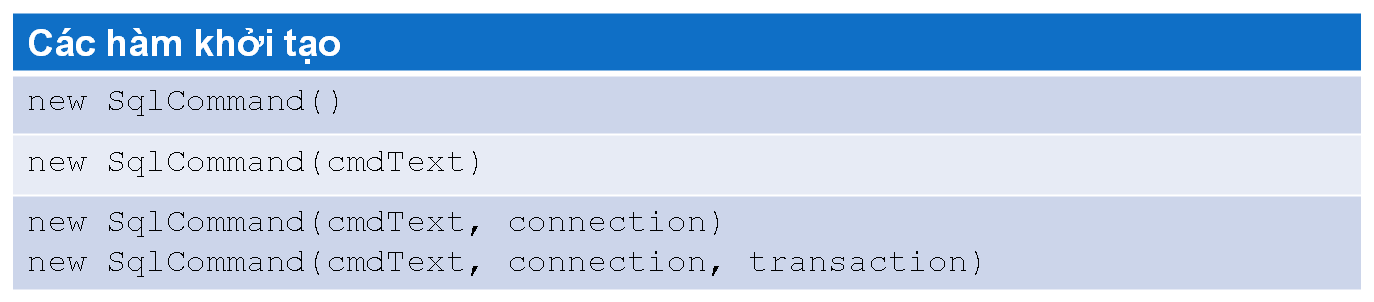
string CnStr = "Server=.;Database=TestDB;UID=sa;PWD=sa";

### Lớp đối tượng SqlCommand

Đối tượng thực hiện các câu lệnh tương tác truy vấn, rút trích dữ liệu từ database khi đã thiết lập kết nối tới dữ liệu và trả về kết quả.

Kết quả trả về được lưu trữ dưới dạng luồng thông qua 2 đối tượng :

* DataReader
* DataSet thông qua một đối tượng SqlDataAdapter



Ví dụ 1:

String sql = “Select \* from SinhVien”

SqlCommand cmd = new SqlCommand();

Cmd.CommandText = sql;

Ví dụ 2:

String sql = “Select \* from SinhVien”

SqlCommand cmd = new SqlCommand(sql);

Ví dụ 3:

String CnStr = “Server=.;Database=”SinhVienDB”;uid=sa;pwd=sa;”;

SqlConnection cn = new SqlConnection(CnStr);

String sql = “Select \* from SinhVien”

SqlCommand cmd = new SqlCommand(sql,cn);

Ví dụ 4:

String CnStr = “Server=.;Database=”SinhVienDB”;uid=sa;pwd=sa;”;

SqlConnection cn = new SqlConnection(CnStr);

String sql = “Select \* from SinhVien”

SqlCommand cmd = new SqlCommand(sql,cn,null);

#### Các thuộc tính của SqlCommand

|  |  |  |
| --- | --- | --- |
| STT | Tên Thuộc Tính | Ý Nghĩa |
| 1 | CommandText | Câu lệnh SQL hay stored procedure kết nối data source |
| 2 | CommandTimeout | Thời gian chờ trước khi ngắt kết nối |
| 3 | CommandType | Giá trị mô tả hoạt động của CommandText |
| 4 | Connection | Thiết lập SqlConnection |
| 5 | Container | IContainer chứa Component |
| 6 | DesignTimeVisible | Giá trị mô tả đối tượng command xuất hiện trong Designer |
| 7 | Parameters | Lấy các tham số |
| 8 | Site | ISite của Component |
| 9 | Transaction | SqlTransaction khi SqlCommand thực thi |
| 10 | UpdatedRowSource | Update DataRow |

#### Các phương thức chính của SqlCommand

|  |  |  |
| --- | --- | --- |
| STT | Tên Phương Thức | Ỳ Nghĩa |
| 1 | SqlCommand | Phương thức khởi tạo |
| 2 | ExecuteReader | Gửi CommandText đến Kết nối để tạo ra SqlDataReader |
| 3 | ExecuteNonQuery | Trả về số lượng dòng bị ảnh hưởng trên CSDL |
| 4 | ExecuteScalar | Trả về 1 giá trị đầu tiên (VD: giá trị tính tổng) |
| 5 | ExecuteXMLReader | Trả về 1 XMLReader |
| 6 | Dispose | Hủy đối tượng |

### Lớp đối tượng SqlParameter

Cần thực hiện câu lệnh nhiều lần với các giá trị khác nhau.

