

TRƯỜNG ĐẠI HỌC TRÀ VINH  
KHOA KỸ THUẬT VÀ CÔNG NGHỆ



# **TÀI LIỆU GIẢNG DẠY**

## **MÔN**

# **LẬP TRÌNH ỨNG DỤNG TRÊN WINDOWS**

*GV biên soạn: Phạm Minh Dương*

**Trà Vinh, 2014**  
**Lưu hành nội bộ**



KHOA KỸ THUẬT VÀ CÔNG NGHỆ

## TÌNH TRẠNG PHÊ DUYỆT TÀI LIỆU GIẢNG DẠY

Tên tài liệu giảng dạy: Lập trình ứng dụng trên Windows

Ngày hoàn chỉnh: 26/6/2014

Tác giả biên soạn: Phạm Minh Dương

Đơn vị công tác: Trường Đại học Trà Vinh

Địa chỉ liên lạc: Bộ môn Công nghệ Thông tin – Khoa Kỹ thuật và Công nghệ

*Trà Vinh, ngày ..... tháng ..... năm 201*

Tác giả

*(Ký & ghi họ tên)*

### PHÊ DUYỆT CỦA BỘ MÔN

Đồng ý sử dụng tài liệu giảng dạy .....  
..... do ..... biên soạn để giảng dạy  
môn.....

*Trà Vinh, ngày ..... tháng ..... năm.....*

**TRƯỞNG BỘ MÔN**

### PHÊ DUYỆT CỦA KHOA

*Trà Vinh, ngày ..... tháng ..... năm 201*

**TRƯỞNG KHOA**

# MỤC LỤC

<b>MỤC LỤC .....</b>	<b>1</b>
<b>DANH MỤC CÁC HÌNH.....</b>	<b>8</b>
<b>CHƯƠNG 1 TỔNG QUAN VỀ .NET FRAMEWORK.....</b>	<b>10</b>
1.1 Tổng quan về kiến trúc của .NET Framework .....	10
1.2 Môi trường thực thi ngôn ngữ chung CLR .....	11
1.2.1 Biên dịch mã lệnh .NET .....	11
1.2.2 Hệ thống kiểu dữ liệu chung CTS (Common Type System).....	13
1.2.3 Assemblies .....	13
1.2.4 Tiền biên dịch một Assembly .....	17
1.2.5 Kiểm chứng mã lệnh (Code Verification) .....	17
<b>CHƯƠNG 2 NGÔN NGỮ LẬP TRÌNH C#.....</b>	<b>18</b>
2.1 Tạo một dự án mới .....	18
2.2 Biến dữ liệu .....	19
2.2.1 Tầm hoạt động của biến.....	19
2.2.2 Hằng dữ liệu .....	20
2.3 Các kiểu dữ liệu định nghĩa sẵn của C# .....	20
2.3.1 Kiểu dữ liệu giá trị được định nghĩa sẵn .....	21
2.3.2 Kiểu dữ liệu tham chiếu được định nghĩa sẵn .....	21
2.4 Cấu trúc điều khiển.....	23
2.4.1 Câu lệnh điều kiện .....	23

2.4.2 Cấu trúc lặp.....	26
2.4.3 Câu lệnh nhảy .....	30
2.5 Hàm và thủ tục .....	31
2.6 Mảng (array).....	32
2.6.1 Cú pháp khai báo array .....	32
2.6.2 Làm việc với array .....	33
2.6.3 Array nhiều chiều .....	36
2.7 Các toán tử.....	36
2.8 Xử lý lỗi trong C# .....	37
2.9 Namespace.....	38
<b>CHƯƠNG 3 HƯỚNG ĐỐI TƯỢNG TRONG C# .....</b>	<b>42</b>
3.1 Lớp đối tượng.....	42
3.2 Thừa kế trong C#.....	43
3.3 Phương thức (Method) .....	44
3.4 Nạp chồng phương thức (Method Overloading) .....	47
3.5 Ghi đè phương thức .....	47
3.6 Gọi phương thức.....	49
3.7 Nạp chồng toán tử .....	50
<b>CHƯƠNG 4 LẬP TRÌNH TRÊN WINDOWS FORM.....</b>	<b>56</b>
4.1 Windows Form .....	56
4.2 Control.....	58
4.3 Các Control thông dụng .....	59

4.3.1 Label, TextBox, Button .....	59
4.3.2 ListBox .....	61
4.3.3 ComboBox.....	62
4.3.4 CheckBox và RadioButton .....	64
4.3.5 ListView .....	66
4.3.6 TreeView .....	68
4.3.7 GroupBox .....	69
4.3.8 Tab Control.....	71
4.3.9 PictureBox .....	72
4.3.10 DateTimePicker .....	73
4.3.11 Timer.....	74
4.3.12 ProgressBar.....	74
4.3.13 ToolTip .....	75
4.3.14 ContextMenuStrip.....	76
4.3.15 ImageList .....	77
<b>CHƯƠNG 5 XỬ LÝ DỮ LIỆU VỚI ADO.NET.....</b>	<b>81</b>
5.1 Kiến trúc tổng quan của ADO.NET .....	81
5.2 Tổng quan về các mô hình xử lý dữ liệu trong ADO.NET .....	83
5.2.1 Mô hình kết nối trực tiếp .....	83
5.2.2 Mô hình ngắt kết nối.....	85
5.3 Làm việc với mô hình Kết nối trong ADO.NET.....	87
5.3.1 Lớp Connection .....	88

5.3.1.1 Connection string .....	89
5.3.2 Đối tượng Command .....	90
5.3.2.1 Tạo đối tượng Command .....	91
5.3.2.2 Thực thi một Command .....	92
5.3.2.3 Thực thi Stored Procedure (thủ tục lưu trữ) với đối tượng Command .....	93
5.3.2.4 Sử dụng tham số trong các lệnh không phải là Stored Procedures.....	98
5.3.3 Đối tượng DataReader .....	98
5.3.3.1 Truy xuất các dòng dữ liệu với DataReader .....	99
5.3.3.2 Truy xuất giá trị của column.....	100
5.4 Làm việc với mô hình Ngắt kết nối: DataSet và DataTable .....	103
5.4.1 Lớp DataSet .....	103
5.4.1.1 DataTable .....	104
5.4.1.2 DataColumn .....	104
5.4.1.3 DataRows .....	107
5.4.1.4 DataView. ....	109
5.4.2 Nạp dữ liệu vào DataSet .....	110
5.4.2.1 Dùng DataReader để nạp dữ liệu vào DataSet.....	110
5.4.2.2 Nạp dữ liệu vào DataSet bằng DataAdapter .....	112
5.4.3 Cập nhật CSDL bằng DataAdapter .....	113
5.4.3.1 CommandBuilder.....	113
5.4.3.2 Đồng bộ hóa dữ liệu giữa DataSet và CSDL.....	115
5.4.4 Định nghĩa Relationships giữa các Table trong DataSet.....	115

5.5 Lựa chọn giữa mô hình kết nối và mô hình ngắt kết nối .....	119
5.6 Thiết kế báo biểu (Report).....	119
5.6.1 Giới thiệu .....	119
5.6.2 Tạo báo biểu .....	120
5.6.3 Nguồn dữ liệu cho báo biểu.....	121
5.6.4 Sử dụng CrystalReportViewer để hiển thị báo cáo .....	121
<b>TÀI LIỆU THAM KHẢO.....</b>	<b>128</b>

## DANH MỤC CÁC HÌNH

Hình 1. Kiến trúc .NET Framework .....	11
Hình 2. Chức năng của CLR .....	12
Hình 3. Các kiểu dữ liệu cơ sở của CTS .....	13
Hình 4. Assembly chỉ gồm 1 file .....	15
Hình 5. Assembly chứa nhiều file.....	16
Hình 6. Tạo một Project.....	18
Hình 7. Ví dụ điều khiển TextBox .....	61
Hình 8. Ví dụ điều khiển ListBox .....	62
Hình 9. Ví dụ điều khiển ComboBox .....	64
Hình 10. Ví dụ điều khiển CheckBox và RadioButton.....	65
Hình 11. Ví dụ điều khiển ListView .....	67
Hình 12. Ví dụ điều khiển TreeView .....	69
Hình 13. Ví dụ điều khiển GroupBox .....	70
Hình 14. Ví dụ điều khiển TabControl .....	72
Hình 15. Ví dụ điều khiển PictureBox .....	73
Hình 16. Ví dụ điều khiển DateTimePicker.....	74
Hình 17. Ví dụ điều khiển Timer .....	74
Hình 18. Ví dụ điều khiển ProgressBar .....	75
Hình 19. Ví dụ điều khiển ToolTip.....	76
Hình 20. Ví dụ điều khiển ContextMenuStrip .....	77



Hình 21. Ví dụ điều khiển ImageList.....	78
Hình 22. Kiến trúc ADO.NET .....	81
Hình 23. Mô hình kết nối trực tiếp.....	84
Hình 24. Mô hình ngắt kết nối .....	86
Hình 25. Mô hình DataSet .....	104
Hình 26. Thêm Crystal Report vào Project.....	120

# CHƯƠNG 1

## TỔNG QUAN VỀ .NET FRAMEWORK

✓ **Mục tiêu học tập:** Sau khi học xong bài này, người học có thể:

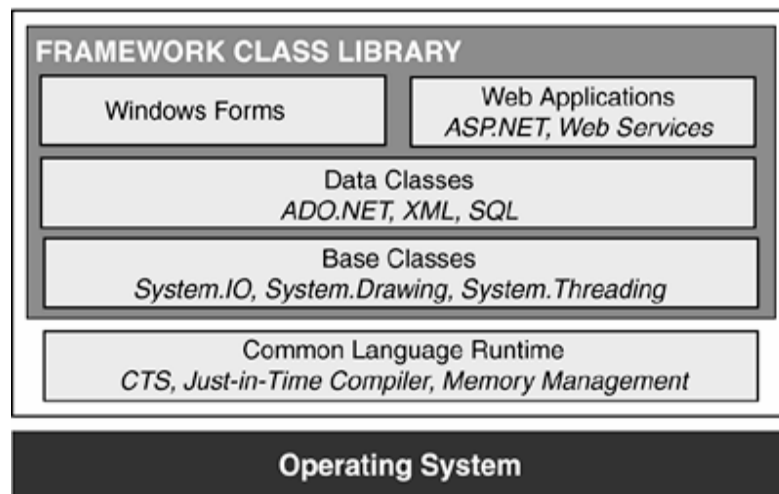
Trình bày tổng quan về kiến trúc .Net Framework

### 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 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.
- 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 thực thi ngôn ngữ chung) và FCL (Framework Class Library – Thư viện lớp khung) như hình 1.



Hình 1. Kiến trúc .NET Framework

CLR 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** (Application Program Interface).

## 1.2 Môi trường thực thi ngôn ngữ chung CLR

CLR 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

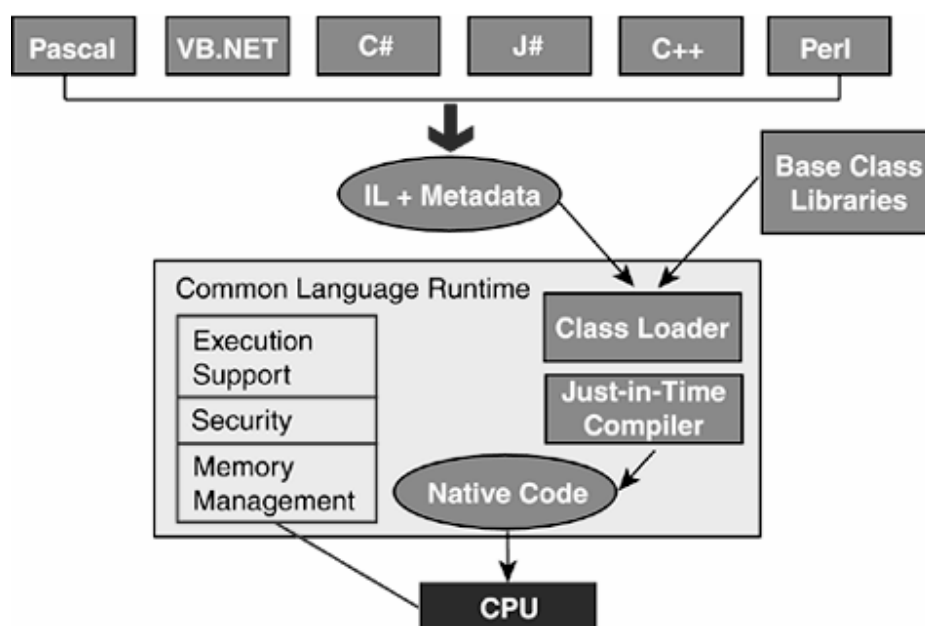
Trình biên dịch tương thích với CLR sẽ sinh mã thực thi cho môi trường thực thi chứ không phải là mã thực thi cho **CPU** (Central Processing Unit) cụ thể. Mã thực thi này được biết đến qua tên gọi CIL (Common Intermediate Language – Ngôn ngữ trung gian chung), hay MSIL (Microsoft Intermediate Language – 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 (Intermediate Language) 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 siêu dữ liệu (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. Quá trình này có thể được minh họa ở hình 2



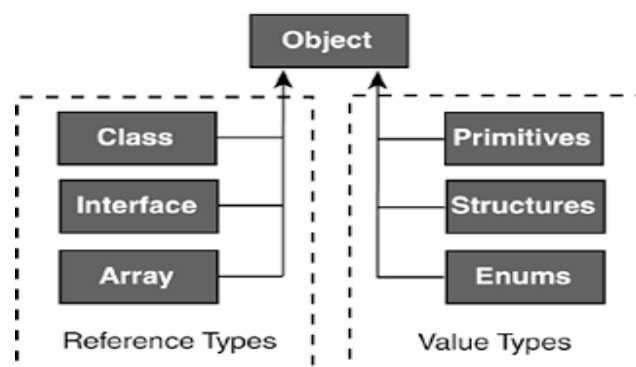
Hình 2. Chức năng của CLR

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 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 assembly có thể chứa một tập hợp gồm một hay nhiều file chứa phần mã lệnh hoặc tài nguyên (như ảnh hoặc dữ liệu XML).

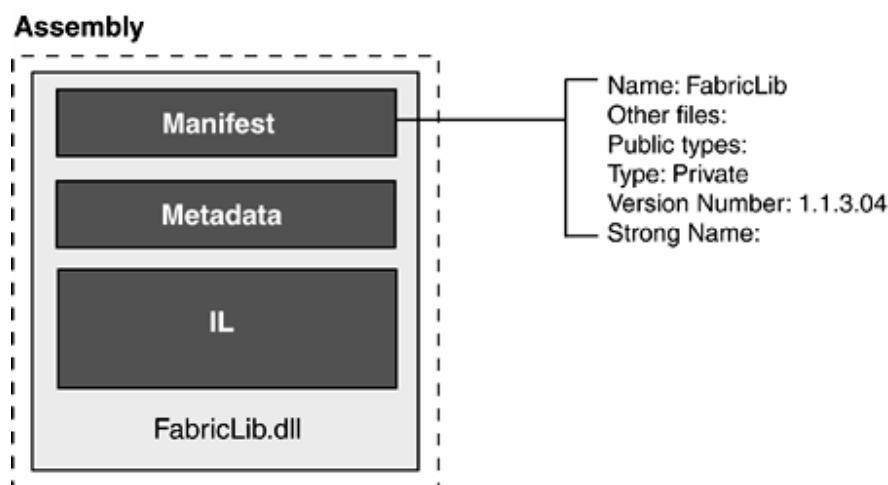
Một assembly được tạo ra khi trình biên dịch tương thích với .NET chuyển một file chứa mã nguồn thành một file .DLL hoặc .EXE. Như minh họa trong hình 4, một

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 4 minh họa cho Assembly chỉ gồm 1 file:



Hình 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.

