ĐẠI HỌC HUẾ
ĐẠI HỌC KHOA HỌC
KHOA CÔNG NGHỆ THÔNG TIN

GIÁO TRÌNH C# VÀ ỨNG DỤNG

NGUYỄN HOÀNG HÀ – NGUYỄN VĂN TRUNG

HUẾ - 2008

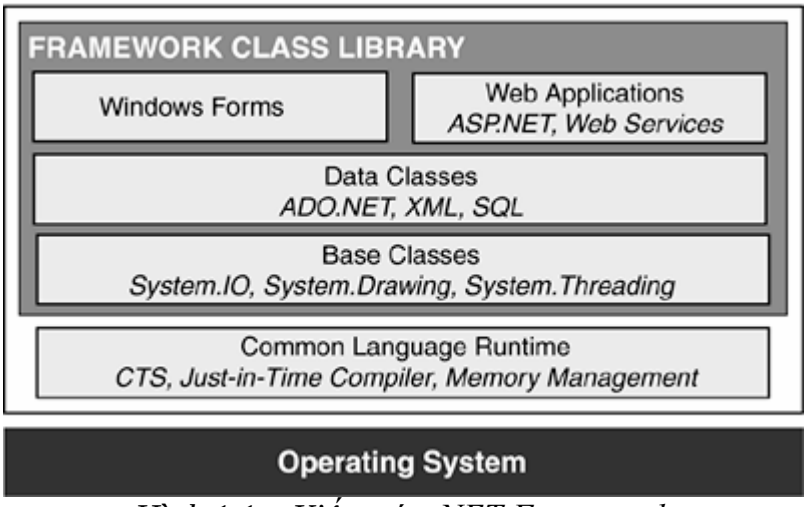
CHƯƠNG 1 TỔNG QUAN VỀ .NET FRAMEWORK

1.1 Tổng quan về kiến trúc của .NET Framework

.NET Framework được thiết kế như là môi trường tích hợp để đơn giản hóa việc phát triển và thực thi các ứng dụng trên Internet, trên desktop dưới dạng Windows Forms, hoặc thậm chí là trên cả các thiết bị di động (với Compact Framework). Các mục tiêu chính mà .NET framework hướng đến là:

- Cung cấp một môi trường hướng đối tượng nhất quán cho nhiều loại ứng dụng
- Cung cấp một môi trường giảm tối thiểu sự xung đột phiên bản ("DLL Hell" – Địa ngục DLL) từng làm điêu đứng các lập trình viên Windows (COM), và đơn giản hóa quá trình triển khai/cài đặt.
- Cung cấp một môi trường linh động, dựa trên các chuẩn đã được chứng nhận để có thể chứa trên bất cứ hệ điều hành nào. C# và một phần chính của môi trường thực thi .NET, CLI (Common Language Infrastructure – Hạ tầng ngôn ngữ chung) đã được chuẩn hóa bởi ECMA.
- Để cung cấp một môi trường quản lý được, trong đó mã được dễ dàng xác thực để thực thi an toàn.

Kiến trúc của .NET Framework được thiết kế thành 2 phần: CLR (Common Language Runtime – Khối thức thi ngôn ngữ chung) và FCL (Framework Class Library – Thư viện lớp khung) như hình dưới



Hình 1.1 – Kiến trúc .NET Framework

CLR, phần cài đặt CLI của Microsoft, làm nhiệm vụ quản lý sự thực thi mã lệnh và tất cả các tác vụ liên quan đến nó: biên dịch, quản lý bộ nhớ, bảo mật, quản lý tuyến đoạn, và thực thi an toàn kiểu. Mã lệnh thực thi trong CLR được gọi là *mã được quản lý (managed code)*, phân biệt với *mã không được quản lý (unmanaged code)*, là mã lệnh không cài đặt những yêu cầu để thực thi trong CLR – chẳng hạn như COM hoặc các thành phần dựa trên Windows API.

FCL là thư viện kiểu dữ liệu có thể tái sử dụng (gồm các class, structure, …) dành cho các ứng dụng thực thi trong .NET. Tất cả các ngôn ngữ hỗ trợ .NET Framework đều sử dụng thư viện lớp dùng chung này.

1.2 Môi trường thực thi ngôn ngữ chung CLR (Common Language Runtime)

CLR (Common Languge Runtime – Môi trường thực thi ngôn ngữ chung) quản lý toàn bộ vòng đời của một ứng dụng: nó nạp các lớp có liên quan, quản lý sự thực thi của các lớp, và đảm bảo quản lý bộ nhớ một cách tự động. Ngoài ra, CLR còn hỗ trợ tích hợp giữa các ngôn ngữ để cho phép mã lệnh được sinh ra bởi các ngôn ngữ khác nhau có thể tương tác với nhau một cách liền mạch.

1.2.1 Biên dịch mã lệnh .NET

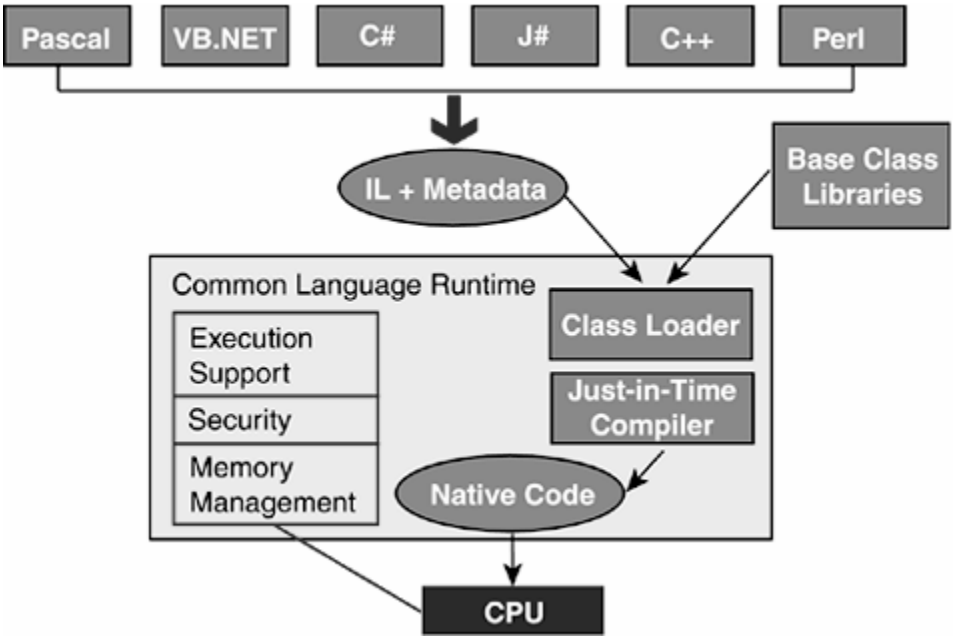
Trình biên dịch tương thích với CLR sẽ sinh mã thực thi cho môi trường thực thi chứ không phải là mã thực thi cho CPU cụ thể. Mã thực thi này được biết đến qua tên gọi CIL (Common Intermediate Language – Ngôn ngữ trung gian chung), hay MSIL (Microsoft Intermediate Language – Ngôn ngữ trung gian của Microsoft); đó là ngôn ngữ kiểu assembler được đóng gói trong các file EXE hoặc DLL. Các file này không phải thuộc dạng file có thể thực thi như thông thường, chúng cần *trình biên dịch JIT (Just-in-Time)* của *môi trường thực thi* để chuyển đổi IL chứa trong nó sang dạng *mã lệnh cụ thể của máy* khi ứng dụng thực sự thực thi.

Quá trình biên dịch, thực thi một chương trình trong .NET framework có thể tóm tắt như sau:

- Chương trình nguồn trước hết sẽ được biên dịch và đóng gói thành một khối gọi là assembly. Khối này sẽ chứa các mã lệnh ngôn ngữ trung gian và các metadata mô tả thông tin cần thiết cho sự hoạt động của khối.

- Mỗi khi có yêu cầu thực thi assembly nói trên, CLR sẽ chuyển đổi mã lệnh ngôn ngữ trung gian trong assembly thành mã lệnh tương thích với CPU cụ thể trước

khi có thể thực thi.

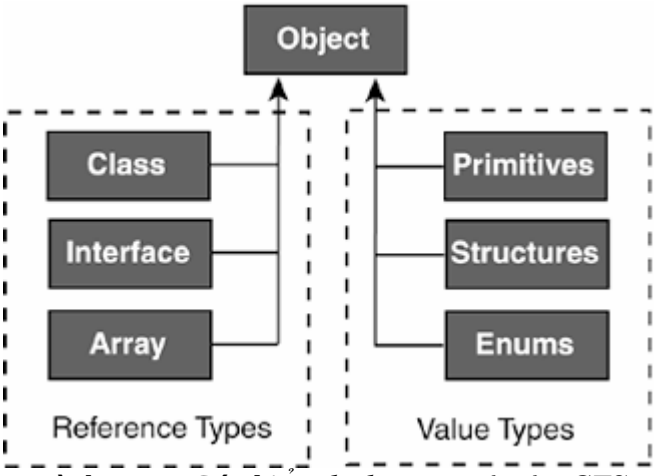


Hình 1.2 – Chức năng của CLR

Như vậy, lập trình viên có thể sử dụng bất cứ ngôn ngữ nào để phát triển ứng dụng trên .NET framework, miễn là ngôn ngữ đó có hỗ trợ .NET framework. Điều đặc biệt là, do sử dụng chung *hệ thống kiểu dữ liệu*, nên tính năng liên thông giữa các ngôn ngữ trên .NET framework là rất cao.

1.2.2 Hệ thống kiểu dữ liệu chung CTS (Common Type System)

CTS cung cấp một tập cơ sở các kiểu dữ liệu cho mỗi ngôn ngữ hoạt động trên .NET platform. Ngoài ra, nó đặc tả cách khai báo và tạo các kiểu dữ liệu tùy biến, cách quản lý vòng đời của một thể hiện của những kiểu dữ liệu này. Hình dưới đây mô tả cách tổ chức CTS của .NET



Hình 1.3 – Các kiểu dữ liệu cơ sở của CTS

Mọi kiểu dữ liệu trong .NET đều được kế thừa từ kiểu dữ liệu `System.Object`. Các kiểu dữ liệu được chia làm hai loại: kiểu tham chiếu và kiểu giá trị. Kiểu dữ liệu tham chiếu được xử lý trong một vùng nhớ đặc biệt gọi là heap thông qua các con trỏ. Kiểu dữ liệu giá trị được tham chiếu trực tiếp trong stack của chương trình.

1.2.3 Assemblies

Tất cả các mã được quản lý thực thi trong .NET đều phải được chứa trong một assembly. Một assembly được xem như là một file EXE hoặc DLL. Một asembly có thể chứa một tập hợp gồm một hay nhiều file chứa phần mã lệnh hoặc tài nguyên (như ảnh hoặc dữ liệu XML).

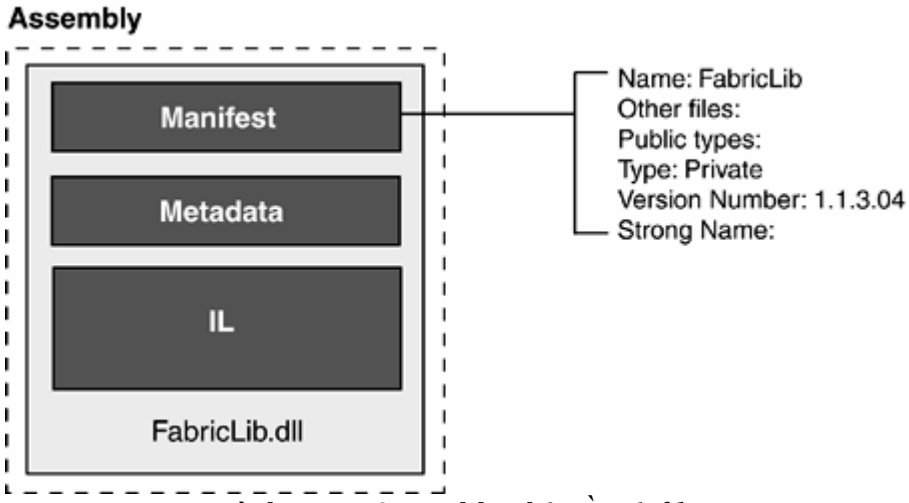
Một assembly được tạo ra khi trình biên dịch tương thích với .NET chuyển một file chứa mã nguồn thành một file DLL hoặc EXE. Như minh họa trong hình 1.4, một assembly chứa một manifest, metadata, và ngôn ngữ trung gian sinh bởi trình biên dịch cụ thể.

Manifest: Mỗi assembly phải có một file chứa một manifest. Manifest này là một tập hợp các bảng chứa các metadata trong đó liệt kê tên của tất cả các file trong assembly, tham chiếu đến các assembly bên ngoài, và các thông tin như tên, phiên bản để định danh assembly đó. Một số assembly còn có cả *chữ ký điện tử duy nhất* (unique digital signature). Khi một assembly được nạp, nhiệm vụ đầu tiên của CLR là mở file chứa manifest để có thể định danh các thành viên có trong assembly.

Metadata: Ngoài các bảng trong manifest vừa định nghĩa, trình biên dịch C# còn sinh ra các bảng định nghĩa và bảng tham chiếu. Bảng định nghĩa cung cấp một ghi chú đầy đủ

về các kiểu chứa trong IL. Ví dụ, có các bảng định nghĩa kiểu, phương thức, trường dữ liệu, tham số, và thuộc tính. Bảng tham chiếu chứa các thông tin về tất cả các tham chiếu về kiểu và các assembly khác. Trình biên dịch JIT phụ thuộc vào các bảng này để chuyển IL sang mã máy.

IL: Vai trò của IL đã được đề cập trước đây. Trước khi CLS có thể sử dụng IL, nó phải được đóng gói vào trong một assembly dạng DLL hoặc EXE. Assembly dạng EXE phải có một điểm nhập (entry point) để nó có thể thực thi. Ngược lại, Assembly dạng DLL, được thiết kế để hoạt động như là một thư viện mã lệnh nắm giữ các định nghĩa kiểu.



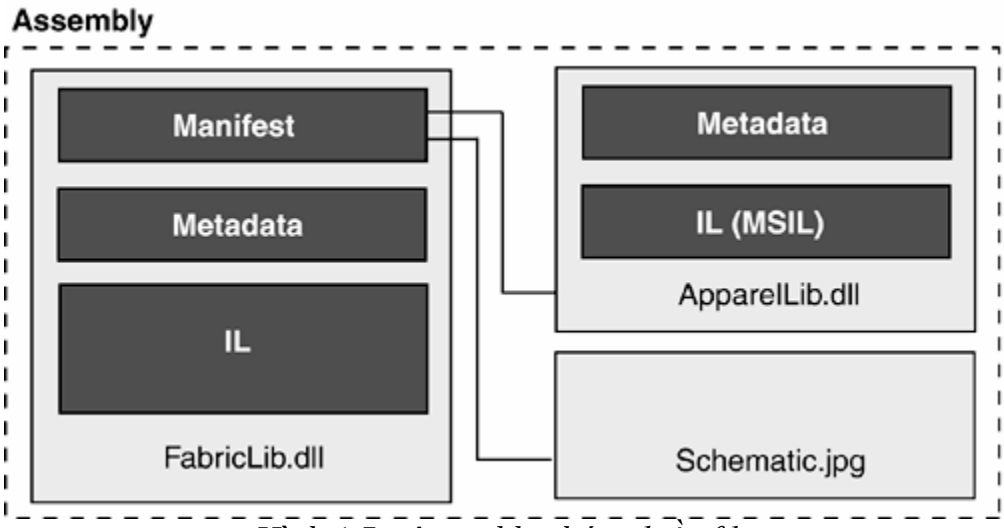
Hình 1.4 – Assembly chỉ gồm 1 file

Assembly không chỉ là cách logic để đóng gói các mã thực thi. Nó quy định mô hình chủ yếu của .NET để triển khai mã lệnh, quản lý phiên bản, và bảo mật.

- Tất cả các mã được quản lý, cho dù là một chương trình đơn, một điều khiển, hay một thư viện DLL chứa các kiểu dữ liệu tái sử dụng, đều được đóng gói vào một assembly. Đây là khối cơ bản nhất có thể triển khai trên hệ thống. Khi một ứng dụng được bắt đầu, chỉ những assembly được yêu cầu cho việc khởi tạo mới cần hiện diện. Các assembly khác sẽ được nạp khi có yêu cầu. Các nhà phát triển có thể phân ứng dụng thành các assembly dựa theo mức độ thường xuyên sử dụng.
- Trong thế giới .NET, một assembly quy định một biên giới phiên bản. Trường Version Number trong manifest áp dụng cho tất cả các kiểu và tài nguyên trong assembly. Vì vậy, mọi file tạo nên assembly được xem như là một đơn vị đơn nhất có cùng phiên bản.
- Một assembly cũng thiết lập một biên giới bảo mật để định ra quyền hạn truy xuất.

C# sử dụng các bổ từ truy cập để điều khiển cách mà các kiểu và thành phần kiểu trong một assembly được truy xuất. Hai trong số này được sử dụng trong assembly, đó là public – cho phép truy xuất tùy ý từ assembly bất kỳ ; và internal – giới hạn truy xuất đến các kiểu và thành viên bên trong assembly.

Như đã đề cập ở trên, một assembly có thể chứa nhiều file. Những file này không giới hạn là các module mã lệnh mà có thể là các file tài nguyên như file hình ảnh hoặc văn bản. Một cách sử dụng tính chất này trong thực tế đó là chúng ta có thể tạo ra ứng dụng đa ngôn ngữ, trong đó ứng dụng sẽ cùng sử dụng chung các module logic, phần giao diện hoặc các tài nguyên khác có thể được triển khai riêng thành các file độc lập. Không có giới hạn về số lượng file trong một assembly. Hình 1.5 minh họa bố cục của một assembly chứa nhiều file.



Hình 1.5 - Assembly chứa nhiều file

Trong minh họa assembly chứa nhiều file, manifest của assembly chứa thông tin để định danh mọi file được sử dụng trong assembly.

Mặc dù hầu hết các assembly đều chứa một file duy nhất. Sau đây là các thuận lợi của assembly chứa nhiều file:

- Có thể tổ hợp các module được tạo ra từ nhiều ngôn ngữ lập trình khác nhau.
- Các module mã lệnh có thể được phân ra để tối ưu cách mà mã lệnh được nạp vào trong CLR. Các mã lệnh có liên quan và được sử dụng thường xuyên nên được đặt vào trong cùng một module; những mã lệnh ít khi được sử dụng sẽ được đặt vào trong module khác. CLR không nạp các module nào khi chưa thực sự cần thiết.

- Các file tài nguyên có thể được đặt vào trong module của riêng nó, qua đó cho phép nhiều ứng dụng có thể chia sẻ tài nguyên dùng chung.

1.2.4 Private Assembly và Shared Assembly

Các assembly có thể được triển khai theo hai dạng: private assembly và global assembly.

Private assembly là assembly được đặt trong thư mục của ứng dụng hoặc thư mục con của nó. Quá trình cài đặt và cập nhật private assembly chỉ đơn giản là chép assembly vào trong thư mục cần thiết, không cần thiết lập thông tin trong registry. Đôi khi, có thể dùng thêm một file cấu hình ứng dụng có thể ghi đè một số thiết lập trong manifest của ứng dụng.

Shared assembly là assembly được cài đặt vào vị trí toàn cục, gọi là Global Assembly Cache (GAC), là nơi có thể truy xuất được từ nhiều ứng dụng. Điểm quan trọng nhất của GAC đó là nó cho phép nhiều phiên bản của một assembly có thể được thực thi. Để hỗ trợ điều này, .NET khắc phục vấn đề xung đột tên bằng cách sử dụng 4 thuộc tính để định danh 1 assembly, bao gồm: Assembly Name (tên assembly), Culture Identity (định danh văn hóa), Version (phiên bản), và Public Key Token (dấu hiệu mã khóa công khai).

Các shared assembly thường được đặt trong thư mục assembly ở dưới thư mục hệ thống của hệ điều hành (WINNT\ trong Windows 2000, WINDOWS\ trong Windows XP). Như mô tả ở hình 1.6, các assembly được liệt kê theo định dạng đặc biệt để hiển thị 4 thuộc tính của chúng (.NET Framework bao gồm một file DLL để mở rộng Windows Explorer cho phép nó có thể hiển thị nội dung GAC).

- Assembly Name: còn được gọi là tên thường gọi, là tên file của assembly không chứa phần mở rộng.
- Version: Mỗi assembly có một số hiệu phiên bản để dùng cho tất cả các file trong assembly. Nó chứa 4 số theo định dạng:

```
<major number>.<minor number>.<build>.<revision>
```

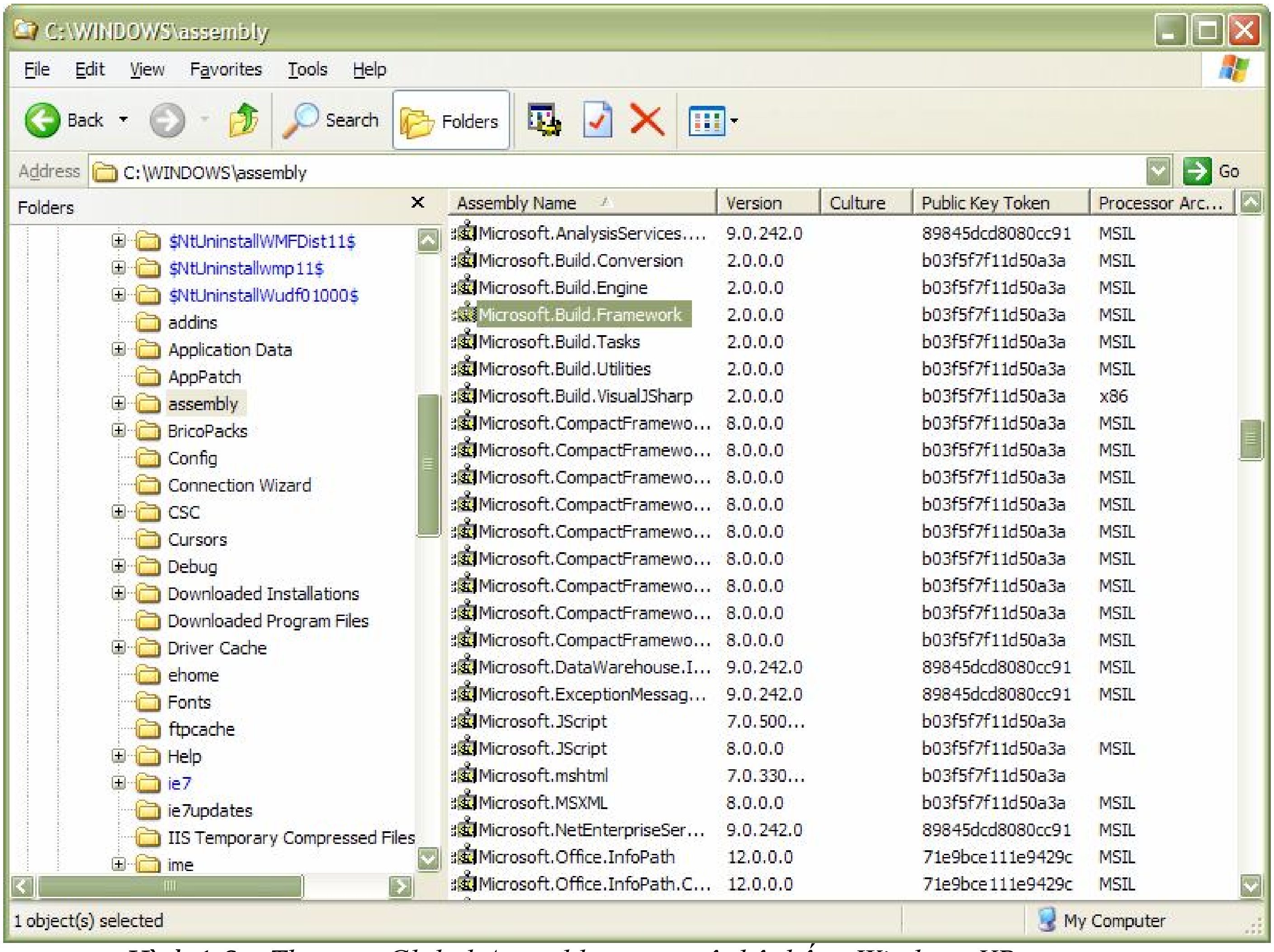
  Thông thường các số <major number> và <minor number> được cập nhật cho những lần thay đổi mang tính phá vỡ tính tương thích ngược. Một số hiệu phiên bản có thể được gán cho một assembly bằng cách đính thuộc tính AssemblyVersion trong phần mã nguồn của assembly.
- Culture Setting: Nội dung của một assembly có thể được kết hợp với một văn hóa hay ngôn ngữ cụ thể. Thiết lập này được chỉ định bằng mã hai ký tự kiểu như “en” cho English, “vi” cho Vietnam, và có thể được gán với thuộc tính

AssemblyCulture đặt trong mã nguồn của assembly

```
[assembly: AssemblyCulture ("fr-CA")]
```

- Public Key Token: Để đảm bảo một shared assembly là duy nhất và đáng tin cậy, .NET yêu cầu người tạo ra assembly phải đánh dấu bằng một định danh mạnh. Quá trình này được gọi là ký, yêu cầu sử dụng cặp khóa công khai/riêng tư. Khi trình biên dịch xây dựng assembly, nó sẽ sử dụng khóa riêng tư để sinh ra một định danh mạnh. Token được sinh ra ở đây là 8 byte cuối cùng của phép băm (hashing) khóa công khai. Token này sẽ được đặt trong manifest của bất kỳ assembly client nào có tham chiếu đến shared assembly và sử dụng nó để định danh assembly trong quá trình thực thi.

  Một assembly được gán một cặp khóa công khai/riêng thì được gọi là một assembly định danh mạnh. Mọi assembly đều phải có định danh mạnh.



Hình 1.6 – Thư mục Global Assembly trong một hệ thống Windows XP

1.2.5 Tiền biên dịch một Assembly

Sau khi một assembly được nạp vào CLR, IL phải được biên dịch sang thành mã máy trước khi thực sự được thực thi. .NET Framework có cung cập một công cụ gọi là Ngen (Native Image Generator), dùng để biên dịch một assembly thành một "native image" được lưu trong native image cache – một vùng dành riêng của GAC. Mỗi khi CLR nạp một assembly, nó sẽ kiểm tra trong cache xem đã có native image tương ứng chưa; nếu có nó sẽ nạp mã đã biên dịch đó chứ không cần biên dịch thêm lần nữa. Đây là tính năng mà nếu được khai thác hợp lý thì có thể tận dụng để cải thiện hiệu năng

1.2.6 Kiểm chứng mã lệnh (Code Verification)

Như là một phần của quá trình biên dịch JIT, CLR thực hiện hai loại kiểm chứng: kiểm

chứng IL và hợp lệ hóa metadata để bảo đảm mã lệnh được an toàn kiểu. Trong thực tế, điều này có nghĩa là các tham số trong *lời gọi* và *phương thức được gọi* phải được kiểm tra để đảm bảo chúng có cùng kiểu dữ liệu, hoặc là một phương thức chỉ trả về đúng kiểu được đặc tả trong khai báo trả về. Nói ngắn gọn, CLR sẽ xem xét trong IL và metadata để đảm bảo mọi giá trị được gán cho một biến là tương thích kiểu; nếu không sẽ có một ngoại lệ xuất hiện.

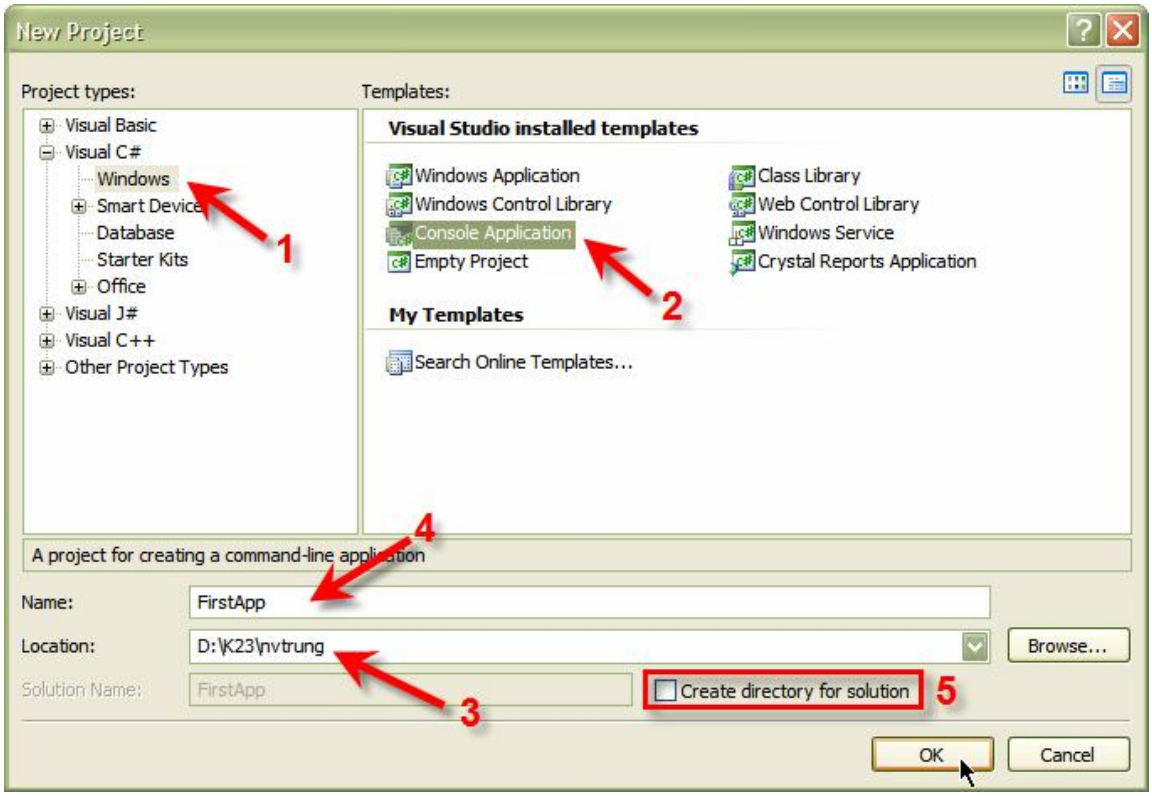
Thuận lợi của mã lệnh được kiểm chứng đó là CLR có thể chắc chắn mã lệnh sẽ không ảnh hưởng đến ứng dụng khác theo kiểu truy xuất đến vùng nhớ ngoài vùng cho phép của nó. Do đó CLR tự do thực thi nhiều ứng dụng trong cùng một tiến trình hay không gian địa chỉ.

CHƯƠNG 2 NGÔN NGỮ LẬP TRÌNH C#

2.1 Chương trình đầu tiên

Chúng ta sẽ làm quen với ngôn ngữ lập trình C# và *môi trường tích hợp phát triển* (IDE – Integrated Development Environment) Visual Studio .NET bằng cách xây dựng một ứng dụng đầu tiên, ứng dụng firstApp. Ứng dụng này cho phép người sử dụng nhập vào 2 số, sau đó in ra màn hình tổng, tích và thương của hai số vừa nhập. Trình tự thực hiện như sau:

1. Khởi động Microsoft Visual Studio 2005. Nhấn Ctrl + Shift + N hoặc chọn menu tương ứng là File → New → Project để tạo mới một project

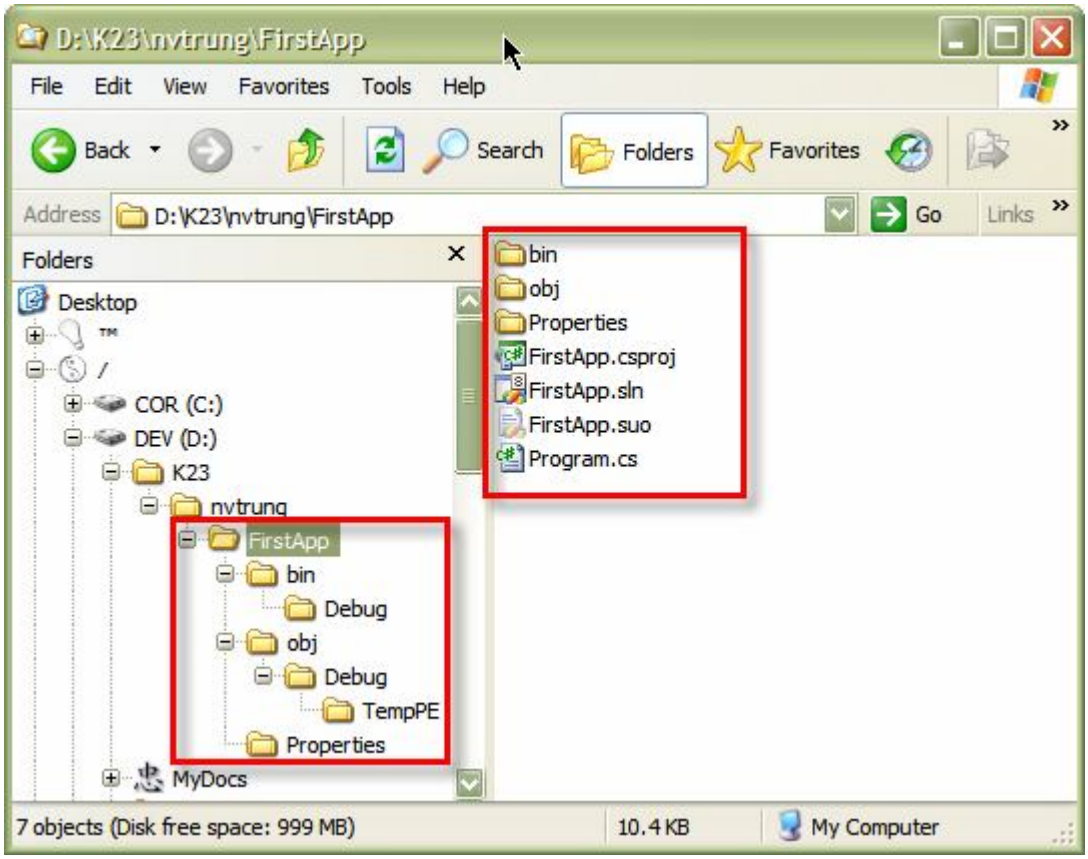


2. Chọn loại ứng dụng cần phát triển là Visual C# → Windows → Console Application. Sau đó, chọn thư mục chứa project và đặt tên cho project như minh họa ở hình trên. Chú ý, ở đây, chúng ta bỏ chọn ở hộp kiểm "Create directory for solution".

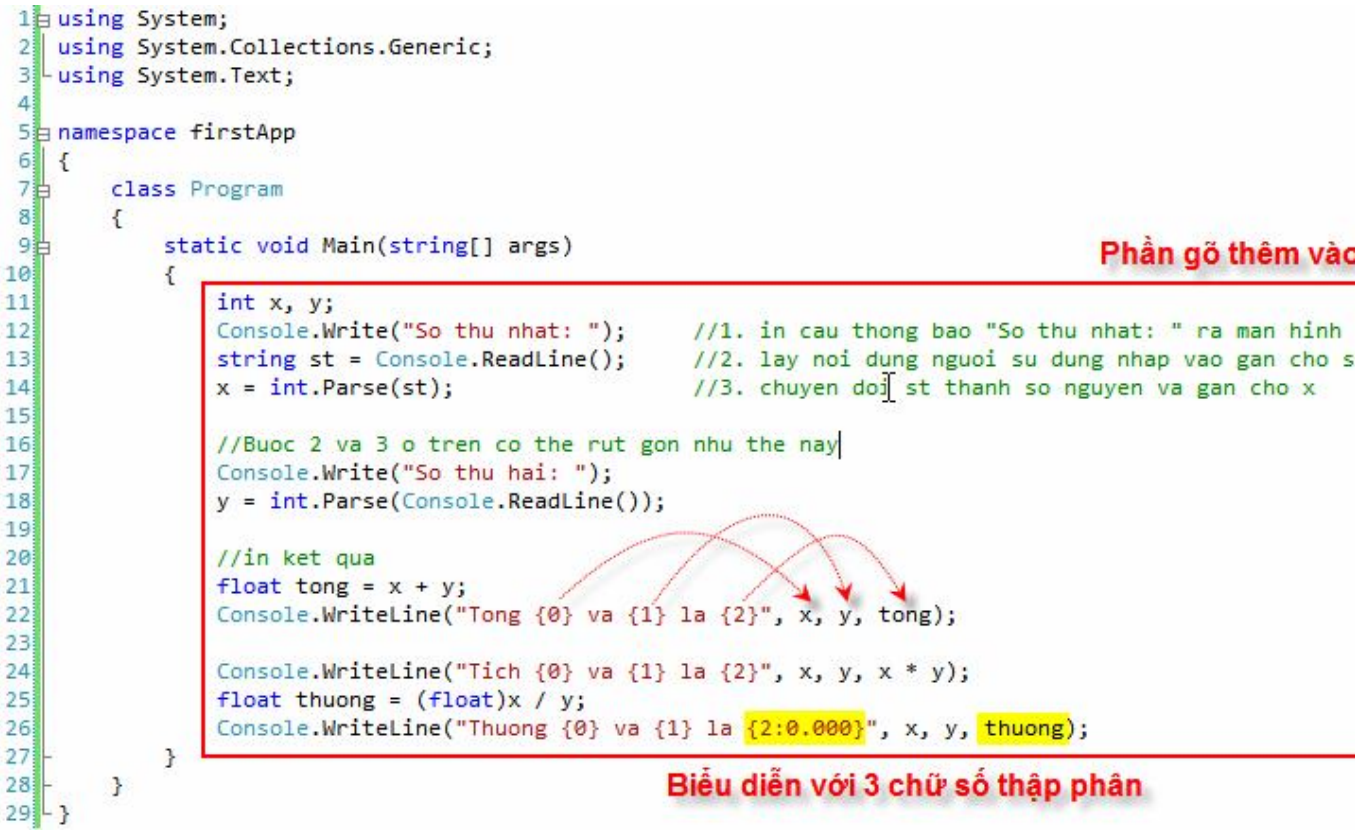
   Chú thích:

Visual Studio .NET coi một "bài toán" cần giải quyết là một solution. Một solution có thể bao gồm một hoặc nhiều project. Một solution, nếu có nhiều project thì nên được tạo ra trong một thư mục riêng để có thể chứa các project trong nó. Ở đây, solution chỉ có duy nhất một project, thế nên không cần thiết phải tạo ra một thư mục cho solution.

3. Sau khi nhấn nút OK, hãy khảo sát xem cấu trúc của thư mục chứa solution của chúng ta. *Bạn phải luôn nắm chắc về sự tồn tại, ý nghĩa của các tập tin, thư mục được tạo ra trong quá trình làm việc!*



4. Gõ mã lệnh như minh họa vào trong phần mã nguồn của tập tin Program.cs




```
using System;
using System.Collections.Generic;
using System.Text;

namespace firstApp
{
    class Program
    {
        static void Main(string[] args)
        {
            int x, y;
            Console.Write("So thu nhat: ");     //1. in cau thong bao "So thu nhat: " ra man hinh
            string st = Console.ReadLine();     //2. lay noi dung nguoi su dung nhap vao gan cho s
            x = int.Parse(st);                  //3. chuyen doi st thanh so nguyen va gan cho x

            //Buoc 2 va 3 o tren co the rut gon nhu the nay
            Console.Write("So thu hai: ");
            y = int.Parse(Console.ReadLine());

            //in ket qua
            float tong = x + y;
            Console.WriteLine("Tong {0} va {1} la {2}", x, y, tong);

            Console.WriteLine("Tich {0} va {1} la {2}", x, y, x * y);
            float thuong = (float)x / y;
            Console.WriteLine("Thuong {0} va {1} la {2:0.000}", x, y, thuong);
        }
    }
}
```

5. Bạn có thể sử dụng MSDN để tra cứu các thông tin bạn chưa biết về:
    a. Lớp Console và các phương thức ReadLine(), WriteLine() của nó
    b. Cách chuyển đổi kiểu chuỗi thành số, ví dụ như int.Parse()
6. Nhấn Ctrl + F5 để thực hiện biên dịch và chạy chương trình. Sau đó quan sát cấu trúc thư mục của solution, cho biết sự thay đổi của nó so với khi mới được tạo ra ở bước 3 (xem thư mục bin và thư mục obj của project).
7. Thử thay đổi kết câu lệnh

```
float thuong = (float)x / y;
```

thành

```
float thuong = x / y;
```

rồi chạy chương trình, quan sát kết quả và rút ra kết luận.

8. Sử dụng thêm các cấu trúc lệnh khác để tinh chỉnh hoạt động của chương trình (xử lý phép chia cho 0, …)

Rõ ràng, đoạn chương trình đơn giản trên không phải là quá phức tạp đối với người đã

từng làm quen với các ngôn ngữ lập trình bậc cao.

2.2 Biến dữ liệu

Biến trong C# được khai báo theo cú pháp như sau:

```
datatype identifier;
```

Ví dụ:

```
int i;
```

Câu lệnh này khai báo một số int tên là i. Trình biên dịch thực sự chưa cho phép sử dụng biến này cho đến khi chúng ta khởi tạo nó bằng một giá trị. Lệnh khai báo này chỉ làm nhiệm vụ cấp phát một vùng nhớ (4 bytes) cho biến i.

Sau khi khai báo, chúng ta có thể gán một giá trị cho biến bằng toán tử gán =, như sau:

```
i = 10;
```

Chúng ta cũng có thể vừa khai báo, vừa khởi tạo giá trị cho biến cùng lúc:

```
int i = 10; // khai bao va khoi tao gia tri cho bien int
double x = 10.25, y = 20;   // khai bao va khoi tao hai bien double
```

2.2.1 Tầm hoạt động của biến

Tầm hoạt động của một biến là vùng mã lệnh mà trong đó biến có thể truy xuất. Nói chung, tầm hoạt động của biến được xác định theo các quy tắc sau:

- Một trường dữ liệu (field), còn được gọi là một biến thành phần của một lớp đối tượng sẽ có tầm hoạt động trong phạm vi lớp chứa nó.
- Một biến cục bộ sẽ có tầm hoạt động trong khối khai báo nó (trong cặp dấu ngoặc nhọn { })
- Một biến cục bộ được khai báo trong các lệnh lặp for, while, … sẽ có tầm hoạt động trong thân vòng lặp

Tất nhiên, trong cùng phạm vi hoạt động, không được có hai biến có trùng tên.

2.2.2 Hằng dữ liệu

Hằng dữ liệu là biến có giá trị không được phép thay đổi trong suốt thời gian tồn tại của nó. Cách khai báo của hằng dữ liệu là tương tự như đối với biến dữ liệu, chỉ khác là được thêm từ khóa const ở đầu.

```
const int a = 100;   // Gia tri nay khong duoc thay doi
```

Hằng dữ liệu có các đặc tính sau:

- Phải được khởi tạo ngay khi nó được khai báo, sau đó không được phép thay đổi giá trị của hằng.
- Giá trị của hằng dữ liệu phải được tính toán trong thời điểm biên dịch. Vì vậy, chúng ta không thể khởi tạo một hằng số có giá trị được lấy từ một biến dữ liệu. Nếu cần điều này, chúng ta sử dụng trường dữ liệu kiểu read-only.

2.3 Các kiểu dữ liệu định nghĩa sẵn của C#

C# phân kiểu dữ liệu thành hai loại (tương tự như cách phân loại chung trong CTS): kiểu dữ liệu giá trị và kiểu dữ liệu tham chiếu. Về mặt khái niệm, điểm khác biệt giữa hai kiểu dữ liệu này đó là, biến kiểu dữ liệu giá trị lưu giữ trực tiếp một giá trị, trong khi đó, biến kiểu tham chiếu lưu giữ tham chiếu đến một giá trị dữ liệu. Về mặt lưu trữ vật lý, biến của hai kiểu dữ liệu này được lưu vào hai vùng nhớ khác nhau của chương trình, đó là vùng nhớ stack (cho biến dữ liệu kiểu giá trị) và vùng nhớ heap (cho biến dữ liệu kiểu tham chiếu). Bạn cần đặc biệt lưu ý hiệu ứng của các phép gán đối với kiểu dữ liệu kiểu tham chiếu.

2.3.1 Kiểu dữ liệu giá trị được định nghĩa sẵn

Các kiểu dữ liệu giá trị được định nghĩa sẵn bao gồm số nguyên, số dấu chấm phẩy động, ký tự và boolean.

2.3.1.1 Các kiểu số nguyên

C# hỗ trợ sẵn 8 kiểu số nguyên:

Tên	Kiểu trong CTS	Mô tả	Vùng biểu diễn (min:max)
sbyte	System.SByte	Số nguyên có dấu 8-bit	$-2^7:2^7-1$
short	System.Int16	Số nguyên có dấu 16-bit	$-2^{15}:2^{15}-1$
int	System.Int32	Số nguyên có dấu 32-bit	$-2^{31}:2^{31}-1$
long	System.Int64	Số nguyên có dấu 64-bit	$-2^{63}:2^{63}-1$
byte	System.Byte	Số nguyên không dấu 8-bit	$0:2^8-1$
ushort	System.UInt16	Số nguyên không dấu 16-bit	$0:2^{16}-1$
uint	System.UInt32	Số nguyên không dấu 32-bit	$0:2^{32}-1$

ulong	System.UInt64	Số nguyên không dấu 64-bit	0:2^{64}-1

2.3.1.2 Các kiểu số dấu chấm động

Các kiểu số thực dấu chấm động được hỗ trợ sẵn của C# bao gồm:

Tên	Kiểu trong CTS	Số chữ số có nghĩa	Vùng biểu diễn tương đối (khoảng)
float	**System.Single**	**7**	**±1.5 × 10-45 to ±3.4 × 1038**
double	**System.Double**	**15/16**	**±5.0 × 10-324 to ±1.7 × 10308**

2.3.1.3 Kiểu số thập phân

Để biểu diễn số thập phân với độ chính xác cao hơn số thực dấu chấm động, C# hỗ trợ kiểu dữ liệu số thập phân:

Tên	Kiểu trong CTS	Số chữ số có nghĩa	Vùng biểu diễn
decimal	**System.Decimal**	**28**	**$\pm 1.0 \times 10^{-28}$ to $\pm 7.9 \times 10^{28}$**

2.3.1.3 Kiểu boolean

Tương ứng với System.Boolean trong CTS, C# có kiểu dữ liệu bool, có thể nhận một trong hai giá trị true hoặc false. Có một điều lưu ý, kiểu dữ liệu bool không được nhận các giá trị nguyên như một số ngôn ngữ (C, C++)

2.3.1.4 Kiểu ký tự

Để lưu trữ giá trị của một ký tự đơn, C# hỗ trợ dữ liệu kiểu ký tự

Tên	Kiểu trong CTS	Giá trị
char	System.Char	Biểu diễn 1 ký tự 16-bit (Unicode)

Các hằng kiểu ký tự được gán bằng cách đóng trong cặp dấu nháy đơn, ví dụ 'A'. Cũng có thể biểu thị hằng ký tự dưới dạng số thập lục phân, kiểu như '\u0041', hoặc ép kiểu như (char)65. Ngoài ra có thể sử dụng một số ký tự escape sau:

Ký tự escape	Ký tự tương ứng
\'	Dấu nháy đơn
\"	Dấu nháy đôi
****	Ký tự \
\0	Null
\a	Ký tự Alert
\b	Ký tự Backspace
\f	Ký tự Form feed
\n	Ký tự xuống dòng
\r	Ký tự Carriage return

	\t	Ký tự Tab
	\v	Ký tự Vertical tab

2.3.2 Kiểu dữ liệu tham chiếu được định nghĩa sẵn

C# hỗ trợ sẵn hai kiểu dữ liệu tham chiếu:

Tên	Kiểu CTS	Mô tả
object	System.Object	Kiểu dữ liệu gốc, mọi kiểu dữ liệu khác trong CTS đều kế thừa từ đây (kể cả các kiểu dữ liệu giá trị)
string	System.String	Chuỗi ký tự Unicode

2.3.2.1 Kiểu dữ liệu object

object là kiểu dữ liệu gốc, cơ bản nhất mà từ đó, tất cả các kiểu dữ liệu khác đều phải kế thừa (trực tiếp hoặc gián tiếp). Các thuận lợi chúng ta có được từ kiểu dữ liệu object là:

- Chúng ta có thể sử dụng tham chiếu đối tượng để gắn kết với một đối tượng của bất kỳ kiểu dữ liệu con nào. Tham chiếu đối tượng cũng được sử dụng trong những trường hợp mà mã lệnh phải truy xuất đến những đối tượng chưa rõ kiểu dữ liệu (tương tự như vai trò con trỏ void ở C++)

- Kiểu object có cài đặt một số phương thức cơ bản, dùng chung, bao gồm: Equals(), GetHashCode(). GetType(), và ToString(). Các lớp do người sử dụng tự định nghĩa có thể cài đặt lại các phương thức này theo kỹ thuật gọi là overriding (ghi đè) trong lập trình hướng đối tượng.