Đối tượng tham số truyền vào cho đối tượng SqlCommand

Có các thuộc tính sau :

|  |  |  |
| --- | --- | --- |
| STT | Tên Thuộc Tính | Ý Nghĩa |
| 1 | ParameterName | Tên tham số |
| 2 | SqlDbType | Kiểu dữ liệu của tham số tương ứng với kiểu dữ liệu của SqlServer |
| 3 | Direction | Input, Output, InputOutput, ReturnValue, … |
| 4 | Size | Kích thước tối đa của dữ liệu |
| 5 | Value | Giá trị của tham số (input / Output) |

Để truyền tham số cho command:

* Khai báo đối tượng command với tham số:

SqlCommand cmd = new SqlCommand ( "select \* from Sach where SachID = @SachID ", conn);

* Định nghĩa các tham số được dùng ở đối tượng command:

SqlParameter param = new SqlParameter();

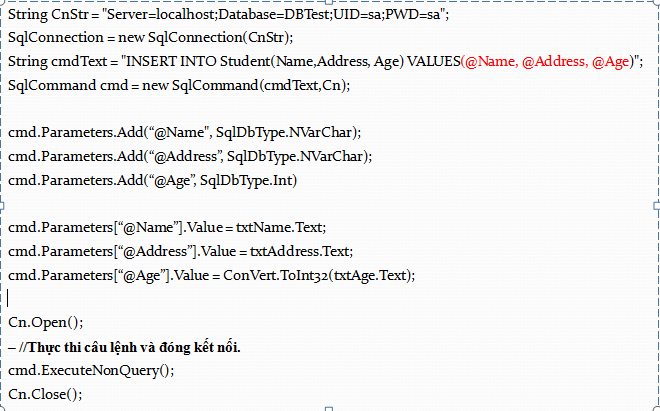
param.ParameterName = "@SachID ";

param.Value = sachID;

* Thêm mới tham số vào đối tượng command:

cmd.Parameters.Add(param);

Ví dụ: truyền nhiều tham số

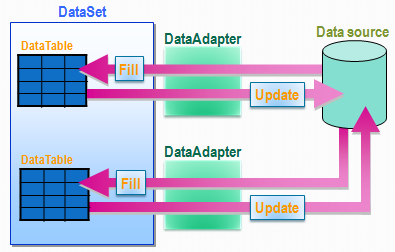


### Lớp đối tượng SqlDataAdapter

Đây là đối tượng rất quan trọng của ADO.NET, nó là cầu nối trung gian của database và dataset (dataset là đối tượng ngắt kết nối), bởi vì đối tượng “ngắt kết nối” dataset không thể liên lạc trực tiếp với database nên nó cần một đối tượng trung gian lấy dữ liệu từ database cho nó

Cung cấp các phương thức và thuộc tính để lấy và lưu dữ liệu giữa DataSet và CSDL

Sử dụng DataSet để lưu trữ dữ liệu, đồng thời, cho cập nhật dữ liệu ngược lại vào Database



Các phương thức chính :

|  |  |  |
| --- | --- | --- |
| STT | Tên Phương Thức | Ý Nghĩa |
| 1 | Fill | Lấy dữ liệu từ data source |
| 2 | Update | Cập nhật dữ liệu vào data source |

Sử dụng Stored Procedure để truy xuất dữ liệu:

* Trước đây khi dùng Query Analyzer chúng ta có thể đặt tên và save các nhóm câu lệnh SQL vào một file dưới dạng script để có thể sử dụng trở lại sau này. Tuy nhiên thay vì save vào text file ta có thể save vào trong SQL Server dưới dạng Stored Procedure.
* Stored Procedure là một nhóm câu lệnh Transact-SQL đã được compiled (biên dịch) và chứa trong SQL Server dưới một tên nào đó và được xử lý như một đơn vị (chứ không phải nhiều câu SQL riêng lẻ).

