

TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT VĨNH LONG
KHOA CÔNG NGHỆ THÔNG TIN



BÀI GIẢNG

LẬP TRÌNH .NET
(DotNET Programming)

(Lưu hành nội bộ)

Vĩnh Long - 2021

MỤC LỤC

Chương 1 TỔNG QUAN VỀ C# (C SHARP)	1
1.1 .NET FRAMEWORK	1
1.2 GIỚI THIỆU VỀ C# (C Sharp)	3
1.2.1 Ngôn ngữ lập trình C#.....	3
1.2.2 Kiểu dữ liệu	6
1.2.3 Biến và hằng	11
1.2.4 Các toán tử.....	13
1.2.5 Lệnh vào khói lệnh	17
1.2.6 Nhập xuất cơ bản trong ứng dụng cửa sổ lệnh C# (Console Application)	17
1.2.7 Bẫy lỗi với try-catch	18
1.3 CÁC CẤU TRÚC ĐIỀU KHIỂN TRONG C#	21
1.3.1 Cấu trúc rẽ nhánh.....	21
1.3.2 Cấu trúc lặp.....	22
1.4 MẢNG (ARRAY)	23
1.4.1 Mảng trong C#.....	23
1.4.2 Mảng 1 chiều	23
1.4.3 Mảng nhiều chiều	25
1.4.4 Mảng là đối tượng	27
1.4.5 Sử dụng foreach trên mảng.....	27
1.5 NAMESPACE	27
1.5.1 Namespace là gì?	27
1.5.2 Khai báo sử dụng Namespace	28
1.5.3 Bí danh cho Namespace	29
1.6 HƯỚNG ĐÓI TƯỢNG TRONG C#.....	29
1.6.1 Lớp và đối tượng	29
1.6.2 Thuộc tính và phương thức	33
1.6.3 Ké thừa	40
1.6.4 Tính đa hình.....	43
CÂU HỎI VÀ BÀI TẬP	46

Chương 2 WINDOWS FORMS VÀ CONTROLS	51
2.1 WINDOWS FORMS	51
2.1.1 Tổng quan về Windows Forms.....	51
2.1.2 Lớp cơ sở: Form class	52
2.1.3 Cấu trúc và thành phần của Form.....	53
2.2 CÁC ĐIỀU KHIỂN (CONTROLS)	57
2.2.1 Hộp công cụ (Toolbox)	57
2.2.2 Các điều khiển cơ bản	58
2.2.3 Các điều khiển nâng cao.....	74
2.2.4 Các điều khiển Date và thành phần Timer	78
2.3 DIALOGBOX	81
2.3.1 Các thành phần và đặc trưng của DialogBox	81
2.3.2 Nhận thông tin từ DialogBox	84
2.3.3 MessageBox.....	84
2.4 PHÁT TRIỂN ỨNG DỤNG MDI (MULTIPLE-DOCUMENT INTERFACE)	86
2.4.1 Ứng dụng MDI là gì?.....	86
2.4.2 Các thành phần của ứng dụng MDI.....	86
2.4.3 Menu	88
2.5 TÍCH HỢP TRỢ GIÚP NGƯỜI DÙNG	90
2.5.1 Vai trò của trợ giúp người dùng (User help) trong các ứng dụng Windows Forms	90
2.5.2 Tạo tập tin .chm	91
2.5.3 Các thành phần và điều khiển trợ giúp người dùng.....	93
CÂU HỎI VÀ BÀI TẬP	97
Chương 3 THAO TÁC DỮ LIỆU VỚI ADO.NET	110
3.1 Kiến trúc tổng quan của ADO.NET	110
3.1.1 ADO.NET	110
3.1.2 Các thành phần của ADO.NET	110
3.2 CÁC MÔ HÌNH TRUY XUẤT DỮ LIỆU	113
3.2.1 Mô hình kết nối.....	113

3.2.2 Mô hình ngắt kết nối.....	114
3.2.3 Lựa chọn giữa mô hình kết nối và mô hình ngắt kết nối.....	116
3.3 LÀM VIỆC VỚI MÔ HÌNH KẾT NỐI TRONG ADO.NET	117
3.3.1 Đối tượng Connection	118
3.3.2 Đối tượng Command	119
3.3.3 Đối tượng DataReader.....	122
3.4 LÀM VIỆC VỚI MÔ HÌNH NGẮT KẾT NỐI: DATASET VÀ DATATABLE	125
3.4.1 DataSet	125
3.4.2 Nạp dữ liệu vào DataSet.....	132
3.4.3 Cập nhật cơ sở dữ liệu bằng DataAdapter.....	134
3.4.4 Định nghĩa các Relationship giữa các DataTable	138
3.4.5 Thao tác dữ liệu với DataGridView	139
3.5 SỬ DỤNG DATABINDING	141
3.5.1 DataBinding là gì?.....	141
3.5.2 Các cách buộc dữ liệu.....	142
3.5.3 Các nguồn dữ liệu của DataBinding.....	142
3.5.4 Ngữ cảnh buộc dữ liệu (BindingContext)	143
CÂU HỎI VÀ BÀI TẬP	145
Chương 4 KẾT XUẤT BÁO BIÊU (REPORT)	166
4.1 GIỚI THIỆU CÁC CÔNG CỤ HỖ TRỢ TẠO BÁO BIÊU.....	166
4.1.1 SAP Crystal Reports.....	166
4.1.2 Các công cụ khác	168
4.2 PHÂN TÍCH YÊU CẦU VÀ THIẾT KẾ BÁO BIÊU.....	169
4.2.1 Phân tích yêu cầu kết xuất báo biểu	169
4.2.2 Phát thảo báo biểu	170
4.2.3 Thiết kế báo biểu dựa trên bản phát thảo	171
4.3 KẾT NỐI BÁO BIÊU VỚI FORMS.....	177
CÂU HỎI VÀ BÀI TẬP	179
Chương 5 HOÀN THIỆN ỨNG DỤNG QUẢN LÝ THÔNG TIN	180
5.1 THIẾT LẬP AN ninh HỆ THỐNG	180

5.1.1 Thiết lập thông tin kết nối cơ sở dữ liệu.....	180
5.1.2 Xây dựng Form xác thực người dùng.....	181
5.1.3 Phân chia chức năng của các nhóm người dùng.....	181
5.1.4 Phân quyền truy xuất trong cơ sở dữ liệu.....	181
5.2 MENU VÀ PHÂN QUYỀN AN NINH TRÊN MENU.....	182
5.2.1 Xây dựng hệ thống menu.....	182
5.2.2 Phân quyền theo Forms	182
5.2.3 Phân quyền theo chức năng	182
5.3 ĐÓNG GÓI VÀ TRIỂN KHAI	182
CÂU HỎI VÀ BÀI TẬP	188

Chương 1

TỔNG QUAN VỀ C# (C SHARP)

1.1 .NET FRAMEWORK

.NET Framework là một khung phát triển phần mềm hỗ trợ cho việc xây dựng và chạy các ứng dụng Windows và các dịch vụ web (web services). .NET Framework là một phần của nền tảng .NET (.NET platform), tập hợp các công nghệ hỗ trợ xây dựng các ứng dụng cho Linux, macOS, Windows, iOS, Android và nhiều hệ điều hành khác.

.NET Framework được thiết kế đáp ứng đầy đủ các mục đích sau:

- Cung cấp môi trường lập trình hướng đối tượng một cách nhất quán cho dù mã đối tượng (object code) được lưu trữ và thực thi cục bộ, thực thi cục bộ nhưng phân tán trên web hay thực thi từ xa.

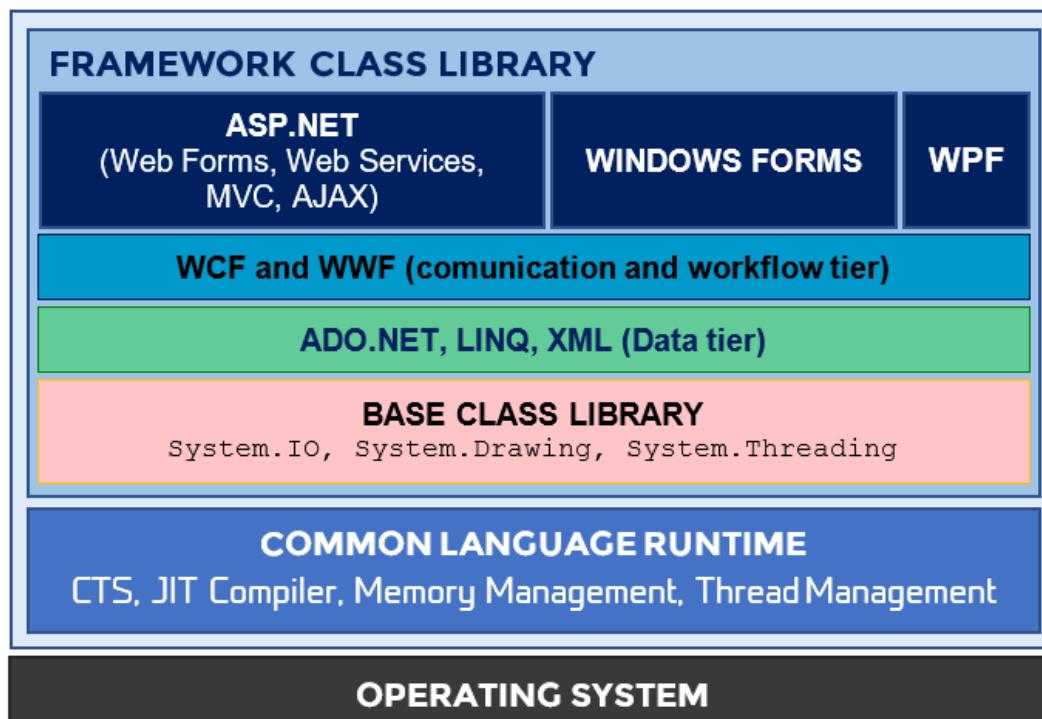
- Cung cấp môi trường thực thi mã lệnh (code-execution environment) giúp giảm thiểu công tác triển khai phần mềm cũng như sự xung đột phiên bản.

- Cung cấp môi trường thực thi mã lệnh thúc đẩy quá trình thực thi mã lệnh an toàn, bao gồm mã lệnh được tạo ra bởi bên thứ ba chưa xác định (unknown) hoặc bán tin cậy (semi-trusted).

- Cung cấp môi trường thực thi mã lệnh giúp loại bỏ các vấn đề hiệu năng của môi trường thông dịch (intercepted) và môi trường kịch bản (scripted).

- Giúp nhà phát triển trải nghiệm sự nhất quán xuyên suốt nhiều loại ứng dụng khác nhau, như ứng dụng Windows và ứng dụng Web.

- Xây dựng tất cả giao tiếp chuẩn công nghiệp đảm bảo mã lệnh tạo ra dựa trên .NET Framework có thể tích hợp với bất kỳ mã lệnh nào khác.