2.3.2.2 Kiểu dữ liệu string

Kiểu dữ liệu string được cung cấp sẵn trong C# với nhiều phép toán và cách thức hoạt động thuận tiện là một trong những kiểu dữ liệu được sử dụng nhiều nhất khi lập trình. Đối tượng string được cấp phát trong vùng nhớ heap, và khi gán một biến string cho một biến khác, chúng ta sẽ có hai tham chiếu đến cùng một chuỗi trong bộ nhớ. Tuy nhiên, khi thay đổi nội dung của một trong các chuỗi này, chuỗi thay đổi sẽ được tạo mới hoàn toàn, không ảnh hưởng đến các chuỗi khác. Hãy xem hiệu ứng này trong đoạn chương trình dưới đây:

```
using System;

class MinhHoaString
{
   public static int Main()
   {
      string s1 = "a string";
      string s2 = s1;
```

```
      Console.WriteLine("s1 is " + s1);
      Console.WriteLine("s2 is " + s2);
      s1 = "another string";
      Console.WriteLine("s1 is now " + s1);
      Console.WriteLine("s2 is now " + s2);
      return 0;
   }
}
```

Kết quả của đoạn chương trình trên là:

```
s1 is a string
s2 is a string
s1 is now another string
s2 is now a string
```

Nói cách khác, việc thay đổi giá trị của s1 không ảnh hưởng gì đến s2, ngược với những gì chúng ta trông đợi ở kiểu dữ liệu tham chiếu.

Hằng kiểu chuỗi được bao trong cặp dấu nháy kép ("…"). Trong chuỗi có thể chứa các dãy ký tự escape như đối với kiểu dữ liệu ký tự. Do dãy ký tự escape được bắt đầu bằng ký tự \ nên ký tự \ phải được lặp đôi:

```
string filepath = "C:\\CSharp\\MinhHoaString.cs";
```

Có một giải pháp khác để biểu diễn ký tự \ trong chuỗi, đó là dùng cú pháp @:

```
string filepath = @"C:\CSharp\MinhHoaString.cs";
```

Cú pháp này còn cho phép chúng ta ngắt dòng trong hằng chuỗi, như sau:

```
string st = @"'Day la dong thu nhat
Day la dong thu hai.";
```

Khi đó, giá trị của chuỗi st sẽ là:

```
'Day la dong thu nhat
Day la dong thu hai.
```

2.4 Luồng điều khiển chương trình

2.4.1 Câu lệnh điều kiện

Các câu lệnh điều kiện cho phép phân nhánh mã lệnh theo các điều kiện cụ thể. C# có hai cấu trúc phân nhánh if và switch.

2.4.1.1 Câu lệnh if

Câu lệnh if của C# được kế thừa từ cấu trúc if của C và C++. Cú pháp của nó là:

```
if (condition)
   statement1(s)
[else
   statement2(s)]
```

Nếu có nhiều hơn một câu lệnh được thực thi tương ứng với một trong hai giá trị của biểu thức logic condition, chúng ta có thể gộp các lệnh này trong cặp dấu ngoặc nhọn ({ ... }) (điều này cũng được áp dụng cho nhiều cấu trúc lệnh khác mà chúng ta sẽ đề cập sau này):

```
bool isZero;
if (i == 0)
{
   isZero = true;
   Console.WriteLine("i is Zero");
}
else
{
   isZero = false;
   Console.WriteLine("i is Non-zero");
}
```

Điều đáng lưu ý nhất khi sử dụng câu lệnh if đó là condition nhất thiết phải là một biểu thức logic (chứ không thể là một số như ở C/C++).

2.4.1.2 Câu lệnh switch

Câu lệnh switch là một câu lệnh điều khiển quản lý nhiều lựa chọn và liệt kê bằng cách chuyển điều khiển đến một trong những câu lệnh **case** trong thân của nó:

```
switch (expression)
{
    case const_1:
        statement_1;
        break;
    case const_2:
        statement_2;
        break;
    ...
    case const_n:
        statement_n;
        break;
    [default:
        statement_n+1;
        break;]
```

```
}
```

Chú ý rằng, điều khiển được chuyển đến nhánh rẽ tương ứng với giá trị của biểu thức. Câu lệnh switch có thể chứa nhiều nhánh rẽ nhưng không có hai nhánh rẽ nào được có cùng giá trị. Việc thực thi thân câu lệnh được bắt đầu tại nhánh được lựa chọn và tiếp tục cho đến khi được chuyển ra ngoài qua lệnh break. Câu lệnh nhảy break là bắt buộc đối với mỗi nhánh rẽ, ngay cả khi đó là nhánh rẽ cuối cùng hoặc là nhánh rẽ default.

Nếu biểu thức không ứng với nhánh nào của lệnh switch thì điều khiển sẽ được chuyển đến các câu lệnh sau nhãn default (nếu có). Nếu không có nhãn default, điều khiển được chuyển ra bên ngoài câu lệnh switch.

Ví dụ:

```
switch (integerA)
{
   case 1:
      Console.WriteLine("integerA =1");
      break;
   case 2:
      Console.WriteLine("integerA =2");
      break;
   case 3:
      Console.WriteLine("integerA =3");
      break;
   default:
      Console.WriteLine("integerA is not 1,2, or 3");
      break;
}
```

2.4.2 Câu lệnh lặp

C# cung cấp bốn loại lệnh lặp (for, while, do...while, và foreach) cho phép lập trình viên có thể thực thi một khối lệnh liên tiếp cho đến khi một điều kiện xác định nào đó được thỏa mãn.

2.4.2.1 Câu lệnh lặp for

Cú pháp của câu lệnh lặp for có cú pháp như sau:

```
for (initializer; condition; iterator)
   statement(s)
```

trong đó:

- Initializer:biểu thức được ước lượng trước khi lần lặp đầu tiên được thực thi (đây

thường là nơi khởi tạo một biến cục bộ như là một "biến đếm").

- Condition: là biểu thức kiểm tra trước khi mỗi vòng lặp được thực thi.
- Iterator: biểu thức được ước lượng sau mỗi vòng lặp (thường dùng để tăng "biến đếm"). Các vòng lặp sẽ kết thúc khi condition được ước lượng là false.

Ví dụ dưới đây in ra 100 số tự nhiên đầu tiên (0, 1, 2, ..., 99), mỗi số trên một dòng:

```
for (int i = 0; i < 100; i = i+1)
{
   Console.WriteLine(i);
}
```

Tất nhiên, chúng ta có thể sử dụng các vòng lặp for lồng nhau, như ví dụ in ra tam giác hình sao dưới đây.

```
for (int i = 0; i < 5; i++)
{
   for (int j = 0; j <= i; j++)
     Console.Write("*");
   Console.WriteLine(i);
}
```

2.4.2.2 Câu lệnh lặp while

Câu lệnh lặp while, còn được gọi là câu lệnh lặp kiểm tra điều kiện trước, có cú pháp như sau:

```
while(condition)
   statements;
```

Ví dụ: Đoạn chương trình sau minh họa việc kiểm tra nhập vào một chuỗi từ dòng lệnh, sẽ dừng khi chuỗi nhập vào là "abc":

```
string correctPwd = "abc", st = "";
while (st != correctPwd)
{
   Console.Write("Password = ");
   st = Console.ReadLine();
}
```

2.4.2.3 Câu lệnh lặp do…while

Câu lệnh lặp do... while được coi là phiên bản kiểm tra điều kiện sau của câu lệnh while, có cú pháp như sau:

```
do {
```

```
   statements;
}
while(condition)
```

Ví dụ: Đoạn chương trình minh họa việc kiểm tra nhập vào một chuỗi từ dòng lệnh, sẽ dừng khi chuỗi nhập vào là "abc", được viết lại theo kiểu câu lệnh lặp do...while như sau:

```
string correctPwd = "abc", st; //để ý rằng st không cần khởi tạo là ""
do {
   Console.Write("Password = ");
   st = Console.ReadLine();
}
while (st != correctPwd)
```

2.4.2.4 Câu lệnh lặp foreach

Câu lệnh lặp foreach cho phép duyệt qua mỗi phần tử có trong một tập phần tử. Kiểu dữ liệu tập hợp phần tử (collection) sẽ được trình bày trong các phần tiếp theo. Xét ví dụ dưới đây:

```
foreach (int temp in arrayOfInts)
{
   Console.WriteLine(temp);
}
```

Ví dụ này sẽ in ra tất cả các phần tử có trong tập hợp arrayOfInts. Có một điều đáng lưu ý, chúng ta không được thay đổi giá trị của biến phần tử lặp. Chẳng hạn, đoạn chương trình dưới đây sẽ bị báo lỗi:

```
foreach (int temp in arrayOfInts)
{
   temp++; //không được thay đổi giá trị temp!!!
   Console.WriteLine(temp);
}
```

2.4.3 Câu lệnh nhảy

C# cung cấp một số câu lệnh nhảy cho phép chuyển điều khiển đến dòng lệnh khác trong chương trình.

2.4.3.1 Câu lệnh goto

Lệnh goto cho phép nhảy trực tiếp đến một dòng cụ thể trong chương trình, được xác định bằng một nhãn (**label**):

```
goto Label1;
Console.WriteLine("Dòng lệnh này sẽ không được thực hiện");
Label1:
```

```
Console.WriteLine("Continuing execution from here");
```

2.4.3.2 Câu lệnh break

Chúng ta đã sử dụng câu lệnh break trong phần câu lệnh rẽ nhánh switch. Có một cách sử dụng khác của câu lệnh này, đó là dùng để nhảy ra khỏi điều khiển của lệnh lặp trực tiếp chứa nó (for, foreach, while, do...while).

Ví dụ sau đây là một phiên bản khác của đoạn lệnh kiểm tra mật khẩu ở trên:

```
string correctPwd = “abc”;
while (true)
{
   Console.Write(“Password = “);
   string st = Console.ReadLine();
   if (st == correctPwd)
     break;
}
```

2.4.3.3 Câu lệnh continue

Lệnh continue cũng tương tự như câu lệnh break, phải được sử dụng trong thân câu lệnh for, foreach, while, hay do… while. Tuy nhiên, nó chỉ thoát từ lần lặp hiện tại của vòng lặp để bắt đầu lần lặp mới.

Ví dụ, đoạn chương trình dưới đây…

```
for (int i = 0; i < 4; i++)
{
   Console.WriteLine(“---“);
   if (i%2 == 0)
     continue;
   Console.WriteLine(“i = {0}”, i);
}
```

… sẽ in ra kết quả như thế này trong cửa sổ Console:

```
---
---
i = 1
---
---
i = 3
```

2.4.3.4 Câu lệnh return

Lệnh return được sử dụng để thoát khỏi phương thức của một lớp, trả điều khiển trở về nơi gọi phương thức. Tùy theo kiểu dữ liệu trả về của phương thức là void hoặc có một

kiểu dữ liệu cụ thể, lệnh return phải tương ứng không trả về kiểu dữ liệu gì, hoặc là trả về một giá trị có kiểu dữ liệu thích hợp.

2.5 Cấu trúc chương trình

Trong phần đầu của chương này, chúng ta đã viết một chương trình C# đơn giản đầu tiên. Tại thời điểm đó, chúng ta chỉ quan tâm đến cách thức quản lý, biên dịch solution, project của Visual Studio. Sau khi đã nắm vững được cấu trúc điều khiển cũng như một số đặc điểm cụ thể của ngôn ngữ, giờ là lúc chúng ta xem xét cấu trúc của một chương trình viết bằng C#.

2.5.1 Lớp đối tượng

Lớp đối tượng đóng vai trò rất lớn trong các chương trình C#. Nói một cách nôm na, lớp đối tượng là khuôn đúc ra các đối tượng cụ thể (gọi là instance), định nghĩa các thành phần dữ liệu và chức năng có thể có cho mỗi đối tượng cụ thể.

Thành viên của lớp đối tượng là các dữ liệu và các hàm bên trong lớp đối tượng nó, gọi là dữ liệu thành phần và hàm thành phần. Các thành viên của lớp đối tượng có thể được khai báo là public (có thể được truy xuất trực tiếp từ bên ngoài lớp đối tượng), hoặc private (chỉ được nhìn thấy ở trong chính khai báo lớp đối tượng), protected (chỉ được truy xuất từ bên trong chính lớp đối tượng hoặc các lớp đối tượng khác kế thừa từ nó).

Dữ liệu thành phần là các thành phần bên trong lớp chứa dữ liệu cho class – đó có thể là các trường dữ liệu (field), hằng số (constant) hoặc là các sự kiện (event).

Trường dữ liệu là các biến được khai báo ở mức lớp đối tượng. Ví dụ dưới đây định nghĩa một lớp đối tượng có tên là PhoneCustomer với 3 trường dữ liệu CustomerID, FirstName và LastName. Lớp này cũng định nghĩa một hằng ở mức lớp là DayOfSendingBill.

```
class PhoneCustomer{
   public const int DayOfSendingBill = 1;
   public int CustomerID;
   public string FirstName;
   public string LastName;
}
```

Khi tạo ra một đối tượng của lớp đối tượng PhoneCustomer, chúng ta có thể truy xuất các trường dữ liệu này theo dạng đốiTượng.TrườngDữLiệu, như ví dụ dưới đây:

```
PhoneCustomer Customer1 = new PhoneCustomer();
```

```
Customer1.FirstName = "Burton";
```

Hàm thành phần là các thành phần cung cấp chức năng xử lý dữ liệu cho lớp đối tượng. Chúng có thể là các phương thức (method), thuộc tính (property), hàm khởi dựng (constructor), hàm hủy bỏ (destructor), hoặc indexer.

Phương thức (method) là các hàm được khai báo trong lớp đối tượng. Chúng có thể là phương thức làm việc với thể hiện cụ thể của lớp, hoặc là phương thức chỉ hoạt động ở mức lớp (phương thức tĩnh – static method).

Thuộc tính (property) là tập các hàm có thể được truy xuất theo cách giống như trường dữ liệu public của lớp đối tượng. C# cung cấp các cú pháp đặc biệt để định nghĩa các thuộc tính chỉ đọc, chỉ ghi hay được truy xuất tự do.

Hàm khởi dựng (constructor) là các hàm đặc biệt, được gọi mỗi khi đối tượng của lớp được tạo mới. Các hàm khởi dụng phải có trùng tên với tên lớp đối tượng.

Hàm hủy bỏ (destructor) là các hàm được gọi khi đối tượng bị hủy. Các hàm này có tên của lớp và được bắt đầu bằng ký tự ~ (dấu ngã).

Danh sách đầy đủ các bổ từ truy cập các hàm thành phần của lớp được cho ở bảng dưới đây:

Bổ từ truy cập	Mô tả
new	Ẩn phương thức có cùng khai báo được kế thừa từ lớp cha
public	Phương thức có thể được truy cập từ mọi nơi
protected	Phương thức có thể được truy cấp bên trong lớp khai báo nó hoặc từ một kiểu dữ liệu khác được dẫn xuất từ lớp khai báo nó.
internal	Phương thức có thể được truy xuất trong phạm vi cùng assembly.
private	Phương thức chỉ có thể truy xuất trong lớp khai báo nó.
static	Phương thức hoạt động ở mức lớp, không hoạt động với một đối tượng cụ thể.
virtual	Phương thức có thể được ghi đè (override) trong lớp dẫn xuất.
abstract	Phương thức chỉ đóng vai trò định nghĩa cú pháp, không cài đặt.
override	Phương thức ghi đè phương thức được định nghĩa là virtual hoặc abstract ở lớp cha.
sealed	Phương thức ghi đè phương thức virtual, nhưng không thể được ghi đè bởi bất cứ lớp nào khi dẫn xuất thêm.
extern	Phương thức được cài đặt từ bên ngoài, có thể là bằng ngôn ngữ khác.

2.5.2 Kiểu dữ liệu cấu trúc – struct

Cú pháp khai báo kiểu dữ liệu cấu trúc trong C# hoàn toàn tương tự như khai báo lớp đối tượng, chỉ thay từ khóa class bằng từ khóa struct.

Chẳng hạn, đây là khai báo kiểu cấu trúc PhoneCustomer:

```
struct PhoneCustomer{
   public const int DayOfSendingBill = 1;
   public int CustomerID;
   public string FirstName;
   public string LastName;
}
```

Điểm khác biệt giữa kiểu cấu trúc và lớp đối tượng đó là cách mà chúng được lưu trữ và truy xuất. Lớp đối tượng là kiểu dữ liệu tham chiếu, được lưu trữ trong vùng nhớ heap, trong khi đó, cấu trúc là kiểu dữ liệu giá trị, được lưu trữ trong vùng nhớ stack. Ngoài ra, cấu trúc không thể được kế thừa như ở lớp đối tượng.

2.6 Phương thức

2.6.1 Khai báo phương thức

Cú pháp định nghĩa một phương thức trong C# tương tự như ở C++. Điểm khác biệt đó là, trong C#, mỗi phương thức được đều được khai báo tầm truy xuất của nó (public, private, protected) và định nghĩa luôn phần thân phương thức. Nghĩa là, không được sử dụng từ khóa "public:" để gộp nhóm nhiều định nghĩa phương thức public.

Dưới đây là cú pháp định nghĩa một phương thức:

```
[modifiers] return_type MethodName([parameters])
{
   // Method body
}
```

Ví dụ, đoạn code dưới đây định nghĩa hai phương thức IsSquare() và Move():

```
public bool IsSquare(Rectangle rect)
{
   return (rect.Height == rect.Width);
}

protected void Move(int dX, int dY)
{
  this.x += dX;
  this.y += dY;
```

```
}
```

2.6.2 Truyền tham số cho phương thức

Các đối số có thể được truyền cho phương thức theo tham chiếu hoặc giá trị. Biến được truyền theo tham chiếu đến một phương thức thì sẽ bị ảnh hưởng bởi mọi thay đổi nếu có trong thân phương thức, trong khi đó, biến được truyền theo giá trị thì không bị ảnh hưởng bởi những thay đổi diễn ra trong thân phương thức.

Dưới đây là minh họa việc sử dụng các đối số trong phương thức:

```
using System;

namespace MinhHoaDoiSo
{
   class ParameterTest
   {
      static void SomeFunction(int[] ints, int i)
      {
         ints[0] = 100;
         i = 100;
      }

      public static int Main()
      {
         int i = 0;
         int[] ints = { 0, 1, 2, 4, 8 };
         // Hien thi danh sach gia tri truoc khi goi phuong thuc
         Console.WriteLine("i = " + i);
         Console.WriteLine("ints[0] = " + ints[0]);
         Console.WriteLine("Thuc hien goi phuong thuc SomeFunction()...");

         // Sau khi goi phuong thuc, gia tri trong mang ints duoc thay doi,
         // nhung gia tri cua i thi khong!
         SomeFunction(ints, i);
         Console.WriteLine("i = " + i);
         Console.WriteLine("ints[0] = " + ints[0]);
         return 0;
      }
   }
}
```

Một minh họa kết quả ở console là như sau:

```
i = 0
ints[0] = 0
Thuc hien goi phuong thuc SomeFunction()...
i = 0
ints[0] = 100
```

Để thay đổi giá trị của đối số i, chúng ta phải truyền nó như là một đối số kiểu tham chiếu. Việc định nghĩa một đối số là kiểu tham chiếu được thực hiện bằng cách thêm từ khóa ref vào đầu định nghĩa đối số, như ví dụ sau:

```
static void SomeFunction(int[] ints, ref int i)
{
   ints[0] = 100;
   i = 100;
}
```

Và khi thực hiện lời gọi phương thức, từ khóa ref cũng phải được thêm vào trước biến truyền cho phương thức:

```
         SomeFunction(ints, ref i);
```

Một điểm cần lưu ý đó là, biến được sử dụng để truyền cho phương thức phải được khởi tạo trước khi thực hiện lời gọi phương thức.

Để truyền một đối số làm nhiệm vụ chứa giá trị đầu ra của một phương thức, chúng ta sử dụng từ khóa out. Biến được truyền theo kiểu như thế này thì không nhất thiết phải được khởi tạo trước khi thực hiện lời gọi phương thức, tuy nhiên, nếu trong thân hàm đối số không được gán một giá trị nào thì trình biên dịch sẽ báo lỗi.

Cách sử dụng đối số kiểu out là như ví dụ dưới đây:

```
static void SomeFunction(out int i)
{
   i = 100; //phai thuc hien thay doi gia tri cua doi so i trong than phuong thuc!!!
}

public static int Main()
{
   int i; // chi khai bao bien i, chua can KHOI TAO GIA TRI cho no
   SomeFunction(out i);
   Console.WriteLine(i);
   return 0;
}
```

2.7 Dữ liệu kiểu array

2.7.1 Cú pháp khai báo array

Array trong C# được khai báo bằng cách gắn cặp dấu ngoặc vuông vào sau kiểu dữ liệu cơ sở, theo cú pháp dưới.

```
type[] arrayName;
```

Ví dụ:

```
int[] daySo; //khai bao daySo la mot array (co the chua cac so int)
```

Để khởi tạo array, chúng ta sử dụng từ khóa new, sau đó chỉ định cụ thể kích thước trong cặp dấu ngoặc vuông. Sau khi khởi tạo, mỗi phần tử trong array được truy xuất thông qua tên array cùng với số hiệu của nó (được đánh số từ 0 trở đi)

```
// Khoi tao mot array voi 32 phan tu du lieu kieu int
int kichThuoc = 10 + 20;
int[] daySo = new int[kichThuoc]; // luu y co the dung BIEN de xac dinh kich thuoc!!!
daySo[0] = 35;   // gan gia tri cho phan tu dau tien trong daySo
daySo[31] = 432; // gan gia tri cho phan tu thu 32 trong daySo
```

Lưu ý rằng, chúng ta có thể sử dụng giá trị của biến khi định nghĩa kích thước cho array. Và, một khi đã khởi tạo xong array với kích thước cụ thể, chúng ta không thể thay đổi kích thước của array đó. Để làm được điều này, chúng ta cần một kiểu dữ liệu khác – ArrayList.

Chúng ta cũng có thể khai báo và định nghĩa một array theo cách chỉ ra các phần tử cụ thể của nó như sau:

```
string[] myArray = {"first element", "second element", "third element"};
```

2.7.2 Làm việc với array

Để lấy kích thước của array, chúng ta sử dụng thuộc tính Length của nó. Một số phương thức thường dùng đối với dữ liệu kiểu array là:

- Array.Sort(arr): hàm tĩnh, sử dụng để sắp xếp array arr; arr là array của các phần tử có kiểu được định nghĩa sẵn trong C#.
- Array.Reverse(arr): hàm tĩnh, sử dụng để đảo ngược vị trí của các phần tử có trong array arr.

Ví dụ sau minh họa việc khai báo một array gồm các string, sắp xếp theo thứ tự ABC, đảo ngược các phần tử, rồi in các phần tử ra Console:

```
string[] hoTen = {"Nguyen Van Trung", "Nguyen Hoang Ha", "Tran Nguyen Phong"};

// in danh sach cac phan tu trong array
for (int i = 0; i < hoTen.Length; i++)
```

```
  Console.WriteLine(hoTen[i]);

Array.Sort(hoTen);
Array.Reverse(hoTen);

// in danh sach cac phan tu trong array su dung cu phap foreach
foreach (string stHoTen in hoTen)
   Console.WriteLine(stHoTen);
```

2.7.3 Array nhiều chiều

C# hỗ trợ hai kiểu array nhiều chiều: kiểu ma trận (chẳng hạn như một ma trận hai chiều là một array trong đó mỗi dòng sẽ có cùng số cột) và array kiểu răng cưa.

Ví dụ dưới đây minh họa cách khai báo và khởi tạo một array kiểu ma trận hai chiều:

```
string[,] beatleName = { {"Lennon","John"},
                         {"McCartney","Paul"},
                         {"Harrison","George"},
                         {"Starkey","Richard"} };
```

Lưu ý rằng chúng ta sử dụng dấu phẩy để phân cách các chiều trong khai báo array, ngay cả khi chúng ta chưa thực sự xác định kích thước của mỗi chiều. Để khai báo một array 3 chiều gồm các string, chúng ta làm như sau:

```
string[,,] my3DArray;
```

Array, sau khi khai báo có thể tiến hành khởi tạo giá trị cho các phần tử bên trong, như sau:

```
double [, ] matrix = new double[5, 10];
for (int i = 0; i < 5; i++)
{
   for (int j=0; j < 10; j++)
      matrix[i, j] = i*j;
}
```

Sau khi khởi tạo array, chúng ta có thể xác định kích thước của từng chiều bằng phương thức GetLength(). Chẳng hạn,

```
double [, ] matrix = new double[5, 10];
Console.WriteLine("Kich thuoc chieu thu nhat: {0}", matrix.GetLength(0)); // 5
Console.WriteLine("Kich thuoc chieu thu nhat: {0}", matrix.GetLength(1)); // 10
```

Kiểu array nhiều chiều thứ hai là array kiểu răng cưa, trong đó kích thước của mỗi chiều là có thể khác nhau. Chính xác, đây là kiểu dữ liệu array của các array. Đoạn code dưới

đây minh họa cách định nghĩa một array kiểu răng cưa:

```
int[][] a = new int[3][];
a[0] = new int[4];
a[1] = new int[3];
a[2] = new int[1];
```

Từng phần tử của array kiểu răng cưa sẽ được truy xuất như là một array một chiều như thông thường. Chương trình dưới đây cho thấy rõ hơn cách thức sử dụng của array kiểu răng cưa. Chương trình này có nhiệm vụ nhận vào một số nguyên n, sau đó thành lập tam giác Pascal rồi in tam giác này ra màn hình.

```
using System;

namespace MinhHoaArray
{
   class TamGiacPascal
   {
   Console.Write("n = ");
   int n = int.Parse(Console.ReadLine());
   long[][] C = new long[n+1][]; // khoi tao C la 1 array gom n+1 phan tu

   for (int i = 0; i <= n; i++)
   {
          C[i] = new long[i+1]; // phan tu thu i cua C la 1 array gom (i+1) phantu

          C[i][0] = C[i][i] = 1;
          for (int j = 1; j < i; j++)
        C[i][j] = C[i-1][j] + C[i-1][j-1];
   }

   for (int i = 0; i < C.Length; i++)
   {
     for (int j = 0; j < C[i].Length; j++)
       Console.Write("{0, 5}", C[i][j]); // in so voi do rong la 5
     Console.WriteLine();
   }

   Console.ReadLine();
    }
}
```

Một minh họa kết quả in ra của chương trình là:

```
n = 5
     1
     1     1
     1     2     1
     1     3     3     1
```

```
    1     4      6      4      1
    1     5     10     10      5      1
```

2.8 Các toán tử

C# hỗ trợ các kiểu toán tử sau đây:

Loại toán tử	Ký hiệu
Số học	`+ - * / %`
Logic	`& | ^ ~ && || !`
Cộng chuỗi	`+`
Tăng và giảm	`++ --`
Dịch bit	`<< >>`
So sánh	`== != < > <= >=`
Phép gán	`= += -= *= /= %= &= |= ^= <<= >>=`
Truy xuất thành phần (cho object và struct)	`.`
Indexing (cho array và các indexers)	`[]`
Ép kiểu	`()`
Điều kiện	`?:`
Tạo đối tượng	`new`
Thông tin về kiểu	`sizeof is typeof as`
Điều khiển Overflow exception	`checked unchecked`
Truy xuất địa chỉ và gián tiếp	`* -> & []`

Cũng giống như các ngôn ngữ tựa C, chúng ta cần *phân biệt phép gán với phép so sánh*. Chẳng hạn

```
x = 3; // gán giá trị 3 cho x
```

trong khi đó, muốn so sánh x với 3, chúng ta phải ghi như thế này

```
if (x == 3)
```

2.8.1 Các toán tử tắt

Dưới đây là các toán tử viết tắt của C#

Toán tử tắt	Tương đương với
x++, ++x	x = x + 1
x--, --x	x = x - 1
x += y	x = x + y

Toán tử tắt	Tương đương với
x -= y	x = x – y
x *= y	x = x * y
x /= y	x = x / y
x %= y	x = x % y
x >>= y	x = x >> y
x <<= y	x = x << y
x &= y	x = x & y
x |= y	x = x | y
x ^= y	x = x ^ y

2.8.2 Toán tử tam nguyên (ternary operator)

Cũng như các ngôn ngữ tựa C, C# cũng cung cấp toán tử tam nguyên:

```
condition ? true_value : false_value
```

2.8.3 Chỉ dẫn checked và unchecked

Xem xét đoạn mã lệnh sau

```
byte b = 255;
b++;
Console.WriteLine(b.ToString());
```

Dữ liệu kiểu byte chỉ có thể nắm giữ được các giá trị trong vùng từ 0 đến 255, vì thế, việc tăng giá trị của b sẽ làm xuất hiện lỗi overflow. Việc quản lý các lỗi kiểu như thế này rất quan trọng, vì thế C# cung cấp một cú pháp cho phép phát hiện ra kiểu lỗi này. Xem đoạn mã lệnh

```
byte b = 255;
checked
{
   b++;
}
Console.WriteLine(b.ToString());
```

Khi thực thi đoạn mã lệnh này, chúng ta sẽ nhận được thông báo lỗi dạng

```
Unhandled Exception: System.OverflowException: Arithmetic operation resulted in an
overflow.
   at MinhHoa.Main(String[] args)
```

Nếu muốn bỏ qua lỗi overflow, chúng ta đánh dấu đoạn mã lệnh tương ứng là unchecked

```
byte b = 255;
unchecked
{
   b++;
}
Console.WriteLine(b.ToString());
```

Trong trường hợp này, không có ngoại lệ nào được phát sinh, nhưng chúng ta sẽ bị mất dữ liệu! Lưu ý rằng, unchecked là trạng thái mặc định!

2.8.4 Toán tử is

Toán tử is cho phép kiểm tra một object có "tương thích" với một kiểu dữ liệu nào đó hay không. Trong phần lập trình hướng đối tượng, chúng ta sẽ có dịp sử dụng toán tử này.

2.8.5 Toán tử sizeof

Toán tử sizeof cho phép xác định kích thước (tính bằng byte) được yêu cầu bởi một kiểu dữ liệu giá trị trên vùng nhớ stack . Xem ví dụ minh họa sau:

```
string s = "A string";
unsafe
{
   Console.WriteLine(sizeof(int)); // in ra kich thuoc cua 1 int, la 4
}
```

Lưu ý rằng, toán tử sizeof chỉ được sử dụng với mã lệnh không an toàn (unsafe code).

2.9 Enumerations – Kiểu liệt kê

Một kiểu liệt kê là một kiểu dữ liệu nguyên do người sử dụng tự định nghĩa. Khi khai báo một kiểu liệt kê, chúng ta xác định một tập các giá trị có thể nhận được thông qua các tên gọi có tính gợi nhớ.

Chúng ta có thể định nghĩa một kiểu liệt kê như ví dụ sau:

```
public enum TimeOfDay
{
   Morning = 0,
   Afternoon = 1,
   Evening = 2
}
```

Việc sử dụng kiểu dữ liệu liệt kê sẽ cho phép viết các đoạn mã lệnh dễ đọc, dễ quản lý lỗi logic hơn

```
class EnumExample
{
```

```
   public static int Main()
   {
      WriteGreeting(TimeOfDay.Morning);
      return 0;
   }

   static void WriteGreeting(TimeOfDay timeOfDay)
   {
      switch(timeOfDay)
      {
         case TimeOfDay.Morning:
            Console.WriteLine("Good morning!");
            break;
         case TimeOfDay.Afternoon:
            Console.WriteLine("Good afternoon!");
            break;
         case TimeOfDay.Evening:
            Console.WriteLine("Good evening!");
            break;
         default:
            Console.WriteLine("Hello!");
            break;
      }
   }
}
```

Một đặc tính đặc biệt của kiểu liệt kê cần lưu ý đó là khả năng chuyển đổi qua lại giữa nó với kiểu dữ liệu string.

```
//enum → string
TimeOfDay time = TimeOfDay.Afternoon;
Console.WriteLine(time.ToString()); // chuoi in ra la “Afternoon”

//string → enum
TimeOfDay time2 = (TimeOfDay) Enum.Parse(typeof(TimeOfDay), "afternoon", true);

Console.WriteLine((int)time2);
```

Việc lấy chuỗi tương ứng của một biến giá trị liệt kê có thể tiến hành đơn giản bằng cách sử dụng phương thức ToString() của biến. Trong khi đó, để lấy giá trị kiểu liệt kê từ một chuỗi tương ứng thì phức tạp hơn, chúng ta phải sử dụng phương thức tĩnh Parse của lớp Enum. Còn có nhiều phương thức khác của lớp Enum; chi tiết tham khảo bạn xem ở tài liệu MSDN.

2.10 Namespace

Namespace là đơn vị gộp nhóm mang tính logic. Khi định nghĩa một lớp đối tượng trong một file C#, chúng ta có thể đưa nó vào trong một namespace nào đó. Sau đó, khi định

nghĩa lớp khác có chức năng liên quan với lớp đối tượng trước đó, chúng ta có thể gộp nó vào trong cùng namespace, qua đó tạo ra một nhóm logic, cho các nhà phát triển biết rằng các lớp đối tượng trong cùng namespace là có liên quan với nhau.

```
namespace BusinessLayer
{
   public struct Student
   {
      // Ma lenh dinh nghia struct Student...
   }
}
```

Kiểu dữ liệu thuộc về một namespace sẽ có tên đầy đủ được xác định bằng

```
namespace.tenKieuDuLieu
```

Như ở ví dụ trên, tên đầy đủ của lớp Student sẽ là

```
BusinessLayer.Student
```

Student còn được gọi là tên ngắn gọn của kiểu dữ liệu.

Chúng ta cũng có thể lồng các namespace bên trong các namespace khác, qua đó tạo ra cấu trúc phân lớp cho các kiểu dữ liệu.

```
namespace HueUni
{
   namespace COSciences
   {
      namespace DoIT
      {
         class Student
         {
            // Ma lenh dinh nghia lop doi tuong Student...
         }
      }
   }
}
```

Tên namespace bao gồm các tên namespace chứa nó, phân cách nhau bằng dấu chấm, bắt đầu bằng namespace ngoài cùng nhất, và kết thúc bằng chính tên ngắn gọn của nó. Ví dụ, tên đầy đủ của namespace DoIT là HueUni.COSciences.DoIT

Chúng ta cũng có thể dùng cú pháp như sau để tổ chức các namespace:

```
namespace HueUni.COSciences.DoIT
```

```
{
   class Student
   {
      // Ma lenh dinh nghia lop doi tuong Student...
   }
}
```

Namespace không tương ứng với assembly. Hoàn toàn có thể có các namespace khác nhau trong cùng một assembly, hay ngược lại, có thể định nghĩa các kiểu dữ liệu cho cùng một namespace ở nhiều assembly khác nhau.

2.10.1 Khai báo sử dụng namespace

Việc sử dụng kiểu dữ liệu với tên đầy đủ rõ ràng là không thuận tiện lắm, đặc biệt là khi kiểu dữ liệu được định nghĩa trong một namespace ở mức quá sâu. C# cho phép sử dụng tên ngắn để xác định kiểu dữ liệu bằng cách xác định trước namespace của kiểu dữ liệu này với từ khóa using ở đầu file mã nguồn. Chẳng hạn, nếu có khai báo

```
using HueUni.COSciences.DoIT
```

ở đầu file mã nguồn thì trong file đó, chúng ta có thể sử dụng tên ngắn của lớp đối tượng `Student` thay cho tên đầy đủ của nó là `HueUni.COSciences.DoIT.Student`

Trong trường hợp tên ngắn của hai kiểu dữ liệu thuộc về hai namespace cùng tham khảo là trùng nhau, khi sử dụng kiểu dữ liệu, chúng ta phải chỉ rõ với namespace cụ thể. Chẳng hạn, giả sử lớp đối tượng có tên Student được định nghĩa trong cả hai namespace HueUni.COSciences.DoiT và HueUni.COSciences.DoP; khi đó chúng ta cần xác định lớp Student bằng một cái tên dài hơn, DoIT.Student hoặc DoP.Student tùy theo từng tình huống.

```
using HueUni.COSciences.DoIT;
using HueUni.COSciences.DoP;

class Test
{
   public static int Main()
   {
      DoIT.Student nvtrung = new DoIT.Student();
      DoP.Student nhha = new DoP.Student();
      return 0;
   }
}
```

2.10.2 Bí danh cho Namespace

Một cách sử dụng khác của từ khóa using đó là gán bí danh cho các lớp đối tượng và các

namespace. Cú pháp của cách sử dụng này là như sau:

```
using alias = NamespaceName;
```

Ví dụ sau đây minh họa cách thức sử dụng bí danh namespace:

```
using System;
using KhoaCNTT = HueUni.COSciences.DoIT;
class Test
{
   public static int Main()
   {
      KhoaCNTT.Student nvtrung = new KhoaCNTT.Student();
      Console.WriteLine(nvtrung.GetNamespace());
      return 0;
   }
}

namespace HueUni.COSciences.DoIT
{
   class Student
   {
      public string GetNamespace()
      {
         return this.GetType().Namespace;
      }
   }
}
```

2.11 Bài thực hành

Bài thực hành 1.1. Chương trình đầu tiên

Tóm tắt

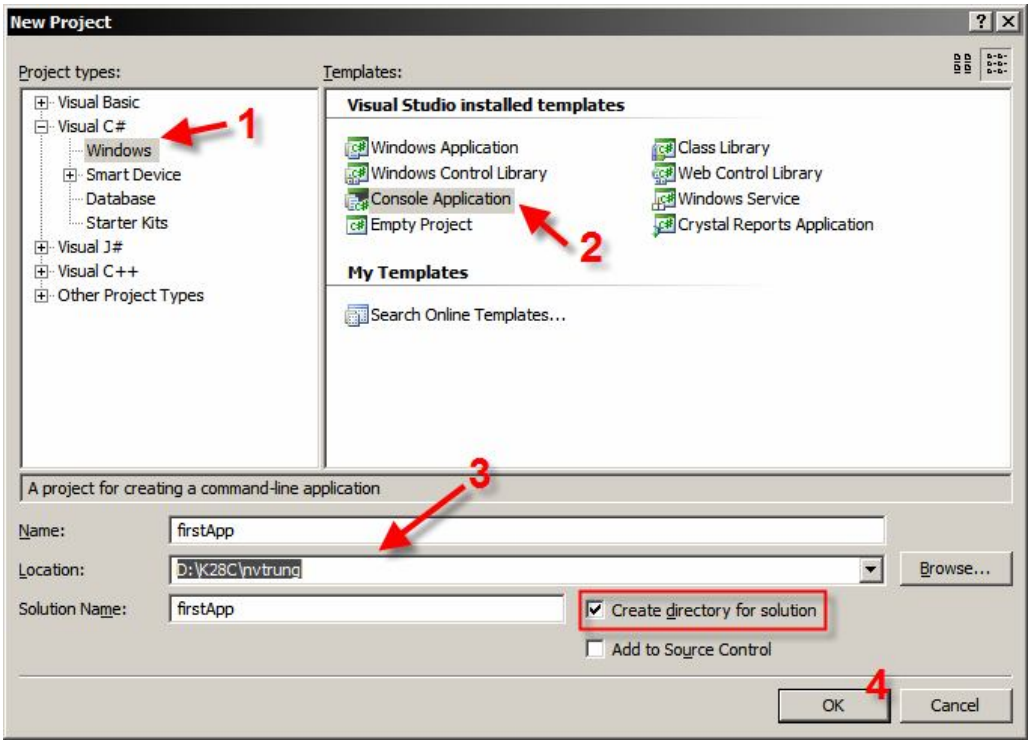
Bài thực hành này giúp bạn làm quen với môi trường Visual Studio 2005 và các thao tác nhập xuất cơ bản thông qua giao diện bàn phím. Cụ thể, chương trình yêu cầu người sử dụng nhập hai số, sau đó in ra màn hình tổng, tích và thương của hai số này.

Kỹ thuật được trình bày

- Làm quen với môi trường Visual Studio 2005. Cấu trúc một solution, project và các tài nguyên có liên quan
- Cách thức sử dụng thư viện MSDN để tra cứu, hướng dẫn
- Sử dụng thao tác nhập xuất cơ bản

Trình tự thực hiện

1. Khởi động Microsoft Visual Studio 2005. Nhấn Ctrl + Shift + N hoặc chọn menu tương ứng là File → New → Project để tạo mới một project



2. Chọn loại ứng dụng cần phát triển là Visual C# → Console Application. Chọn thư mục chứa project và đặt tên cho project. Về mặt thực chất, Visual Studio coi project thuộc về một solution nào đó, và một solution có thể chứa nhiều project. Tuy nhiên, trong nhiều "bài toán" đơn giản (như ví dụ của chúng ta chẳng hạn), một solution chỉ có 1 project.
3. Đặt tên cho project của chúng ta thành firstApp. Sau khi nhấn nút OK, hãy khảo sát xem cấu trúc của thư mục chứa solution của chúng ta. Bạn phải luôn nắm chắc về ý nghĩa của các tập tin, thư mục được tạo ra trong quá trình làm việc.
4. Gõ mã lệnh như minh họa vào trong phần mã nguồn của tập tin Program.cs

```
using System;
using System.Collections.Generic;
using System.Text;

namespace firstApp
{
    class Program
    {
        static void Main(string[] args)
        {
            int x, y;
            Console.Write("So thu nhat: ");      //in cau thong bao "So thu nhat: " ra man hinh
            string st = Console.ReadLine();      //lay noi dung nguoi su dung nhap vao gan cho st
            x = int.Parse(st);                   //chuyen doi st thanh dang so va gan cho x

            //ngan gon hon, ban co the lam nhu the nay
            Console.Write("So thu hai: ");
            y = int.Parse(Console.ReadLine());

            //in ket qua
            float tong = x + y;
            Console.WriteLine("Tong {0} va {1} la {2}", x, y, tong);

            Console.WriteLine("Tich {0} va {1} la {2}", x, y, x*y);
            float thuong = (float)x / y;
            Console.WriteLine("Thuong {0} va {1} la {2:0.000}", x, y, thuong);
        }
    }
}
```

Biểu diễn với 3 chữ số thập phân

5. Sử dụng MSDN để tra cứu các thông tin bạn chưa biết về:
    a. Console và các phương thức ReadLine(), WriteLine() của nó
    b. Cách chuyển đổi kiểu chuỗi thành số, ví dụ như int.Parse()
6. Nhấn Ctrl + F5 để thực hiện chạy chương trình. Sau đó quan sát cấu trúc thư mục của solution, cho biết sự thay đổi của nó so với khi mới được tạo ra ở bước 3.
7. Thử thay đổi kết câu lệnh thành

    float thuong = (float)x / y;

    float thuong = x / y;

    rồi chạy chương trình, quan sát kết quả và rút ra kết luận.

8. Sử dụng thêm các cấu trúc lệnh khác để tinh chỉnh hoạt động của chương trình (xử lý phép chia cho 0, …)

Bài thực hành 1.2. Module hóa chương trình

Tóm tắt

Viết chương trình nhập vào một số nguyên N từ bàn phím. Sau đó

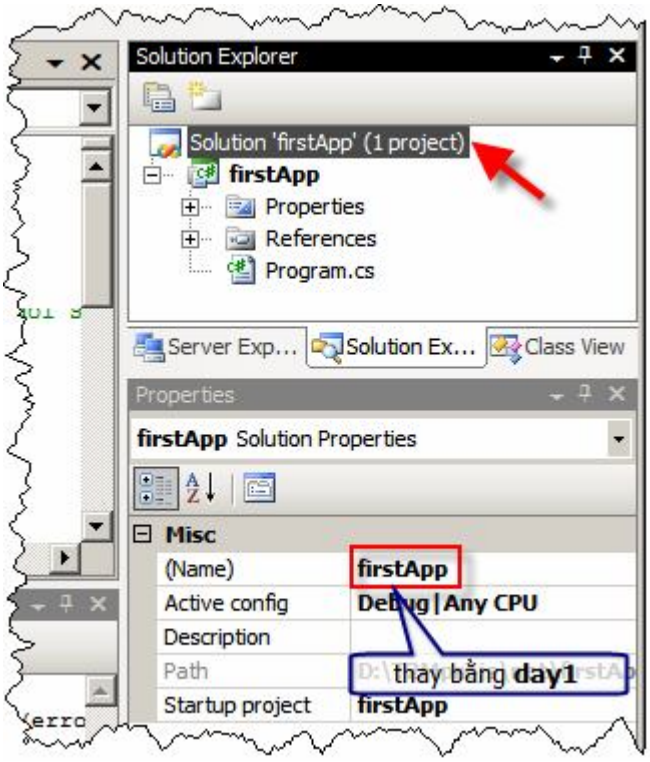
a. In ra màn hình giá trị N!.
b. Nhập thêm một số nguyên K từ bàn phím. Sau đó in ra C^K_N = N!/(K!*(N-K)!)

Kỹ thuật được trình bày

- Cấu trúc, cách quản lý logic và vật lý, cách làm việc của solution và project
- Thực hiện chia nhỏ ứng dụng thành để chuyên môn hóa các phần
- Cơ bản về các kiểu phương thức trong một lớp

Trình tự thực hiện

1. Mở solution đã làm ở Bài thực hành 1.1. Chỉnh sửa tên của solution từ "firstApp" thành "day1" cho có ý nghĩa. Xem cấu trúc thư mục của solution sau khi thay đổi.



2. Thêm một project vào solution này bằng menu lệnh File → Add → New project… . Tương tự như cách tạo mới project ở bài thực hành trước, chọn thể loại project là Console Application. Đặt tên cho project mới là "modular".

3. Quan sát cấu trúc cây thư mục của solution trong cửa sổ Solution Explorer và cả trong Windows Explorer. Để ý rằng, trong cửa sổ Solution Explorer, project firstApp được tô đậm. Điều này có nghĩa, firstApp đóng vai trò là "Startup project". Khi nhấn Ctrl + F5 thì project này sẽ được gọi thực thi chứ không phải là project modular mà ta mới tạo ra.

   Trong cửa sổ Solution Explorer, nhắp phải chuột lên "modular". Trong menu hiện ra, chọn menu lệnh "Set as Startup project" để thiết lập lại startup project cho solution.

4. Việc nhập n, tính n! rồi in kết quả bạn hoàn toàn có thể thực hiện được bằng các câu lệnh đơn giản. Tuy nhiên, để tăng tính rõ ràng và tái sử dụng, bạn nên tạo ra một phương thức để hỗ trợ việc tính toán n!. Xem mã lệnh bên dưới

```
using System;
using System.Collections.Generic;
using System.Text;

namespace modular
{
    class Program
    {
        static long giaiThua(int n)
        {
            long kq = 1;
            for (int i = 1; i <= n; i++)
                kq *= i;
            return kq;
        }

        static void Main(string[] args)
        {
            Console.Write("n = ");
            int n = int.Parse(Console.ReadLine());

            Console.WriteLine("n! = {0}! = {1}", n, giaiThua(n));
        }
    }
}
```

5. Chạy thử chương trình để xem kết quả. Hãy để ý rằng, khai báo phương thức giaiThua là static long giaiThua(int n). Thử xóa static trong khai báo này rồi chạy lại chương trình.

   ⇨ Lỗi nhận được cho biết *chỉ các phương thức static mới được triệu gọi, sử dụng lẫn nhau*

6. Bằng cách tạo ra phương thức long giaiThua() như trên, chúng ta có thể giải quyết

được vấn đề tính C^k_n một cách dễ dàng. Lời gọi để tính C^k_n như sau: GiaiThua(n)/(GiaiThua(n-k)*GiaiThua(k))

7. Hãy tạo ra một phương thức để tính tổ hợp chập k của n phần tử (bạn tự quyết định các tham số và kiểu dữ liệu trả về).

Bài thực hành 1.3. Tạo thư viện sử dụng chung

Tóm tắt

Trong thực tế, một *ứng dụng* có thể là có khả năng thực thi (executable) hoặc chỉ đơn thuần là thư viện để chứa các chức năng, lớp đối tượng.

Bài thực hành này hướng dẫn bạn tạo thư viện chứa các phương thức thường dùng. Với mục đích minh họa, thư viện này chỉ chứa 2 hàm tiện ích giúp tính giai thừa và tổ hợp chập. Sau khi biên dịch, bạn sẽ có được một file nhị với phần mở rộng là DLL. Thư viện này, khi cần, sẽ được tham chiếu đến trong các ứng dụng khác.

Kỹ thuật được trình bày

- Tạo loại ứng dụng loại thư viện

Trình tự thực hiện

1. Tạo mới một project, đặt tên là commonUtils (common utilities - các tiện ích dùng chung). Chú ý chọn loại ứng dụng cần tạo là Class Library
2. Mặc định Visual Studio 2005 sẽ tạo ra trong namespace CommonUtils một lớp tên là Class1. Đổi tên lớp này lại thành Math. Sau đó cài đặt các phương thức như sau:

```
using System;
using System.Collections.Generic;
using System.Text;

namespace commonUtils
{
    public class Math
    {
        public static long giaiThua(int n)
        {
            long kq = 1;
            for (int i = 1; i < n; i++)
                kq *= i;
            return kq;
        }

        public static long C(int n, int k)
        {
            return giaiThua(n) / (giaiThua(k) * giaiThua(n - k));
        }
    }
}
```

3. Rõ ràng, đây không phải là một chương trình để chạy như các ứng dụng bạn đã viết

trước đó - class Math không có phương thức static public Main() – tức là bạn không thể nhấn Ctrl + F5 để chạy chương trình. Biên dịch project này bằng menu lệnh *Build → Build commonUtils*. Kết quả, bạn sẽ có một thư viện commonUtils.dll trong thư mục bin\Release hoặc bin\Debug của project tùy theo cách chọn chế độ biên dịch. Thư viện này sẽ được dùng để tham chiếu đến trong các ứng dụng cần nó.