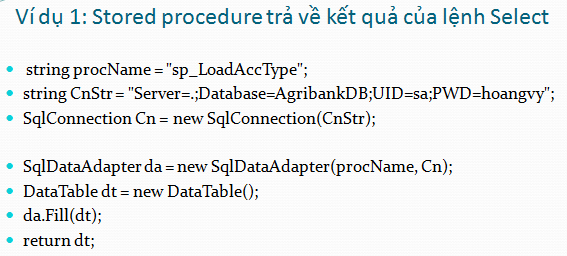
Ưu điểm:

* Khi thực thi một câu lệnh SQL thì SQL Server phải kiểm tra permission xem user gởi câu lệnh đó có được phép thực hiện câu lệnh hay không đồng thời kiểm tra cú pháp rồi mới tạo ra một execute plan và thực thi.
* Nếu có nhiều câu lệnh như vậy gởi qua network có thể làm giảm đi tốc độ làm việc của server. SQL Server sẽ làm việc hiệu quả hơn nếu dùng stored procedure vì người gởi chỉ gởi một câu lệnh đơn và SQL Server chỉ kiểm tra một lần sau đó tạo ra một execute plan và thực thi.
* Nếu stored procedure được gọi nhiều lần thì execute plan có thể được sử dụng lại nên sẽ làm việc nhanh hơn. Ngoài ra cú pháp của các câu lệnh SQL đã được SQL Sever kiểm tra trước khi save nên nó không cần kiểm lại khi thực thi.
* Một khi stored procedure được tạo ra nó có thể được sử dụng lại. Ðiều này sẽ làm cho việc bảo trì (maintainability) dễ dàng hơn do việc tách rời giữa business rules (tức là những logic thể hiện bên trong stored procedure) và database.
* Ví dụ nếu có một sự thay đổi nào đó về mặt logic thì ta chỉ việc thay đổi code bên trong stored procedure mà thôi. Những ứng dụng dùng stored procedure này có thể sẽ không cần phải thay đổi mà vẫn tương thích với business rule mới.
* Cũng giống như các ngôn ngữ lập trình khác stored procedure cho phép ta đưa vào các input parameters (tham số) và trả về các output parameters đồng thời nó cũng có khả năng gọi các stored procedure khác.
* Security : Giả sử chúng ta muốn giới hạn việc truy xuất dữ liệu trực tiếp của một user nào đó vào một số tables, ta có thể viết một stored procedure để truy xuất dữ liệu và chỉ cho phép user đó được sử dụng stored procedure đã viết sẵn mà thôi chứ không thể "đụng" đến các tables đó một cách trực tiếp. Ngoài ra stored procedure có thể được encrypt (mã hóa) để tăng cường tính bảo mật.

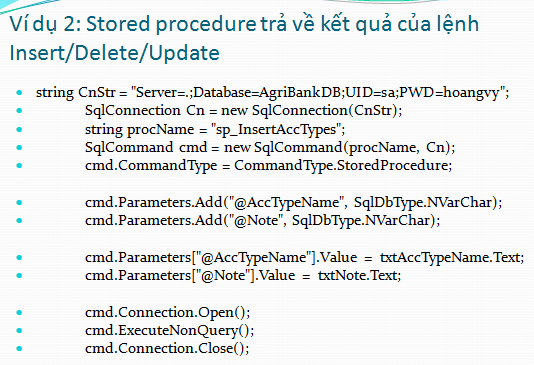
Cách dùng:

* Tạo stored procedure trong sqlServer
* Tạo Sqlcommand với giá trị command text là tên Stored procedure
  + SqlCommand cmd = new SqlCommand(“Ten stored”,cn)
  + Gán commandType:
  + cmd.CommandType = CommandType.StoredProcedure
  + Truyền các tham số vào command nếu StoredProcedure có yêu cầu
  + Thực thi command

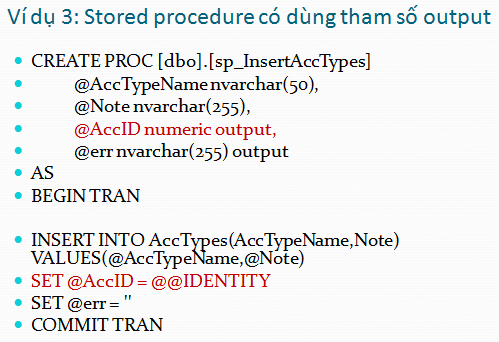
Ví dụ 1:



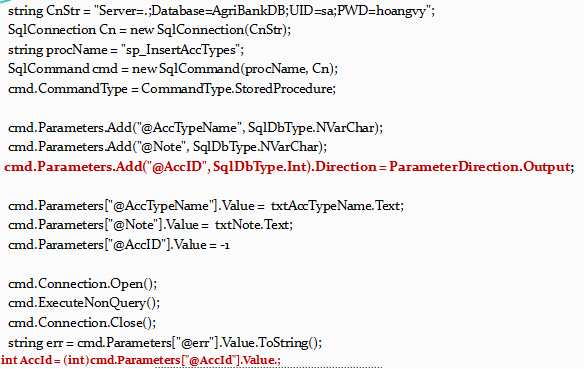
Ví dụ 2:



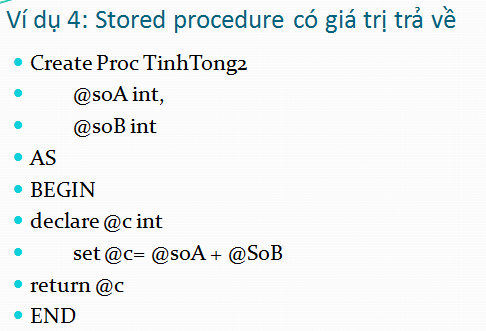
Ví dụ 3:



Gọi stored procedure và có truyền tham số:



Ví dụ 4:





## OleDb Data Provider

Các lớp chính trong OleDb Data Provider

|  |  |
| --- | --- |
| Tên Lớp | Ý Nghĩa |
| OleDbCommand | Thực thi SQL queries, câu lệnh hoặc lưu trữ thủ tục |
| OleDbConnection | Tạo kết nối tới data source |
| OleDbDataAdapter | Cầu nối trung gian giữa dataset và data source |
| OleDbError | Lưu trữ thông tin về lỗi và cảnh cáo (warning) |
| OleDbException | Các ngoại lệ trong trường hợp SQL Server lỗi và cảnh báo |
| OleDbParameter | Tham sô biên command |
| OleDbTransaction | Transaction của data source |

### Lớp đối tượng OleDbConnection

Lớp đối tượng OleDbConnection là một lớp đối tượng dùng để kết nối cơ sở dữ liệu.

#### Các thuộc tình của OleDbConnection

|  |  |  |
| --- | --- | --- |
| STT | Tên Thuộc Tính | Ý Nghĩa |
| 1 | ConnectionString | Chuỗi kết nối database |
| 2 | ConnectionTimeout | Thời gian chờ trước khi ngắt kết nối với database |
| 3 | Container | Icontainer chứa các Component |
| 4 | Database | Tên của database hiện tại sau khi kết nối |
| 5 | DataSource | Tên của server, tên file chứa dữ liệu |
| 6 | Provider | Tên của OLEDB provider |
| 7 | ServerVersion | Version của server |
| 8 | Site | Isite của Component |
| 9 | State | Trạng thái của liên kết |

#### Các phương thức chính của OleDbConnection

|  |  |  |
| --- | --- | --- |
| STT | Tên Phương Thức | Ỳ Nghĩa |
| 1 | OleDbConnection | Phương thức khởi tạo |
| 2 | Open | Mở kết nối tới database |
| 3 | Close | Đóng kết nối với database |
| 4 | Dispose | Hủy đối tượng |

### Lớp đối tượng OleDbCommand

Lớp đối tượng OleDbCommand thực hiện các câu truy vấn SQL hoặc stored procedure trên một data source.

#### Các thuộc tính của OleDbCommand

|  |  |  |
| --- | --- | --- |
| STT | Tên Thuộc Tính | Ý Nghĩa |
| 1 | CommandText | Câu lệnh SQL hay stored procedure kết nối data source |
| 2 | CommandTimeout | Thời gian chờ trước khi ngắt kết nối |
| 3 | CommandType | Giá trị mô tả hoạt động của CommandText |
| 4 | Connection | Thiết lập OleDbConnection |
| 5 | Container | IContainer chứa Component |
| 6 | DesignTimeVisible | Giá trị mô tả đối tượng command xuất hiện trong Designer |
| 7 | Parameters | Lấy các tham số |
| 8 | Site | ISite của Component |
| 9 | Transaction | OleDBTransaction khi OleDbCommand thực thi |
| 10 | UpdatedRowSource | Update DataRow |