Hình 1.1: Kiến trúc cơ bản .NET Framework

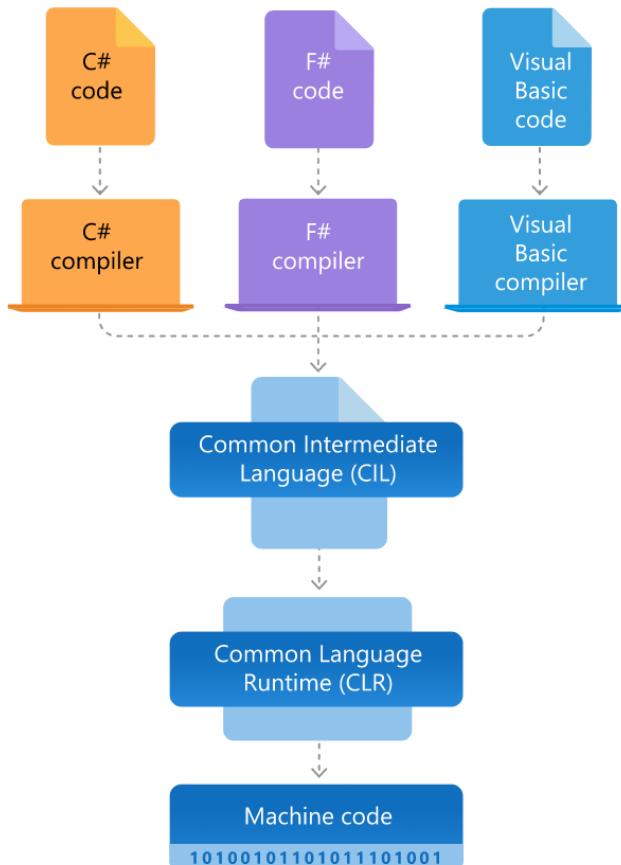
Hai thành phần chính của .NET Framework là khối thực thi ngôn ngữ chung (Common Language Runtime, CLR) và thư viện lớp .NET Framework (.NET Framework Class Library, FCL). Trong đó:

- **CLR** là một công cụ thực thi xử lý các ứng dụng đang chạy. Nó cung cấp các dịch vụ giống như quản lý luồng (thread management), thu gom rác thải, an toàn kiểu, xử lý ngoại lệ và nhiều dịch vụ khác.

- **FCL** cung cấp một tập các API và các kiểu cho các chức năng dùng chung. Nó cung cấp các kiểu dữ liệu cho chuỗi (strings), ngày tháng (dates), số (numbers)... FCL bao gồm các API đọc và ghi tập tin (reading and writing files), kết nối cơ sở dữ liệu (connecting to databases), vẽ (drawing) và nhiều API khác.

Các ứng dụng .NET được viết bởi các ngôn ngữ lập trình như C#, F# hay Visual Basic. Mã lệnh được biên dịch thành một ngôn ngữ trung gian thông dụng bất khả tri (Common Intermediate Language, CIL). Mã lệnh biên dịch được lưu trữ ở dạng hợp ngữ trong các tập hợp – các tập tin với phần mở rộng là .dll hoặc .exe.

Khi một ứng dụng chạy, CLR nhận tập hợp và sử dụng trình biên dịch JIT (Just-in-Time) để chuyển nó thành mã máy (machine code) và có thể thực thi trên kiến trúc máy tính cụ thể mà nó đang chạy.



Hình 1.2: Ứng dụng .NET được biên dịch thành ngôn ngữ máy khi chạy

1.2 GIỚI THIỆU VỀ C# (C Sharp)

1.2.1 Ngôn ngữ lập trình C#

a/- Đặc trưng của ngôn ngữ lập trình C#

C# (phát âm “C Sharp” hoặc “C thăng”) là ngôn ngữ lập trình hướng đối tượng được phát triển bởi Microsoft và là khởi đầu cho kế hoạch .NET của họ. C# được thiết kế cho việc xây dựng các ứng dụng đa dạng chạy trên nền .NET Framework.

C# là ngôn ngữ lập trình khá đơn giản với hơn tám mươi từ khóa và hơn mươi kiểu dữ liệu được dựng sẵn, có ý nghĩa to lớn khi nó cho phép thực thi các khái niệm lập trình hiện đại như hỗ trợ cấu trúc (structure), hỗ trợ thành phần (component), an toàn kiểu (safe-type), hướng đối tượng (object-oriented)…

Ngôn ngữ C# có các đặc trưng sau:

Đơn giản

C# loại bỏ được một vài sự phức tạp và rối rắm của các ngôn ngữ lập trình khác như Java hay C++ vốn được xem là nền tảng phát triển của C#. Ngoài việc khá giống với C/C++ về diện mạo, cú pháp, biểu thức và toán tử, các chức năng của C# cũng được lấy trực tiếp từ ngôn ngữ C/C++. Tuy nhiên chúng đã được cải tiến để làm cho ngôn ngữ trở nên đơn giản hơn. Ví dụ, trong C# đã loại bỏ các macro, template, đa kế thừa hay những lớp cơ sở ảo (virture base class).

Hiện đại

Như đã trình bày ở trên, C# mang đặc tính của một ngôn ngữ lập trình hiện đại như: xử lý ngoại lệ, thực hiện thu gom bộ nhớ tự động, có những kiểu dữ liệu mở rộng, bảo mật mã nguồn.

Hướng đối tượng

C# hỗ trợ tất cả những đặc tính của một ngôn ngữ hướng đối tượng: sự đóng gói (encapsulation), sự kế thừa (inheritance), tính đa hình (polymorphism).

Mạnh mẽ và mềm dẻo

Đối với ngôn ngữ lập trình C#, việc giới hạn chỉ là ở chính bản thân người lập trình. Bởi vì ngôn ngữ này không đặt ra các ràng buộc lên những việc có thể làm. C# được sử dụng cho nhiều dự án khác nhau như tạo ra các ứng dụng xử lý văn bản, ứng dụng đồ họa, xử lý bảng tính, thậm chí tạo ra những trình biên dịch cho các ngôn ngữ khác.

Tuy C# là ngôn ngữ sử dụng giới hạn những từ khóa, phần lớn chúng được dùng để mô tả thông tin nhưng không vì thế mà C# kém đi phần mạnh mẽ so với với các ngôn ngữ lập trình khác.

Hướng module

Mã lệnh C# được viết trong các lớp (Class), các lớp có chứa các phương thức (method) thành viên của chúng. Các lớp và phương thức này có thể được tái sử dụng trong những ứng dụng hay chương trình khác dưới dạng các module.

C# sẽ trở nên phổ biến

Một trong những lý do chính để C# trở nên phổ biến là Microsoft và sự cam kết của .NET. Microsoft muốn C# trở nên phổ biến nên họ đã trực tiếp sử dụng C# trong việc phát triển các sản phẩm của mình, nhiều sản phẩm của Microsoft được được chuyển đổi và viết lại bằng C#. Bằng cách sử dụng ngôn ngữ này, Microsoft đã xác nhận khả năng và sự cần thiết của C# cho những người lập trình.

Bên cạnh đó, Microsoft .NET cũng là một lý do khác để đem đến sự thành công của C#. .NET là một sự thay đổi trong cách xây dựng và thực thi các ứng dụng.

C# cũng trở nên phổ biến nhờ vào những đặc tính của ngôn ngữ này đã được đề cập trong các mục trước: đơn giản, hướng đối tượng, mạnh mẽ và mềm dẻo, hướng module.

b/- C# và các ngôn ngữ lập trình khác

Có rất nhiều ngôn ngữ lập trình khác nhau, trong đó gần gũi với C# nhất có thể kể đến là Visual Basic, C/ C++ và Java. Đâu là sự khác biệt giữa C# và các ngôn ngữ này và tại sao lại có nhiều người chọn C# để học? Có rất nhiều nguyên nhân khác nhau và sau đây sẽ là một số so sánh giúp phần nào giải đáp những thắc mắc này:

- Microsoft nói rằng C# mang đến sức mạnh của ngôn ngữ C++ với sự dễ dàng của ngôn ngữ Visual Basic. Trong đó, C# đã loại bỏ một vài các đặc tính của C++, nhưng bù lại nó tránh được những lỗi thường gặp trong ngôn ngữ C++. C# cũng từ bỏ ý tưởng đa kế thừa như trong C++. Ngoài ra, C# cũng đưa thêm thuộc tính vào trong một lớp giống như trong Visual Basic. Và những thành viên của lớp được gọi duy nhất bằng toán tử “.” khác với C++ có nhiều cách gọi trong các tình huống khác nhau

- Giống như C++ và C#, Java được phát triển dựa trên C. Vì vậy, chúng ta sẽ tìm được nhiều cái mà học từ C# có thể được áp dụng vào Java.

c/- Định danh trong C#

Một định danh là một tên được sử dụng để nhận diện một lớp, biến, hàm hoặc bất kỳ đối tượng nào do người dùng định nghĩa (user-defined).

Định danh trong C# tuân thủ quy tắc của chuẩn Unicode, ngoài trừ dấu gạch dưới “_” được phép sử dụng để bắt đầu một định danh (giống với ngôn ngữ lập trình C truyền thống).

Theo đó, khi định danh trong C# cần tuân thủ các quy tắc sau:

- Chỉ sử dụng các ký tự trong bảng chữ cái (a..z A..Z), các ký tự số (0..9) và các dấu gạch dưới khi định danh.

- Một định danh không được chứa khoảng trắng hoặc các ký tự đặc biệt như ? - + ! @ # % ^ & * () [] { } . ; : " ' / và \.

- Ký tự đầu tiên của một định danh phải là chữ cái hoặc dấu gạch dưới “_”, không sử dụng ký tự số để bắt đầu một định danh.

- Không nên trùng với các từ khóa trong ngôn ngữ lập trình C#. Nếu muốn sử dụng từ khóa của C# như một định danh, cần thêm tiền tố “@” phía trước từ khóa.

- C# phân biệt chữ hoa và chữ thường.

Trong bảng sau đây sẽ trình bày các ví dụ về định danh đúng và sai trong C#.

Bảng 1.1: Ví dụ về định danh trong C#

Định danh	Đúng/Sai
Sinhvien	Đúng
sinhvien	Đúng
_point	Đúng
Product_value	Đúng
@goto	Đúng
3giangvien	Sai, vì ký tự đầu là số
static	Sai, vì trùng từ khóa của C#
Product-value	Sai, vì có chứa ký tự đặc biệt

d/- Từ khóa trong C#

Từ khóa là những từ dành riêng được định nghĩa từ trước bởi người biên soạn ra ngôn ngữ C#. Những từ khóa đó không thể sử dụng như một định danh. Tuy nhiên, nếu muốn sử dụng từ khóa đó như định danh, chúng ta có thể thêm tiền tố @ vào trước từ khóa. Trong C#, một số định danh có ý nghĩa đặc biệt trong phạm vi mã lệnh, chẳng hạn như get và set được gọi là từ khóa theo ngữ cảnh.

Bảng 1.2: Các từ khóa trong C#

Từ khóa dành riêng (Reserved Keywords)						
abstract	as	base	bool	break	byte	case
catch	char	checked	class	const	continue	decimal
default	delegate	do	double	else	enum	event
explicit	extern	false	finally	fixed	float	for
foreach	goto	if	implicit	in	in (generic modifier)	int
interface	internal	is	lock	long	namespace	new
null	object	operator	out	out (generic modifier)	override	params
private	protected	public	readonly	ref	return	sbyte
sealed	short	sizeof	stackalloc	static	string	struct
switch	this	throw	true	try	typeof	unit
ulong	unchecked	unsafe	ushort	using	virtual	void
volatile	while					

Từ khóa ngữ cảnh (Contextual Keywords)						
add	alias	ascending	descending	dynamic	from	get
global	group	into	join	let	orderby	partial (type)
partial (method)	remove	select	set			

e/- Chú thích trong C#

Trong quá trình lập trình, các lập trình viên cần phải chú thích thêm các đoạn mã lệnh do mình phát triển. Chú thích này sẽ giúp cho các thành viên trong nhóm hiểu rõ hơn về ý nghĩa của các đoạn mã lệnh này. Chú thích có ý nghĩa quan trọng khi chương trình đang ở các giai đoạn phát triển, nâng cấp và cập nhật thêm chức năng. Ngoài ra khi thực hiện kiểm thử mức đơn vị (unit test), người kiểm thử cần hiểu chính xác ý nghĩa của các phương thức, các lớp được tạo ra của từng module chương trình.