Mở rộng

Bổ sung các phương thức thường dùng khác vào thư viện, chẳng hạn như phương thức xác định xem một số có phải là nguyên tố hay không, phương thức hoán đổi giá trị của hai số cho trước, …

Bài thực hành 1.4. Tam giác Pascal

Tóm tắt

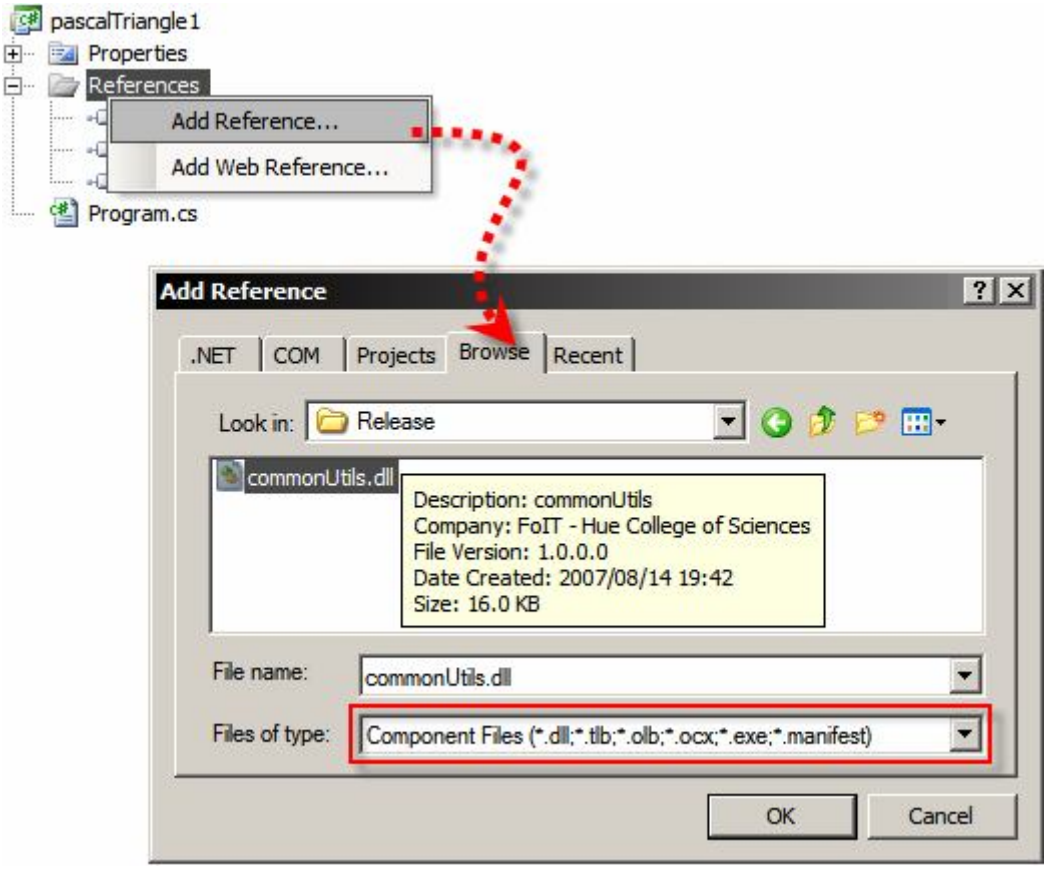
Viết chương trình nhập một số nguyên N từ bàn phím, sau đó in ra màn hình N dòng đầu tiên của tam giác Pascal.

Kỹ thuật được trình bày

- Sử dụng thư viện có sẵn

Trình tự thực hiện

1. Tạo mới một ứng dụng kiểu Console Application. Đặt tên project là pascalTriangle1
2. Thực hiện bổ sung tham khảo đến thư viện commonUtils bằng cách:
   - Nhắp phải chuột vào project pascalTriangle1 trong cửa sổ Solution Explorer
   - Trong menu hiện ra, chọn Add Reference…



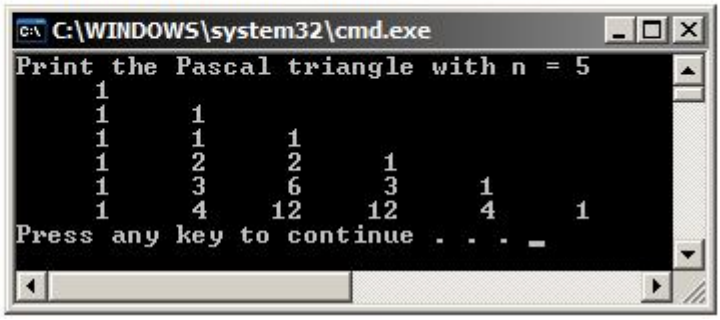
Trong tab Browse của hộp thoại Add Reference, tìm đến thư viện

commonUtils.dll đã tạo ra trước đó.

➔ Dễ thấy rằng thư viện được tham khảo đến không chỉ có dạng DLL mà có thể có các dạng khác, bao gồm EXE, OCX, …

3. Hoàn thiện phần mã nguồn có sử dụng tham chiếu đến thư viện vừa bổ sung như hình dưới:

```
using System;
using System.Collections.Generic;
using System.Text;

namespace pascalTriangle1
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.Write("Print the Pascal triangle with n = ");
            int n = int.Parse(Console.ReadLine());

            for (int i = 0; i <= n; i++)
            {
                for (int k = 0; k <= i; k++)
                    Console.Write("{0, 6}", commonUtils.Math.C(i, k));
                Console.WriteLine();
            }
        }
    }
}
```



Mở rộng

Hãy tự rút ra những ghi chú cần thiết về việc:

- Khai báo phương thức C(int n, int k) trong commonUtils là

  public static long C(int n, int k)

  static, public ở đây có ý nghĩa gì, có thể thay thế hoặc bỏ đi?

- Tương tự cho phương thức giaiThua(int n);

  Tại sao trong quá trình sử dụng phương thức C() lại phải ghi đầy đủ là

commonUtils.Math.C()? Chỉ cần ghi Math.C() có được không?

Bài thực hành 1.5. Tam giác Pascal – array version

Tóm tắt

Sử dụng array để xây dựng tam giác Pascal như Bài thực hành 1.4.

Kỹ thuật được trình bày

- Sử dụng array

Trình tự thực hiện

1. Tạo mới một project kiểu Console Application với tên là pascalTriangle2
2. Sử dụng các tính chất $C_0^0 = C_k^k = 1$, $C_n^k = C_{n-1}^{k-1} + C_{n-1}^k$, ta sẽ xây dựng một jagged array từ nhỏ đến lớn. Chi tiết như phần mã nguồn phía dưới:

```
using System;
using System.Collections.Generic;
using System.Text;

namespace pascalTriangle2
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.Write("Print the Pascal triangle with n = ");
            int n = int.Parse(Console.ReadLine());

            //calculate
            int[][] c = new int[n + 1][];
            for (int i = 0; i <= n; i++)
            {
                c[i] = new int[i + 1];
                c[i][0] = 1;
                c[i][i] = 1;
                for (int k = 1; k < i; k++)
                    c[i][k] = c[i - 1][k - 1] + c[i - 1][k];
            }

            //print out
            for (int i = 0; i <= n; i++)
            {
                for (int k = 0; k <= i; k++)
                    Console.Write("{0, 5}", c[i][k]);
                Console.WriteLine();
            }
        }
    }
}
```

Mở rộng

Có thể dùng array nhiều chiều trong trường hợp này không? Nếu có thì có sự khác nhau nào so với dùng jagged array?

CHƯƠNG 3 LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG VỚI C#

Trong phần trước chúng ta đã thấy cách sử dụng của lớp đôi tượng. Ở đây, chúng ta sẽ nhắc lại một số khái niệm và tính chất của lớp đối tượng trước khi nói về tính hướng đối tượng.

Lớp đối tượng được định nghĩa theo cú pháp:

```
class MyClass
{
   private int someField;

   public string SomeMethod(bool parameter)
   {
   }
}
```

Lớp đối tượng chứa các thành viên – thành viên là thuật ngữ được sử dụng để nói đến dữ liệu hoặc hàm được định nghĩa trong lớp. Thuật ngữ hàm (funciton) được dùng để nói đến bất kỳ thành viên nào có chứa mã lệnh, bao gồm phương thức (method), thuộc tính (property), hàm khởi dựng (constructor), hàm nạp chồng toán tử (operator overload).

Lớp đối tượng trong C# là kiểu dữ liệu tham chiếu. Điều này có nghĩa là khi bạn khai báo một biến có kiểu dữ liệu lớp thì xem như bạn có một biến có thể chứa tham chiếu đến một thể hiện của lớp đối tượng đó. Bạn cũng cần phải khởi tạo ra đối tượng bằng cách dùng toán tử new.

```
MyClass myObject;
myObject = new MyClass();
```

Cả hai thao tác khai báo và khởi tạo đối tượng có thể được làm một lần như thế này:

```
MyClass myObject = new MyClass();
```

Do là dữ liệu kiểu tham chiếu, nên phép gán hai biến tham chiếu có ý nghĩa là cho hai biến dữ liệu tham chiếu đến cùng một đối tượng.

```
MyClass myObjectRef = myObject;
```

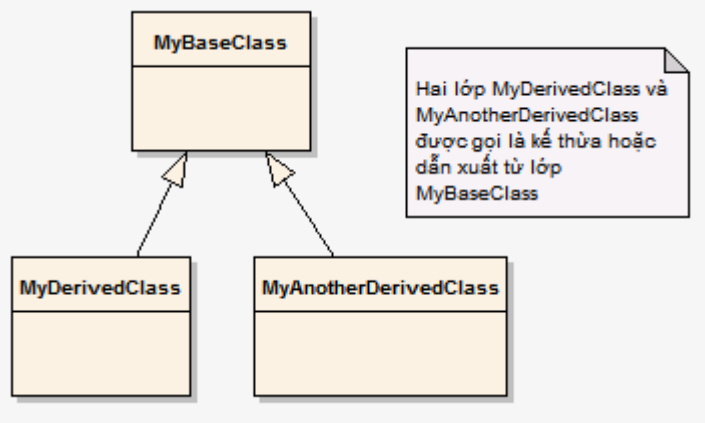
Tức là myObjectRef sẽ cùng tham chiếu đến cùng đối tượng mà myObject đang tham chiếu đến. Các phương thức của đối tượng được tham chiếu đến có thể được triệu gọi từ

cả myObjectRef và myObject.

3.1 Đơn kế thừa trong C#

C# hỗ trợ đơn kế thừa cho tất cả các lớp đối tượng, tức là một lớp *chỉ có thể dẫn xuất trực tiếp nhiều nhất là từ một lớp đối tượng khác*. Lớp cơ sở nhất trong C# là lớp System.Object

```
class MyDerivedClass : MyBaseClass
{
   // functions and data members here
}
```



Cũng như một số ngôn ngữ lập trình hướng đối tượng, C# có một số bổ từ truy cập để quy định phạm vi mã lệnh được phép truy xuất một thành viên trong lớp đối tượng.

Mức truy cập	Mô tả
public	Biến hoặc phương thức có thể được truy xuất từ bất cứ nơi nào
internal	Biến hoặc phương thức chỉ có thể truy xuất trong phạm vi cùng assembly
protected	Biến hoặc phương thức chỉ có thể truy xuất từ bên trong kiểu dữ liệu mà nó thuộc về, hoặc các kiểu dữ liệu dẫn xuất
protected internal	Biến hoặc phương thức có thể được truy xuất trong phạm vi assembly hiện tại, hoặc từ các kiểu dữ liệu dẫn xuất từ kiểu dữ liệu chứa nó
private	Biến hoặc phương thức chỉ có thể được truy xuất từ bên trong kiểu dữ liệu mà nó thuộc về

3.2 Nạp chồng phương thức (Method Overloading)

C# hỗ trợ nạp chồng phương thức, cho phép có nhiều phiên bản cho một phương thức có các *chữ ký* khác nhau. Khái niệm *chữ ký của phương thức* ở đây được hiểu là tên phương

thức, số lượng đối số, kiểu đối số được sử dụng trong phương thức.

Chẳng hạn, lớp đối tượng Student dưới đây có hai phương thức nạp chồng Display():

```
class Student
{
   // ……

   void Display(string stMessage)
   {
      // implementation
   }

   void Display()
   {
      // implementation
   }
}
```

3.3 Ghi đè phương thức và che dấu phương thức

Bằng cách khai báo một hàm ở lớp cơ sở là virtual, chúng ta có thể ghi đè hàm đó ở lớp dẫn xuất của lớp này.

```
class MyBaseClass
{
   public virtual string VirtualMethod()
   {
      return "Phuong thuc nay la virtual trong MyBaseClass";
   }
}
```

Điều này có nghĩa là chúng ta *có thể cài đặt lại* phương thức VirtualMethod() (với cùng chữ ký phương thức) trong lớp dẫn xuất của MyBaseClass. Khi chúng ta gọi phương thức này từ một thể hiện của lớp dẫn xuất thì phương thức của lớp dẫn xuất sẽ được triệu gọi chứ không phải là phương thức của lớp cơ sở.

```
class MyDerivedClass: MyBaseClass
{
   public override string VirtualMethod()
   {
      return "Phuong thuc nay duoc dinh nghia de` trong MyDerivedClass";
   }
}
```

Đoạn mã lệnh dưới đây minh họa hiệu ứng của việc ghi đè phương thức:

```
MyBaseClass obj;
```

```
obj = new MyBaseClass();
obj.VirtualMethod(); // in ra Phuong thuc nay là virtual trong MyBaseClass

obj = new MyDerivedClass();
obj.VirtualMethod(); // in ra Phuong thuc nay duoc dinh nghia de` trong
                     // MyDerivedClass
```

Ở đoạn mã lệnh trên, chúng ta thấy rằng, việc quyết định phiên bản nào của phương thức VirtualMethod (ở lớp MyBaseClass hay MyDerivedClass) được sử dụng là tùy thuộc vào *nội dung hiện tại* của đối tượng mà obj tham chiếu đến. Nói cách khác, việc quyết định phiên bản phương thức để triệu gọi được quyết định trong *thời gian thực thi chương trình* chứ không phải là *trong lúc biên dịch chương trình*! Đặc tính này còn được gọi là tính *gắn kết muộn* (late-binding) trong kỹ thuật lập trình.

Trong lớp đối tượng, các trường dữ liệu hoặc các hàm tĩnh không được khai báo là virtual.

Nếu một phương thức với chữ ký được khai báo trong cả lớp cơ sở và lớp dẫn xuất, nhưng các phương thức không được khai báo tương ứng là virtual và override, thì phiên bản phương thức ở lớp dẫn xuất được gọi là đã che dấu phiên bản ở lớp cơ sở. Trong tình huống này, phiên bản của phương thức được sử dụng để hoạt động sẽ tùy thuộc vào *kiểu dữ liệu của biến* được sử dụng để tham chiếu đến đối tượng thể hiện chứ không phải là chính đối tượng thể hiện. Điều này được thể hiện trong cách thức hoạt động của đoạn mã lệnh dưới đây:

```
class MyBaseClass
{
   public string VirtualMethod() // KHONG DUOC KHAI BAO LA virtual NUA!!!
   {
      return "Phuong thuc nay la virtual trong MyBaseClass";
   }
}

class MyDerivedClass: MyBaseClass
{
   public string VirtualMethod() // KHONG DUOC KHAI BAO LA override NUA!!!
   {
      return "Phuong thuc nay duoc dinh nghia de` trong MyDerivedClass";
   }
}
……
MyBaseClass obj; // obj la bien kieu MyBaseClass → phien ban VirtualMethod() duoc
                 // su dung DUOC QUYET DINH NGAY LUC BIEN DICH la cua MyBaseClass
obj = new MyBaseClass();
obj.VirtualMethod(); // in ra Phuong thuc nay là virtual trong MyBaseClass
```

```
obj = new MyDerivedClass();
obj.VirtualMethod(); // van in ra Phuong thuc nay là virtual trong MyBaseClass!!!
```

Trên thực tế, khi biên dịch đoạn mã lệnh tương tự như trên, trình biên dịch sẽ đưa ra cảnh báo về việc phương thức bị che giấu. Để tránh khỏi cảnh báo như vậy, bạn khai báo phương thức được định nghĩa lại trong lớp dẫn xuất thêm với từ khóa new.

3.4 Gọi phương thức với phiên bản của lớp cơ sở

C# có một cú pháp đặc biệt để cho phép trong lớp dẫn xuất có thể triệu gọi phương thức với phiên bản được cài đặt ở lớp cơ sở: `base.<MethodName>()`. Ví dụ:

```
class Student
{
   public virtual void Dispaly()
   {
      Console.WriteLine("Thong tin chung cua sinh vien...");
   }
}

class ITStudent: Student
{
   public override void Display()
   {
      base.Display();
      Console.WriteLine("Thong tin rieng doi voi sinh vien CNTT....");
   }
}
```

Lưu ý rằng cách gọi phương thức base.<MethodName>() để gọi mọi phương thức của lớp cơ sở là có thể được sử dụng cho bất kỳ phương thức nào trong lớp dẫn xuất, chứ không nhất thiết là trong cùng phương thức được ghi đè.

3.5 Lớp trừu tượng và hàm trừu tượng

C# cho phép cả lớp đối tượng và hàm được khai báo là abstract (trừu tượng). Một lớp trừu tượng thì không thể được tạo thể hiện, trong khi đó, một hàm trừu tượng thì không thể có phần cài đặt, và phải được ghi đè bởi một hàm không trừu tượng trong lớp dẫn xuất (hàm trừu tượng mặc nhiên được xem là virtual trong lớp cơ sở).

Trong một lớp trừu tượng, chỉ có thể được khai báo trường dữ liệu thành phần và các chữ ký của phương thức, không có phần cài đặt của phương thức.

```
abstract class SinhVien
{
```

```
   private bool damaged = false;                // field
   public abstract decimal DiemTrungBinh();    // abstract method
}
```

3.6 Lớp bị niêm phong và phương thức bị niêm phong

C# cho phép cả lớp đối tượng và phương thức được khai báo là niêm phong. Lớp bị niêm phong (sealed class) là lớp không được dẫn xuất thêm. Phương thức bị niêm phong (sealed method) là phương thức không thể được ghi đè ở lớp dẫn xuất.

Ví dụ về lớp bị niêm phong:

```
sealed class FinalClass
{
   // cai dat cho lop FinalClas
}

class DerivedClass : FinalClass       // LOI KHI BIEN DICH!!!
{
   ////
}
```

Ví dụ về phương thức bị niêm phong:

```
class MyClass
{
   public sealed override void FinalMethod()
   {
      // cai dat cho phuong thuc bi niem phong
   }
}

class DerivedClass : MyClass
{
   public override void FinalMethod()  // LOI KHI BIEN DICH
   {
   }
}
```

3.7 Nạp chồng toán tử

C# cung cấp cơ chế nạp chồng toán tử, cho phép cài đặt mã lệnh để quyết định cách thức một lớp đối tượng làm việc với toán tử thông thường. Cú pháp để nạp chồng một toán tử là như sau:

```
public static <return type> operator <op> (parameter list)

{ cai dat ma lenh o day}
```

Các quy tắc cần tuân thủ khi cài đặt và sử dụng phương thức nạp chồng toán tử:

- Bắt buộc phải có bổ từ truy cập `public` và `static`.
- Kiểu dữ liệu trả về là kiểu lớp đối tượng khi làm việc với các lớp đối tượng. Kiểu dữ liệu trả về không được là void.
- `op` là toán tử hai ngôi, một ngôi (unary), hoặc toán tử quan hệ. Cả hai toán tử == và != phải được cài đặt theo cặp.
- Các toán tử hai ngôi yêu cầu hai đối số, toán tử một ngôi chỉ yêu cầu một đối số..

Ví dụ dưới đây xây dựng một lớp mô phỏng kiểu dữ liệu số phức với cách sử dụng các phép toán +, - đơn giản:

```
public class ComplexNumber
{
    private int real;
    private int imaginary;

    public ComplexNumber() : this(0, 0)  // constructor
    {
    }

    public ComplexNumber(int r, int i)  // constructor
    {
        real = r;
        imaginary = i;
    }

    // Ghi de phuong thuc ToString() de hien thi so ao theo dang thong thuong:
    public override string ToString()
    {
        return(System.String.Format("{0} + {1}i", real, imaginary));
    }

    // Nap chong toan tu '+':
    public static ComplexNumber operator+(ComplexNumber a, ComplexNumber b)
    {
        return new ComplexNumber(a.real + b.real, a.imaginary + b.imaginary);
    }

    // Nap chong toan tu '-':
    public static ComplexNumber operator-(ComplexNumber a, ComplexNumber b)
    {
        return new ComplexNumber(a.real - b.real, a.imaginary - b.imaginary);
    }
}

class TestComplexNumber
{
```

```
    static void Main()
    {
        ComplexNumber a = new ComplexNumber(10, 12);
        ComplexNumber b = new ComplexNumber(8, 9);

        System.Console.WriteLine("a = {0}", a.ToString());
        System.Console.WriteLine("b = {0}", b.ToString());

        ComplexNumber c = a + b;
        System.Console.WriteLine("c = a + b = {0}", c.ToString());

        ComplexNumber d = a - b;
        System.Console.WriteLine("d = a - b = {0}", d.ToString());
    }
}
```

Như chương trình minh họa, sau khi nạp chồng toán tử + và -, bạn có thể sử dụng hai phép toán + và – đối với dữ liệu ComplexNumber một cách trực tiếp. Ở đây, kết quả nhận được sẽ là:

```
a = 10 + 12i
b = 8 + 9i
c = a + b = 18 + 21i
d = a - b = 2 + 3i
```

3.8 Bài thực hành

Bài thực hành 3.1: Quản lý sinh viên

Tóm tắt

Viết chương trình quản lý sinh viên của một trường. Sinh viên có thể học các chuyên ngành Công nghệ Thông tin, Vật lý, Ngữ văn. Mỗi chuyên ngành tương ứng có các môn học khác nhau.

- Sinh viên khoa Công nghệ Thông tin phải học 3 môn Pascal, C# và SQL.
- Sinh viên khoa Vật lý phải học 4 môn: Cơ học, Điện học, Quang học, Vật lý hạt nhân.
- Sinh viên khoa Văn phải học 2 môn Văn học cổ điển và Văn học Hiện đại

Chương trình cho phép nhập danh sách sinh viên, sau đó in danh sách sinh viên cùng với điểm trung bình của họ ra màn hình.

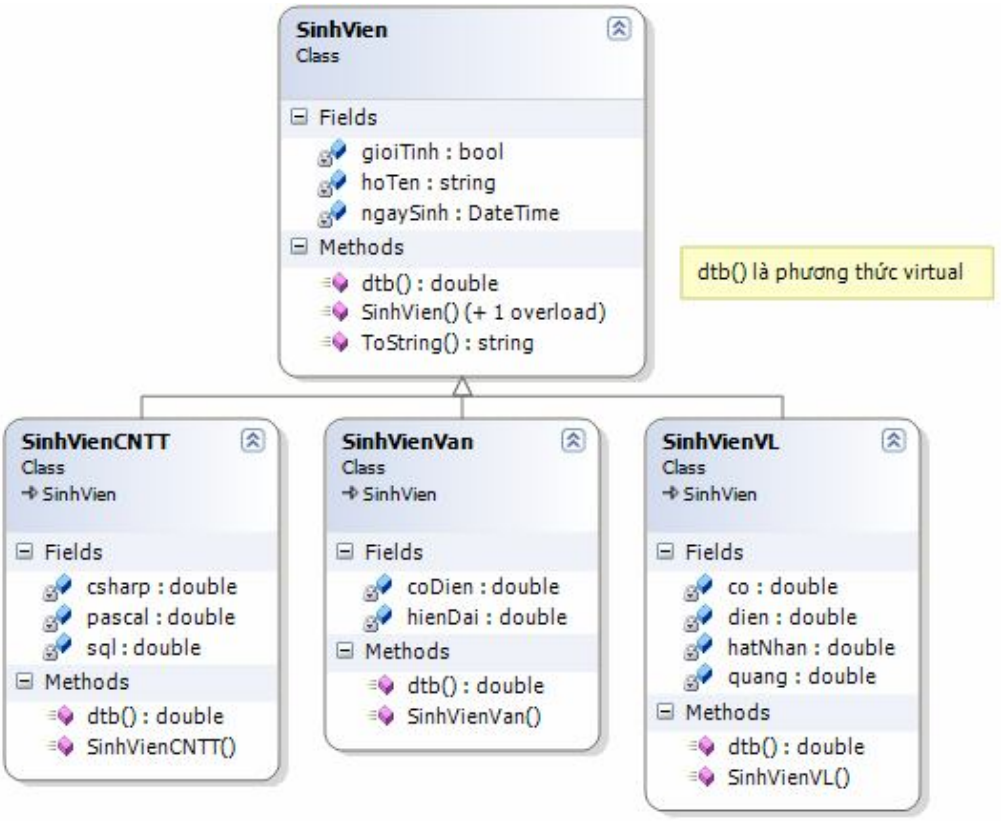
In ra danh sách những sinh viên có điểm trung bình cao trên 5.0 ra màn hình. Thông tin hiển thị có dạng Họ tên, Chuyên ngành đào tạo, Điểm trung bình.

Kỹ thuật được trình bày

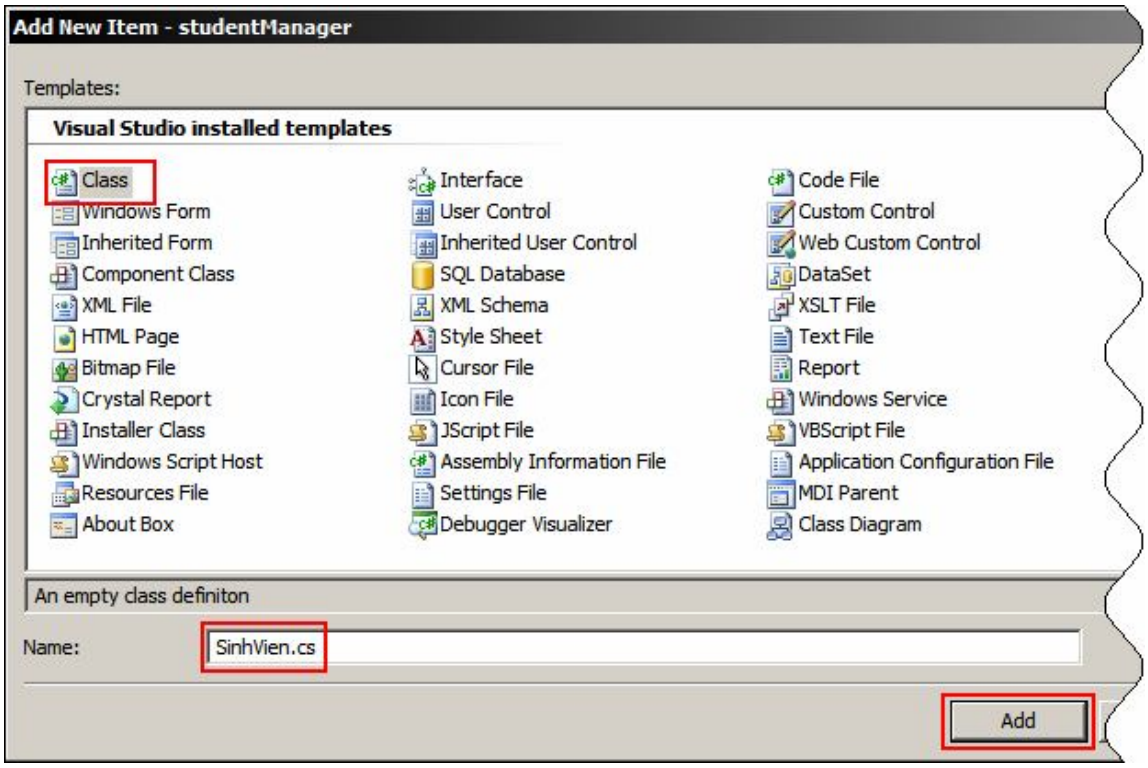
- Truy xuất tập tin có định dạng cho trước
- Sử dụng một phương thức của lớp String
- Các kỹ thuật hướng đối tượng được sử dụng trong bài toán thực tế

Trình tự thực hiện

1. Trước khi tiến hành cài đặt, ta khảo sát qua sơ đồ lớp được sử dụng. Với những mô tả khá rõ ràng trong yêu cầu bài toán, ta có được cái nhìn tổng quan về các lớp như sau:



   Lưu ý rằng, phương thức dtb() được cài đặt là virtual để chúng ta có thể override một cách cụ thể, chi tiết hơn trong các lớp kế thừa từ class SinhVien. Phương thức ToString() được cài đặt override từ lớp object để sử dụng trong việc in "nội dung" của đối tượng.
2. Tạo mới một project kiểu Console Application với tên là studentManager
3. Tại cây phân cấp Solution Explorer nhắp phải chuột và chọn Add →New Item… Trong hộp thoại hiện ra, chọn tạo mới class SinhVien.cs



4.Cài đặt các thành phần cơ bản cho lớp SinhVien

```
class SinhVien
{
    string hoTen;
    bool gioiTinh = true;
    DateTime ngaySinh;

    public SinhVien(string name, string sex, DateTime dob)...

    public SinhVien(string name, string sex, string dob)
    {
        this.hoTen = name;
        sex = sex.Trim().ToUpper();
        this.gioiTinh = (sex == "NAM" || sex == "MALE" || sex == "1");
        try
        {
            this.ngaySinh = DateTime.Parse(dob);
        }
        catch (Exception ex)
        {
            this.ngaySinh = DateTime.Today;
        }
    }

    /// <summary> ...
    public virtual double dtb()
    {
        //throw new System.NotImplementedException();
        return -10;
    }

    public override string ToString()
    {
        return string.Format("{0, 30} {1:0.00}", this.hoTen, this.dtb());
    }
}
```

5. Bổ sung thêm các class SinhVienCNTT, SinhVienVan, SinhVienVL theo phân tích thiết kế lớp từ trước. Dưới đây là phần mô tả cài đặt cho lớp SinhVienVan. Hai lớp còn lại SinhVienCNTT, SinhVienVL được cài đặt một cách tương tự.

```
using System;
using System.Collections.Generic;
using System.Text;

namespace studentManager
{
    class SinhVien...

    public class SinhVienCNTT : SinhVien...

    public class SinhVienVan : SinhVien
    {
        double coDien, hienDai;

        public override double dtb()
        {
            return (coDien + hienDai) / 2;
        }

        public SinhVienVan(string name, string sex, string dob, double classic, double modern)
            :base(name, sex, dob)
        {
            this.coDien = classic;
            this.hienDai = modern;
        }
    }

    public class SinhVienVL : SinhVien...
}
```

6. Trong phần chương trình (tập tin Program.cs) chúng ta thực hiện yêu cầu bài toán như sau:

```
Console.Write("So luong sinh vien: ");
int n = int.Parse(Console.ReadLine());

SinhVien[] sv = new SinhVien[n];
for (int i = 0; i < n; i++)
{
    Console.WriteLine("Nhap thong tin cho sinh vien thu {0}", i+1);
    Console.Write("\tHo ten: ");
    string hoTen = Console.ReadLine();
    Console.Write("\tGioi tinh: [1]Nam [0]Nu");
    string gioiTinh = Console.ReadLine();
    Console.Write("Ngay sinh: ");
    string ngaySinh = Console.ReadLine();
    Console.Write("Chuyen nganh cua sinh vien [1]Van [2]CNTT [any key]Vat ly]");

    char k = char.Parse(Console.ReadLine());
    switch (k)
    {
        case '1':
            #region Nhap diem cho sinh vien van
            double coDien, hienDai;
            Console.Write("\tDiem van co dien: ");
            coDien = double.Parse(Console.ReadLine());
            Console.Write("\tDiem van hien dai: ");
            hienDai = double.Parse(Console.ReadLine());

            sv[i] = new SinhVienVan(hoTen, gioiTinh, ngaySinh, coDien, hienDai);
            break;
            #endregion
        case '2':
            Nhap diem cho sinh vien van
        default:
            Nhap diem cho sinh vien Vat ly
    }
}

//in danh sach sinh vien
for (int i = 0; i < n; i++)
    Console.WriteLine("{0,3}. {1}", i+1, sv[i]);
```

Yêu cầu thêm

- In ra 3 sinh viên có điểm trung bình cao nhất trường.
- Chỉnh sửa để người sử dụng có thể nhập danh sách mà không biết trước số lượng sinh viên (sử dụng vòng lặp while, do, …)
- Chỉnh sửa để có thể nhập dữ liệu các sinh viên từ file.

3.2 Bài thực hành trên Winform

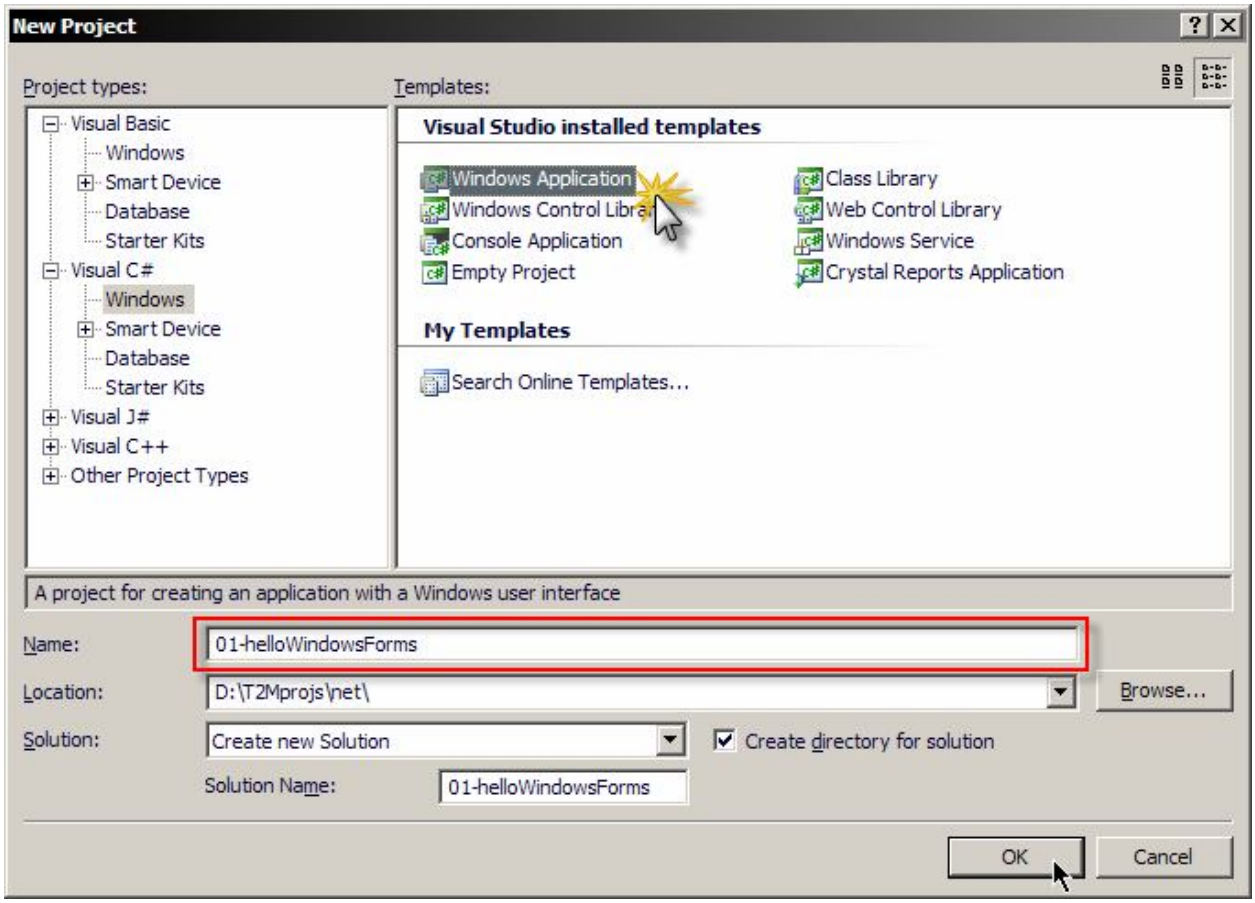
Bài thực hành 3.2.1 helloWinForms

Kỹ thuật được trình bày

- Cấu trúc của và cơ chế hoạt động của một project Windows Form Application.
- Cơ chế xử lý sự kiện của các Control trong một Windows Form
- Một số phương thức, thuộc tính, sự kiện quan trọng của các điều khiển trong một Windows Form.

Trình tự thực hiện

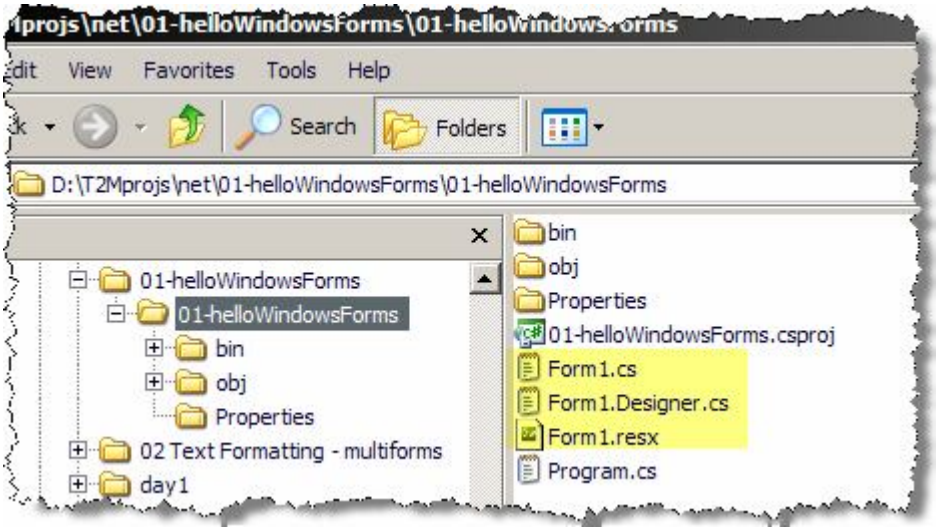
1. Tạo mới một ứng dụng kiểu Windows Form Application với tên là 01-helloWindowsForm như hình vẽ



2. Theo mặc định, một solution với một project được tạo ra. Project này có một lớp Form1.

   Khảo sát nội dung của project trong Windows Explorer, chúng ta sẽ thấy cấu trúc

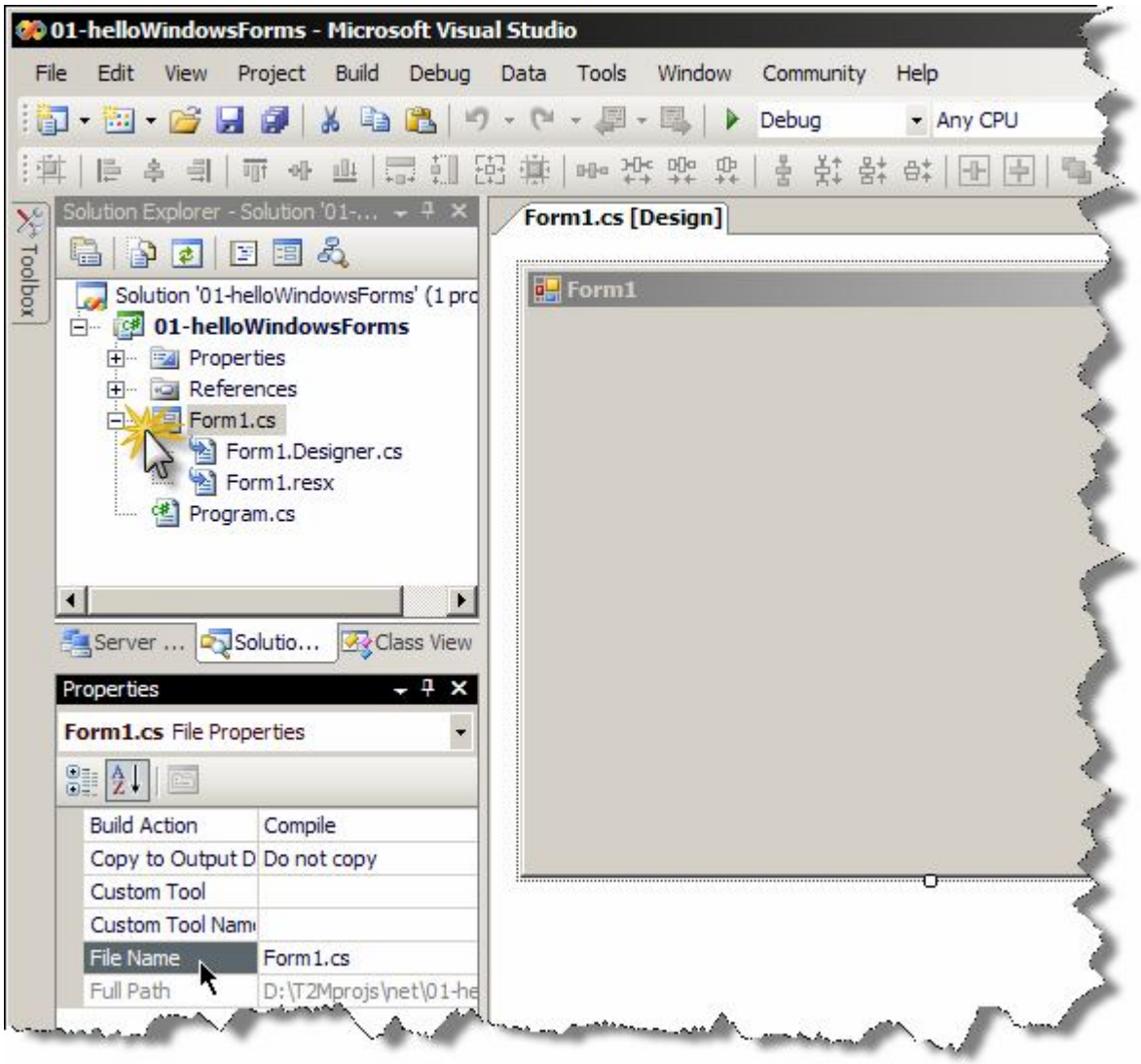
của thư mục và các tập tin tương tự như hình dưới:



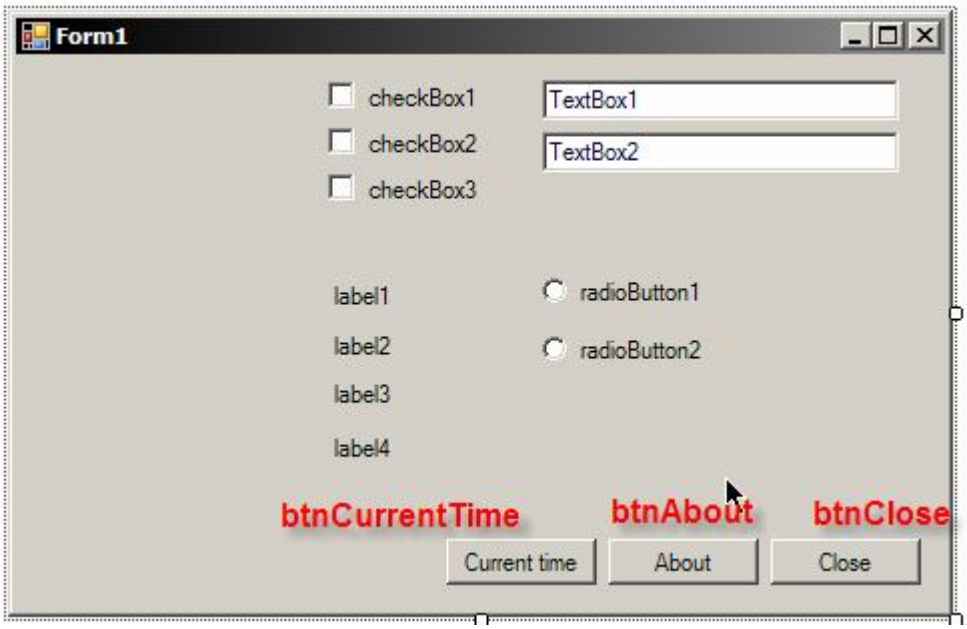
Có thể thấy, mỗi Form được tạo ra tương ứng với 3 tập tin có tiếp đàu ngữ là giống nhau, lấy ví dụ là Form1

- Form1.Designer.cs: chứa các mã lệnh do Form Designer tự sinh ra tương ứng với các thao tác do người sử dụng *kéo thả các Control từ ToolBox vào bề mặt Form* hay *thực hiện các thiết lập đối với các Control*.
- Form1.cs: chứa phần mã lệnh và khai báo thêm do người sử dụng cài đặt.
- Form1.resx: chứa các mô tả, khai báo về các tài nguyên được sử dụng trong Form.