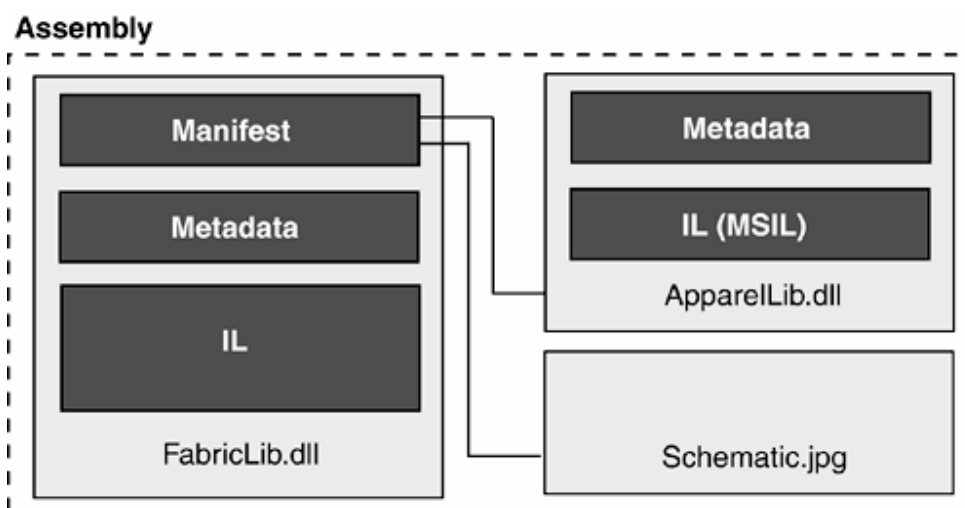
- Trong thế giới .NET, một assembly quy định một biên giới phiên bản. Trường Version Number trong manifest áp dụng cho tất cả các kiểu và tài nguyên trong assembly. Vì vậy, mọi file tạo nên assembly được xem như là một đơn vị đơn nhất có cùng phiên bản.

- Một assembly cũng thiết lập một biên giới bảo mật để định ra quyền hạn truy xuất.

C# sử dụng các bộ 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 5 minh họa cho Assembly chứa nhiều file:



Hình 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 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. 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

#### 1.2.5 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.

#### ▼ Câu hỏi (bài tập) củng cố:

Trình bày tổng quan về kiến trúc .Net Framework?

## CHƯƠNG 2

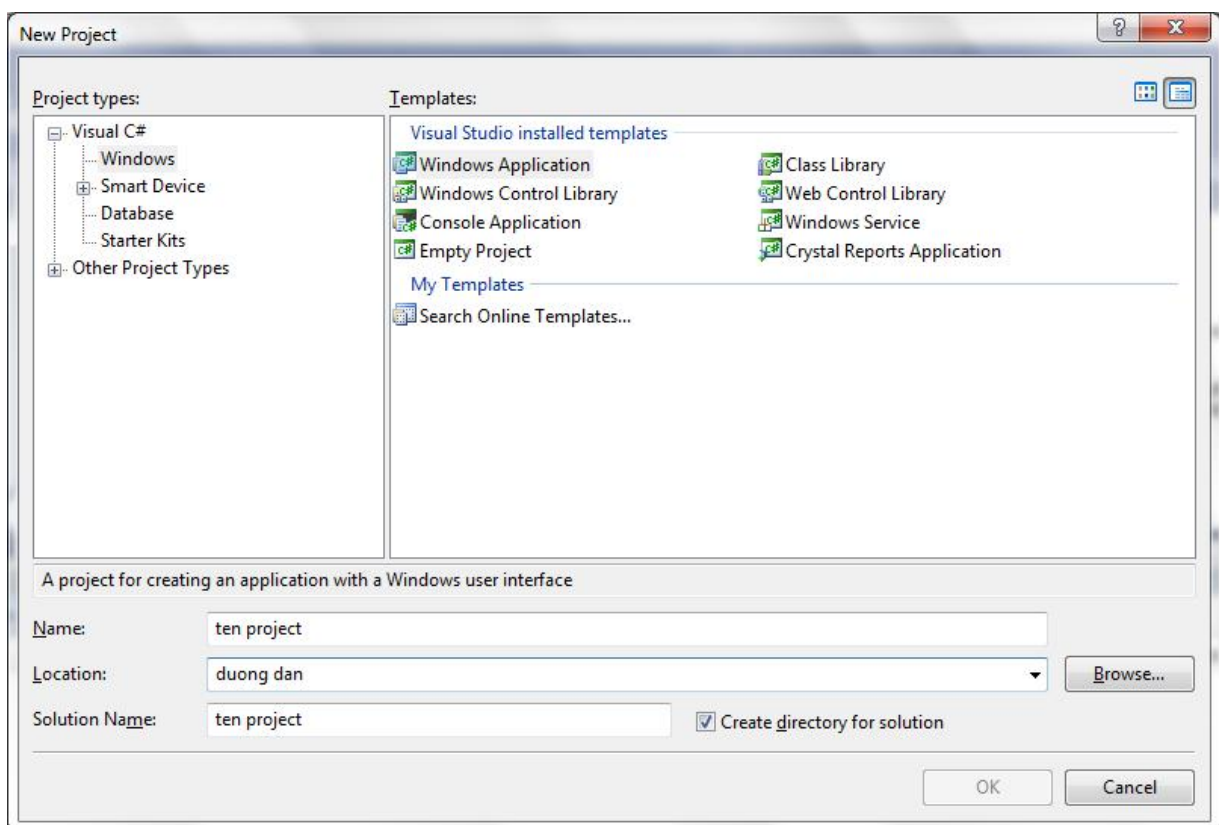
### NGÔN NGỮ LẬP TRÌNH C#

✓ **Mục tiêu học tập:** Sau khi học xong bài này, người học có thể:

Vận dụng kiến thức cơ bản về ngôn ngữ C# để giải quyết một bài toán cụ thể.

#### 2.1 Tạo một dự án mới

Nội dung phần này chủ yếu nhắc lại cách khởi tạo một dự án mới trong môi trường Visual Studio.Net. Bằng cách khởi động Microsoft Visual Studio.Net, sau đó nhấn **Ctrl + Shift + N** hoặc chọn menu tương ứng là **File à New à Project** để tạo mới một project. Hình 6 minh họa tạo một dự án mới trong Visual Studio 2005:



Hình 6. Tạo một Project

Visual Studio.Net xem 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.

## 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:

Ví dụ:

```
i = 10;
```

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

Ví dụ:

```
int i = 10; // khai báo và khởi tạo giá trị cho biến int  
  
double x = 10.25, y = 20; // khai báo và khởi tạo hai biến 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. 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.

Ví dụ :

```
const int a = 100; // Gia trị nay không được thay đổi
```

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 động, và boolean.

*Kiểu số nguyên*

sbyte	System.SByte	8-bit
short	System.Int16	16-bit
int	System.Int32	32-bit
long	System.Int64	64-bit
byte	System.Byte	8-bit
ushort	System.UInt16	16-bit
uint	System.UInt32	32-bit
ulong	System.UInt64	64-bit

*Kiểu dấu chấm động*

Float	System.Single	32-bit
Double	System.Double	64-bit

*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++).

### 2.3.2 Kiểu dữ liệu tham chiếu được định nghĩa sẵn

*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 ghi đè (override) trong lập trình hướng đối tượng.

*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 = "Hello";
        string s2 = s1;

        Console.WriteLine("s1 is " + s1);
        Console.WriteLine("s2 is " + s2);
    }
}
```

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

Kết quả thực thi:

*s1 is a Hello*

*s2 is a Hello*

*s1 is now World*

*s2 is now Hello*

## 2.4 Cấu trúc điều khiển

### 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.

*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++).

*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 Cấu trúc 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.

*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:

```
for (int i = 0; i < 100; i+=10)
{
    for (int j = i; j < i + 10; j++)
    {
        Console.Write(" " + j);
    }
    Console.WriteLine();
}
```

Kết quả thực thi:

0	1	2	3	4	5	6	7	8	9
10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29
30	31	32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47	48	49

50	51	52	53	54	55	56	57	58	59
60	61	62	63	64	65	66	67	68	69
70	71	72	73	74	75	76	77	78	79
80	81	82	83	84	85	86	87	88	89
90	91	92	93	94	95	96	97	98	99

*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:

```
int i = 1;

while (i<=5)

{ //Vòng lặp thực hiện đến khi điều kiện đúng

    //lệnh

    i++;

}
```

*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:

```
int a, b, tong;  
a = 15;  
b = 5;  
do  
{  
    tong = a + b;  
    a += b;  
} while(a<10)
```

*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.

```
foreach(<kiểu dữ liệu> <phần tử> in <tập hợp>)  
{  
    //lệnh  
}
```

Ví dụ:

```
int[] mangnguyen = new int[5];  
int tong = 0;  
foreach(int phantu in mangnguyen)  
{
```

```
tong + = phantu;  
}
```

### 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.

*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**).

*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).

*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.

*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.

Ví dụ:

```
public static void Main()  
{  
    int i = 0;  
    lap: // nhãn  
    Console.WriteLine("i:{0}", i);  
    i++;  
    if ( i < 10)
```

```
        goto lap; // nhảy về nhãn lap
    Console.ReadLine();
}
```

## 2.5 Hàm và thủ tục

Trong C#, hàm và thủ tục được gọi chung là phương thức, cú pháp như sau:

```
Kiểu_trả_về tên_phương_thức (danh sách các tham số)
{
    //lệnh
    return giá_trị
}
```

Ví dụ 1:

```
double phep_toan(double a, double b, string toantu)
{
    double kq = 0;
    switch(toantu)
    {
        case "+":
            kq = a + b;
            break;
        case "-":
            kq = a - b;
            break;
    }
}
```

```
}  
  
    return kq;  
  
}
```

Phương thức sau khi tạo có thể gọi sử dụng thông qua tên của nó, cú pháp như sau:

tên\_phương\_thức (danh sách các tham số)

Ví dụ: Gọi phương thức phép toán đã khai báo ở trên như sau:

```
double kq = 0;  
  
kq = pheptoa(5, 5, "+");//ket qua kq = 10;
```

## 2.6 Mảng (array)

Mảng là tập hợp các biến có cùng kiểu dữ liệu, cùng tên nhưng có chỉ số khác nhau. Trong C#, mảng có chỉ số bắt đầu là 0 và luôn luôn là mảng động.

### 2.6.1 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)

*// Khởi tạo một array với 32 phần tử dữ liệu kiểu int*

```
int n = 32;
```



```
int[] daySo = new int[n]; // dung n de xac dinh kích thước  
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. 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"};
```

### 2.6.2 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 1: In danh sách số nguyên

```
int []so = {2, 5, 3, 4}  
  
// in danh sach cac phan tu trong array  
for (int i = 0; i <so.Length; i++)  
  
    Console.WriteLine(so[i]);  
  
Array.Sort(so);
```

```

        Array.Reverse(so);

        // in danh sach cac phan tu trong array su dung cu phap
foreach

        foreach (int stt in hoso)

        Console.WriteLine(stt);

```

Ví dụ 2: Viết chương trình nhập vào một mảng một chiều và một phần tử x. Tìm xem phần tử x đó có mặt trong mảng một chiều không.

```

class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine("Nhap so phan tu cua mang:
");

        int So_pt = Int32.Parse(Console.ReadLine());
        int[] a = new int[So_pt];
        a = Nhap_mang(So_pt);
        Console.WriteLine("Nhap pt can tim");
        int x = Int32.Parse(Console.ReadLine());
        int kq = Tim(a, x);
        if (kq == -1)
            Console.WriteLine("Phan tu {0} khong co
trong mang", x);
        else

```

```

        Console.WriteLine("Phan tu {0} co trong
mang", x);

        Console.ReadLine();
    }

    public static int[] Nhap_mang(int So_pt)
    {
        int[] a = new int[So_pt];
        for(int i=0; i<So_pt; i++)
        {
            Console.WriteLine("Nhap pt thu {0}:", i);
            a[i] = Int32.Parse(Console.ReadLine());
        }
        return a;
    }

    public static int Tim(int[] a, int n)
    {
        int Kq = -1; // Không tìm thấy
        for (int i = 0; i < a.Length; i++)
            if (a[i] == n)
                Kq = i;

        return Kq;
    }

```

```

    }
}

```

### 2.6.3 Array nhiều chiều

Khai báo:

```

type[,] array-name;

```

Ví dụ:

```

- int[,] myRectArray = new int[2,3]; // mảng 2 hàng 3 cột
- int[,] myRectArray = new int[,]{ {1,2},{3,4},{5,6},{7,8}};
  mảng 4 hàng 2 cột
- string[,] beatleName = { {"Lennon","John"},
                           {"McCartney","Paul"},
                           {"Harrison","George"},
                           {"Starkey","Richard"} };

```

## 2.7 Các toán tử

Toán tử là ký hiệu chỉ ra phép toán nào được thực hiện trên các toán hạng. C# hỗ trợ các kiểu toán tử sau đây:

<i>Loại toán tử</i>	<i>Ký hiệu</i>
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	* -> & []

## 2.8 Xử lý lỗi trong C#

Trong môi trường .NET, lỗi xảy ra khi chương trình đang chạy sẽ phát sinh một ngoại lệ (Exception), nó là đối tượng của lớp System.Exception dùng để chứa thông tin về lỗi lúc thực thi. Với C#, sử dụng Exception với cú pháp try là cách xử lý lỗi đơn giản và hiệu quả.

```
try
{
    // khối lệnh có thể gây ra lỗi
}
```

```

    }

    catch(Exception ex)

    {

        //khởi lệnh bắt và xử lý lỗi

    }

    Finally

    {

        //khởi lệnh kết thúc

    }

```

Ví dụ: Xử lý lỗi không thể chia cho 0

```

int t;

int a, b;

try

{

    t = a/b;

}

catch(Exception ex)

{

    Console.WriteLine("không thể chia cho 0");

}

```

## 2.9 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.

**▼ Câu hỏi (bài tập) củng cố:**

Bài 1: Viết chương trình nhập vào điểm 3 môn học: Toán, Lý, Hóa. Tính và xuất điểm trung bình cộng điểm 3 môn.

Bài 2: Viết chương trình giải phương trình bậc hai.

Bài 3: Viết chương trình nhập vào hai giờ. Tính và in ra tổng và hiệu của hai giờ đó.

Bài 4: Viết chương trình nhập vào 4 số nguyên dương và tìm số lớn nhất trong 4 số nguyên dương đó.

