Nhập môn DotNet.

Bài 1

Yêu cầu

- Biết được sự ra đời, quá trình phát triển, tương lai cũng như cấu trúc của bộ sản phẩm .Net
- Nắm được các khái niệm cơ bản trong C# như: cấu trúc chương trình, từ khóa, các toán tử...
- Biết cách sử dụng một số kiểu dữ liệu cơ bản: mảng, xâu ký tự, liệt kê, struct.

Giới thiệu Microsoft.Net

- Cùng với sự phát triển liên tục của CNTT nhất là phần mềm, hệ điều hành, các môi trường phát triển phần mềm→ các ứng dụng liên tục được ra đời, tuy nhiên phát triển chưa được thống nhất.
- Java ra đời đã có sức mạnh đáng kể, đặc biệt với các ứng dụng trên Internet.

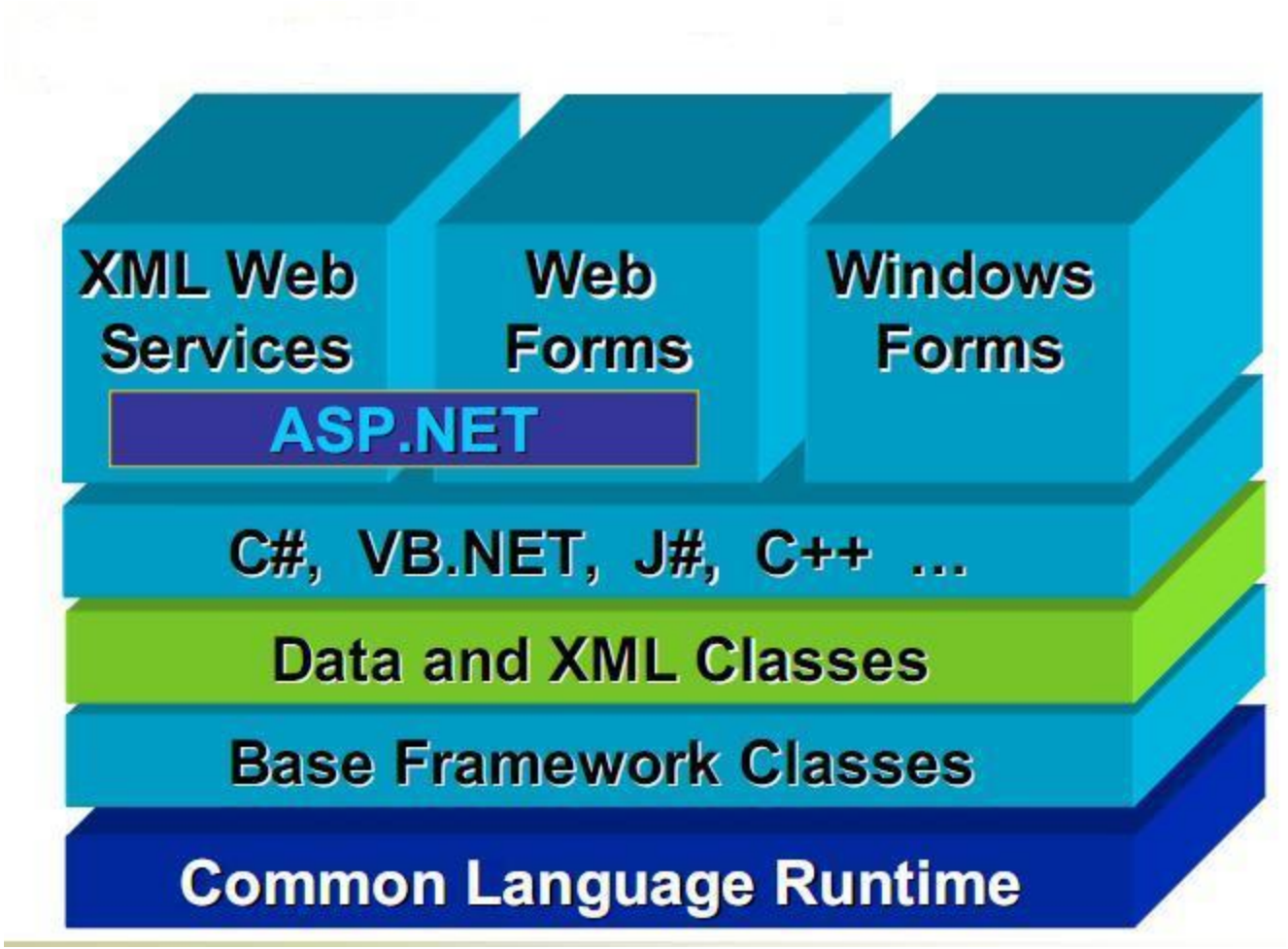
Giới thiệu Microsoft.Net

- Microsoft đã dùng ASP để làm giảm ảnh hưởng của JAVA. Ngoài ra khi làm việc trên môi trường web còn một số ngôn ngữ như CGI-Perl, PHP
- Lập trình ứng dụng cũng có nhiều công cụ: Visual C++, Delphi, Visual Basic…
- Tháng 7/2000 Microsoft phát hành phiên bản beta của .Net

Cấu trúc Microsoft.Net

- Microsoft.Net bao gồm hai phần chính: Framework và Integrated Development Environment (IDE).
- Thành phần Framework là tinh hoa, nền tảng của .Net, còn IDE chỉ là môi trường để triển khai các ứng dụng. Trong .Net toàn bộ các ngôn ngữ C#, Visual C++, Visual Basic.Net đều dùng cùng một IDE.

Microsoft.Net Framework



Microsoft.Net Framework

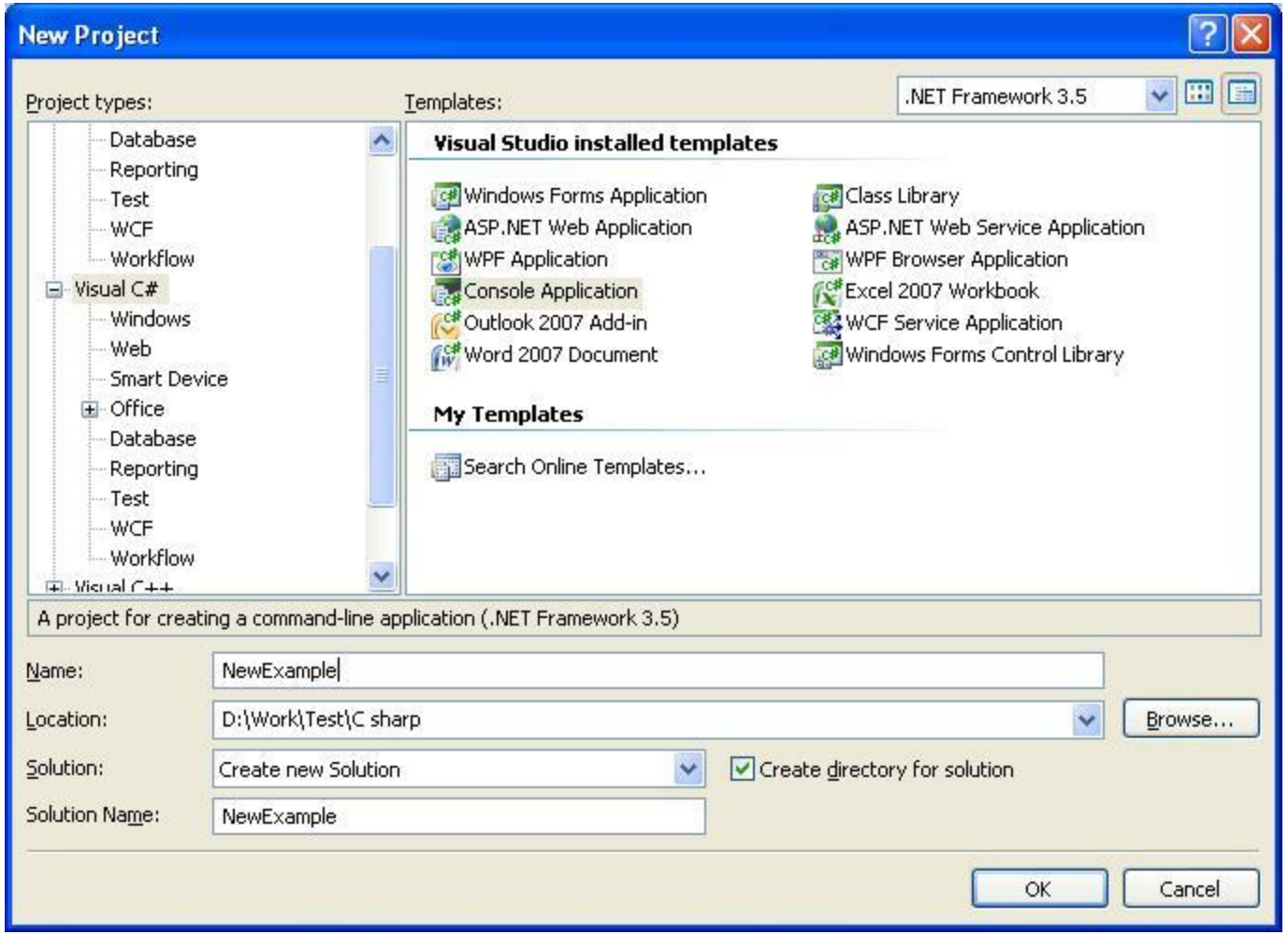
- Thành phần quan trọng nhất của .Net Framework là CLR-Common Language Runtime, cung cấp môi trường cho ứng dụng thực thi.
- .Net hỗ trợ tích hợp ngôn ngữ, có thể kế thừa các lớp, bắt các biệt lệ, đa hình thông qua nhiều ngôn ngữ. .Net framework sử dụng đặc tả Common Type System-CTS (hệ thống kiểu chung) và Common Language Specification-CLS (đặc tả ngôn ngữ chung).

Ngôn ngữ trung gian MSIL

- Với .Net chương trình không biên dịch thành tập tin thực thi mà biên dịch thành ngôn ngữ trung gian MSIL-Microsoft Intermediate Language. Sau đó chúng được CLR thực thi. Các tập tin MSIL biên dịch từ C# đồng nhất với các tập tin MSIL biên dịch từ ngôn ngữ .Net khác.
- Khi chạy chương trình IL được biên dịch (hay thông dịch) một lần nữa bằng trình Just In Time-JIT→mã máy.

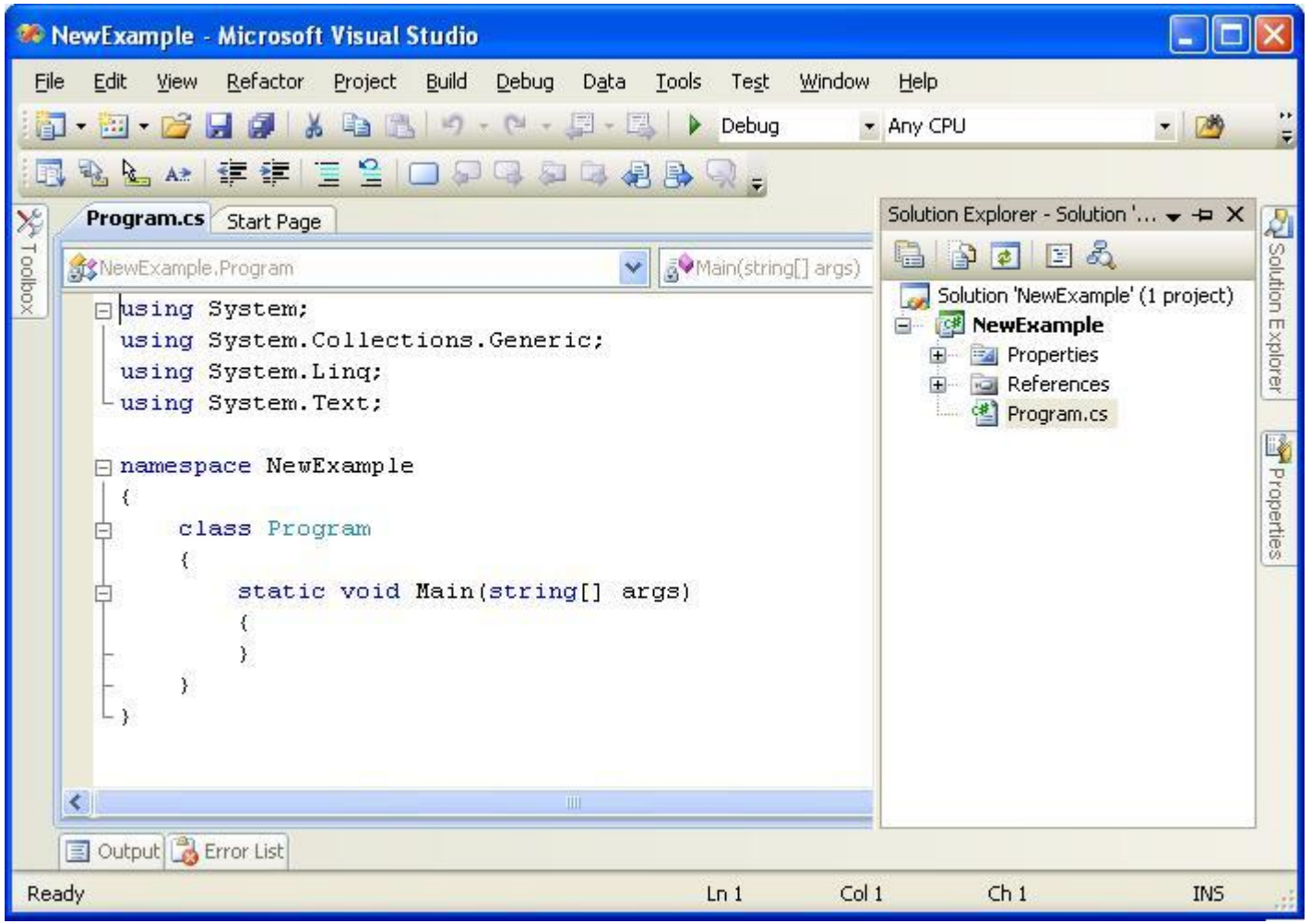
Visual Studio.Net

Tạo một Project mới



Visual Studio.Net

Môi trường làm việc



Ngôn ngữ C#

C# là một ngôn ngữ rất đơn giản, được kế thừa có chọn lọc, phát triển từ C++ và Java, với khoảng 80 từ khóa và hơn 10 kiểu dữ liệu dựng sẵn. C# hỗ trợ lập trình có cấu trúc, lập trình hướng đối tượng và hướng thành phần (component oriented).

Cấu trúc chương trình C#

```
using System;
namespace example
{
class Program
    {
        static void Main(string[] args)
        {
            //In ra man hinh
            Console.WriteLine("Lap trinh C#");
            Console.ReadLine();
        }
    }
}
```

Cấu trúc chương trình C#

- Lớp, đối tượng: Bản chất của lập trình HĐT là tạo ra các kiểu mới. Giống như các ngôn ngữ HĐT khác một kiểu trong C# cũng định nghĩa bằng từ khóa class (gọi là lớp). Thể hiện của một lớp gọi là đối tượng.
- Hành vi của lớp được gọi là các phương thức thành viên
- Ghi chú: tron C# sử dụng hai kiểu ghi chú quen thuộc là “//” và “/*…*/”

namespace (không gian tên)

- Console là một lớp trong cả ngàn lớp của bộ thư viện .Net.
- Một vùng tên có thể có nhiều lớp và vùng tên con khác. Nếu vùng tên A nằm trong vùng tên B, ta nói vùng tên A là vùng tên con của vùng tên B, khi đó các lớp trong vùng tên A được ghi như sau: B.A.ten_lop_trong_vung_ten_A
- System là vùng tên chứa nhiều lớp hữu ích cho việc giao tiếp với hệ thống hoặc các lớp công dụng chung như Console, Math, Exception...

Toán tử chấm “.”

Toán tử chấm dùng để truy xuất dữ liệu và phương thức của một lớp, đồng thời cũng để chỉ định tên lớp trong một vùng tên

Ví dụ: System.Console

Toán tử chấm cũng dùng để truy xuất các vùng tên con của một vùng tên

Ví dụ: VungTen.VungTenCon.VungTenConCon

Biên dịch, thực thi chương trình

Có hai cách biên dịch, thực thi chương trình đó là dùng môi trường phát triển tích hợp IDE (Visual Studio) hoặc viết bằng trình soạn thảo văn bản và dịch bằng dòng lệnh

Sau khi đầy đủ mã nguồn ta tiến hành biên dịch: nhấn “Crtl+Shift+B” hoặc Build>Build Solution. Nhấn F5 để bắt đầu chạy chương trình.