#### Các phương thức của OleDbCommand

|  |  |  |
| --- | --- | --- |
| STT | Tên Phương Thức | Ỳ Nghĩa |
| 1 | OleDbCommand | Phương thức khởi tạo |
| 2 | ExecuteReader | Gửi CommandText đến Kết nối để tạo ra SqlDataReader |
| 3 | ExecuteNonQuery | Trả về số lượng dòng bị ảnh hưởng trên CSDL |
| 4 | ExecuteScalar | Trả về 1 giá trị đầu tiên (VD: giá trị tính tổng) |
| 3 | Dispose | Hủy đối tượng |

### Lớp đối tượng OleDbParameter

|  |  |  |
| --- | --- | --- |
| STT | Tên Thuộc Tính | Ý Nghĩa |
| 1 | ParameterName | Tên tham số |
| 2 | OleDbType | Kiểu dữ liệu của tham số tương ứng với kiểu dữ liệu của data source |
| 3 | Direction | Input, Output, InputOutput, ReturnValue, … |
| 4 | Size | Kích thước tối đa của dữ liệu |
| 5 | Value | Giá trị của tham số (input / Output) |

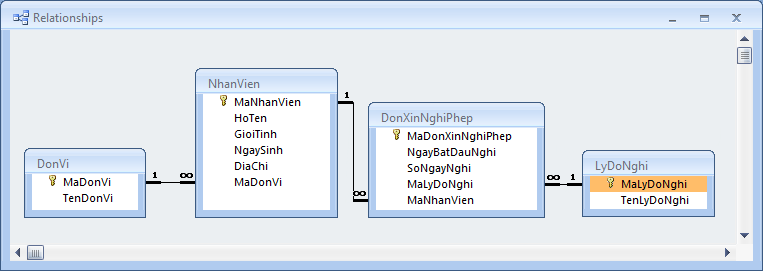
### Lớp đối tượng OleDbDataAdapter

|  |  |  |
| --- | --- | --- |
| STT | Tên Phương Thức | Ý Nghĩa |
| 1 | Fill | Lấy dữ liệu từ data source |
| 2 | Update | Cập nhật dữ liệu vào data source |

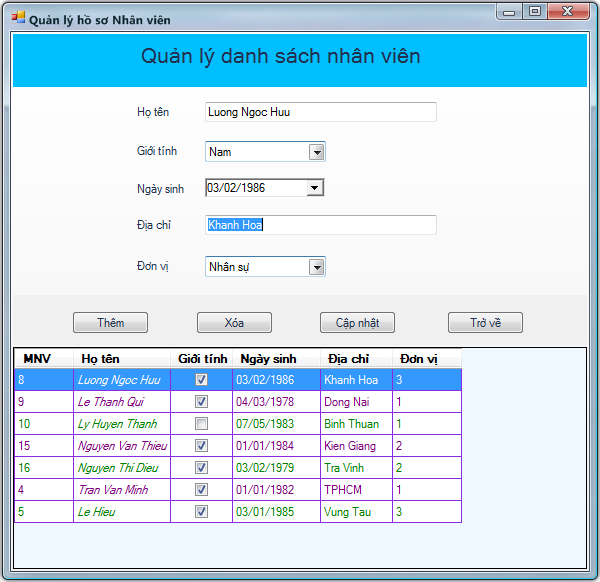
## Ví dụ về truy suất cơ sở dữ liệu

### Mô tả chương trình

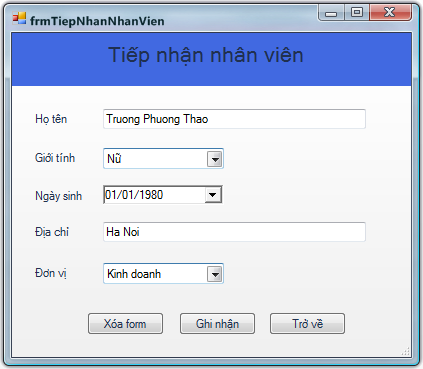
Sơ đồ cơ sở dữ liệu Nhân Viên



Màn hình Quản lý danh sách nhân viên



Màn hình tiếp nhận nhân viên



### Lớp DataServices đảm nhận công việc thao tác đến cơ sở dữ liệu

using System.Data.OleDb;

using System.Data;

public class DataServices

{

OleDbConnection m\_Connection;

public OleDbDataAdapter m\_DataAdapter = new OleDbDataAdapter();