C# hỗ trợ ba dạng chú thích:

- Chú thích trên dòng: // nội dung chú thích
- Chú thích trên nhiều dòng: /* nội dung chú thích */
- Chú thích sử dụng XML: /// nội dung chú thích

Chú thích XML được sử dụng để tạo ra các tài liệu kỹ thuật nhằm mô tả chi tiết các lớp (class), phương thức (method) thực thi. Để tạo tài liệu XML, chúng ta có thể sử dụng câu lệnh csc thông qua cú pháp sau:

csc /doc:<Tên tập tin .Xml sẽ tạo thành> <Đường dẫn file nguồn chứa .cs>

Ví dụ 1.1:

```
csc /doc:"E:\Projects\Add.xml" "E:\Products\Example\AddNumber.cs"
```

1.2.2 Kiểu dữ liệu

Tương tự như trong C/C++ hay Java, kiểu dữ liệu trong C# được chia thành hai tập chính: kiểu dữ liệu dựng sẵn (built-in types) do ngôn ngữ lập trình cung cấp cho người lập trình, kiểu dữ liệu do người dùng định nghĩa (user-defined types) do người lập trình tạo ra. Một chương trình C# điển hình không chỉ sử dụng các kiểu từ FCL mà còn sử dụng các kiểu do người dùng định nghĩa (user-defined types).

Thông tin được lưu trữ trong một kiểu dữ liệu bao gồm:

- Không gian lưu trữ
- Giá trị lớn nhất và nhỏ nhất có thể biểu diễn
- Các thành phần (phương thức (methods), trường (fields), sự kiện (events))
- Kiểu cơ sở mà nó kế thừa
- Vị trí trong bộ nhớ biến sẽ được phân bổ trong thời gian chạy.
- Các loại hoạt động không được phép của đối tượng.

Trình biên dịch sử dụng thông tin của kiểu để chắc chắn rằng tất cả các thao tác được thực hiện trong mã lệnh là an toàn kiểu. Ví dụ, khi chúng ta định nghĩa một biến kiểu số

nguyên (int) trình biên dịch sẽ cho phép chúng ta sử dụng biến đó với các phép toán cộng và trừ. Nếu chúng ta cố gắng thực hiện các phép toán tương tự trên một biến kiểu luận lý (bool), trình biên dịch sẽ sinh ra lỗi và hiển thị lỗi như trong ví dụ sau:

Ví dụ 1.2:

```
int a = 5;
int b = a + 2; //Đúng
bool test = true;
/* Lỗi. Phép toán '+' không thể áp dụng giữa các toán hạng
"int" và "bool"*/
int c = a + test;
```

a/- Chỉ định kiểu khi khai báo

Khi khai báo một biến (variable) hoặc hằng (constant) trong chương trình, chúng ta hoặc là phải chỉ định rõ kiểu của nó hoặc là sử dụng từ khóa **var** để cho trình biên dịch tự suy luận ra kiểu của nó. Ví dụ sau đây thể hiện cách khai báo biến dùng kiểu dữ liệu số được dựng sẵn và sử dụng các kiểu người dùng định nghĩa phức tạp:

Ví dụ 1.3:

```
// Chỉ khai báo:
float temperature;
string name;
MyClass myClass;
// Khai báo có khởi tạo:
char firstLetter = 'C';
var limit = 3;
int[] source = { 0, 1, 2, 3, 4, 5 };
var query = from item in source
    where item <= limit
    select item;
```

b/- Kiểu dữ liệu dựng sẵn (built-in types)

C# cung cấp một tập chuẩn của các kiểu dữ liệu số dựng sẵn để biểu diễn số nguyên (integers), giá trị dấu chấm động (floating point values), biểu thức luận lý (boolean expression), ký tự văn bản, giá trị thập phân (decimal values), và các kiểu dữ liệu khác. Có cả các kiểu dữ liệu chuỗi và đối tượng dựng sẵn.

Các kiểu dữ liệu dựng sẵn trong C# được phân thành hai loại:

- **Kiểu dữ liệu giá trị:** biến kiểu dữ liệu giá trị lưu trữ trực tiếp một giá trị (kiểu số nguyên, kiểu số thực, kiểu boolean, kiểu ký tự).

- **Kiểu dữ liệu tham chiếu:** biến kiểu tham chiếu lưu trữ tham chiếu đến một giá trị dữ liệu (kiểu object, kiểu string).

Về mặt vật lý, biến của 2 kiểu dữ liệu này được lưu vào 2 vùng nhớ khác nhau của chương trình. Biến kiểu giá trị được lưu giữ kích thước thật trong vùng nhớ đã được cấp

phát là stack. Trong khi địa chỉ của biến kiểu tham chiếu được lưu trong stack còn đối tượng thực sự thì lại được lưu vào vùng nhớ heap.

❖ Kiểu dữ liệu giá trị

Kiểu	Tên	Kiểu trong CTS (Kiểu dữ liệu .NET)	Mô tả	Miền giá trị
Số nguyên	sbyte	System.Sbyte	Số nguyên có dấu 8-bit	$-2^7: 2^7 - 1$
	short	System.Int16	Số nguyên có dấu 16-bit	$-2^{15}: 2^{15} - 1$
	int	System.Int32	Số nguyên có dấu 32-bit	$-2^{31}: 2^{31} - 1$
	long	System.Int64	Số nguyên có dấu 64-bit	$-2^{63}: 2^{63} - 1$
	byte	System.Byte	Số nguyên không dấu 8-bit	$0: 2^8 - 1$
	ushort	System.UInt16	Số nguyên không dấu 16-bit	$0: 2^{16} - 1$
	uint	System.UInt32	Số nguyên không dấu 32-bit	$0: 2^{32} - 1$
	ulong	System.UInt64	Số nguyên không dấu 64-bit	$0: 2^{64} - 1$
Số thực	float	System.Single	Độ chính xác 7 chữ số	$-3.4 \times 10^{38} : 3.4 \times 10^{38}$
	double	System.Double	Độ chính xác 15-16 chữ số	$\pm 5.0 \times 10^{-324} : \pm 1.7 \times 10^{308}$
	decimal	System.Decimal	Độ chính xác 28-29 chữ số	$-7.9 \times 10^{28} : 7.9 \times 10^{28}$
	bool	System.Boolean		true hoặc false
	char	System.Char	Ký tự Unicode 16-bit	

Lưu ý:

- Việc lựa chọn kiểu dữ liệu số nguyên để sử dụng như short, int, long tùy vào độ lớn của giá trị muốn sử dụng.
- Kiểu int với kích thước 4 bytes đủ để lưu trữ các giá trị nguyên phổ biến thường được sử dụng nhiều nhất.
 - Khi có lý do chính đáng để sử dụng kiểu số nguyên không dấu các kiểu số nguyên không dấu được xem xét lựa chọn.
 - Kiểu float, double, và decimal đưa ra nhiều mức độ khác nhau về kích thước cũng như độ chính xác. Với thao tác trên các phân số nhỏ thì kiểu float là thích hợp nhất. Tuy nhiên lưu ý rằng trình biên dịch luôn luôn hiểu bất cứ một số thực nào cũng là một số kiểu double trừ khi chúng ta khai báo rõ ràng. Để gán một số kiểu float thì số phải có ký tự f theo sau.
 - Kiểu bool trong C# không nhận các giá trị nguyên như một số ngôn ngữ (C, C++)
 - Trong C#, các hằng kiểu ký tự được gán bằng cách:
 - + Đóng trong cặp dấu nháy đơn: 'A'
 - + Biểu thị hằng ký tự dưới dạng số thập lục phân: '\u0041'
 - + Hoặc ép kiểu: (char)65

- C# cũng hỗ trợ một số ký tự Escape sau:

Bảng 1.3: Một số ký tự Escape trong C#

Ký tự Escape	Ký tự tương ứng
\'	Dấu nháy đơn
\\"	Dấu nháy đôi
\\"	Ký tự \
\0	Ký tự Null
\a	Ký tự Alert
\b	Ký tự Backspace
\f	Ký tự Form feed
\n	Ký tự xuống dòng
\r	Ký tự về đầu dòng
\t	Ký tự Tab
\v	Ký tự Tab dọc (Vertical Tab)

❖ Kiểu dữ liệu tham chiếu

Ngoài các kiểu dữ liệu giá trị, C# hỗ trợ sẵn các kiểu dữ liệu tham chiếu như object, dynamic, string

Kiểu object

Kiểu object là lớp cơ sở (kiểu dữ liệu gốc) cho tất cả kiểu dữ liệu trong C#. Tương ứng với System.Object trong CTS, tất cả các kiểu dữ liệu khác đều kế thừa từ đây. Các kiểu object có thể được gán giá trị của bất kỳ kiểu, kiểu giá trị, kiểu tham chiếu, kiểu tự định nghĩa (user-defined) khác. Tuy nhiên, trước khi gán các giá trị, nó cần một sự chuyển đổi kiểu.

Khi một kiểu giá trị được chuyển đổi thành kiểu object, nó được gọi là boxing và ngược lại, khi một kiểu object được chuyển đổi thành kiểu giá trị, nó được gọi là unboxing.

Ví dụ 1.4:

```
object obj = 100; // Đây là ví dụ boxing
```

Thuận lợi có được từ kiểu dữ liệu object:

- Sử dụng tham chiếu đối tượng để gắn kết với một đối tượng của bất kỳ kiểu dữ liệu con nào hoặc trong những trường hợp mà mã lệnh phải truy xuất đến những đối tượng chưa rõ kiểu dữ liệu.

- Có cài đặt một số phương thức cơ bản, dùng chung: Equals(), GetHashCode(), GetType() và ToString(). Các lớp do người dùng sử dụng tự định nghĩa có thể cài đặt lại các phương thức này.

Kiểu string

Kiểu dữ liệu `string` trong C# được cung cấp sẵn với nhiều phép toán và cách thức hoạt động, được xem là một trong những kiểu dữ liệu được sử dụng nhiều nhất trong các chương trình. Kiểu dữ liệu `string` C# là một alias cho lớp `System.String` trong CTS.

Một đối tượng `string` được cấp phát vùng nhớ trong heap, và khi gán một biến `string` cho một biến khác chúng ta sẽ có 2 tham chiếu đến cùng một chuỗi trong bộ nhớ. 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.