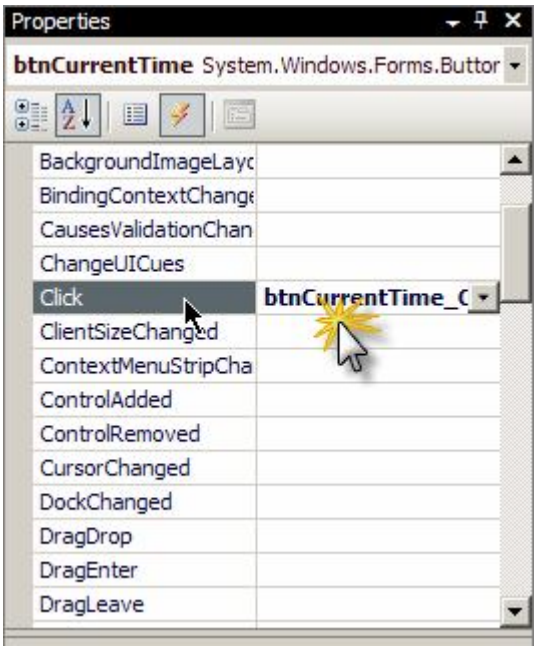
3. Chúng ta cũng có thể quan sát cấu trúc của solution hay project bằng cách khảo sát cửa sổ Solution Explorer:



4. Từ cửa sổ Solution Explorer, đổi tên tập tin Form1.cs thành FormMain.cs. Để ý rằng, cả ba tập tin liên quan đến Form1 đều được thay đổi theo một cách đồng bộ.
5. Thiết kế giao diện cho FormMain như hình vẽ



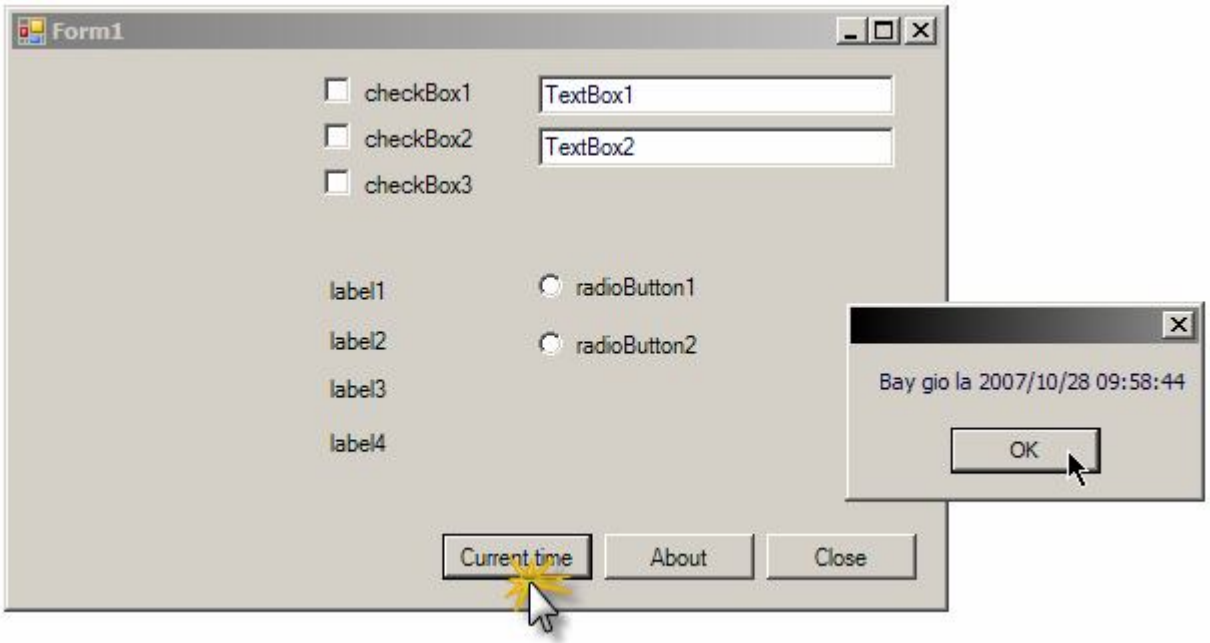
6. Bước tiếp theo, chúng ta sẽ thực hiện cài đặt phương thức xử lý sự kiện Click của nút bấm btnCurrentTime:
   a. Chọn điều khiển nút bấm btnCurrentTime trong cửa số thiết kế Form.
   b. Ở trang Event trong cửa sổ Properties Windows, nhắp đúp chuột vào sự kiện Click (xem hình vẽ dưới). Form Designer sẽ sinh ra phương thức xử lý sự kiện có tên mặc định là btnCurrentTime_Click(…). (Phương thức xử lý sự kiện được mặc định đặt tên là <tênĐiềuKhiển>_<TênSựKiện>)



Soạn thảo phần mã lệnh cho phương thức này như sau:

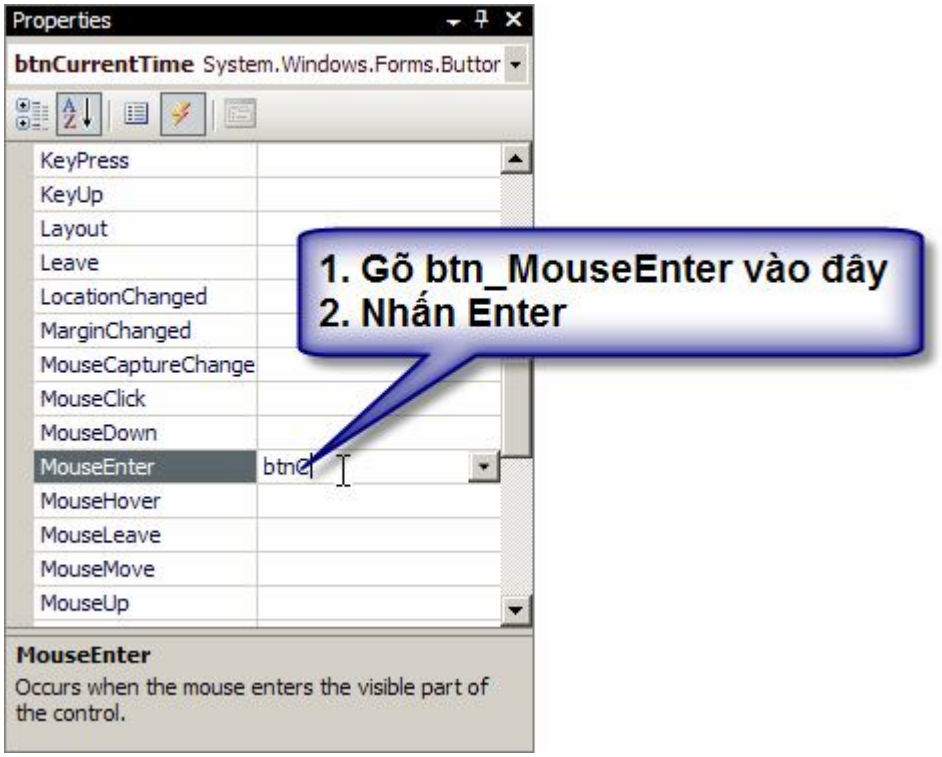
```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

namespace _1_helloWindowsForms
{
    public partial class FormMain : Form
    {
        public FormMain()
        {
            InitializeComponent();
        }

        private void btnCurrentTime_Click(object sender, EventArgs e)
        {
            MessageBox.Show("Bay gio la " + DateTime.Now.ToString());
        }
    }
}
```

7. Thực hiện chạy chương trình, khi nhấn vào nút bấm btnCurrentTime, một hộp thông báo được hiển thị ra như hình vẽ



8. Thực ra chúng ta có thể tự đặt tên cho phương thức xử lý sự kiện. Chẳng hạn, để cài đặt phương thức xử lý sự kiện MouseEnter cho nút bấm btnCurrentTime, trong cửa sổ Properties ở trang Events, tìm đến mục MouseEnter và:

a. Nhập vào tên phương thức xử lý sự kiện: btn_MouseEnter
b. Nhấn Enter
c. FormDesigner sẽ tạo ra phương thức với tên tương ứng



d. Tiến hành cài đặt mã lệnh cho phương thức xử lý sự kiện trên như sau:

```
private void btn_MouseEnter(object sender, EventArgs e)
{
    btnCurrentTime.ForeColor = Color.Red;
}
```

9. Tương tự, chúng ta cài đặt tiếp phương thức xử lý sự kiện MouseLeave cho nút bấm btnCurrentTime như sau

```
private void btn_MouseLeave(object sender, EventArgs e)
{
    btnCurrentTime.ForeColor = SystemColors.ControlText;
}
```

10. Chạy chương trình và quan sát kết quả: Điều khiển nút bấm btnCurrentTime sẽ có hiệu ứng *mouse hover* khá ấn tượng: khi rê con trỏ chuột vào nút bấm btnCurrentTime, màu chữ của nó sẽ đổi sang màu đỏ; màu chữ của nút bấm trở

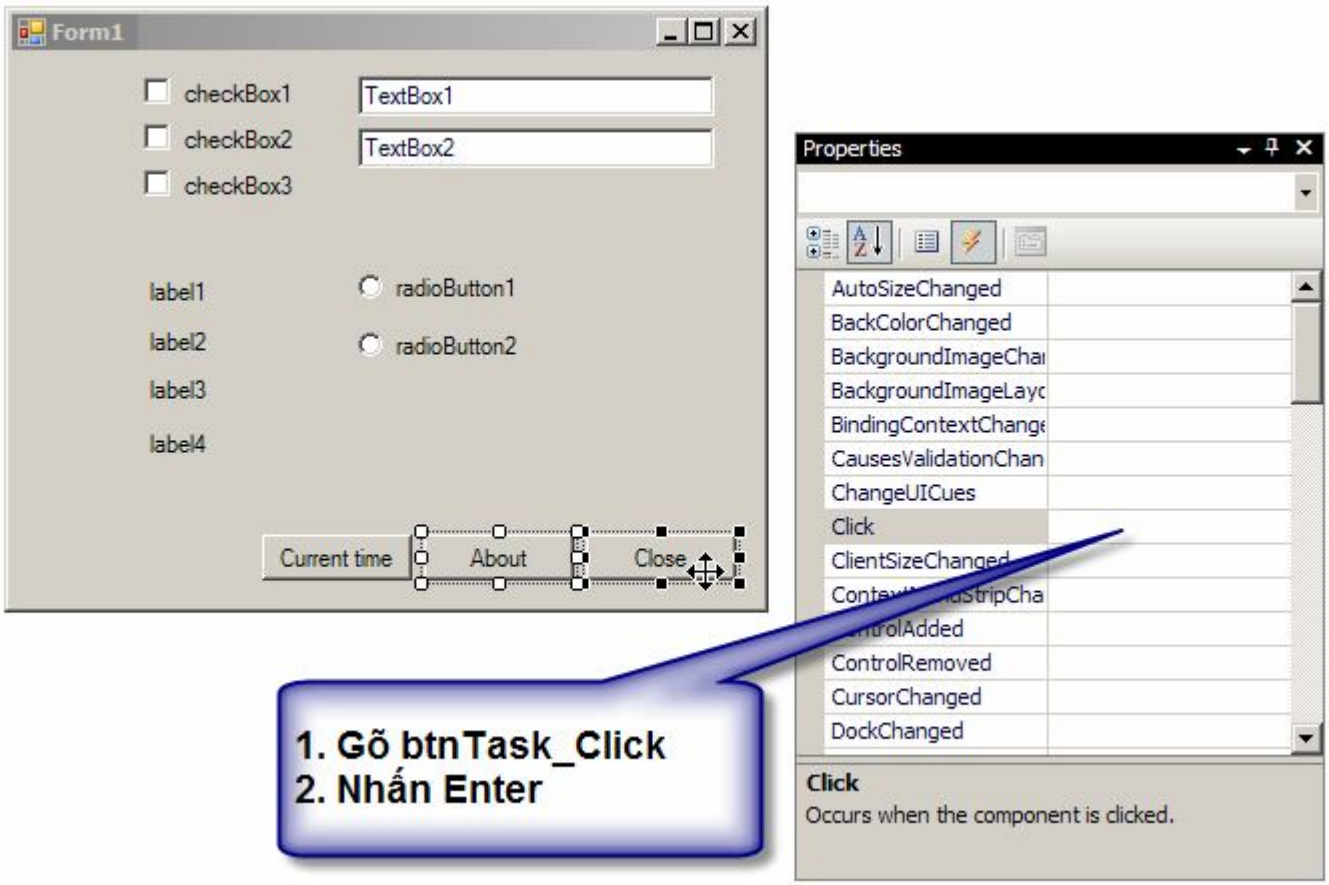
thành bình thường (màu ControlText) khi con trỏ chuột rê ra khỏi nút bấm.

11. Để tìm hiểu kỹ hơn bản chất của việc gắn kết phương thức xử lý sự kiện, chúng ta nhắp đúp chuột vào FormMain.Designer.cs trong cửa sổ Solution Explorer để xem phần nội dung được sinh ra bởi Form Designer:

```
#region Windows Form Designer generated code

/// <summary>
/// Required method for Designer support - do not modify
/// the contents of this method with the code editor.
/// </summary>
private void InitializeComponent()
{
    this.btnClose = new System.Windows.Forms.Button();
    this.btnAbout = new System.Windows.Forms.Button();
    this.btnCurrentTime = new System.Windows.Forms.Button();
    //
    // btnCurrentTime
    //
    this.btnCurrentTime.Anchor = ((System.Windows.Forms.AnchorStyles)((System.Windows.Forms
    this.btnCurrentTime.Location = new System.Drawing.Point(215, 241);
    this.btnCurrentTime.Name = "btnCurrentTime";
    this.btnCurrentTime.Size = new System.Drawing.Size(75, 23);
    this.btnCurrentTime.TabIndex = 0;
    this.btnCurrentTime.Text = "Current time";
    this.btnCurrentTime.UseVisualStyleBackColor = true;
    this.btnCurrentTime.MouseLeave += new System.EventHandler(this.btn_MouseLeave);
    this.btnCurrentTime.Click += new System.EventHandler(this.btnCurrentTime_Click);
    this.btnCurrentTime.MouseEnter += new System.EventHandler(this.btn_MouseEnter);
    //
    // FormMain
    //
    this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
    this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
    this.ClientSize = new System.Drawing.Size(464, 276);
    this.Controls.Add(this.btnCurrentTime);
    this.Controls.Add(this.btnAbout);
    this.Controls.Add(this.btnClose);
    this.Name = "FormMain";
    this.Text = "Form1";
    this.ResumeLayout(false);
    this.PerformLayout();

}

#endregion

private System.Windows.Forms.Button btnClose;
private System.Windows.Forms.Button btnAbout;
private System.Windows.Forms.Button btnCurrentTime;
private System.Windows.Forms.CheckBox checkBox1;
private System.Windows.Forms.CheckBox checkBox2;
```

Chú ý những phần được tô sáng trong hình vẽ nói trên; từ đó suy ra được bản chất của việc gắn kết phương thức xử lý sự kiện trong khi thiết kế.

12. Đóng file nội dung FormMain.Designer.cs lại. Các bước tiếp theo sẽ minh họa cách thức dùng chung một phương thức xử lý sự kiện cho nhiều đối tượng khác nhau.
13. Trong cửa sổ thiết kế của FormMain, thực hiện
    a. Chọn cả hai đối tượng btnClose và btnAbout
    b. Trong trang Events của cửa sổ Properties, gõ tên phương thức xử lý sự kiện Click cho cả hai điều khiển nút bấm này là **btnTask_Click** rồi nhấn Enter (xem hình vẽ)



14. Thực hiện cài đặt mã lệnh cho phương thức này như sau:

```
private void btnTask_Click(object sender, EventArgs e)
{
   if (sender == btnClose)
      this.Close();
   else if (sender == btnAbout)[1]
      MessageBox.Show("Day la chuong trinh minh hoa", "Thong bao");
}
```

Trong phương thức trên, chúng ta sử dụng đối số sender để nhận biết điều khiển nào phát sinh sự kiện. Chúng ta cũng có thể thực hiện như thế này:

```
private void btnTask_Click(object sender, EventArgs e)
{
   string stTask = (sender as Button).Text; [2]
   if (stTask == "Close")
      this.Close();
   else if (stTask == "About")
      MessageBox.Show("Day la chuong trinh minh hoa", "Thong bao");
}
```

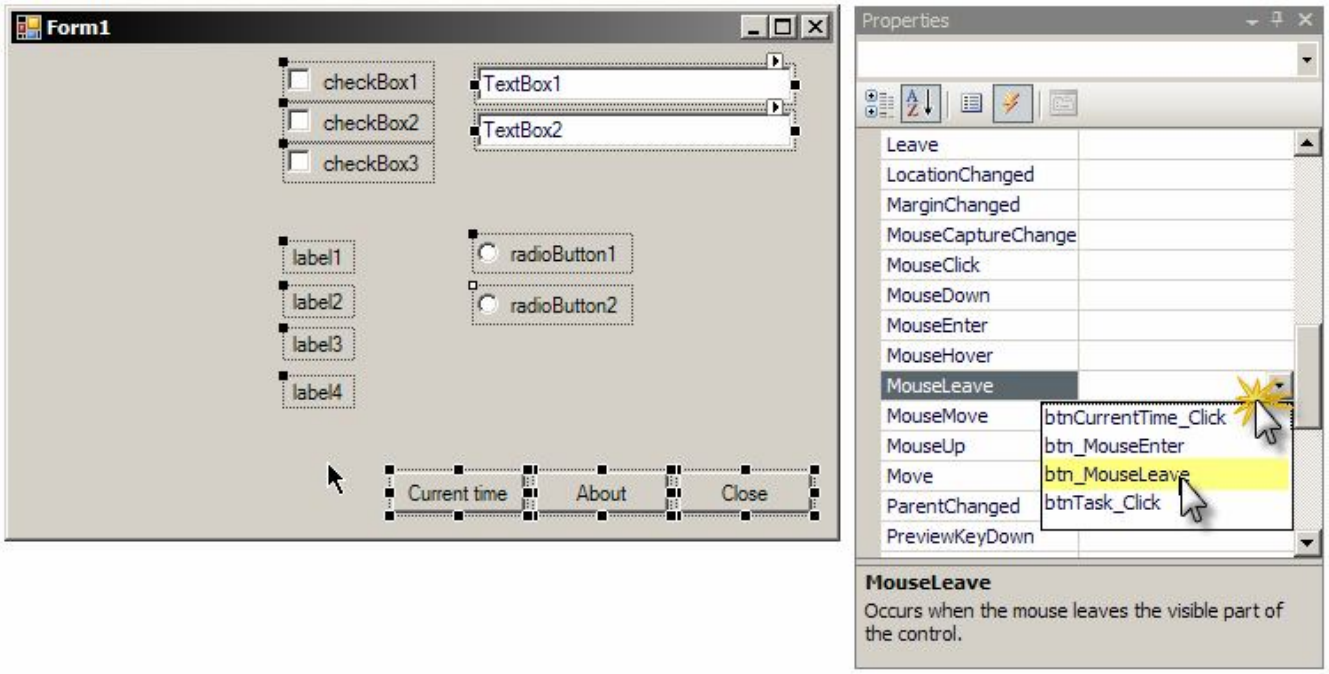
15. Bây giờ, chúng ta tinh chỉnh thêm để chương trình hỗ trợ hiệu ứng *mouse hover cho tất cả các điều khiển* trong form:
    a. Sửa lại phần mã nguồn cho 2 phương thức xử lý sự kiện btn_MouseEnter và btn_MouseLeave như sau:

```
private void btn_MouseEnter(object sender, EventArgs e)
{
   (sender as Control).ForeColor = Color.Red;
}

private void btn_MouseLeave(object sender, EventArgs e)
{
   (sender as Control).ForeColor = SystemColors.ControlText;
```

[1] Thực ra không nhất thiết phải có nhánh else if, chỉ cần else là đủ, bởi vì ở đây chúng ta chỉ áp dụng phương thức này cho hai điều khiển btnClose và btnAbout!.
[2] Phép chuyển kiểu (sender as Button) trong câu lệnh này là thành công vì cả btnClose và btnAbout đều là các điều khiển kiểu Button

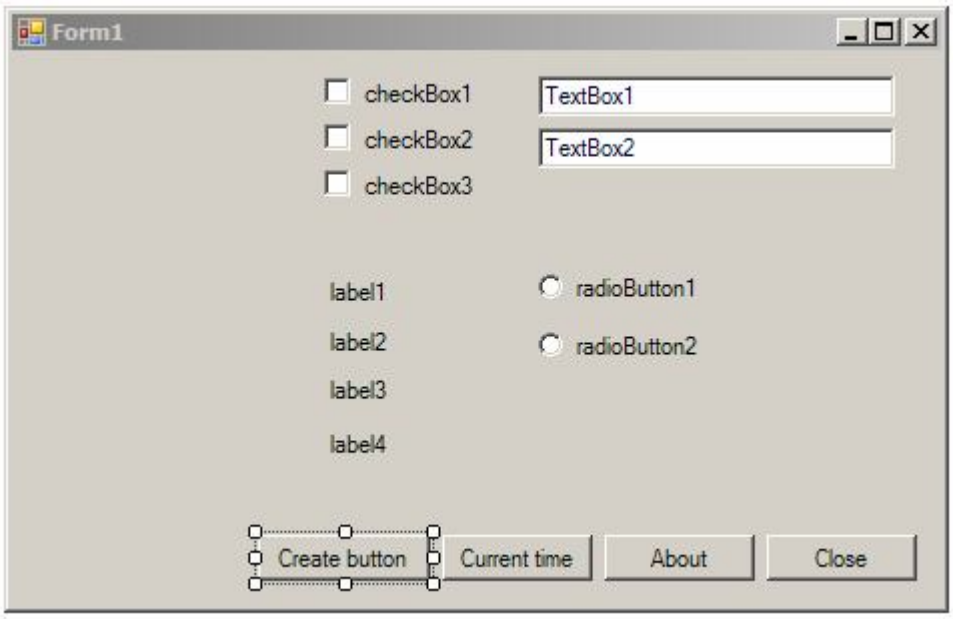
}

b. Trong phần FormDesigner, chọn tất cả các đối tượng trên bề mặt Form.
c. Trong cửa sổ Properties, chọn phương thức xử lý sự kiện MouseLeave cho *tất cả các đối tượng đang chọn* là btn_MouseLeave (xem hình vẽ)



d. Làm tương tự để gán phương thức xử lý sự kiện MouseEnter cho tất cả các điều khiển nói trên là btn_Enter.
e. Chạy chương trình để xem hiệu ứng: khi rê con trỏ chuột qua các điều khiển, font chữ của chúng sẽ được đổi thành màu đỏ.

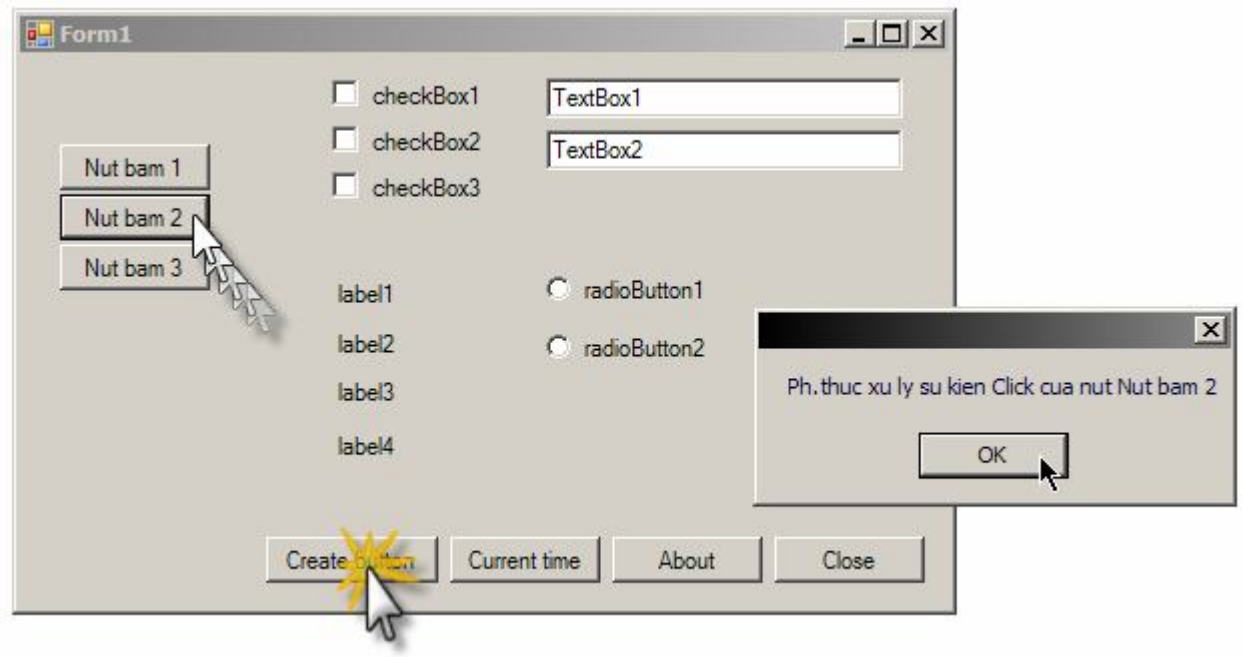
16. Trong bước 11, chúng ta đã biết được cách thức đưa một thành phần điều khiển vào giao diện của một Windows Form thông qua mã lệnh (bằng cách tìm hiểu phần mã sinh ra bởi Form Designer). Bây giờ, chúng ta sẽ áp dụng để thực hiện thêm các điều khiển vào Form và gán phương thức xử lý sự kiện cho chúng trong thời gian thực thi chương trình
    a. Bổ sung vào Form một nút bấm btnCreateButton



b. Cài đặt phương thức xử lý sự kiện Click cho nút bấm này như sau:

```
using System.Text;
using System.Windows.Forms;

namespace _1_helloWindowsForms
{
    public partial class FormMain : Form
    {
        int soLuong = 0;

        public FormMain()...

        private void btnCurrentTime_Click(object sender, EventArgs e)...

        private void btn_MouseEnter(object sender, EventArgs e)...

        private void btn_MouseLeave(object sender, EventArgs e)...

        private void btnTask_Click(object sender, EventArgs e)...

        void btn_Click(object nutBam, EventArgs thamSo)
        {
            MessageBox.Show("Ph.thuc xu ly su kien Click cua nut "
                + (nutBam as Button).Text);
        }

        private void btnCreateButton_Click(object sender, EventArgs e)
        {
            soLuong++;
            Button btn = new Button();
            btn.Location = new System.Drawing.Point(20, 20 + soLuong*25);
            btn.Name = "btn" + soLuong.ToString();
            btn.Size = new System.Drawing.Size(75, 23);
            btn.Text = "Nut bam " + soLuong.ToString();
            btn.UseVisualStyleBackColor = true;
            btn.Click += new System.EventHandler(btn_Click);
            btn.MouseEnter += new System.EventHandler(btn_MouseEnter);
            btn.MouseLeave += new System.EventHandler(btn_MouseLeave);
            this.Controls.Add(btn);
        }
    }
}
```

c. Chạy chương trình và quan sát kết quả.



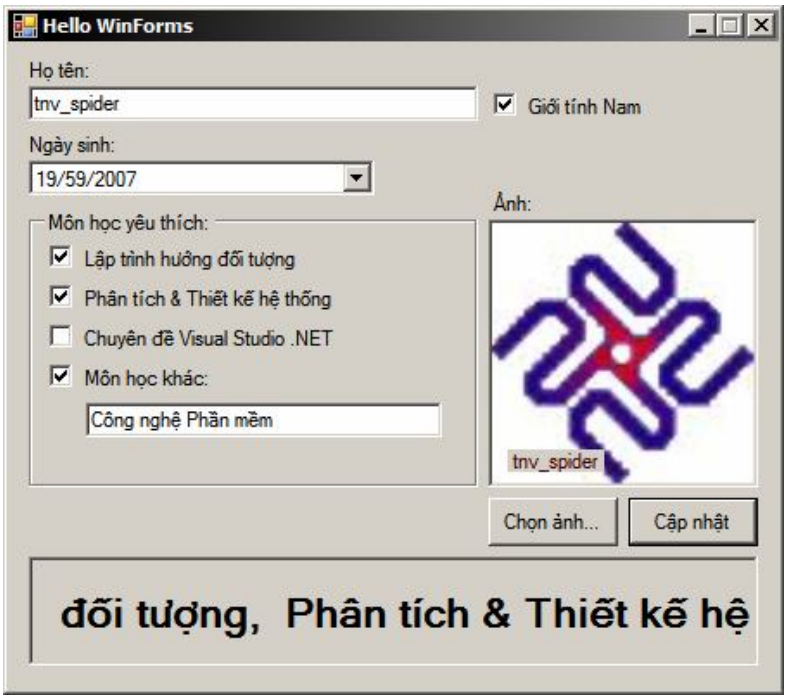
Mở rộng

- Hãy tìm hiểu ý nghĩa của việc cài đặt mã lệnh ở bước 15.a: (sender as Control). Có thể sử dụng phép ép kiểu nào khác không? Tại sao?
- Điều chỉnh trong giao diện chương trình, trong đó có một số điều khiển (Label, TextBox, RadioButton, CheckBox hoặc Button) sử dụng màu khác với màu mặc định (là SystemColors.ControlText). Khi đó, hiệu ứng *mouse hover* hoạt động không đúng nữa. Hãy chỉnh sửa chương trình để khắc phục phát sinh này.

Bài thực hành 3.2.2 usingControls

Tóm tắt

Xây dựng chương trình điền thông tin cá nhân như minh họa



Kỹ thuật được trình bày

- Giới thiệu một ứng dụng WinForms cơ bản
- Cách thức lưu file với nội dung tiếng Việt
- Các thành phần điều khiển cơ bản: Button, Label, TextBox, PictureBox, Timer, …
- Nạp một ảnh từ file

Trình tự thực hiện

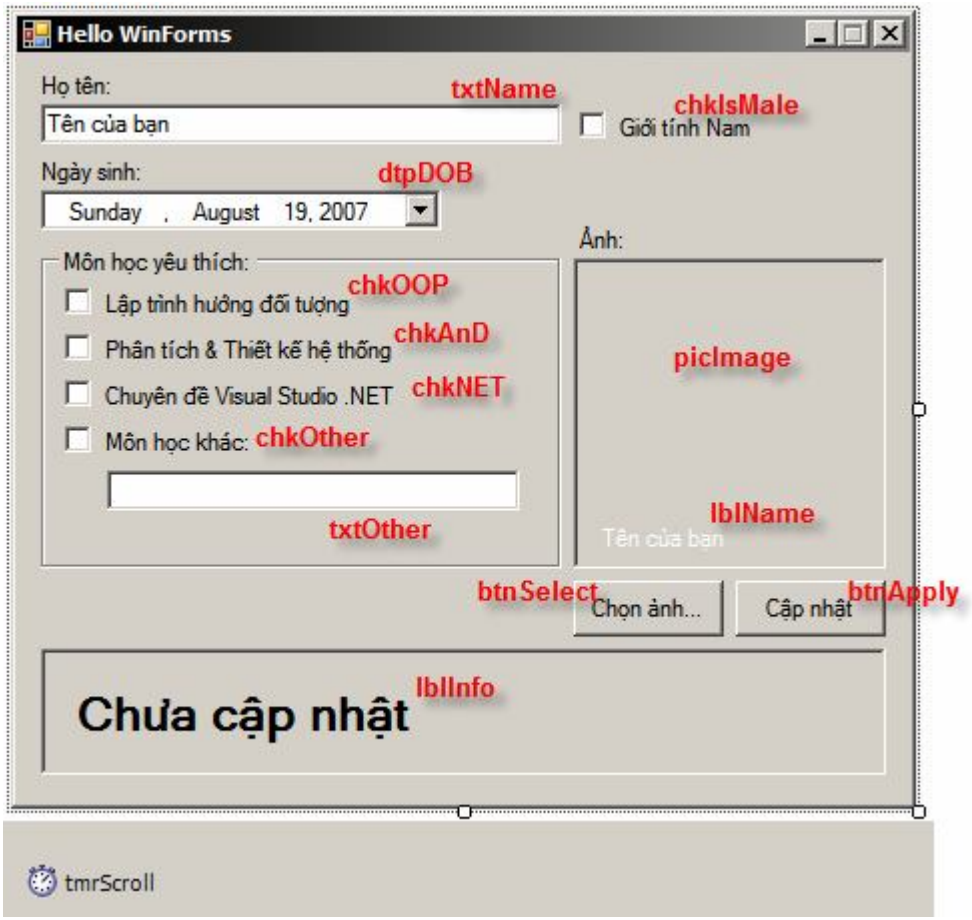
1. Tạo mới một project loại Windows Application, đặt tên là usingControls
2. Theo mặc định, một lớp Form1 được sinh ra. Chỉnh sửa các thuộc tính của Form1 với các giá trị như bảng dưới:

Thuộc tính	Giá trị	Ghi chú
Name	FormMain	

Text	Hello WinForms	Tiêu đề của cửa sổ
FormBorderStyle	FixedSingle	Kích thước của cửa sổ sẽ không được thay đổi khi chạy chương trình
MaximizeBox	False	Vô hiệu hóa nút Maximize của cửa sổ

Chú ý rằng, những thuộc tính có thay đổi giá trị so với mặc định sẽ được hiển thị trong cửa sổ Properties dưới dạng chữ in đậm

3. Thiết kế giao diện của form như minh họa. Lưu ý, với mỗi điều khiển bạn đưa vào form, nếu dự định truy xuất nó trong phần mã nguồn khi lập trình thì hãy đặt tên nó thay vì để như tên mặc định.

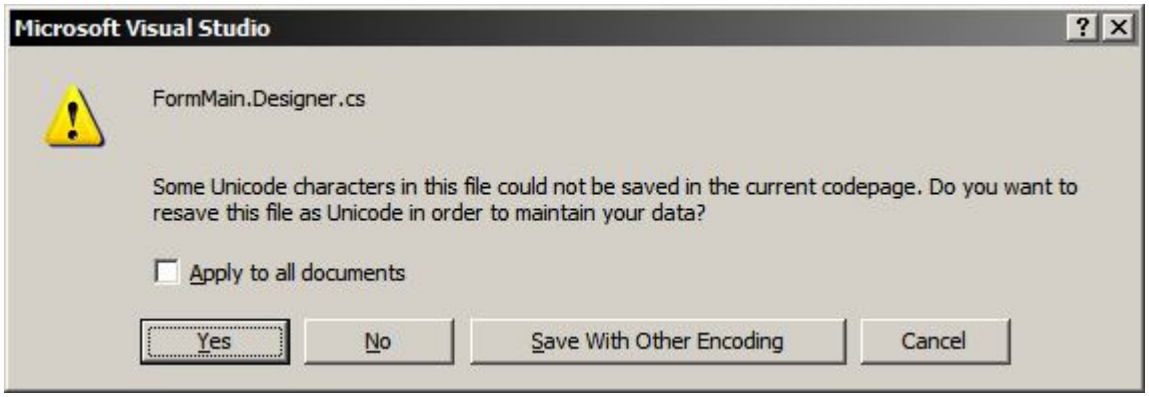


Chỉnh sửa thuộc tính của một số đối tượng như sau:

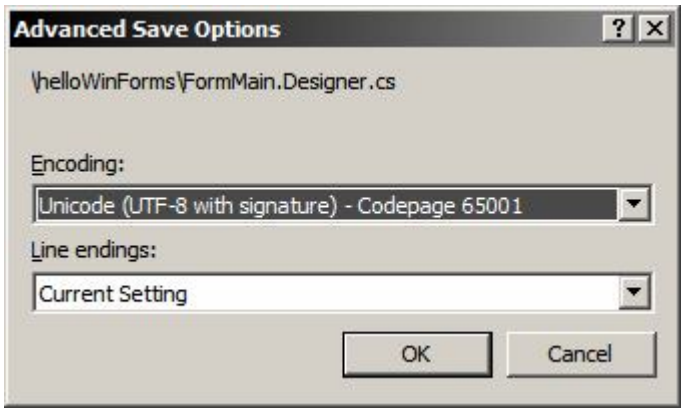
Điều khiển	Thuộc tính	Giá trị

dtpDOB	Format	Custom
	CustomFormat	dd/MM/yyyy
txtOther	Enable	False
lblInfo	Font	Chọn font thích hợp, in đậm
picImage	SizeMode	StretchImage
lblName	BackColor	Transparent (Web)
tmrScroll	Interval	120

4. Nhấn Ctrl + S để lưu nội dung project. Do chúng ta có sử dụng ký tự tiếng Việt trong Form nên Visual Studio có hiển thị hộp thoại để yêu cầu chỉ định bảng mã lưu ký tự:



Nhấn nút "Save With Other Encoding" để chọn bảng mã thích hợp – sau đó bạn có thể chọn cách lưu theo UTF8 như hình dưới (cũng có thể chọn tùy chọn Unicode – Codepage 1200):



5. Cài đặt phần mã lệnh cho sự kiện Click của nút bấm btnSelectImage như sau:

```
private void btnSelectImage_Click(object sender, EventArgs e)
{
    OpenFileDialog dlgOpen = new OpenFileDialog();
    dlgOpen.Filter = "Tập tin ảnh|*.bmp;*.jpg;*.gif|File tùy ý (*.*)|*.*";
    if (dlgOpen.ShowDialog(this) == DialogResult.OK)
    {
        try
        {
            picImage.Image = Image.FromFile(dlgOpen.FileName);
        }
        catch (Exception exc)
        {
            MessageBox.Show(String.Format("Không thể nạp file ảnh [{0}]", dlgOpen.FileName));
        }

    }
}
```

Khi người sử dụng nhấn vào nút này, một hộp thoại sẽ hiện ra cho phép chọn ảnh. Chỉ các tập tin có phần mở rộng là BMP, JPG, GIF mới được hiển thị để lựa chọn. Điều này được thiết lập thông qua thuộc tính Filter của đối tượng dlgOpen (thuộc lớp OpenFileDialog).

6. Khi người sử dụng gõ tên của họ vào txtName thì nội dung của lblName cũng thay đổi theo. Muốn vậy, ta cài đặt mã lệnh cho sự kiện TextChanged của txtName như (1) – xem minh họa code ở dưới
7. Đối tượng txtOther chỉ được sử dụng (Enabled) khi mà chkOther được check vào, do đó ta cũng cài đặt mã lệnh cho sự kiện CheckChanged của chkOther như (2)
8. Khi nhấn nút "Cập nhật" thì nội dung của lblInfo được cập nhật theo như phần mã lệnh cài đặt cho sự kiện Click của btnUpdate (3)
9. Người sử dụng có thể bật tắt chế độ cuộn nội dung dòng chữ lblInfo bằng cách nhấn chuột vào nó. Cài đặt mã lệnh cho sự kiện Click của lblInfo như (5)

10. Để cuộn nội dung dòng chữ, cài đặt mã lệnh cho sự kiện Tick của tmrScroll như (4)

```
private void txtName_TextChanged(object sender, EventArgs e)
{
    lblName.Text = txtName.Text;
}
```

1

```
private void chkOther_CheckedChanged(object sender, EventArgs e)
{
    txtOther.Enabled = chkOther.Checked;
}
```

2

```
private void btnApply_Click(object sender, EventArgs e)
{
    lblInfo.Text = (chkIsMale.Checked ? "Anh " : "Chị ") + txtName.Text + ". Môn học yêu thích: ";
    if (chkOOP.Checked)
        lblInfo.Text += " " + chkOOP.Text + ", ";
    if (chkAnD.Checked)
        lblInfo.Text += " " + chkAnD.Text + ", ";
    if (chkNET.Checked)
        lblInfo.Text += " " + chkNET.Text + ", ";
    if (chkOther.Checked && txtOther.Text.Trim() != "")
        lblInfo.Text += " " + txtOther.Text + ", ";
}
```

3

```
private void tmrScroll_Tick(object sender, EventArgs e)
{
    lblInfo.Text = lblInfo.Text.Substring(1) + lblInfo.Text.Substring(0, 1);
}
```

4

```
private void lblInfo_Click(object sender, EventArgs e)
{
    tmrScroll.Enabled = !tmrScroll.Enabled;
}
```

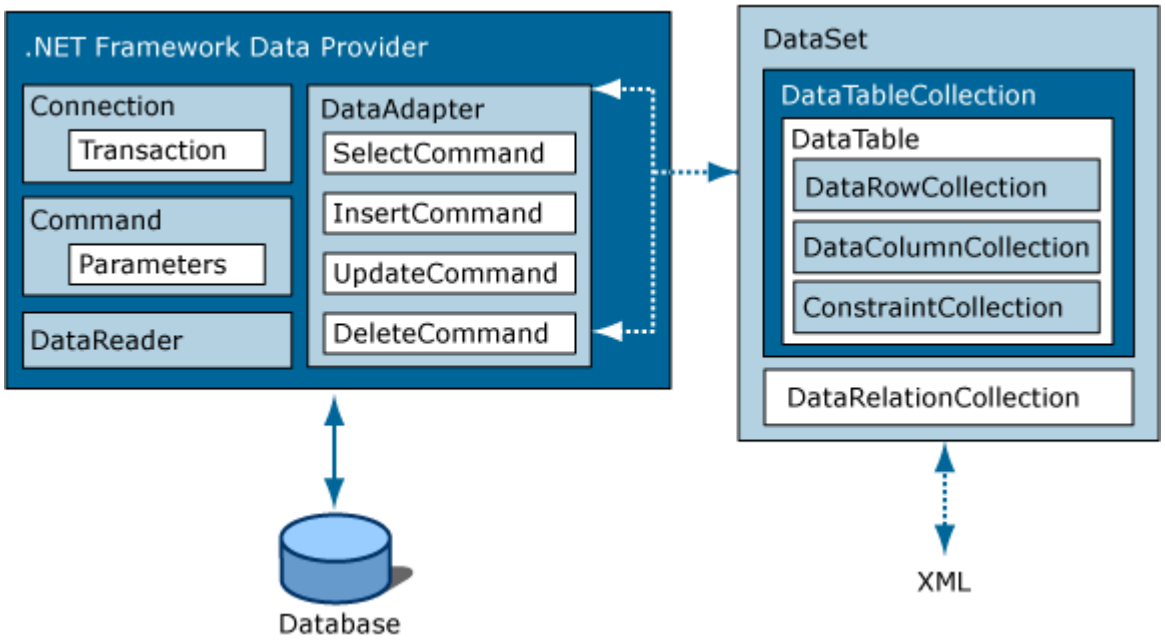
5

CHƯƠNG 4. XỬ LÝ DỮ LIỆU VỚI ADO.NET

Xử lý dữ liệu là nhiệm vụ phổ biến và quan trọng của nhiều chương trình ứng dụng. Dữ liệu được truy xuất, xử lý của một chương trình ứng dụng có thể là một tập tin văn bản, tập tin các bản ghi, hay là một nguồn dữ liệu từ CSDL nào đó. .NET Framework cung cấp một lượng lớn các thành phần giao diện (Win Forms, Web Forms) hỗ trợ cho việc trình bày, kết buộc (bind) dữ liệu. Cùng với đó là nền tảng xử lý dữ liệu ADO.NET cung cấp cách thức làm việc với nhiều loại nguồn dữ liệu khác nhau một cách linh động. Do tính chất quan trọng của việc xử lý dữ liệu trong một ứng dụng cùng với sự phức tạp của ADO.NET, trước khi bắt tay vào thực hiện các bài tập thực hành, chúng ta khảo sát qua một số điểm lý thuyết cơ bản.

4.1 Kiến trúc tổng quan của ADO.NET

Kiến trúc của ADO.NET được mô tả như hình dưới, bao gồm hai thành phần chính: *Thành phần truy cập nguồn dữ liệu* và *thành phần lưu trữ xử lý dữ liệu*.



Thành phần thứ nhất:.NET Framework Data Provider được thiết kế để thực hiện các thao tác kết nối, gửi các lệnh xử lý đến CSDL (thành phần này còn được gọi với một tên khác là *lớp kết nối – Connectectivity Layer*). Trong ADO.NET, có 4 đối tượng chính với các chức năng cơ bản như sau:

- Connection: giúp thực hiện kết nối đến các CSDL
- Command: giúp truy cập đến CSDL và thực hiện các phát biểu SQL hay thủ tục lưu trữ sẵn (stored procedure) của CSDL
- DataReader: dùng để đọc nhanh nguồn dữ liệu, chỉ được duyệt tuần tự theo chiều tiến của các record
- DataAdapter: dùng để chuyển dữ liệu truy vấn được cho các đối tượng lưu trữ và xử lý (DataSet, DataTable). DataAdapter chủ yếu thực hiện các thao tác như SELECT, INSERT, UPDATE, DELETE

Về mặt thực chất, thành phần .NET Framework Data Provider cung cấp giao diện lập trình chung để làm việc với các nguồn dữ liệu. Mỗi *nhà cung cấp* [3] đặc thù sẽ đưa ra một loại data provider riêng. Dưới đây là bảng mô tả giao diện lập trình cùng với các lớp cơ bản của các data provider mà ADO.NET cung cấp sẵn:

Interface	SQL Server Provider	Oracle Provider	OLEDB Provider	ODBC Provider
IDbConnection	SqlConnection	OracleConnection	OledbConnection	OdbcConnection
IDbDataAdapter	SqlDataAdapter	OracleDataAdapter	OledbDataAdapter	OdbcDataAdapter
IDbCommand	SqlCommand	OracleCommand	OledbCommand	OdbcCommand
IDbDataReader	SqlDataReader	OracleDataReader	OledbDataReader	OdbcDataReader

Để sử dụng data provider nào, chúng ta phải tiến hành khái báo using namspace tương ứng. Chẳng hạn, using System.Data.SqlClient;

Ngoài những data provider mô tả ở bảng trên, chúng ta có thể reference đến các data provider khác không được tích hợp sẵn bởi ADO.NET trong Visual Studio .NET, chẳng hạn như data provider dùng cho MySQL, Postgre, …

Thành phần thứ hai trong kiến trúc ADO.NET – DataSet – được xem như là container dùng để lưu trữ đối tượng liên quan đến dữ liệu như DataTable, DataRelation, DataView. Thành phần này còn được gọi là lớp không kết nối (disconected layer).

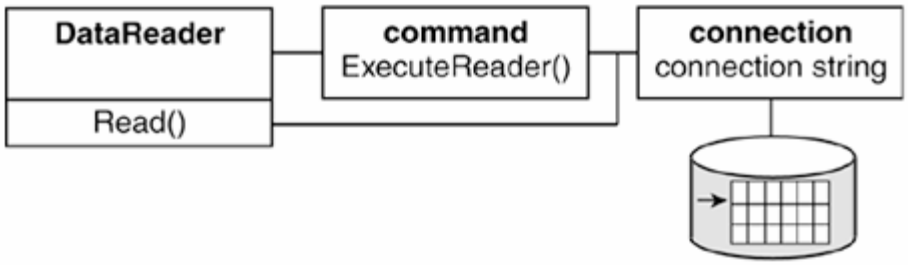
[3] Nhà cung cấp ở đây được hiểu theo nghĩa cả về loại nguồn dữ liệu lẫn cách thức truy xuất nguồn dữ liệu. Ví dụ, ngoài data provider SqlClient do Microsoft cung cấp, cũng có thể có một tổ chức khác phát triển một provider khác để truy xuất loại nguồn dữ liệu này.

DataSet như là một CSDL thu nhỏ tại máy client, có thể chứa các đối tượng table, view, constaint, ralationship giữa các table, … Tất cả dữ liệu từ nguồn dữ liệu thực sẽ được nạp vào DataSet dưới dạng các DataTable, đó là một snapshot của nguồn dữ liệu thực. Khối dữ liệu này sẽ được chỉnh sửa độc lập, sau đó nếu cần sẽ được cập nhật trở lại nguồn dữ liệu thực. Theo nguyên tắc này, chúng ta không cần duy trì kết nối liên tục một cách không cần thiết với nguồn dữ liệu thực trong suốt quá trình thao tác với nó.

4.2 Tổng quan về các mô hình xử lý dữ liệu trong ADO.NET

4.2.1 Mô hình Kết nối

Trong mô hình kết nối của ADO.NET, có một connection hoạt động được duy trì giữa đối tượng *DataReader* của ứng dụng và một *data source* (nguồn dữ liệu). Một dòng dữ liệu (data row) được trả về từ data source mỗi khi phương thức Read của đối tượng DataReader được thực thi. Điểm quan trọng nhất của mô hình kết nối đó là dữ liệu được lấy từ tập dữ liệu (các record được trả về bởi một lệnh SQL nào đó) theo kiểu từng record một cho một lần đọc, chỉ đọc (read-only), và chỉ theo một hướng tiến (forward-only). Hình dưới đây mô tả cách sử dụng DataReader trong chế độ *kết nối*.



Các bước điển hình để làm việc với đối tượng DataReader là như sau:

1. Tạo đối tượng Connection bằng cách truyền một chuỗi Connection string cho hàm khởi dựng của nó.
2. Khởi tạo một biến chuỗi và gán cho câu lệnh SQL dựa theo dữ liệu muốn nạp về.
3. Khởi tạo một đối tượng Command từ với nội dung câu lệnh SQL đã xác định ở trên.
4. Tạo đối tượng DataReader bằng cách thực thi phương thức Command.ExecuteReader(). Đối tượng này sau đó sẽ được dùng để đọc kết quả của câu truy vấn mỗi dòng một lần.

Đoạn code sau minh họa các bước trên với Data Provider SqlClient. Đoạn code sẽ đọc

danh sách họ tên các sinh viên trong một bảng SinhVien của cơ sở dữ liệu và hiển thị lên một điều khiển ListBox. Chi tiết về các đối tượng DataReader, Command, Connection sẽ được đề cập chi tiết sau.

```
using System.Data.SqlClient;
...
// (1) Tao Connection
SqlConnection cn = new SqlConnection(chuoiKetNoi);
cn.Open();
// (2) Chuoi SQL thuc hien lay danh sach ten cac sinh vien xep tang dan theo NgaySinh
string sql = "SELECT HoTen FROM SinhVien ORDER BY NgaySinh";
// (3) Tao doi tuong Command
SqlCommand cmd = new SqlCommand(sql, conn);
DbDataReader rdr;
// (4) Tao doi tuong DataReader
rdr = cmd.ExecuteReader(CommandBehavior.CloseConnection);
while (rdr.Read())

   listBox1.Items.Add(rdr["HoTen"]);  // Fill ListBox
rdr.Close();  // Dong datareader sau khi da su dung xong
```

Tham số được sử dụng trong phương thức ExecuteReader xác định đối tượng Connection sẽ được đóng sau khi DataReader được đóng.

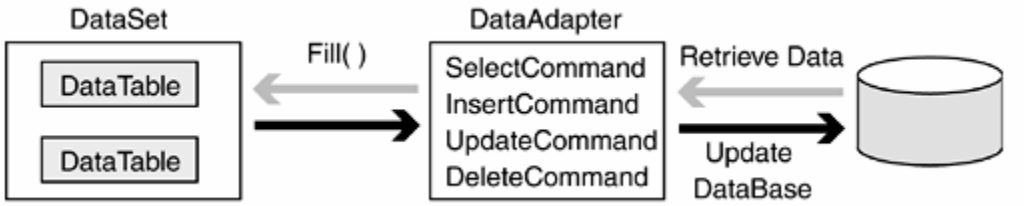
4.2.2 Mô hình Ngắt Kết nối

Triết lý của mô hình Ngắt kết nối đó là: Dữ liệu được nạp – sử dụng một lệnh SQL – từ nguồn dữ liệu bên ngoài vào bộ nhớ đệm tại máy client; tập kết quả được xử lý tại máy cục bộ; mọi cập nhật sau đó sẽ được truyền từ dữ liệu trong bộ nhớ ngược trở lại nguồn dữ liệu.

Mô hình được gọi là "ngắt kết nối" bởi vì đối tượng kết nối chỉ được mở đủ lâu để đọc dữ liệu từ nguồn dữ liệu và tiến hành các thao tác cập nhật. Bằng cách đưa dữ liệu về phía máy client, tài nguyên của server – chẳng hạn như thông tin dữ liệu Connection, bộ nhớ, thời gian xử lý – sẽ được giải phóng bớt. Tuy vậy, mô hình này cũng có nhược điểm về thời gian cần để nạp tập dữ liệu và bộ nhớ dùng để chứa dữ liệu tại máy client.

Như hình dưới đây minh họa, các thành phần chính của mô hình ngắt kết nối đó là DataApdapter và DataSet. DataAdapter làm nhiệm vụ như là cầu nối giữa nguồn dữ liệu và DataSet, nạp dữ liệu vào các bảng của DataSet và đẩy các thay đổi ngược trở lại nguồn

dữ liệu. Một DataSet đóng vai trò như là một cơ sở dữ liệu quan hệ nằm trong bộ nhớ, chứa một hay nhiều DataTables, giữa các DataTable này cũng có thể có các mối quan hệ với nhau như trong một cơ sở dữ liệu quan hệ thực. Một DataTable chứa các dòng và các cột dữ liệu thường được lấy từ cơ sở dữ liệu nguồn.



Trong số các phương thức và thuộc tính của DataAdapter thì Fill() và Update() là hai phương thức quan trọng nhất. Fill() chuyển một query đến cơ sở dữ liệu và lưu tập kết quả trả về trong một DataTable nào đó; phương thức Update() thực hiện một thao tác thêm, xóa, cập nhật dựa trên những thay đổi của đối tượng DataSet. Các lệnh cập nhật thực sự được chứa trong các thuộc tính của DataAdapter. Chi tiết về DataAdapter sẽ được đề cập ở phần sau.

Để minh họa cách thức làm việc với DataAdapter và DataSet, đoạn code dưới đây giới thiệu cách tạo ra một đối tượng DataTable, nạp dữ liệu từ một cơ sở dữ liệu, và đưa nó vào một DataSet.

```
string sql = "SELECT MaSinhVien, HoTen, NgaySinh FROM SinhVien";
string connStr = "Data Source=MYSERVER;Initial Catalog=qlsinhvien;
            User Id=k28;Password=k28;";
// (1) Tao doi tuong data adapter
SqlDataAdapter da = new SqlDataAdapter(sql, connStr);
// (2) Tao doi tuong dataset
DataSet ds = new DataSet();
// (3) Tao mot Table co ten “SinhVien” trong dataset va nap du lieu cho no
da.Fill(ds, "SinhVien");
// (4) Hien thi danh sach ten sinh vien ra list box
DataTable dt = ds.Tables["SinhVien"];
for (int i=0; i< dt.Rows.Count;i++)
{
  DataRow row = dt.Rows[i];
  listBox1.Items.Add(row["HoTen"]);
}
```

Bước đầu tiên là tạo ra một thể hiện của SqlDataAdapter bằng cách truyền một câu lệnh SELECT và chuỗi kết nối cho phương thức khởi dựng của lớp này. DataAdapter sẽ lo đến việc tạo ra đối tượng Connection cũng như việc mở, đóng Connection khi cần thiết. Sau khi một DataSet rỗng sẽ được tạo ra, phương thức Fill() của DataAdapter sẽ tạo ra một DataTable có tên là "SinhVien" trong DataSet và nạp các dòng dữ liệu vào DataTable này (bằng câu lện SQL dạng SELECT của DataAdapter). Mỗi column của DataTable sẽ tương ứng với một column trong bảng của cơ sở dữ liệu nguồn. Dữ liệu trong bảng dữ liệu sau đó được đưa vào một ListBox bằng cách duyệt qua danh sách các dòng của DataTable.