OleDbCommand m\_Command = new OleDbCommand();

public DataServices()

{

string strConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=db.mdb";

m\_Connection = new OleDbConnection(strConnectionString);

}

public void OpenConnection()

{

m\_Connection.Open();

}

public void CloseConnection()

{

m\_Connection.Close();

}

public DataTable ExecuteQuery(string sql)

{

DataTable dt = new DataTable();

m\_DataAdapter = new OleDbDataAdapter(sql, m\_Connection);

m\_DataAdapter.Fill(dt);

if ((dt.Rows.Count == 0))

{

return null;

}

return dt;

}

public void ExecuteNonQuery(string sql)

{

OpenConnection();

m\_Command = new OleDbCommand(sql, m\_Connection);

m\_Command.CommandType = CommandType.Text;

m\_Command.ExecuteNonQuery();

CloseConnection();

}

}

### Xử lý thao tác ghi nhận nhân viên mới

private void btnGhiNhan\_Click(System.Object sender, System.EventArgs e)

{

NhanVien nv = new NhanVien();

nv.HoTen = txtHoTen.Text;

nv.DiaChi = txtDiaChi.Text;

nv.GioiTinh = true;

if ((cmbGioiTinh.SelectedIndex == 1))

{

nv.GioiTinh = false;

}

nv.MaDonVi = int.Parse(cmbDonVi.SelectedValue.ToString());

nv.NgaySinh = dtpNgaySinh.Value;

if ((nv.KiemTraTuoiHopLe() == false))

{

MessageBox.Show("Tuổi nhân viên không hợp lệ");

return;

}

nv.ThemNhanVien();

MessageBox.Show("Tiếp nhận nhân viên mới thành công");

}

public void ThemNhanVien()

{

string sql = null;

sql = "Insert into NhanVien(HoTen,GioiTinh,NgaySinh,DiaChi,MaDonVi) values (";

sql += "'" + m\_HoTen + "',";

sql += m\_GioiTinh.ToString() + ",";

sql += "'" + m\_NgaySinh + "',";

sql += "'" + m\_DiaChi + "',";

sql += m\_MaDonVi.ToString() + ")";

DataServices ds = new DataServices();

ds.ExecuteNonQuery(sql);

}

### Xử lý thao tác cập nhật thông tin nhân viên cũ

private void btnCapNhat\_Click(System.Object sender, System.EventArgs e)

{

if (dgvNhanVien.SelectedRows.Count > 0) {

int maNhanVien = dgvNhanVien.SelectedRows(0).Cells(0).Value;

NhanVien nv = new NhanVien();

nv.HoTen = txtHoTen.Text;

nv.DiaChi = txtDiaChi.Text;

nv.GioiTinh = true;

if ((cmbGioiTinh.SelectedIndex == 1)) {

nv.GioiTinh = false;

}

nv.MaDonVi = int.Parse(cmbDonVi.SelectedValue.ToString());

nv.NgaySinh = dtpNgaySinh.Value;

if ((nv.KiemTraTuoiHopLe() == false)) {

MessageBox.Show("Tuổi nhân viên không hợp lệ", "Quản lý nhân viên");

return;

}

nv.CapNhat(maNhanVien);

MessageBox.Show("Cập nhật thông tin nhân viên thành công", "Quản lý nhân viên");

LoadData();

//

}

}

public void CapNhat(int maNhanVien)

{

string sql = null;

sql = "Update NhanVien Set HoTen='" + m\_HoTen + "',";

sql += "GioiTinh=" + m\_GioiTinh.ToString() + ",";

sql += "NgaySinh='" + m\_NgaySinh.ToString() + "',";

sql += "DiaChi='" + m\_DiaChi + "',";

sql += "MaDonVi=" + m\_MaDonVi.ToString();

sql += " Where MaNhanVien = " + maNhanVien.ToString();

DataServices ds = new DataServices();

ds.ExecuteNonQuery(sql);

}