Ví dụ 1.5: Giả sử trong phương thức Main của chương trình ta có đoạn mã lệnh sau:

```
string s1 = "a string";
string s2 = s1;
Console.WriteLine("Chuoi s1: " + s1);
Console.WriteLine("Chuoi s2: " + s2);
s1 = "another string";
Console.WriteLine("Chuoi s1 hien gio la: " + s1);
Console.WriteLine("Chuoi s2 hien gio la: " + s2);
```

Kết quả thu được ta thấy rằng việc thay đổi giá trị của `s1` không ảnh hưởng gì đến `s2`, ngược với những gì chúng ta trông đợi ở kiểu dữ liệu tham chiếu:

```
Chuoi s1: a string
Chuoi s2: a string
Chuoi s1 hien gio la: another string
Chuoi s2 hien gio la: a string
```

❖ Chuyển đổi giữa các kiểu dữ liệu

Những đối tượng của một kiểu dữ liệu này có thể được chuyển thành những đối tượng của một kiểu dữ liệu khác thông qua cơ chế chuyển đổi kiểu tự động hay ngầm định

- Chuyển đổi ngầm định được thực hiện một cách tự động, trình biên dịch sẽ thực hiện công việc này.

- Chuyển đổi tường minh diễn ra khi thực hiện gán ép một giá trị cho kiểu dữ liệu khác.

Xét ví dụ, chúng ta có thể gán ngầm định một số kiểu `short` (2 bytes) vào một số kiểu `int` (4 bytes) một cách ngầm định

```
short x = 10;
int y = x; // chuyển đổi ngầm định
```

Trình biên dịch sẽ không thực hiện chuyển đổi ngầm định cho chiều ngược lại (từ kiểu `int` sang `short`)

```
short x;
int y = 100;
x = y; // Không biên dịch, lỗi!!!
```

Để không bị lỗi, cần khai báo chuyển đổi kiểu một cách tường minh như sau:

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

Tuy nhiên, việc chuyển đổi từ kiểu dữ liệu kích thước lớn xuống kiểu dữ liệu có kích thước nhỏ hơn có thể sẽ làm mất dữ liệu. Trở lại ví dụ trên, nếu giá trị của số nguyên đó lớn hơn 32.767 thì nó sẽ bị cắt khi chuyển đổi sang kiểu short.

1.2.3 Biến và hằng

a/- Biến trong C#

Biến thực tế không là gì cả, nó chỉ là một cái tên được đặt cho một khu vực lưu trữ mà chương trình có thể thao tác. Nói cách khác, biến là một đơn vị được các ngôn ngữ lập trình tổ chức để lưu trữ và xử lý dữ liệu.

Mỗi biến trong C# có kiểu chỉ định, điều này sẽ giúp nhận biết kích thước cũng như kết cấu vùng nhớ của biến, phạm vi giá trị có thể lưu trữ trong vùng nhớ đó và tập các phép toán có thể thực hiện trên biến.

Trong C# để tạo một biến chúng ta phải khai báo biến và gán cho biến một tên duy nhất theo cú pháp sau:

```
<Tên kiểu> <Tên biến>;
```

Ví dụ 1.6:

```
int a, b, c;
float bien1, _bien2;
string str, Str="";
```

Lưu ý: tên biến cần được đặt theo quy cách định danh trong C#.

Biến có thể được khởi tạo giá trị ngay khi được khai báo, hay nó cũng có thể được gán một giá trị mới tại bất kỳ vị trí nào trong chương trình thông qua các phép gán.

Ví dụ 1.7:

```
static void Main(string[] args)
{
    int a = 9;
    Console.WriteLine("Sau khi khai tao: a={0}", a);
    a = 15;
    Console.WriteLine("Sau khi gan: a={0}", a);
    Console.ReadLine();
}
```

Một điều khác biệt của C# so với C/C++ đó là, các biến trong C# cần phải được gán giá trị xác định trước khi sử dụng. Việc sử dụng (xuất ra màn hình, thực hiện các phép toán...) biến trước khi được khởi tạo giá trị được xem là không hợp lệ trong C#.

Tuy nhiên không nhất thiết lúc nào chúng ta cũng phải khởi tạo biến. Nhưng để dùng được thì bắt buộc phải gán cho chúng một giá trị trước khi có một lệnh nào tham chiếu đến biến đó.

Ví dụ 1.8:

```
static void Main(string[] args)
{
    int a;
    // Lỗi, do a chưa được gán giá trị xác định
    Console.WriteLine("Sau khi khai tao: a={0}", a);
    a = 15;
    Console.WriteLine("Sau khi gan: a={0}", a);
    Console.ReadLine();
}
```

b/- Phạm vi hoạt động của biến

Phạm vi hoạt động của biến là vùng mã lệnh mà trong đó biến có thể được truy xuất, trong cùng phạm vi hoạt động, không được có hai biến trùng tên với nhau.

Phạm vi 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 có tầm hoạt động là trong phạm vi lớp chứa nó.
- Một biến cục bộ sẽ có tầm hoạt động trong khối mà nó được khai báo (trong cặp dấu ngoặc mớc { }).
- Một biến cục bộ được khai báo trong các lệnh lặp for, while... thì sẽ có phạm vi hoạt động trong thân vòng lặp.

Ví dụ 1.9:

```
static void Main(string[] args)
{
    for (int i = 0; i < 10; i++) {
        Console.WriteLine(i);
    } // biến i ra khỏi phạm vi hoạt động
    for(int i=9; i <= 0; i--){
        // biến i ở đây hoàn toàn độc lập với biến i ở trên
        Console.WriteLine(i);
    } // biến i ra khỏi phạm vi hoạt động
    Console.ReadLine();
}
```

c/- Hằng trong C#

Hằng là các giá trị bất biến, được nhận biết tại thời điểm biên dịch và không thay đổi trong suốt vòng đời của chương trình.

Trong C#, hằng được khai báo như biến tuy nhiên sẽ có thêm từ khóa const ở đầu và giá trị cần gán cho hằng là bắt buộc. Cú pháp khai báo hằng trong C# như sau:

```
const <Tên kiểu> <Tên hằng> = <giá trị>;
```

Ví dụ 1.10:

```
const int a = 100;
const char c = 'A';
const float f = 7.5f;
```

Hằng trong C# có các thuộc tính sau:

- Phải được khởi tạo ngay khi nó được khai báo, không được thay đổi giá trị (sử dụng lệnh gán để gán lại giá trị).

- Giá trị của hằng được tính toán tại thời điểm biên dịch, không thể khởi tạo giá trị của hằng bằng một biến.

- Hằng bao giờ cũng là static, tuy nhiên ta không đưa từ khóa static vào khai báo hằng.

1.2.4 Các toán tử

C# cung cấp nhiều toán tử khác nhau, chúng là biểu tượng chỉ định cho trình biên dịch các phép toán (toán học, chỉ mục, luận lý...) thực hiện trong một biểu thức. Chúng ta có thể nạp chồng toán tử để thay đổi ý nghĩa của chúng khi vận dụng trên các kiểu do người dùng định nghĩa.

C# có một tập đa dạng các toán tử được dựng sẵn được phân thành các loại sau:

a/- Toán tử số học (Arithmetic operators)

Bảng sau trình bày tất cả các toán tử số học được hỗ trợ trong C#. Giả sử với hai biến số nguyên A có giá trị là 10 và B có giá trị là 20.

Bảng 1.4: Các toán tử số học trong C#

Toán tử	Miêu tả	Ví dụ
+	Cộng hai toán hạng	A + B = 30
-	Trừ số hạng thứ nhất cho số hạng thứ hai	A - B = -10
*	Nhân hai toán hạng	A * B = 200
/	Chia lấy phần nguyên giữa tử số và mẫu số	B / A = 2
%	Chia lấy phần dư giữa tử số và mẫu số	B % A = 0
++	Toán tử tự tăng, tăng giá trị nguyên thêm một đơn vị	A++ = 11
--	Toán tử tự giảm, giảm giá trị nguyên xuống một đơn vị	A-- = 9

b/- Toán tử quan hệ (Relation operators)

Giả sử có 2 biến số nguyên A và B với các giá trị lần lượt là 10 và 20. Bảng sau biểu diễn tất cả các toán tử quan hệ được C# hỗ trợ:

Bảng 1.5: Các toán tử quan hệ trong C#

Toán tử	Miêu tả	Ví dụ
<code>==</code>	Kiểm tra giá trị của hai toán hạng có bằng nhau không. Nếu có, kết quả trả về là true ngược lại là false	$(A == B)$ (<code>false</code>)
<code>!=</code>	Kiểm tra giá trị của hai toán hạng có bằng nhau không. Nếu không, kết quả trả về là true, ngược lại là false	$(A != B)$ (<code>true</code>)
<code>></code>	Kiểm tra giá trị của toán hạng bên trái có lớn hơn giá trị toán hạng bên phải hay không. Nếu có, kết quả trả về là true, ngược lại là false	$(A > B)$ (<code>false</code>)
<code><</code>	Kiểm tra giá trị của toán hạng bên trái có nhỏ hơn giá trị toán hạng bên phải hay không. Nếu có, kết quả trả về là true, ngược lại là false	$(A < B)$ (<code>true</code>)
<code>>=</code>	Kiểm tra giá trị của toán hạng bên trái có lớn hơn hoặc bằng giá trị toán hạng bên phải hay không. Nếu có, kết quả trả về là true, ngược lại là false	$(A >= B)$ (<code>false</code>)
<code><=</code>	Kiểm tra giá trị của toán hạng bên trái có nhỏ hơn hoặc bằng giá trị toán hạng bên phải hay không. Nếu có, kết quả trả về là true, ngược lại là false	$(A <= B)$ (<code>true</code>)

c/- Toán tử luận lý (logical operators)

Các toán tử luận lý được hỗ trợ trong C# sẽ được thể hiện trong bảng bên dưới với giả thuyết chúng ta có 2 biến kiểu luận lý (bool) với A có giá trị true và B có giá trị false.

Bảng 1.6: Các toán tử luận lý trong C#

Toán tử	Miêu tả	Ví dụ
<code>&&</code>	Toán tử luận lý AND. Nếu cả hai toán hạng có giá trị khác 0 (<code>false</code>) thì điều kiện sẽ trở thành true	$(A \&\& B)$ (<code>false</code>)
<code> </code>	Toán tử luận lý OR. Nếu một trong hai toán hạng có giá trị khác 0 (<code>false</code>) thì điều kiện sẽ trở thành true	$(A B)$ (<code>true</code>)
<code>!</code>	Toán tử luận lý NOT. Dùng để phủ định lại trạng thái luận lý toán hạng của nó. Nếu một điều kiện là true thì sau đó, toán tử luận lý NOT sẽ làm cho nó trở thành false.	$! (A \&\& B)$ (<code>true</code>)

d/- Toán tử trên bit (Bitwise operators)

Các toán tử trên bit làm việc với các bit, thực hiện các tính toán giữa các bit với nhau.