Bài 5: Viết chương trình tính tiền đi taxi biết số km đã đi. Biết rằng: 1km đầu tiên giá 5000 đồng. Từ km thứ 2 đến km thứ 5 giá trung bình là 4500 đồng/km. Từ km thứ 6 trở đi, giá trung bình mỗi km là 3500 đồng/km. Nếu khách hàng đi trên 120 km thì sẽ được giảm 10% trên tổng số tiền tính theo qui định.

Bài 6: Viết chương trình nhập vào hai số nguyên. Sau đó hiển thị ra màn hình menu các lựa chọn:

1. Thực hiện phép cộng
2. Thực hiện phép trừ
3. Thực hiện phép nhân
4. Thực hiện phép chia
5. Thoát

Người dùng nhập vào các số từ 1 đến 4 để thực hiện lựa chọn của mình.

Bài 7: Viết chương trình cho phép nhập vào một số nguyên dương có 2 chữ số. Hãy in ra cách đọc của số nguyên này. Ví dụ: 56: Năm mươi sáu, 15: Mười lăm, 21: Hai mươi mốt...

Bài 8: Viết chương trình tính tiền thuê phòng biết số ngày thuê và loại phòng với qui định của khách sạn như sau:

1. Loại A: 250.000 đ/ngày
2. Loại B: 200.000 đ/ngày
3. Loại C: 150.000 đ/ngày

Nếu thuê quá 10 ngày thì % được giảm trên tổng số tiền (được tính theo giá qui định) là 10%.

Bài 9: Giả sử lương của công nhân được tính theo công thức:

Tiền lương = (mức lương mỗi ngày \* số ngày) + tiền thưởng - tiền phạt.

Biết rằng tiền thưởng bao gồm 2 khoản:

1. Tiền thưởng thêm cho mỗi giờ làm việc tăng ca là 10000 đồng.
2. Tiền thưởng thêm cho mỗi ngày là 15000 đồng kể từ ngày làm việc thứ 25 trở đi nếu số ngày làm việc lớn hơn 24 ngày.

Tiền phạt đối với mỗi ngày đi trễ là 20000 đồng.

Viết chương trình nhập vào mức lương mỗi ngày, số ngày làm việc, số giờ làm việc tăng ca, số ngày đi trễ, tính và in ra tiền lương của công nhân.

Bài 10: Viết chương trình nhập vào 1 số nguyên dương N. Liệt kê ra màn hình tất cả các ước số của số nguyên N. Ví dụ: N=12. Các ước số của N là: 1, 2, 3, 4, 6, 12.

Bài 11: Viết chương trình tìm ước số chung lớn nhất của hai số.

Bài 12: Viết chương trình nhập vào một số nguyên dương n. Tính tổng của dãy số sau:



$$S = 1 + 2 + 3 + \dots + n$$

Bài 13: Viết chương trình nhập vào một số nguyên dương n. Tính tổng của dãy số sau:

$$S = \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \dots + \frac{1}{2^n}$$

Bài 14: Viết chương trình in ra màn hình bảng cửu chương.

Bài 15: Viết chương trình nhập vào một số nguyên dương n. Tính tổng của dãy số theo dạng sau:

$$S = 1^1 + 2^2 + 3^3 + \dots + n^n$$

Bài 16: Viết chương trình nhập vào một mảng một chiều. Tìm phần tử nhỏ nhất và lớn nhất của mảng.

Bài 17: Viết chương trình nhập vào 1 mảng số nguyên có n phần tử. Kiểm tra trong mảng vừa nhập có số dương hay không.

Bài 18: Viết chương trình nhập vào 1 mảng số nguyên có n phần tử. Tính tổng các phần tử dương trong mảng

Bài 19: Viết chương trình nhập vào 1 mảng số nguyên có n phần tử. Đếm số phần tử dương, số phần tử âm, số phần tử bằng 0 trong mảng.

Bài 20: Viết chương trình nhập vào 1 mảng số nguyên có n phần tử. Tính tổng các phần tử âm trong mảng

Bài 21: Viết chương trình nhập vào 1 mảng số nguyên có n phần tử. Tính tổng các phần tử là số chẵn trong mảng

Bài 22: Viết chương trình nhập vào 1 mảng số nguyên có n phần tử, kiểm tra trong mảng vừa nhập có số âm hay không.

## CHƯƠNG 3

### HƯỚNG ĐỐI TƯỢNG TRONG C#

✓ **Mục tiêu học tập:** Sau khi học xong bài này, người học có thể:

Vận dụng kiến thức lập trình hướng đối tượng trong C# để giải quyết bài toán cụ thể

#### 3.1 Lớp đối tượng

Lớp đối tượng được định nghĩa theo cú pháp:

```
class MyClass
{
    private int someField; //thuoc tinh

    public string SomeMethod(bool parameter) //phuong thuc
    {
    }
}
```

Lớp đối tượng chứa các thành viên, thành viên là thuật ngữ được sử dụng để nói đến dữ liệu hoặc hàm (function) được định nghĩa trong lớp. Thuật ngữ hàm được dùng để nói đến bất kỳ thành viên nào có chứa 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 khai báo một biến có kiểu dữ liệu lớp thì xem như có một biến có thể chứa tham chiếu đến một thể hiện của lớp đối tượng đó và cũng cần phải khởi tạo ra đối tượng bằng cách dùng toán tử new.

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

Có thể khai báo và khởi tạo đối tượng: `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.

### 3.2 Thừa kế 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 MyNewClass : MyClass //lớp MyNewClass kế thừa từ MyClass  
{  
  
    // functions and data members here  
  
}
```

Cũng như một số ngôn ngữ lập trình hướng đối tượng, C# có một số từ khóa truy cập để quy định phạm vi mà 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

<i>Mức truy cập</i>	<i>Mô tả</i>
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

### 3.3 Phương thức (Method)

Phương thức, hàm là tập hợp các lệnh dùng để xử lý thông tin cho 1 mục đích nào đó, thông thường phương thức được tạo ra và được đặt tên để có thể gọi sử dụng thông qua đối tượng của Class. Cú pháp để định nghĩa 1 phương thức được mô tả như sau:

```
<access_modifier> <return_type> <Method_Name> ([list_of_parameter])
{
    <lệnh>;
}
```

Trong đó :

<access\_modifier>: Chính là phạm vi (*còn gọi là tầm vực*) truy xuất của phương thức trong chương trình.

<return\_type>: Là kiểu dữ liệu mà sau khi thi hành, hàm sẽ trả về giá trị thuộc kiểu đã khai báo cho nơi gọi

<method\_name>: Tên của phương thức, hàm. Tên của phương thức khi đặt phải tuân theo quy ước về đặt tên cho danh định trong chương trình.

[list\_of\_parameter]: Chính là danh sách các tham số tượng trưng cho dữ liệu sẽ truyền vào trong hàm để xử lý khi gọi nó.

Lưu ý: Tên của hàm khi đặt phải tuân theo các yêu cầu sau:

- Không được trùng với từ khóa thuộc ngôn ngữ C Sharp
- Không được có khoảng trắng
- Không được bắt đầu bởi ký số
- Chỉ có thể bắt đầu bởi ký tự, dấu gạch dưới hoặc dấu @

Để sử dụng Method đã định nghĩa trong **Class** ta chỉ việc gọi thông qua đối tượng của **Class** là Method đó:

```
Class_name.Method_Name([list_of_parameter]);
```

Ví dụ: Viết 1 chương trình cho phép tính diện tích và chu vi của các hình tròn, hình vuông. Trong đó ta sẽ xây dựng phương thức tính bìnhPhuong để tính bình phương cho các số truyền vào khi gọi hàm này. Chương trình được viết như sau:

```
using System;
using System.Collections.Generic;
using System.Text;
namespace tinhToanCacHinh
{
    class tinhToan{
        public static float binhPhuong(float a){
            return a * a;
        }
    }
    class hìnhTron{
        float _banKinh;
        /// Hàm khởi tạo 1 tham số của lớp hình tròn
        public hìnhTron(float gt){
```

```

        _banKinh = gt;
    }
    public float dienTich(){
        return tinhToan.binhPhuong(_banKinh);
    }
}
class hìnhVuong{
    long _canh;
    /// Hàm khởi tạo 1 tham số của lớp hình vuông
    public hìnhVuong(long gt){
        _canh = gt;
    }
    public float dienTich(){
        return tinhToan.binhPhuong(_canh);
    }
}
class Program
{
    static void Main(string[] args)
    {
        hìnhTron h1 = new hìnhTron(5);
        hìnhVuong h2 = new hìnhVuong(9);
        Console.WriteLine("Diện tích hình tròn {0}",
h1.dienTich());
        Console.WriteLine("Diện tích hình vuông {0}",
h2.dienTich());
        Console.ReadLine();
    }
}

```

```
}  
}
```

### 3.4 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* ở đây được hiểu là 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  
    }  
}
```

### 3.5 Ghi đè 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() 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 MyNewClass: MyBaseClass
{
    public override string VirtualMethod()
    {
        return "Phuong thuc nay duoc dinh nghia de trong MyNewClass";
    }
}
```

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

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



```
obj = new MyDerivedClass();
```

```
obj.VirtualMethod(); // in ra Phương thức này được định nghĩa de trong MyNewClass
```

Ở đoạn mã lệnh trên, chúng ta thấy rằng, việc quyết định phiên bản nào của phương thức VirtualMethod (ở lớp MyBaseClass hay MyNewClass) 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.

Trong lớp đối tượng, các trường dữ liệu hoặc các hàm tính 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.

### 3.6 Gọi phương thức

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 SinhVien
{
    public virtual void HienThi()
    {
        Console.WriteLine("Thông tin chung của sinh viên");
    }
}
```

```

    }

}

class SVIT: SinhVien
{
    public override void Hienthi()
    {
        base.HienThi();

        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.

### 3.7 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)
{
    //command
}

```

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

- Bắt buộc phải có từ khóa truy cập là 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 đè phương thức ToString() để hiển thị số ảo theo
    // dạng thông thường:

    public override string ToString()
```

```

    {
        return (System.String.Format("{0} + {1}", 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 Program
{
    static void Main(string[] args)
    {

```

```

        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 -, có thể sử dụng hai phép toán + và - đối với dữ liệu ComplexNumber một cách trực tiếp.

Kết quả thực thi:

$$a = 10 + 12i$$

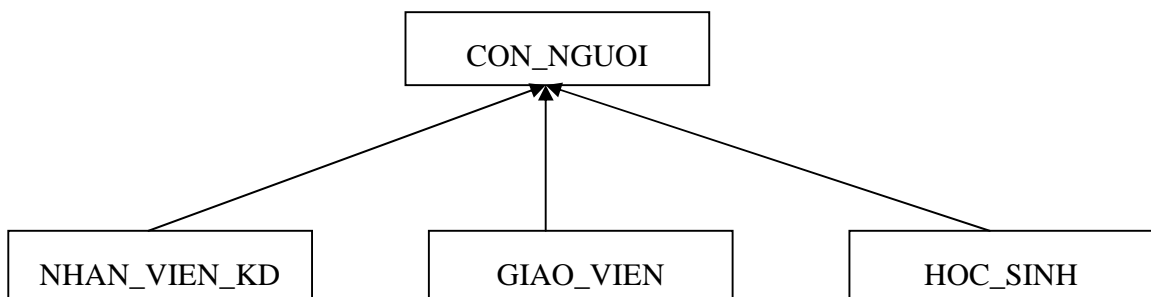
$$b = 8 + 9i$$

$$c = a + b = 18 + 21i$$

$$d = a - b = 2 + 3i$$

**▼ Câu hỏi (bài tập) củng cố:**

1. Xây dựng lớp phân số có các thuộc tính tử và mẫu, các phương thức nhập phân số, xuất phân số, tổng, hiệu, tích và thương của hai phân số.
2. Tương tự như yêu cầu của bài tập 1, hãy nạp chồng các toán hạng cộng (+), trừ (-), nhân (\*) và chia (/) của 2 phân số.
3. Xây dựng 1 lớp hình tròn và phương thức tính diện tích. Sau đó định nghĩa lớp hình trụ thừa kế từ lớp hình tròn đồng thời ghi đè (override) lại phương thức tính diện tích tại lớp hình trụ. Phương thức tính diện tích, khi đối tượng là thể hiện của lớp cơ sở (*lớp hình tròn*) thì phương thức này sẽ sử dụng công thức tính  $\pi \cdot r \cdot r$  nhưng nếu là đối tượng của lớp thừa kế (*lớp hình trụ*) thì phương thức này sẽ sử dụng công thức tính  $2 \cdot \pi \cdot r \cdot (h + r)$ .
4. Cài đặt các lớp trong cây phả hệ:



CON\_NGUOI bao gồm các thông tin: Họ tên, tuổi, giới tính địa chỉ và các chức năng ảo như: nhập thông tin, xuất thông tin, tính là những phương thức ảo.

NHAN\_VIEN\_KD kế thừa từ CON\_NGUOI và các thông tin khác như: Mã nhân viên, lương trên ngày, số ngày công.

GIAO\_VIEN kế thừa từ CON\_NGUOI và các thông tin khác như: Mã giáo viên, lương cơ bản, phụ cấp.

HOC\_SINH kế thừa từ CON\_NGUOI và các thông tin khác như: Mã số, lớp, điểm

toán, điểm lý, hóa.

Anh (chị) hãy xây dựng lại các chức năng từ CON\_NGUOI ứng với từng đối tượng. Đối với chức năng tính:

NHAN\_VIEN\_KD: kết quả là *lương trên ngày \* số ngày công*

GIAO\_VIEN: kết quả là *lương cơ bản\*650 + phụ cấp*

HOC\_SINH: kết quả là *(điểm toán\*2+ điểm lý + điểm hóa)/4*

## CHƯƠNG 4

### LẬP TRÌNH TRÊN WINDOWS FORM

✓ **Mục tiêu học tập:** Sau khi học xong bài này, người học có thể:

Vận dụng kiến thức lập trình Windows Form với C# để giải quyết bài toán cụ thể.

#### 4.1 Windows Form

Sử dụng GUI (Graphical User Interface) làm nền tảng và tổ chức sự kiện cho các đối tượng trên form. Ứng dụng dựa trên một form chứa các thành phần Menu, Toolbar, textbox, Label, Button...

Lớp cơ sở cho các form của ứng dụng là Form

#### **System.Windows.Forms. Form**

Một số thuộc tính, phương thức và sự kiện của Form:

<i>Thuộc tính</i>	<i>Mô tả</i>
Name	Tên của form sử dụng trong project
AcceptButton	Thiết lập button là click khi user nhấn Enter
CancelButton	Thiết lập button là click khi user nhấn Esc
ControlBox	Hiển thị control box trong caption bar
FormBorderStyle	Biên của form: none, single, 3D, sizable



StartPosition	Xác định vị trí xuất hiện của form trên màn hình
Text	Nội dung hiển thị trên title bar
Font	Font cho form và mặc định cho các control
<i>Phương thức</i>	<i>Mô tả</i>
Close	Đóng form và free resource
Hide	Ẩn form
Show	Hiển thị form đang ẩn
ShowDialog	Nạp form lên màn hình
<i>Sự kiện</i>	<i>Mô tả</i>
Load	Sự kiện đầu tiên khi form chạy lên
Click	Xảy ra khi người dùng nhấn chuột lên form
KeyPress	Xảy ra khi 1 phím được nhấn
Resize	Xảy ra khi kích thước form thay đổi

## 4.2 Control

Control là một thành phần cơ bản trên form bao gồm có các thành phần như thuộc tính, phương thức và sự kiện.