Kiểu dữ liệu định sẵn

Kiểu	Kích thước(byte)	Kiêu .Net	Giá trị
byte	1	Byte	0..255
char	1	Char	Mã Unicode
bool	1	Bollean	True hoặc False
sbyte	1	Sbyte	-128..127
short	2	Int16	-32768...32767
ushort	2	Uint16	0..65535
int	4	Int32	-2147483647..
float	4	Single	Số thực

Chuyển đổi kiểu định sẵn

Một đối tượng có thể chuyển từ kiểu này sang kiểu kia một cách ngầm định hoặc tường minh

```
short x=3;
int y;
y=x;
x=(short) y;
```

Biến và hằng

Khi dùng một biến trước hết nó phải được khởi tạo

```
int x;
x=5;
int y=x;
```

Hằng là một biến nhưng giá trị không thay đổi theo thời gian.

```
const int hs=10;
```

Vào ra dữ liệu

```
int n;
    Console.Write("Nhap gia tri n:");
    n = Convert.ToInt32(Console.ReadLine());
    n = int.Parse(Console.ReadLine());
```

Các toán tử

Nhóm toán tử	Toán tử	Ý nghĩa
Toán học	+ - * / %	cộng , trừ, nhân chia, lấy phần dư
Logic	& | ^ ! ~ && || true false	phép toán logic và thao tác trên bit
Ghép chuỗi	+	ghép nối 2 chuỗi
Tăng, giảm	++, --	tăng / giảm toán hạng lên / xuống 1. Đứng trước hoặc sau toán hạng.
Dịch bit	<< >>	dịch trái, dịch phải
Quan hệ	== != < > <= >=	bằng, khác, nhỏ/lớn hơn, nhỏ/lớn hơn hoặc bằng
Gán	= += -= *= /= %= &= |= ^= <<= >>=	phép gán
Chỉ số	[]	cách truy xuất phần tử của mảng
Ép kiểu	()	
Indirection và Address	* -> [] &	dùng cho con trỏ

Các toán tử

n++, ++n là các toán tử tăng giá trị giá trị của n được tăng thêm 1. Sự khác nhau chỉ thể hiện rõ trong các biêu thức toán học.

Toán tử tam phân:

<Biểu thức đk>?<biểu thức1>:<biểu thức2>;

Các lệnh rẽ nhánh không điều kiện

Có hai loại câu lệnh rẽ nhánh không điều kiện: lệnh gọi phương thức và sử dụng từ khóa goto, break, continue, return.

Các lệnh rẽ nhánh có điều kiện

Các từ khóa if-else, while, do-while, for, switch-case dùng để điều khiển dòng chảy của chương trình

```
if(biểu thức logic)
        khối lệnh ;


if(biểu thức logic)
        khối lệnh 1;
else khối lệnh 2;
```

Các lệnh rẽ nhánh có điều kiện

```
switch(biểu thức lựa chọn)
{
   case biểu_thức_hằng:
        khối lệnh;
        lệnh nhảy;
   [default:
        khối lệnh;
        lệnh nhảy;]
}
```

Lệnh lặp

- **Lệnh goto** có thể dùng để tạo vòng lặp có điều kiện, cụ thể: tạo nhãn và goto đến nhãn đó
- **Lệnh while**

  Cú pháp:

  while (biểu_thức_logic)

  khối lệnh;

Lệnh lặp

- **Lệnh do-while**

  do khối lệnh;

  while (biểu_thức_logic)

- **Vòng lặp for**

  for ([khởi_tạo_biến_đếm];[biểu_thức];[tăng_biến_đếm])

  khối lệnh;

Các kiểu dữ liệu cơ bản

Trong C# cũng hỗ trợ các kiểu dữ liệu như: mảng, chuỗi, liệt kê, cấu trúc giống như trong C++ và Java

- Kiểu mảng:
  - kiểu[] tên_mảng;
  - kiểu[,] tên_mảng;
  - Kiểu[][] tên mảng

Các kiểu dữ liệu cơ bản

- Kiểu chuỗi
  - Cú pháp: string tên_chuỗi;
  - Tạo chuỗi bằng phương thức ToString() của đối tượng.
- Kiểu liệt kê: là tập hợp các tên hằng có giá trị không thay đổi
  - Cú pháp

    enum <ten_liet_ke>[kieu_co_so]
    {danh_sach_cac_thanh_phan_liet_ke};

Các kiểu dữ liệu cơ bản

- Cấu trúc: là kiểu dữ liệu đơn giản, kích thước nhỏ, dùng thay thế lớp, cũng có thể chứa các phương thức, thuộc tính, các trường, các toán tử, các kiểu dữ liệu lồng bên trong.
  - struct <tên_cấu_trúc> [:danh_sách giao_diên]
    {
      [thành_viên_cấu trúc];
    }

Các kiểu dữ liệu cơ bản

```
using System;
public struct Location
  {public Location( int xCoordinate, int yCoordinate)
    {xVal = xCoordinate;
     yVal = yCoordinate;}
    public int x
    {get{return xVal;}
     set{xVal = value;}}
    public int y
    {get{return yVal;}
     set{yVal = value;}}
```

Các kiểu dữ liệu cơ bản

```
public override string ToString()
{return (String.Format("{0}, {1}",
   xVal, yVal));}
// thuộc tính private lưu toạ độ x, y
private int xVal;
private int yVal;
}
```

Lập trình hướng đối tượng C#

Bài 3

Yêu cầu

- Định nghĩa lớp và tạo được các thể hiện của lớp (đối tượng).
- Khai báo và sử dụng các phương thức trong lớp, cách nạp chồng phương thức.
- Truyền tham số, các từ khóa ref, out và params.
- Cơ chế ủy quyền và sự kiện (delegate - event).

Định nghĩa lớp

Để định nghĩa một kiểu dữ liệu mới hay một lớp đầu tiên phải khai báo rồi sau đó mới định nghĩa các thuộc tính và phương thức của kiểu dữ liệu đó.

[bổ sung truy cập] class <định danh lớp>[:Lớp cơ sở]

{

<phần thân của lớp bao gồm định nghĩa các thuộc tính và phương thức hành động>

}

Bổ sung truy cập

Bổ sung truy cập	Giới hạn truy cập
public	Không hạn chế.
private	Chỉ được truy cập bởi các phương thức trong cùng một lớp.
protected	Truy cập bởi các phương thức trong lớp A và những lớp dẫn xuất từ A
internal	Truy cập bởi những phương thức của bất cứ lớp nào trong khối hợp ngữ của A
protected internal	Truy cập bởi các phương thức của lớp A, dẫn xuất từ A và các lớp nằm cùng trong khối hợp ngữ với A.

Định danh lớp

- Định danh lớp là tên của lớp do người xây dựng chương trình tạo ra được viết theo đúng quy ước chuẩn.
- Lớp cơ sở là lớp mà đối tượng sẽ kế thừa.
- Tất cả các thành viên của lớp được định nghĩa trong thân của lớp, được bao bọc bởi hai dấu ({})

Ví dụ minh họa

Tạo một lớp thời gian, hiển thị thời gian trong ngày.

```
using System;
public class ThoiGian
{  private int Nam;
   private int Thang;
   private int Ngay;
   private int Gio;
   private int Phut;
   private int Giay;
```

Ví dụ minh họa

```
public void ThoiGianHienHanh()
   {
Console.WriteLine("Hien thi thoi gian hien
   hanh");
   }
}
public class Tester
{  static void Main()
   {    ThoiGian t = new ThoiGian();
        t.ThoiGianHienHanh();
   }
}
```

Tham số của phương thức

Một phương thức có thể lấy bất kỳ số lượng tham số nào. Mỗi tham số phải khai báo kèm với kiểu dữ liệu

```
void Method(int p1, string p2)
{
    //thân của phương thức
}
```

Tạo đối tượng

Sử dụng từ khóa new để tạo một đối tượng

```
ThoiGian t= new ThoiGian();
```

Các đối tượng là kiểu dữ liệu tham chiếu và được tạo ra trên heap >< kiểu dữ liệu giá trị được tạo ra trên stack

Bộ khởi dựng

- Một phương thức khởi dựng (constructor) sẽ được gọi thực hiện khi ta tạo một đối tượng. Chức năng của nó là tạo các đối tượng được xác định bởi một lớp và đặt trạng thái này hợp lệ.
- Nếu không tạo bộ khởi dựng thì CLR sẽ tự động tạo bộ khởi dựng mặc định, các thành viên được khởi tạo giá trị tầm thường (int, long, byte →0)
- Khai báo bộ khởi dựng là khai báo một phương thức có tên trùng với tên lớp.

Bộ khởi dựng

```
public ThoiGian( System.DateTime dt )
{
   Nam = dt.Year;
   Thang = dt.Month;
   Ngay = dt.Day;
   Gio = dt.Hour;
   Phut = dt.Minute;
   Giay = dt.Second;
}
```

Bộ khởi dựng

```
public class Tester
{
static void Main()
{
System.DateTime  currentTime =
   System.DateTime.Now;
ThoiGian t = new ThoiGian( currentTime );
t.ThoiGianHienHanh();
}
}
```

Bộ khởi dựng sao chép

Bộ khởi dựng sao chép thực hiện việc tạo một đối tượng mới bằng cách sao chép tất cả các biến từ một đối tượng đã có và cùng một kiểu dữ liệu.

```
public ThoiGian( ThoiGian tg)
{ Nam = tg.Nam;
  Thang = tg.Thang;
  Ngay = tg.Ngay;
  Gio = tg.Gio;
  Phut = tg.Phut;
  Giay = tg.Giay;
}
```

Bộ khởi dựng tĩnh

Nếu một lớp khai báo bộ khởi dựng tĩnh (static constructor), thì được đảm bảo rằng bộ khởi dựng tĩnh này sẽ được thực hiện trước bất kỳ thể hiện nào của lớp được tạo ra

```
static ThoiGian()
{
    Ten = “Thoi gian”;
}
```

Không có bất cứ thuộc tính truy cập nào như public trước bộ khởi dựng tĩnh.

Bộ khởi dựng private

C# không có phương thức toàn cục và hằng số toàn cục. Do vậy chúng ta có thể tạo ra những lớp tiện ích nhỏ chỉ để chứa các phương thức tĩnh. Để ngăn ngừa việc tạo bất cứ thể hiện của lớp ta tạo ra bộ khởi dựng không có tham số và không làm gì cả, tức là bên trong thân của phương thức rỗng, và thêm vào đó phương thức này được đánh dấu là private. Do không có bộ khởi dựng public, nên không thể tạo ra bất cứ thể hiện nào của lớp.

Khởi tạo biến thành viên

Các biến thành viên có thể được khởi tạo trực tiếp khi khai báo, thay vì phải thực hiện khởi tạo các biến trong bộ khởi dựng.

```
private int Giay=30;
```

Khi xác định giá trị khởi tạo như vậy biến sẽ không nhận giá trị mặc định mà chương trình cung cấp. Nếu các biến này không được gán lại giá trị trong bộ khởi dựng thì nó sẽ nhận giá trị đã được khởi tạo

Từ khóa this

Từ khóa this dùng để tham chiếu đến thể hiện hiện hành của một đối tượng, được xem con trỏ ẩn của tất cả các phương thức không có thuộc tính tĩnh trong một lớp.

Tham chiếu this được sử dụng:

- Khi các biến thành viên bi che lấp bởi các tham số đưa vào

```
public void SetYear(int Nam)
        {
                this.Nam=Nam;
        }
```

Từ khóa this

- Sử dụng tham chiếu this để truyền đối tượng hiện hành vào một tham số của phương thức đối tượng khác
- Sử dụng tham chiếu this là mảng, chỉ mục (indexer)

```
public string this [ int index ]
{ get {… }
  set {…}
}
```

Sử dụng các thành viên tĩnh (static)

Thuộc tính và phương thức trong một lớp có thể là thành viên thể hiện (**instance members**) hay thành viên tĩnh (**static members**). Thành viên thể hiện hay thành viên của đối tượng liên quan đến thể hiện của một kiểu dữ liệu. Trong khi thành viên tĩnh được xem như một phần của lớp. Chúng ta có thể truy cập đến thành viên tĩnh của một lớp thông qua tên lớp đã được khai báo. Còn để truy cập đến thành viên thể hiện buộc phải thông qua thể hiện của lớp (đối tượng)

Gọi một phương thức tĩnh

```
using System;
public class Class1
{
   public void SomeMethod(int p1, float p2)
   {
        Console.WriteLine("Ham nhan duoc hai
                tham so: {0} va {1}", p1,p2);
   }
}
```

Gọi một phương thức tĩnh

```
public class Tester
{
   static void Main()
   {
        int var1 = 5;
        float var2 = 10.5f;
        Class1 c = new Class1();
        c.SomeMethod( var1, var2 );
   }
}
```

Nạp chồng phương thức

Khi xây dựng lớp, ta có mong muốn tạo ra nhiều hàm có cùng tên nhưng nhận tham số khác nhau. Chức năng này gọi là nạp chồng phương thức

```
void myMethod( int p1 );
void myMethod( int p1, int p2 );
void myMethod( int p1, string p2 );
```

Truyền tham số

Tham số có kiểu dữ liệu là giá trị thì sẽ được truyền giá trị vào cho phương thức. Tuy nhiên,C# còn cung cấp khả năng cho phép ta truyền các đối tượng có kiểu giá trị dưới hình thức là tham chiếu.

- ref cho phép truyền các đối tượng giá trị vào trong phương thức theo kiểu tham chiếu.
- out trong trường hợp muốn truyền dưới dạng tham chiếu mà không cần phải khởi tạo giá trị ban đầu cho tham số truyền.
- params cho phép phương thức chấp nhận số lượng nhiều các tham số.

Truyền tham chiếu

```
using System;
public class Time
{  public void DisplayCurrentTime()
    { Console.WriteLine("{0}/{1}/{2}/
    {3}:{4}:{5}", Date,
          Month, Year, Hour, Minute, Second);
          }
public void GetTime(int h, int m, int s)
{
h = Hour;
m =  Minute;
s = Second;
}
```

Truyền tham chiếu

```
public Time( System.DateTime dt)
{ Year = dt.Year;
   Month = dt.Month;
   Date = dt.Day;
   Hour = dt.Hour;
   Minute = dt.Minute;
   Second = dt.Second;}
private int Year;          private int Month;
private int Date;          private int Hour;
private int Minute;        private int Second;
}
```

Truyền tham chiếu

```
public class Tester
{  static void Main()
{System.DateTime currentTime =
   System.DateTime.Now;
  Time t = new Time( currentTime);
  t.DisplayCurrentTime();
  int theHour = 0;
  int theMinute = 0;
  int theSecond = 0;
  t.GetTime( theHour, theMinute, theSecond);
  System.Console.WriteLine("Current time:
   {0}:{1}:{2}",
    theHour, theMinute, theSecond);
} }
```

delegate

Ủy quyền (**delegate**) là kiểu dữ liệu tham chiếu được dùng để đóng gói một phương thức với tham số và kiểu trả về xác định. Chúng ta có thể đóng gói bất cứ phương thức thích hợp nào vào trong một đối tượng ủy quyền. Nó không cần biết đến những lớp đối tượng mà nó tham chiếu tới. Điều cần quan tâm đến những đối tượng đó là các đối mục của phương thức và kiểu trả về phải phù hợp với đối tượng ủy quyền khai báo.

```
public delegate int WhichIsFirst(object obj1, object obj2);
```

delegate

```
namespace NewExample
{
   public delegate void ExampleDelagate();
    class Example
    {  public void Method1()
       {Console.WriteLine("Method 1"); }
       public void Method2()
       {Console.WriteLine("Method 2"); }
    }
```

delegate

```
class Program
{  public static void Main()
  {  Example   a=new Example();
     ExampleDelegate deg;
     deg=new ExampleDeleage(a.Method1);
     deg();
     deg=new ExampleDeleage(a.Method2);
     deg();
  }
}
```

event

Trong môi trường giao diện đồ họa (**Graphical User Interfaces**: GUIs), Windows hay trong trình duyệt web đòi hỏi các chương trình phải đáp ứng các sự kiện (**event**). Một sự kiện có thể là một nút lệnh được nhấn, một mục trong menu được chọn, hành động sao chép tập tin hoàn thành,...Một hành động nào đó xảy ra, và ta phải đáp ứng lại sự kiện đó. Chúng ta không thể đoán trước được khi nào thì các sự kiện sẽ xuất hiện. Hệ thống sẽ chờ cho đến khi nhận được sự kiện, và sẽ chuyển vào cho trình xử lý sự kiện thực hiện.