4.3 Làm việc với mô hình Kết nối trong ADO.NET

Như đã mô tả tổng quan trong phần trước, mô hình Kết nối được dựa trên việc thiết lập một Connection đến CSDL và sau đó sử dụng các Command để thực hiện việc thêm, xóa, sửa, hay đọc dữ liệu từ data source (nguồn dữ liệu) được kết nối. Đặc điểm phân biệt của mô hình này đó là các Command được phát sinh, làm việc với data source thông qua một Connection đang hoạt động – Connection này sẽ mở cho đến khi các thao tác được hoàn tất. Cho dù là làm việc với mô hình Kết nối hay Ngắt kết nối, bước đầu tiên trong quá trình truy xuất một data source đó là tạo ra một đối tượng Connection để làm đường truyền giữa ứng dụng với data source.

4.3.1 Lớp Connection

Có nhiều lớp Connection trong ADO.NET – mỗi lớp tương ứng với một Data Provider – bao gồm SqlConnection, OracleConnection, OleDbConnection, OdbcConnection. Mặc dù mỗi lớp có thể gồm những đặc tính riêng, nhưng các lớp này đều phải implement interface IDbConnection. Bảng dưới đây tóm tắt các thành phần được định nghĩa bởi interface này.

Loại	Tên	Mô tả
Property	ConnectionString	Get/Sets chuỗi kết nối đến data source.
Property	ConnectionTimeout	Khoảng thời gian tối đa tính bằng giây để chờ thực hiện việc kết nối đến data source
Property	Database	Tên CSDL ứng với Connection hiện tại
Property	State	Trạng thái hiện tại của Connection. Trả về một giá trị kiểu liệt kê (enumeration): Broken, Closed, Connecting, Executing, Fetching, hoặc Open

Loại	Tên	Mô tả
Method	Open Close	Mở một Connection. Roll back mọi thao tác đang làm dở. Đóng Connection – trả Connection cho Connection Pool nếu như có sử dụng Connection Pool
Method	BeginTransaction	Khởi tạo một database transaction
Method	ChangeDatabase	Thay đổi CSDL hiện tại cho Connection đang mở. Chuỗi mô tả tên CSDL mới được truyền cho phương thức này
Method	CreateCommand	Tạo ra một đối tượng Command ứng với Connection

4.3.1.1 Connection string

Thuộc tính ConnectionString xác định data source và các thông tin cần thiết để truy xuất data source, chẳng hạn như User ID và Password, … Ngoài những thông tin cơ bản này, Connection string còn có thể chứa các giá trị cho các trường dữ liệu đặc trưng cho data provider. Ví dụ, Connection string cho Ms Sql Server có thể chứa các giá trị để quy định Connection Timeout và Packet Size.

Dưới đây là các ví dụ về cách thành lập chuỗi kết nối cho các data provider thường gặp. Danh sách đầy đủ về cách thành lập các chuỗi kết nối được cho ở **Error! Reference source not found.**.

- SqlConnection sử dụng cơ chế xác thực kiểu SQL Server:

  “server=(1);database=(2);uid=(3);pwd=(4)” hoặc

  “Data Source=(1);Initial Catalog=(2);User ID=(3);Password=(4)”

- SqlConnection sử dụng cơ chế xác thực kiểu Windows:

  “Server=(1);Database=(2);Trusted_Connection=yes”

  Ở đây, (1) là tên/máy chủ chứa CSDL, (2) là tên CSDL, (3) là tên đăng nhập, (4) là mật khẩu tương ứng.

  Ví dụ:

  ```
  “server=192.168.0.1;database=qlnhanvien;uid=k28;pwd=spider
  ”
  ```

hoặc
`“Server=192.168.0.1;Database=qlnhanvien;Trusted_Connection=yes”`

- OleDbConnection sử dụng để kết nối CSDL Access phiên bản trước 2003:

  `“Provider=Microsoft.Jet.OLEDB.4.0;DataSource=①”`

  Ví dụ:

  - `string stConnection = string.Format(“Provider=Microsoft.Jet.OLEDB.4.0;DataSource={0}”, @”c:\program files\qlnhanvien.mdb”);`
  - Sử dụng trong ứng dụng Internet: `string stConnection = string.Format(“Provider=Microsoft.Jet.OLEDB.4.0;DataSource={0}”, Server.MapPath(“/data/qlnhanvien.mdb”);`

- ODBC: `“DSN=①”` với ① là Data Source Name (DSN), ví dụ `“DSN=qlnhanvien”`

Các Connection string được dùng để tạo ra đối tượng Connection. Cách thực hiện thông thường là truyền chuỗi này cho hàm khởi dựng như ví dụ dưới đây:

```
string stConnection =
        "Data Source=192.168.0.1;” +
    “Initial Catalog=films;” +
    “User Id=k28;”+
    “Password=spider";
SqlConnection cn = new SqlConnection(stConnection);
cn.Open(); //Open connection
```

4.3.1.2 Connection Pooling

Tạo một Connection là một quá trình tốn nhiều thời gian – trong một số trường hợp, việc này thậm chí còn tốn thời gian hơn việc thực thi các Command. Để loại bỏ điều này, ADO.NET cung cấp một khái niệm gọi là connection pool. Connection pool quản lý các Connection có trùng Connection string để tối ưu hóa số lần thiết lập, hợp lệ hóa thông tin kết nối.

Các quy tắc quy định connection pool cần biết:

- Cơ chế Connection pooling được kích hoạt theo mặc định. Cơ chế này được tắt bằng cách thêm vào Connection string `"Pooling=false"` đối với SqlConnection hoặc `"OLE DB Services=-4"` đối với OleDbConnection.

- Mỗi connection pool được ứng với một connection string duy nhất. Khi có một Connection được yêu cầu, pool handler (bộ quản lý pool) sẽ so sánh connection string với những connection string trong các pools đang tồn tại. Nếu có một Connection trùng khớp thì Connection tương ứng sẽ được xác định trong pool.

- Nếu tất cả các connection trong pool đang được sử dụng khi lại có yêu cầu về connection thì yêu cầu đó sẽ được xếp vào hàng đợi cho đến khi có một connection rảnh. Các connection sẽ được giải phóng khi phương thức Close hay Dispose của đối tượng connection được gọi.

- Connection pool được đóng khi tất cả các connection trong nó được giải phóng và hết thời gian (time out).

Đối với SQL Server, bạn có thể điều khiển hành vi của conneciton pooling bằng cách gộp các cặp key-value vào connection string. Các key này có thể được sử dụng để thiết lập số lượng nhỏ nhất và lớn nhất các connection trong pool cũng như xác định xem một connection có cần phải reset khi nó được lấy từ pool ra hay không. Một key đặc biệt chú ý đó là key có tên Lifetime, xác định thời gian mà connection có thể tồn tại trước khi nó bị hủy bỏ. Giá trị này được kiểm tra khi một connection được trả về cho pool. Nếu connection đã được mở trước đó, và lâu hơn giá trị Lifetime thì nó sẽ bị hủy.

Đoạn code dưới đây minh họa các dùng các key này cho SqlClient:

```
string stConnection =
    "Server=192.168.0.1;” +
    “Trusted_Connection=yes;” +
    “database=qlnhanvien;" +
    "connection reset=false;" +
    "connection Lifetime=60;" +  // Seconds
    "min pool size=1;" +
    "max pool size=50"; // Default=100
SqlConnection cn = new SqlConnection(cnString);
```

4.3.2 Đối tượng Command

Sau khi một đối tượng connection được tạo ra, bước tiếp theo trong quá trình truy xuất CSDL – đối với mô hình Kết nối – đó là tạo ra một đối tượng Command để gửi một query (select) hay một action command (thêm, xóa, sửa) đến data source. Có nhiều loại lớp Command ứng với các data provider; các lớp này đều implement interface IDbCommand.

4.3.2.1 Tạo đối tượng Command

Bạn có thể dùng một trong nhiều hàm khởi dựng để tạo đối tượng Command một cách trực tiếp hoặc sử dụng cách tiếp cận ProviderFactory.

Đoạn code dưới đây minh họa các tạo ra một đối tượng Command và thiết lập các thuộc tính của nó.

```
SqlConnection conn = new SqlConnection(connstr);
conn.open();
string sql =
"INSERT INTO SinhVien (MaSinhVien, HoTen) VALUES (@pMaSinhVien, @pHoTen)";
SqlCommand cmd = new SqlCommand();
cmd.Connection = conn;
cmd.commandText = sql;
cmd.Parameters.AddWithValue ("@pMaSinhVien", 12);
cmd.Parameters.AddWithValue ("@pHoTen", "tnv spider");
```

Trong trường hợp ứng dụng có thể phải sử dụng nhiều data provider, bạn nên sử dụng cách tiếp cận provider factory. Factory được tạo ra bằng cách truyền chuỗi data provider cho hàm khởi dựng của nó. Tiếp đến, phương thức CreateCommand được gọi để trả về một đối tượng command.

```
string provider = "System.Data.SqlClient";
DBProviderFactory factory = DbProviderFactories.GetFactory(provider);
DbCommand cmd   = factory.CreateCommand(); // DbCommand là một lớp trừu tượng
cmd.CommandText = sql;     // sql là một chuỗi query hay command
cmd.Connection  = conn;    // conn là một Connection
```

4.3.2.2 Thực thi một Command

Lệnh SQL được gán trong thuộc tính CommandText của đối tượng Command sẽ được thực thi bằng một trong các phương thức được chỉ ra ở bảng dưới đây

Phương thức	**Mô tả**

ExecuteNonQuery	Thực thi truy vấn hành động (action query) và trả về số lượng dòng dữ liệu bị ảnh hưởng bởi truy vấn đó: cmd.CommandText = "DELETE SinhVien WHERE MaSinhVien=12"; int soLuong = cmd.ExecuteNonQuery();
ExecuteReader	Thực thi một query và trả về đối tượng DataReader để có thể truy cập tập kết quả của query đó. Phương thức này nhận một tham số tùy chọn kiểu CommandBehavior để có thể tăng hiệu năng thực thi query. cmd.CommandText = "SELECT * FROM SinhVien" + "WHERE YEAR(NgaySinh) > 1981"; SqlDataReader rdr= cmd.ExecuteReader();
ExecuteScalar	Thực thi một query và trả về giá trị của cột đầu tiên trong dòng đầu tiên của tập kết quả. cmd.CommandText="SELECT COUNT(MaSinhVien) FROM SinhVien"; int soSinhVien = (int)cmd.ExecuteScalar();
ExecuteXmlReader	Chỉ có cho data provider SQL Server. Trả về một đối tượng XmlReader dùng để truy xuất tập dữ liệu. Tham khảo thông tin về XmlReader trong MSDN

ExecuteReader là phương thức quan trọng nhất trong các phương thức kể trên. Phương thức này trả về một đối tượng DataReader giúp truy xuất đến các dòng dữ liệu trả về bởi query. Xem ví dụ dưới đây:

```
dr = cmd.ExecuteReader(sql, ①);
```

Ở đây, ① là một giá trị kiểu CommandBehavior để chỉ định behavior (hành vi) thực thi của query. Một số data providers sử dụng ① để tối ưu quá trình thực thi query. Danh sách

các giá trị và tác dụng của tham số ① được mô tả chi tiết như dưới đây:

- SingleRow. Chỉ định rằng query chỉ trả về 1 dòng dữ liệu. Behavior mặc định là trả về nhiều tập kết quả.
- SingleResult. Query trả về một giá trị tuyến tính đơn nhất (single scalar value).
- KeyInfo. Trả về thông tin về column và primary key. Behavior này được sử dụng với phương thức GetSchema của DataReader để lấy thông tin về các column trong lược đồ (schema).
- SchemaOnly. Dùng để lấy về tên của các cột trong tập dữ liệu trả về:

  Ví dụ

```
dr = cmd.ExecuteReader(CommandBehavior.SchemaOnly);
string col1 = dr.GetName(0);  // tên cột đầu tiên
```

- SequentialAccess. Cho phép dữ liệu trong dòng trả về có thể được truy xuất tuần tự theo column. Behavior này được dùng cho các trường dữ liệu BLOB hay TEXT.
- CloseConnection. Đóng connection khi DataReader được đóng.

3.3.2.3 Thực thi Stored Procedure (thủ tục lưu trữ sẵn) với đối tượng Command

Một stored procedure là một đoạn code SQL được lưu sẵn trong CSDL và có thể được thực thi như là một script. ADO.NET hỗ trợ việc thực thi các stored procedure cho các data provider OleDb , SqlClient, ODBC, và OracleClient.

Các bước để thực thi một stored procedure:

- Thiết lập thuộc tính SqlCommand.CommandText thành tên của procedure;
- Thiết lập thuộc tính CommandType là CommandType.StoredProcedure;
- Thiết lập các Parameter (nếu có) cho procedure
- Thực thi phương thức ExecuteNonQuery.

Thủ tục dưới đây cho phép các mẫu tin lấy về từ bảng SinhVien được phân thành từng nhóm các trang, mỗi trang 10 record. Đầu vào của của procedure là @pTrang (số hiệu trang cần lấy); đầu ra của procedure là số trang tổng cộng của tập dữ liệu. Đoạn code minh họa phía dưới thực hiện việc thiết lập để lấy về trang dữ liệu đầu tiên.

```
SqlCommand cmd  = new SqlCommand();
cmd.CommandText = "spListSinhVien";  // tên procedure
```

```
cmd.CommandType = CommandType.StoredProcedure;
cmd.Parameters.Add(“@pTrang", SqlDbType.Int);
cmd.Parameters.Add(“@pTongSoTrang", SqlDbType.Int);
cmd.Parameters[0].Direction= ParameterDirection.Input;
cmd.Parameters[0].Value= 1;        // thiết lập để lấy về trang đầu tiên
cmd.Parameters[1].Direction=ParameterDirection.Output;
cmd.CommandTimeout=10;  // Cho command tối đa 10s để thực thi
SqlDataReader dr = cmd.ExecuteReader();
while (dr.Read())
{
  // xử lý tập dữ liệu ở đây
}
dr.Close();  // DataReader phải được đóng trước khi đọc tham số đầu ra
int tongSoTrang =  cmd.Parameters[1].Value;
```

Ví dụ này sử dụng data provider SqlClient. Có thể chỉnh sửa một phần nhỏ thì nó cũng có thể hoạt động với OleDb. Điểm khác biệt mấu chốt giữa SqlClient và OleDb đó là cách quản lý các parameter. SqlClient yêu cầu tên parameter phải đúng với tên parameter của stored procedure; trong khi đó OleDb lại truyền các parameter cho stored procedure dựa vào vị trí, vì vậy tên parameter là không quan trọng. Nếu procedure trả về giá trị kết quả, OleDb phải thiết kế để parameter đầu tiên trong danh sách làm nhiệm vụ này. Với SqlClient, chúng ta chỉ cần thêm một parameter với một tên nào đó và xác định hướng trả về (direction) của parameter này là Return Value.

Phần code của stored procedure là như sau:

```
CREATE PROCEDURE spListSinhVien
  @pTrang int,
  @pTongSoTrang int output
AS
  /*
   Thủ tục trả về một tập kết quả gồm các SinhVien xếp theo HoTen.
   Tập kết quả được phân thành các trang, mỗi trang 10 SinhVien.
  */
  SET NOCOUNT ON
  SELECT @pSoTrang = CEILING(COUNT(*)/10) FROM SinhVien
  if @pTrang = 1 or @pTrang <1
  begin
    SELECT TOP MaSinhVien, HoTen FROM SinhVien ORDER BY HoTen
    set @pTrang = 1
```

```
    return 0
  end
  if @pTrang > @pTongSoTrang
  begin
    SET @pTrang = @pTongSoTrang
    declare @RowCount int
    set @RowCount = (@pTrang * 10)

    exec (
      'SELECT *
      FROM (
            SELECT TOP 10 a.*
          FROM (
                SELECT TOP ' + @RowCount + ' *
              FROM SinhVien
              ORDER BY HoTen
          ) a
          ORDER BY HoTen desc
      ) b
      ORDER BY HoTen'
   )

     return 0
  end
```

4.3.2.4 Sử dụng Parameter trong các Command không phải là Stored Procedures

Trong các query được thành lập trực tiếp (chứ không phải là stored procedure như ở trên), chúng ta cũng có thể sử dụng các Parameter. Ví dụ dưới đây minh họa cách thức bổ sung một record vào bảng SinhVien:

```
string sql =
"INSERT INTO SinhVien (MaSinhVien, HoTen) VALUES (@pMaSinhVien, @pHoTen)";
SqlCommand cmd = new SqlCommand();
cmd.Connection = conn;
cmd.commandText = sql;
```

```
cmd.Parameters.AddWithValue("@pMaSinhVien", 12);
cmd.Parameters.AddWithValue("@pHoTen", "tnv spider");
```

Một cách khác để thực hiện việc bổ sung record như trên là sử dụng phép nối chuỗi[4] như thế này:

```
int iMaSinhVien = 12;
string stHoTen = "tnv spider";
sql = string.Format("INSERT INTO SinhVien (MaSinhVien, HoTen) VALUES ({0},
'{1}')",
   iMaSinhVien, stHoTen);
SqlCommand cmd = new SqlCommand(sql, conn);
```

4.3.3 Đối tượng DataReader

Như đã thấy trong các ví dụ trước, một DataReader cho phép lấy các dòng và cột dữ liệu của dữ liệu trả về khi thực thi một query. Việc truy xuất dòng được định nghĩa bởi interface IDataRecord. Dưới đây là các member quan trọng của interface này.

4.3.3.1 Truy xuất các dòng dữ liệu với DataReader

DataReader lấy về từng dòng đơn (single row) từ một tập dữ liệu trả về mỗi khi phương thức Read của nó được thực thi. Nếu không có dòng dữ liệu nào thì phương thức này trả về giá trị false. DataReader phải được đóng sau khi các thao tác xử lý các dòng được hoàn tất để giải phóng tài nguyên hệ thống. Bạn có thể sử dụng thuộc tính DataReader.IsClosed để biết được DataReader đã được đóng hay chưa.

Mặc dù DataReader là ứng với một Command đơn, nhưng Command này lại có thể chứa nhiều query trong đó, do đó có thể trả về nhiều tập dữ liệu kết quả. Đoạn code dưới đây minh họa cách xử lý các dòng dữ liệu trả về bởi 2 query trong một Command.

```
string q1 = "SELECT * FROM SinhVien WHERE YEAR(NgaySinh) < 1981";
string q2 = "SELECT * FROM SinhVien WHERE YEAR(NgaySinh) > 1990";
```

[4] Trong thực tế, giải pháp nối chuỗi ít khi được sử dụng vì lý do an toàn dữ liệu. Hãy hình dung trong đoạn code này, nếu `stHoTen` được gán giá trị là `"tnv spider'); DELETE * FROM SinhVien;--"`. Khi đó query được thực thi sẽ là `"INSERT INTO SinhVien (MaSinhVien, HoTen) VALUES (12, 'tnv spider'); DELETE * FROM SinhVien;--)"`
Lỗ hổng kiểu này thường được gọi với tên SQL Injection.

```
cmd.CommandText = q1 + ";" + q2; // hai query được ngăn cách nhau bởi dấu ;
DbDataReader rdr = cmd.ExecuteReader();
bool readNext = true;
while (readNext)
{
  while (rdr.Read())
    MessageBox.Show(rdr.GetString(1));
  readNext = rdr.NextResult(); // kiem tra xem con tap du lieu nao nua khong
}
rdr.Close();
conn.Close();
```

DataReader không có thuộc tính hay phương thức nào cho biết số lượng dòng dữ liệu trả về trong tập dữ liệu của nó (do đặc tính forward-only của DataReader), tuy nhiên, chúng ta có thể sử dụng thuộc tính HasRows (kiểu Boolean) của DataReader để xác định xem nó có 1 hay nhiều dòng để đọc hay không.

4.3.3.2 Truy xuất giá trị của column

Có nhiều cách để truy xuất dữ liệu chứa trong các columns của dòng hiện tại của DataReader:

- Truy xuất như là một array dùng số thứ tự column (bắt đầu từ 0) hoặc dùng tên column
- Sử dụng phương thức GetValue bằng cách truyền cho phương thức này số thứ tự của column
- Sử dụng một trong các phương thức định kiểu GetXXX, bao gồm GetString, GetInt32, GetDateTime, GetDouble, …

Đoạn code dưới đây minh họa các thức truy xuất giá trị dữ liệu của các column.

```
cmd.CommandText =
   "SELECT MaSinhVien, Hoten, GioiTinh, NgaySinh FROM SinhVien " +
   "WHERE YEAR(NgaySinh) = 1981";
dr = cmd.ExecuteReader();
dr.Read();
// Các cách để lấy dữ liệu kiểu string ở cột thứ 2 (HoTen)
string stHoTen;
stHoTen = dr.GetString(1);
stHoTen = (string)dr.GetSqlString(1); // SqlClient provider
stHoTen = (string)dr.GetValue(1);
```

```
stHoTen = (string)dr["HoTen"];
stHoTen = (string)dr[1];
// Lấy dữ liệu kiểu DateTime ở cột thứ 4 (NgaySinh) có kiểm tra giá trị NULL
if (!dr.IsDbNull(3))
    DateTime dtNgaySinh = dr.GetDateTime(3);
```

Phương thức GetString có điểm thuận lợi trong việc ánh xạ nội dung dữ liệu từ CSDL sang kiểu dữ liệu của .NET. Các cách tiếp cận khác đều trả về các kiểu đối tượng có yêu cầu phép chuyển kiểu. Vì lý do này, bạn nên sử dụng các phương thức GetXXX cho các kiểu dữ liệu xác định. Cũng lưu ý rằng, phương thức GetString không yêu cầu phép chuyển kiểu, nhưng bản thân nó không thực hiện bất cứ phép chuyển đổi nào; chính vì thế, nếu dữ liệu là không đúng như kiểu dữ liệu trông đợi sẽ có Exception được trả ra.

Nhiều ứng dụng phụ thuộc vào tầng xử lý dữ liệu để cung cấp DataReader. Với những trường hợp như thế, ứng dụng có thể sử dụng metadata (siêu dữ liệu) để xác định tên column, kiểu dữ liệu của column, và các thông tin khác về column. Đoạn code sau đây minh họa việc in ra danh sách các tên và kiểu dữ liệu của các column mà đối tượng DataReader đang quản lý:

```
// In ra danh sách các tên column của một đối tượng DataReader có tên dr
for (int i = 0; i < dr.FieldCount; i++)
  Console.WriteLine("Column {0} co kieu du lieu {1}",
        dr.GetName(i), dr.GetDataTypeName(i));  // Column name
```

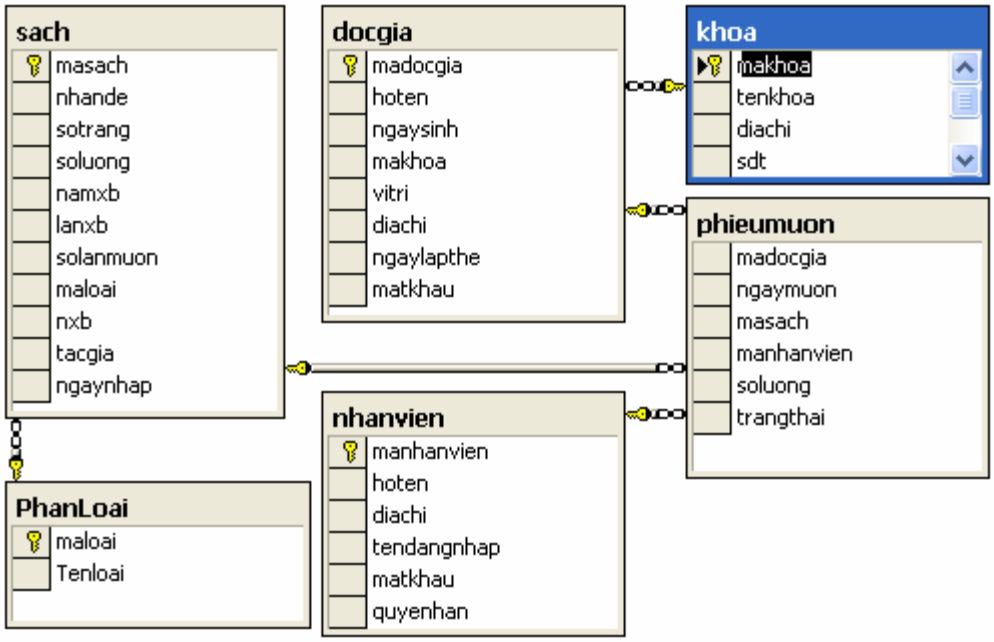
Có một cách khác toàn diện hơn để quản lý toàn bộ thông tin về lược đồ (schema) của tập dữ liệu kết quả trả về, đó là sử dụng phương thức GetSchemaTable. Phương thức này trả về một đối tượng DataTable mà mỗi dòng trong DataTable này sẽ biểu diễn một column trong tập dữ liệu kết quả. Đoạn code dưới đây minh họa cách truy xuất tất cả các thông tin về các column của một tập dữ liệu trả về.

```
DataTable schemaTable = dr.GetSchemaTable();
int stt = 0;
foreach (DataRow r in schemaTable.Rows)
{
  foreach (DataColumn c in schemaTable.Columns)
  {
    Console.WriteLine(stt.ToString() + " " + c.ColumnName + ": " + r[c]);
    stt++;
  }
```

```
}
Kết quả hiển thị:
      0 ColumnName: movie_ID
      1 ColumnOrdinal: 0
      … //không liệt kê
      12 DataType: System.Int32
      … //không liệt kê
```

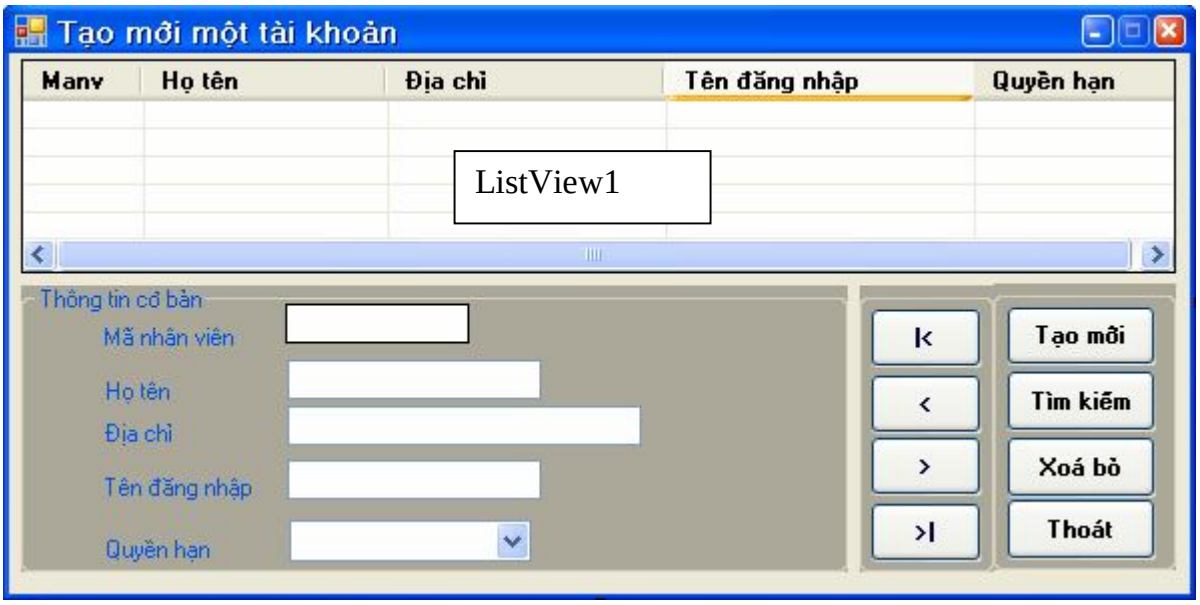
4.3.4 Bài thực hành

Giả sử ta đã có cơ sở dữ liệu quanlythuvien trong SQL Server có quan hệ như sau:



Bài thực hành về đối tượng Connection, Command và DataReader

Thiết kế Form để tạo mới 1 tài khoản như sau (làm việc trên bảng nhanvien):



Frmtaomoitk sử dụng các trường, phương thức và sự kiện sau:

```
namespace baimau
{   public partial class Frmtaomoitk : Form
    {
        SqlConnection cn;
        SqlCommand cmdSelect;
        SqlCommand cmdInsert;
        SqlCommand cmdXoa;
        int i = 0;
        public Frmtaomoitk()...
        private void Moketnoi()...
        private void LoadListView()...
        private void LoadItem(int i)...
        private void LoadCombox()...
        private void XoaTextBox()...
        private int KiemTraMa(string ma)...
        private void Frmtaomoitk_Load(object sender, EventArgs e)...
        private void butdau_Click(object sender, EventArgs e)...
        private void buttien_Click(object sender, EventArgs e)...
        private void butlui_Click(object sender, EventArgs e)...
        private void butTaomoi_Click(object sender, EventArgs e)...
        private void butXoabo_Click(object sender, EventArgs e)...
        private void buttimkiem_Click(object sender, EventArgs e)...
    }
}
```

Các điều khiển

Tên điều khiển	Thuộc tính
Form	**Name**: Frmtaomoitk **Text**:Tạo mới một tài khoản sử dụng chương trình
listView	**Name**:listView1 **Columns**: Add thêm 4 cột: Họ tên, Địa chỉ, Tên đăng nhập và Quyền hạn View: Details GridLines:True
groupBox	**Name:**groupBox1 Text: Thông tin cơ bản
Label	Tạo ra 5 label để hiển thị: Mã nhân viên, Họ tên, Địa chỉ, Tên đăng nhập và quyền hạn.
TextBox	Tạo ra 4 TextBox lần lượt với các tên: txtmanv, txthoten, txtdiachi, txttendangnhap
Button	Tạo 8 button lần lượt với các tên butdau, butlui, buttien, butcuoi, buttaomoi, buttimkiem, butxoabo,butthoat

Các trường:

Tên trường	Ý nghĩa
Cn	Dùng để kết nối đến cơ sở dữ liệu quanlythuvien
cmdSelect	sqlCommand sử dụng câu lệnh select để hiển thị và tìm kiếm
cmdInsert	sqlCommand sử dụng câu lệnh Insert để tạo thêm 1 tài khoản
cmdXoa	sqlCommand sử dụng câu lệnh Delete để xóa 1 tài khoản
I	Tài khoản thứ i

Các phương thức

+ Hàm dựng **Frmtaomoitk** để tạo giao diện

```
public Frmtaomoitk()
      {     InitializeComponent(); }
```

+ Phương thức Moketnoi(): Kiểm tra đường kết nối, nếu đang mở thì đóng lại, sau đó mở lại đường kết nối

```
private void Moketnoi()
      {
         if (cn.State == ConnectionState.Open)
```

```
            cn.Close();
        cn.Open();
    }
```

+ Phương thức **LoadListView**: Lấy dữ liệu của bảng nhanvien nạp dữ liệu lên listView1. Phương thức này được gọi khi thay đổi dữ liệu trong bảng nhận viên như nhập thêm hoặc xóa đi 1 nhân viên. Sử dụng 2 đối tượng SqlCommand, SqlDataReader

```
private void LoadListView()
    {
        Moketnoi();
        cmdSelect = new SqlCommand("select * from nhanvien", cn);
        SqlDataReader r = cmdSelect.ExecuteReader();
        listView1.Items.Clear(); // Xóa tất cả dữ liệu trong listView1
        while (r.Read())
        {
            string[] st = new string[5];
            st[0] = r[0].ToString();
            st[1] = r[1].ToString();
            st[2] = r[2].ToString();// Không hiển thị mật khẩu, nên không có r[3]
            st[3] = r[4].ToString();
            st[4] = r[5].ToString();
            ListViewItem lv = new ListViewItem(st);
            listView1.Items.Add(lv);
        }
        cmdSelect.Dispose();
    }
```

+ Phương thức **LoadItem**: Lấy dữ liệu từ dòng thứ i của listView1 đưa vào txtmanv, txthoten, txtdiachi, txttendangnhap và comboBox1. Phương thức này được gọi khi di chuyển qua lại thông tin của các nhân viên.

```
private void LoadItem(int i)
    {
        txtmanv.Text = listView1.Items[i].Text;
        txthoten.Text = listView1.Items[i].SubItems[1].Text;
        txtdiachi.Text = listView1.Items[i].SubItems[2].Text;
        txttendangnhap.Text = listView1.Items[i].SubItems[3].Text;
        comboBox1.Text = listView1.Items[i].SubItems[4].Text;
    }
```

+ Phương thức **LoadCombox**: Đưa dữ liệu vào cho comboBox1. Giả sử chỉ có 3 quyền hạn: admin, sinhvien và Thuthu. Phương thức này được gọi khi vừa nạp Form lên

```
private void LoadCombox()
      {
         comboBox1.Items.Add("Admin");
         comboBox1.Items.Add("Sinhvien");
         comboBox1.Items.Add("ThuThu");
         comboBox1.Text = "Admin";
 }
```

+ Phương thức **XoaTextBox**: Xóa hết dữ liệu trong các textBox, phương thức này được goi khi nhập thêm 1 tài khoản.

```
 private void XoaTextBox()
      {
         txtmanv.Clear();
         txthoten.Clear();
         txtdiachi.Clear();
         txttendangnhap.Clear();
         txtmanv.Focus();
      }
```

+ Phương thức **KiemTraMa:** Kiểm tra xem có mã nhân viên nào bằng với ma hay không. Phương thức này được gọi khi nhập thêm 1 tài khoản

```
 private int KiemTraMa(string ma)
      {
         Moketnoi();
         cmdSelect = new SqlCommand("select count(*) from nhanvien
                where manhanvien='"+ma.Trim()+"'");
         cmdSelect.Connection = cn;
         return (int)cmdSelect.ExecuteScalar();
      }
```

+ Sự kiện **Frmtaomoitk_Load:** Tạo và mở ra đường kết nối đến cơ sở dữ liệu quanlythuvien, tên máy chủ nhha, sử dụng cơ chế xác thực kiểu Windows, nạo dữ liệu vào cho các điều khiển.

```
private void Frmtaomoitk_Load(object sender, EventArgs e)
      { try
         {
            cn = new SqlConnection("Data Source=nhha;Initial Catalog=quanlythuvien;
                        Trusted_Connection=yes");
            cn.Open();
         }
```

```
        catch (Exception loi) { MessageBox.Show("Không thể kết nối được"); }
        LoadListView(); //Nạp dữ liệu vào cho listView1
        i = 0;
        LoadItem(i);// Nạp dữ liệu vào cho các textBox và comboBox1
        LoadCombox();
    }
```

+ Sự kiện **butdau_Click:** Nạp dữ liệu của dòng đầu tiên từ listView1 vào cho các textBox và comboBox

```
private void butdau_Click(object sender, EventArgs e)
    {
        i = 0;
        LoadItem(i);
    }
```

+ Sự kiện **buttien_Click:** Nạp dữ liệu của dòng tiếp theo từ listView1 vào cho các textBox và comboBox

```
private void buttien_Click(object sender, EventArgs e)
    {
        i++;
        if (i == listView1.Items.Count) i = listView1.Items.Count - 1;
        LoadItem(i);
    }
```

+ Sự kiện **butlui_Click:**

```
private void butlui_Click(object sender, EventArgs e)
    {
        i--;
        if (i < 0) i = 0;
        LoadItem(i);
    }
```

+ Sự kiện **butcuoi_Click**:

```
private void butcuoi_Click(object sender, EventArgs e)
  {
    i = listView1.Items.Count - 1;
    LoadItem(i);
  }
```

+ Sự kiện **butTaomoi_Click:** Được sử dụng để thêm 1 tài khoản (1 nhân viên), nút butTaomoi có 2 trạng thái tạo mới và lưu. Nếu người sử dụng kích vào nút Tạo mới sẽ

chuyển sang trạng thái là lưu và ngược lại.

```
private void butTaomoi_Click(object sender, EventArgs e)
    {
       if (butTaomoi.Text.Equals("Tạo mới"))
       {
          XoaTextBox();
          butTaomoi.Text = "Luu";
       }
       else
      // Kiểm tra xem mã nhân viên này có hay chưa ?
       if (KiemTraMa(txtmanv.Text)==1)          {
          MessageBox.Show("Mã này đã có");
          txtmanv.Clear();
          txtmanv.Focus();
       }
       else
       {
          string ma = txtmanv.Text;
          string hoten = txthoten.Text;
          string diachi = txtdiachi.Text;
          string tendangnhap = txttendangnhap.Text;
          string matkhau = "";// Khi tạo 1 tài khoản thì mật khẩu ban đầu là rỗng
          string quyenhan = comboBox1.Text;
          Moketnoi();
          string sql="insert into nhanvien values("+"'"+ma+"','"+hoten+"','"+diachi
                              +"','" +tendangnhap+"','"+matkhau+"','"+quyenhan +"')";
          cmdInsert = new SqlCommand(sql,cn);
          cmdInsert.ExecuteNonQuery();
          MessageBox.Show("Đã lưu thành công");
          LoadListView(); //Nạp lại dữ liệu mới vào listView1
          butTaomoi.Text = "Tạo mới";
          cmdInsert.Dispose();
       }
    }
```

+ Sự kiện **buttimkiem_Click:** Khi người sử dụng gõ 1 mã nhân viên vào txtmanv và kích vào nút buttimkiem, nếu tìm thấy mã nhân viên này sẽ hiển thị kết quả lên các textBox và comboBox

```
private void buttimkiem_Click(object sender, EventArgs e)
    {
```

```
        Moketnoi();
        string sql = "select * from nhanvien where manhanvien='" + txtmanv.Text + "'";
        cmdSelect = new SqlCommand(sql,cn);
        SqlDataReader dr = cmdSelect.ExecuteReader();
        if (dr.Read())// Đã tìm thấy
        {
           txtmanv.Text = dr[0].ToString();
           txthoten.Text = dr[1].ToString();
           txtdiachi.Text = dr[2].ToString();
           txttendangnhap.Text = dr[4].ToString();
           comboBox1.Text = dr[5].ToString();
        }
        else
           MessageBox.Show("Không tìm thấy");
     }
```

+ Sự kiện **butXoabo_Click:** Xóa nhân viên có mã nhân viên ở txtmanv

```
private void butXoabo_Click(object sender, EventArgs e)
     {
        DialogResult dr = MessageBox.Show("Chắc chắn xóa hay không ?",
                                  "Thông báo", MessageBoxButtons.YesNo);
        if (dr == DialogResult.Yes) // Nếu người sử dụng chọn nút yes
        {
           Moketnoi();
          string Sql = "delete from nhanvien where manhanvien='" + txtmanv.Text + "'";
           cmdXoa = new SqlCommand(Sql,cn);
           if (cmdXoa.ExecuteNonQuery() == 1)
           {
              MessageBox.Show("Xóa thành công");
              LoadListView();
              LoadItem(0);
           }
           else
              MessageBox.Show("Không tồn tại mã nhân viên " + txtmanv.Text);
           cmdXoa.Dispose();
        }
     }
```

4.4 Làm việc với mô hình Ngắt kết nối: DataSet và DataTable

Mô hình Ngắt Kết nối của ADO.NET dựa trên cơ sở sử dụng đối tượng DataSet như là một vùng nhớ đệm. Một đối tượng DataAdapter làm nhiệm vụ trung gian giữa DataSet và

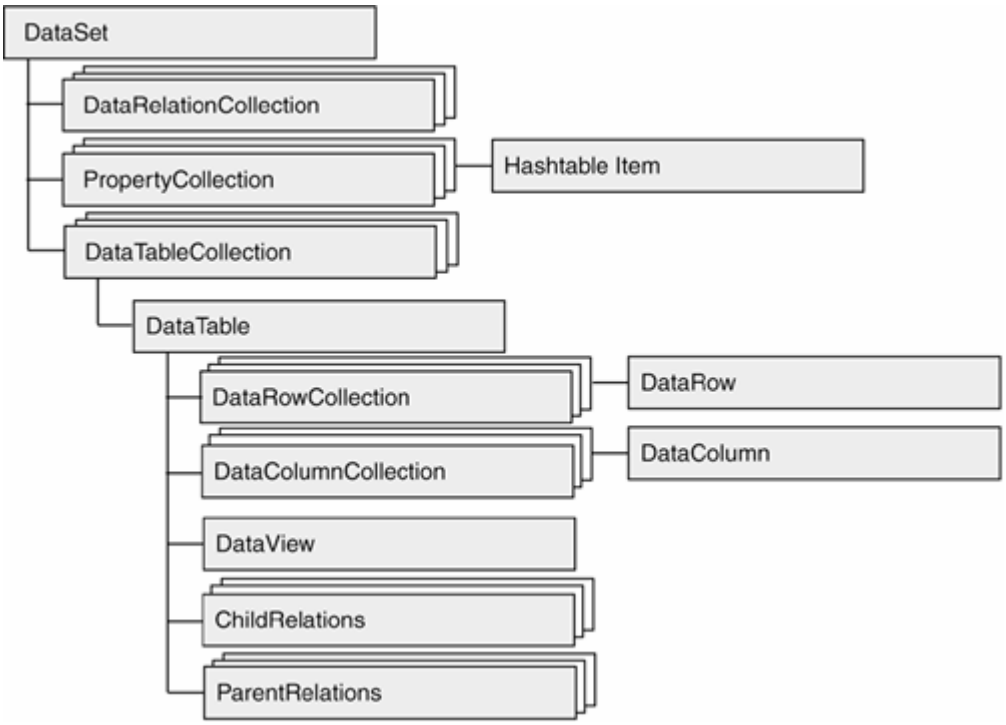
data source (nguồn dữ liệu) để nạp dữ liệu vào vùng nhớ đệm. Sau khi DataAdapter hoàn thành nhiệm vụ nạp dữ liệu, nó sẽ trả đối tượng Connection về pool, vì thế nó ngắt kết nối khỏi nguồn dữ liệu.

4.4.1 Lớp DataSet

DataSet đóng vai trò của một CSDL in-memory (CSDL nằm trong bộ nhớ). Thuộc tính Tables của DataSet là một tập hợp các DataTable chứa dữ liệu và lược đồ dữ liệu (data schema) mô tả dữ liệu trong DataTable. Thuộc tính Relations chứa tập hợp các đối tượng DataRelation xác định cách thức liên kết các đối tượng DataTable của DataSet. Lớp DataSet cũng hỗ trợ việc sao chép, trộn, và xóa DataSet thông qua các phương thức tương ứng là Copy, Merge, và Clear.

DataSet và DataTable là phần lõi của ADO.NET và chúng không là đặc trưng của một data provider nào (giống như ở các lớp Connection, DataReader, DataAdapter). Một ứng dụng có thể định nghĩa và nạp dữ liệu từ nguồn bất kỳ (chứ không nhất thiết là từ một CSDL) vào DataSet.

Bên cạnh các DataTable và các DataRelation, một DataSet còn có thể chứa các thông tin tùy biến khác được định nghĩa bởi ứng dụng. Hình dưới đây mô tả cả lớp chính trong DataSet. Trong số các thuộc tính này, chú ý thuộc tính PropertyCollection; đó là các thuộc tính được lưu trữ dưới dạng một hash table (bảng băm), thường chứa một giá trị time stamp hay các thông tin đặc tả như các yêu cầu hợp lệ hóa (validation requirements) cho column trong các DataTable trong DataSet.



4.4.1.1 DataTable

Thuộc tính DataSet.Tables chứa các đối tượng DataTable. Mỗi đối tượng trong tập hợp này có thể được truy xuất bằng chỉ số hoặc bằng tên.

Các DataTable trong tập hợp DataSet.DataTables mô phỏng các Table trong CSDL quan hệ (các row, column, …). Các thuộc tính quan trọng nhất của lớp DataTable là Columns và Rows định nghĩa cấu trúc và nội dung bảng dữ liệu.

4.4.1.2 DataColumn

Thuộc tính DataTable.Columns chứa một tập các đối tượng DataColumn biểu diễn các trường dữ liệu trong DataTable. Bảng dưới đây tóm tắt các thuộc tính quan trọng của lớp DataColumn.

Phương thức	**Mô tả**
ColumnName	Tên column
DataType	Kiểu của dữ liệu chứa trong column này Ví dụ: col1.DataType = System.Type.GetType("System.String")
MaxLength	Độ dài tối đa của một text column. -1 nếu không xác định độ dài tối đa
ReadOnly	Cho biết giá trị của column có được chỉnh sửa hay không

Phương thức	Mô tả
AllowDBNull	Giá trị Boolean cho biết column này có được chứa giá trị NULL hay không
Unique	Giá trị Boolean cho biết column này có được chứa các giá trị trùng nhau hay không
Expression	Biểu thức định nghĩa cách tính giá trị của một column Ví dụ: colTax.Expression = "colSales * .085";
Caption	Tiêu đề hiển thị trong thành phần điều khiển giao diện đồ họa
DataTable	Tên của đối tượng DataTable chứa column này

Các column của DataTable được tạo ra một cách tự động khi table được nạp dữ liệu từ kết quả của một database query hoặc từ kết quả đọc được ở một file XML. Tuy nhiên, chúng ta cũng có thể viết code để tạo động các column. Đoạn code dưới đây sẽ tạo ra một đối tượng DataTable, sau đó tạo thêm các đối tượng DataColumn, gán giá trị cho các thuộc tính của column, và bổ sung các DataColumn này vào DataTable.

```
DataTable tb = new DataTable("DonHang");
DataColumn dCol = new DataColumn("MaSo", Type.GetType("System.Int16"));
dCol.Unique = true;  // Dữ liệu của các dòng ở column này không được trùng nhau
dCol.AllowDBNull = false;
tb.Columns.Add(dCol);
dCol = new DataColumn("DonGia", Type.GetType("System.Decimal"));
tb.Columns.Add(dCol);
dCol = new DataColumn("SoLuong",Type.GetType("System.Int16"));
tb.Columns.Add(dCol);
dCol= new DataColumn("ThanhTien",Type.GetType("System.Decimal"));
dCol.Expression= "SoLuong*DonGia";
tb.Columns.Add(dCol);
// Liệt kê danh sách các Column trong DataTable
foreach (DataColumn dc in tb.Columns)
{
  Console.WriteLine(dc.ColumnName);
  Console.WriteLine(dc.DataType.ToString());
}
```

Để ý rằng column MaSo được định nghĩa để chứa các giá trị duy nhất. Ràng buộc này giúp cho column này có thể được dùng như là trường khóa để thiết lập relationship kiểu parent-child với một bảng khác trong DataSet. Để mô tả, khóa phải là duy nhất – như

trong trường hợp này – hoặc được định nghĩa như là một primary key của bảng. Ví dụ dưới đây mô tả cách xác định primary key của bảng:

```
DataColumn[] col = {tb.Columns["MaSo"]};
tb.PrimaryKey = col;
```

Nếu một primary key chứa nhiều hơn 1 column – chẳng hạn như HoDem và Ten – bạn có thể tạo ra một ràng buộc unique constraint trên các như ví dụ dưới đây:

```
DataColumn[] cols = {tb.Columns["HoDem"], tb.Columns["Ten"]};
tb.Constraints.Add(new UniqueConstraint("keyHoVaTen", cols));
```

Chúng ta sẽ xem xét cách thức tạo relationship cho các bảng và trộn dữ liệu ở phần tiếp theo.

4.4.1.3 DataRows

Dữ liệu được đưa vào table bằng cách tạo mới một đối tượng DataRow, gán giá trị cho các column của nó, sau đó bổ sung đối tượng DataRow này vào tập hợp Rows gồm các DataRow của table.

```
DataRow row;
row = tb.NewRow();      // Tạo mới DataRow
row["DonGia"] = 22.95;
row["SoLuong"] = 2;
row["MaSo"] = 12001;
tb.Rows.Add(row);       // Bổ sung row vào tập Rows
Console.WriteLine(tb.Rows[0]["ThanhTien"].ToString()); // 45.90
```

Một DataTable có các phương thức cho phép nó có thể commit hay roll back các thay đổi được tạo ra đối với table tương ứng. Để thực hiện được điều này, nó phải nắm giữ trạng thái của mỗi dòng dữ liệu bằng thuộc tính DataRow.RowState. Thuộc tính này được thiết lập bằng một trong 5 giá trị kiểu enumeration DataRowState sau: Added, Deleted, Detached, Modifed, hoặc Unchanged. Xem xét ví dụ sau:

```
tb.Rows.Add(row);                 // Added
tb.AcceptChanges();               // ...Commit changes
Console.Write(row.RowState);      // Unchanged
tb.Rows[0].Delete();              // Deleted
// Undo deletion
tb.RejectChanges();               // ...Roll back
```

```
Console.Write(tb.Rows[0].RowState);  // Unchanged
DataRow myRow;
MyRow = tb.NewRow();                 // Detached
```

Hai phương thức AcceptChanges và RejectChanges của DataTable là tương đương với các thao tác commit và rollback trong một CSDL. Các phương thức này sẽ cập nhất mọi thay đổi xảy ra kể từ khi table được nạp, hoặc từ khi phương thức AcceptChanges được triệu gọi trước đó. Ở ví dụ trên, chúng ta có thể khôi phục lại dòng bị xóa do thao tác xóa là chưa được commit trước khi phương thức RejectChanges được gọi. Điều đáng lưu ý nhất đó là, những thay đổi được thực hiện là ở trên table chứ không phải là ở data source.