Tất cả các control chứa trong namespace: `System.Windows.Forms`

Một số thuộc tính của control

<i>Thuộc tính</i>	<i>Mô tả</i>
BackColor	Màu nền của control
BackgroundImage	Ảnh nền của control
ForeColor	Màu hiển thị text trên form
Enabled	Xác định khi control trạng thái enable
Focused	Xác định khi control nhận focus
Font	Font hiển thị text trên control
TabIndex	Thứ tự tab của control
TabStop	Nếu true, user có thể sử dụng tab để select control
Text	Text hiển thị trên form
TextAlign	Canh lề text trên control

Visible	Xác định hiển thị control
---------	---------------------------

### 4.3 Các Control thông dụng

#### 4.3.1 Label, TextBox, Button

**Label:** Cung cấp chuỗi thông tin chỉ dẫn, chỉ đọc và được định nghĩa bởi lớp Label dẫn xuất từ Control.

**TextBox:** Là điều khiển được dùng để hiển thị và nhập dữ liệu, cho phép nhập dạng Password.

**Button:** Cho phép cài đặt 1 hành động.

Một số thuộc tính và sự kiện:

##### *Label*

<i>Thuộc tính</i>	<i>Mô tả</i>
Font	Font hiển thị của text
Text	Nội dung text hiển thị
TextAlign	Canh lề text
ForeColor	Màu text
Visible	Trạng thái hiển thị

##### **TextBox**

<i>Thuộc tính</i>	<i>Mô tả</i>
AcceptsReturn	Nếu true: nhấn enter tạo thành dòng mới trong chế độ multiline
Multiline	Nếu true: textbox ở chế độ nhiều dòng, mặc định là false
PasswordChar	Chỉ hiển thị ký tự đại diện cho text
ReadOnly	Nếu True: textbox hiển thị nền xám, và không cho phép nhập liệu, mặc định là False
ScrollBars	Thanh cuộn cho chế độ multiline
<i>Sự kiện</i>	<i>Mô tả</i>
TextChanged	Kích hoạt khi text bị thay đổi, trình xử lý được khởi tạo mặc định khi kích đúp vào textbox trong màn hình design view

### ***Button***

<i>Thuộc tính</i>	<i>Mô tả</i>
Text	Chuỗi hiển thị trên bề mặt button
<i>Event</i>	<i>Mô tả</i>
Click	Xảy ra khi người dùng click chuột vào Button

Ví dụ: Chuyển chữ hoa thành chữ thường khi người dùng nhập liệu vào TextBox



Đoạn mã thể hiện chương trình

```
private void textBox1_TextChanged(object sender,
EventArgs e){
    String text;
    Text = ((TextBox)sender).Text
    ((TextBox)sender).Text = text.ToUpper();
}
```

Hình 7. Ví dụ điều khiển TextBox

#### 4.3.2 ListBox

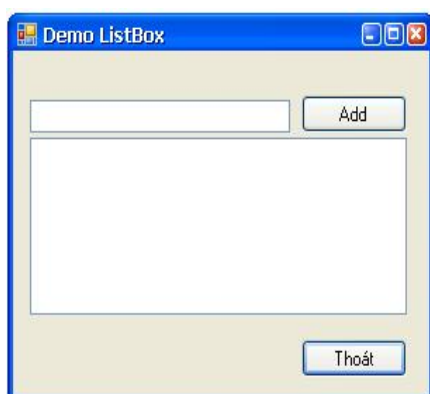
Cung cấp một danh sách các phần tử (item) cho phép người dùng chọn và cho phép hiển thị thanh cuộn (scroll) nếu các item vượt quá vùng thể hiện của ListBox.

Một số thuộc tính, phương thức và sự kiện

<i>Thuộc tính</i>	<i>Mô tả</i>
Items	Cho phép thiết lập phần tử trong điều khiển
Multicolumn	Trình bày danh sách phần tử trên điều khiển có nhiều cột
SelectedItem	Ứng với giá trị phần tử được chọn
SelectedIndex	Lấy giá trị chỉ mục của phần tử đang chọn
Sorted	Sắp xếp tăng giảm tương ứng với true/false
SelectionMode	Cho phép chọn nhiều phần tử

<i>Phương thức</i>	<i>Mô tả</i>
Add	Thêm chuỗi hay đối tượng vào điều khiển
RemoveAt	Xóa phần tử trong điều khiển
<i>Sự kiện</i>	<i>Mô tả</i>
Mouse Click	Xảy ra khi người dùng click chuột lên điều khiển
SelectedIndexChanged	Xảy ra khi thay đổi chỉ mục của phần tử
SelectedValueChanged	Xảy ra khi thay đổi giá trị của phần tử

Ví dụ: Thêm phần tử vào ListBox từ TextBox



Đoạn mã thể hiện chương trình

```
private void button2_Click(object sender, EventArgs e)
{
    if (listBox1.Items.IndexOf(textBox1.Text) >= 0)
        listBox1.SelectedItem = textBox1.Text;
    else if (textBox1.Text.Length > 0)
        listBox1.Items.Add(textBox1.Text);
}
```

Hình 8. Ví dụ điều khiển ListBox

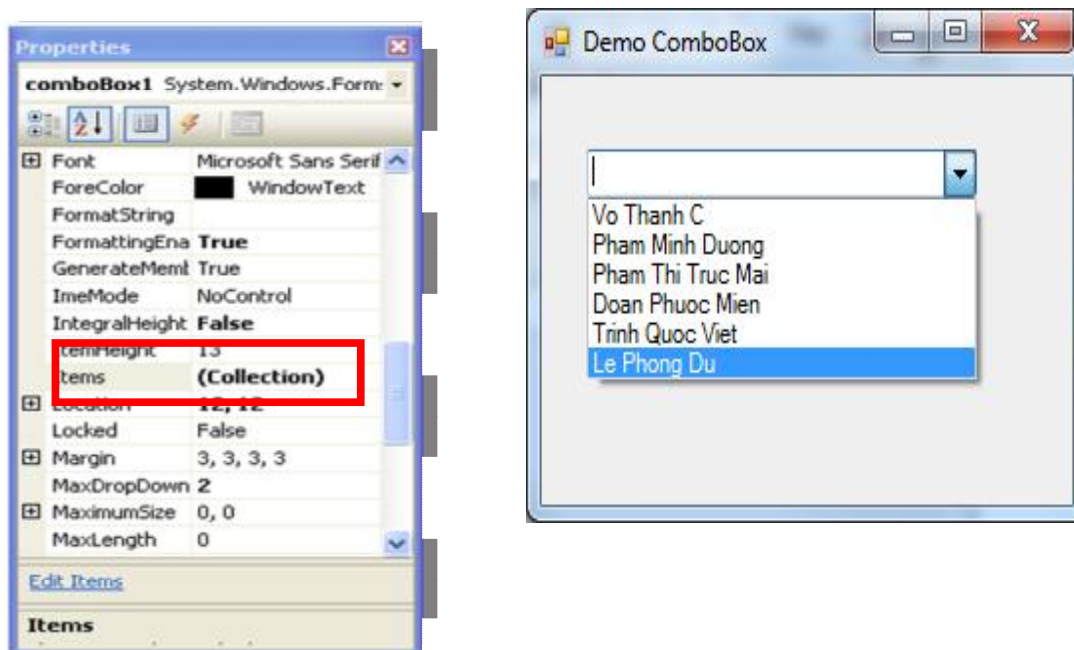
### 4.3.3 ComboBox

Kết hợp TextBox với một danh sách dạng kéo thả và cho phép người dùng kích chọn item trong danh sách đó.

Một số thuộc tính, phương thức và sự kiện

<i>Thuộc tính</i>	<i>Mô tả</i>
Data Source	Tập dữ liệu cho điều khiển
Items	Cho phép thiết lập phần tử trong điều khiển
SelectedItem	Ứng với giá trị phần tử được chọn
SelectedIndex	Lấy giá trị chỉ mục của phần tử đang chọn
SelectedValue	Giá trị phần tử đang chọn
ValueMember	Giá trị ứng với khóa nếu phần tử có khóa và giá trị
<i>Phương thức</i>	<i>Mô tả</i>
Add	Thêm chuỗi hay đối tượng vào điều khiển
<i>Sự kiện</i>	<i>Mô tả</i>
Mouse Click	Xảy ra khi người dùng click chuột lên điều khiển
SelectedIndexChanged	Xảy ra khi thay đổi chỉ mục của phần tử
SelectedValueChanged	Xảy ra khi thay đổi giá trị của phần tử

Ví dụ: Tạo một danh sách bằng ComboBox



Hình 9. Ví dụ điều khiển ComboBox

#### 4.3.4 CheckBox và RadioButton

Checkbox: Đưa ra một giá trị cho trước, người dùng có thể chọn giá trị (Checked = true) và không chọn giá trị (Checked = false) và có thể chọn nhiều lựa chọn.

RadioButton: Cho phép người chọn một tùy chọn một trong số nhóm tùy chọn.

Một số thuộc tính và sự kiện

##### **CheckBox**

<i>Thuộc tính</i>	<i>Mô tả</i>
Appearance	Hình dạng của điều khiển
Checked	Trạng thái của điều khiển
ThreeState	3 trạng thái (chọn, không chọn và không xác định)

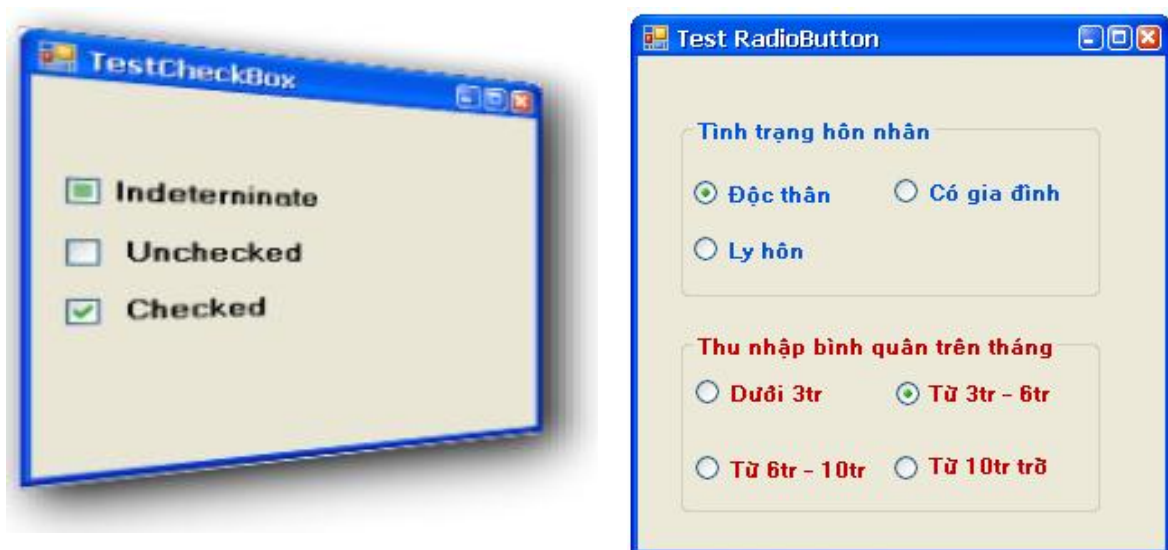


<i>Sự kiện</i>	<i>Mô tả</i>
CheckedChanged	Xảy ra khi click vào điều khiển

### ***RadioButton***

<i>Thuộc tính</i>	<i>Mô tả</i>
Appearance	Hình dạng của điều khiển
Checked	Trạng thái của điều khiển
<i>Sự kiện</i>	<i>Mô tả</i>
CheckedChanged	Xảy ra khi click vào điều khiển

Ví dụ minh họa



Hình 10. Ví dụ điều khiển CheckBox và RadioButton

#### 4.3.5 ListView

ListView là một control dùng để hiển thị một danh sách các item với các biểu tượng và có thể sử dụng một ListView để tạo ra một giao diện giống như cửa sổ bên phải của Windows Explorer.

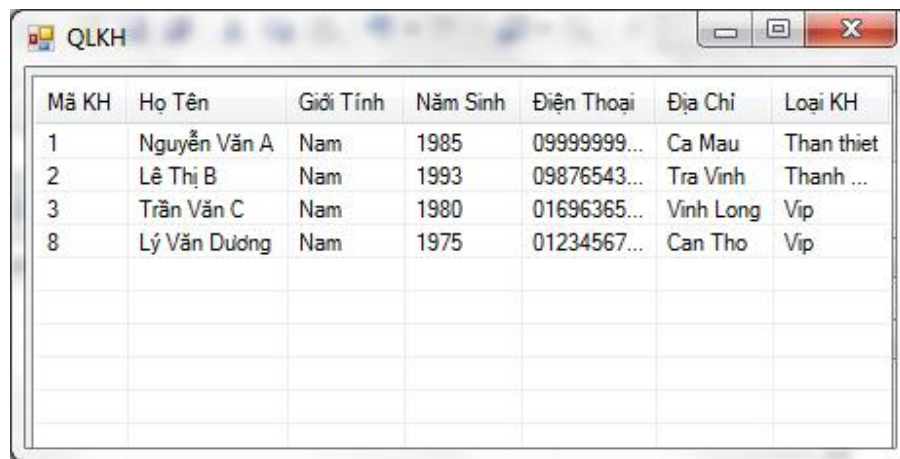
Lớp định nghĩa Item là System.Windows.Forms.ListViewItem, mỗi item trong ListView có các item phụ gọi là subItem, lớp ListViewItem.ListViewSubItem định nghĩa các subItem của ListView.

Một số thuộc tính, phương thức và sự kiện

<i>Thuộc tính</i>	<i>Mô tả</i>
Columns	Số cột của điều khiển
FullRowSelect	Tô khối dòng được chọn
Gridlines	Chế độ hiển thị dạng lưới
Items	Phần tử của điều khiển
MultiSelect	Chọn nhiều phần tử cùng lúc
SmallImageList	Kết hợp với View là SmallIcon
LargeImageList	Kết hợp với View là LargeIcon
View	Chế độ hiển thị của điều khiển (List, Detail, LargeIcon, SmallIcon, Title)
<i>Phương thức</i>	<i>Mô tả</i>
Clear	Xóa tất cả các phần tử của điều khiển

<i>Sự kiện</i>	<i>Mô tả</i>
SelectedIndexChanged	Xảy ra khi thay đổi phần tử trong điều khiển

Ví dụ 1: Tạo danh sách khách hàng trên ListView bằng cách vào thuộc tính Columns để tạo các cột và thuộc tính Items để tạo dữ liệu mẫu.