Hướng đối tượng C#(tt)

Bài 4

Yêu cầu

- Khai báo sử dụng thuộc tính của lớp, sử dụng các loại thuộc tính khác nhau.
- Định nghĩa nạp chồng toán tử, các toán tử có thể nạp chồng và cú pháp nạp chồng toán tử.

Định nghĩa thuộc tính

Thuộc tính là khái niệm cho phép truy cập trạng thái của lớp thông qua phương thức của lớp thay vì truy cập trực tiếp tới các biến thành viên. Đặc tính này cung cấp khả năng bảo vệ các trường bên trong một lớp.

Ví dụ minh họa

```
class Circle
    {
        private double radius;
        public Circle(double radius)
        {
            this.radius=radius;
        }
        public double Radius
        {
            get { return radius; }
            set { radius = value; }
        }
    }
```

Truy cập lấy dữ liệu (get accessor)

Khai báo giống như một phương thức của lớp dùng để trả về một đối tượng có kiểu dữ liệu của thuộc tính.

```
get
{
    return radius;
}
```

Bất cứ khi nào ta tham chiếu đến một thuộc tính hay gán giá trị thuộc tính cho một biến thì bộ truy cập lấy dữ liệu sẽ được thực hiện

```
Circle c=new Circle(5);
double r=c.Radius;
```

Truy cập thiết lập dữ liệu (set accessor)

Bộ truy cập này sẽ thiết lập một giá trị mới cho thuộc tính. Khi định nghĩa bộ truy cập thiết lập dữ liệu ta phải dùng từ khóa **value**

```
set
{
    radius=value;
}
```

Khi ta gán một giá trị cho thuộc tính, bộ truy cập thiết lập dữ liệu sẽ được tự động thực hiện

```
    double r=10;
    c.Radius=r;
```

Thuộc tính chỉ đọc, chỉ viết

Ta có thể tạo ra thuộc tính chỉ đọc bằng cách bỏ thủ tục **set** trong khai báo và có thể tạo thuộc tính chỉ ghi bằng cách bỏ thủ tục **get** trong khai báo

```
private string name;
public string Name
    {
        get
            {return name;}
    }
```

Lưu ý

C# không cho phép cài đặt những bổ từ khác nhau cho thủ tục **set** và **get**. Nếu muốn tạo ra một thuộc tính có **public** để đọc, nhưng lại muốn hạn chế **protected** trong gán thì đầu tiên phải tạo thuộc tính chỉ đọc với **public** và sau đó tạo một phương thức **set** với bổ từ **protected** bên ngoài thuộc tính đó.

Lưu ý

```
public string Name
{
   get {return name;}
}
protected void SetName (string value)
{
   if (value.Length>20)
        //code xử lý khi dl không hợp lệ
   else
        name=value;
}
```

Thuộc tính virtual, abstract, override

C# cho phép tạo các thuộc tính **virtual**, **abstract** hoặc **overriden**. Để khai báo **virtual**, **abstract** hay **overriden** ta chỉ cần thêm các từ khóa này trong lúc định nghĩa thuộc tính

```
public abstract string Name
{
    get;
    set;
}
```

Thuộc tính trong giao diện

Giao diện là ràng buộc, giao ước đảm bảo cho các lớp hay các cấu trúc sẽ thực hiện một điều gì đó. Một giao diện thì giống như một lớp chỉ chứa các phương thức trừu tượng.

```
interface IStorable
{       void Read();
        void Write(object obj);
        int Status
            {get;}
            {set;}
}
```

Thuộc tính mảng

Index trong C# cho phép truy xuất những tập hợp nằm trong lớp, sử dụng cú pháp []. Index được xem như là một thuộc tính khá đặc biệt kèm theo phương thức **get** và **set**. Khai báo Indexer cho phép tạo ra những lớp hoạt động tương tự như mảng ảo.

```
class StringList
{    public string [] list;
     public string this[int index]
     {
       get {return list[index];}
       set {list[index]=value;}
     }
}
```

Nạp chồng toán tử

Hướng thiết kế của C# là các lớp do người dùng định nghĩa có tất cả các chức năng của lớp được xây dựng sẵn.

Giả sử ta định nghĩa một lớp số phức, đảm bảo rằng lớp này có tất cả các chức năng như lớp được xây dựng sẵn

```
SoPhuc sp=sp1+sp2;
```

Tại sao phải nạp chồng toán tử

Nạp chồng toán tử làm mã nguồn chương trình trực quan, những hành động của lớp xây dựng giống như các lớp được xây dựng sẵn.

Nếu ta nạp chồng toán tử so sánh bằng (==) để kiểm tra hai đối tượng có bằng nhau hay không thì đồng thời cũng phải nạp chồng toán tử nghịch với toán tử bằng là toán tử không bằng (!=)

Quy tắc nạp chồng toán tử:

- Định nghĩa các toán tử trong kiểu dữ liệu giá trị, kiểu do ngôn ngữ xây dựng sẵn
- Phương thức nạp chồng toán tử chỉ bên trong lớp
- Sử dụng tên và ký hiệu quy ước trong Common Language Specification (CLS)

Danh sách các toán tử

Biểu tượng	Tên phương thức thay thế	Tên toán tử
+	Add	Toán tử cộng
-	Subtract	Toán tử trừ
*	Multiply	Toán tử nhân
/	Divide	Toán tử chia
%	Mod	Toán tử chia lấy dư
==	Equals	Toán tử so sánh bằng
!=	Compare	Toán tử so sánh ko bằng
>	Compare	Toán tử so sánh lớn hơn
<	Compare	Toán tử so sánh nhỏ hơn

Danh sách các toán tử (tt)

Biểu tượng	Tên phương thức thay thế	Tên toán tử
^	Xor	Toán tử or lấy ngoại trừ
&	BitwiseAnd	Toán tử and nhị phân
|	BitwiseOr	Toán tử or nhị phân
&&	And	Toán tử and logic
||	Or	Toán tử or logic
=	Assign	Toán tử gán
>>	RightShift	Toán tử dịch phải
<<	LeftShift	Toán tử dịch trái

Hỗ trợ ngôn ngữ .Net khác

C# cung cấp khả năng cho phép nạp chồng toán tử các lớp mà chúng ta xây dựng, nhưng điều này không hoặc rất ít được đề cập trong CLS.

Những ngôn ngữ .Net khác như VB.Net không hỗ trợ việc nạp chồng toán tử, do đó nếu chúng ta nạp chồng toán tử (+) thì ta cũng nên cung cấp chức năng Add để cộng hai đối tượng.

Từ khóa operator

Các toán tử là các phương thức tĩnh, giá trị trả về của nó thể hiện kết quả của một toán tử và các toán hạng. Khi ta tạo một toán tử cho một lớp là ta đã tiến hành nạp chồng những toán tử đó. Để nạp chồng toán tử + chúng ta viết như sau:

```
public static SoPhuc operator +( SoPhuc sp1, SoPhuc sp2)
```

Toán tử so sánh bằng

Nếu ta nạp chồng toán tử bằng (==) thì ta cũng cần phủ quyết phương thức ảo **Equals**(). Điều này giúp cho lớp tương thích với các ngôn ngữ .Net khác không hỗ trợ nạp chồng toán tử nhưng hỗ trợ nạp chồng phương thức:

```
public override bool Equals(object o)
    {
            if(!(o is SoPhuc))
                    return false;
            return this==(SoPhuc)o;
    }
```

Toán tử chuyển đổi

Có thể tiến hành chuyển đổi dữ liệu theo hai cách:

- *Ngầm định-**implicit** (Đảm bảo dữ liệu chuyển đổi một cách an toàn không bi mất mát)*
- *Tường minh-**explicit** (Không đảm bảo tính toàn vẹn của dữ liệu trong quá trình chuyển đổi)*

Toán tử chuyển đổi

Ví dụ ta có một lớp số phức với hai thành phần là phần thực và phần ảo

Thực hiện các toán tử chuyển đổi đối với double a=5; SoPhuc b;

- An toàn: b=5;
- Không an toàn: a=(double) b

Chuyển đổi ngầm định, tường minh

```
public static implicit operator SoPhuc (double a)
{
   return new SoPhuc(a);
}

public static explicit operator double (SoPhuc sp)
{
   return sp.phanThuc;
}
```

Phủ quyết phương thức ToString()

Nếu ta không tiến hành phủ quyết phương thức **ToString**(), phương thức này mặc định luôn trả về kiểu của đối tượng.

Mong muốn rằng:

```
SoPhuc sp=new SoPhuc(3,4);
string s=sp.ToString();
```

sẽ ra kết quả s=3+4i

Phủ quyết phương thức ToString()

```
public override string ToString()
  {
   return phanThuc.ToString() + "+" +
   phanAo.ToString() + "i";
   }
```

Bài tập

Xây dựng lớp điểm trong không gian hai chiều với:

- *Bộ khởi dựng:*
  - *Mặc định*
  - *Nhận một tham số*
  - *Hai tham số.*
- *Định nghĩa thuộc tính truy cập trạng thái của lớp chỉ đọc.*
- *Nạp chồng toán tử cộng, trừ, nhân, chia*
- *Phủ quyết phương thức* ***ToString****() ra dạng (hoành độ x,tung độ y)*

Hướng đối tượng C# (tt)

Bài 5

Yêu cầu

- Hiểu được khái niệm kế thừa, đa hình. Tầm quan trong của vấn đề này trong LTHĐT.
- Biết cách thực thi kế thừa, sử dụng các kiểu đa hình khác nhau.
- Xây dựng lớp cài đặt giao diện, thực thi các giao diện khác nhau.
- Một số giao diện chuẩn trong thư viện C#.

Đặc biệt hóa, tổng quát hóa

- Lớp và thể hiện của lớp tuy không tồn tại trong cùng một khối, nhưng chúng tồn tại trong một mạng lưới phụ thuộc và quan hệ lẫn nhau
- Đặc biệt hóa và tổng quát hóa là hai mối quan hệ đối ngẫu và phân cấp với nhau

Đặc biệt hóa, tổng quát hóa

Ví dụ: Ta có thể nói xe máy, ôtô là trường hợp đặc biệt của xe, vì: ngoài những đặc điểm của xe nói chung, xe máy và ôtô còn có những đặc điểm riêng.

Tương tự Honda, Suzuki, Yamaha là những trường hợp đặc biệt của xe máy

BMW, Nissan, Toyota, Honda, Huyndai là những trường hợp đặc biệt của xe ôtô

Sự kế thừa (inheritance)

Trong C# quan hệ đặc biệt hóa được thực thi bằng cách sử dụng sự kế thừa. Đây là cách chung nhất, tự nhiên nhất để thực thi quan hệ này

Ta có thể nói xe máy, ôtô được kế thừa hay dẫn xuất từ lớp Xe. Lớp Xe được coi là lớp cơ sở, xe máy, ôtô được coi là lớp dẫn xuất.

Thực thi kế thừa

Để tạo một lớp dẫn xuất từ một lớp ta thêm dấu hai chấm vào sau tên lớp và trước tên của lớp cơ sở.

```
public class XeMay:Xe
public class Oto:Xe
```

Lớp dẫn xuất sẽ kế thừa tất cả phương thức, biến thành viên của lớp cơ sở. Lớp dẫn xuất cũng có thể tạo phương thức mới bằng việc đánh dấu với từ khóa **new**

Sử dụng lớp dẫn xuất

```
namespace Example
{    public class Xe
    {  private string name;
       public Xe(string name)
       { this.name = name; }
        public void Who()
    {Console.WriteLine("Toi la mot chiec xe");
        }
   }
```

Sử dụng lớp dẫn xuất

```
public class XeMay : Xe
    {   private int sobanh;
        public XeMay(string name, int sobanh)
            : base(name)
        {       this.sobanh = sobanh;       }
        public new void Who()
        {   base.Who();
            Console.WriteLine("Xe may {0}
    banh",sobanh);
        }
    }       }
```

Sử dụng lớp dẫn xuất

```
class Tester
{    static void Main()
     {
          Xe xe1=new Xe("Xe");
          xe1.Who();
          XeMay xe2=new XeMay("Xe
may",2);
          xe2.Who();

     }
}
```

Gọi phương thức khởi dựng

Các lớp không được kế thừa phương thức khởi dựng của lớp cơ sở, do đó lớp dẫn xuất phải thực thi phương thức khởi dựng của riêng nó.

Chỉ có thể sử dụng phương thức khởi dựng của lớp cơ sở thông qua việc gọi tường minh.

```
public XeMay(string name, int sobanh)
        : base(name)
```

Đa hình (polymorphism)

Đa hình là khả năng cho phép gởi cùng một thông điệp đến những đối tượng khác nhau có cùng chung một đặc điểm, nói cách khác thông điệp được gởi đi không cần biết thực thể nhận thuộc lớp nào, chỉ biết rằng tập hợp các thực thể nhận có chung một tính chất nào đó.

VD:thông điệp "vẽ hình" được gởi đến cả hai đối tượng hình hộp và hình tròn. Trong hai đối tượng này đều có chung phương thức vẽ hình, tuy nhiên tuỳ theo thời điểm mà đối tượng nhận thông điệp, hình tương ứng sẽ được vẽ lên.

Phương thức đa hình

Để tạo một phương thức đa hình, cần khai báo khóa **virtual** trong phương thức của lớp cơ sở.

Ví dụ: `public virtual void Who()`

Lúc này các lớp dẫn xuất được tự do thực thi các cách xử lý của riêng mình trong các phiên bản mới của phương thức Who().

Để làm được điều này cần thêm từ khóa **override** để chồng lên phương thức ảo Who() của lớp cơ sở.

Phương thức đa hình

```
namespace Example
{    public class Xe
    {  private string name;
       public Xe(string name)
       { this.name = name; }
        public virtual void Who()
    {Console.WriteLine("Toi la mot chiec xe");
        }
   }
```

Phương thức đa hình

```
public class XeMay : Xe
    {   private int sobanh;
        public XeMay(string name, int sobanh)
            : base(name)
        {       this.sobanh = sobanh;       }
        public override void Who()
        {   base.Who();
            Console.WriteLine("Xe may {0}
    banh",sobanh);
        }
    }       }
```

Phương thức đa hình

```
class Tester
{    static void Main()
     {       Xe xe1=new Xe("Xe");
             xe1.Who();
             XeMay xe2=new XeMay("Xe may",2);
             xe2.Who();
             Xe[] xeArr=new Xe[3];
             xeArr[0]=new Xe("Xe");
             xeArr[1]=new XeMay("Xe may1",2);
             xeArr[2]=new Xemay("Xe may2",2);
             for (int i=0;i<3;i++)
                     xeArr[i].Who();
     }
}
```

Lớp trừu tượng (abstract)

Mỗi lớp con của lớp **Xe** nên thực thi một phương thức Who(), nhưng điều này không bắt buộc. Để yêu cầu các lớp con phải thực thi một phương thức của lớp cơ sở, chúng ta phải thiết kế một cách trừu tượng.

Lớp trừu tượng được thiết lập như là cơ sở cho những lớp dẫn xuất, việc tạo các thể hiện cho các lớp trừu tượng là không hợp lệ.

Lớp trừu tượng (abstract)

```
namespace Example
{   abstract public class Xe
    {  protected string name;
       public Xe(string name)
       { this.name = name; }
        abstract public void Who();
        //abstract public void Run();
    }
```

Lớp trừu tượng (abstract)

```
public class XeMay : Xe
    {   private int sobanh;
        public XeMay(string name, int sobanh)
            : base(name)
        {       this.sobanh = sobanh;       }
        public override void Who()
        {Console.WriteLine("Xe may {0}
    banh",sobanh);   }
        }
    }
```

Lớp trừu tượng (abstract)

```
class Tester
{    static void Main()
     {       XeMay xe2=new XeMay("Xe may",2);
             xe2.Who();
             Xe[] xeArr=new Xe[3];
             xeArr[0]=new Xe("Xe");
             xeArr[1]=new XeMay("Xe may1",2);
             xeArr[2]=new Xemay("Xe may2",2);
             for (int i=0;i<3;i++)
                    xeArr[i].Who();
     }
}
```

Lớp trừu tượng (abstract)

Những lớp trừu tượng không có sự thực thi căn bản; chúng thể hiện ý tưởng về một sự trừu tượng, điều này thiết lập một sự giao ước cho tất cả các lớp dẫn xuất. Các lớp trừu tượng mô tả một phương thức chung của tất cả các lớp được thực thi một cách trừu tượng.