Ví dụ, chúng ta có 2 biến số nguyên A và B có giá trị lần lượt là 60 và 13 với giá trị nhị phân như sau:

$$A = 0011\ 1100$$

$$B = 0000\ 1101$$

Các toán tử trên bit được C# hỗ trợ và các ví dụ minh họa cho từng toán tử cụ thể được thể hiện trong bảng sau:

Bảng 1.7: các toán tử trên bit trong C#

Toán tử	Miêu tả	Ví dụ
&	Toán tử nhị phân AND sao chép một bit vào kết quả nếu nó tồn tại trong chuỗi bit của cả hai số hạng.	$(A \& B) = 12,$ $(0000\ 1100)$
	Toán tử nhị phân OR sao chép một bit vào kết quả nếu nó tồn tại trong một hoặc cả hai toán hạng	$(A B) = 61,$ $(0011\ 1101)$
^	Toán tử nhị phân XOR, sao chép một bit vào kết quả nếu nó chỉ tồn tại trong một toán hạng mà không phải trong cả hai toán hạng	$(A ^ B) = 49,$ $(0011\ 0001)$
~	Toán tử nhị phân đảo bit (0 thành 1 và 1 thành 0)	$(\sim A) = -61,$ $(1100\ 0011\ (số có dấu))$
<<	Phép dịch bit sang trái. Giá trị toán hạng bên trái sẽ dịch chuyển về trái bằng số bit được chỉ định bởi toán hạng bên phải	$A << 2 = 240,$ $(1111\ 0000)$
>>	Phép dịch bit sang phải. Giá trị toán hạng bên trái sẽ dịch chuyển về phải bằng số bit được chỉ định bởi toán hạng bên phải	$A >> 2 = 15,$ $(0000\ 1111)$

e/- Toán tử gán (Assignment operators)

Trong C# có hỗ trợ các toán tử gán như sau:

Bảng 1.8: Một số toán tử gán thông dụng trong C#

Toán tử	Miêu tả	Ví dụ
=	Toán tử gán đơn giản. Gán giá trị từ toán hạng bên phải cho toán hạng bên trái	$C = A + B,$ gán giá trị tổng của A và B cho C
+=	Toán tử cộng gán, sẽ thực hiện cộng toán hạng bên phải vô toán hạng bên trái và sau đó gán kết quả lại cho toán hạng bên trái	$C += A,$ tương đương với $C = C + A$
-=	Toán tử trừ gán, sẽ thực hiện trừ toán hạng bên trái cho số hạng bên phải và sau đó gán kết quả lại cho toán hạng bên trái	$C -= A,$ tương đương với $C = C - A$
*=	Toán tử nhân gán, sẽ thực hiện nhân toán hạng bên phải với toán hạng bên trái rồi gán kết quả lại cho toán hạng bên trái	$C *= A,$ tương đương với $C = C * A$
/=	Toán tử chia gán, sẽ thực hiện chia toán hạng bên trái cho toán hạng bên phải và sau đó gán kết quả lại cho toán hạng bên trái. Lưu ý, toán hạng bên phải khác 0	$C /= A,$ tương đương với $C = C / A$
%=	Toán tử chia lấy dư gán, thực hiện phép chia lấy dư số hạng bên trái cho số hạng bên phải và sau đó gán kết quả lại cho toán hạng bên trái	$C %= A,$ tương đương với $C = C \% A$

Ngoài các toán tử gán nói trên trong C# còn hỗ trợ các toán tử gán &=, |=, ^=, <<=, >>= với các toán tử trên bit.

f/- Các toán tử khác (Miscellaneous operators)

Bảng 1.9: Một số toán tử khác trong C#

Toán tử	Miêu tả	Ví dụ
sizeof()	Trả về kích cỡ của một kiểu dữ liệu	sizeof(int), trả về 4
typeof()	Trả về kiểu của một lớp	typeof(StreamReader);
&	Trả về địa chỉ của một biến	&a; trả về địa chỉ thực sự của biến
*	Trỏ tới một biến	*a; tạo con trỏ với tên là a tới một biến
? :	Biểu thức điều kiện (Conditional Expression), toán tử tam nguyên	C = ĐK ? X : Y Nếu điều kiện là true ? thì giá trị X : nếu không thì Y
is	Xác định đối tượng là một kiểu cụ thể hay không	if(Ford is Car) // Kiểm tra nếu Ford là một đối tượng của lớp Car
as	Ép kiểu không tạo một exception nếu việc ép kiểu thất bại	Object obj = new StringReader("Hello"); StringReader r = obj as StringReader;

g/- Thứ tự ưu tiên của các toán tử

Bảng 1.10: Thứ tự ưu tiên của các toán tử trong C#

Loại	Toán tử	Tính ghép nối
Postfix	() [] -> . ++ --	Trái sang phải
Unary	+ - ! ~ ++ -- (type)* & sizeof	Phải sang trái
Tính nhân	* / %	Trái sang phải
Tính cộng	+ -	Trái sang phải
Dịch chuyển	<< >>	Trái sang phải
Quan hệ	< <= > >=	Trái sang phải
Cân bằng	== !=	Trái sang phải
Phép AND bit	&	Trái sang phải
Phép XOR bit	^	Trái sang phải
Phép OR bit		Trái sang phải
Phép AND logic	&&	Trái sang phải
Phép OR logic		Trái sang phải
Điều kiện	? :	Phải sang trái
Gán	= += -= *= /= %= >>= <<= &= ^= =	Phải sang trái
Dấu phẩy	,	Trái sang phải

Thứ tự ưu tiên toán tử trong C# xác định cách biểu thức được tính toán. Ví dụ, toán tử nhân có quyền ưu tiên hơn toán tử cộng, và nó được thực hiện trước.

Ví dụ, $x = 7 + 3 * 2$; ở đây, x được gán giá trị 13, chứ không phải 20 bởi vì toán tử $*$ có quyền ưu tiên cao hơn toán tử $+$, vì thế đầu tiên nó thực hiện phép nhân $3 * 2$ và sau đó thêm với 7.

Bảng dưới đây liệt kê thứ tự ưu tiên của các toán tử. Các toán tử với quyền ưu tiên cao nhất xuất hiện trên cùng của bảng, và các toán tử có quyền ưu tiên thấp nhất ở bên dưới cùng của bảng. Trong một biểu thức, các toán tử có quyền ưu tiên cao nhất được tính toán đầu tiên.

1.2.5 Lệnh vào khối lệnh

Hoạt động của một chương trình được thực hiện thông qua các câu lệnh. Các hoạt động thường bao gồm khai báo biến, gán giá trị, gọi phương thức, lặp qua các tập hợp, phân nhánh đến một hoặc một khối mã nguồn khác dựa trên điều kiện đã cho. Thứ tự các câu lệnh thực thi trong một chương trình được gọi là luồng điều khiển hay luồng thực thi. Luồng điều khiển có thể thay đổi ở mỗi lần chạy của chương trình, tùy thuộc vào cách chương trình phản ứng với dữ liệu đầu vào mà nó nhận được ở lần chạy đó.

Một lệnh có thể bao gồm một dòng mã lệnh (code) đơn, kết thúc bằng dấu chấm phẩy (;); nhưng nó cũng có thể là một chuỗi các dòng lệnh đơn trong một khối. Một khối lệnh được bao đóng trong cặp dấu ngoặc mớc ({ }) và có thể lồng vào nhau. Xem lại ví dụ 1.9 để thấy rõ hơn về minh họa cho lệnh và khối lệnh trong C#.

Đặc trưng của khối lệnh đó là phạm vi thực hiện trong khối lệnh. Một biến được khai báo trong khối lệnh được gọi là biến cục bộ, chỉ có tính khả thi trong khối lệnh chứa nó.

1.2.6 Nhập xuất cơ bản trong ứng dụng cửa sổ lệnh C# (Console Application)

Với các lập trình viên mới học lập trình C#, luôn phải sử dụng cửa sổ dòng lệnh để thực hiện nhập dữ liệu của chương trình. Để thực hiện điều này thì lập trình viên bắt buộc phải nắm được các phương thức nhập xuất cơ bản của C#.

Trong ứng dụng Console có 3 luồng cơ bản dùng để xử lý vào/ra:

- Nhập dữ liệu chuẩn (Standard in): Dùng để nhận các tham số đầu vào cho chương trình xử lý.

- Xuất dữ liệu chuẩn (Standard out): Dùng hiển thị kết quả đầu ra.

- Thông báo lỗi (Standard err): Dùng hiển thị các thông báo lỗi.

a/- Phương thức xuất dữ liệu

Trong C#, lớp Console trong namespace System (System.Console) dùng để thực hiện các hoạt động trên cửa sổ lệnh. Để hiển thị thông báo ra màn hình ta có thể sử dụng một trong hai phương thức sau:

- Console.WriteLine(): Hiển thị kết quả ra màn hình

- Console.ReadLine(): Hiển thị kết quả ra màn hình và xuống dòng

Chú ý:

- Khi sử dụng phương thức Console.WriteLine(), nếu ta muốn hiển thị thông tin của nhiều biến đầu ra thì ta sử dụng Placeholder để chứa các thông tin về vị trí xuất dữ liệu cũng như là định dạng (format) cho dữ liệu xuất.

- Khi sử dụng phương thức xuất dữ liệu cần phải chú ý các định dạng dữ liệu khi xuất đó chính là các định dạng kiểu số thực, kiểu Datetime.

b/- Phương thức nhập dữ liệu

Trong C#, để đọc dữ liệu từ bàn phím ta sử dụng nhập chuẩn Standard in và thông qua 2 phương thức của lớp Console.

- Console.Read() : Đọc một ký tự từ bàn phím.

- Console.ReadLine() : Đọc một chuỗi ký tự từ bàn phím.

Chú ý: Khi sử dụng hàm đọc dữ liệu Console.ReadLine() thì cần phải sử dụng các phương thức chuyển kiểu để thực hiện chuyển kiểu dữ liệu về giá trị mong muốn.

Ví dụ 1.11:

```
static void Main(string[] args)
{
    int number1, number2;
    Console.Write("Nhập số thứ nhất: ");
    number1 = Convert.ToInt32(Console.ReadLine());
    Console.Write("Nhập số thứ hai: ");
    number2 = Convert.ToInt32(Console.ReadLine());
    Console.WriteLine("Hai số vừa nhập là: {0} và {1}",
    number1, number2);
    Console.ReadLine();
}
```

1.2.7 Bẫy lỗi với try-catch

Lệnh try-catch bao gồm một khối try được theo sau bởi một hay nhiều mệnh đề catch, chỉ định các bộ xử lý cho các ngoại lệ (exception) khác nhau.

Khi một ngoại lệ bắt ra, CLR tìm mệnh đề catch xử lý ngoại lệ tương ứng. Nếu như phương thức đang thực thi không chứa bất kỳ mệnh đề catch nào, CLR tìm phương thức đã gọi phương thức hiện thời, và tiếp tục gọi lên như gọi dạng ngăn xếp. Nếu không có khối catch nào được tìm thấy, CLR sẽ hiển thị tin nhắn ngoại lệ không được xử lý đến người dùng và dừng thực thi chương trình.

Khối try chứa mã lệnh có thể dẫn đến ngoại lệ. Khối này được thực thi cho đến khi một ngoại lệ bắt ra hoặc đến khi nó hoàn thành thành công. Ví dụ sau cho thấy việc thử ép kiểu một đối tượng null có thể bắt ngoại lệ NullReferenceException:

```

object o2 = null;
try
{
    int i2 = (int)o2;    // Error
}

```

Mặc dù mệnh đề catch có thể được dùng ở dạng không đổi số để bẫy bất kỳ loại ngoại lệ nào, nhưng cách dùng này không được khuyến khích. Nói chung, chúng ta chỉ nên bẫy các ngoại lệ mà chúng ta biết cách khôi phục. Vì vậy, chúng ta nên luôn chỉ định một đối tượng làm đối số được dẫn xuất từ lớp System.Exception, ví dụ:

```

catch (InvalidOperationException e)
{
}

```

Chúng ta có thể dùng nhiều hơn một mệnh đề catch riêng biệt trong cùng một lệnh try-catch. Trong trường hợp này, thứ tự của các mệnh đề catch rất quan trọng bởi vì các mệnh đề catch sẽ được kiểm tra theo thứ tự. Bắt những ngoại lệ nhiều riêng biệt hơn trước các ngoại lệ ít riêng biệt hơn. Trình biên dịch sẽ tạo ra lỗi nếu chúng ta sắp xếp các khối catch của mình mà không bao giờ đạt được khối catch sau.