Hình 11. Ví dụ điều khiển ListView

Ví dụ 2: Có thể tạo các cột cho ListView bằng mã lệnh sau:

```

ColumnHeader columnHeader1 = new ColumnHeader();
ColumnHeader columnHeader2 = new ColumnHeader();
columnHeader1.Text = "Họ tên";
columnHeader2.Text = "Địa chỉ";
columnHeader3.Text = "Điện thoại";
listView1.Columns.Add(columnHeader1);
listView1.Columns.Add(columnHeader2);
listView1.Columns.Add(columnHeader3);

```

Tạo dữ liệu mẫu cho ListView với mã lệnh sau:

```

ListViewItem item1 = new ListViewItem();

ListViewItem.ListViewSubItem subitem1;
ListViewItem.ListViewSubItem subitem2;

subitem1 = new ListViewItem.ListViewSubItem();
subitem2 = new ListViewItem.ListViewSubItem();

item1.Text = "Nguyễn Văn A";

subitem1.Text = "Cà Mau";

subitem2.Text="0999999999";

item1.SubItems.Add(subitem1);

item1.SubItems.Add(subitem2);

listView1.Items.Add(item1);

```

#### 4.3.6 TreeView

Dùng để trình bày danh sách phân tử phân cấp theo từng nút (TreeNode) hình cây. Trong đó mỗi đối tượng TreeNode bao gồm các Node con.

Các thuộc tính, phương thức và sự kiện

<i>Thuộc tính</i>	<i>Mô tả</i>
CheckBox	Xuất hiện checkbox bên cạnh từng node trên control
Nodes	Khai báo số node của control
ShowPlusMinus	Cho phép dấu + và - xuất hiện trên mỗi node
ShowRootLine	Cho phép trình bày node gốc

ImageList	Chứa danh sách hình ảnh từ 0 đến n-1
ImageIndex	Chọn chỉ mục hình ảnh từ 0 đến n-1 ứng với node
<i>Phương thức</i>	<i>Mô tả</i>
CollapseAll	Trình bày tất cả các node trên control
ExpandAll	Thu gọn tất cả các node trên control
<i>Sự kiện</i>	<i>Mô tả</i>
Click	Xảy ra khi người dùng click lên node của control
AfterCheck	Xảy ra sau khi người dùng check vào checkbox trên control
AfterSelect	Xảy ra sau khi người dùng click lên node của control



Hình 12. Ví dụ điều khiển TreeView

#### 4.3.7 GroupBox

Hiển thị một khung bao quanh một nhóm control, khi xóa một GroupBox thì các control chứa trong nó bị xóa theo.

Một số thuộc tính và phương thức

<i>Thuộc tính</i>	<i>Mô tả</i>
Text	Chuỗi xuất hiện trên tựa đề control
BackColor	Màu nền cho control
BackgroundImage	Ảnh nền cho control
Enabled	Ẩn/hiện control
FlatStyle	Hình thức thể hiện control
<i>Phương thức</i>	<i>Mô tả</i>
Add	Thêm điều khiển vào control
Remove	Loại bỏ điều khiển trong control



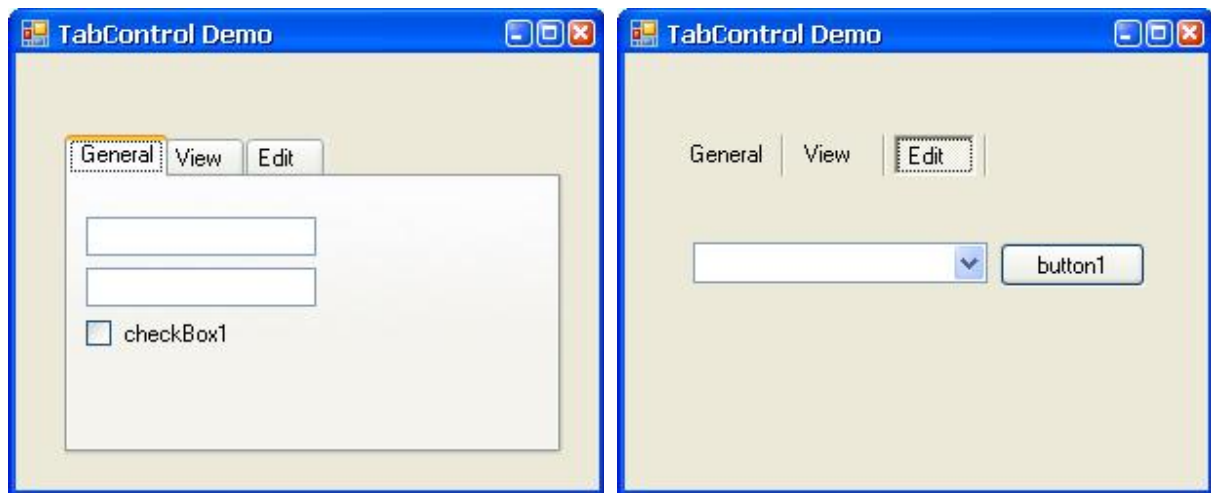
Hình 13. Ví dụ điều khiển GroupBox

#### 4.3.8 Tab Control

Cho phép thể hiện nhiều control trên một form và các control có cùng nhóm chức năng sẽ được tổ chức trong một tab (page).

Một số thuộc tính và phương thức

<i>Thuộc tính</i>	<i>Mô tả</i>
Appearance	Hình thức xuất hiện control
ImageList	Chứa danh sách các ảnh cho phép chọn từng Tab
Enabled	Ẩn/hiện control
Multiline	Cho phép sắp xếp danh sách tab trên nhiều hàng
TabPage	Chứa các điều khiển TabPages
<i>Phương thức</i>	<i>Mô tả</i>
Click	Xảy ra khi người dùng click vào control
Selected	Xảy ra khi người dùng click trên Tab của TabControl



Hình 14. Ví dụ điều khiển TabControl

#### 4.3.9 PictureBox

Sử dụng để hiển thị ảnh dạng bitmap, metafile, icon, JPEG, GIF.

Một số thuộc tính

<i>Thuộc tính</i>	<i>Mô tả</i>
Image	Ảnh cần hiển thị
SizeMode	<p>Normal: Bình thường</p> <p>StretchImage: Căng ra</p> <p>AutoSize: Tự động chỉnh kích thước</p> <p>CenterImage: Canh giữa</p> <p>Zoom: Phóng đại</p>





Hình 15. Ví dụ điều khiển PictureBox

#### 4.3.10 DateTimePicker

Cho phép chọn ngày trong khoảng xác định thông qua giao diện đồ họa dạng lịch.

Một số thuộc tính:

- Format: định dạng hiển thị
  - Long, short, time, custom
- CustomFormat:
  - dd: Hiển thị 2 con số của ngày
  - MM: Hiển thị 2 con số của tháng
  - yyyy: hiển thị 4 con số của năm
- MaxDate: Giá trị ngày lớn nhất
- MinDate: Giá trị ngày nhỏ nhất

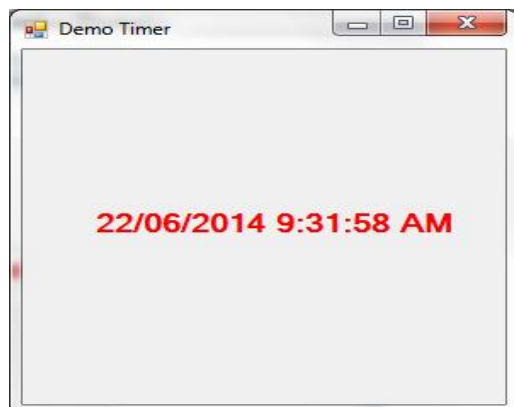
- Value: Giá trị ngày hiện tại đang chọn



Hình 16. Ví dụ điều khiển DateTimePicker

#### 4.3.11 Timer

Bộ định thời gian, thiết lập một khoảng thời gian xác định (interval) và khi hết khoảng thời gian đó Timer sẽ phát sinh sự kiện tick.



Đoạn mã thể hiện chương trình

```
private void timer1_Tick(object sender, EventArgs e)
{
    // Lấy thời gian hệ thống
    DateTime now = DateTime.Now;
    label1.Text = now.ToString();
}
```

Hình 17. Ví dụ điều khiển Timer

#### 4.3.12 ProgressBar

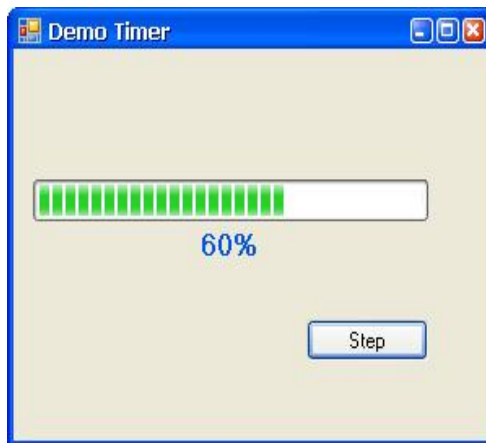
Hiển thị tiến độ thực hiện của một công việc.

Các thuộc tính

- Minimum: Giá trị nhỏ nhất.
- Maximum: Giá trị lớn nhất.
- Step: Số bước tăng khi gọi hàm PerformStep.
- Value: Giá trị hiện tại.
- Style: Kiểu của progress bar.

Phương thức

- PerformStep(): Tăng thêm step.
- Increment(int value): Tăng vị trí hiện tại của tiến độ với giá trị xác định.



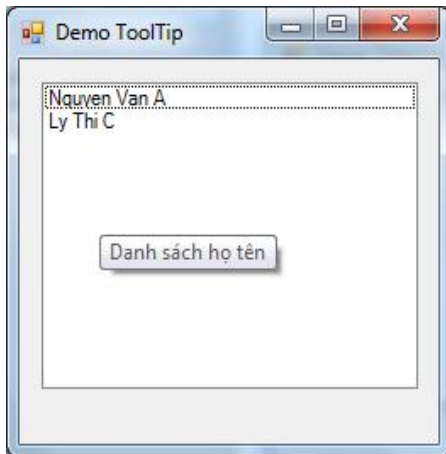
Đoạn mã thể hiện chương trình

```
private void button1_Click(object sender, EventArgs e)
{
    // tăng thêm một bước: value + step
    progressBar1.PerformStep();
    // hiển thị tiến độ * lên label
    label1.Text = progressBar1.Value.ToString() + "%";
}
```

Hình 18. Ví dụ điều khiển ProgressBar

#### 4.3.13 ToolTip

Cung cấp chức năng hiển thị một khung text nhỏ khi người dùng di chuyển chuột vào control bất kỳ.



Đoạn mã thể hiện chương trình

```
private void Form1_Load(object sender, EventArgs e)
{
    // thiết lập tooltip cho listBox1
    tooltip1.SetToolTip(listBox1, "Danh sách họ tên");
}
```

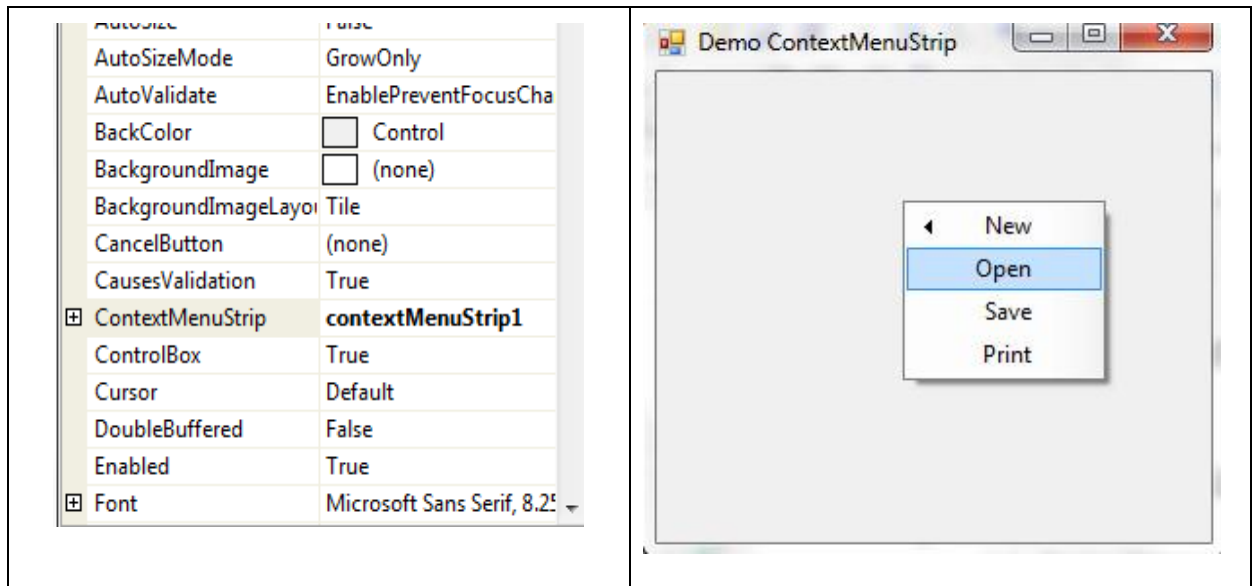
Hình 19. Ví dụ điều khiển ToolTip

#### 4.3.14 ContextMenuStrip

Dùng để thiết kế menu dạng popup cho phép xuất hiện điều khiển các phần tử menu dạng Shortcut mỗi khi người dùng click phải chuột lên form hay control.

Một số thuộc tính

<i>Thuộc tính</i>	<i>Mô tả</i>
Items	Tập danh sách phần tử điều khiển
RightToLeft	Kiểu trình bày menu
ShowImageMargin	Che giấu trình bày phần hình ảnh



Hình 20. Ví dụ điều khiển ContextMenuStrip

#### 4.3.15 ImageList

Cung cấp tập hợp những đối tượng image cho các control khác sử dụng.