Lớp cô lập (sealed class)

Ngược với các lớp trừu tượng là các lớp **cô lập**. Một lớp trừu tượng được thiết kế cho các lớp dẫn xuất và cung cấp các khuôn mẫu cho các lớp con theo sau. Trong khi một lớp **cô lập** thì không cho phép các lớp dẫn xuất từ nó. Để khai báo một lớp **cô lập** ta dùng từ khóa ***sealed*** đặt trước khai báo của lớp không cho phép dẫn xuất. Hầu hết các lớp thường được đánh dấu ***sealed*** nhằm ngăn chặn các tai nạn do sự kế thừa gây ra.

Giao diện (interface)

- Giao diện là ràng buộc, giao ước đảm bảo cho các lớp hay các cấu trúc sẽ thực hiện một điều gì đó.
- Một giao diện đưa ra một sự thay thế cho các lớp trừu tượng để tạo ra các sự ràng buộc giữa những lớp và các thành phần client của nó. Những ràng buộc này được khai báo bằng cách sử dụng từ khóa **interface**, từ khóa này khai báo một kiểu dữ liệu tham chiếu để đóng gói các ràng buộc.

Thực thi giao diện

Cú pháp để định nghĩa một giao diện như sau:

[bổ sung truy cập] interface <tên giao diện> [: danh sách cơ sở]

{

<phần thân giao diện>

}

Thực thi giao diện

Giả sử cần tạo một giao diện nhằm mô tả những phương thức và thuộc tính của một lớp cần thiết để lưu trữ và truy cập từ một cơ sở dữ liệu. Giao diện này là IStorage.

```
interface IStorable
{
    void Read();
    void Write(object o);
}
```

Thực thi giao diện

Mục đích của một giao diện là để định nghĩa những khả năng mà chúng ta muốn có trong một lớp.

*Tạo một lớp tên là **Document**, lớp này lưu trữ các dữ liệu trong cơ sở dữ liệu, do đó này thực thi giao diện **IStorable**.*

```
public class Document : IStorable
{
   public Document( string s)
{ Console. WriteLine("Creating document with: {0}", s);}
        public void Read()
        {        ....       }
        public void Write()
        {        ....       }
}
```

Thực thi nhiều giao diện

C# cho phép thực hiện nhiều hơn một giao diện. Nếu lớp Document được lưu trữ và dữ liệu cũng được nén.→ Document thực thi hai giao diện:

```
public class Document : IStorable, ICompressible
{
   public Document( string s)
{ Console. WriteLine("Creating document with: {0}", s);}
        public void Read()
        {        ....      }
        public void Write()
        {        ....      }
        public void Compress()
        {        .....     }
        public void Decompress()
        {        .....     }
}
```

Mở rộng giao diện

C# cung cấp chức năng cho chúng ta mở rộng một giao diện đã có bằng cách thêm các phương thức và các thành viên hay bổ sung cách làm việc cho các thành viên. Ví dụ, chúng ta có thể mở rộng giao diện ICompressible với một giao diện mới là ILoggedCompressible.

Giao diện mới này mở rộng giao diện cũ bằng cách thêm phương thức ghi log các dữ liệu đã lưu:

```
interface ILoggedCompressible : ICompressible
{
    void LogSavedBytes();
}
```

Một số giao diện chuẩn trong C#

Giao diện	Mục đích
IEnumerable	Liệt kê thông qua tập hợp
ICollection	Thực thi bởi tất cả tập hợp→ cung cấp CopyTo(), Count, ISReadOnly…
IComparer	So sánh hai đối tượng
IList	Tập hợp mảng đc chỉ mục
IDictionary	Dùng trong các tập hợp dựa trên khóa và giá trị như Hashtable, SortedList

Giao diện IEnumerable

Giao diện này chỉ có một phương thức duy nhất là **GetEnumerator**(), công việc của phương thức là trả về một sự thực thi đặc biệt của **IEnumerator**.

Enumerator phải thực thi những phương thức và thuộc tính **IEnumerator**.

Giao diện IEnumerable

```
public class ListBoxTest: IEnumerable
{  private class ListBoxEnumerator : IEnumerator
   {  private ListBoxTest lbt;
      private int index;
      public ListBoxEnumerator(ListBoxTest lbt)
     { this.lbt = lbt;
       index = -1;  }
     public bool MoveNext()
     { index++;
       if (index >= lbt.strings.Length)
            return false;
        else
            return true;}
```

Giao diện IEnumerable

```
public void Reset()
{ index = -1; }
public object Current
 { get
   { return( lbt[index]); }
  }
}
public IEnumerator GetEnumerator()
{
return (IEnumerator) new ListBoxEnumerator(this);
}
```

Giao diện IEnumerable

```
private string[] strings;
private int ctr = 0;
public ListBoxTest (params string[] initStr)
{  strings = new string[10];
   foreach (string s in initStr)
   {    strings[ctr++] = s; }
}
public string this[int index]
   {get { if ( index < 0 || index >= strings.Length)
          {        ….       }
          return strings[index];        }
     set {  strings[index] = value;  }
}
```

Bài tập

Xây dựng một giao diện IDisplay khai báo thuộc tính Name kiểu chuỗi. Viết hai lớp Dog và Cat thực thi giao diện IDisplay, cho biết thuộc tính Name là tên của đối tượng.

Cấu trúc dữ liệu trong C#

Bài 7

Yêu cầu

- Nắm được các khái niệm cơ bản về danh sách liên kết, hàng đợi, ngăn xếp…
- Biết cách thao tác, ứng dụng của danh sách liên kết, hàng đợi, ngăn xếp, … vào các vấn đề cụ thể.

Danh sách liên kết

```
using System;
using System.Collections.Generic;
using System.Text;
public class lk
    {  public Node head, current;
       public class Node
       {  public Node next;
          public int item;
       }
    }
```

Danh sách liên kết

```
static void Main(string[] args)
    {lk danhsach = new lk();
     danhsach.head = null;
     for (int i = 1; i <= 4; i++)
     {
      danhsach.current = new lk.Node();
       danhsach.current.item=i*10;
       danhsach.current.next = danhsach.head;
       danhsach.head = danhsach.current;
    Console.WriteLine(danhsach.current.item);
    }     }
```

Hàng đợi (Queue)

- ***Hàng đợi*** *là một tập hợp trong đó có thứ tự vào trước và ra trước (**FIFO**).*
- ***Hàng đợi*** *là kiểu dữ liệu tốt để quản lý những nguồn tài nguyên giới hạn. Ví dụ, chúng ta muốn gởi thông điệp đến một tài nguyên mà chỉ xử lý được duy nhất một thông điệp một lần. Khi đó chúng ta sẽ thiết lập một hàng đợi thông điệp để xử lý các thông điệp theo thứ tự đưa vào.*

Hàng đợi (Queue)

Phương thức- thuộc tính	Mục đích
Synchronized()	Phương thức static trả về một Queue wrapper được thread-safe.
Count	Thuộc tính trả về số thành phần trong hàng đợi
IsReadOnly	Thuộc tính xác định hàng đợi là chỉ đọc
IsSynchronized	Thuộc tính xác định hàng đợi được đồng bộ
SyncRoot	Thuộc tính trả về đối tượng có thể được sử dụng để đồng bộ truy cập Queue.
Clear()	Xóa tất cả các thành phần trong hàng đợi
Clone()	Tạo ra một bản sao
Contains()	Xác định xem một thành phần có trong mảng.
CopyTo()	Sao chép những thành phần của hàng đợi đến mảng một chiều đã tồn tại
Dequeue()	Xóa và trả về thành phần bắt đầu của hàng đợi.
Enqueue()	Thêm một thành phần vào hàng đợi.
GetEnumerator()	Trả về một enumerator cho hàng đợi.
Peek()	Trả về phần tử đầu tiên của hàng đợi và không xóa nó.
ToArray()	Sao chép những thành phần qua một mảng mới

Hàng đợi (Queue)

```
public static void Main()
{Queue  intQueue  = new Queue();
 for(int i=0; i <5; i++)
{intQueue.Enqueue(i*5);}
  Console.Write("intQueue values:\t");
  PrintValues( intQueue);
  Console.WriteLine("\nDequeue\t{0}",
   intQueue.Dequeue());
  Console.Write("intQueue values:\t");
  PrintValues(intQueue);
  Console.WriteLine("\nPeek   \t{0}",
   intQueue.Peek());
  Console.Write("intQueue values:\t");
  PrintValues(intQueue);
 }
```

Hàng đợi (Queue)

```
public static void PrintValues(IEnumerable
   myCollection)
{
   IEnumerator myEnumerator =
   myCollection.GetEnumerator();
   while (myEnumerator.MoveNext())
   Console.Write("{0}  ",
   myEnumerator.Current);
   Console.WriteLine();
}
```

Ngăn xếp (Stack)

- **Ngăn xếp** là một tập hợp mà thứ tự là vào trước ra sau hay vào sao ra trước (**LIFO**).
- Hai phương thức chính cho việc thêm và xóa từ **Stack** là *Push* và *Pop*, ngoài ra ngăn xếp cũng đưa ra phương thức *Peek* tương tự như *Peek* trong hàng đợi.
- Phương thức và thuộc tính của lớp **Stack**:

Ngăn xếp (Stack)

Phương thức- thuộc tính	Mục đích
Synchronized()	Phương thức static trả về một Stack wrapper được thread-safe.
Count	Thuộc tính trả về sô thành phần trong ngăn xếp
IsReadOnly	Thuộc tính xác định ngăn xếp là chỉ đọc
IsSynchronized	Thuộc tính xác định ngăn xếp được đồng bộ
SyncRoot	Thuộc tính trả về đối tượng có thể được sử dụng để đồng bộ truy cập Stack.
Clear()	Xóa tất cả các thành phần trong ngăn xếp
Clone()	Tạo ra một bản sao
Contains()	Xác định xem một thành phần có trong mảng.
CopyTo()	Sao chép những thành phần của ngăn xếp đến mảng một chiều đã tồn tại
Pop()	Xóa và trả về phần tử đầu Stack
Push()	Đưa một đối tượng vào đầu ngăn xếp
GetEnumerator()	Trả về một enumerator cho ngăn xếp.
Peek()	Trả về phần tử đầu tiên của ngăn xếp và không xóa nó.
ToArray()	Sao chép những thành phần qua một mảng mới

Ngăn xếp (Stack)

```
public class Tester
{static void Main()
 {Stack intStack = new Stack();
   for (int i=0; i < 8; i++)
   {intStack.Push(i*5);}
   Console.Write("intStack values:\t");
   PrintValues( intStack );
   Console.WriteLine("\nPop\t{0}",
   intStack.Pop());
```

Ngăn xếp (Stack)

```
Console.Write("intStack values:\t");
   PrintValues( intStack );
   Console.WriteLine("\nPeek   \t{0}",
   intStack.Peek());
   Console.Write("intStack values:\t");
PrintValues( intStack );
Array targetArray =
   Array.CreateInstance(typeof(int), 12);
for(int i=0; i <=8; i++)
{
targetArray.SetValue(100*i, i);
}
```

Ngăn xếp (Stack)

```
Console.WriteLine("\nTarget array:  ");
PrintValues( targetArray );
intStack.CopyTo( targetArray, 6);
Console.WriteLine("\nTarget array after
   copy:  ");
PrintValues( targetArray );
Object[] myArray = intStack.ToArray();
Console.WriteLine("\nThe new array:  ");
PrintValues( myArray );
```

Kiểu từ điển

- Từ điển là kiểu tập hợp trong đó có hai thành phần chính liên hệ với nhau là khóa và giá trị.
- Kiểu dữ liệu từ điển trong .NET Framework có thể kết hợp bất cứ kiểu khóa nào như kiểu chuỗi, số nguyên, đối tượng...với bất cứ kiểu giá trị nào (chuỗi, số nguyên, kiểu đối tượng).

Bảng băm (Hashtables)

- **Hashtable** là một kiểu từ điển được tối ưu cho việc truy cập được nhanh.
- Trong một **Hashtable**, mỗi giá trị được lưu trữ trong một vùng. Mỗi vùng được đánh số tương tự như là từng offset trong mảng. Do khóa có thể không phải là số nguyên, nên phải chuyển các khóa thành các khóa số để ánh xạ đến vùng giá trị được đánh số.

Giao diện từ điển (IDictionary)

- Hashtable là một từ điển ví nó thực thi giao diện IDictionary. IDictionary cung cấp một thuộc tính public là Item.
- Trong ngôn ngữ C# thuộc tính Item được khai báo như sau:

```
object  this[object key]
{ get;  set;}
```

Giao diện từ điển (IDictionary)

```
// tạo và khởi tạo hashtable
Hashtable hashTable = new Hashtable();
hashTable.Add("00440123","Ngoc Thao");
hashTable.Add("00123001","My Tien");
hashTable.Add("00330124","Thanh Tung");
// truy cập qua thuộc tính Item
Console.WriteLine("myHashtable[\"00440123
   \"]: {0}",
hashTable["00440123"]);
```

Lập trình tổng quát trong C#

Bài 8

Nội dung

- Khái niệm, khai báo, cách sử dụng lập trình tổng quát (Generic).
- Lập trình tổng quát lớp, cấu trúc và các hàm
- Một số cấu trúc dữ liệu tổng quát được xây dựng sẵn trong C# như Collection, List, Dictionary…

Giới thiệu lập trình tổng quát

- Trong C++ đã đề cập tới khái niệm **Template** (thường gọi là mẫu), **Template** được dùng để tạo các hàm, các class mà không cần quan tâm đến kiểu dữ liệu của đối số. **Template** được đưa ra với mục đích tăng tính năng sử dụng lại mã nguồn.
- **Generic** trong C# đưa ra các tính năng tương tự như **Template** trong C++

Giới thiệu lập trình tổng quát (tt)

Khái niệm **Generic** được đưa vào C# từ version 2.0 và CLR. **Generic** mang đến .Net framework khái niệm mới về kiểu tham số. Các lớp, các hàm đĩnh nghĩa không cần chỉ rõ tham số đưa vào thuộc kiểu dữ liệu gì, tất cả được sử dụng một cách chung nhất. Người dùng có thể phát triển thành từng lớp, từng hàm với đối số là một kiểu dữ liệu xác định.

Giới thiệu lập trình tổng quát (tt)

```
public class GenericList<T>
{    void Add(T input){}              }
class TestGenericList
{    private class ExampleClass    {   }
     static void Main()
{ GenericList<int> list1=new
GenericList<int>();
   GenericList<string> list2=new
     GenericList<string>();
 GenericList<ExampleClass> list3=new
     GenericList<ExampleClass>(); }    }
```

Đặc điểm của lập trình tổng quát

- **Generic** định nghĩa một thao tác dữ liệu với kiểu dữ liệu chung nhất nhằm tối đa hóa việc sử dụng lại code trong chương trình, tạo ra các kiểu dữ liệu an toàn, đem lại hiệu suất cao nhất.
- Ứng dụng phổ biến nhất của **Generic** là tạo ra các collection class (lớp dữ liệu tập hợp).
- Trong .NET framework có chứa sẵn các collection framework trong namespace **System.Collections.Generic**.
- **Generic** có thể tạo được các interface, class, method, event và delegates.

Lớp tập hợp (class collection)

- Lớp tập hợp (class collection) được dùng để lặp từng phần tử trong một lớp thông qua câu lệnh **foreach**.
- Phần lớn các lớp tập hợp xuất phát từ việc thực thi các giao diện chuẩn của C# như: ICollection, IComparer, IEnumerable, IList, IDictionary, IDictionaryEnumerator và các generic tương đương của chúng.