Sử dụng các đối số catch là một phương pháp để lọc các ngoại lệ mà chúng ta muốn xử lý. Chúng ta cũng có thể sử dụng một bộ lọc ngoại lệ để kiểm tra thêm ngoại lệ và quyết định xem liệu có xử lý nó hay không. Nếu bộ lọc ngoại lệ trả về false, thì quá trình tìm kiếm bộ xử lý sẽ tiếp tục.

```

catch (ArgumentException e) when (e.ParamName == "...")
{
}

```

Một lệnh throw có thể được dùng trong khối catch để ném lại ngoại lệ bị bắt bởi lệnh catch. Ví dụ sau đây trích xuất thông tin nguồn từ một ngoại lệ IOException, và sau đó ném ngoại lệ đến phương thức cha.

Ví dụ 1.12:

```

catch (FileNotFoundException e)
{
    // FileNotFoundExceptions are handled here.
}
catch (IOException e)
{
    // Extract some information from this exception, and then
    // throw it to the parent method.
    if (e.Source != null)
        Console.WriteLine("IOException      source: {0}",
e.Source);
    throw;
}

```

Chúng ta có thể bẫy một ngoại lệ và ném một ngoại lệ khác. Khi chúng ta làm điều này, chỉ định ngoại lệ chúng ta bắt như các ngoại lệ bên trong.

```
catch (InvalidOperationException e)
{
    // Perform some action here, and then throw a new
exception.
    throw new YourCustomException("Put your error message
here.", e);
}
```

Chúng ta cũng có thể ném lại một ngoại lệ với một điều kiện cụ thể cho giá trị true.

Ví dụ 1.13:

```
catch (InvalidOperationException e)
{
    if (e.Data == null)
    {
        throw;
    }
    else
    {
        // Take some action.
    }
}
```

Lưu ý, chúng ta có thể sử dụng bộ lọc ngoại lệ để có kết quả tương tự theo cách rõ ràng hơn (cũng như không sửa đổi ngăn xếp), Ví dụ sau có một hành vi tương tự khi ví dụ trước. Hàm ném lại `InvalidOperationException` cho bộ gọi khi `e.Data` là null

Ví dụ 1.14:

```
catch (InvalidOperationException e) when (e.Data != null)
{
    // Take some action.
}
```

Từ bên trong khối try, chỉ khởi tạo các biến được khai báo trong đó. Nếu không một ngoại lệ có thể xảy ra trước khi hoàn thành việc thực thi khối. Ví dụ, trong đoạn code sau, biến `n` được khởi tạo trong khối try. Có sử dụng biến này bên ngoài khối try trong lệnh `Write(n)` có thể tạo ra lỗi biên dịch chương trình.

Ví dụ 1.15:

```
static void Main()
{
    int n;
    try
    {
        // Do not initialize this variable here.
        n = 123;
    }
```

```

    catch
    {
    }
    // Error: Use of unassigned local variable 'n'.
    Console.WriteLine(n);
}

```

1.3 CÁC CẤU TRÚC ĐIỀU KHIỂN TRONG C#

Quyết định tạo các cấu trúc yêu cầu lập trình viên phải xác định một hoặc nhiều điều kiện để chương trình có thể đánh giá hoặc kiểm tra. Song song đó một câu lệnh hoặc nhiều câu lệnh sẽ được thực thi nếu điều kiện đúng, và tùy chọn các câu lệnh khác sẽ được thực thi khi điều kiện sai.

Tương tự C/C++, C# hỗ trợ hai cấu trúc điều khiển đó là cấu trúc rẽ nhánh và cấu trúc lặp.

1.3.1 Cấu trúc rẽ nhánh

Phân nhánh mã lệnh theo một điều kiện cụ thể. Trong C# có 2 câu lệnh hỗ trợ cấu trúc rẽ nhánh đó là if và switch

❖ Câu lệnh if...[else...]

Cú pháp: **if**(<bíểu thức điều kiện>)
 <Công việc 1>
 [else]
 <Công việc 2>]

- Trong đó, Công việc 1 và Công việc 2 có thể là một lệnh đơn (1 lệnh) hoặc một khối lệnh (nhiều hơn 1 lệnh), nếu là khối lệnh phải được bao đóng bởi cặp dấu ngoặc mỏng ({}).

Quy tắc hoạt động của câu lệnh **if** như sau: xét biểu thức điều kiện, nếu điều kiện cho kết quả đúng thực hiện khối lệnh thân **if** (Công việc 1), nếu điều kiện sai:

- **if** không **else**: chương trình sẽ không thực hiện gì cả.
- **if** có **else**: chương trình sẽ thực hiện khối lệnh thân **else** (Công việc 2).

❖ Câu lệnh switch

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

Cú pháp: **switch**(<bíểu thức>)
 {
 case <giá trị 1>: <công việc 1>
 case <giá trị 2>: <công việc 2>

 case <giá trị n>: <công việc n>
 [default]: <công việc n+1>
 }

Trong đó:

- Biểu thức: phải là kiểu số nguyên hoặc kiểu liệt kê, hoặc là kiểu lớp trong đó lớp có một hàm biến đổi đơn giản đến kiểu số nguyên hoặc kiểu liệt kê.

- Giá trị 1, giá trị 2, ..., giá trị n: đây là các biểu thức **hằng** cùng kiểu với biến (biểu thức) trong switch và nó phải là **hằng số**.

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 case 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.

1.3.2 Cấu trúc lặp

C# cung cấp bốn câu lệnh lặp: for, foreach, while, do...while, cho phép chương trình thực thi lặp đi lặp lại một khối lệnh cho đến khi một điều kiện nào đó đã được xác định vẫn còn thỏa.

❖ Câu lệnh for

Cú pháp: **for** (<khởi tạo>;<điều kiện>;<tăng giảm biến đếm>)
 <công việc>

❖ Câu lệnh while

Cú pháp: **while** (<điều kiện>)
 <công việc>

❖ Câu lệnh do...while

Cú pháp: **do**{
 <công việc>
 }while (<điều kiện>);

❖ Câu lệnh foreach

Cú pháp: **foreach** (<tên kiểu> <phần tử> **in** <tập phần tử>)
 <công việc>

Câu lệnh foreach thực hiện duyệt qua từng phần tử trong tập các phần tử, thực hiện lấy thông tin (giá trị) của các phần tử.

Không sử dụng foreach cho việc thêm hoặc xóa các phần tử.

Ví dụ 1.16:

```
static void Main(string[] args)
{
    int[] array = new int[] { 1, 3, 5, 7, 9 };
    foreach (int temp in array)
        Console.WriteLine(temp);
    Console.WriteLine();
    Console.ReadLine();
}
```

❖ Các lệnh hỗ trợ cấu trúc lặp

Lệnh ***break***: thực hiện việc dừng vòng lặp. Khi gặp lệnh break chương trình sẽ lập tức chấm dứt vòng lặp dù điều kiện lặp vẫn còn cho phép chạy tiếp. Trong trường hợp có nhiều vòng lặp lồng nhau, break sẽ giúp chương trình thoát khỏi sự điều khiển của lệnh lặp trực tiếp chứa nó (for, foreach, while, do...while).

Lệnh ***continue***: khi thực hiện vòng lặp, đôi khi người lập trình cần thực hiện bỏ qua một số dòng lệnh để tiếp tục thực hiện lần lặp tiếp theo. Lệnh continue thực hiện việc chuyển sang lần lặp kế tiếp và bỏ qua các lệnh còn lại sau nó trong vòng lặp.

Ví dụ 1.17:

```
static void Main(string[] args)
{
    for (int i = 1; i <= 10; i++)
    {
        Console.WriteLine(i + " ");
        if (i % 3 == 0) {
            Console.WriteLine(); continue;
        }
        Console.WriteLine("Không chia hết cho 3");
        Console.WriteLine();
    }
    Console.ReadLine();
}
```

1.4 MẢNG (ARRAY)

1.4.1 Mảng trong C#

Mảng (Array) là một dãy các phần tử có cùng kiểu được sắp liền kề nhau liên tục trong bộ nhớ. Các phần tử của mảng có cùng tên đó là tên của mảng và để phân biệt các phần tử này với nhau, chúng sẽ được đánh chỉ mục theo thứ tự vị trí.

Trong C#, chỉ mục của các phần tử bắt đầu bằng giá trị 0. Việc thao tác với mảng trong C# tương tự với trong hầu hết các ngôn ngữ lập trình phổ biến khác, tuy nhiên, nó vẫn có một số điểm khác biệt cần lưu ý.

Ngôn ngữ C# cung cấp các loại mảng sau:

- + Mảng một chiều (single-dimensional arrays).
- + Mảng nhiều chiều (multi-dimensional arrays, rectangular arrays).
- + Mảng răng cưa (array-of-arrays, jagged arrays).

1.4.2 Mảng 1 chiều

a/- Khai báo

Mảng một chiều trong C# được khai báo với cú pháp như sau:

<tên kiểu>[] <tên mảng>;

Khác với trong C/C++ khi khai báo, mảng trong C# sẽ không có kích thước (độ lớn, số phần tử) của mảng. Kích thước của mảng sau đó sẽ được cấp phát (xác định) nhờ vào toán tử **new**.

Ví dụ 1.18:

```
int[] numbers; // Khai báo mảng với tên numbers có kích thước bất kỳ.

numbers = new int[10]; //mảng numbers có 10 phần tử.
numbers = new int[20]; //mảng numbers có 20 phần tử.
```

Mảng một chiều trong C# có thể vừa được khai báo vừa được cấp phát cùng nhớ theo cú pháp sau:

<đối tượng>[] <tên mảng> = new <đối tượng>[<số phần tử>];

b/- Khởi tạo giá trị các phần tử mảng

C# cho phép khởi tạo giá trị ban đầu của các phần tử trong mảng tại thời điểm khai báo bằng cách đính kèm các giá trị này trong cặp dấu ngoặc mớc. Các giá trị khởi tạo này sẽ được phân cách nhau bằng dấu phẩy “,”.

Ví dụ 1.19:

```
int[] numbers = new int[5] { 1, 2, 3, 4, 5 };
```

Việc khởi tạo giá trị các phần tử mảng cũng có thể bỏ qua kích thước của mảng như trong ví dụ sau:

Ví dụ 1.20:

```
int[] numbers = new int[] { 1, 2, 3, 4, 5 };
```

Toán tử **new** cũng có thể được bỏ qua khi thực hiện việc khởi tạo giá trị các phần tử của mảng như trong ví dụ sau:

Ví dụ 1.21:

```
int[] numbers = { 1, 2, 3, 4, 5 };
```

c/- Truy xuất phần tử mảng

Tương tự như trong C/C++, việc truy xuất một phần tử trong mảng được thực hiện dựa trên chỉ mục (chỉ số vị trí) của phần tử đó.