Các thuộc tính thường dùng

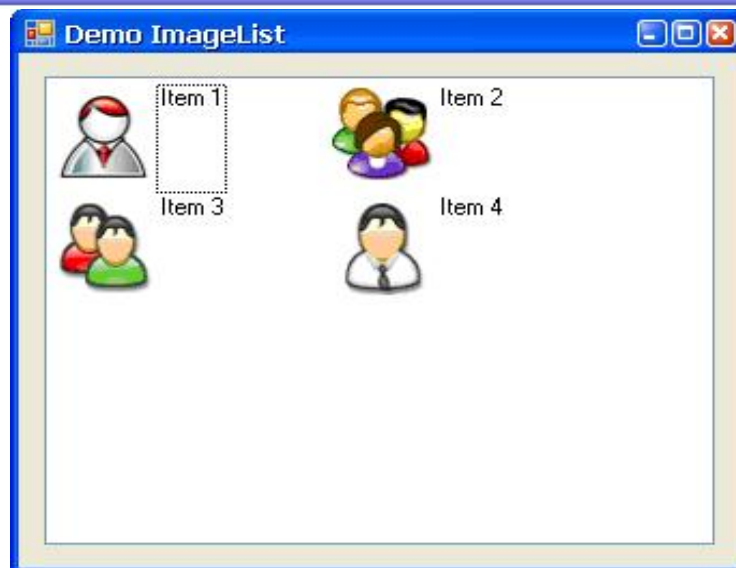
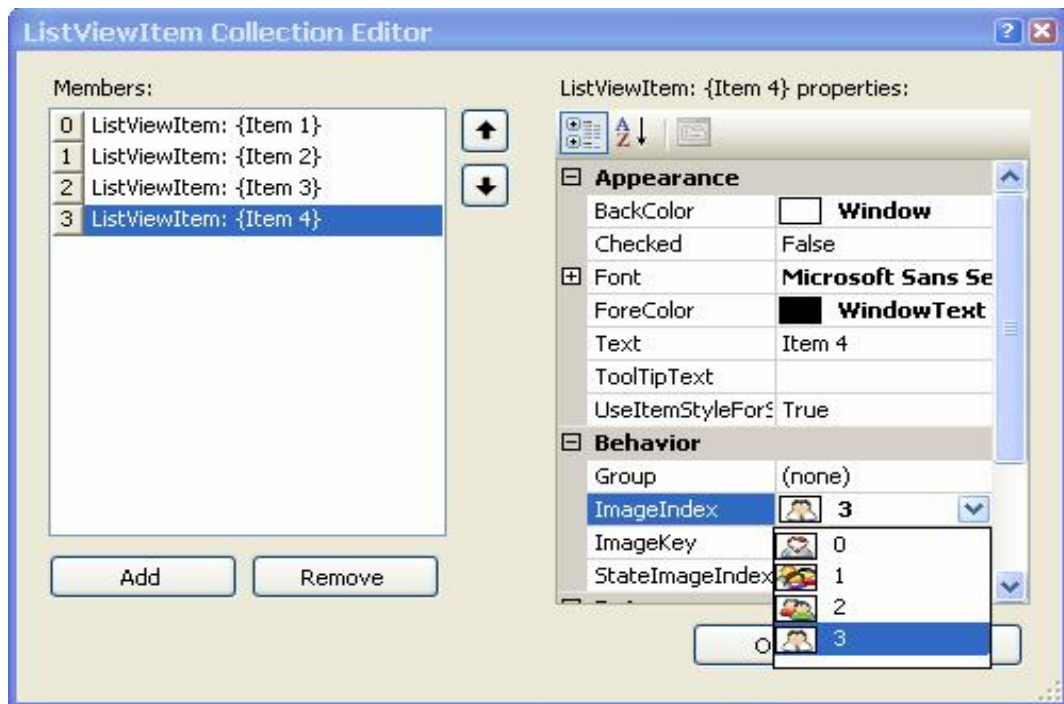
- ColorDepth: Độ sâu của màu.
- Images: Trả về ImageList.ImageCollection.
- ImageSize: Kích thước ảnh.
- TransparentColor: Xác định màu là transparent.

Sử dụng ImageList cho các control

- Khai báo nguồn image là imagelist vừa tạo cho control, thường là thuộc tính

ImageList

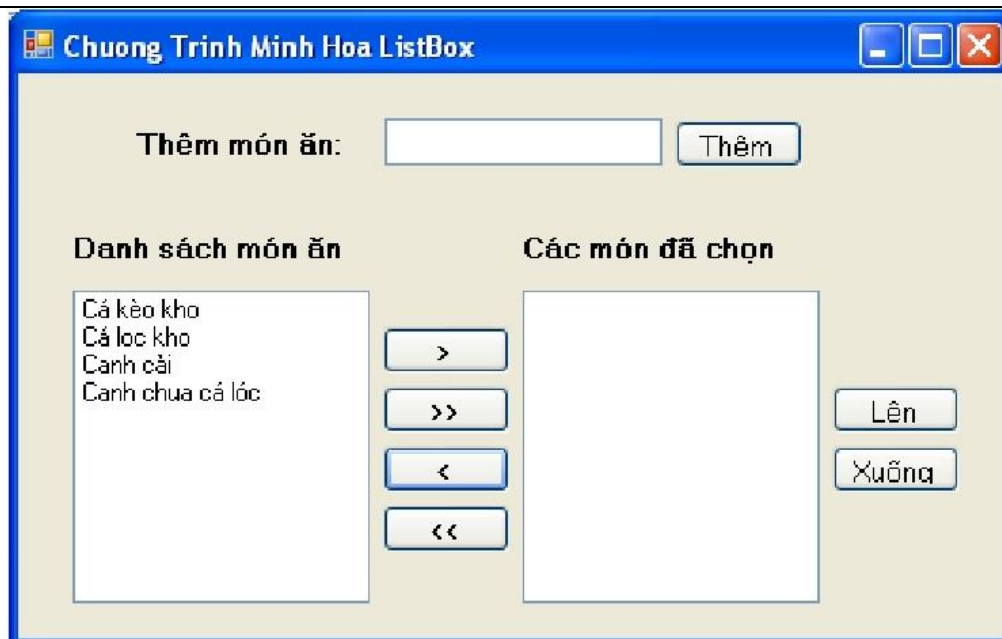
- Thiết lập các item/node với các ImageIndex tương ứng, việc thiết lập có thể ở màn hình design view hoặc code view



Hình 21. Ví dụ điều khiển ImageList

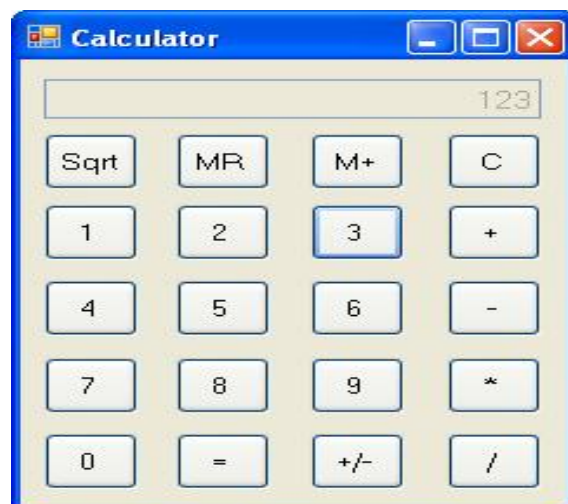
✓ **Câu hỏi (bài tập) củng cố:**

1. Thiết kế và cài đặt chương trình sau:



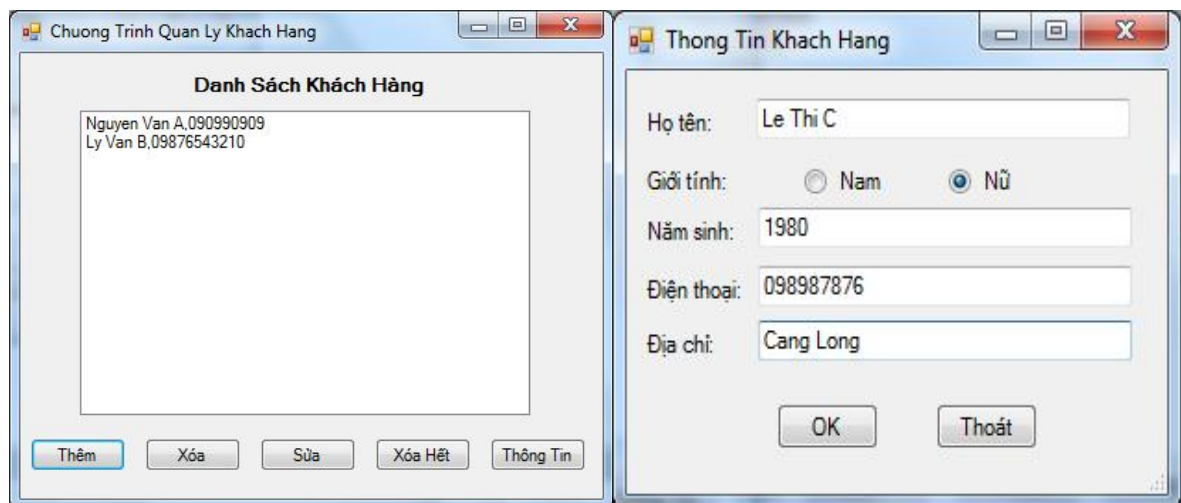
- Cho phép người dùng thêm món ăn mới vào danh sách món ăn
- Các button >, >>: Cho phép chọn một, chọn nhiều món ăn ; button <, << cho phép bỏ một và bỏ hết các món ăn đã chọn.
- Button lên, xuống cho phép thay đổi thứ tự đã chọn.

2. Thiết kế và cài đặt chương trình sau:



- Xây dựng chương trình giả lập một máy tính cá nhân cho phép thực hiện các phép toán: cộng, trừ, nhân, chia, căn bậc hai...
- Ngoài ra, chương trình có tính năng nhớ số, thực hiện các phép toán liên tiếp nhau, đổi dấu một số.
- Chương trình cho phép nhập số từ bàn phím và chuột.

3. Thiết kế và cài đặt chương trình sau:



- Xây dựng chương trình quản lý thông tin khách hàng, cho phép thêm, xóa, sửa thông tin này. Thông tin khách hàng được lưu trữ trong *vùng nhớ*.
- Khi chưa có khách hàng trong danh sách thì khóa các chức năng như xóa, sửa, xem thông tin...
- Khi xem thông tin người dùng không có quyền chỉnh sửa thông tin khách hàng.



## CHƯƠNG 5

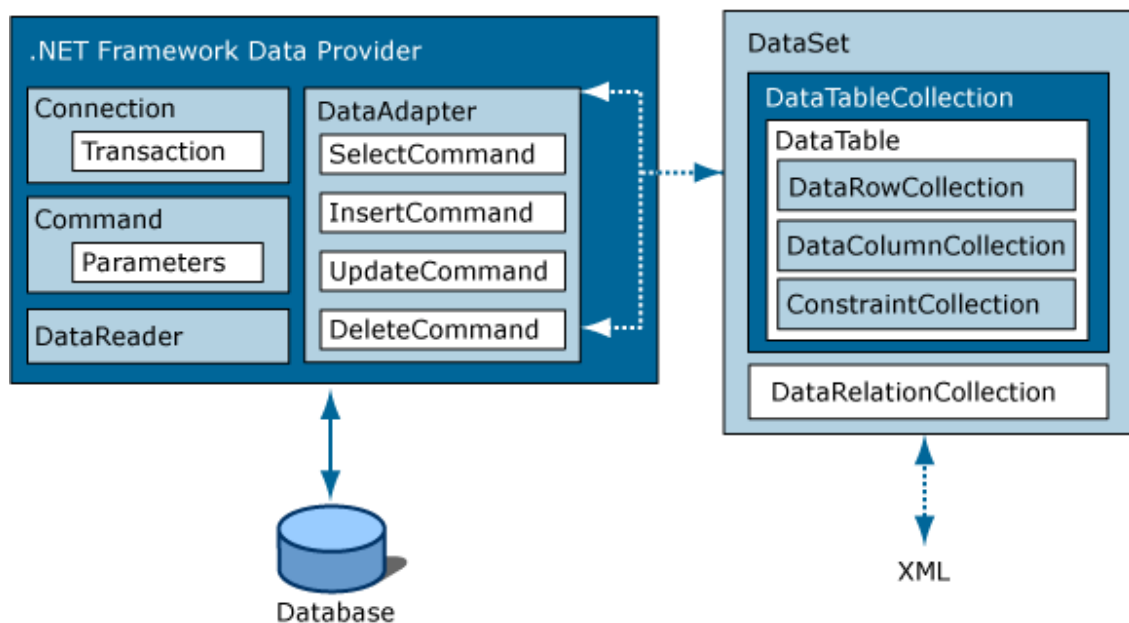
### XỬ LÝ DỮ LIỆU VỚI ADO.NET

✓ **Mục tiêu học tập:** Sau khi học xong bài này, người học có thể:

Vận dụng kiến thức lập trình C# với cơ sở dữ liệu để xây dựng ứng dụng cụ thể.

#### 5.1 Kiến trúc tổng quan của ADO.NET

Kiến trúc của ADO.NET được mô tả như hình 22, bao gồm hai thành phần chính: Thành phần truy cập nguồn dữ liệu và thành phần lưu trữ xử lý dữ liệu.



Hình 22. Kiến trúc ADO.NET

*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 (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 nguồn dữ liệu 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:

<b>Interface Provider</b>	<b>IDbConnection</b>	<b>IDbDataAdapter</b>	<b>IDbCommand</b>	<b>IDbDataReader</b>
<b>SQL Server</b>	SqlConnection	SqlDataAdapter	SqlCommand	SqlDataReader
<b>Oracle Provider</b>	OracleConnection	OracleDataAdapter	OracleCommand	OracleDataReader
<b>OLEDB Provider</b>	OleDbConnection	OleDbDataAdapter	OleDbCommand	OleDbDataReader
<b>ODBC Provider</b>	OdbcConnection	OdbcDataAdapter	OdbcCommand	OdbcDataReader

Để sử dụng data provider nào, chúng ta phải tiến hành khai báo using namespace tương ứng. Ví dụ: using System.Data.SqlClient, sử dụng cho Sql Server.

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, ...

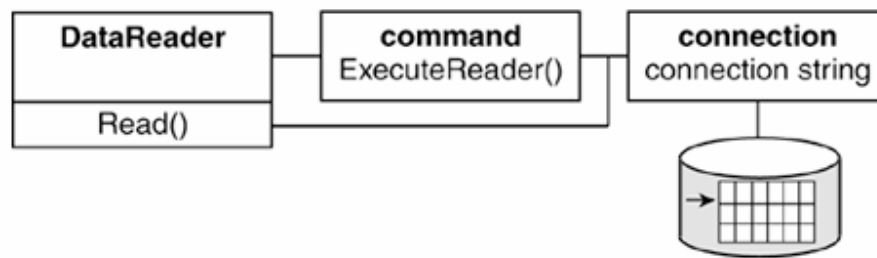
*Thành phần thứ hai* trong kiến trúc ADO.NET là DataSet, được xem như container dùng để lưu trữ đối tượng liên quan đến dữ liệu như bảng (DataTable), mối quan hệ (DataRelation), khung nhìn (DataView). Thành phần này còn được gọi là lớp không kết nối (disconnected layer).

DataSet như là một CSDL thu nhỏ tại máy client, có thể chứa các đối tượng Table, View, Constraint, Relationship 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.

## **5.2 Tổng quan về các mô hình xử lý dữ liệu trong ADO.NET**

### **5.2.1 Mô hình kết nối trực tiếp**

Trong mô hình kết nối của ADO.NET, có một kết nối (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.



Hình 23. Mô hình kết nối trực tiếp

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 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

```

```

string sql = "SELECT HoTen FROM SinhVien";

// (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.

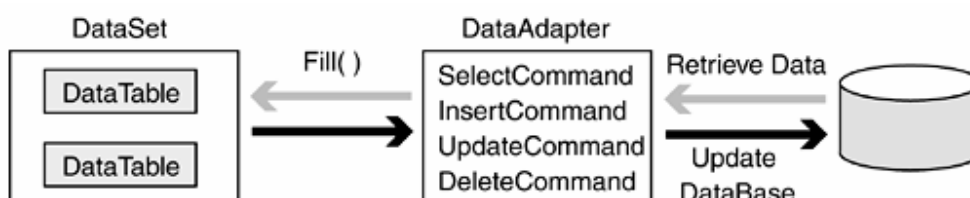
### 5.2.2 Mô hình ngắt kết nối

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 DataAdapter 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.



Hình 24. Mô hình ngắt kết nối

Trong số các phương thức và thuộc tính của DataAdapter thì Fill() và Update() là hai phương thức quan trọng nhất. Phương thức 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=Duong;Initial Catalog=qlsv;
User Id=dmp;Password=tvu;";

// (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.