Lớp tập hợp (tt)

```
public class Tokens: IEnumerable
{
private string[] elements;
public Tokens(string source, char[]
delimiters)
{
    elements=source.Split(delimiters);
}
public IEnumerator GetEnumerator()
{
return new TokenEnumerator(this)
}
```

Lớp tập hợp (tt)

```
private class TokenEnumerator:IEnumerator
{
   private int position = -1;
   private Tokens t;
   public TokenEnumerator(Tokens t)
    {
        this.t = t;
    }
```

Lớp tập hợp (tt)

```
public bool MoveNext()
{
 if (position < t.elements.Length - 1)
{  position++; return true; }
else { return false; }
}
public void Reset()
{ position = -1; }
public object Current
{
get { return t.elements[position]; }
}
```

Lớp tập hợp (tt)

```
    static void Main()
  {
    Tokens f = new Tokens("This is a well-
    done program.", new char[] {' ','-'});
    foreach (string item in f)
    {
     Console.WriteLine(item);
    }
  }
}
```

Khai, sử dụng báo kiểu Generic

```
List<int> intList=new List<int>();
intList.Add(3);
intList.Add(23);
intList.Add(6.0);
foreach(int val in intList)
    Console.WriteLine(val);
```

Khai, sử dụng báo kiểu Generic

```
List<string> stringList=new List<string>();
   stringList.Add("Lap");
   stringList.Add("trinh");
   stringList.Add("C#");
   foreach(string val in stringList)
        Console.WriteLine(val);
```

Lập trình Generic với Class

Việc lập trình Generic với Class giúp tối thiểu hóa được thời gian lập trình, tăng khả năng sử dụng lại mã nguồn mà không hề ảnh hưởng đến hiệu năng và tính hướng đối tượng của chương trình

```
public class Couple<T, E>
{ public T elementA;
   public E elementB;
   public Couple(T inA, E inB)
       {  elementA = inA;
          elementB = inB;
       }   }
```

Lập trình Generic với Class

Lớp này được dùng khi cần có một đối tượng tạm thời chỉ gồm hai phần tử, bình thường phải định nghĩa một class mới để phù hợp với kiểu của 2 phần tử nói trên

Ví dụ: muốn một đối tượng gồm 2 phần tử string và một số int, khai báo như sau :

```
Couple<string, int> couple = new
Couple<string, int>(”Age”, 29);
```

couple.elementA sẽ có kiểu string nhận giá trị “Age”

couple.elementB sẽ có kiểu int nhận giá trị 29.

Lập trình Generic với Class

Chú ý: có thể bổ sung thêm **Property** cho lớp Couple<T, E> nếu thấy cần thiết (khi lập trình aspx):

```
public class Couple<T, E>
{       public T elementA;
        public E elementB;
        public Couple(T inA, E inB)
    {     elementA = inA;
          elementB = inB;        }
        public T ElementA
        {   get{return elementA;}
              set{elementA = value;}        }
}
```

Lập trình Generic với Class

```
static void Main()
{ Couple<int,int> a=new
   Couple<int,int>(2,3);
   Console.WriteLine(a.elementB);
   Couple<int,string> a=new
   Couple<int,string>(2,"3");
   Console.WriteLine(a.elementB);

}
```

Lập trình Generic với Class

Tương tự như vậy, có thể khai báo thêm các lớp Generic Triple<T, E, F> (bộ ba) và Quad<T, E, F, V> (bộ bốn)

```
public class Triple<T, E, F>
{public T elementA;
  public E elementB;
  public F elementC;
  public Triple(T inA, E inB, F inC)
{    elementA = inA;
      elementB = inB;
     elementC = inC;
   }
 }
```

Lập trình Generic với struct

Nói chung không có nhiều khác biệt giữa lập trình Generic Class và struct:

```
public struct Couple<T, E>
{       public T elementA;
        public E elementB;
        public Couple(T inA, E inB)
    {     elementA = inA;
            elementB = inB;         }
        public T ElementA
        {   get{return elementA;}
                set{elementA = value;}        }
}
```

Lập trình Generic với function

C# cũng cho phép lập trình Generic với hàm, sau đây là một ví dụ :

```
public string toString<T>(List<Couple<string, T>> list)
{   string result = "";
    foreach (Couple<string, T> pair in list)
    {
    string tmp = pair.elementA + " : " +
    pair.elementB.ToString();
    result += tmp + '\n';
    }
    return result;
 }
```

Generic Dictionaries trong C#

```
public static void Main()
{
    Dictionary<string, string> openWith = new
    Dictionary<string, string>();
    openWith.Add("txt", "notepad.exe");
    openWith.Add("bmp", "paint.exe");
    openWith.Add("dib", "paint.exe");
    openWith.Add("rtf", "wordpad.exe");
    try { openWith.Add("txt", "winword.exe"); }
    catch (ArgumentException)
    { Console.WriteLine("An element with Key =
    \"txt\" already exists."); }
}
```

Generic Dictionaries trong C#

```
Console.WriteLine("For key = \"rtf\", value =
    {0}.", openWith["rtf"]);
openWith["rtf"] = "winword.exe";

foreach( KeyValuePair<string, string> kvp in
    openWith ) { Console.WriteLine("Key = {0},
    Value = {1}", kvp.Key, kvp.Value); }

Dictionary<string, string>.ValueCollection
    valueColl = openWith.Values;
foreach( string s in valueColl ) {
    Console.WriteLine("Value = {0}", s); }
```

Generic Collection trong C#

- Các cấu trúc tổng quát còn lại như Collection, List thao tác cũng tương tự như cấu trúc từ điển tổng quát.
- Các ví dụ cụ thể về các trường hợp này có thể xem thêm trong **MSDN**

Lập trình Windows Form

Bài 10

Nội dung

- Thao tác với form controls và các loại controls cơ bản
- Sử dụng các loại hộp thoại
- Giới thiệu các loại menu cơ bản
- Xây dựng ứng dụng MDI

Các điều khiển cơ bản

- 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

- **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 vào nó để thể hiện một chức năng.
- **CheckBox** đối tượng cho phép chọn hoặc không chọn

Các điều khiển cơ bản

- **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 cho phép chứa các đối tượng khác
- **Panel**: đối tượng chứa các đối tượng khác

Form

- Các điều khiển của **Form**
  - 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.
  - Chú ý: Các thuộc tính, sự kiện của các đối tượng có cùng tên → cùng ý nghĩa.

Form

- 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**

Form

- Các thuộc tính thường dùng
  - **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**: Ví trí bắt đầu khi chạy Form
    - **CenterScreen**: Nằm giữa màn hình
  - **WindowState**: Xác định trạng thái ban đầu Form

Form

- Các phương thức thường dùng
  - **Close**: Đóng Form và giải phóng các tài nguyên. Một Form đã đóng không thể mở lại.
  - **Hide**: Ẩn Form và không giải phóng tài nguyên của Form.
  - **Show**: Hiển thị một Form đã ẩn.
- Các sự kiện thường dùng
  - **Load**: Xẩy ra khi chạy Form (ngầm định khi nháy đúp chuột trong chế độ thiết kế).
  - **FormClosing**: Xảy ra khi đóng Form.

Label

- Đối tượng hiển thị văn bản kết hợp hình ảnh
- Không 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 (ngầm định là không)
  - **`PasswordChar`**: Nhập ký tự làm mật khẩu
  - **`ReadOnly`**: Có/không cho phép sửa dữ liệu của đối tượng (ngầm định là có)
  - **`Text`**: Văn bản nhập (hiển thị) của đối tượng.

TextBox

- 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 ấn một phím bất kỳ trên đối tượng.
  - **KeyUp**: Xảy ra khi thả một phím ấn 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).

Các thuộc tính chung

- **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
- **Enable**: có/không cho phép thao tác với đối tượng
- **Font**: Font chữ của đối tượng
- **ForeColor**: màu chữ của đối tượng

Các thuộc tính chung

- **TabIndex**: thứ tự ấn phím tab để chuyển con trỏ đến đối 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
- **Dock**: cố định đối tượng trong đối tượng chứa
- **Location**: vị trí của đối tượng so với đối tượng chứa

Các sự kiện bàn phím

- **KeyDown**: xảy ra khi một phím được ấn trên đối tượng
- **KeyUp**: xảy ra khi một phím được thả trên đối tượng
  - KeyEventArg: tham số cho sự kiện KeyDown, KeyUp.
- **KeyPress**: xảy ra khi ấn và thả một phím trên đối tượng
  - KeyPressEventArg: tham số cho sự kiện KeyPress

Các sự kiện chuột

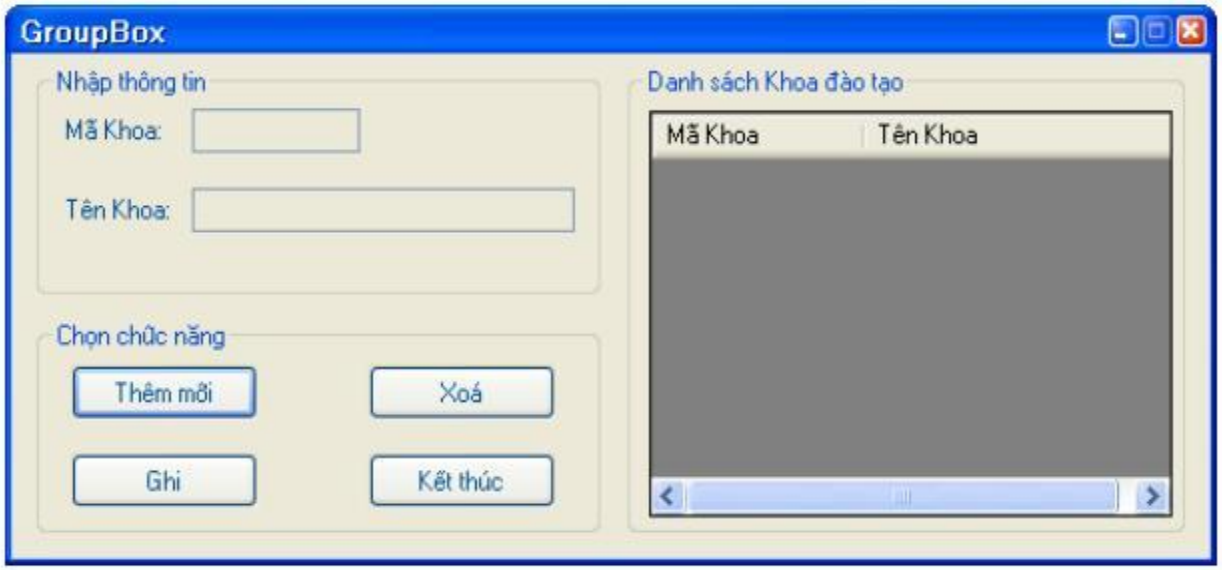
- **MouseEnter**: xảy ra khi đưa con trỏ vào vùng của đối tượng
- **MouseLeave**: xảy ra khi đưa con trỏ khỏi vùng của đối tượng
- **MouseDown**: xảy ra khi ấn nút chuột khi con trỏ chuột đang nằm trong vùng của đối tượng
- **MouseUp**: xảy ra khi nhả nút chuột trong khi con trỏ chuột đang nằm trong cùng của đối tượng
- **MouseMove**: xảy ra khi di chuyển con trỏ chuột trong vùng của đối tượng

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)

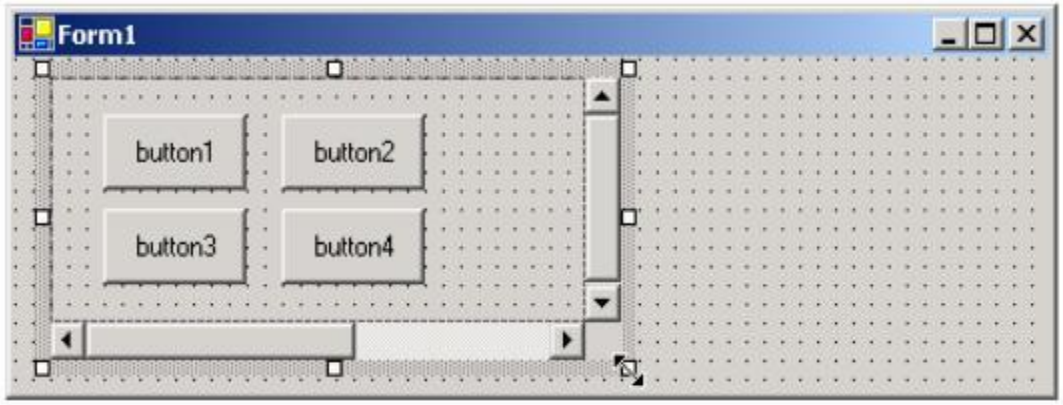
GroupBox

- 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 tieu đề **GroupBox**



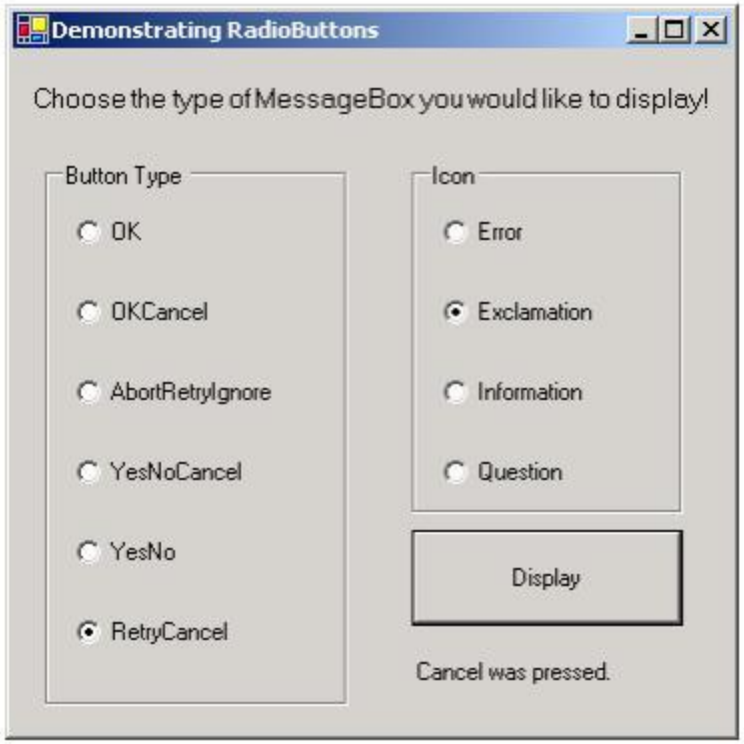
Panel

- 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`**)



RadioButton

- Đối tượng cho phép chọn/không chọn giá trị
- Cho phép chọn một đối tượng ở một thời điểm
  - Để chọn nhiều đối tượng phải đặt các điều khiển trong GroupBox hoặc Panel



RadioButton

- 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
  - **Click**: Xẩy 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)

ListBox

- Cho phép xem và chọn các dòng dữ liệu

ListBox

- 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.

```
MultiColum = true
```

ListBox

- Các thuộc tính thường dùng
  - **SelectedIndex**: Trả về dòng hiện thời được chọn
    - Nếu chọn nhiều dòng thì trả về 1 giá trị tuỳ 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.

ListBox

- Các thuộc tính thường dùng
  - **`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

`Sorted and SelectionMode`

ListBox

- Các phương thức thường dùng
  - **`GetSelected(index)`**: Trả về **`True`** dòng Index được chọn, ngược lại trả về false.
  - **`Add`**: Thêm một dòng vào ListBox
    - listBox1.**`Items.Add`**(“Cat”);
    - listtBox1.**`Items.Add`**(“Mouse”);
  - **`RemoveAt(row)`** : Xoá dòng ở vị trí row
    - listBox1.**`Items.RemoveAt`**(row);
  - **`Clear`**: Xoá tất cả các dòng
    - listBox1.**`Items.Clear()`**;

CheckedListBox

- **`CheckedListBoxe`** là sự mở rộng của ListBox bằng cách thêm CheckBox ở phía bên trái mỗi dòng
  - Có thể chọn các dòng

CheckedListBox

- 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.

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.