ADO.NET quản lý 2 giá trị - ứng với 2 phiên bản hiện tại và nguyên gốc - cho mỗi column trong một dòng dữ liệu. Khi phương thức RejectChanges được gọi, các giá trị hiện tại sẽ được đặt khôi phục lại từ giá trị nguyên gốc. Điều ngược lại được thực hiện khi gọi phương thức AcceptChanges. Hai tập giá trị này có thể được truy xuất đồng thời thông qua các giá trị liệt kê DataRowVersion là: Current và Original:

```
DataRow r = tb.Rows[0];
r["DonGia"]= 14.95;
r.AcceptChanges();
r["DonGia"]= 16.95;
Console.WriteLine("Current: {0} Original: {1} ",
     r["Price", DataRowVersion.Current],
     r["Price", DataRowVersion.Original]);
Kết quả in ra:
Current: 16.95  Original: 14.95
```

4.4.1.4 DataView.

DataView đóng vai trò như tầng hiển thị dữ liệu lưu trữ trong DataTable. Nó cho phép người sử dụng sắp xếp, lọc và tìm kiếm dữ liệu.

```
//Giả sử đã có 1 dataset có tên là ds chứa dữ liệu của bảng DonHang
DataView dv = new DataView(ds.Tables["DonHang”];
// Lọc ra tất cả các hàng có giá từ 10 đến 100
dv.RowFilter = "soluong>=10 and soluong<=100";
//Sắp xếp tăng dần theo số lượng nếu số lượng bằng nhau thì sắp xếp giảm dần thêm đơn giá
dv.Sort = "soluong, dongia DESC";
```

4.4.2 Nạp dữ liệu vào DataSet

Chúng ta đã biết cách thành lập một DataTable và xử lý dữ liệu theo kiểu từng dòng một. Phần này sẽ trình bày phương pháp để dữ liệu và lược đồ dữ liệu được nạp tự động từ CSDL quan hệ vào các table trong DataSet.

4.4.2.1 Dùng DataReader để nạp dữ liệu vào DataSet

Đối tượng DataReader có thể được sử dụng để liên hợp đối tượng DataSet hay DataTable trong việc nạp các dòng dữ liệu kết quả (của query trong DataReader).

```
cmd.CommandText = "SELECT * FROM nhanvien";
DBDataReader rdr = cmd.ExecuteReader(CommandBehavior.CloseConnection);
DataTable dt = new DataTable("nhanvien");
dt.Load(rdr);   // Nạp dữ liệu và lược đồ vào table
Console.WriteLine(rdr.IsClosed);   // True
```

Đối tượng DataReader được tự động đóng sau khi tất cả các dòng dữ liệu được nạp vào table. Do đã sử dụng tham số CommandBehavior.CloseConnection trong phương thức ExecuteReader nên connection được đóng sau khi DataReader được đóng.

Nếu table đã có dữ liệu, phương thức Load sẽ trộn dữ liệu mới với các dòng dữ liệu đang có trong nó. Việc trộn này xảy ra chỉ khi các dòng dữ liệu có chung primary key. Nếu không có primary key được định nghĩa, các dòng dữ liệu sẽ được nối vào sau tập dữ liệu hiện tại. Chúng ta có thể sử dụng phương thức nạp chồng khác của phương thức Load để quy định cách thức làm việc. Phương thức Load với tham số kiểu enumeration LoadOption gồm 1 trong 3 giá trị OverwriteRow, PreserveCurrentValues, hoặc UpdateCurrentValues tương ứng với tùy chọn ghi đè nguyên dòng, giữ lại các giá trị hiện tại, hoặc cập nhật các giá trị hiện tại. Đoạn code dưới đây minh họa cách trộn dữ liệu vào các dòng hiện tại theo kiểu ghi đè các giá trị hiện tại:

```
cmd.CommandText = "SELECT * FROM nhanvien WHERE diachi='a'";
DBDataReader rdr = cmd.ExecuteReader();
DataTable dt = new DataTable("nhanvien");
dt.Load(rdr);
Console.Write(dt.Rows[0]["HoTen"]);  // giả sử giá trị nhận được là "tnv spider"
// Gán khóa chính
DataColumn[] col = new DataColumn[1];
col[0] = dt.Columns["Manv"];
```

```
dt.PrimaryKey = col;
DataRow r = dt.Rows[0];   // lấy dòng đầu tiên
r["HoTen"] = "ten moi";   // thay đổi giá trị của cột HoTen
// Do reader đã bị đóng sau khi nạp vào data table nên phải giờ phải fill lại
rdr = cmd.ExecuteReader(CommandBehavior.CloseConnection);
// Trộn dữ liệu với các dòng hiện tại. Ghi đè các giá trị hiện tại
dt.Load(rdr, LoadOption.UpdateCurrentValues);
// Giá trị cập nhật đã bị ghi đè!!!
Console.Write(dt.Rows[0]["HoTen"]);  // "tnv spider"
```

4.4.2.2 Nạp dữ liệu vào DataSet bằng DataAdapter

Đối tượng DataAdapter có thể được dùng để nạp một table hiện có vào một table khác, hoặc tạo mới và nạp dữ liệu cho table từ kết quả của một query. Bước đầu tiên là tạo ra một đối tượng DataAdapter tương ứng với data provider cụ thể. Dưới đây là các ví dụ để tạo ra đối tượng DataAdapter:

(1) Tạo từ Connection string và câu truy vấn SELECT:

```
String sql = "SELECT * FROM nhanvien";
SqlDataAdapter da = new SqlDataAdapter(sql, connStr);
```

(2) Tạo từ đối tượng Connection và câu truy vấn SELECT:

```
SqlConnection conn = new SqlConnection(connStr);
SqlDataAdapter da = new SqlDataAdapter(sql, conn);
```

(3) Gán đối tượng Command cho thuộc tính SelectCommand

```
SqlDataAdapter da = new SqlDataAdapter();
SqlConnection conn = new SqlConnection(connStr);
da.SelectCommand = new SqlCommand(sql, conn);
```

Sau khi đối tượng DataAdapter đã được tạo ra, phương thức Fill của nó được thực thi để nạp dữ liệu vào table (đang tồn tại hoặc tạo mới). Ở ví dụ dưới đây, một table mới được tạo ra với tên mặc định là "Table":

```
DataSet ds = new DataSet();
// Tạo ra một DataTable, nạp dữ liệu vào DataTable, và đưa DataTable vào DataSet
int nRecs = da.Fill(ds);  // trả về số lượng record được nạp vào DataTable
// Nếu muốn đặt tên cho DataTable trong DataSet thay vì lấy tên mặc định
// thì sử dụng code như thế này
```

```
int nRecs = da.Fill(ds, "nhanvien ")
```

Với một table đang tồn tại, tác dụng của lệnh Fill tùy thuộc vào table có primary hay không. Nếu có, những dòng dữ liệu có khóa trùng với dòng dữ liệu mới sẽ được thay thế. Các dòng dữ liệu mới không trùng với dữ liệu hiện có sẽ được nối vào sau DataTable.

4.4.3 Bài thực hành

Bài thực hành về DataAdapter và DataSet

Ví dụ này sử dụng cơ sở dữ liệu **quanlythuvien** như trong bài thực hành 4.3.4

Thiết kế form **frmtimkiemsach** để tìm theo tên sách hoặc tên tác giả như sau:

frmtimkiemsach sử dụng các trường, phương thức và sự kiện sau:

```
using System.Data;
using System.Data.SqlClient;
namespace baimau
{
    public partial class frmtimkiemsach : Form
    {
        SqlConnection cn = new SqlConnection("Data Source=nhha;Initial Catalog="+
         "quanlythuvien;Trusted_Connection=yes");
        SqlCommand cmd=new SqlCommand();
        SqlDataAdapter da= new SqlDataAdapter() ;
        DataSet ds= new DataSet() ;
        public frmtimkiemsach()...
        private int  Tongsoluong()...
        private int Tongsoluongtk(DataView dv)...
        private void txttimkiem_KeyPress(object sender, KeyPressEventArgs e)...
        private void frmtimkiemsach_Load(object sender, EventArgs e)...
    }
}
```

Các điều khiển

Tên điều khiển	**Thuộc tính**
Form	**Name**: **frmtimkiemsach** **Text**:Tìm kiếm theo nhan đề hoặc tên tác giả
Label	**Text**: Nhập tên sách hoặc tên tác giả cần tìm
TextBox	**Name**: txttimkiem
dataGridView	**Name**: dataGridView1
statusStrip	**Name:** thanhtrangthai **Items:** Add thêm 2 statusLabel: với tên ketquatim và tóngoluong

Các trường:

Tên trường	**Ý nghĩa**
Cn	Dùng để kết nối đến cơ sở dữ liệu quanlythuvien
cmd	sqlCommand sử dụng câu lệnh select để hiển thị và tìm kiếm sách
da	SqlDataAdapter chứa cmd và cn
ds	DataSet chứa dữ liệu của bảng sách hoặc chứa kết quả tìm kiếm

Các phương thức

+ Hàm dựng frmtimkiemsach để tạo giao diện

```
public frmtimkiemsach()
      {
         InitializeComponent();
      }
```

+ Phương thức Tongsoluong: được sử dụng để tính tổng số lượng sách của các sách lưu trong Dataset ds.

```
  private int  Tongsoluong()
      {  int  s=0;
         foreach (DataRow r in ds.Tables["sach"].Rows)
         {
            s += (int)r["soluong"];
         }
         return s;
      }
```

+ Phương thức **Tongsoluongtk** tính tổng số lượng sách trong DataView dv, dv chứa thông tin các sách tìm kiếm được.

```
 private int Tongsoluongtk(DataView dv)
      {
         int s = 0;
         foreach (DataRow r in dv.ToTable("sach").Rows )
         {
            s += (int)r["soluong"];
         }
         return s;
      }
```

+ Sự kiện **frmtimkiemsach_Load:**Nạp thông tin của 4 quyển sách đầu tiên theo thứ tự giảm dần của ngaynhap vào DataSet ds với tên bảng trong DataSet là sach, sau đó hiển thị thông tin của bảng sach trong DataSet vào dataGridView1, đưa tổng số sách và tổng số lượng sách trong DataSet vào thanh trạng thái

```
    private void frmtimkiemsach_Load(object sender, EventArgs e)
      {
         cn.Open(); // Mở kết nối
         cmd.CommandText = "select top 4 * from sach order by ngaynhap desc" ;
         cmd.Connection = cn;
         da.SelectCommand = cmd;
         da.Fill(ds, "sach"); // Nạp dữ liệu vào DataSet
         dataGridView1.DataSource = ds.Tables["sach"]; // Nạp dữ liệu vào dataGridView1
        // Nạp dữ liệu vào thanh trạng thái
   thanhtrangthai.Items[0].Text    = "Tổng số sách:" +
                  ds.Tables["sach"].Rows.Count.ToString();
```

```
        thanhtrangthai.Items[1].Text = "Tổng số lượng:" + Tongsoluong().ToString();
    }
```

+ Sự kiện **txttimkiem_KeyPress:** Khi người sử dụng nhấn Enter trên txttimkiem thì việc tìm kiếm tương đối bắt đầu: Tạo ra 1 DataView dv chứa dữ liệu của bảng sách trong DataSet ds, lọc trong DataView dv ra thông tin của các quyển sách gần giống với dữ liệu nhập trên txttimkiem, sau đó đưa kết quả lọc ra trên dataGridView1 và thanh trạng thái.

```
private void txttimkiem_KeyPress(object sender, KeyPressEventArgs e)
    {
        if (e.KeyChar == 13)
        {
            DataView dv = new DataView(ds.Tables["sach"]); //Nạp dữ liệu vào DataView
     //bắt đầu lọc dữ liệu
            dv.RowFilter = "nhande like '%" + txttimkiem.Text + "%' or tacgia like '%"
                                + txttimkiem.Text + "%'";
     dataGridView1.DataSource = dv; //Nạp kết quả lọc trong dv vào dataGridView1
          // Đưa số quyển sách và tổng số lượng sách lọc được vào thanh trang thái
            thanhtrangthai.Items[0].Text = "Số kết quả tìm thấy được: " +
    dv.Count.ToString() + "/" + ds.Tables["sach"].Rows.Count.ToString();
            thanhtrangthai.Items[1].Text = "Tổng số lượng tìm thấy được:" +
    Tongsoluongtk(dv).ToString() + "/" + Tongsoluong().ToString();
        }
    }
```

4.4.4 Cập nhật CSDL bằng DataAdapter

Sau khi DataAdapter đã nạp dữ liệu vào table, connection sẽ được đóng, và các thay đổi sau đó đối sau đó tạo ra cho dữ liệu sẽ chỉ có ảnh hưởng trong DataSet chứ không phải là ở dữ liệu nguồn! Để thực sự cập nhật các thay đổi này lên nguồn dữ liệu, DataAdapter phải được sử dụng để khôi phục connection và gửi các dòng dữ liệu đã được thay đổi lên CSDL.

Ngoài SelectCommand, DataAdapter có thêm 3 thuộc tính Command nữa, gồm InsertCommand, DeleteCommand và UpdateCommand, làm nhiệm vụ thực hiện các thao tác tương ứng với tên thuộc tính của chúng (chèn, xóa, cập nhật). Các Command này được thực thi khi phương thức Update của DataAdapter được triệu gọi. Khó khăn nằm ở chỗ tạo ra các query command phức tạp này (cú pháp của câu lệnh SQL tương ứng càng dài dòng và phức tạp khi số lượng column nhiều lên). Rất may là các data provider đều có cài đặt một lớp gọi là CommandBuilder dùng để quản lý việc tạo các Command nói trên

một cách tự động.

4.4.4.1 CommandBuilder

Một đối tượng CommandBuilder sẽ sinh ra các Command cần thiết để thực hiện việc cập nhật nguồn dữ liệu tạo ra bởi DataSet. Cách tạo đối tượng CommandBuilder là truyền đối tượng DataAdapter cho phương thức khởi dựng của nó; sau đó, khi phương thức DataAdapter.Update được gọi, các lệnh SQL sẽ được sinh ra và thực thi. Đoạn code dưới đây minh họa cách thức thay đổi dữ liệu ở một DataTable và cập nhật lên CSDL tương ứng bằng DataAdapter:

```
//Giả sử đã có 1 DataSet ds chứa dữ liệu của bảng khoa
DataTable dt= ds.Tables["khoa"];
// (1) Dùng commandBuilder để sinh ra các Command cần thiết để update
SqlCommandBuilder sb = new SqlCommandBuilder(da);
// (2) Thực hiện thay đổi dữ liệu: thêm 1 khoa mới
DataRow drow = dt.NewRow();
drow["Makhoa"] = 12;
drow["tenkhoa"] = "abc";
dt.Rows.Add(drow);
// (3) Thực hiện thay đổi dữ liệu: xóa 1 khoa
dt.Rows[4].Delete();
// (4) Thực hiện thay đổi dữ liệu: thay đổi giá trị 1 dòng dữ liệu
dt.Rows[5]["tenkhoa"] = "this must be changed";
// (5) Tiến hành cập nhật lên CSDL
int nUpdate = da.Update(ds, "khoa");
MessageBox.Show("Số dòng được thay đổi: " + nUpdate.ToString());  // → 3
```

Có một số hạn chế khi sử dụng CommandBuilder: Command Select ứng với DataAdapter chỉ được tham chiếu đến 1 table, và table nguồn trong CSDL phải bao gồm một primary key hoặc một column chứa các giá trị duy nhất. Column này (hay tổ hợp các columns) phải được bao gồm trong command Select ban đầu.

4.4.4.2 Đồng bộ hóa dữ liệu giữa DataSet và CSDL

Như đã minh họa trong ví dụ này, việc sử dụng DataAdapter làm đơn giản hóa và tự động hóa quá trình cập nhật CSDL hoặc bất kỳ data source nào. Tuy nhiên, có một vấn đề ở đây: multi-user (nhiều người sử dụng). Mô hình Ngắt kết nối được dựa trên cơ chế Optimistic Concurrency, một cách tiếp cận mà trong đó các dòng dữ liệu ở data source không bị khóa (lock) giữa thời gian mà chúng được đọc và thời gian mà các cập nhật được áp dụng cho data source. Trong khoảng thời gian này, user khác có thể cũng cập

nhật data source. Nếu có thay đổi xảy ra kể từ lần đọc trước đó thì phương thức Update sẽ nhận biết được và không cho áp dụng thay đổi đối với các dòng dữ liệu đó.

Có hai phương án cơ bản để giải quyết lỗi concurrency (tương tranh) khi có nhiều cập nhật được áp dụng: roll back tất cả các thay đổi nếu như xuất hiện xung đột (violation), hoặc áp dụng các cập nhật không gây ra lỗi và xác định những cập nhật có gây ra lỗi để có thể xử lý lại.

4.4.4.3 Sử dụng Transactions để Roll Back nhiều cập nhật

Khi thuộc tính DataAdapter.ContinueUpdateOnErrors được thiết lập là false, một ngoại lệ sẽ được ném ra khi một thay đổi dòng dữ liệu không thể thực hiện được. Điều này sẽ ngăn các cập nhật tiếp theo được thực thi, nhưng lại không ảnh hưởng đến các cập nhật đã xuất hiện trước ngoại lệ đó. Do những cập nhật có thể có liên quan với nhau, ứng dụng thường là cần chiến lược *hoặc là tất cả, hoặc là không (all-or-none).* Cách dễ nhất để thực thi chiến lược này là tạo ra một transaction trong đó tất cả các command update sẽ được thực thi. Để thực hiện điều này, tạo ra một đối tượng SqlTransaction và gắn nó với SqlDataAdapter.SelectCommand bằng cách truyền nó cho hàm khởi dựng của nó. Nếu có ngoại lệ xảy ra, phương thức Rollback sẽ được thực thi để undo mọi thay đổi trước đó; nếu không có ngoại lệ nào xuất hiện, phương thức Commit được thực thi để áp dụng tất cả các command update. Dưới đây là một ví dụ:

```
SqlDataAdapter da = new SqlDataAdapter();
SqlCommandBuilder sb = new SqlCommandBuilder(da);
SqlTransaction tran;
SqlConnection conn = new SqlConnection(connStr);
conn.Open();      // Connection phải được dùng với Transaction
// (1) Tạo ra một transaction
SqlTransaction tran = conn.BeginTransaction();
// (2) Gắn SelectCommand với transaction
da.SelectCommand = new SqlCommand(sql, conn, tran);
DataSet ds = new DataSet();
da.Fill(ds, "docgia");
//
// Code ở phần này thực hiện các cập nhật lên các dòng dữ liệu ở DataSet
try
{
  int updates = da.Update(ds, "docgia");
  MessageBox.Show("Cập nhật: " + updates.ToString());
}
```

```
// (3) Nếu có ngoại lệ xuất hiện, roll back mọi cập nhật trong transaction
catch (Exception ex)
{
  MessageBox.Show(ex.Message);   // Lỗi khi cập nhật
  if (tran != null)
  {
    tran.Rollback();      // Roll back mọi cập nhật
    tran = null;
    MessageBox.Show("Tất cả các cập nhật đã bị roll back.");
  }
}
finally
{
  // (4) Nếu không có lỗi, commit tất cả các cập nhật
  if (tran != null)
  {
     tran.Commit();
     MessageBox.Show("Tất cả đã được cập nhật thành công. ");
     tran = null;
  }
}
conn.Close();
```

4.4.4.4 Xác định các dòng gây ra lỗi cập nhật

Khi thuộc tính DataAdapter.ContinueUpdateOnErrors được thiết lập là True, các xử lý sẽ không ngưng nếu có một dòng dữ liệu không thể cập nhật được. Thay vào đó, DataAdapter sẽ cập nhật tất cả các dòng dữ liệu không gây ra lỗi. Sau đó, lập trình viên có thể xác định các dòng dữ liệu không cập nhật được và quyết định cách xử lý chúng.

Các dòng dữ liệu không cập nhật được có thể dễ dàng được xác định qua thuộc tính DataRowState của chúng (đã được trình bày trong phần mô tả DataRow). Các dòng dữ liệu đã được cập nhật thành công sẽ có trạng thái Unchanged; trong khi đó các dòng dữ liệu không cập nhật thành công sẽ mang trạng thái Added, Deleted hoặc Modified. Đoạn code dưới đây minh họa cách lặp qua các dòng dữ liệu và xác định các dòng chưa được cập nhật.

```
// SqlDataAdapter da nạp dữ liệu của bảng docgia
da.ContinueUpdateOnError = true;
DataSet ds = new DataSet();
try
```

```
{
  da.Fill(ds, "docgia");
  DataTable dt = ds.Tables["docgia"];
  SqlCommandBuilder sb = new SqlCommandBuilder(da);
  // ... Các thao tác cập nhật
  dt.Rows[29].Delete();                    // Delete
  dt.Rows[30]["HoTen"] = "try to change";    // Update
  dt.Rows[30][Madocgia] = 1234;            // Update
  dt.Rows[31]["HoTen"] = "XYZ";              // Update
  DataRow drow = dt.NewRow();
  drow["HoTen"] = "tnv spider";
  drow["Madocgia"] = 25;
  dt.Rows.Add(drow);                       // insert
  // Submit updates
  int updates = da.Update(ds, "docgia");
  if (ds.HasChanges())
  {
    // Load rows that failed into a DataSet
    DataSet failures = ds.GetChanges();
    int rowsFailed = failures.Rows.Count;
    Console.WriteLine("Số dòng không thể cập nhật: " + rowsFailed);
    foreach (DataRow r in failures.Tables[0].Rows )
    {
      string state = r.RowState.ToString());
      // Phải hủy bỏ thay đổi để hiển thị dòng đã bị xóa
      if (r.RowState == DataRowState.Deleted)
          r.RejectChanges();
      string iMadocgia= ((int)r["Madocgia"]).ToString();
      string msg = state + " Madocgia: " + iMadocgia;
      Console.WriteLine(msg);
    }
  }
```

Lưu ý rằng ngay cả khi thao tác xóa xuất hiện trước, nó cũng không có tác dụng đối với các thao tác khác. Câu lệnh SQL xóa hay cập nhật một dòng dữ liệu được dựa theo giá trị của primary key chứ không liên quan đến vị trí của nó. Ngoài ra các cập nhật trên cùng một dòng được kết hợp và đếm như là 1 cập nhật cho dòng đó bởi phương thức Update. Ở ví dụ trên, các cập nhật cho dòng 30 được tính như là 1 cập nhật.

4.4.5 Định nghĩa Relationships giữa các Table trong DataSet

Một DataRelation là một mối quan hệ parent-child giữa hai đối tượng DataTables. Nó được định nghĩa dựa trên việc so khớp các columns trong 2 DataTable. Cú pháp hàm khởi dựng là như sau:

```
public DataRelation(
  string relationName,
  DataColumn parentColumn,
  DataColumn childColumn)
```

Một DataSet có một thuộc tính Relations giúp quản lý tập hợp các DataRelation đã được định nghĩa trong DataSet. Sử dụng phương thức Relations.Add để thêm các DataRelation vào tập hợp Relations. Ví dụ dưới đây thiết lập mối quan hệ giữa hai bảng khoa và docgia để có thể liệt kê danh sách các docgia của mỗi khoa.

```
string connStr="Data Source=tên máy chủ;Initial Catalog=quanlythuvien;
Trusted_Connection=yes";
DataSet ds = new DataSet();
// (1) Fill bảng docgia
string sql = "SELECT * FROM docgia”;
SqlConnection conn = new SqlConnection(connStr);
SqlCommand cmd = new SqlCommand();
SqlDataAdapter da = new SqlDataAdapter(sql, conn);
da.Fill(ds, "docgia");
// (2) Fill bảng khoa
sql = "SELECT * FROM khoa”;
da.SelectCommand.CommandText = sql;
da.Fill(ds, "khoa");
// (3) Định nghĩa relationship giữa 2 bảng khoa và docgia
DataTable parent = ds.Tables["khoa"];
DataTable child = ds.Tables["docgia"];
DataRelation relation = new DataRelation("khoa_docgia", parent.Columns["makhoa"],
    child.Columns["makhoa"]);
// (4) Đưa relationship vào DataSet
ds.Relations.Add(relation);
// (5) Liệt kê danh sách các đọc giả của từng khoa
foreach (DataRow r in parent.Rows)
{
  Console.WriteLine(r["tenkhoa"]);      // Tên khoa
  foreach (DataRow rc in r.GetChildRows("khoa_docgia"))
  {
    Console.WriteLine("  " + rc["HoTen"]);
```

```
  }
}
/*
  Ví dụ kết quả:
    Khoa Tin
      Nguyễn Văn Trung
      Ngô Anh Tuấn
      Lê Thanh Hoa
    Khoa Toán
      Nguyễn Thị Hoa
      Trần Văn Phúc
*/
```

Khi một relationship được định nghĩa giữa 2 tables, nó cũng sẽ thêm một ForeignKeyConstraint vào tập hợp Constraints của DataTable con. Constraint này quyết định cách mà DataTable con bị ảnh hưởng khi các dòng dữ liệu ở phía DataTable cha bị thay đổi hay bị xóa. Trong thực tế, điều này có nghĩa là khi bạn xóa 1 dòng trong DataTable cha, các dòng con có liên quan cũng bị xóa – hoặc cũng có thể là các key bị đặt lại giá trị thành NULL. Tương tự như thế, nếu một giá trị key bị thay đổi trong DataTable cha, các dòng dữ liệu có liên quan trong DataTable con cũng có thể bị thay đổi theo hoặc bị đổi giá trị thành NULL.

Các luật như trên được xác định bởi các thuộc tính DeleteRule và UpdateRule của constraint. Các luật này được nhận các giá trị liệt kê sau đây:

- Cascade. Xóa/Cập nhật các dòng dữ liệu có liên quan trong DataTable con. Đây là giá trị mặc định.
- None. Không làm gì.
- SetDefault. Thiết lập các giá trị khóa trong các dòng dữ liệu có liên quan của DataTable con thành giá trị mặc định của column tương ứng.
- SetNull. Thiết lập các giá trị khóa trong các dòng dữ liệu có liên quan của DataTable con thành null.

Xem xét ví dụ dưới đây:

```
// (1) Thêm một dòng với khóa mới vào DataTable con
DataRow row = child.NewRow();
row["Makhoa"] = 999; // giả sử trong bảng khoa không có record nào có Makhoa = 999
child.Rows.Add(row); // Không được do 999 không tồn tại trong DataTable cha
```

```
// (2) Xóa một dòng trong DataTable cha
row = parent.Rows[0];
row.Delete();        // Xóa các dòng trong DataTable con có khóa này
// (3) Tạm thời vô hiệu hóa constraints và thử thêm dòng mới
ds.EnforceConstraints = false;
row["Makhoa"] = 999;
child.Rows.Add(row);   // Được chấp nhận!!!
ds.EnforceConstraints = true;    // Kích hoạt constraint trở lại
// (4) Thay đổi constraint để đặt các dòng dữ liệu thành null nếu DataTable   thay đổi
((ForeignKeyConstraint)child.Constraints[0]).DeleteRule = Rule.SetNull;
```

Lưu ý rằng thuộc tính EnforeceConstraint được đặt thành false sẽ làm vô hiệu hóa tất cả các constraint – điều này trong thuật ngữ CSDL gọi là bỏ qua tính toàn vẹn tham chiếu. Điều này cho phép một khoa được bổ sung vào thậm chí khi cả column Makhoa không tương ứng với dòng nào trong bảng khoa. Nó cũng cho phép một dòng khoa được xóa ngay cả khi có nhiều dòng tương ứng với nó trong bảng docgia.

4.5 Sử dụng Data Binding

Chúng ta đã đề cập đến các điều khiển để thiết kế giao diện như TextBox, ListBox, RadioButton, ComBoBox và các các điều khiển của ADO.NET như DataSet, DataTable và DataView. Các điều khiển này làm việc một cách độc lập với nhau, tuy nhiên trong một số tình huống chúng cần kết hợp lại với nhau. Ví dụ ta cần hiển thị tên khoa từ cơ sở dữ liệu ra 1 TextBox, khi đó ta cần tạo ra 1 DataSet chứa dữ liệu của bảng khoa và 1 TextBox, sau đó liên kết dữ liệu trong DataSet vào TextBox. Sự kết hợp giữa hai điều khiển này có thể sử dụng DataBinding.

4.5.1 Các loại của Binding

ADO.NET cung cấp 2 loại Binding:

- *DataBinding đơn giản* (Simple DataBinding): Tại một thời điểm, một giá trị đơn trong DataSet có thể bị buộc vào bất kỳ một điều khiển.

Ví dụ: giả sử đã có 1 DataSet ds chứa dữ liệu của bảng Khoa, cần buộc tên khoa vào TextBox txttenkhoa:

```
txttenkhoa.DataBindings.Add("Text", ds, "khoa.tenkhoa");
```

Khi đó mọi thay đổi trên DataSet ds sẽ ảnh hưởng đến TextBox txtdocgia và ngược lại.

- *DataBinding phức tạp* (Complex DataBinding): Các dữ liệu trong DataSet bị buộc vào một điều khiển thay vì chỉ một giá trị đơn. Chỉ có DataGidView và ComboBox hỗ trợ chức năng DataBinding phức tạp.

Ví dụ: giả sử đã có 1 DataSet ds chứa dữ liệu của bảng Khoa, cần buộc tên khoa vào ComboBox cmbkhoa và buộc toàn bộ dữ liệu của bảng khoa vào DataSet ds:

```
//Buộc tenkhoa của bảng khoa trong DataSet ds  vào  cmbkhoa
      cmbkhoa.DataSource = ds;
       cmbkhoa.DisplayMember = "khoa.tenkhoa";
      //Buộc toàn bộ dữ liệu của bảng khoa trong DataSet ds vào DataGridView dgvkhoa
       dgvKhoa.DataSource = ds;
       dgvKhoa.DataMember = "khoa";
```

4.5.2 Các nguồn dữ liệu của DataBinding

Nhiều thành phần có thể hoạt động như nguồn dữ liệu của DataBinding. Bất kỳ các thành phần được cài đặt từ giao diện Ilist có thể xem là như nguồn dữ liệu của DataBinding. Các ví dụ sau minh họa bằng cách nào để sử dụng DataTable, DataView, DataSet và Mảng như là nguồn dữ liệu để cài đặt DataBinding đơn giản và phức tạp.

- **DataTable**: Loại dữ liệu này lưu trữ dữ liệu của một bảng trong cơ sở dữ liệu.

```
      DataTable t = ds.Tables["khoa"];
      //DataBinding đơn giản
       txttenkhoa.DataBindings.Add("Text", t, "tenkhoa");
      //DataBinding phức tạp
       cmbkhoa.DataSource = t;
       cmbkhoa.DisplayMember = "tenkhoa";
       dgvKhoa.DataSource = t;
```

- **DataView:**

```
DataView  dv =new DataView (ds.Tables["khoa"]);
        //DataBinding đơn giản
       txttenkhoa.DataBindings.Add("Text", dv, "tenkhoa");
      //DataBinding phức tạp
cmbkhoa.DataSource = dv;
       cmbkhoa.DisplayMember = "tenkhoa";
       dgvKhoa.DataSource = dv;
```

- **DataSet:** như ví dụ ở phần 3.5.1

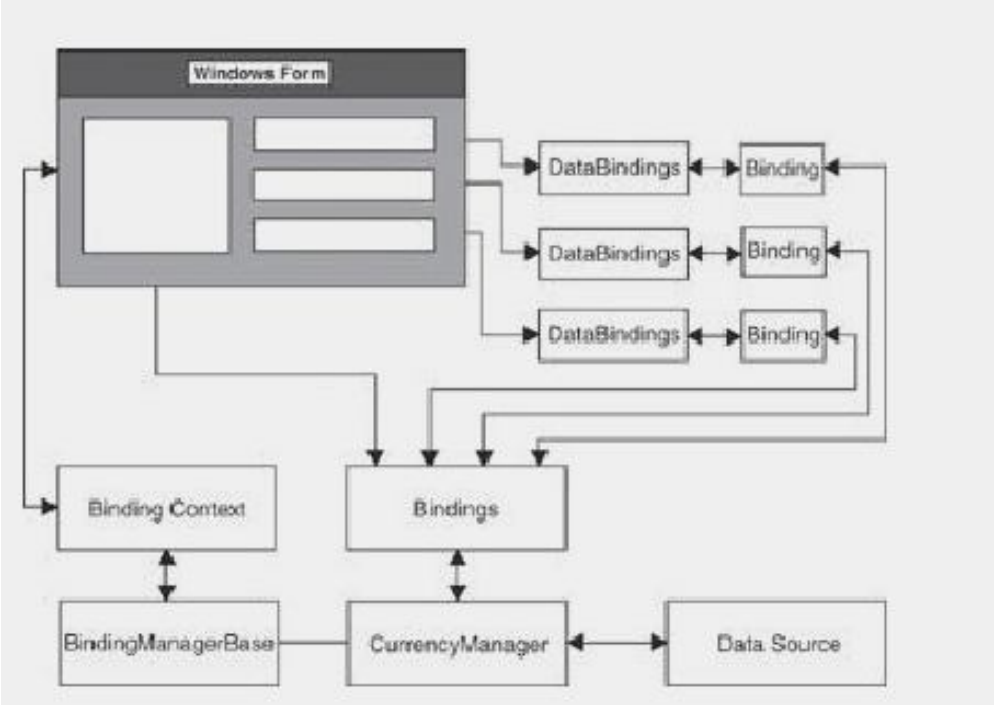
- **Mảng:**

```
      int[] t = new int[4] { 12, 2, 3, 4 };
//DataBinding đơn giản
txttenkhoa.DataBindings.Add("Text", t, "");
```

```
        //DataBinding phức tạp
    cmbkhoa.DataSource = t;
```

4.5.3 BindingContext

Sơ đồ bên dưới chỉ ra cách buộc dữ liệu từ nguồn dữ liệu vào các điều khiển trên Form. Phần này chủ yếu thảo luận về các lớp BindingContext, CurrencyManager và chỉ ra bằng cách nào chúng tương tác khi dữ liệu bị buộc vào một hoặc nhiều điều khiển trên Form:



BindingContext

Mỗi Windows Form đều có một thuộc tính BindingContext. Một BindingContext có một tập hợp các BindingManagerBase, các đối tượng này được tạo ra khi dữ liệu buộc vào một điều khiển.

Nếu nguồn dữ liệu bao gồm một danh sách các đối tượng như a DataTable, DataView hoặc bất kỳ đối tượng nào cài đặt trên giao diện Ilist khi đó CurrencyManager sẽ được sử dụng. Một CurrencyManager có thể duy trì một vị trí hiện thời (current position) bên trong nguồn dữ liệu. Nếu nguồn dữ liệu chỉ trả về một giá trị đơn khi đó PropertyManager sẽ được lưu trữ bên trong BindingContext .

Một CurrencyManager hoặc PropertyManager chỉ được tạo ra một lần cho một nguồn dữ liệu. Nếu hai TextBox bị buộc vào 1 dòng của DataTable khi đó chỉ có mộ

CurrencyManager được tạo ra trong BindingContext

Khi tạo ra một điều khiển trên form, điều khiển này sẽ được liên kết với bộ quản lý buộc dữ liệu trên form (form's binding manager). Khi tạo ra một điều khiển thì thuộc tính BindingContext của nó bằng null. Để buộc dữ liệu vào một điều khiển ta dùng thuộc tính DataBindings như ví dụ bên dưới buộc mã khoa và tên khoa trong DataSet ds vào TextBox1 và textBox2 .

```
textBox1.DataBindings.Add("Text", ds, "khoa.makhoa");
textBox2.DataBindings.Add("Text", ds, "khoa.tenkhoa");
```

CurrencyManager and PropertyManager

Khi buộc dữ liệu vào 1 điều khiển trên form khi đó một CurrencyManager hoặc một PropertyManager tương ứng sẽ được tạo ra. Mục đích của lớp này là xác định vị trí của mẫu tin hiện thời bên trong nguồn dữ liệu và khi vị trí này thay đổi thì dữ liệu trên các điều khiển bị buộc trên form sẽ tự động thay đổi theo.

Các thuộc tính của BindingContext:

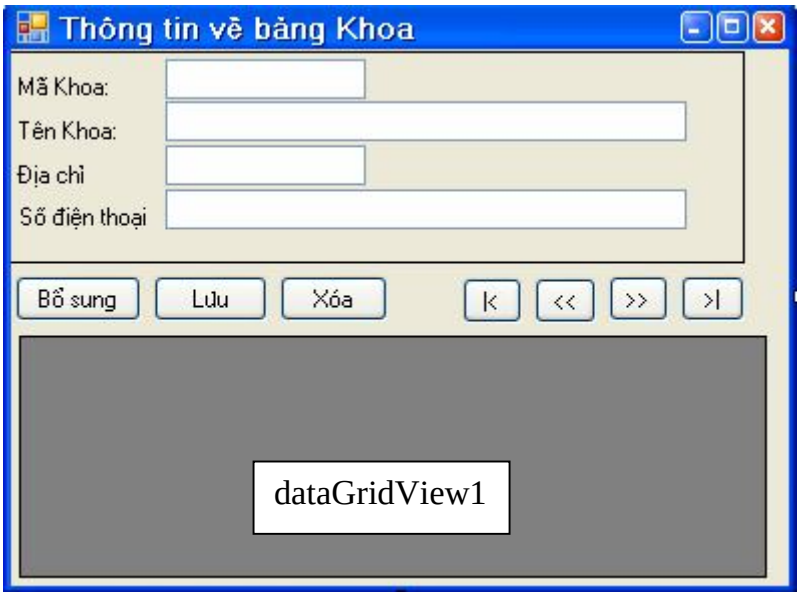
Thuộc tính	Mô tả
Bindings	Tập hợp các đối tượng Binding được quản lý bởi CurrencyManager
Count	Số dòng được quả lý trong CurrencyManager
Current	Giá trị của các đối tượng hiện thời trong nguồn dữ liệu
Position	Gets hoặc sets đối tượng hiện thời trong danh sách các đối tượng được quản lý trong CurrencyManager

4.5.4 Bài thực hành

Bài thực hành về DataSet và DataBinding

Ví dụ này sử dụng cơ sở dữ liệu **quanlythuvien** như trong bài thực hành 4.3.4

Thiết kế form **frmkhoa** để nhập, xóa, lưu và duyệt qua các mẫu tin trong bảng khoa như sau:



frmkhoa sử dụng các trường, phương thức và sự kiện sau:

```
using System.Data.SqlClient;
namespace baimau
{ public partial class frmKhoa : Form
    {
        SqlConnection cn = new SqlConnection("Data Source=nhha;Initial Catalog=" +
            "quanlythuvien;Trusted_Connection=yes");
        SqlCommand cmdkhoa = new SqlCommand();
        SqlDataAdapter dakhoa = new SqlDataAdapter();
        DataSet ds = new DataSet();
        SqlCommandBuilder cb;
        public frmKhoa()...

        private void BuocCacDieuKhien()...
        private void frmKhoa_Load(object sender, EventArgs e)...
        private void butFirst_Click(object sender, EventArgs e)...
        private void butPre_Click(object sender, EventArgs e)...
        private void butNext_Click(object sender, EventArgs e)...
        private void butLast_Click(object sender, EventArgs e)...
        private void butBosung_Click(object sender, EventArgs e)...
        private void butLuu_Click(object sender, EventArgs e)...
        private void butXoa_Click(object sender, EventArgs e)...

    }
}
```

Các điều khiển

Tên điều khiển	Thuộc tính
Form	**Name**: **frmKhoa** **Text**: Thông tin về bảng Khoa
Label	Tạo ra 4 lable với các **Text**: Mã Khoa, Tên Khoa, Địa chỉ, Số điện thoại
TextBox	Tạo ra 4 TextBox với các Name: txtMakhoa, txtTenKhoa, txtDiachi, txtSodienthoai
Button	Tạo ra 7 Button với các Name: butBosung, butLuu, butXoa, butFirst, butPre, butNext, butLast
dataGridView	**Name**: dataGridView1

Các phương thức

+ Hàm dựng frmKhoa để tạo giao diện:

```
public frmKhoa()
      {   InitializeComponent();   }
```

+ Phương thức BuocCacDieuKhien(): Buộc dữ liệu vào dataGridView1 và các textBox

```
 private void BuocCacDieuKhien()
      {  //Buộc dữ liệu vào dataGridView1
         dataGridView1.DataSource = ds;dataGridView1.DataMember = "khoa";
       // Buộc dữ liệu vào các  textBox
         txtMaKhoa.DataBindings.Add("Text", ds, "khoa.makhoa");
         txtTenKhoa.DataBindings.Add("Text", ds, "khoa.tenkhoa");
         txtdiachi.DataBindings.Add("Text", ds, "khoa.diachi");
         txtSodienthoai.DataBindings.Add("Text", ds, "khoa.sdt");
      }
```

+ **Sự kiện**: frmKhoa_Load() được sử dụng để kết nối dữ liệu, tạo ra DataSet ds chứa toàn bộ dữ liệu của bảng khoa, buộc dữ liệu vào cho các điều khiển và tạo ra 1SqlCommandBuilder cb để quản lý việc nhập thêm, xóa và lưu dữ liệu của SqlDataAdapter dakhoa .

```
 private void frmKhoa_Load(object sender, EventArgs e)
   {
      cn.Open(); // Kết nối dữ liệu
      cmdkhoa = new SqlCommand("select * from khoa", cn);
      dakhoa = new SqlDataAdapter(cmdkhoa);
      dakhoa.Fill(ds, "khoa");
      BuocCacDieuKhien();
      cb = new SqlCommandBuilder(dakhoa);
   }
```

+ **Sự kiện**: butFirst_Click: Di chuyển con trỏ về mẫu tin đầu tiên

```
private void butFirst_Click(object sender, EventArgs e)
      {
         this.BindingContext[ds, "khoa"].Position = 0;
```

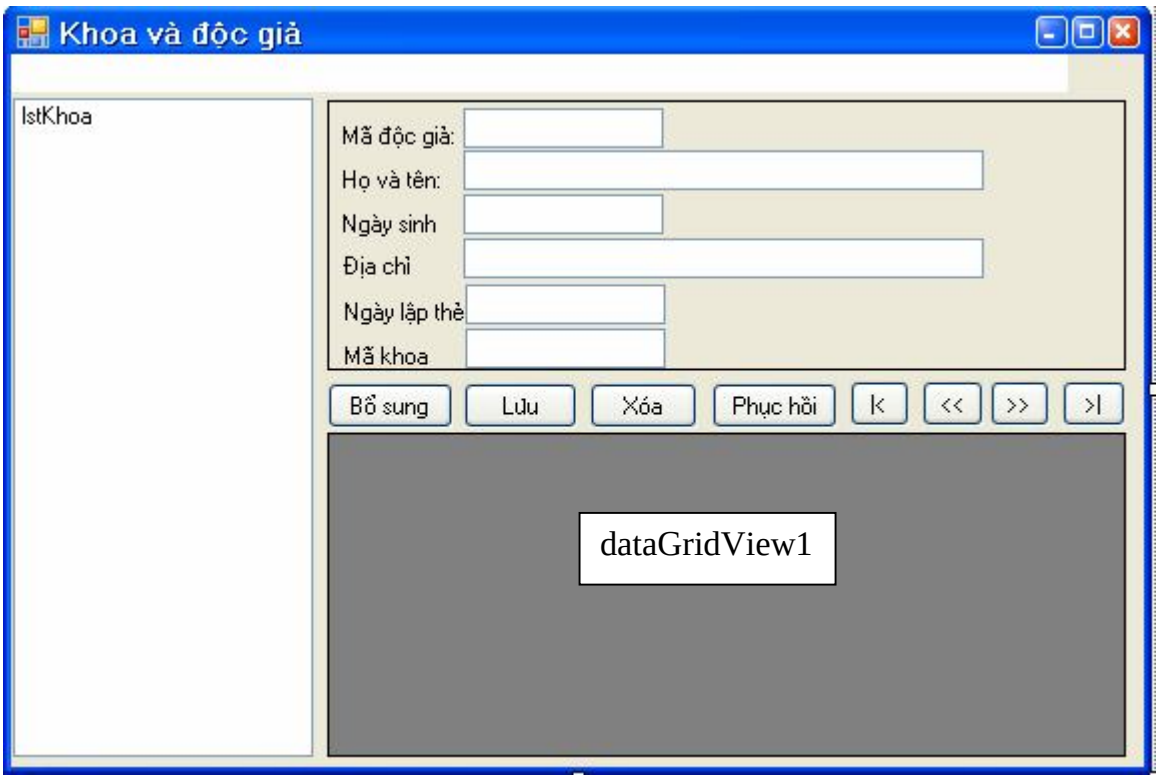
```
    }
```
+ **Sự kiện**: butPre_Click: Di chuyển con trỏ về mẫu tin trước mẫu tin hiện thời
```
 private void butPre_Click(object sender, EventArgs e)
    {
       this.BindingContext[ds, "khoa"].Position--;
    }
```
+ **Sự kiện**: butNext_Click: Di chuyển con trỏ đến mẫu tin kế tiếp
```
private void butNext_Click(object sender, EventArgs e)
    {
       this.BindingContext[ds, "khoa"].Position++;
    }
```
+ **Sự kiện**: butLast_Click: Di chuyển con trỏ về mẫu tin cuối cùng
```
    private void butLast_Click(object sender, EventArgs e)
    {
       int ViTriMauTinCuoiCung = this.BindingContext[ds, "khoa"].Count - 1;
       this.BindingContext[ds, "khoa"].Position = ViTriMauTinCuoiCung;
    }
```
+ **Sự kiện**: butBosung_Click: Tạo mới một dòng
```
private void butBosung_Click(object sender, EventArgs e)
    {
       this.BindingContext[ds, "khoa"].AddNew();
    }
```
+ **Sự kiện**: butLuu_Click: Di chuyển con trỏ về mẫu tin cuối cùng, nếu có thay đổi trong DataSet ds thì cập nhật lại dữ liệu, việc cập nhật nhờ vào SqlCommandBuilder cb. Các thao tác bổ sung và xóa chỉ được cập nhật vào cơ sở dư liệu khi người sử dụng kích chuột vào nút Lưu
```
 private void butLuu_Click(object sender, EventArgs e)
    {
       this.BindingContext[ds, "khoa"].EndCurrentEdit();
       if (ds.HasChanges() == true)
       {
          try
          {
             dakhoa.Update(ds, "khoa");
             MessageBox.Show("Da cap nhat");
          }
          catch (Exception ll) { MessageBox.Show(ll.Message); }
       }
    }
```
+ **Sự kiện**: butXoa_Click: Lấy vị trí của con trỏ hiện thời, sau đó xóa đi mẫu tin này.

```
private void butXoa_Click(object sender, EventArgs e)
      {
         int donghientai;
         donghientai = this.BindingContext[ds, "khoa"].Position;
         this.BindingContext[ds, "khoa"].RemoveAt(donghientai);
      }
```

Bài thực hành về đặt quan hệ giữa các bảng, DataSet và DataBinding

Ví dụ này sử dụng cơ sở dữ liệu **quanlythuvien** như trong bài thực hành 4.3.4. Ví dụ này liên quan đến hai bảng dữ liệu: Khoa và docgia

Thiết kế form **frmKhoa_Docgia** để nhập, xóa, lưu, phục hồi và duyệt qua các mẫu tin trong bảng docgia cho từng khoa như sau:



frmkhoa sử dụng các trường, phương thức và sự kiện sau:

```
namespace baimau
{ public partial class frmKhoa_Docgia : Form
    {   SqlConnection cn = new SqlConnection("Data Source=nhha;Initial Catalog=" +
          "quanlythuvien;Trusted_Connection=yes");
        SqlCommand cmdkhoa = new SqlCommand();SqlCommand cmddocgia = new SqlCommand();
        SqlDataAdapter dakhoa = new SqlDataAdapter();
        SqlDataAdapter dadocgia = new SqlDataAdapter();
        DataSet ds = new DataSet();SqlCommandBuilder cb;
        public frmKhoa_Docgia()...
        private void Datquanhe(string bangchinh, string bangphu,
            string khoachinh, string khoaphu,string tenquanhe)...
        private void BuocCacDieuKhien()...
        private void frmKhoa_Docgia_Load(object sender, EventArgs e)...
        private void butLuu_Click(object sender, EventArgs e)...
        private void butXoa_Click(object sender, EventArgs e)...
        private void butBosung_Click(object sender, EventArgs e)...
        private void butFirst_Click(object sender, EventArgs e)...
        private void butLast_Click(object sender, EventArgs e)...
        private void butPre_Click(object sender, EventArgs e)...
        private void butNext_Click(object sender, EventArgs e)...
        private void butPhuchoi_Click(object sender, EventArgs e)...
    }
}
```

Các điều khiển

Tên điều khiển	**Thuộc tính**
ListBox	**Name:** lstKhoa
Form	**Name**: **frmKhoa_Docgia** **Text**: Khoa và độc giả
Label	Tạo ra 6 lable với các **Text**: Mã độc giả, Họ và tên, Ngày sinh, Địa chỉ, Ngày lập thẻ và Mã khoa
TextBox	Tạo ra 6 TextBox với các Name: txtMadocgia, txtHoten, txtNgaysinh, txtDiachi, txtNgaylapthe, txtMakhoa
Button	Tạo ra 8 Button với các Name: butBosung, butLuu, butXoa, butPhuchoi, butFirst (|<), butPre (<<), butNext (>>), butLast (>|)
dataGridView	**Name**:

Các phương thức

+ Hàm dựng frmKhoa_Docgia() để tạo giao diện:

```
public frmKhoa_Docgia()
    {    InitializeComponent();  }
```

+ Phương thức Datquanhe bao gồm các tham số: bảng chính, bảng phụ, khóa chính, khóa

phụ và tên quan hệ. Phương thức này nạp dữ liệu của 2 bảng: bảng chính và bảng phụ vào DataSet DataSet ds, sau đó đặt quan hệ giữa 2 bảng trong DataSet DataSet ds.

```
private void Datquanhe(string bangchinh, string bangphu,
        string khoachinh, string khoaphu,string tenquanhe)
    {
        cn.Open();
        cmdkhoa = new SqlCommand ("select * from "+ bangchinh, cn);
        dakhoa = new SqlDataAdapter(cmdkhoa);
        cmddocgia = new SqlCommand("select * from " + bangphu, cn);
        dadocgia = new SqlDataAdapter(cmddocgia);
        ds = new DataSet();
        dakhoa.Fill(ds, bangchinh);
        dadocgia.Fill(ds, bangphu);
        DataColumn chinh=ds.Tables[bangchinh].Columns[khoachinh];
        DataColumn phu=ds.Tables[bangphu].Columns[khoaphu];
        DataRelation r = new DataRelation(tenquanhe,chinh ,phu );
        ds.Relations.Add(r);
     }
```

+ Phương thức BuocCacDieuKhien(): Buộc dữ liệu vào lstKhoa, dataGridView1 và các textBox

```
private void BuocCacDieuKhien()
    {
        lstKhoa.DataSource = ds;
        lstKhoa.DisplayMember = "khoa.tenkhoa";
        dataGridView1.DataSource = ds;
        dataGridView1.DataMember = "khoa.khoa_docgia";
       //khoa_docgia là tên quan hệ của 2 bảng khoa và docgia trong DataSet ds
        txtMadocgia.DataBindings.Add("Text", ds, "khoa.khoa_docgia.madocgia");
        txtHoten.DataBindings.Add("Text", ds, "khoa.khoa_docgia.hoten");
        txtNgaysinh.DataBindings.Add("Text", ds, "khoa.khoa_docgia.ngaysinh");
        txtdiachi.DataBindings.Add("Text", ds, "khoa.khoa_docgia.diachi");
        txtNgaylapthe.DataBindings.Add("Text", ds, "khoa.khoa_docgia.ngaylapthe");
        txtMakhoa.DataBindings.Add("Text", ds, "khoa.khoa_docgia.makhoa");
    }
```

+ **Sự kiện** frmKhoa_Docgia_Load: Đặt quan hệ giữa 2 bảng khoa và docgia trong DataSet DataSet ds, tạo ra 1 SqlCommandBuilder để quản lý việc lưu dữ liệu vào cơ sở dữ liệu ,buộc dữ liệu vào các điều khiển trên form:

```
private void frmKhoa_Docgia_Load(object sender, EventArgs e)
{
    Datquanhe("khoa", "docgia", "makhoa", "makhoa", "khoa_docgia");
    cb = new SqlCommandBuilder(dadocgia);
```

```
        BuocCacDieuKhien();
    }
```

+ **Sự kiện**: butFirst_Click: Di chuyển con trỏ về mẫu tin đầu tiên

```
private void butFirst_Click(object sender, EventArgs e)
        {
            this.BindingContext[ds, "khoa.khoa_docgia"].Position = 0;;
        }
```

+ **Sự kiện**: butPre_Click: Di chuyển con trỏ về mẫu tin trước mẫu tin hiện thời

```
  private void butPre_Click(object sender, EventArgs e)
        {
            this.BindingContext[ds, "khoa.khoa_docgia"].Position--;
        }
```

+ **Sự kiện**: butNext_Click: Di chuyển con trỏ đến mẫu tin kế tiếp

```
private void butNext_Click(object sender, EventArgs e)
        {
            this.BindingContext[ds, "khoa.khoa_docgia"].Position++;
        }
```

+ **Sự kiện**: butLast_Click: Di chuyển con trỏ về mẫu tin cuối cùng

```
        private void butLast_Click(object sender, EventArgs e)
        {
            int ViTri = this.BindingContext[ds"khoa.khoa_docgia"].Count - 1;
            this.BindingContext[ds, "khoa"].Position = ViTri;
        }
```

+ **Sự kiện**: butBosung_Click: Tạo mới một dòng

```
private void butBosung_Click(object sender, EventArgs e)
        {
            this.BindingContext[ds, "khoa.khoa_docgia"].].AddNew();
        }
```

+ **Sự kiện**: butLuu_Click: Di chuyển con trỏ về mẫu tin cuối cùng, nếu có thay đổi trong DataSet ds thì cập nhật lại dữ liệu, việc cập nhật nhờ vào SqlCommandBuilder cb. Các thao tác bổ sung và xóa chỉ được cập nhật vào cơ sở dư liệu khi người sử dụng kích chuột vào nút Lưu

```
  private void butLuu_Click(object sender, EventArgs e)
        {
            this.BindingContext[ds, "khoa.khoa_docgia"].EndCurrentEdit();
            if (ds.HasChanges() == true)
            {
                try
                {
                    dakhoa.Update(ds, "docgia");
```

```
            MessageBox.Show("Da cap nhat");
        }
        catch (Exception ll) { MessageBox.Show(ll.Message); }
    }
}
```

+ **Sự kiện**: butXoa_Click: Lấy vị trí của con trỏ hiện thời, sau đó xóa đi mẫu tin này.

```
private void butXoa_Click(object sender, EventArgs e)
    {
        int donghientai;
        donghientai = this.BindingContext[ds, "khoa.khoa_docgia"].Position;
        this.BindingContext[ds, "khoa.khoa_docgia"].RemoveAt(donghientai);
    }
```

+ **Sự kiện**: buttPhuchoi_Click: Phục hồi lại các thao tác Bổ sung và xóa, dữ liệu chỉ được phục hồi khi chưa lưu vào cơ sở dữ liệu

```
private void buttPhuchoi_Click(object sender, EventArgs e)
    {
        this.BindingContext[ds, "khoa.khoa_docgia"].CancelCurrentEdit();
        ds.RejectChanges();
    }
```

4.6 Lựa chọn giữa mô hình Kết nối và mô hình Ngắt kết nối

DataReader và DataSet đưa ra hai cách tiếp cận khác nhau để xử lý dữ liệu. DataReader cho phép truy xuất kiểu forward-only, read-only. Bằng cách xử lý từng dòng một, cách tiếp cận này giúp tiết kiệm bộ nhớ. DataSet, ngược lại, cho phép truy xuất theo kiểu read/write, nhưng lại yêu cầu phải có đủ bộ nhớ để quản lý bản sao dữ liệu nạp về từ data source. Như vậy, bạn có thể suy ra một số quy tắc chung: Nếu ứng dụng không cần tính năng cập nhật dữ liệu và hầu như chỉ hiển thị và chọn lọc dữ liệu, DataReader là lựa chọn thích hợp; nếu ứng dụng cần cập nhật dữ liệu, giải pháp sử dụng DataSet nên được xem xét.