### 5.3 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.

### 5.3.1 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.

<i>Loại</i>	<i>Tên</i>	<i>Mô tả</i>
<b>Property</b>	ConnectionString	Gán chuỗi kết nối đến data source.
<b>Property</b>	ConnectionTimeout	Khoảng thời gian tối đa tính bằng giây để chờ thực hiện việc kết nối đến data source.
<b>Property</b>	Database	Tên CSDL ứng với Connection hiện tại
<b>Property</b>	State	Trạng thái hiện tại của Connection. Trả về một giá trị kiểu liệt kê (enumeration): Broken, Closed, Connecting, Executing, Fetching, hoặc Open
<b>Method</b>	Open	Mở một Connection.
	Close	Đóng Connection
<b>Method</b>	BeginTransaction	Khởi tạo một database transaction



<b>Method</b>	ChangeDatabase	Thay đổi CSDL hiện tại cho Connection đang mở. Chuỗi mô tả tên CSDL mới được truyền cho phương thức này
<b>Method</b>	CreateCommand	Tạo ra một đối tượng Command ứng với Connection.

### 5.3.1.1 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.

– `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=duong;database=qlnhanvien;uid=dmp;pwd=tvu"
```

hoặc

```
"Server=duong;Database=qlsv;Trusted_Connection=yes"
```

- OleDbConnection sử dụng để kết nối CSDL Access:

```
"Provider=Microsoft.Jet.OLEDB.4.0;DataSource= (1)"
```

Ví dụ:

```
string stConnection = string.Format ("Provider=Microsoft.Jet.OLEDB.4.0;DataSource={0}", @"c:\programfiles\ qlsv.mdb");
```

hoặc

```
string stConnection=string.Format("Provider=Microsoft.Jet.OLEDB.4.0;DataSource={0}", Server.MapPath("/data/qlsv.mdb"));
```

- ODBC:

```
"DSN= (1)" với (1) là Data Source Name (DSN, ví dụ "DSN=qlnv"
```

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=duong; Initial Catalog=qlnv; User Id=dmp; pwd=tvu";  
SqlConnection cn = new SqlConnection(stConnection);  
cn.Open(); //Open connection  
//command  
Cn.Close();//close connection
```

### 5.3.2 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.

### 5.3.2.1 Tạo đối tượng Command

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", "Nguyen Van A");
```

Trong trường hợp ứng dụng có thể phải sử dụng nhiều data provider, 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 p = "System.Data.SqlClient";  
DBProviderFactory fac = DbProviderFactories.GetFactory(p);  
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
```

### 5.3.2.2 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	<p>Thực thi truy vấn hành động và trả về số lượng dòng dữ liệu bị ảnh hưởng bởi truy vấn đó.</p> <p>Ví dụ:</p> <pre>cmd.CommandText = "DELETE SinhVien WHERE MaSinhVien=12";  int sosv = cmd.ExecuteNonQuery();</pre>
ExecuteReader	<p>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.</p> <p>Ví dụ:</p> <pre>cmd.CommandText = "SELECT * FROM SinhVien" + "WHERE YEAR(NgaySinh) &gt; 1990";  SqlDataReader rdr = cmd.ExecuteReader();</pre>

<i>Phương thức</i>	<i>Mô tả</i>
ExecuteScalar	Thực thi một query và trả về giá trị của cột đầu tiên trong dòng đầu tiên của tập kết quả.  Ví dụ:  cmd.CommandText="SELECT COUNT(MaSinhVien) FROM SinhVien";  int sosv = (int)cmd.ExecuteScalar();
ExecuteXmlReader	Chỉ có cho data provider SQL Server. Trả về một đối tượng XmlReader dùng để truy xuất tập dữ liệu.

### 5.3.2.3 Thực thi Stored Procedure (thủ tục lưu trữ) 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 thêm phòng ban (Department)

```
// Khai báo và khởi tạo đối tượng Command, truyền vào tên
thủ tục tương ứng

SqlCommand cmd = new SqlCommand("SP_InsertDepartment", conn);
```

```

// Khai báo kiểu thực thi là Thực thi thủ tục
cmd.CommandType = CommandType.StoredProcedure;

// Khai báo và gán giá trị cho các tham số đầu vào của thủ
tục

// Khai báo tham số thứ nhất @Name - là tên tham số được tạo
trong thủ tục
SqlParameter p = new SqlParameter("@Name", txtName.Text);
cmd.Parameters.Add(p);

// Khởi tạo tham số thứ 2 trong thủ tục là @Description
p = new SqlParameter("@Description",txtDescription.Text);
cmd.Parameters.Add(p);

// Thực thi thủ tục
int count = cmd.ExecuteNonQuery();
if (count > 0)
{
    MessageBox.Show("Thêm mới thành công");
    LoadData();
}
else MessageBox.Show("Không thể thêm mới");

```

Tương tự cho thủ tục xóa phòng ban:

```

SqlCommand cmd = new SqlCommand("SP_DeleteDepartment",
conn);

```

```

cmd.CommandType = CommandType.StoredProcedure;

int id = (int)dgvDeparts.CurrentRow.Cells["clID"].Value;
SqlParameter p = new SqlParameter("@ID", id);
cmd.Parameters.Add(p);

int count = cmd.ExecuteNonQuery();

if (count > 0)
{
    MessageBox.Show("Xóa thành công!");
    LoadData();
}

else MessageBox.Show("Không thể xóa bản ghi hiện thời!");

```

Đoạn mã cho thủ tục sửa phòng ban:

```

SqlCommand cmd = new SqlCommand("SP_UpdateDepartment",
conn);

cmd.CommandType = CommandType.StoredProcedure;

int id = (int)dgvDeparts.CurrentRow.Cells["clID"].Value;
SqlParameter p = new SqlParameter("@ID", id);
cmd.Parameters.Add(p);

p = new SqlParameter("@Name", txtName.Text);
cmd.Parameters.Add(p);

p = new SqlParameter("@Description", txtDescription.Text);
cmd.Parameters.Add(p);

```

```

int count = cmd.ExecuteNonQuery();

if (count > 0)
{
    MessageBox.Show("Sửa thành công!");
    LoadData();
}
else MessageBox.Show("Không sửa được!");

```

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.

Phần code của một số Stored Procedure là như sau:

```

/* Thủ tục thêm mới phòng ban*/
CREATE PROC [dbo].[SP_InsertDepartment]
(
    @Name nvarchar(250),
    @Description nvarchar(250)
)
AS

```



```

INSERT INTO Departments
VALUES(@Name,@Description);

/* Thủ xóa một phòng ban*/
CREATE PROC [dbo].[SP_DeleteDepartment]
(
    @ID int
)
AS
DELETE Departments
WHERE DepartmentID= @ID;

/* Thủ tục sửa thông tin 1 phòng ban*/
CREATE PROC [dbo].[SP_UpdateDepartment]
(
    @ID int,
    @Name nvarchar(250),
    @Description nvarchar(250)
)
AS
UPDATE Departments
SET DepartmentName = @Name,
Description = @Description
WHERE DepartmentID= @ID;

```

#### 5.3.2.4 Sử dụng tham số trong các lệnh 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 tham số (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", "Nguyen Van A");

hoặc

int intMaSinhVien = 12;

string stHoTen = "Nguyen Van A";

sql = string.Format("INSERT INTO SinhVien (MaSinhVien,
HoTen)VALUES (intMaSinhVien, stHoTen);

SqlCommand cmd = new SqlCommand(sql, conn);
```

#### 5.3.3 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 thành phần quan trọng của interface này.

### 5.3.3.1 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) < 1980";

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

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(); // kiểm tra xem còn tập dữ liệu nào nữa không
}
```

```
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.

### 5.3.3.2 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) = 1990";  
  
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} có kiểu dữ liệu {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

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

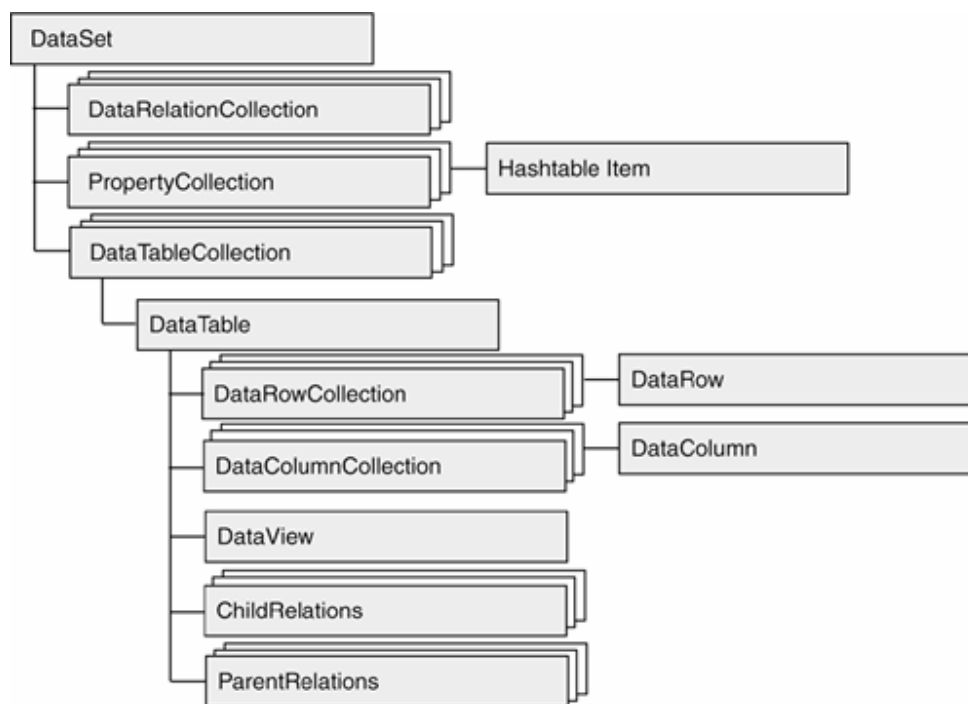
Mô hình ngắt kết nối của ADO.NET dựa trên cơ sở sử dụng đối tượng DataSet như là một vùng nhớ đệm. Một đối tượng DataAdapter làm nhiệm vụ trung gian giữa DataSet và data source (nguồn dữ liệu) để nạp dữ liệu vào vùng nhớ đệm. Sau khi DataAdapter hoàn thành nhiệm vụ nạp dữ liệu, nó sẽ ngắt kết nối khỏi nguồn dữ liệu.

### 5.4.1 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), các thông tin đặc tả như các yêu cầu hợp lệ hóa (validation requirements) cho column trong các DataTable trong DataSet.



Hình 25. Mô hình DataSet

#### 5.4.1.1 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.

#### 5.4.1.2 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.



<i>Phương thức</i>	<i>Mô tả</i>
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
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());
}

```

Đề 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 quan hệ kiểu cha – con (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 khóa chính (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, 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.

#### 5.4.1.3 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()); //
```

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, Modified, hoặc Unchanged.

Ví dụ:

```
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 nhật 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

#### **5.4.1.4 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";
```

### 5.4.2 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.

#### 5.4.2.1 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 "Nguyen Van A"
// 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"]); // "Nguyen Van A"
```

#### 5.4.2.2 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  
  
int nRecs = da.Fill(ds, "nhanvien ")
```

Với một table đang tồn tại, tác dụng của phương thức 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.

### 5.4.3 Cập nhật CSDL bằng DataAdapter

Sau khi DataAdapter đã nạp dữ liệu vào table, connection sẽ được đóng, và các thay đổi 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.

#### 5.4.3.1 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["Makhhoa"] = 12;
drow["tenkhoa"] = "CNTT";
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());
```

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.

#### 5.4.3.2 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 (tranh chấp) khi có nhiều cập nhật được áp dụng: quay lại (rollback) 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.

#### 5.4.4 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 độc 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"]);
    }
}
}

```

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.

Ví dụ:

```
// (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 EnforceConstraint 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.

## 5.5 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, 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.

## 5.6 Thiết kế báo biểu (Report)

### 5.6.1 Giới thiệu

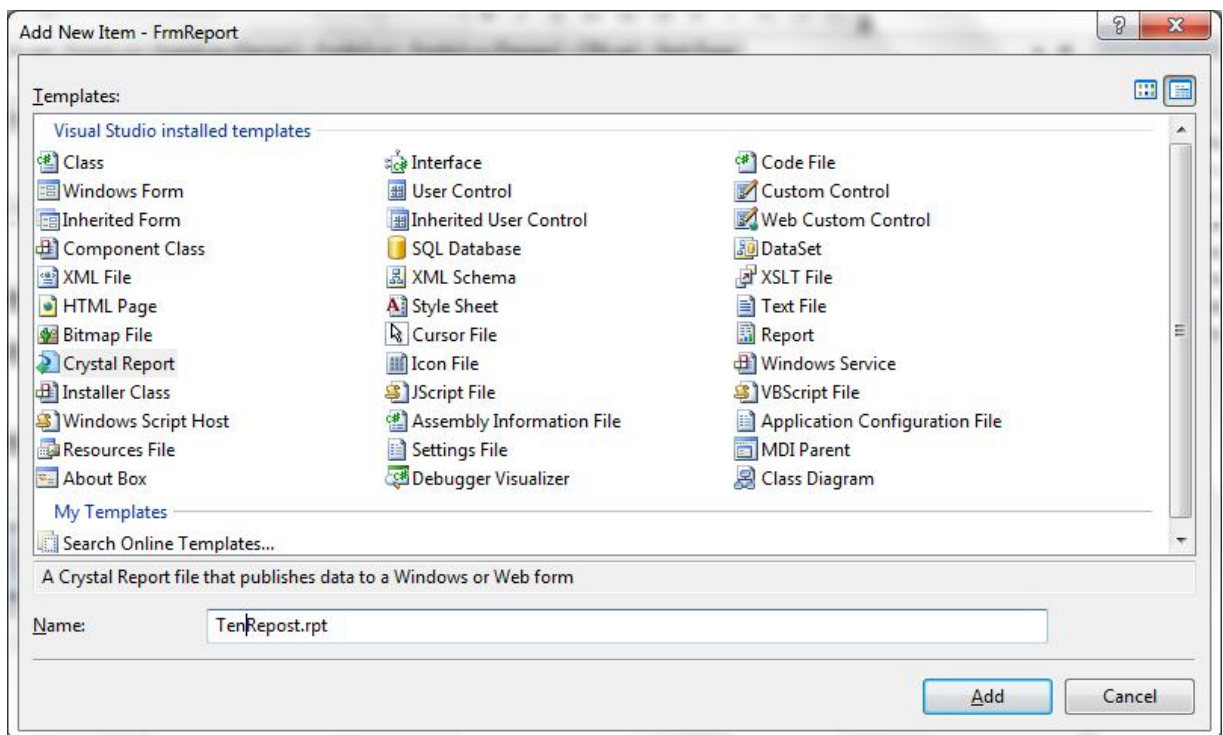
Crystal Report là phần mềm thiết kế báo biểu chuyên nghiệp được tích hợp trong các phiên bản của Visual Studio. Bản thân Crystal Report là một phần mềm tạo báo biểu độc lập với rất nhiều chức năng thiết kế báo biểu và dịch vụ. Người dùng có thể kết nối với nhiều nguồn dữ liệu khác nhau bằng các ODBC Driver. Báo biểu khi tạo ra cũng có thể được lưu trữ thành những file .rpt độc lập, ở dạng có dữ liệu hay không có dữ liệu. Sau đó, file .rpt có thể được chuyển tới người dùng khác và mở bằng Crystal Report hay có thể kết hợp với các ứng dụng viết trên nền tảng .Net.