ComboBox

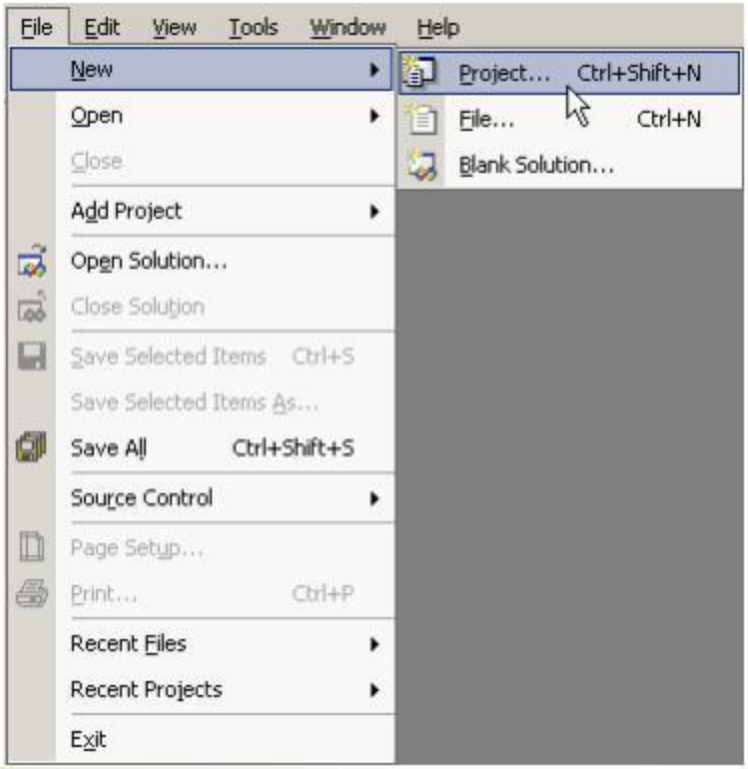
- Các thuộc tính thường dùng
  - **`Items`**: Mảng các dòng trong ComboBox
  - **`SelectedIndex`**: Chỉ số dòng được chọn. Nếu không chọn có giá trị **`-1`**.
  - **`SelectedItem`**: Giá trị dòng được chọn.
  - **`Sorted`**: Có/Không sắp xếp dữ liệu trong ComboBox. Ngầm định là **`false`**.
- Sự kiện thường dùng
  - **`SelectedIndexChanged`**: Xảy ra khi chọn 1 dòng.

ComboBox

- 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()**;

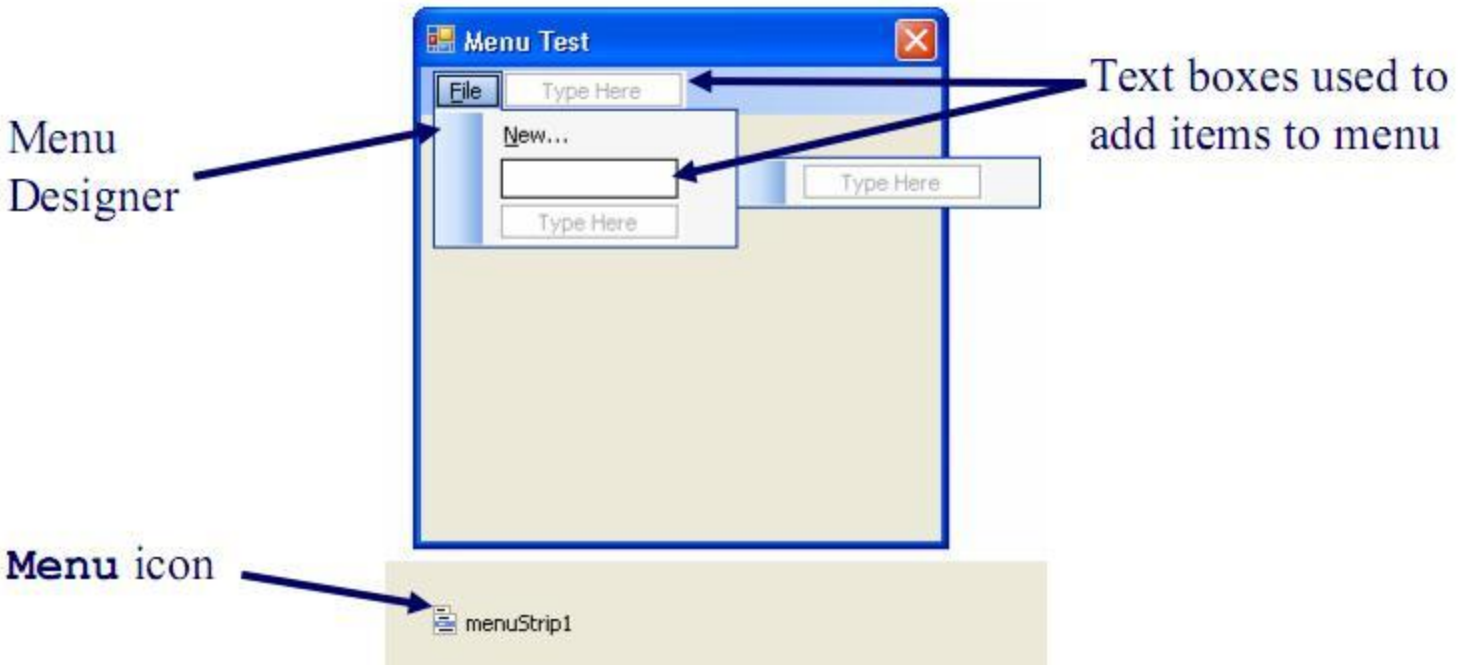
Menu

- 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
  - ...



Menu

- Xây dựng menu
  - Kéo biểu tượng MenuStrip vào Form



Menu

- Xây dựng menu
  - 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`**

Menu

- 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.

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

ListView

- Các thuộc tính thường dùng
  - **`CheckBoxes`**: Có/không xuất hiện các checkbox trên các dòng dữ liệu (ngầm định là **`False`**)
  - **`Columns`**: Các cột hiển thị trong chế độ **`Details`**.
  - **`FullRowSelect`**: Indicates whether all SubItems are hightlighted along with the Item when selected.
  - **`GridLines`**: Hiển thị lưới (chỉ hiển thị trong chế độ **`Details`**.

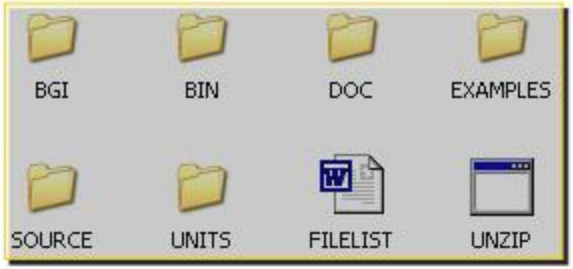
ListView

- Các thuộc tính thường dùng
  - **`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.

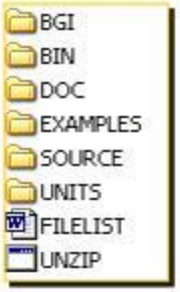
ListView

- Các thuộc tính thường dùng
  - **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

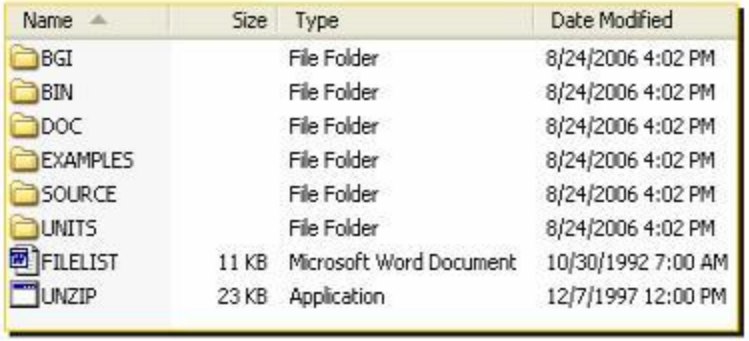
Icons



List



Details



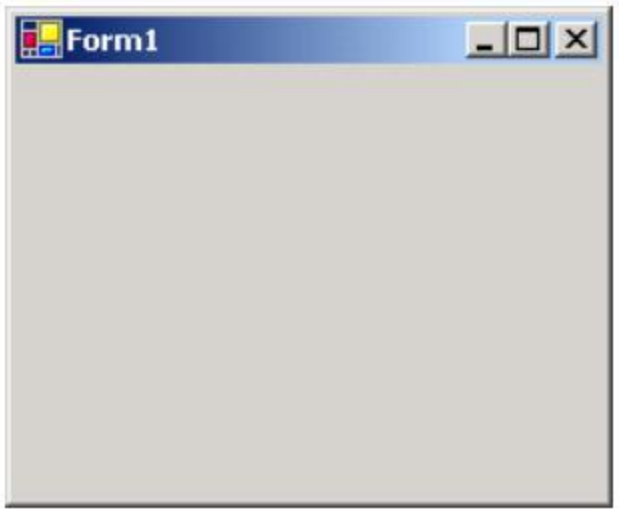
Name	Size	Type	Date Modified
BGI		File Folder	8/24/2006 4:02 PM
BIN		File Folder	8/24/2006 4:02 PM
DOC		File Folder	8/24/2006 4:02 PM
EXAMPLES		File Folder	8/24/2006 4:02 PM
SOURCE		File Folder	8/24/2006 4:02 PM
UNITS		File Folder	8/24/2006 4:02 PM
FILELIST	11 KB	Microsoft Word Document	10/30/1992 7:00 AM
UNZIP	23 KB	Application	12/7/1997 12:00 PM

ListView

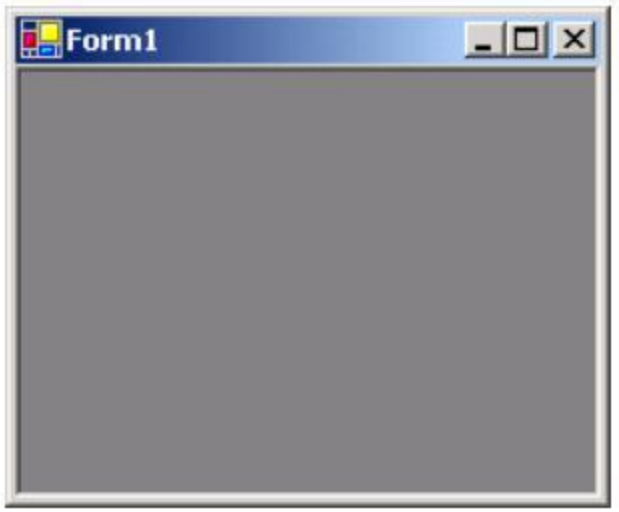
- Các phương thức thường dùng
  - **`Add`**: Thêm một dòng vào ListView
  - **`Clear`**: Xoá tất cả các dòng của ListView
  - **`Remove`**: Xoá một dòng trong ListView
  - **`RemoveAt(index)`** : Xoá một dòng ở vị trí index
- Sự kiện thường dùng
  - **`ItemSelectionChanged`**: Xảy ra khi chọn một dòng.

Ứng dụng MDI Windows

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



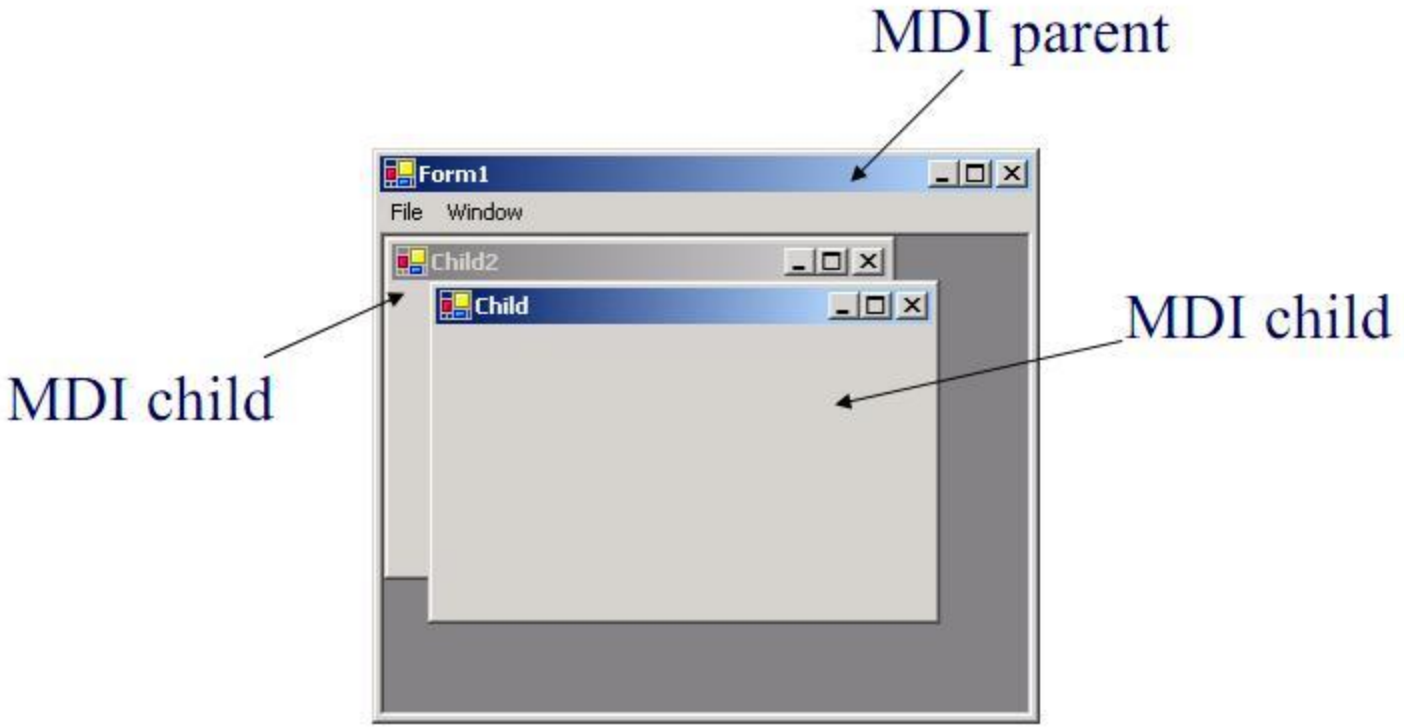
Single Document Interface (SDI)



Multiple Document Interface (MDI)

Ứng dụng MDI Windows

- Ví dụ MDI Parent và MDI Child



Ứng dụng MDI Windows

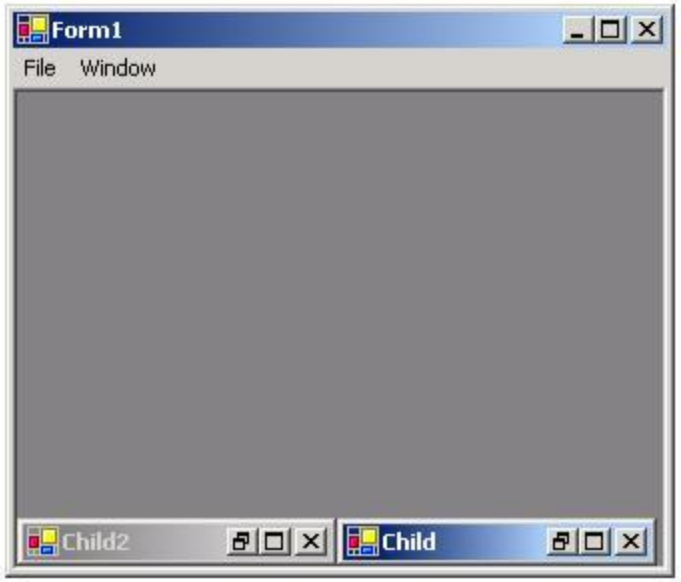
- Thuộc tính MDI Parent của Form
  - **`IsMdiContainer`**: Có/Không một Form là form MDI Parent. Ngầm định là False.
  - **`ActiveMdiChild`**: Trả về **`Form`** Child đang được kích hoạt.
- Thuộc tính MDI Child
  - **`IsMdiChild`**: Có/Không **`Form`** là một MDI child (thuộc tính read-only).
  - **`MdiParent`**: Chi ra một MDI parent của **`Form`**
    - **`<Tên form>.MdiParent = this`**

Ứng dụng MDI Windows

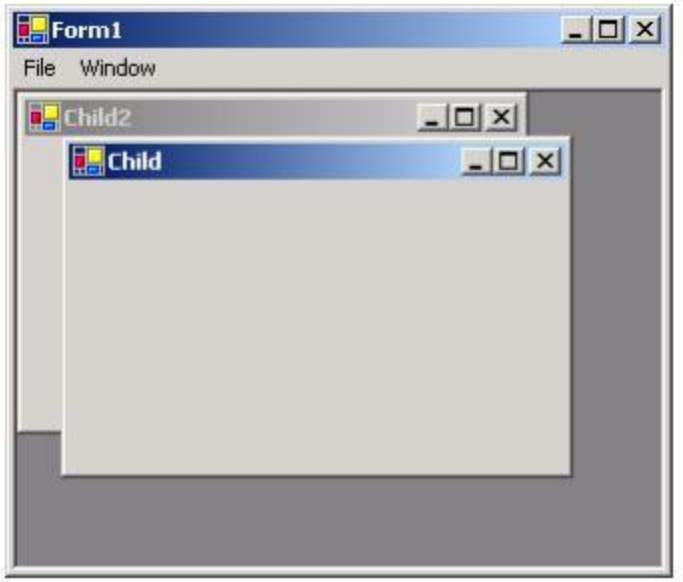
- Phương thức thường dùng
  - **`LayoutMdi`**: Xác định kiểu hiển thị của Form con trong MDI Form.
    - **`ArrangeIcons`**: Sắp xếp các biểu tượng dưới MDI
    - **`Cascade`**: Sắp xếp các cửa sổ chồng nhau
    - **`TileHorizontal`**: Sắp xếp cửa sổ theo chiều ngang
    - **`TileVertical`**: Sắp xếp cửa sổ theo chiều dọc