Tất nhiên, cũng có một số tình huống đi ngược lại với các quy tắc chung nói trên. Chẳng hạn, nếu data source chứa một số lượng lớn các record, khi đó DataSet phải yêu cầu quá nhiều tài nguyên bộ nhớ; hoặc nếu dữ liệu chỉ yêu cầu một vài thao tác cập nhật, thì sự kết hợp giữa DataReader và Command để thực thi cập nhật sẽ có thể có ý nghĩa hơn.

Tóm lại, một DataSet là một lựa chọn tốt khi:

- Dữ liệu cần được serialize (tuần tự hóa) và/hoặc gửi đi bằng HTTP.
- Các điều khiển read-only trên Form Win Form được kết buộc (bind) với data source.
- Một điều khiển Win Form như GridView hay DataView được kết buộc với một data source có khả năng cập nhật được.
- Một ứng dụng desktop cần thêm, xóa, sửa các dòng dữ liệu.

Trong khi đó, DataReader là lựa chọn cho những trường hợp:

- Cần quản lý một số lượng lớn các record, lớn đến mức mà bộ nhớ và thời gian để nạp dữ liệu cho DataSet là phi thực tế.
- Dữ liệu là read-only và được kết buộc với một điều khiển loại danh sách (list control) của Win Form hoặc Web Form.
- CSDL là không ổn định và thay đổi thường xuyên.

CHƯƠNG 5. XÂY DỰNG ỨNG DỤNG VỚI WEB VỚI WEBFORM

Công nghệ .NET được dùng để xây dựng các ứng dụng Web là ASP.NET, nó cung cấp hai vùng tên khá mạnh và đầy đủ phục vụ cho việc tạo các ứng dụng Web là **System.Web** và **System.Web.UI**. Trong chương này chúng ta sẽ tập trung chủ yếu vào việc dùng ngôn ngữ C# để lập trình với ASP.NET.

Bộ công cụ Web Form cũng được thiết kế để hỗ trợ mô hình phát triển nhanh (RAD). Với Web Form, ta có thể kéo thả các điều khiển trên Form thiết kế cũng như có thể viết mã trực tiếp trong tập tin **.aspx** hay **.aspx.cs**. Ứng dụng Web sẽ được triển khai trên máy chủ, còn người dùng sẽ tương tác với ứng dụng thông qua trình duyệt. .NET còn hỗ trợ ta bộ cung cụ để tạo ra các ứng dụng tuân theo mô hình n - lớp (tầng - n tier), giúp ta có thể quản lý được ứng dụng được dễ dàng hơn và nhờ thế nâng cao hiệu suất phát triển phần mềm.

5.1 Tìm hiểu về Web Forms

Web Form là bộ công cụ cho phép thực thi các ứng dụng mà các trang Web do nó tạo động ra được phân phối đến trình duyệt thông qua mạng Internet. Với Web Forms, ta tạo ra các trang HTML với nội dung tĩnh và dùng mã C# chạy trên Server để xử lý dữ liệu tĩnh này rồi tạo ra trang Web động, gửi trang này về trình duyệt dưới mã HTML chuẩn. Web Forms được thiết để chạy trên bất kỳ trình duyệt nào, trang HTML gửi về sẽ được định dạng sao cho thích hợp với phiên bản của trình duyệt. Ngoài ngôn ngữ C#, ta cũng có thể dùng ngôn ngữ VB.NET để tạo ra các ứng dụng Web tương tự. Web Forms chia giao diện người dùng thành hai phần: phần thấy trực quan (hay *UI*) và phần trang mã phía sau của UI. Quan điểm này thì tương tự với Windows Form, nhưng với Web Forms, hai phần này nằm trên hai tập tin riêng biệt. Phần giao diện UI được lưu trữ trong tập tin có phần mở rộng là .aspx, còn mã được lưu trữ trong tập tin có phần mở rộng là .aspx.cs. Với môi trường làm việc được cung cấp bởi bộ Visual Studio .NET, tạo các ứng dụng Web đơn giản chỉ là mở Form mới, kéo thả và viết mã quản lý sự kiện thích hợp. Web Forms được tích hợp thêm một loạt các điều khiển thực thi trên Server, có thể tự kiểm tra sự hợp lệ của dữ liệu ngay trên máy khách mà ta không phải viết mã mô tả gì cả.

5.2 Các sự kiện của Web Forms

Một sự kiện (Events) được tạo ra khi người dùng nhấn chọn một Button, chọn một mục trong ListBox hay thực hiện một thao tác nào đó trên UI. Các sự kiện cũng có thể được

phát sinh hệ thống bắt đầu hay kết thúc. Phương thức đáp ứng sự kiện gọi là trình quản lý sự kiện, các trình quản lý sự kiện này được viết bằng mã C# trong trang mã (code-behind) và kết hợp với các thuộc tính của các điều khiển thuộc trang.

Trình quản lý sự kiện là một "Delegate", phương thức này sẽ trả về kiểu void, và có hai đối số. Đối số đầu tiên là thể hiện của đối tượng phát sinh ra sự kiện, đối số thứ hai là đối tượng EventArg hay một đối tượng khác được dẫn xuất từ đối tượng EventArgs. Các sự kiện này được quản lý trên Server.

5.2.1 Sự kiện PostBack và Non-PostBack

PostBack là sự kiện sẽ khiến Form được gửi về Server ngay lập tức, chẳng hạn sự kiện đệ trình một Form với phương thức Post. Đối lập với PostBack là Non- PostBack, sự kiện này không gửi Form nên Server mà nó lưu sự kiện trên vùng nhớ Cache cho tới khi có một sự kiện PostBack nữa xảy ra. Khi một điều khiển có thuộc tính AutoPostBack là true thì sự kiện PostBack sẽ có tác dụng trên điều khiển đó:mặc nhiên thuộc tính AutoPostBach của điều khiển DropDownList là false, ta phải đặt lại là true thì sự kiện chọn một mục khác trong DropDownList này mới có tác dụng.

5.2.2 Trạng thái của ứng dụng Web (State)

Trạng thái của ứng dụng Web là giá trị hiện hành của các điều khiển và mọi biến trong phiên làm việc hiện hành của người dùng. Web là môi trường không trạng thái, nghĩa là mỗi sự kiện Post lên Server đều làm mất đi mọi thông tin về phiên làm việc trước đó. Tuy nhiên ASP.NET đã cung cấp cơ chế hỗ trợ việc duy trì trạng thái về phiên của người dùng. Bất kỳ trang nào khi được gửi lên máy chủ Server đều được máy chủ tổng hợp thông tin và tái tạo lại sau đó mới gửi xuống trình duyệt cho máy khách. ASP.NET cung cấp một cơ chế giúp duy trì trạng thái của các điều khiển phía máy chủ (Server Control) một cách tự động. Vì thế nếu ta cung cấp cho người dùng một danh sách dữ liệu ListBox, và người dùng thực hiện việc chọn lựa trên ListBox này, sự kiện chọn lựa này sẽ vẫn được duy trì sau khi trang được gửi lên máy chủ và gửi về cho trình duyệt cho máy khách.

5.2.3 Chu trình sống của một Web-Form

Khi có yêu cầu một trang Web trên máy chủ Web sẽ tạo ra một chuỗi các sự kiện ở máy chủ đó, từ lúc bắt đầu cho đến lúc kết thúc một yêu cầu sẽ hình thành một chu trình sống (Life-Cycle) cho trang Web và các thành phần thuộc nó. Khi một trang Web được yêu cầu, máy chủ sẽ tiến hành mở (Load) nó và khi hoàn tất yêu cầu máy chủ sẽ đóng trang này lại, kết xuất của yêu cầu này là một trang HTML tương ứng sẽ được gửi về cho trình duyệt. Dưới đây sẽ liệt kê một số sự kiện, ta có thể bắt các sự kiện để xử lý thích hợp hay

bỏ qua để ASP.NET xử lý mặc định.

Khởi tạo (*Initialize*) Là sự kiện đầu tiên trong chu trình sống của trang, ta có thể khởi tạo bất kỳ các thông số cho trang hay các điều khiển thuộc trang.

Mở trạng thái vùng quan sát (*Load View State*) Được gọi khi thuộc tính **ViewState** của điều khiển được công bố hay gọi. Các giá trị trong ViewState sẽ được lưu trữ trong một biến ẩn (Hidden Field), ta có thể lấy giá trị này thông qua hàm LoadViewState() hay lấy trực tiếp. **Kết thúc (Dispose)** Ta có thể dùng sự kiện này để giải phóng bất kỳ tài nguyên nguyên nào: bộ nhớ hay hủy bỏ các kết nối đến cơ sở dữ liệu.

Ví dụ: Hiển thị chuỗi lên trang

Đầu tiên ta cần chạy Visual Studio .NET, sau đó tạo một dự án mới kiểu WebApplication, ngôn ngữ chọn là C# và ứng dụng sẽ có tên là *ProgrammingCSharpWeb*.Url mặc nhiên của ứng dụng sẽ có tên là *http://localhost/ ProgrammingCSharpWeb.*

Visual Studio .NET sẽ đặt hầu hết các tập tin nó tạo ra cho ứng dụng trong thư mụcWeb mặc định trên máy người dùng, ví dụ: *D:\Inetpub\wwwroot\ProgrammingCSharpWeb.* Trong .NET, một giải pháp (Solution) có một hay hiều dự án (Project), mỗi dự án sẽ tạo ra một thư viện liên kết động (DLL) hay tập tin thực thi (EXE). Để có thể chạy được ứng dụng Web Form, ta cần phải cài đặt IIS và FrontPage Server Extension trên máy tính.

Khi ứng dụng Web Form được tạo, .NET tạo sẵn một số tập tin và một trang Web có tên mặc định là WebForm1.aspx chỉ chứa mã HTML và **WebForm1.cs** chứa mã quản lý trang. Trang mã **.cs** không nằm trong cửa sổ Solution Explorer, để hiển thị nó ta chọn Project\Show All Files, ta chỉ cần nhấn đúp chuột trái trên trang Web là cửa sổ soạn thảo mã (Editor) sẽ hiện nên, cho phép ta viết mã quản lý trang. Để chuyển từ cửa số thiết kế kéo thả sang cửa sổ mã HTML của trang, ta chọn hai Tab ở góc bên trái phía dưới màn hình.

Đặt tên lại cho trang Web bằng cách nhấn chuột phải lên trang và chọn mục **Rename** để đổi tên trang thành **HelloWeb.aspx**, .NET cũng sẽ tự động đổi tên trang mã của trang thành **HelloWeb.cs**.

.NET đã phát sinh ra một số mã ASP.NET:

<%@ Page language="c#"

Codebehind="HelloWeb.cs"

AutoEventWireup="false"

Inherits="ProgrammingCSharpWeb.WebForm1" %>

Thuộc tính **language** chỉ ra ngôn ngữ lập trình được dùng trong trang mã để quản lý trang, ở đây là C#. **Codebehide** xác định trang mã quản lý có tên **HelloWeb.cs** và thuộc tính **Inherits** chỉ trang Web được thừa kế từ lớp **WebForm1** được viết trong **HelloWeb.cs:**

public class WebForm1: System.Web.UI.Page

Ta thấy trang này được thừa kế từ lớp **System.Web.UI.Page**, lớp này do ASP.NET cung cấp, xác định các thuộc tính, phương thức và các sự kiện chung cho các trang phía máy chủ. Mã HTML phát sinh định dạng thuộc tính của Form:

<form id="Form1" method="post" runat="server">

Thuộc tính **id** làm định danh cho Form, thuộc tính **method** có giá trị là "**POST**" nghĩa là Form sẽ được gởi lên máy chủ ngay lập tức khi nhận một sự kiện do người dùng phát ra (như sự kiện nhấn nút) và cờ **IsPostBack** trên máy chủ khi đó sẽ có giá trị là true. Biến cờ này có giá trị là false nếu Form được đệ trình với phương thức "**GET**" hay lần đầu tiên trang được gọi. Bất kỳ điều khiển nào hay Form có thuộc tính **runat="server"** thì điều khiển hay Form này sẽ được xử lý bởi ASP.NET Framework trên máy chủ. Thuộc tính MS_POSITIONING ="GridLayout" trong thẻ <Body>, cho biết cách bố trí các điều khiển trên Form theo dạng lưới.

Hiện giờ Form của ta là trống, để hiển thị một chuỗi gì đó lên màn hình, ta gõ dòng mã sau trong thẻ **<body>:**

Hello World! It is now <% = DateTime.Now.ToString( ) %>

Giống với ASP, phần nằm trong dấu **<% %>** được xem như là mã quản lý cho trang, ở đây là mã C#. Dấu **=** chỉ ra một giá trị nhận được từ một biến hay một đối tượng nào đó, ta cũng có thể viết mã trên lại như sau với cùng chức năng:

Hello World! It is now

<% Response.Write(DateTime.Now.ToString( )); %>

Thực thi trang này (Ctrl-F5), kết quả sẽ hiện trên trình duyệt như sau:

Hello World! It is now 26/02/08 9:28:56 AM

Để thêm các điều khiển cho trang, hoặc là ta có thể viết mã trong của sổ HTML hoặc là

kéo thả các điều khiển trên bộ công cụ của Web Form vào cửa sổ thiết kế trang. ASP.NET sẽ tự động phát sinh ra kết quả từ mã HTML thành các điều khiển cũng như từ các điều khiển trên trang thiết thành mã HTML tương ứng.

Các điều khiển của ASP.NET, có thêm chữ "asp:" phía trước tên của điều khiển đó, được thiết kế mang tính hướng đối tượng nhiều hơn. Ví dụ:

<asp:RadioButton>

<asp:CheckBox>

<asp:Button>

<asp:TextBox rows="1">

<asp:TextBox rows="5">

Ngoài các điều khiển của ASP.NET, các điều khiển HTML chuẩn cũng được ASP.NET hỗ trợ. Tuy nhiên các điều khiển không tạo sự dễ đọc trong mã nguồn do tính đối tượng trên chúng không rõ ràng, các điều khiển HTML chuẩn ứng với năm điều khiển trên là:

<input type = "radio">

<input type="checkbox">

<input type="button">

<input type="text">

<textarea>

5.3 Một số ví dụ mẫu minh họa

Một cách thuận tiện nhất để học một công nghệ mới chính là dựa vào các ví dụ, vì vậy trong phần này chúng ta sẽ khảo sát một vài ví dụ để minh họa cho phần lý thuyết của chúng ta. Như ta đã biết, ta có thể viết mã quản lý theo hai cách: hoặc là viết trong tập tin **.cs** hoặc là viết trực tiếp trong trang chứa mã HTML. Trên ví dụ 1, ta sẽ viết mã quản lý trực tiếp trên trang HTML.

5.3.1 Ví dụ 1: Kết buộc dữ liệu không thông qua thuộc tính DataSource

Ứng dụng của chúng ta đơn giản chỉ hiện lên trang tên khách hàng và số hóa đơn bằng cách dùng hàm DataBind(). Hàm này sẽ kết buộc dữ liệu của mọi thuộc tính hay của bất kỳ đối tượng.

<html>

```
<head>
// mã quản lý C# sẽ được viết trong thẻ <script> này
<script language="C#" runat="server">
// trang sẽ gọi hàm này đầu tiên, ta sẽ thực hiện kết buộc
// trực tiếp trong hàm này
void Page_Load(Object sender, EventArgs e) {
        Page.DataBind();
  }
// lấy giá trị của thuộc tính thông qua thuộc tính // get
string custID{
get {
return "ABCD";
}
}
int orderCount{
get {
return 11;
}
}
</script>
</head>
<body>
<h3><font face="Verdana"> Ket buoc khong dung DataSource
</font></h3>
<form runat=server>
Khach hang: <b><%# custID %></b><br>
```

So hoa don: <%# **orderCount** %>

</form>

</body>

</html>

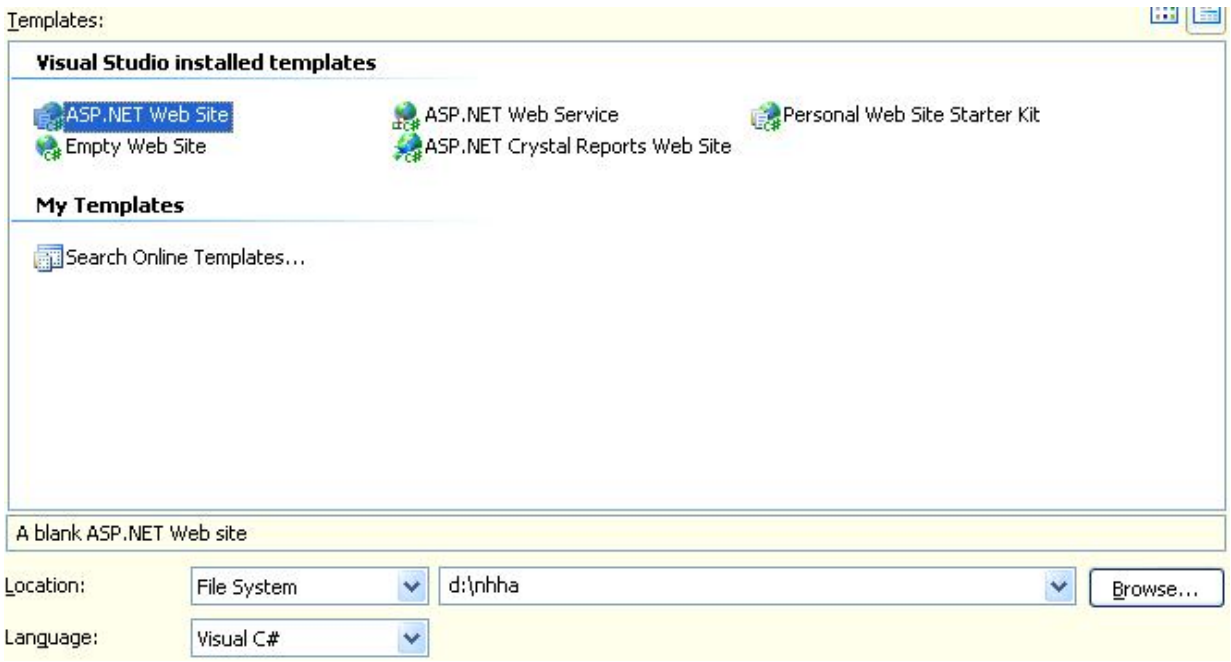
5.3.2 Ví dụ 2.

Yêu cầu: tạo 1 Web site cho sinh viên đặt chỗ để mượn sách trên thư viện bao gồm các chức năng đăng nhập, sửa đổi tài khoản, hiển thị các thông tin về độc giả, hiển thị, tìm kiếm và mượn sách với giao diện như sau:

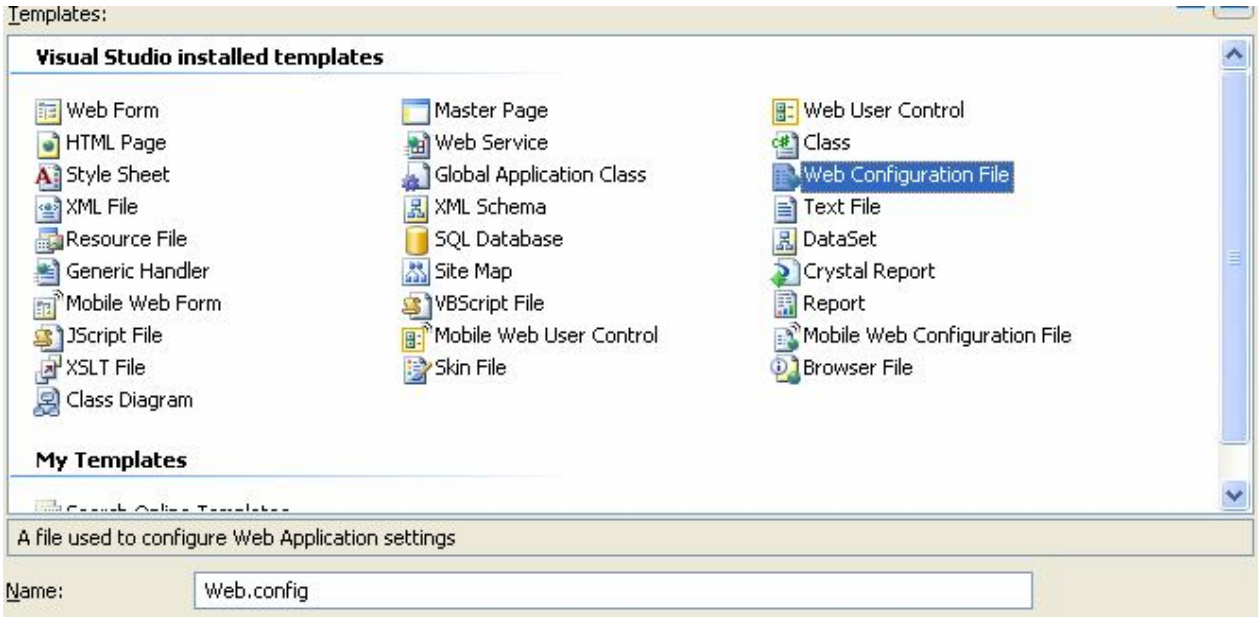
+ Khi độc giả chọn được sách cần mượn thì độc giả phải đến thư viện gặp người quản thư để lấy sách.

+Ví dụ này sử dụng cơ sở dữ liệu quanlythuvien ở bài thực hành 4.3.4.

+ Tạo Project có tên là thuvien: File|New|Web site chọn ASP.NET Web Site



+ Tạo file cấu hình: Web site | Add New Item chọn Web Configuration File



+ Tạo đường kết nối đến cơ sở dữ liệu quanlythuvien

Mở file Web.config và sửa lại:

```
<?xml version="1.0"?>
...
<configuration>
  <connectionStrings>
    <add name="thuvien" connectionString=
      "server=(local);Integrated Security=true;database=quanlythuvien" />
  </connectionStrings>
  <system.web>...
</configuration>
```

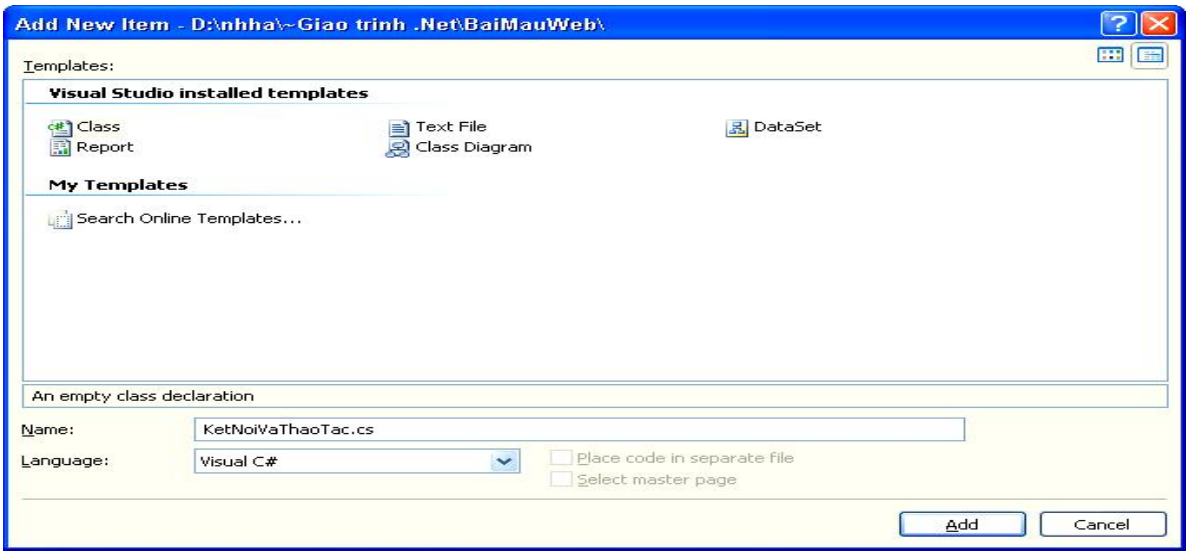
5.3.1 Xây dựng các lớp dùng chung.

Để sử dụng lại các thành phần trong nhiều dự án ta tiến hành xây dựng các lớp dùng chung trong các ứng dụng ASP.NET với cơ sở dữ liệu SQL SERVER như sau:

5.3.1.1 Xây dựng lớp dùng chung: KetNoiVaThaoTac.cs

+ Để kết nối đến cơ sở dữ liệu, thực thi các tác vụ trên dữ liệu và truy vấn dữ liệu:

+ Kích chuột phải lên App_Code ở cửa sổ Solution Explored | Add New Item| Class và gõ tên lớp:



+ Khi lập trình Web sử dụng cở sở dữ liệu SQL Server ta sử dụng các thư viện sau (các thư viện này dùng chung cho tất cả các lớp trong ví dụ này):

```
using System;
using System.Data;
using System.Configuration;
using System.Web;
using System.Web.Security;
```

```
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;
using System.Data.SqlClient;
using System.Web.Configuration;
```

+ Trong lớp KetNoiVaThaoTac ta sử dụng các trường, thuộc tính và các phương thức dùng chung như sau:

```
public class KetNoiVaThaoTac
{
    private string ChuoiKetNoi;
    private SqlConnection cn;
    private SqlCommand cmd;
    private SqlDataAdapter da;
    private string commandText = "";
    private string[] parameterCollection;
    private CommandType commandType;
    private string[] valueCollection;
    public KetNoiVaThaoTac(string ChuoiKetNoi)...
    public void MoDuongKetNoi()...
    public void DongKetNoi()...
    public string CommandText...
    public CommandType CommandType...
    public string[] ParameterCollection...
    public string[] ValueCollection...
    public void AddParameters(SqlCommand cmd)...
    public int ExecuteNonQuery()...
    public object ExecuteScalar()...
    public DataSet GetDataSet()...
    public DataTable GetDataTable()...
}
```

+ Khi khai báo và khởi tạo lớp KetNoiVaThaoTac ta có thể mở kết nối đến cơ sở dữ liệu:

```
public KetNoiVaThaoTac(string ChuoiKetNoi)
  {
    this.ChuoiKetNoi = ChuoiKetNoi;
    MoDuongKetNoi();
    }
```

Trong đó, phương thức `MoDuongKetNoi();`dùng để mở kết nối đến cơ sở dữ liệu SQL Server bằng cách dùng phương thức Open của đối tượng SqlConnection:

```
public void MoDuongKetNoi()
  {
    if (cn == null || cn.State == ConnectionState.Closed)
    {
      cn = new SqlConnection(ChuoiKetNoi);
      cn.Open();
    }
  }
```

+ Khi không có nhu cầu kết nối đến cơ sở dữ liệu, ta khai báo phương thức để đóng kết nối và giải phóng đối tượng SqlConnection:

```
public void DongKetNoi()
  {
    if (cn.State == ConnectionState.Open)
    {
      cn.Close();
    }
    cn.Dispose();
    }
```

+ Thuộc tính `CommandText` để người lập trình có thể truyền câu lệnh SQL hoặc tên thủ tục lưu trữ:

```
public string CommandText
  {
    set { commandText = value; }
    get { return commandText; }
  }
```

+ Để chỉ định người lập trình sử dụng câu lệnh SQL hay tên thủ tục lưu trữ ta khai báo

thuộc tính `CommandType` như sau:

```
public CommandType CommandType
  {
    set { commandType = value; }
    get { return commandType; }
  }
```

+ Nếu câu lệnh SQL hoặc thủ tục lưu trữ có tham số, ta khai báo thuộc tính `ParameterCollection` để truyền vào mảng các tham số như sau:

```
 public string[] ParameterCollection
  {
    set { parameterCollection = value; }
    get { return parameterCollection; }
  }
```

+ Khi truyền vào mảng các tham số thì người lập phải truyền mảng giá trị tương ứng thông qua thuộc tính `ValueCollection`:

```
public string[] ValueCollection
  {
    set { valueCollection = value; }
    get { return valueCollection; }
    }
```

+ Để khai báo tham số và giá trị cho tham số từ hai thuộc tính ParameterCollection, ValueCollection vừa khai báo ở trên, ta khai báo phương thức AddParameters

```
public void AddParameters(SqlCommand cmd)
  {
    for (int i = 0; i < parameterCollection.Length; i++)
    {
```

```
            cmd.Parameters.AddWithValue(
            parameterCollection[i], valueCollection[i]);
        }
    }
```

+ Trong ADO.NET phương thức thường để thực hiện câu lệnh SQL hoặc thủ tục lưu trữ là: `ExecuteNonQuery`. Ta có thể xây dựng lại phương thức `ExecuteNonQuery` như sau:

```
public int ExecuteNonQuery()
    {
        cmd = new SqlCommand();
        cmd.CommandText = commandText;
        cmd.Connection = cn;
        cmd.CommandType = commandType;
        if (ParameterCollection.Length > 0)
            AddParameters(cmd);
        int rows = cmd.ExecuteNonQuery();
        cmd.Dispose();
        return rows;// Trả về số mẫu tin thực thi
        }
```

+ Khi thực hiện câu lệnh SQL dạng hành động hay thủ tục lưu trữ và lấy ra giá trị dạng đối tượng, ta sử dụng phương thức `ExecuteScalar`, để thuận tiện cho việc sử dụng ta có thể xây dựng lại phương thức `ExecuteScalar` như sau:

```
 public object ExecuteScalar()
    {
        cmd = new SqlCommand();
        cmd.CommandText = commandText;
```

```
cmd.Connection = cn;
cmd.CommandType = commandType;
if (ParameterCollection != null)
   AddParameters(cmd);
object obj = cmd.ExecuteScalar();
cmd.Dispose();
return obj;
}
```

+ Nếu chúng ta cần lấy ra một tập dữ liệu gồm nhiều đối tượng DataTable, ta có thể sử dụng đối tượng DataSet :

```
public DataSet GetDataSet()
{
   cmd = new SqlCommand();
   cmd.CommandText = commandText;
   cmd.Connection = cn;
   cmd.CommandType = commandType;
   if (ParameterCollection != null)
      AddParameters(cmd);
   da = new SqlDataAdapter(cmd);
   DataSet dataSet = new DataSet();
   da.Fill(dataSet);
   cmd.Dispose();
   da.Dispose();
   return dataSet;
```

```
    }
```

+ Để lấy ra một tập dữ liệu, ta có thể sử dụng đối tượng DataTable:

```
public DataTable GetDataTable()
  {
    cmd = new SqlCommand();
    cmd.CommandText = commandText;
    cmd.Connection = cn;
    cmd.CommandType = commandType;
    if (ParameterCollection != null)
       AddParameters(cmd);
    da = new SqlDataAdapter(cmd);
    DataTable dataTable = new DataTable();
    da.Fill(dataTable);
    cmd.Dispose();
    da.Dispose();
    return dataTable;
    }
```

5.3.1.2 Xây dựng lớp dùng chung: DuaDulieuVaoDieuKhien.cs

+ Để nạp dữ liệu vào cho các điều khiển trong ví dụ này ta chỉ nạp dữ liệu vào cho hai điều khiển GridView và DataList:

+ Kích chuột phải lên App_Code ở cửa sổ Solution Explored | Add New Item| Class và gõ tên lớp: DuaDulieuVaoDieuKhien.cs

+ Trong lớp DuaDulieuVaoDieuKhien.cs ta sử dụng lại lớp KetNoiVaThaoTac bao gồm các trường, thuộc tính và các phương thức dùng chung như sau:

```
public class DuaDulieuVaoDieuKhien
{
    KetNoiVaThaoTac LopChung;
    public DuaDulieuVaoDieuKhien()...
    public DuaDulieuVaoDieuKhien(string cnstr)...
    private string commandText = "";
    public string CommandText...
    private CommandType commandType = CommandType.Text;
    public CommandType CommandType...
    private string[] parameterCollection;
    public string[] ParameterCollection...
    private string[] valueCollection;
    public string[] ValueCollection...
    public void KhoiTao()...
    public void KetThuc()...
    public int  DemDong()...
    public void NapDuLieuVaoGridView(GridView gridView)...
    public void NapDuLieuVaoGridView( GridView gridView, int pageSize,
    int pageCount, PagerButtons pagerButtons)...
    public int NapDuLieuVaoDataList(DataList dataList)...
    public void TaoLienKetTrongGridView(GridView GridView1, string url,
    int TruongHienThi, int TruongLienket, string Thongbao)...
}
```

+ Hàm dựng DuaDulieuVaoDieuKhien được dùng để tạo ra 1 lớp KetNoiVaThaoTac

```
public DuaDulieuVaoDieuKhien(string cnstr)
  {
    LopChung = new KetNoiVaThaoTac(cnstr);
    }
```

+ Tương tự như lớp KetNoiVaThaoTac ta xây dựng các thuộc tính sau:

```
private string commandText = "";
  public string CommandText
  {
    set { commandText = value; }
    get { return commandText; }
  }
```

```
private CommandType commandType = CommandType.Text;
public CommandType CommandType
{
   set { commandType = value; }
   get { return commandType; }
}
private string[] parameterCollection;
public string[] ParameterCollection
{
   set { parameterCollection = value; }
   get { return parameterCollection; }
}
private string[] valueCollection;
public string[] ValueCollection
{
   set { valueCollection = value; }
   get { return valueCollection; }
   }
```

+ Để gán các thuộc tính `CommandText`, `CommandType`, `ParameterCollection`, `ValueCollection` cho lớp KetNoiVaThaoTac ta khai báo phương thức KhoiTao() như sau:

```
public void KhoiTao()
{
   LopChung.CommandText = commandText;
```

```
        LopChung.CommandType = commandType;
        LopChung.ParameterCollection = parameterCollection;
        LopChung.ValueCollection = valueCollection;
    }
```

+ Để đóng kết nối ta dùng phương thức KetThuc() như sau:

```
public void KetThuc()
    {
        LopChung.DongKetNoi();
    }
```

+ Để lấy được kết quả thống kê ta xây dựng hàm:

```
public int  DemDong()
    {
        int totalRecord = 0;
        object obj = LopChung.ExecuteScalar();
        if (obj != null) totalRecord = (int)obj;
        return totalRecord;
    }
```

+ Hàm NapDuLieuVaoGridView được sử dụng để đưa dữ liệu vào điều khiển GridView

```
    public void NapDuLieuVaoGridView(GridView gridView)
    {
        DataTable dataTable =  LopChung.GetDataTable();
        if (dataTable != null)
        {
```

```
        gridView.DataSource = dataTable;
    }
    else
        gridView.DataSource = null;
    gridView.DataBind();
}
```

+ Trong trường cần phân trang, ta có thể nạp chồng hàm NapDuLieuVaoGridView như sau:

```
public void NapDuLieuVaoGridView( GridView gridView, int pageSize,
int pageCount, PagerButtons pagerButtons)
{

    gridView.AllowPaging = true;
    DataTable dataTable =  LopChung.GetDataTable();
    if (dataTable != null)
    {
        gridView.DataSource = dataTable;
        gridView.PageSize = pageSize;
        gridView.PagerSettings.PageButtonCount = pageCount;
        gridView.PagerSettings.Mode = pagerButtons;
    }
    else
        gridView.DataSource = null;
    gridView.DataBind();
```

```
}
```

+ Tương tự như GridView ta có thể nạp dữ liệu vào cho điều khiển DataList như sau:

```
public int NapDuLieuVaoDataList(DataList dataList)
{
    int totalRecord = 0;
    DataTable dataTable =  LopChung.GetDataTable();
    if (dataTable != null)
    {
        dataList.DataSource = dataTable;
        totalRecord = dataTable.Rows.Count;
    }
    else
        dataList.DataSource = null;
    dataList.DataBind();
    return totalRecord;
}
```

+ Phương thức TaoLienKetTrongGridView dùng để tạo liên kết trên cột thứ TruongHienThi, khi chọn liên kết sẽ mở url với tham số là giá trị trên cột thứ TruongHienThi, khi rê chuột lại liên kết sẽ xuất hiện Thongbao

```
public void TaoLienKetTrongGridView(GridView GridView1, string url,
  int TruongHienThi, int TruongLienket, string Thongbao)
{
    foreach (GridViewRow  rows in GridView1.Rows)
    {
```

```
        HyperLink hyperLink = new HyperLink();
        hyperLink.Text =rows.Cells[TruongHienThi].Text;
        hyperLink.NavigateUrl = url +  rows.Cells[TruongLienket].Text  ;
        hyperLink.ToolTip = Thongbao;
        rows.Cells[1].Controls.Add(hyperLink);
    }
    }
```

5.3.2 Xây dựng các lớp để thao tác trên các bảng: Docgia, sach và phieumuon.

5.3.2.1 Xây dựng lớp Docgia

+ Kích chuột phải lên App_Code ở cửa sổ Solution Explored | Add New Item| Class và gõ tên lớp: Docgia.cs

+ Trong lớp Docgia.cs sử dụng các trường và phương thức sau:

```
public class Docgia
{string cnstr = WebConfigurationManager.ConnectionStrings["thuvien"].ConnectionString;
    public Docgia()...
    public   int KiemDangNhap(string UserName, string PassWord)...
    public string[] GetDocGia(string madocgia)...
    public void HienThiThongTin(GridView gv,string madocgia)...
    public int DoiMatKhau(string madg,string MatKhauCu,string MatKhauMoi)...
}
```

Trong đó:

+ Trường cnstr được sử dụng để lấy chuỗi kết nối trong file cấu hình Web.config

```
string cnstr =
WebConfigurationManager.ConnectionStrings["thuvien"].ConnectionString;
```

+ Xây dựng hàm dựng Docgia() rỗng

+ Phương thức KiemDangNhap để kiểm tra xem mã độc giả và mật khẩu độc giả nhập có đúng không.

```
public  int KiemDangNhap(string UserName, string PassWord)
  {
    DuaDulieuVaoDieuKhien  dl = new DuaDulieuVaoDieuKhien(cnstr);
    dl.CommandText = "select count(*) from Docgia where madocgia=
```

```
                                        @madg and matkhau=@pwd";
    dl.CommandType = CommandType.Text;
    dl.ParameterCollection = new string[2] { "@madg","pwd" };
    dl.ValueCollection = new string[2] { UserName,PassWord};
    dl.KhoiTao();
    int kq= dl.DemDong() ;
    return kq;
  }
```

+ Phương thức GetDocGia đưa vào mã độc giả, trả về Họ tên và tên khoa của độc giả

```
  public string[] GetDocGia(string madocgia)
  {
    string[] st = new string[2];
    KetNoiVaThaoTac  dl = new KetNoiVaThaoTac(cnstr);
    dl.CommandText = "select madocgia,Hoten,tenkhoa from Docgia,khoa where"
                    +" Docgia.makhoa=khoa.makhoa and madocgia=@madg";
    dl.CommandType = CommandType.Text;
    dl.ParameterCollection = new string[1] { "@madg" };
    dl.ValueCollection = new string[1] { madocgia };
    dl.MoDuongKetNoi();
    DataTable dt= dl.GetDataTable();
    if (dt.Rows.Count   != 0)
    {
      st[0] = dt.Rows[0][1].ToString();
```

```
        st[1] = dt.Rows[0][2].ToString();
      }
      else
        st = null;
      return st;
    }
```

+ Phương thức HienThiThongTin đưa tất cả thông tin của độc giả có mã độc giả là madocgia vào 1 GridView theo thứ tự giảm dần theo ngày mượn

```
public void HienThiThongTin(GridView gv,string madocgia)
{
 DuaDulieuVaoDieuKhien dl = new DuaDulieuVaoDieuKhien(cnstr);
 dl.CommandText = "select Docgia.madocgia,hoten,tenkhoa, "+
       nhande,phieumuon.soluong, ngaymuon,trangthai from Docgia, sach, khoa, "+
       "phieumuon where khoa.makhoa=docgia.makhoa and docgia.madocgia=" +
       "phieumuon.madocgia and sach.masach=phieumuon.masach and"+
       "Docgia.madocgia=@madocgia order by ngaymuon DESC";
      dl.CommandType = CommandType.Text;
      dl.ParameterCollection = new string[1] { "@madocgia" };
      dl.ValueCollection = new string[1] { madocgia };
      dl.KhoiTao();
      dl.NapDuLieuVaoGridView(gv);
    }
```

+ Phương thức DoiMatKhau được sử dụng để đổi mật khẩu cũ thành mật khẩu mới cho độc giả có mã là madocgia

```
 public int DoiMatKhau(string madg,string MatKhauCu,string MatKhauMoi)
```

```
{
    if (KiemDangNhap(madg,MatKhauCu)==1)
    {
        KetNoiVaThaoTac dl = new KetNoiVaThaoTac(cnstr);
        dl.CommandText = "update docgia set matkhau=@matkhaumoi where
                matkhau=@matkhaucu and madocgia=@madg";
        dl.CommandType = CommandType.Text;
            dl.ParameterCollection = new string[3] {
                                        "@madg","@matkhaucu","@matkhaumoi" };
        dl.ValueCollection = new string[3] { madg,MatKhauCu,MatKhauMoi};
        dl.MoDuongKetNoi();
        return dl.ExecuteNonQuery();
    }
    else
        return 0;
    }
```

5.3.2.2 Xây dựng lớp Sach

+ Kích chuột phải lên App_Code ở cửa sổ Solution Explored | Add New Item| Class và gõ tên lớp: Sach.cs

+ Trong lớp Sach.cs sử dụng các trường và phương thức sau:

```
public class Sach
{
    string cnstr =
        WebConfigurationManager.ConnectionStrings["thuvien"].ConnectionString;
    public Sach()...
    public string[] GetSach(string masach)...
    public int GiamSoLuong(string masach)...
    public int TangSoLuong(string masach)...
}
```

Trong đó:

+ Trường cnstr giống như lớp Docgia
+ Hàm dựng Sach() rỗng
+ Phương thức Getsach trả về nhan đề (tên sách) và tác giả của sách có mã là masach

```
public string[] GetSach(string masach)
  {
    string[] st = new string[2];
    KetNoiVaThaoTac dl = new KetNoiVaThaoTac(cnstr);
    dl.CommandText =
            "select masach,nhande,tacgia from sach where masach=@masach";
    dl.CommandType = CommandType.Text;
    dl.ParameterCollection = new string[1] { "@masach" };
    dl.ValueCollection = new string[1] { masach  };
    dl.MoDuongKetNoi();
    DataTable dt = dl.GetDataTable();
    if (dt.Rows.Count != 0)
    {
      st[0] = dt.Rows[0][1].ToString();
      st[1] = dt.Rows[0][2].ToString();
    }
    else
      st = null;
    return st;
  }
```

+ Phương thức GiamSoLuong: giảm số lượng đi 1 đối với sách có mã là masach

```
public int GiamSoLuong(string masach)
{
  KetNoiVaThaoTac dl = new KetNoiVaThaoTac(cnstr);
  dl.CommandText = "update sach  set soluong=soluong-1 where
                                  masach=@masach and soluong>=1";
  dl.CommandType = CommandType.Text;
  dl.ParameterCollection = new string[1] { "@masach" };
  dl.ValueCollection = new string[1] { masach };
  dl.MoDuongKetNoi();
  return dl.ExecuteNonQuery() ;
  }
```

+ Phương thức TangSoLuong: tăng số lượng lên 1 đối với sách có mã là masach

```
public int TangSoLuong(string masach)
    {
       KetNoiVaThaoTac dl = new KetNoiVaThaoTac(cnstr);
       dl.CommandText =
     "update sach  set soluong=soluong+1 where masach=@masach";
       dl.CommandType = CommandType.Text;
       dl.ParameterCollection = new string[1] { "@masach" };
       dl.ValueCollection = new string[1] { masach };
       dl.MoDuongKetNoi();
       return dl.ExecuteNonQuery();
      }
```

5.3.2.3 Xây dựng lớp PhieuMuon

+ Kích chuột phải lên App_Code ở cửa sổ Solution Explored | Add New Item| Class và gõ tên lớp: PhieuMuon.cs

+ Trong lớp PhieuMuon.cs sử dụng các trường và phương thức sau:

```
public class PhieuMuon
{
    string cnstr =
        WebConfigurationManager.ConnectionStrings["thuvien"].ConnectionString;
    public PhieuMuon()...
    public int KiemTraMuon(string Madocgia)...
    public int CapNhat(string madg,DateTime ngaymuon,string masach,
    int sl,int trangthai )...
    public int HuyDatCho(string masach)...
}
```

Trong đó:

+ Trường cnstr giống như lớp Docgia

+ Hàm dựng PhieuMuon() rỗng

+ Phương thức KiemTraMuon kiểm tra xem độc giả có mã là madocgia đã trả sách hay chưa. Ta dựa vào trường trangthai trong bảng phieumuon: nếu trạng thai=0 thì thì độc giả này đã trả sách, trangthai=1 đã mượn sách trên Web nhưng chưa lấy sách, trangthai=2 đã lấy sách.

```
public int KiemTraMuon(string Madocgia)
  {
    DuaDulieuVaoDieuKhien dl = new DuaDulieuVaoDieuKhien(cnstr);
    dl.CommandText = "select count(*) from phieumuon where madocgia=@madg
                      and trangthai!=0";
    dl.CommandType = CommandType.Text;
    dl.ParameterCollection = new string[1] { "@madg"};
    dl.ValueCollection = new string[1] { Madocgia };
    dl.KhoiTao();
    return dl.DemDong();
    }
```

+ Phương thức Capnhap được sử dụng để đưa vào bảng phieumuon 1 mẫu tin

```
public int CapNhat(string madg,DateTime ngaymuon,string masach,
 int sl,int trangthai )
```

```
{
    Sach s = new Sach();
    if (s.GiamSoLuong(masach) == 1)
    {
      KetNoiVaThaoTac dl = new KetNoiVaThaoTac(cnstr);
      dl.CommandText = "insert into phieumuon(madocgia,ngaymuon,masach,
     soluong,trangthai)values(@madg,@ngaymuon,@masach,@sl,@trangthai)";
      dl.CommandType = CommandType.Text;
      dl.ParameterCollection = new string[5]
                        { "@madg", "@ngaymuon", "@masach", "@sl", "@trangthai" };
      dl.ValueCollection = new string[5] { madg,
      ngaymuon.ToShortDateString() , masach,sl.ToString(), trangthai.ToString() };
      dl.MoDuongKetNoi();
      return dl.ExecuteNonQuery() ;
    }
    else
      return 0;
     }
```

+ Phương thức HuyDangCho được sử dụng để độc giả hủy đi 1 cuốn sách đã đặt, mỗi độc giả tại 1 thời điểm chỉ mượn được 1 quyển sách, nên ta căn cứ vào trangthai của độc giả để xóa đi sách đã đặt (theo quy ước trangthai=0 đã trả hết sách, =1 đặt sách nhưng chưa lấy, =2 đã mượn sách). Sau khi hủy đặt chỗ ta phải tăng số lượng trong bảng Sach lên 1.