### 5.6.2 Tạo báo biểu

Vào Project, chọn Add New Item và trong phần templates, chọn Crystal Report để tạo một báo biểu mới.

Trong phần Name của hộp thoại nhập tên báo biểu cần tạo. Sau đó chọn tạo mới báo biểu bằng hai hình thức Wizard (mẫu) và Blank report (tự thiết kế).

Lưu ý rằng chúng ta không thể chuyển sang chế độ preview để xem kết quả thiết kế báo biểu, thay vào đó cần phải sử dụng đối tượng CrystalReportView để hiển thị kết quả.



Hình 26. Thêm Crystal Report vào Project



### 5.6.3 Nguồn dữ liệu cho báo biểu

Một điểm đặc biệt trong Visual Studio.Net là có thể tạo nguồn dữ liệu cho báo biểu từ một DataSet. Sử dụng DataSet làm nguồn dữ liệu cho báo biểu cho phép tạo những báo biểu không cần kết nối trực tiếp với cơ sở dữ liệu.

Để gán nguồn dữ liệu cho báo biểu, chúng ta cần một biến đối tượng để thể hiện báo biểu đã thiết kế. Khi đó DataSet được gán thành nguồn dữ liệu cho báo biểu thông qua phương thức SetDataSource của biến đối tượng báo biểu đó.

```
//tao DataSet

DataSet ds = new DataSet();

//Tao DataTable

DataTable Kq = new DataTable("ketqua");

//Gan DataTable vào DataSet

Ds.Tables.Add(ketqua);

//khai bao the hien cua bao bieu

Rpt_baobieu bb = new Rpt_baobieu();

bb.SetDataSource(ds);
```

### 5.6.4 Sử dụng CrystalReportViewer để hiển thị báo cáo

Để xuất kết quả cho người dùng xem cần phải có một điều khiển hỗ trợ gọi là CrystalReportViewer.

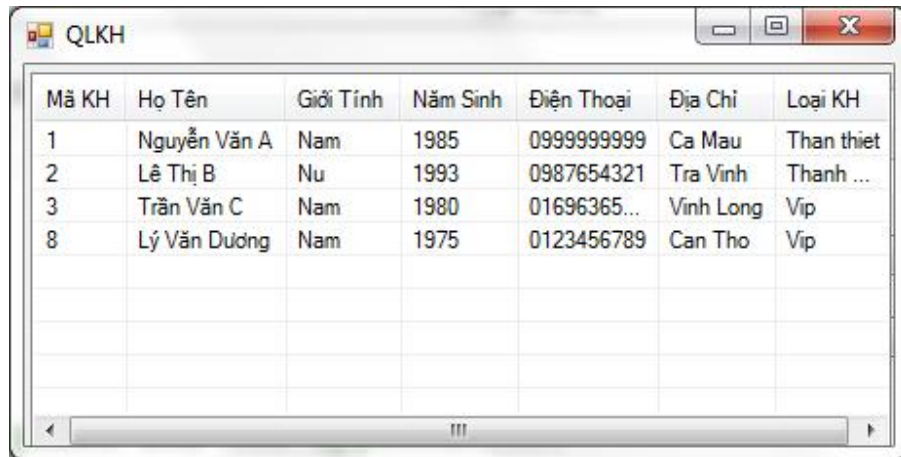
```
//tao doi tuong CrystalreportViewer

CrystalReportViewer crv = new CrystalReportViewer();

Crv.ReportSource = bb;
```


▼ **Câu hỏi (bài tập) củng cố:**

1. Thiết kế cơ sở dữ liệu với tên là *QLKH* gồm có bảng kháchhang(mskh, hoten, gioitinh, namsinh, dienthoai, diachi, loaiKH). Thiết kế và cài đặt chương trình để hiển thị thông tin khách hàng như sau:



Mã KH	Họ Tên	Giới Tính	Năm Sinh	Điện Thoại	Địa Chỉ	Loại KH
1	Nguyễn Văn A	Nam	1985	0999999999	Ca Mau	Than thiết
2	Lê Thị B	Nu	1993	0987654321	Tra Vinh	Thanh ...
3	Trần Văn C	Nam	1980	01696365...	Vinh Long	Vip
8	Lý Văn Dương	Nam	1975	0123456789	Can Tho	Vip

2. Thiết kế và cài đặt chương trình sau:



**Chương Trình Quản Lý Khách Hàng**

**Danh Sách Khách Hàng**

Nguyễn Văn A,090990909  
Ly Van B,09876543210

Thêm Xóa Sửa Xóa Hết Thông Tin

**Thông Tin Khách Hàng**

Họ tên: Lê Thị C

Giới tính: ☐ Nam ☒ Nữ

Năm sinh: 1980

Điện thoại: 098987876

Địa chỉ: Cang Long

OK Thoát

- Thiết kế cơ sở dữ liệu với tên là *QLKH* gồm có bảng kháchhang(mskh, hoten, gioitinh, namsinh, dienthoai, diachi)
- Xây dựng chương trình quản lý thông tin khách hàng, cho phép thêm, xóa, sửa thông tin này. Thông tin khách hàng được lưu trữ trong *cơ sở dữ liệu*.

- Khi chưa có khách hàng trong danh sách thì khóa các chức năng như xóa, sửa, thông tin...
  - Khi xem thông tin người dùng không có quyền chỉnh sửa thông tin khách hàng.
3. Tập tin cơ sở dữ liệu QLSinhVien.mdb
- a. Chương trình kiểm tra kết nối với Access Database

**KHOA – Danh mục khoa**

<i>Field Name</i>	<i>Field Type</i>	<i>Field Size</i>	<i>Format</i>	<i>Description</i>
<b>Ma khoa</b>	Text	2		Mã khoa
Ten khoa	Text	50		Tên khoa

**MON HOC – Danh mục môn học**

<i>Field Name</i>	<i>Field Type</i>	<i>Field Size</i>	<i>Format</i>	<i>Description</i>
<b>Ma mon</b>	Text	2		Mã môn
Ten mon	Text	50		Tên môn

**SINH VIEN – Danh mục sinh viên**

<i>Field Name</i>	<i>Field Type</i>	<i>Field Size</i>	<i>Format</i>	<i>Description</i>
<b>Ma sinh vien</b>	Text	3		Mã sinh viên
Ho sinh vien	Text	30		Họ
Ten sinh vien	Text	20		Tên
Ngay sinh	Date/Time		DD/MM/YYYY	Ngày sinh
Gioi tinh	Yes/No			Yes: Nam; No: Nữ
Dia chi	Text	50		Địa chỉ
Hoc bong	Long Integer			Học bổng
Ma khoa	Text	2		Mã khoa

**KET QUA – Kết quả học tập**

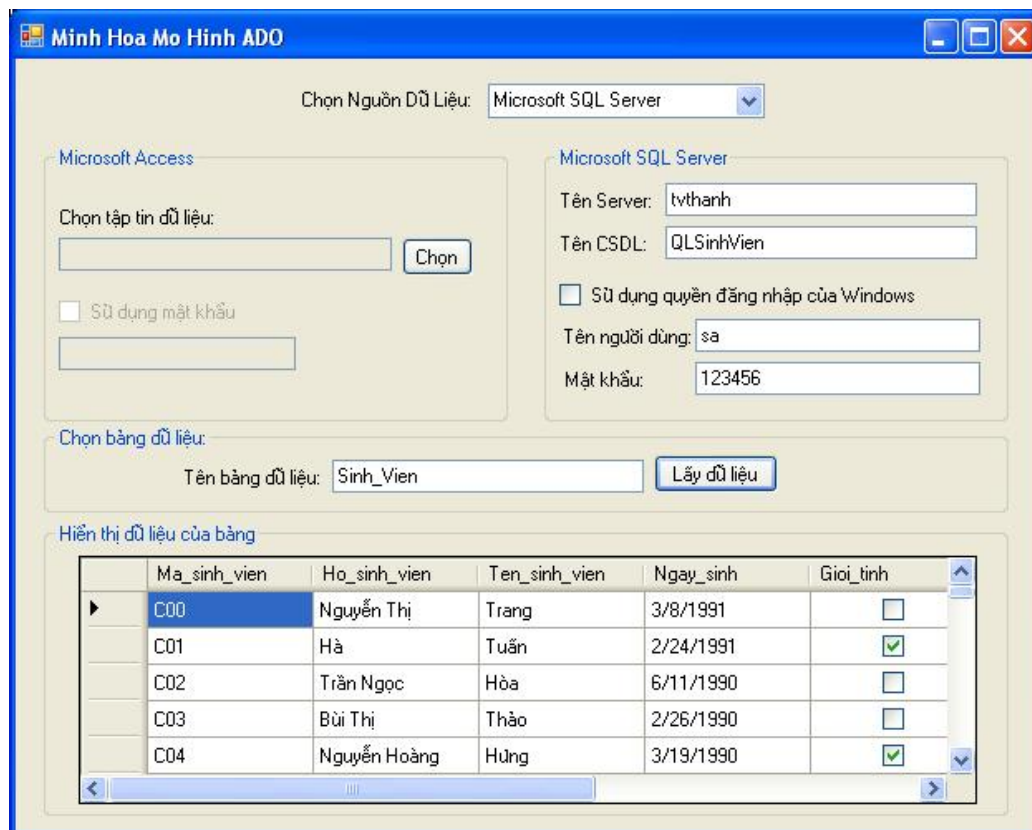
<i>Field Name</i>	<i>Field Type</i>	<i>Field Size</i>	<i>Format</i>	<i>Description</i>
<b>Ma sinh vien</b>	Text	3		Mã sinh viên
<b>Ma mon</b>	Text	2		Mã môn
Diem	Single			Điểm



b. Chương trình kiểm tra kết nối với Sql Server

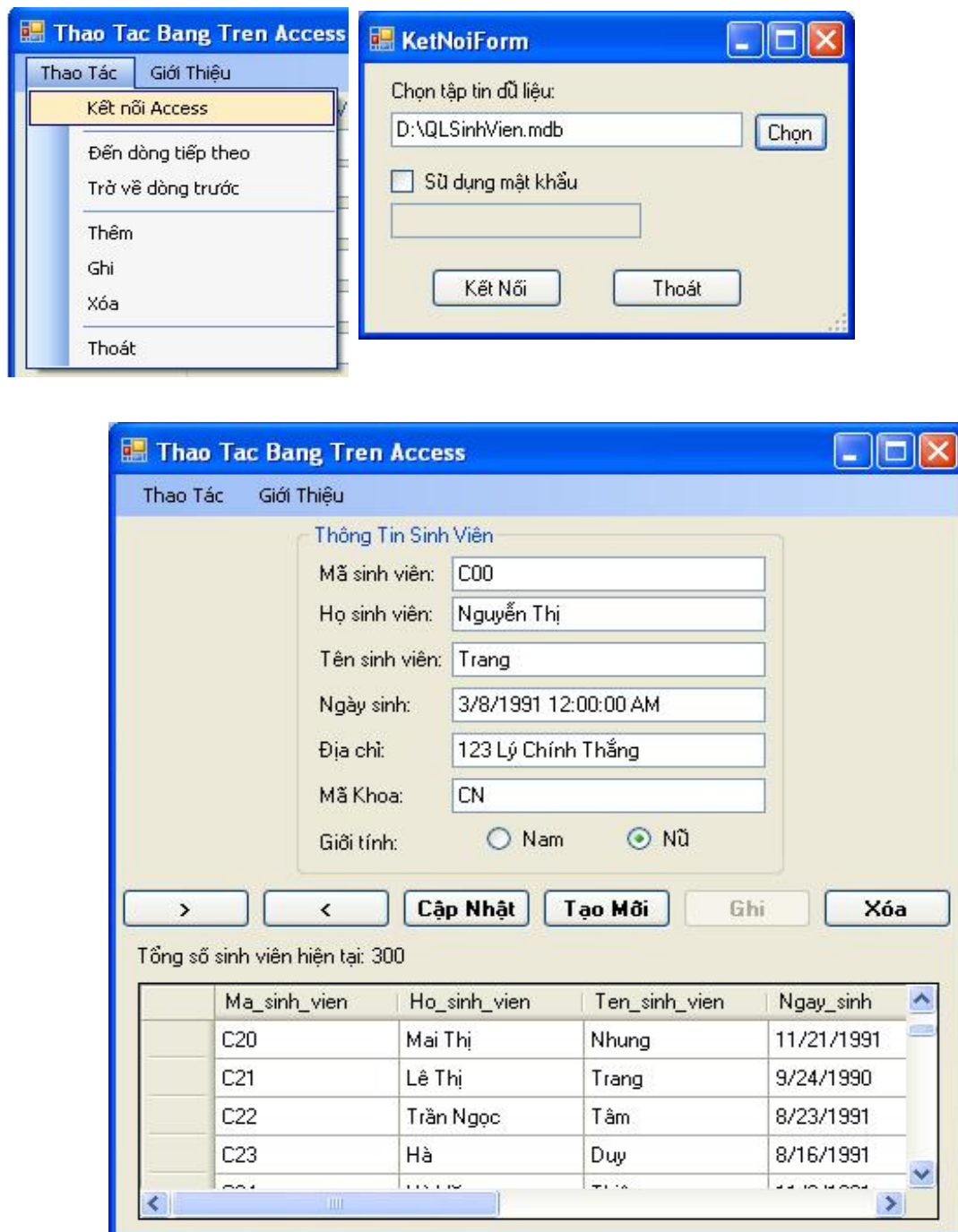


c. Chương trình minh họa mô hình kết nối ADO



4. Chương trình cho phép kết nối với tập tin cơ sở dữ liệu Access thông qua thực đơn “Kết nối Access”. Sau khi kết nối thành công, chương trình cho phép duyệt qua

các dòng dữ liệu trong bảng sinh viên. Chương trình cũng cho phép thêm thông tin một sinh viên mới, cập nhật thông tin của sinh viên cũ và xóa thông tin sinh viên.



5. Như yêu cầu bài 3, chương trình cho phép kết nối với tập tin cơ sở dữ liệu Sql Server.



6. Lọc sinh viên theo khoa và theo môn học



7. Phát triển bài tập 1, thiết kế thêm button in để in thông tin khách hàng theo từng loại.

## TÀI LIỆU THAM KHẢO

Tiếng Việt:

- [1]. Nguyễn Anh Tài (2006), “*Lập trình ứng dụng C#*”, Trung Tâm Tin Học – ĐHKHTN
- [2]. Nguyễn Hoàng Hà, Nguyễn Văn Trung (2008), *Giáo trình C# và ứng dụng*, Đại học Huế.
- [3]. Phạm Hữu Khang (2008), *C# 2005*, Nhà xuất bản Lao Động Xã Hội.

Tiếng Anh:

- [4]. Ian Griffiths (2012), *Programming C# 5.0*, O'Reilly Media
- [5]. Jesse Liberty (2001), *Programing C#*, O'Reilly
- [6]. Christian Nagel, Bill Evjen, Jay Glynn, Karli Watson, Morgan Skinner (2012), *Professional C#*, Wrox
- [7]. Stephen C. Perry (2005), *Core C# and .NET*, Prentice Hall PTR,

Trang Web:

- [8]. Microsoft Corporation, *MSDN*, trang web: <http://msdn.microsoft.com>