Ứng dụng MDI Windows

- Các kiểu sắp xếp



ArrangeIcons

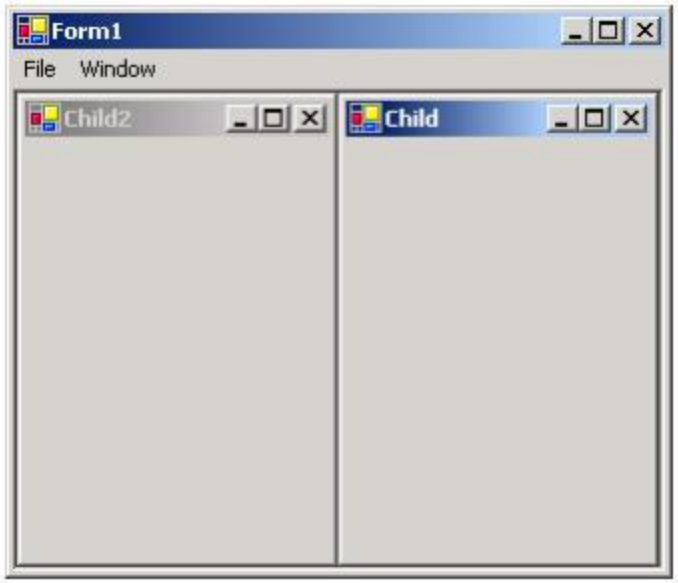


Cascade

Ứng dụng MDI Windows

- Các kiểu sắp xếp

TileHorizontal



TileVertical

Lập trình cơ sở dữ liệu

Bài 13+14

Nội dung

- Khái niệm về cơ sở dữ liệu.
- ADO và ADO.Net.
- Các đối tượng của ADO.Net.
- Kết nối đến Cơ sở dữ liệu MS Access và SQL Server.
- Tạo và hiển thị dữ liệu từ DataSet.
- Nạp dữ liệu vào các Controls cơ bản

Cơ sở dữ liệu (database)

Phần dữ liệu được lưu giữ trong máy tính theo một quy định nào đó được gọi là cơ sở dữ liệu (**database**).

Ưu điểm:

- *Giảm sự trùng lặp thông tin xuống mức thấp nhất và do đó bảo đảm được tính nhất quán và toàn vẹn dữ liệu.*
- *Đảm bảo dữ liệu có thể được truy xuất theo nhiều cách khác nhau.*
- *Khả năng chia sẻ thông tin cho nhiều người sử dụng và nhiều ứng dụng khác nhau.*

Cơ sở dữ liệu (database)

- Cơ sở dữ liệu quan hệ:
  - *Chứa dữ liệu trong các bảng, được cấu tạo bởi các dòng còn gọi là các mẩu tin, và cột còn gọi là các trường.*
  - *Cho phép lấy về (hay truy vấn) các tập hợp dữ liệu con từ các bảng.*
  - *Cho phép nối các bảng với nhau cho mục đích truy cập các mẩu tin liên quan với nhau chứa trong các bảng khác nhau.*

Khái niệm ADO, ADO.Net

- 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

Khái niệm ADO, ADO.Net

- Quá trình hình thành ADO.NET
  - ODBC
    - OLE DB
      - ADO (*ActiveX Data Objects*)
        - ADO.NET

Khái niệm ADO.Net

- 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.

ADO.Net

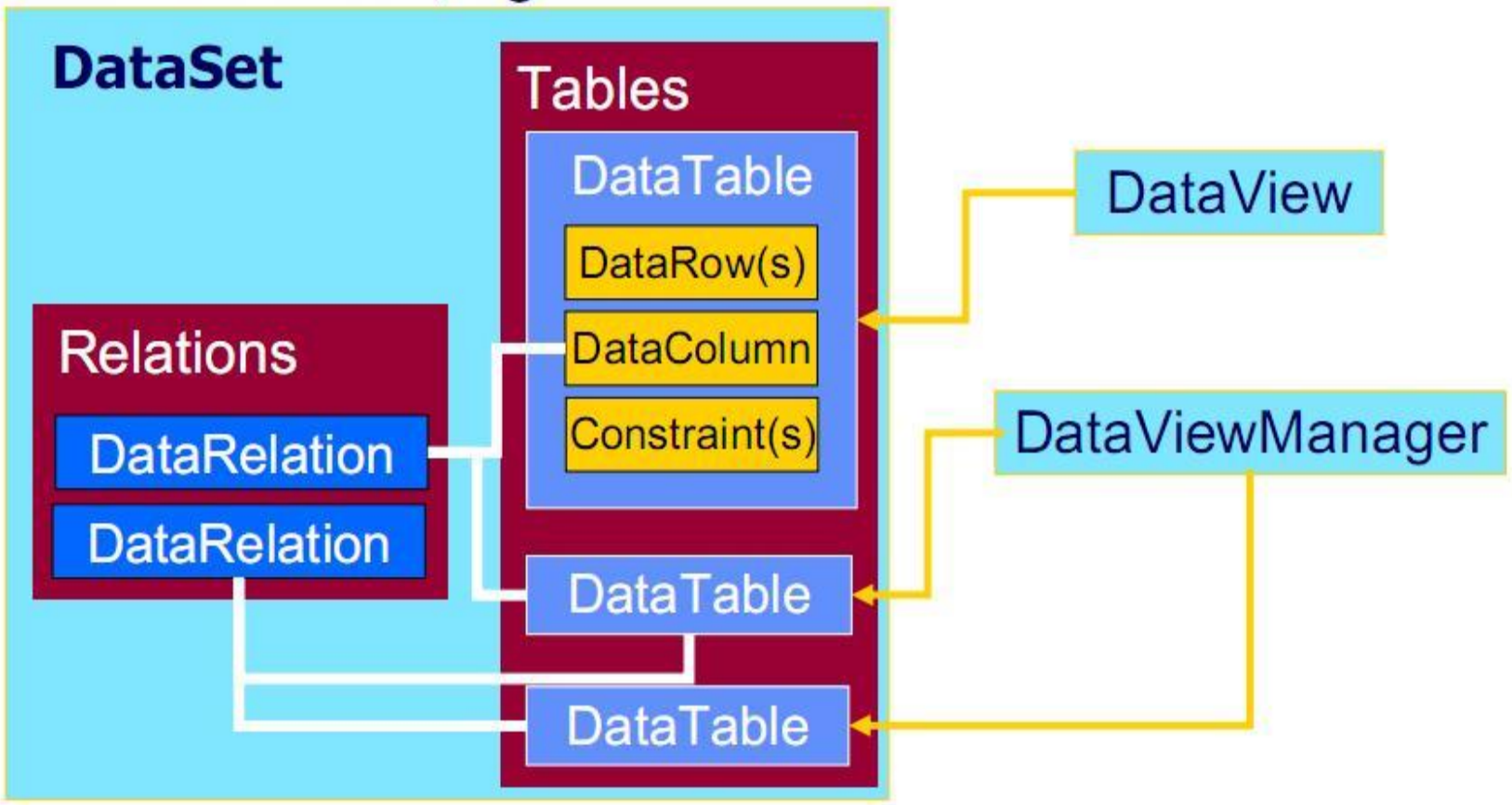
- **Disconnected Objects**
  - DataSet
  - DataTable
  - DataView
  - DataRow
  - DataColumn
  - Constraint
  - DataRelation
- **Connected Objects**
  - Connection
  - Transaction
  - DataAdapter
  - Command
  - Parameter
  - DataReader

ADO.Net

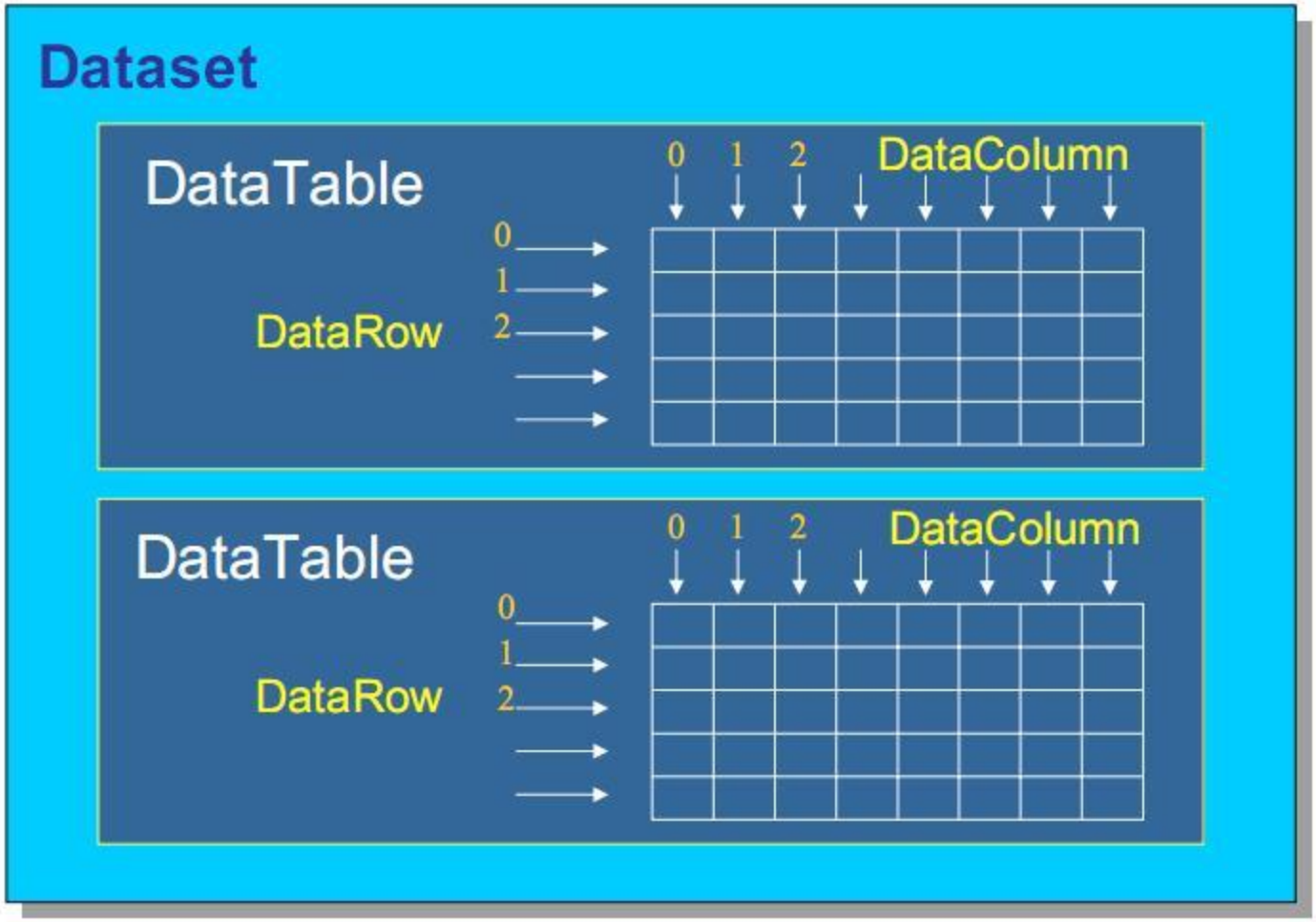
- Các đối tượng
  - DataSet: Một tập DataTable trong bộ nhớ
  - DataTable: Một bảng dữ liệu trong bộ nhớ
  - DataRow: Một bản ghi trong DataTable
  - DataColumn: Một cột dữ liệu trong DataTable
  - DataRelation: Đặt quan hệ của 2 DataTable
  - DataViewManager: Tạo Views của DataSet

ADO.Net

- Các đối tượng



ADO.Net

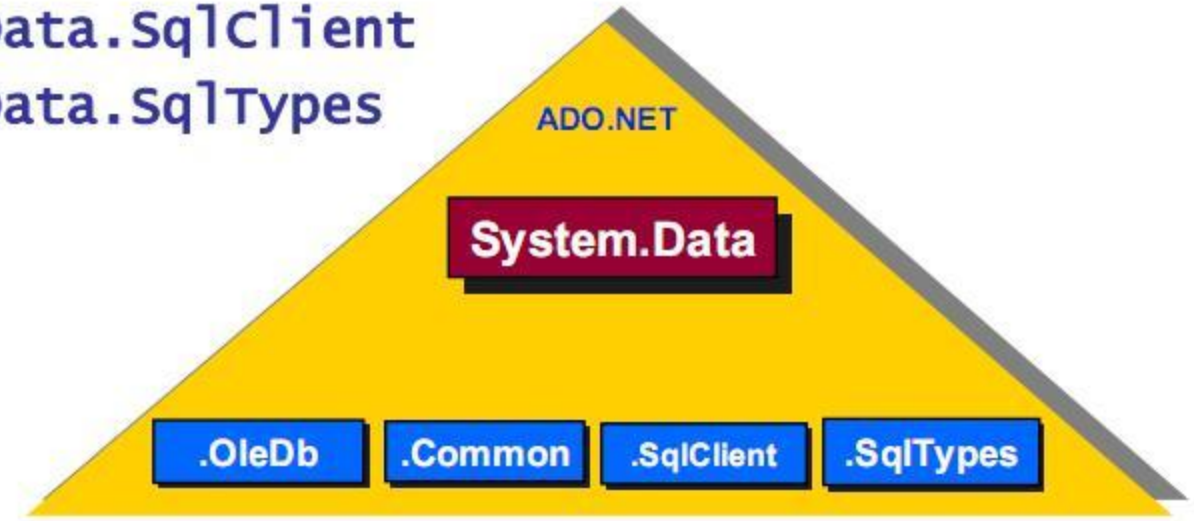

Dataset
DataTable
0
1
2
DataColumn
0
1
2
DataRow
DataTable
0
1
2
DataColumn
0
1
2
DataRow

ADO.Net

- 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

ADO.Net

- ADO.NET tổ chức thành mô hình đối tượng
  - System.Data
  - System.Data.OleDb
  - System.Data.Common
  - System.Data.SqlClient
  - System.Data.SqlTypes