Ví dụ 1.22:

```
int[] arr = {10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0};
Console.WriteLine("Phan tu vi tri [4]: {0}", arr[4]);
// trả về 6
arr[4] = 5; // Gán giá trị 5 cho phần tử vị trí [4]
Console.WriteLine("Phan tu vi tri [4]: {0}", arr[4]);
// trả về 5
```

1.4.3 Mảng nhiều chiều

a/- Khai báo

❖ Mảng hai chiều

Trong phạm vi học phần chúng ta chỉ xét mảng hai chiều là một đại diện của mảng nhiều chiều. Mảng hai chiều trong C# được khai báo dựa trên cú pháp sau với dấu phẩy “,” được sử dụng cho việc phân cách giữa các chiều của mảng:

<định danh kiểu>[,] <định danh mảng>;

Ví dụ 1.23:

```
string[,] names;
names = new string[5,4];
```

Mảng names cũng có thể được khai báo lại như sau:

```
string[,] names = new string[5,4];
int[, ,] buttons = new int[4,5,3];
```

❖ Mảng rỗng cửa

Mảng rỗng cửa (mảng lõm chỏm) trong C# là một mảng với các phần tử của nó là các mảng có số phần tử khác nhau.

Mảng dạng này trong C# được khai báo với cú pháp như sau:

<định danh kiểu>[][] <định danh mảng>;

Ví dụ 1.24:

```
int[][] arr;
arr = new int[4][3];//Sai
arr = new int[4][];//Đúng
```

Mảng arr cũng có thể được khai báo lại như sau:

```
int[][] arr = new int[4][];
```

Sau khi khai báo mảng arr, các mảng phần tử của arr cần được khai báo và cấp phát vùng nhớ như sau:

```
for(int i=0;i<arr.length;i++)
    arr[i]=new int[3];
```

Ví dụ 1.25:

```
static void Main(string[] args)
{
    //Mảng rỗng cửa
    int[][] scores = new int[5][];
    for (int i = 0; i < scores.Length; i++)
        scores[i] = new int[i];
    int j = 1;

    foreach (int[] arr in scores)
    {
        Console.WriteLine("Chiều dài của dòng thứ {0} là:
{1}", j, arr.Length);
```

```

        j++;
    }
    Console.ReadLine();
}

```

Kết quả thu được khi chạy đoạn chương trình trên sẽ là:

```

Chieu dai cua dong thu 1 la: 0
Chieu dai cua dong thu 2 la: 1
Chieu dai cua dong thu 3 la: 2
Chieu dai cua dong thu 4 la: 3
Chieu dai cua dong thu 5 la: 4
-
```

b/- Khởi tạo giá trị các phần tử mảng nhiều chiều

Mỗi phần tử trong mảng nhiều chiều được xem là một mảng thành phần và được khởi tạo bằng bộ giá trị tương ứng cho các phần tử của mảng thành phần đó.

Ví dụ 1.26: Khởi tạo giá trị các phần tử mảng hai chiều

```

string[,] siblings = new string[2,2] { {"Mike", "Amy"}, 
{ "Mary", "Albert"} };
string[,] siblings = new string[,] { {"Mike", "Amy"}, 
{ "Mary", "Albert"} };

```

Do các mảng thành phần của mảng răng cưa có số phần tử khác nhau nên cần được mỗi mảng thành phần cần được cấp phát và khởi tạo giá trị như trong ví dụ sau.

Ví dụ 1.27: Khởi tạo giá trị các phần tử mảng răng cưa

```

int[][] numbers2 = new int[2][] { new int[] {2,4}, new
int[] {1,3,5} };

int[][] numbers2 = new int[] [] { new int[] {2,4}, new
int[] {1,3,5} };

```

Tương tự mảng 1 chiều, toán tử new cũng có thể được bỏ qua khi thực hiện việc khởi tạo giá trị các phần tử của mảng như trong ví dụ sau:

Ví dụ 1.28:

```

int[] numbers = { 1, 2, 3, 4, 5 };
string[,] siblings = { {"Mike", "Amy"}, {"Mary", "Albert"} };
int[][] numbers2 = { new int[] {2,4}, new int[] {1,3,5} };

```

c/- Truy xuất phần tử mảng nhiều chiều

Việc truy xuất phần tử mảng nhiều chiều cũng được thực hiện dựa trên chỉ số vị trí phần tử.

Ví dụ 1.29:

```

int[,] a2 = { {1,2}, {3,4}, {5,6}, {7,8}, {9,10} };
Console.WriteLine("Phan tu vi tri [1,1]: {0}", a2[1,1]); // trả về 4
a2[1,1] = 7;
// Gán giá trị 7 cho phần tử ở vị trí [1,1]
Console.WriteLine("Phan tu vi tri [1,1]: {0}", a2[1,1]); // trả về 7

```

Ví dụ 1.30: Cho mảng răng cưa và các phép gán như sau. Phép gán nào sai?

```
int[][] scores2 = new int[][] { new int[] {1, 3}, new int[] {2, 4, 6} };
scores2[0][0] = 68;
scores2[1][2] = 546;
scores2[1][3] = 342;
```

1.4.4 Mảng là đối tượng

Trong C#, các mảng thực chất là các đối tượng. Lớp **System.Array** là kiểu dữ liệu cơ sở trùu tượng của tất cả các kiểu mảng, vì vậy chúng ta có thể sử dụng các thuộc tính và các thành phần khác của **System.Array** cho việc xử lý mảng.

Ví dụ 1.31: Cho mảng 1 chiều sau:

```
int[] numbers = {1, 2, 3, 4, 5};
```

Thuộc tính **Length** cho biết chiều dài của mảng

```
int LengthOfNumbers = numbers.Length;
```

Phương thức **Sort** của lớp **System.Array** cho phép sắp xếp lại mảng

```
System.Array.Sort(numbers);
```

1.4.5 Sử dụng foreach trên mảng

Ngôn ngữ C# cung cấp câu lệnh **foreach** cho phép duyệt qua các phần tử trong mảng, thao tác với giá trị của các phần tử đó.

Ví dụ 1.32: Sử dụng **foreach** duyệt qua tất cả các phần tử mảng, thực hiện in giá trị các phần tử của mảng.

Mảng 1 chiều:

```
int[] numbers = {4, 5, 6, 1, 2, 3, -2, -1, 0};
foreach (int i in numbers)
    Console.WriteLine(i + "\t");
```

Kết quả thu được sẽ là: 4 5 6 1 2 3 -2 -1 0

Mảng 2 chiều:

```
int[,] numbers2 = new int[3, 2] {{9, 99}, {3, 33}, {5, 55}};
foreach (int i in numbers)
    Console.WriteLine("{0} ", i);
```

Kết quả thu được sẽ là: 9 99 3 33 5 55

1.5 NAMESPACE

1.5.1 Namespace là gì?

Trong C#, từ khóa **namespace** cho phép khai báo một phạm vi bao gồm tập các đối tượng có quan hệ với nhau. Chúng ta sử dụng **namespace** cho việc tổ chức các thành phần mã nguồn (code elements) và tạo ra các kiểu mang tính duy nhất toàn cục.

Chúng ta cũng có thể khai báo các thành phần sau trong thân của một **namespace**: namespace khác, class, interface, struct, enum, delegate.

Ví dụ 1.33:

```
namespace SampleNamespace
{
```

```

class SampleClass { }

interface SampleInterface { }

struct SampleStruct { }

enum SampleEnum { a, b }

delegate void SampleDelegate(int i);

namespace SampleNamespace.Nested
{
    class SampleClass2 { }
}

}

```

Để truy xuất các thành phần của namespace chúng ta sử dụng cú pháp sau:

<tên namespace>.<tên thành phần>

Ví dụ 1.34:

```

SampleNamespace.SampleStruct
SampleNamespace.SampleClass
SampleNamespace.SampleInterface

```

Việc lồng các **namespace** bên trong các **namespace** khác cho phép tạo ra cấu trúc phân lớp cho các kiểu dữ liệu. Tên của một **namespace** sẽ bao gồm tên các **namespace** chứa nó, phân cách bởi dấu chấm, bắt đầu bằng **namespace** ngoài cùng nhất và kết thúc bằng tên ngắn gọn của chính nó.

Ví dụ 1.35: tên đầy đủ của namespace Nested là SampleNamespace.Nested

1.5.2 Khai báo sử dụng Namespace

C# cho phép sử dụng tên ngắn gọn để xác định kiểu dữ liệu bằng cách chỉ định trước namespace của kiểu dữ liệu này với từ khóa **using** ở đầu tập tin mã nguồn.

Ví dụ 1.36:

Giả sử trong C# chúng ta có namespace VLUTE với tổ chức như sau:

```

namespace VLUTE
{
    namespace FIT
    {
        namespace CSharp
        {
            class student
            {
                //Mã nguồn định nghĩa lớp student...
            }
        }
    }
}

```

Ở phần đầu một tập tin mã nguồn khác thực hiện khai báo sử dụng namespace như sau: **using VLUTE.FIT.CSharp**. Thì trong tập tin mã nguồn đó chúng ta có thể sử dụng

tên ngắn gọn của lớp đối tượng student thay vì phải sử dụng tên đầy đủ của nó là VLUTE.FIT.CSharp.student.

Trong trường hợp hai kiểu dữ liệu được tham chiếu từ hai namespace có tên trùng nhau thì phải chỉ định rõ tên namespace khi sử dụng. Giả sử trong tổ chức namespace trên ta định nghĩa thêm một namespace đồng cấp với CSharp có tên là WebProgramming và trong đó cũng có chứa một lớp đối tượng tên student; trong tập tin mã nguồn đã đề cập có khai báo sử dụng cả hai namespace này. Vậy khi sử dụng lớp student chúng ta phải chỉ định rõ bằng tên dài hơn của nó đó là CSharp.student hoặc WebProgramming.student theo từng tình huống cụ thể.

Ví dụ 1.37:

```
CSharp.Student nva = new CSharp.Student();
WebProgramming.Student htb = new WebProgramming.Student();
```

1.5.3 Bí danh cho Namespace

Một cách sử dụng khác của từ khóa **using** đó là gán bí danh cho các lớp đối tượng và các namespace. Cú pháp của cách sử dụng này là như sau:

```
using <bí danh> = <tên namespace>;
```

Ví dụ 1.38:

```
//khai báo
using cs = VLUTE.FIT.CSharp;
using web = VLUTE.FIT.WebProgramming;
//Sử dụng
cs.Student nva = new cs.Student();
web.Student htb = new web.Student();
```

1.6 HƯỚNG ĐÓI TƯỢNG TRONG C#

1.6.1 Lớp và đối tượng

a/- Khái niệm

Thuật ngữ “lớp” (class) và “đối tượng” (object) đổi khi được sử dụng để thay thế cho nhau, nhưng thực tế là lớp mô tả kiểu của các đối tượng trong khi đối tượng là các thể hiện khả dụng của lớp.

Hay nói cách khác, lớp là khuôn đúc ra các đối tượng, một đối tượng là một thể hiện cụ thể của một lớp. Ví dụ, một lớp có tên là Sinhvien, và sinhvienA là một thể hiện hay một đối tượng cụ thể của lớp Sinhvien.

b/- Khai báo lớp

Lớp là một kiến trúc cho phép tạo ra các kiểu tùy biến bằng cách gồm nhóm các biến thuộc các kiểu dữ liệu khác nhau, phương thức và sự kiện.

Mỗi lớp gồm nhiều thành phần khác nhau được khai báo với các chỉ dẫn truy cập private, protected, internal hoặc public:

- + **Thuộc tính (properties)**: định nghĩa dữ liệu của lớp
- + **Phương thức (methods)**: mô tả hành vi của lớp.

+ **Sự kiện (events)**: cung cấp cách thức giao tiếp giữa các lớp hoặc các đối tượng thuộc lớp.