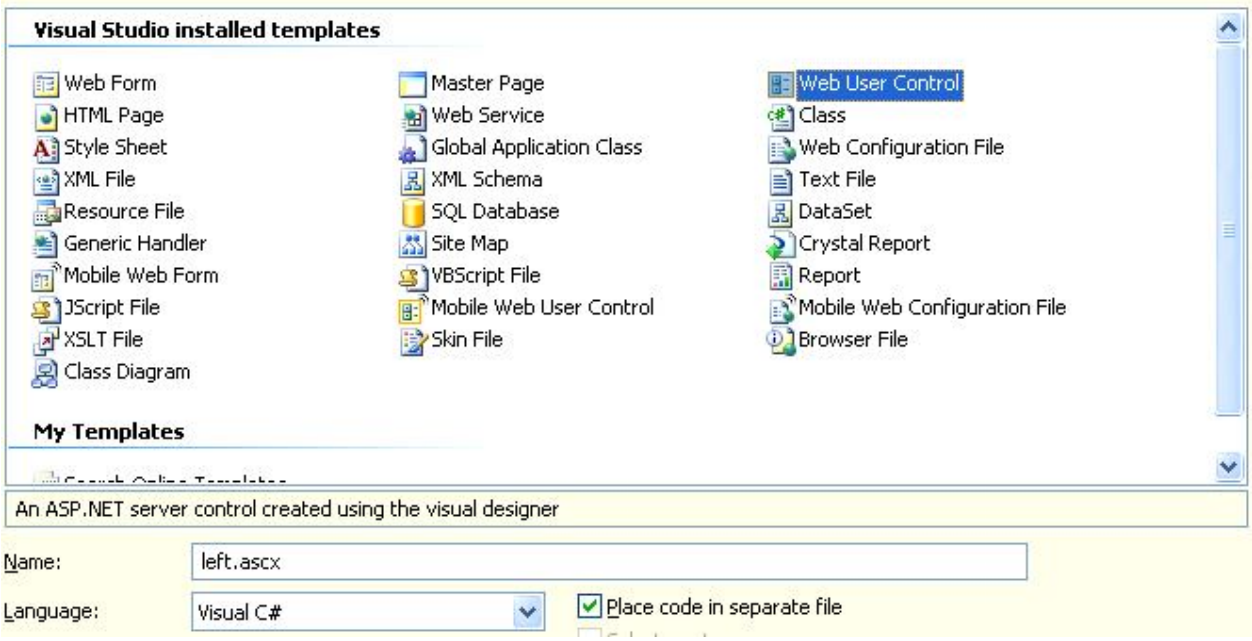
```
public int HuyDatCho(string masach)
    {  KetNoiVaThaoTac dl = new KetNoiVaThaoTac(cnstr);
        dl.CommandText = "delete phieumuon where trangthai=1";
        dl.CommandType = CommandType.Text;
        dl.ParameterCollection = null
```

```
        dl.ValueCollection = null
        dl.MoDuongKetNoi();
        int kq= dl.ExecuteNonQuery();
        if (kq == 1)
        {
            Sach s = new Sach();
            s.TangSoLuong(masach);
        }
        return kq;
    }
```

5.3.3 Xây dựng các WebUserControl.

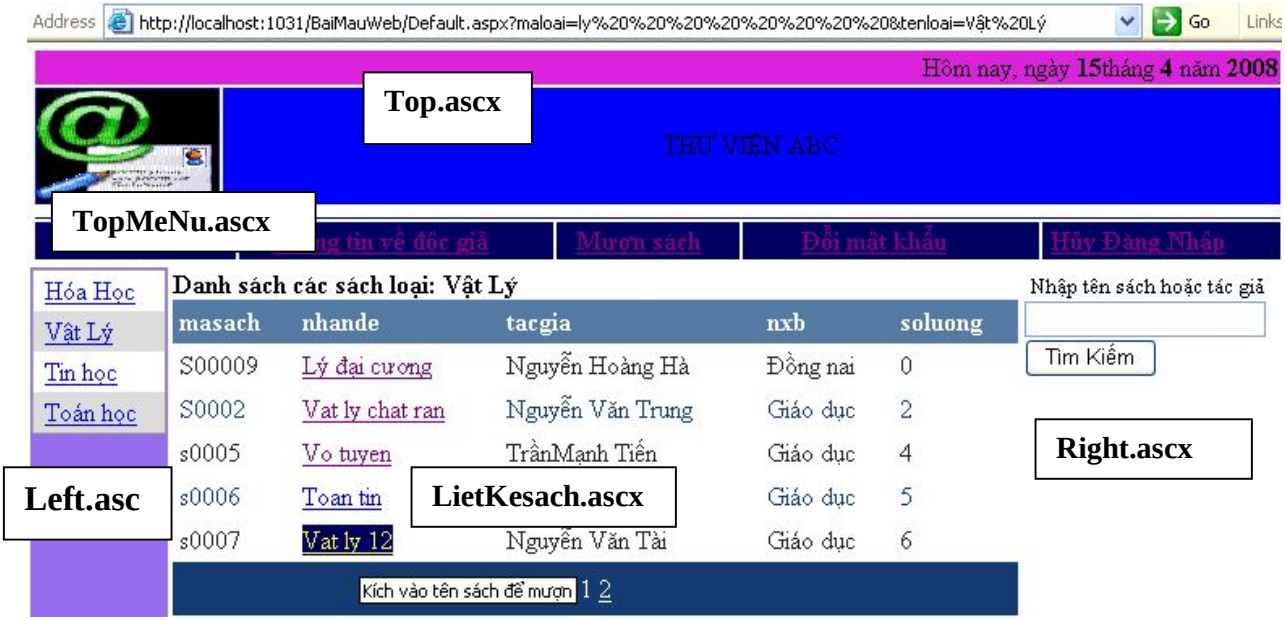
- UserControl là thành phần có sẵn của ASP.NET, cho phép ta thiết kế điều khiển dựa trên các kỹ thuật mà ta đã lập trình trên trang ASP.NET.
- Khi sử dụng UserControl, ta phải nhúng chúng vào trang ASP.NET thay vì thực thi một mình giống như trang ASP.NET, phần mở rộng của UserControl là ASCX.
- Sau khi thiết kế xong UserControl ta chọn chúng từ cửa sổ Solution Explored và kéo thả vào trang ASP.NET

+ Để để quản lý ta tạo một thư mục để lưu các WebUserControl:
Web Site | New Folder| gõ tên thư mục là UC
+ Tạo ra 1 WebUserControl: Kích chuột phải lên tên thư mục UC | Add New Item | Web User Control | gõ tên WebUserControl | Add:



Để xây dựng Web site như ví dụ ta cần xây dựng các WebUserControl: Top.ascx, TopMenu.ascx: Hiển thị các mục trên menu như: Home, Thông tin về độc

giả,…Left.ascx: Hiển thị các loại sách. Right.ascx: Tìm kiếm theo tên sách hoặc tác giả. và Lietkesach.ascx như sau:



5.3.3.1 Xây dựng WebUserControl: Top.ascx

+ Chọn HTML ở cửa sổ ToolBox: Tạo ra 1 Table: 2 dòng 2 cột, sau đó trộn 2 ô ở dòng đầu tiên thành 1 ô

+ Chọn Standard ở cửa sổ ToolBox: Tạo ra 3 Label: Lable1 để hiển thị ngày, Label2 để hiển thị tháng, Label 3 để hiển thị năm

+ Chọn Standard ở cửa sổ ToolBox: tạo ra 1 Image, đưa ảnh vào bằng thuộc tính ImageUrl ở cửa sổ Properties.

+ Kích chuột phải lên từng ô chọn Style để tạo màu cho các ô

+ Trên sự kiện Page_Load ta đưa ngày, tháng năm vào cho các Label.

```
protected void Page_Load(object sender, EventArgs e)
  {
     Label1.Text = DateTime.Now.Day.ToString();
     Label2.Text = DateTime.Now.Month.ToString();
```

```
    Label3.Text = DateTime.Now.Year.ToString();
  }
```

5.3.3.2 Xây dựng WebUserControl: TopMenu.ascx:

+ Chọn HTML ở cửa sổ ToolBox: Tạo ra 1 Table: 2 dòng 2 cột, sau đó trộn 2 ô ở dòng đầu tiên thành 1 ô

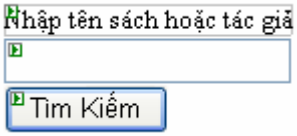
+ Chọn Standard ở cửa sổ ToolBox: Tạo ra 5 LinkButton: lần lượt với tên LinkButton1, LinkButton2, LinkButton3, LinkButton4, LinkButton5, tại thuộc tính Text của mỗi LinkButton lần lượt gõ Home, ThongTinDocGia, Mượn sách, Đổi mật khẩu và Đăng nhập. Khi người sử dụng chọn 1 LinkButton sẽ liên kết đến 1 trang Web nào đó ta sử dụng đối tượng Response.Redirect(url), cụ thể như sau:

```
protected void LinkButton1_Click(object sender, EventArgs e)
  {     Session["UserName"] = null;
    Response.Redirect("~/Default.aspx");
  }
protected void LinkButton2_Click(object sender, EventArgs e)
  {
    Response.Redirect("~/ThongTinDocGia.aspx");
  }
protected void LinkButton3_Click(object sender, EventArgs e)
  {
    Response.Redirect("~/MuonSach.aspx");
  }
protected void LinkButton5_Click(object sender, EventArgs e)
  {
    Response.Redirect("~/DoiMatkhau.aspx");
```

```
}
protected void LinkButton5_Click(object sender, EventArgs e)
{
    Response.Redirect("~/DangNhap.aspx");
}
```

5.3.3.3 Xây dựng WebUserControl: Right.ascx

+ Để tìm theo tên sách hoặc tên tác giả với giao diện như sau:



+ Chọn Standard ở cửa sổ ToolBox:

Tạo 1 Label với thuộc tính text là Nhập tên sách hoặc tác giả:

Tạo 1 TextBox với tên Textbox1

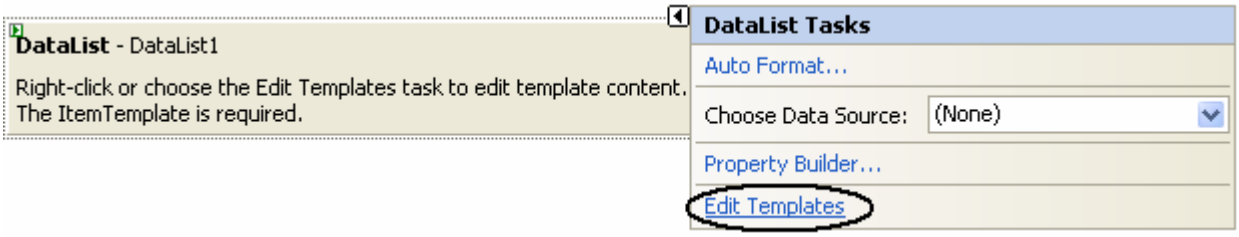
Tạo 1 Button với thuộc tính Name: Button1 và Text: Tìm kiếm, trên sự kiện click của nút này liên kết sang trang Default.aspx với tên tham số là ma và giá trị của tham số là: Textbox1.Text:

```
protected void Button1_Click(object sender, EventArgs e)
{
    Response.Redirect("Default.aspx?ma=" + TextBox1.Text);
}
```

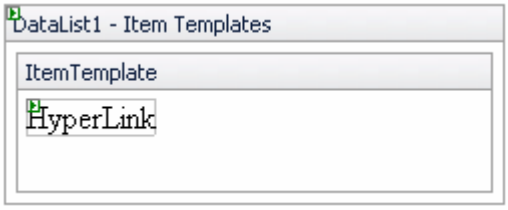
5.3.3.4 Xây dựng WebUserControl: Left.ascx

+ Để hiển thị ra tất cả loại sách, khi người sử dụng chọn 1 loại sách sẽ hiển thị ra tất cả sách thuộc loại này:

+ Chọn Data ở cửa sổ ToolBox: tạo ra 1 DataList | Edit Templates như sau:



+ Chọn Standard ở cửa sổ ToolBox: Tạo 1 HyperLink ở ItemTemplate



+ Vào Source sửa lại như sau:

```
<%@ Control Language="C#" AutoEventWireup="true" CodeFile="WebUserControl.ascx.cs"
Inherits="UC_WebUserControl" %>
<asp:DataList ID="DataList1" runat="server">
    <ItemTemplate>
        <asp:HyperLink ID="HyperLink1" runat="server" CssClass="A:Hover" NavigateUrl=
  '<%# "~/Default.aspx?maloai="+Eval("maloai")+"&tenloai="+Eval("tenloai")%>'
      Text='<%# Bind("TenLoai") %>'></asp:HyperLink>
    </ItemTemplate>
</asp:DataList>
```

Design Source

Trong đó: Text='<%# Bind("TenLoai") %>' sẽ hiển thị tên liên kết là tên của tất cả loại sách

NavigateUrl='<%#
"~/Default.aspx?maloai="+Eval("maloai")+"&tenloai="+Eval("tenloai")%>'

Khi chọn tên loại sẽ liên kết sang trang Default.aspx và truyền 2 tham số với tên maloai và tenloai và giá trị của hai tham số này tương ứng với tên loại đã chọn.

+ Trên sự kiện Page_Load ta nạp dữ liệu vào cho DataList :

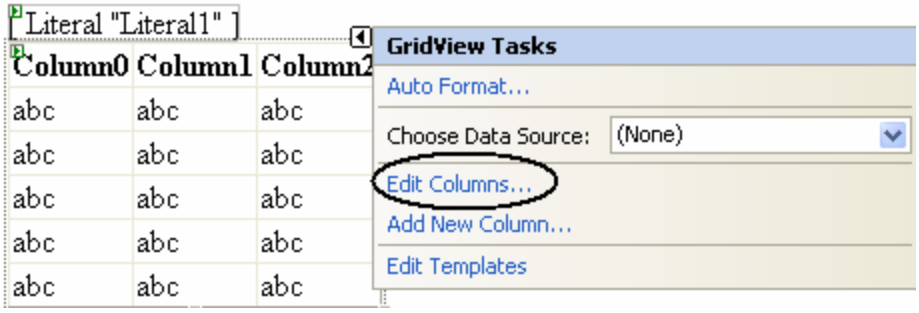
```
protected void Page_Load(object sender, EventArgs e)

{if (!IsPostBack)

 {
```

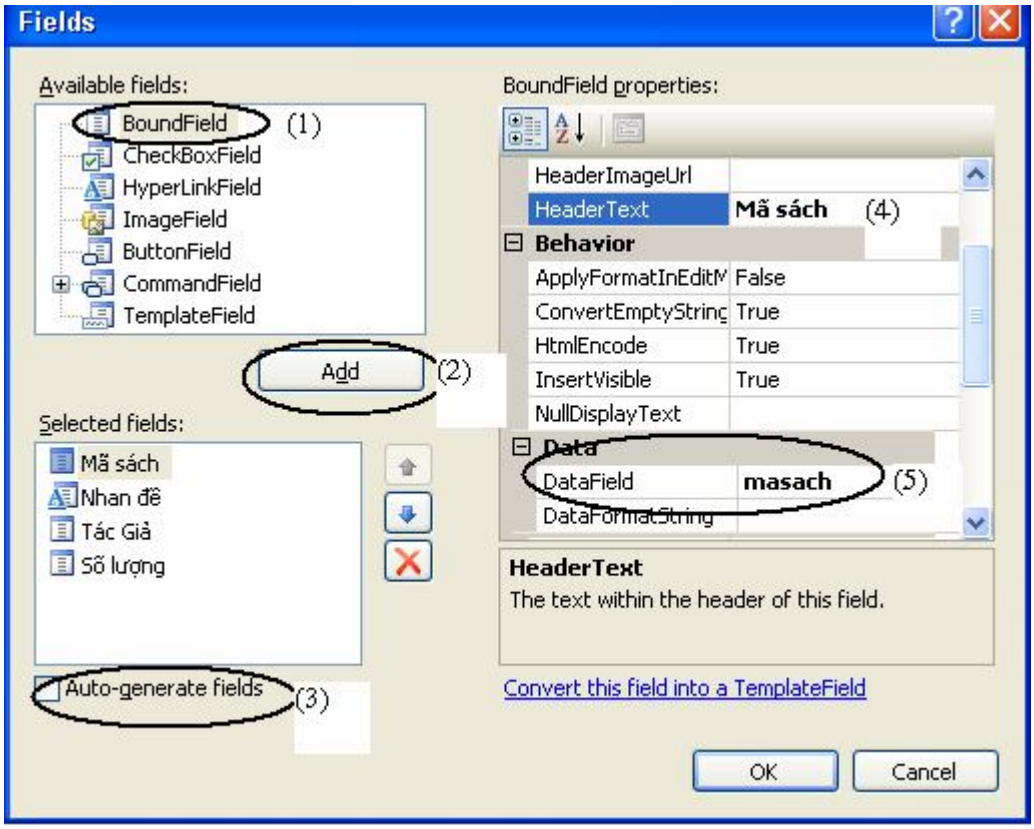
```
  string cnstr =
    WebConfigurationManager.ConnectionStrings["thuvien"].ConnectionString;
  DuaDulieuVaoDieuKhien dl = new DuaDulieuVaoDieuKhien(cnstr);
  dl.CommandText = "select * from PhanLoai";
  dl.KhoiTao();
  dl.NapDuLieuVaoDataList(DataList1);
 }
}
```

5.3.3.5 Xây dựng WebUserControl: LietKeSach.ascx

+ Để hiển thị thông tin của các quyển sách có tên loại chọn ở left.ascx hoặc hiển thị thông tin của các quyển sách có tên sách hoặc tên tác giả nhập ở Right.ascx:

+ Chọn Data ở cửa sổ ToolBox: tạo ra 1 Literal
+ Chọn Data ở cửa sổ ToolBox: tạo ra 1 GridView | Edit Columns để tạo các cột:





(1),(2): Chọn BoundField và Add để chèn thêm 1 trường vào GridView.

(3): Không cho phát sinh thêm trường, (4) Đặt tiêu đề cho cột là Mã sách.

(5) Dữ liệu để trường vào cho cột này là masach

+ Tương tự như vậy ta tạo ra các cột Tác giả và số lượng

+ Riêng cột Nhan đề ta vừa hiển thị dữ liệu vừa tạo liên kết, để tạo cột Nhan đề tại (1) ta chọn HyperLinkField | Add, tại (4) gõ tiêu đề là Nhan đề, (5) ta chọn như sau:

Trong đó:

- DataTextField: nhande là trường cần hiển thị dữ liệu và tạo liên kết
- DataNavigateUrlFormatString: ~/MuonSach.aspx?masach={0}, khi ta chọn nhan đề sách sẽ liên kết sang trang MuonSach.aspx và truyền 1 tham số là masach sang trang MuonSach.aspx
- DataNavigateUrlField: masach là giá trị của tham số truyền sang trang Muonsach.aspx.

+ Tại cửa sổ Properties ta đặt 2 thuộc tính cho GridView:

- Width: 100%

- EnableSortingAndPagingCallbacks:True

+ Sau khi thực hiện các thao tác trên ta nhận được mã HTML phát sinh như sau:

```
<asp:Literal ID="Literal1" runat="server"></asp:Literal><br />
<asp:GridView ID="GridView1" runat="server" AutoGenerateColumns="False"
  EnableSortingAndPagingCallbacks="True" Width="100%">
    <Columns>
        <asp:BoundField DataField="masach" HeaderText="M&#227; s&#225;ch" />
        <asp:HyperLinkField DataNavigateUrlFields="masach"
        DataNavigateUrlFormatString="~/MuonSach.aspx?masach={0}"
            DataTextField="nhande" HeaderText="Nhan đề" />
        <asp:BoundField DataField="tacgia" HeaderText="T&#225;c Giả" />
        <asp:BoundField DataField="soluong" HeaderText="Số lượng" />
    </Columns>
</asp:GridView>
```

+ Trên sự kiện Page Load ta nhận về mã loại và tên loại từ left.ascx hoặc nhận về tên sách hoặc tên tác giả từ right.aspx, từ đó ta hiển thị sách ra GridView1:

- Đầu tiên ta xây dựng hàm GetSql để xác định người dùng chọn tên loại ở Left.ascx hay tìm kiếm theo tên sách hoặc tên tác giả ở Right.ascx, hàm xác định câu lệnh sql tương ứng.

```
string[] GetSql()

 {

 string maloai = Request.QueryString.Get("maloai");//Lấy về mã loại từ Left.ascx

 //Lấy về tên sách hoặc tên tác giả từ Right.ascx

 string ma = Request.QueryString.Get("ma");

 string[] sql = new string[2];

 if (maloai != "" && maloai != null)//nếu người dùng chọn tên loại ở Left.asxx

  {sql[0] = "select masach,nhande,tacgia,nxb,soluong from sach,Phanloai where

                    sach.maloai=phanloai.maloai and Phanloai.maloai=@maloai";

   sql[1] = maloai;

  }

  if (ma != "" && ma != null)// Nếu người dùng đi tìm kiếm ở Right.ascx
```

```
    {sql[0] = "select masach,nhande,tacgia,nxb,soluong from sach where
                                    nhande like @maloai or tacgia like @maloai";
     sql[1] = "%" + ma + "%";
     }
   return sql;
 }
protected void Page_Load(object sender, EventArgs e)
 {
   string[] ma = new string[2] { "","" };
  ma= GetSql();
   if (ma[0] != "" && ma[0] != null)
  { string cnstr =
        WebConfigurationManager.ConnectionStrings["thuvien"].ConnectionString;
        DuaDulieuVaoDieuKhien dl = new DuaDulieuVaoDieuKhien(cnstr);
     dl.CommandText = ma[0];
     dl.ParameterCollection = new string[1] { "@maloai" };
     dl.ValueCollection = new string[1] { ma[1] };
     dl.KhoiTao();
    //Hiển thị tối đa 5 dòng trên GridView, số trang hiển thị là 10
     dl.NapDuLieuVaoGridView(this.GridView1, 5, 10, PagerButtons.Numeric);
     //tạo liên kết trên cột số 1 (nhan đề), truyền tham số với tên là masach, giá trị tham số trên cột
    // thứ 0 (masach) sang trang MuonSach.aspx
        dl.TaoLienKetTrongGridView(GridView1, "~/MuonSach.aspx?masach=", 1, 0,
                                          "Kích vào tên sách để mượn");
```

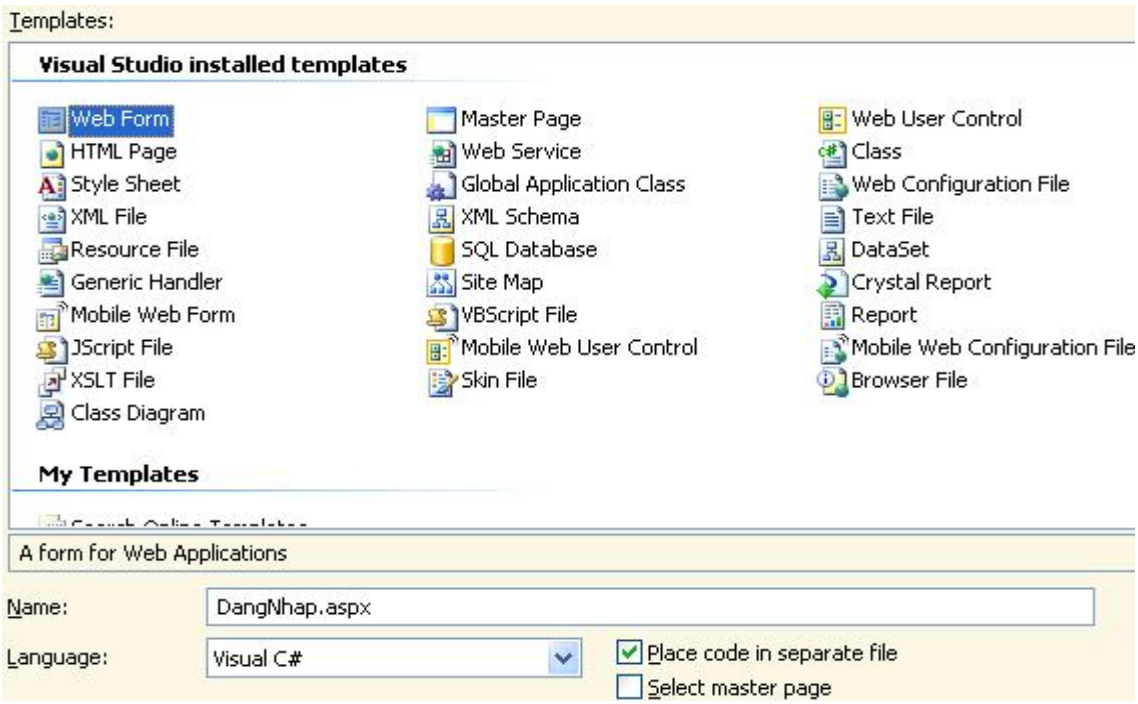
```
    //nếu người dùng chọn tên loại ở left.ascx
        if (Request.QueryString.Get("tenloai") != null)
     Literal1.Text = "<b> Danh sách các sách loại: " +
                Request.QueryString.Get("tenloai") + "</b>";
    else
      Literal1.Text = "<b> Cac sach tim thay <\b>";
  }
 else
 {Literal1.Text = "<B> SINH VIÊN HÃY CHỌN LOẠI SÁCH HOẶC TÌM KIẾM SÁCH ĐỂ MƯỢN</B>";
   }
}
```

+ Trên sự kiện PageIndexChanging ta xác định trang thứ mấy cần hiển thị

```
protected void GridView1_PageIndexChanging(object sender, GridViewPageEventArgs e)
  {
     GridView1.PageIndex = e.NewPageIndex ;
  }
```

5.3.4. Xây dựng các các trang WEB

Theo yêu cầu của ví dụ ta cần xây dựng 5 trang WEB sau:
DangNhap.asp: cho phép độc giả đăng nhập tài khoản.
Default.aspx (trang chủ): để độc giả chọn hoặc tìm sách để mượn.
Muonsach.aspx: Người dùng mượn sách
ThongTinDocGia.aspx: Hiển thị tất cả các thông tin của độc giả
DoiMatKhau.aspx: Độc giả thay đổi mật khẩu
Tạo 1 trang WEB: Web site | Add New Item| Gõ tên trang | Add



5.3.4.1 Tạo trang DangNhap.asp

+ Chọn Standard ở Tool Box:

Tạo ra 1 Literal để hiển thị thông báo nếu đăng nhập không thành công,

Tạo 2 label để hiển thị Mã độc giả và Mật khẩu

Tạo 2 TextBox để gõ mã độc giả (TxtMaDocGia) và gõ mật khẩu (TxtMatKhau).Trên TxtMatKhau đặt thuộc tính: Text Mode: Pasword

Tạo Button Đăng Nhập (Button1)

+ Trên sự kiện Page_Load ta tạo 1 biến kt thuộc lớp Session để kiểm tra xem độc giả đã đăng nhập hay chưa. Biến thuộc lớp Session quản lý một phiên làm việc của một độc giả.

```
protected void Page_Load(object sender, EventArgs e)
  {
```

```
    Session["kt"] = null;
  }
```

+ Trên sự kiện Click của nút Đăng Nhập (Button1) ta kiểm tra xem độc giả đăng nhập đúng hay không nếu đăng nhập đúng thì mở trang Default.aspx, trong sự kiện này ta sử dụng phương thức KiemDangNhap của lớp Docgia.

```
protected void Button1_Click(object sender, EventArgs e)
 { Docgia dg = new Docgia();
  if (dg.KiemDangNhap(TxtMaDocGia.Text,TxtMatKhau.Text)==1)
   {
    Session["kt"] = 1;
        Session["UserName"] = TxtMaDocGia.Text;
    Response.Redirect("~/Default.aspx");
   }
  else
  { Literal1.Text = "<font color=#FF3300><b> Sai Ma doc gia hoac Mat Khau <b></font>";
  }
 }
```

+ Kích chuột phải lên DangNhap.aspx ở cửa sổ Solution Explored | Set As Start Page: để chọn trang DangNhap làm trang chạy mặc định.

5.3.4.2 Tạo trang Default.asp (trang chủ)

+ Chọn HTML trên ToolBox: Tạo 1 Table: 2 dòng, 3 cột, trộn 3 ô của dòng đầu thành 1 ô (bôi đên 3 ô đầu | Kích chuột phải | Merge Cells) :

(1)	(2)	(3)

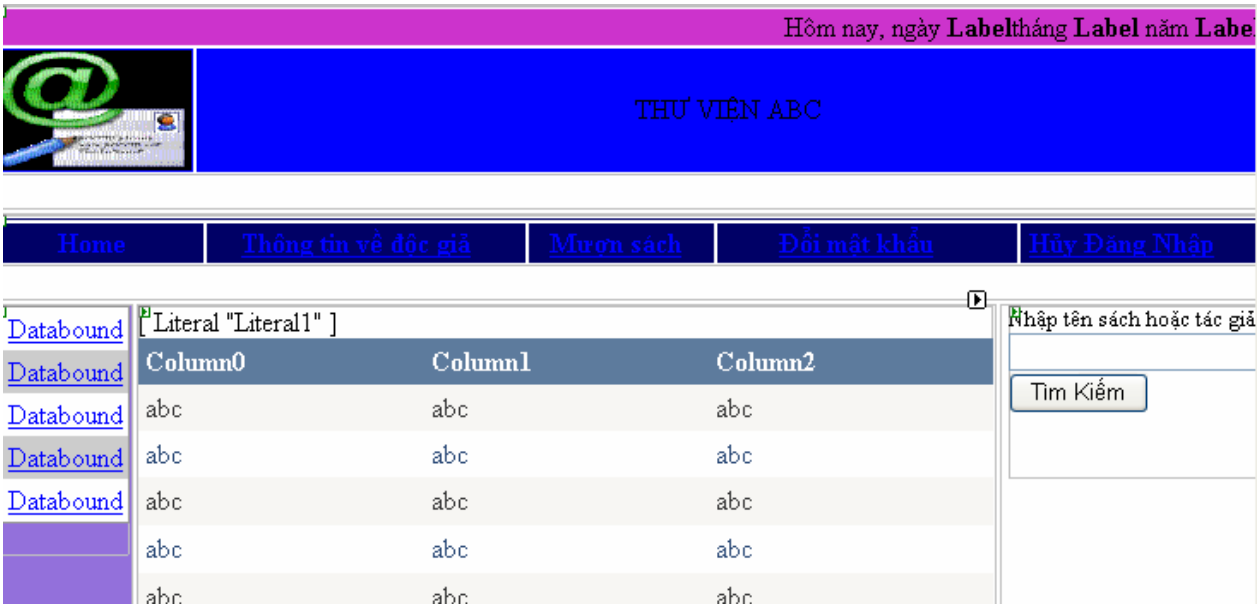
+ Chọn thư mục UC ở cửa sổ Solution Explored:

Kéo Top.ascx và TopMenu.ascx vào dòng đầu tiên.

Kéo Left.ascx vào (1)

Kéo LietKeSach.ascx vào (2)

Kéo Right.ascx vào (3) khi đó ta được giao diện như sau:



+ Nhấn F5 để chạy thử

5.3.4.3 Tạo trang Muonsach.asp

+ Chọn HTML trên ToolBox: Tạo 1 Table: 1 dòng, 1 cột:

+ Chọn thư mục UC ở cửa sổ Solution Explored:

Kéo Top.ascx và TopMenu.ascx vào dòng đầu tiên.

+ Tại dòng thứ 2 của bảng:

Chọn Standard trên ToolBox lần lượt tạo ra các Label, TextBox, Literal và các Button giống như hình trên.

+ Trên sự kiện Page_Load ta kiểm tra xem độc giả đã đăng nhập chưa, nếu đã đăng nhập thì cho mượn sách, nếu chưa thì phải đăng nhập

```
protected void Page_Load(object sender, EventArgs e)
  {
    if (Session["kt"] == null)// Độc giả chưa đăng nhập
    {
       Response.Redirect("~/DangNhap.aspx");
    }
    else
    {    //txtma chứa mã độc giả là mã độc giả đăng nhập
                txtma.Text = Session["UserName"].ToString();
                KiemtraMaDocGia(); // Hiển thị tên độc giả và tên khoa
                TxtNgayMuon.Text = DateTime.Now.ToShortDateString();
     //Khi độc giả chọn tên sách cần mượn, ta có thể lấy mã sách từ
     //LietKesach.ascx truyền sang
           string ma = Request.QueryString.Get("masach");
              if (ma != null && ma != "")
              {
                 TxtMasach.Text = ma;
                 KiemTrasach(); // Hiển thị tên sách, tên tác giả
                 ButKiemTra.Enabled = false;
```

```
            }
        }
    }
```

+ Hàm KiemtraMaDocGia kiểm tra xem mã độc giả nhập có đúng không nếu đúng thì hàm hiển thị họ tên và tên khoa của độc giả

```
void KiemtraMaDocGia()
    {
        Docgia dg = new Docgia();
        string[] st = new string[2];
        st = dg.GetDocGia(txtma.Text);
        if (st != null)
        {
            txtHoTen.Text = st[0];
            TxtKhoa.Text = st[1];
        }
    }
```

+ Hàm KiemTraSach kiểm tra xem mã sách nhập có đúng không nếu đúng thì hàm hiển thị tên sách và tác giả

```
    void KiemTrasach()
    {
        Literal1.Text = "";
        Sach dg = new Sach();
        string[] st = new string[2];
```

```
    st = dg.GetSach(TxtMasach.Text);
    if (st != null)
    {
       TxtTensach.Text = st[0];
       TxtTacGia.Text = st[1];
    }
    else
    {
       Literal1.Text = "Khong co ma sach nay";
    }
  }
```

+ Khi người sử dụng không chọn tên sách cần mượn mà tự gõ mã sách khi đó ta đi tìm xem thử có mã sách này không, nếu có thì hiển thị tên sách và tác giả:

```
  protected void ButKiemTra_Click(object sender, EventArgs e)// Nút Kiểm tra
  {
     KiemTrasach();
  }
```

+ Khi độc giả chọn nút mượn thì kiểm tra xem độc giả này còn mượn sách trong thư viên không, nếu đã trả hết sách thì cho độc giả này mượn sách

```
protected void ButMuon_Click(object sender, EventArgs e)
  {  PhieuMuon pm = new PhieuMuon();
     if ( TxtMasach.Text == "")Literal2.Text = "Ban chua nhap ma sach ";
     else
     if (pm.KiemTraMuon(txtma.Text) == 1)
```

```
    {
      Literal2.Text = "Ban chua tra sach";
    }
    else
    {  if (pm.CapNhat(txtma.Text, DateTime.Now, TxtMasach.Text, 1, 1)==1)
       Literal2.Text = "Ban da muon sach";
      else
       Literal2.Text = "Sach na da het trong thu vien";
    }
  }
```

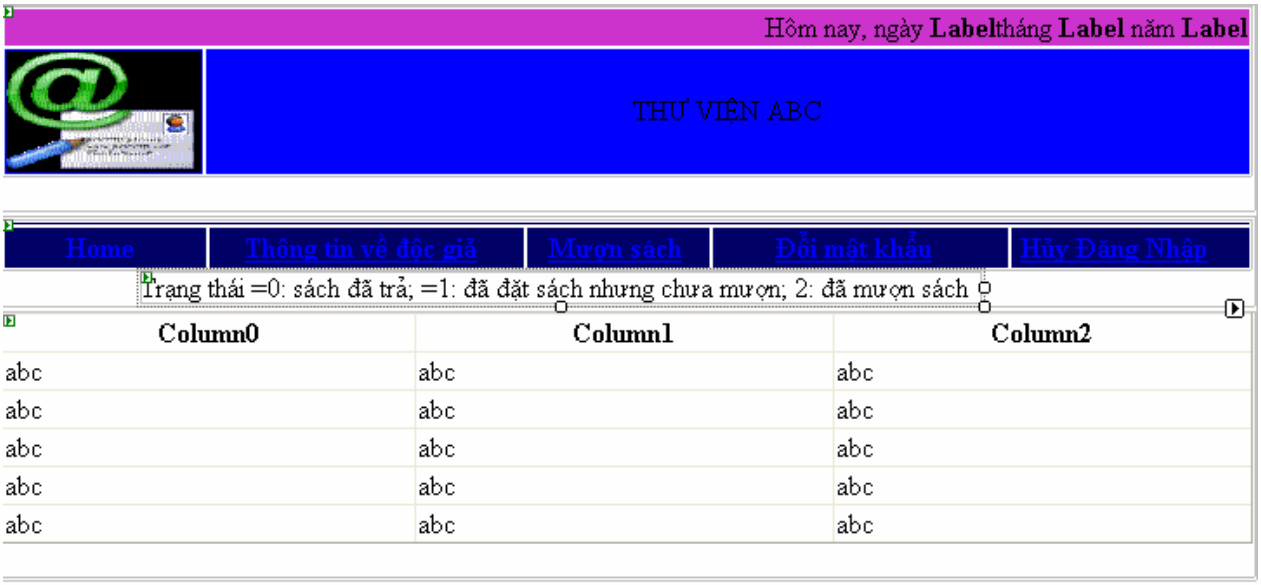
5.3.4.3 Tạo trang ThongTinDocGia.aspx

+ Chọn HTML trên ToolBox: Tạo 1 Table: 1 dòng, 1 cột:
+ Chọn thư mục UC ở cửa sổ Solution Explored:

Kéo Top.ascx và TopMenu.ascx vào dòng đầu tiên.

+ Tại dòng thứ 2 của bảng:

Chọn Standard trên ToolBox tạo ra 1 Label và 1 DataGrid:



+ Trên sự kiện Page_Load ta kiểm tra xem độc giả đã đăng nhập chưa, nếu đã đăng nhập thì cho hiển thị thông tin, nếu sách nào đã đặt chổ nhưng chưa lấy ta tạo liên kết để độc giả có thể hủy đặt chổ:

```
    protected void Page_Load(object sender, EventArgs e)
     {
         if (Session["kt"] == null)
         {
             Response.Redirect("~/DangNhap.aspx");
         }
         else
         {
             Docgia dg = new Docgia();
             dg.HienThiThongTin(GridView1,
                                Session["UserName"].ToString());
             taolienket();
       //Lấy về mã sách cần hủy đặt chổ
             string masach = Request.QueryString["masach"];
             if (masach != null)//độc giả đã chọn sách cần hủy
             {
                 PhieuMuon pm = new PhieuMuon();
                 pm.HuyDatCho(masach);
                 dg.HienThiThongTin(GridView1,
                  Session["UserName"].ToString());
             }
         }
     }
void taolienket()
    {
        foreach (GridViewRow rows in GridView1.Rows)
        {
            HyperLink hyperLink = new HyperLink();
            hyperLink.Text = rows.Cells[3].Text;
            if (Int16.Parse(rows.Cells[6].Text) == 1)
            {
                hyperLink.NavigateUrl = "ThongTinDocGia.aspx?masach=" +
rows.Cells[3].Text;
                hyperLink.ToolTip = "Chọn tên sách để hủy";
            }
                rows.Cells[3].Controls.Add(hyperLink);
        }
    }
```

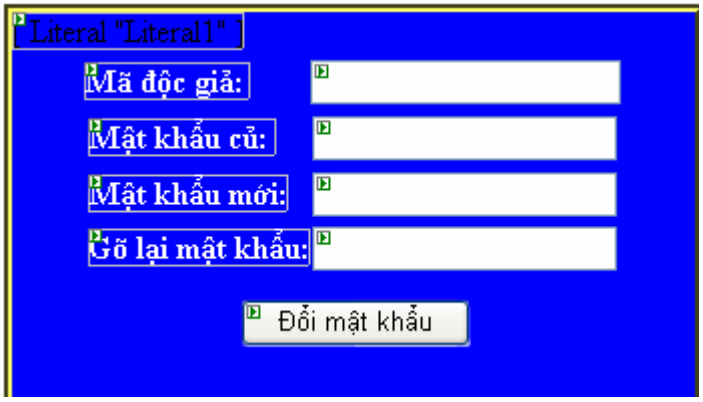
Trong đó: phương thức taolienket() để tìm xem trong GridView có dòng nào có

trangthai=1 hay không, nếu có thì tạo liên kết trên trường masach.

```
void taolienket()
    {
        foreach (GridViewRow rows in GridView1.Rows)
        {
            HyperLink hyperLink = new HyperLink();
            hyperLink.Text = rows.Cells[3].Text;
            if (Int16.Parse(rows.Cells[6].Text) == 1)
            {
                hyperLink.NavigateUrl = "ThongTinDocGia.aspx?masach=" +
       rows.Cells[3].Text;
                hyperLink.ToolTip = "Chọn tên sách để hủy";
            }
                rows.Cells[3].Controls.Add(hyperLink);
        }
    }
```

5.3.4.4 Tạo trang DoiMatKhau.aspx

Chọn Standard trên ToolBox tạo ra các Label , TextBox và Literal như sau:



+ Mã độc giả chính là UserName độc giả đăng nhập

```
protected void Page_Load(object sender, EventArgs e)

  {

     TxtMaDocGia.Text = Session["UserName"].ToString();

  }
```

+ Khi độc giả chọn Button Đổi mật Khẩu thì kiểm tra xem mã độc giả nhập có đúng không, mật khẩu và mới có trùng nhau hay không, nếu đúng thì cho thay đổi

```
protected void Button1_Click(object sender, EventArgs e)
```

```
{
 Docgia dg = new Docgia();
if (TxtMatkhaumoi.Text.Equals(TxtGoLaiMatKhau.Text))
{
  if(dg.DoiMatKhau(TxtMaDocGia.Text,TxtMatKhaucu.Text,TxtMatkhaumoi.Text)==1)
    Literal1.Text = "Doi mat khau thanh cong";
  else
    Literal1.Text = "Ma doc gia hoac mat khau cu nhap sai";
}
else
 Literal1.Text="Mat khau khong khop nhau";
}
```

5.3.5 Yêu cầu bổ sung:

+ Với ví dụ trên chỉ tập trung vào việc đặt trước chỗ để mượn sách, chưa quan tâm đến chức năng của người quản thư.

+ Yêu cầu bổ sung, hãy tạo ra các trang thực hiện các công việc sau:

- Để người quản thư đăng nhập tài khoản
- Để người quản thư cho mượn sách (căn cứ vào các độc giả đã đặt chỗ trước)
- Hiển thị thông tin độc giả đã mượn quá hạn
- Để độc giả trả sách.

PHỤ LỤC

Bảng tương quan/chuyển đổi kiểu dữ liệu ở .NET Framework với các Data Provider

.NET Framework type	System.Data.DbType	SqlDbType	OleDbType	OdbcType	OracleType

.NET Framework type	System.Data.DbType	SqlDbType	OleDbType	OdbcType	OracleType
bool	**Boolean**	**Bit**	**Boolean**	**Bit**	**Byte**
byte	**Byte**	**TinyInt**	**UnsignedTinyInt**	**TinyInt**	**Byte**
byte[]	**Binary**	**VarBinary.** Việc chuyển đổi ngầm định này là không đúng nếu mảng byte là lớn hơn kích thước tối đa của một **VarBinary** (8000 bytes).	**VarBinary**	**Binary**	**Raw**
char		Không hỗ trợ.	**Char**	**Char**	**Byte**
DateTime	**DateTime**	**DateTime**	**DBTimeStamp**	**DateTime**	**DateTime**
Decimal	**Decimal**	**Decimal**	**Decimal**	**Numeric**	**Number**
double	**Double**	**Float**	**Double**	**Double**	**Double**
float	**Single**	**Real**	**Single**	**Real**	**Float**
Guid	**Guid**	**UniqueIdentifier**	**Guid**	**UniqueIdentifier**	**Raw**
Int16	**Int16**	**SmallInt**	**SmallInt**	**SmallInt**	**Int16**
Int32	**Int32**	**Int**	**Int**	**Int**	**Int32**
Int64	**Int64**	**BigInt**	**BigInt**	**BigInt**	**Number**
object	**Object**	**Variant**	**Variant**	Không hỗ trợ.	**Blob**

.NET Framework type	System.Data.DbType	SqlDbType	OleDbType	OdbcType	OracleType
string	**String**	**NVarChar**. Chuyển đổi ngầm định này là không đúng nếu string lớn hơn kích thước tối đa của một **NVarChar** (4000 ký tự).	**VarWChar**	**NVarChar**	**NVarChar**
TimeSpan	**Time**	Không hỗ trợ.	**DBTime**	**Time**	**DateTime**
UInt16	**UInt16**	Không hỗ trợ.	**UnsignedSmallInt**	**Int**	**UInt16**
UInt32	**UInt32**	Không hỗ trợ.	**UnsignedInt**	**BigInt**	**UInt32**
UInt64	**UInt64**	Không hỗ trợ.	**UnsignedBigInt**	**Numeric**	**Number**
	AnsiString	**VarChar**	**VarChar**	**VarChar**	**VarChar**
	AnsiStringFixedLength	**Char**	**Char**	**Char**	**Char**
	Currency	**Money**	**Currency**	Không hỗ trợ.	**Number**
	Date	Không hỗ trợ.	**DBDate**	**Date**	**DateTime**
	SByte	Không hỗ trợ.	**TinyInt**	Không hỗ trợ.	**SByte**
	StringFixedLength	**NChar**	**WChar**	**NChar**	**NChar**
	Time	Không hỗ trợ.	**DBTime**	**Time**	**DateTime**
	VarNumeric	Không hỗ trợ.	**VarNumeric**	Không hỗ trợ.	**Number**

TÀI LIỆU THAM KHẢO

(1) Stephen C. Perry, Core C# and .NET, Prentice Hall PTR, 2005
(2) Phạm Hữu Khang, C# 2005, Tập 5 Lập trình ASP.NET 2.0, quyển 4: Đối tượng ADO.NET và XML, Nhà xuất bản Lao Động Xã Hội.
(3) Microsoft Corporation, MSDN 2005