ADO.Net

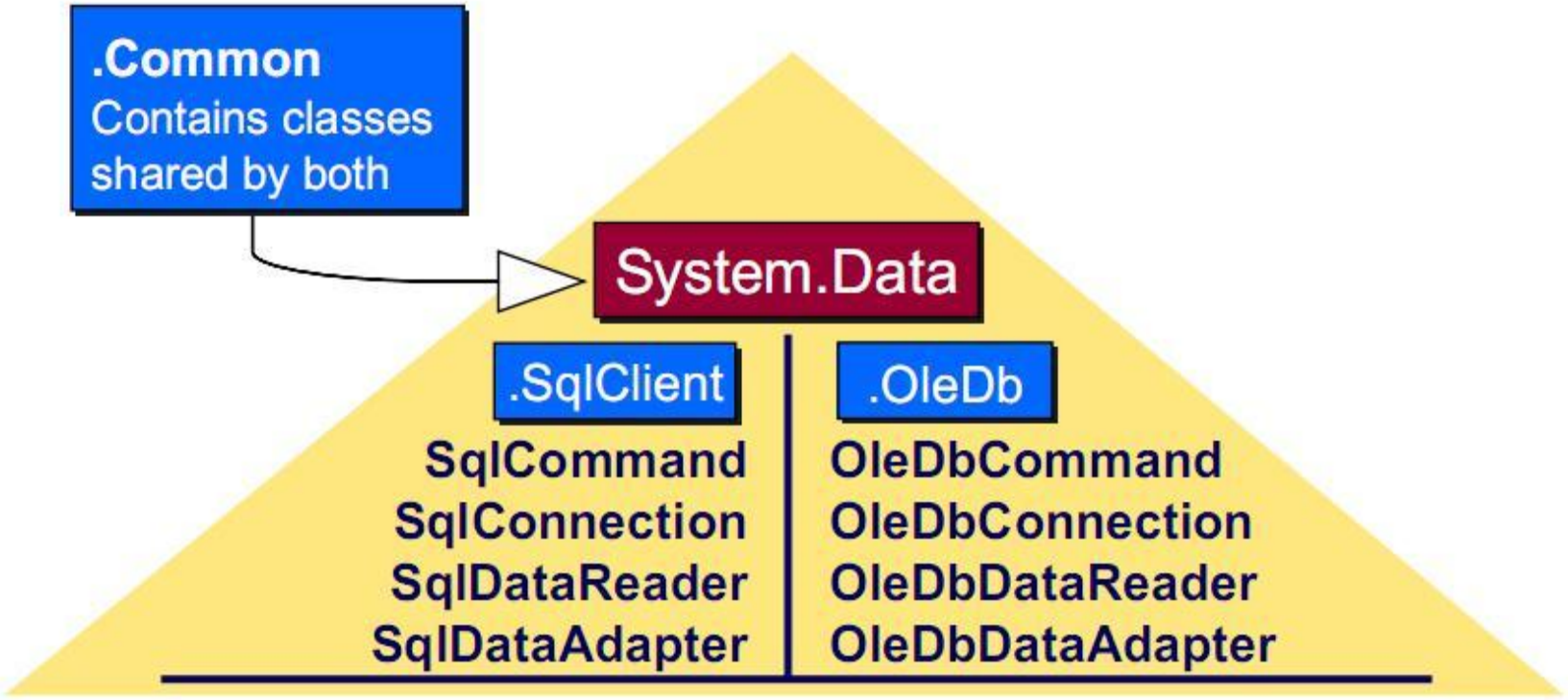
- **`System.Data`**: Các lớp của ADO.NET
- **`System.Data.OleDb`**: Các lớp làm việc với dữ liệu OLE DB
- **`System.Data.SqlClient`**: Các lớp làm việc với cở sở dữ liệu SQL Server

Các đối tượng ADO.Net

- ADO.NET Data Providers
  - Là các lớp truy nhập dữ liệu nguồn
    - Microsoft SQL Server™ 2000, SQL Server 7
    - Oracle
    - Microsoft Access
  - Thiết lập kết nối giữa **DataSet**s 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

Các đối tượng ADO.Net

- ADO.NET Data Providers



Các đối tượng ADO.Net

- Đối tượng Connection
  - Biểu diễn kết nối tới cơ sở dữ liệu

```
//Ket noi toi co so du lieu MS Access
string conStr = "Provider=Microsoft.Jet.OLEDB.4.0;" +
                "Data Source=<DataName>";
OleDbConnection myConnection = new OleDbConnection(conStr);
myConnection.Open();
```

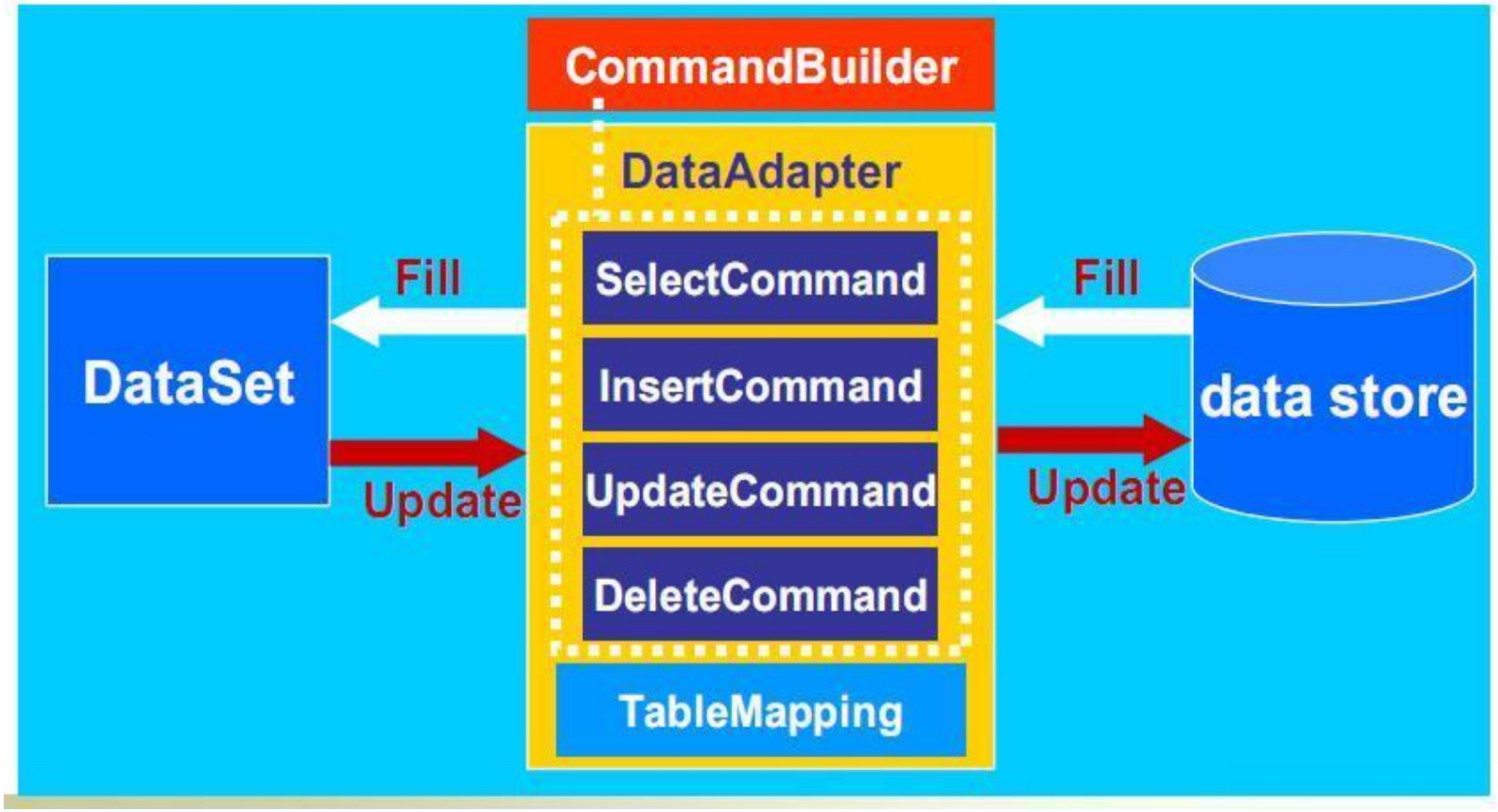
```
//Ket noi toi co so du lieu SQL Server
string conStr = "Data Source = <Computer Name>;" +
                "Persist Security Info = true;"  +
                "Initial Catalog = <DataName>;"  +
                "User Id =name; Password=psw;"   +
                "Connect Timeout = <seconds>";
SqlConnection myConnection = new SqlConnection(conStr);
myConnection.Open();
```

Các đối tượng ADO.Net

- 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

Các đối tượng ADO.Net

- Đối tượng DataAdapter



Các đối tượng ADO.Net

- Đối tượng DataAdapter
  - 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 ";
//Ket noi toi co so du lieu
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 minh họa

- Đối tượng DataAdapter
  - Ví dụ xây dựng form hiển thị dữ liệu

Name: txtMaKhoa

Name: txtTenKhoa

Name: btnFirst, btnPrevious, btnNext, btnLast

Ví dụ minh họa

- Đối tượng DataAdapter

```
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();
}
```

Ví dụ minh họa

- Đối tượng DataAdapter

```
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();
}
```

Ví dụ minh họa

- Đối tượng DataAdapter

```
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();
}
```

Các đối tượng

- 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 **`Edit columns`**
    - 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.

Các đối tượng

- Đối tượng DataGridView
  - **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

Các đối tượng

- Đối tượng DataGridView
  - Sự kiện thường dùng
    - **`RowEnter`**: Xảy ra khi con trỏ đưa vào một dòng
      - **`e.RowIndex`**: Dòng hiện thời
      - **`e.ColumnIndex`**: Cột hiện thời

Các đối tượng

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

Các đối tượng

- Đối tượng DataGridView

```
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;
}
```

Các đối tượng

- Đối tượng DataGridView

```
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)
    {
    }
}
```

Các đối tượng

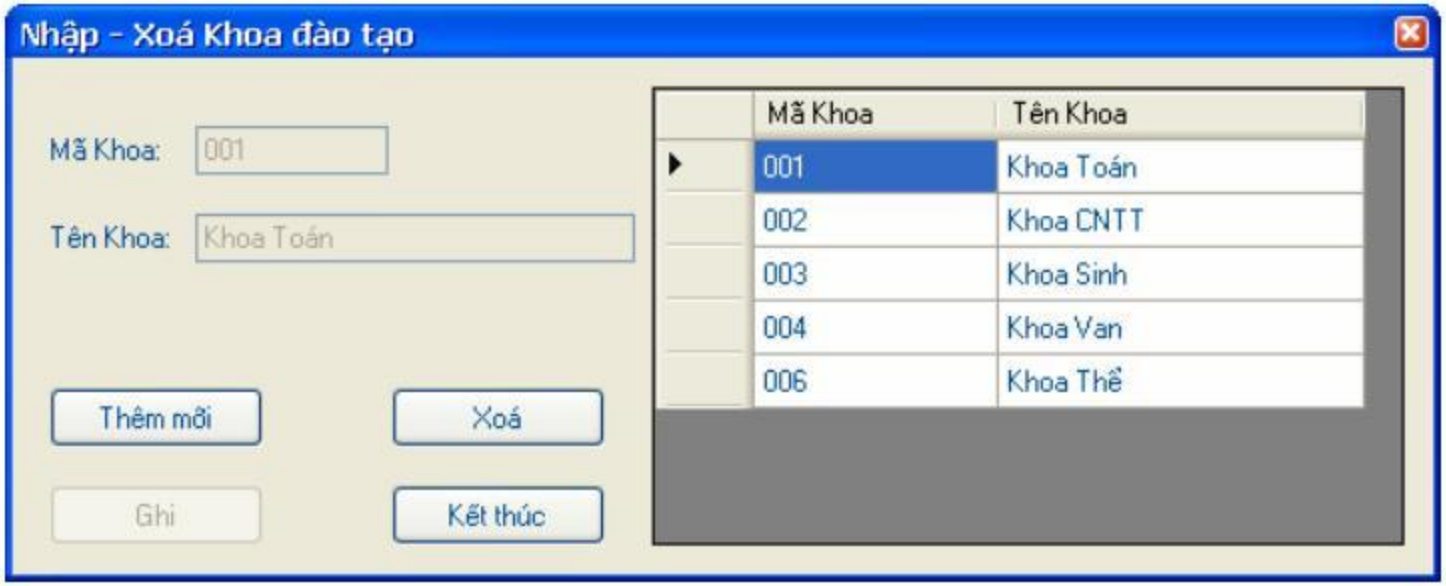
- Đối tượng SqlCommand
  - Dùng để thực hiện câu lệnh SQL
    - Insert
    - Update
    - Delete

Các đối tượng

- Đối tượng SqlCommand
  - 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

Các đối tượng

- Đối tượng SqlCommand
  - Thiết kế Form cho phép nhập, xoá bảng dữ liệu tblKhoaDaoTao



Các đối tượng

- Đối tượng SqlCommand

```
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;
}
```

Các đối tượng

- Đối tượng SqlCommand

```
private void Display()
{
    string SqlStr = "SELECT * FROM tblKhoaDaoTao";
    //Tao thong qua xau ket noi da mo
    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);
}
```

Các đối tượng

- Đối tượng SqlCommand

```
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();
}
```

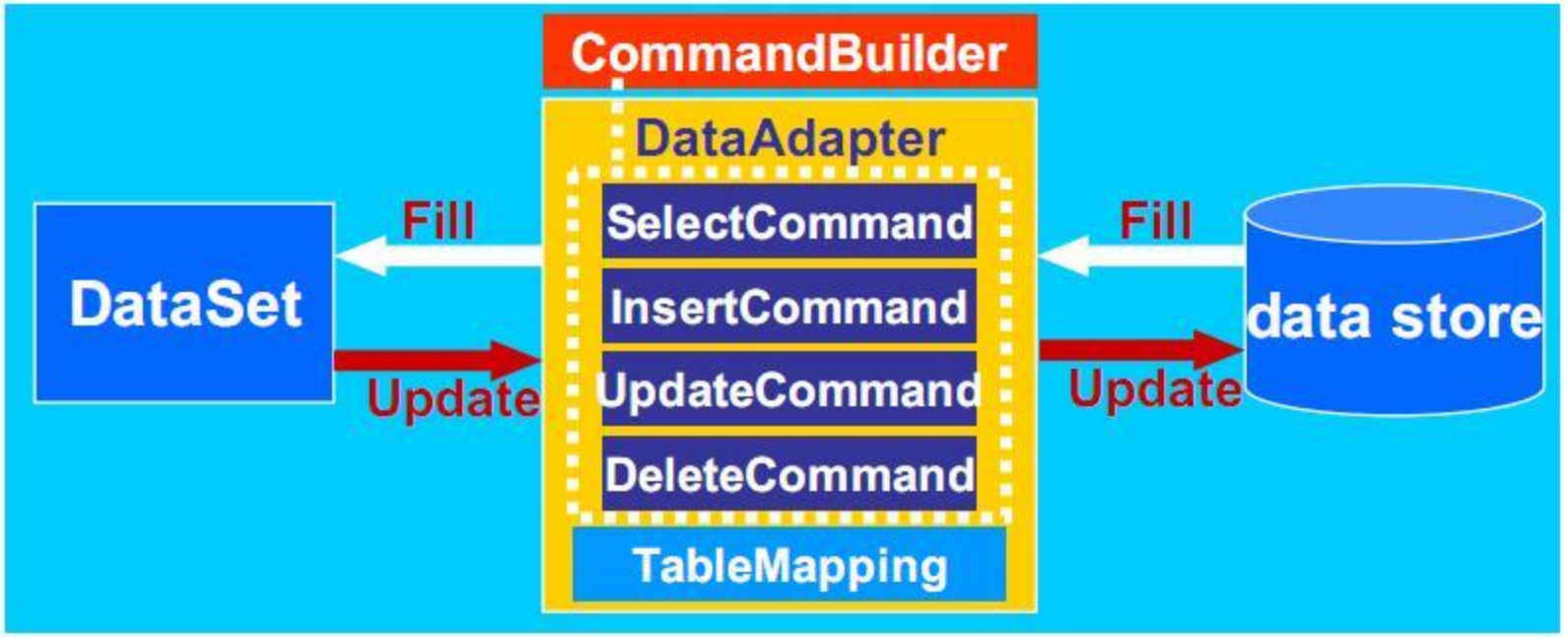
Đối tượng SqlCommand

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

Các đối tượng

- Đối tượng SqlCommandBuilder
  - Tự động thực hiện `Update, Insert, Delete`



Các đối tượng

- Đối tượng SqlCommandBuilder
  - 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;`

Các đối tượng

- Đối tượng SqlCommandBuilder
  - 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["...."];

Các đối tượng

- Đối tượng SqlCommandBuilder
  - Xoá một dòng
    - `myTable.Rows[pos].Delete();`
    - `myDataAdapter.Update(myTable);`
    - `Note: pos là dòng cần xoá`
  - Thêm một dòng
    - `DataRow newRow = myTable.NewRow();`
    - `newRow["MAKHOA"] = txtMakhoa.Text;`
    - `newRow["TENKHOA"] = txtTen.Text;`
    - `myTable.Rows.Add(newRow);`
    - `myDataAdapter.Update(myTable);`

Các đối tượng

- Đối tượng SqlCommandBuilder
  - Sửa một dòng
    - `DataRow editRow =myTable.Rows[pos];`
    - `editRow["MAKHOA"] = txtMakhoa.Text;`
    - `editRow["TENKHOA"] = txtTenkhoa.Text;`
    - `myDataAdapter.Update(myTable);`
    - `Note: pos là dòng cần sửa`
  - Loại bỏ sửa đổi dòng
    - `myTable.RejectChanges();`

Nạp dữ liệu vào các Cotrols

- Đối 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 **LixtBox**

Các đối tượng

- Đối tượng ListBox và ComboBox
  - **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

Các đối tượng

- Đối tượng ListBox và ComboBox
  - 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.SelectValue`