Lớp trong C# được khai báo bằng từ khóa **class** với cú pháp như sau:

```
[<chỉ dẫn truy cập>] class <tên lớp>
{
    // Các thành phần của lớp
}
```

Ví dụ 1.39:

```
class Sinhvien
{
    private long mssv;
    public void InMSSV() { }
} //Khai báo lớp Sinhvien có 2 thành phần
```

Lưu ý:

- Việc đặt tên lớp cần tuân thủ các quy tắc sau đây:

+ Sử dụng danh từ để đặt tên cho lớp, phải dễ hiểu, có ý nghĩa, đơn giản

+ Không đặt tên lớp trùng với tên các từ khóa trong C#.

+ Tên lớp không được sử dụng lộn xộn giữa chữ hoa và chữ thường. Nên sử dụng chữ hoa cho ký tự đầu tiên

+ Không được sử dụng chữ số làm ký tự đầu tiên của tên lớp

+ Có thể sử dụng "@" hoặc dấu "_" cho ký tự đầu tiên, tuy nhiên không khuyến khích sử dụng theo cách này

- Nếu lớp không được khai báo với từ khóa **static** thì phải khai báo đối tượng hoặc thể hiện cụ thể để truy xuất các thành phần thuộc lớp.

c/- Tạo đối tượng

Sau khi định nghĩa lớp cần phải tạo các đối tượng là các thể hiện của lớp để có thể truy xuất các thành phần của nó. Việc tạo đối tượng thuộc lớp trong C# được thực hiện thông qua toán tử **new** theo cú pháp sau:

```
<tên lớp> <tên đối tượng> = new <tên lớp>();
```

Ví dụ 1.40:

```
Sinhvien sv1 = new Sinhvien(); //tạo đối tượng sv1 thuộc lớp Sinhvien
```

```
Mathang mathang = new Mathang(); //tạo đối tượng mathang thuộc lớp Mathang
```

Xét ví dụ sau:

Ví dụ 1.41:

```
customer object1 = new customer();
customer object2 = object1;
```

Trong ví dụ trên, hai tham chiếu `object1` và `object2` đã được tạo ra nhưng cùng chỉ đến một đối tượng. Khi có sự thay đổi được thực hiện thông qua `object1`, nó sẽ được phản ảnh qua `object2` ở các lần sử dụng tiếp theo.

Bởi vì các đối tượng của lớp được gọi bằng cách tham chiếu nên lớp là kiểu dữ liệu tham chiếu.

Trong C#, tất cả các lớp đều được dẫn xuất từ lớp cơ sở **System.Object** và mỗi lớp thường được định nghĩa trong một tập tin *.cs. Trong Visual Studio, khi tạo lớp mới, tập tin này sẽ được thêm vào dự án (project).

Khi thao tác với các thành phần của lớp, cũng giống như các ngôn ngữ C++ và Java, C# cung cấp con trỏ **this** dùng để tham chiếu đến một thể hiện (instance) đang xét của lớp.

Ví dụ 1.42:

```
class Cat
{
    //fields
    private string _name;
    //properties
    public string name
    {
        get { return this._name; }
        set { this._name = value; }
    }
    //constructors
    public Cat(string name)
    {
        this._name = name;
    }
    //methods
    public void Speak()
    {
        Console.WriteLine("Meo meo, Everyone calls me {0}",
            this._name);
    }
}
```

d/- Ví dụ minh họa

Tiếp theo ví dụ 1.38, sau khi định nghĩa lớp Cat, chúng ta có thể khai báo các đối tượng thuộc lớp này trong hàm Main của lớp Program để có thể truy xuất các thành phần (không có chỉ dẫn truy xuất private).

```
class Program
{
    static void Main(string[] args)
    {
        Cat cat = new Cat("Ella");
```

```

        cat.Speak();
    }
}

/*Output: Meo meo, Everyone calls me Ella*/

```

Ví dụ 1.43: Khai báo lớp **Student** với các thành phần:

fields: Name (string), Age (int)

methods: phương thức Display cho phép hiển thị thông tin của sinh viên bao gồm tên sinh viên, tuổi sinh viên

```

class Student
{
    //fields
    private string Name = "Huynh Tuan Nghia";
    private int Age = 19;
    //methods
    public void Display() {
        Console.WriteLine("Ten sinh vien: {0}", Name);
        Console.WriteLine("Tuoi sinh vien: {0}", Age);
    }
}

```

Trong hàm Main của class Program khai báo đối tượng oStudent thuộc lớp Student và truy xuất phương thức Display của lớp như sau:

```

class Program
{
    static void Main(string[] args)
    {
        Student oStudent = new Student();
        oStudent.Display();
    }
}

/*Output:
Ten sinh vien: Huynh Tuan Nghia
Tuoi sinh vien: 19
*/

```

e/- Dữ liệu kiểu cấu trúc (struct)

C# cũng cung cấp một phiên bản nhẹ hơn class được gọi là **struct (structure)**. Các kiểu **struct** sẽ trở nên hữu dụng khi chúng ta cần tạo ra một mảng lớn các đối tượng và không muốn tiêu tốn quá nhiều bộ nhớ. Khác với class, **struct** là kiểu dữ liệu giá trị, dùng để bao đóng một nhóm nhỏ các biến có liên quan với nhau. Ví dụ, các chiều của một hình chữ nhật, các đặc trưng của một mục trong bảng kiểm kê.

Cú pháp khai báo **struct**:

```

[<chỉ dẫn truy cập>] struct < tên cấu trúc>
{
    // Các thành phần của cấu trúc
}

```

Ví dụ 1.44: public struct Book

```
{
    public decimal price;
    public string title;
    public string author;
}
```

Các thành phần của **struct** bao gồm các phương thức khởi tạo (constructors), hằng (constants), trường (fields), thuộc tính (properties), phương thức (methods), sự kiện (events), chỉ mục (indexer), toán tử (operators) và các kiểu lồng (nested types). Mặc dù **struct** có hỗ trợ các thành phần được yêu cầu, tuy nhiên chúng ta nên xem xét sử dụng class thay cho **struct**.

Mặc dù có thể thực thi giao diện nhưng **struct** không thể kế thừa từ các **struct** khác. Vì lý do đó, các thành phần của **struct** không thể khai báo với chỉ dẫn protected.

1.6.2 Thuộc tính và phương thức

a/- Trường (Fields) và thuộc tính (Properties)

❖ Trường (Fields)

Một trường (field) là một biến có kiểu bất kỳ được khai báo trực tiếp trong một class hoặc một struct, là thành viên của class hoặc struct đó (kiểu chứa nó).

Một class hay một struct có thể chứa các trường thực thể (instance fields) hoặc các trường tĩnh (static fields) hoặc cả hai.

+ **Trường thực thể** là trường sẽ xác định các thực thể khác nhau của kiểu.

Ví dụ, có một class T, trong T có một trường thực thể F. Chúng ta có thể tạo ra hai thực thể độc lập bằng cách cung cấp hai giá trị khác nhau cho trường F. Khi chúng ta thay đổi giá trị trường F của đối tượng này sẽ không ảnh hưởng đến đối tượng còn lại.

+ **Trường tĩnh**, chính bản thân nó phụ thuộc vào class và được chia sẻ cho tất cả các thực thể của lớp (dùng chung). Nhưng thay đổi trong thực thể A có thể nhìn thấy ngay lập tức ở các thực thể B và C nếu chúng truy xuất vào trường tĩnh.

Nhìn chung, chỉ có thể truy cập trực tiếp các trường ở 2 chế độ private và protected. Việc truy xuất dữ liệu (fields) của class chỉ có thể thực hiện thông qua methods, properties và indexers. Một trường với chỉ dẫn truy cập private được “phô bày” (expose) thông qua các properties có chỉ dẫn truy cập public được gọi là backing store hay backing field. Cơ chế hạn chế truy xuất này giúp bảo vệ chống lại những giá trị input không hợp lệ.

Các trường lưu trữ dữ liệu điển hình có thể được truy xuất từ nhiều phương thức lớp và được lưu trữ lâu hơn “thời gian sống” của một phương thức đơn.

❖ Thuộc tính (Properties)

C# cho phép hoặc là tạo ra các trường private lưu trữ các giá trị thuộc tính hoặc là sử dụng cái gọi là thuộc tính thực thi tự động (auto-implemented properties) thực hiện tạo ra các trường này một cách tự động ở lớp bên dưới và cung cấp luận lý cơ bản cho các thủ tục thuộc tính.

Để định nghĩa thuộc tính thực thi tự động, chúng ta có thể thực hiện như minh họa sau:

```
class SampleClass
{
    public int SampleProperty { get; set; }
}
```

Các thuộc tính với các thủ tục get và set của chúng cung cấp cơ chế phức tạp cho việc đọc, ghi và tính toán các giá trị thuộc tính lưu trữ trong các trường private.

```
class SampleClass
{
    private int _sample;

    public int Sample
    {
        // Trả về giá trị được lưu trữ tại field.
        get { return _sample; }
        // Lưu trữ giá trị vào field.
        set { _sample = value; }
    }
}
```

Thuộc tính được truy xuất như các thành phần dữ liệu public. Nhưng thực tế nhờ vào các phương thức đặc biệt được gọi là các giám định viên (accessor) get và set, thuộc tính cho phép dữ liệu được truy xuất một cách dễ dàng nhưng vẫn giúp thúc đẩy sự an toàn, phức tạp của các phương thức. Thuộc tính cho phép một lớp phơi bày cách thức nhận và thiết lập các giá trị, trong khi các mã thực thi và xác minh được giấu đi.

Ví dụ 1.45:

```
class TimePeriod
{
    //fields
    private int seconds;
    //properties
    public int hours
    {
        get { return seconds / 3600; }
        set { this.seconds = value * 3600; }
    }
}
```

Trong thân phương thức Main của class Program khai báo đối tượng t thuộc lớp TimePeriod, thuộc tính hours được sử dụng như sau:

```
class Program
```

```

{
    static void Main(string[] args) {
        TimePeriod t = new TimePeriod();
        //Gán giá trị cho hours bằng cách gọi set
        t.hours = 24;
        //Định giá trị của hours bằng cách gọi get
        Console.WriteLine("Time in hours: {0}", t.hours);
    }
}

```

b/- Phương thức

Một phương thức là một hành vi mà đối tượng thuộc lớp chứa phương thức có thể thực hiện. Phương thức là một khối mã lệnh bao gồm một loạt các chỉ thị. Các đối tượng là thể hiện của lớp chỉ có thể truy xuất đến phương thức khi và chỉ khi lớp chứa nó cho phép truy xuất (thông qua các chỉ dẫn truy cập).

Chương trình sinh ra các chỉ thị được thực thi bằng cách gọi phương thức và định rõ các đối số được yêu cầu. Trong C#, mỗi chỉ lệnh thực thi được thực hiện trong ngữ cảnh của một phương thức. Phương thức Main là đầu vào của mỗi ứng dụng C# và nó được gọi bởi CLR khi chương trình bắt đầu chạy.

C# cung cấp 2 loại phương thức: non-static methods và static methods

- + **Non-static methods:** chỉ được truy xuất thông qua đối tượng của lớp, mang tính đặc thù của đối tượng. Từng đối tượng khác nhau sẽ trả về kết quả khác nhau.

- + **Static methods:** đặc trưng cho tất cả các đối tượng của lớp, không thể gọi từ đối tượng mà phải gọi trực tiếp từ lớp. Kết quả của phương thức không phụ thuộc vào đối tượng mà phụ thuộc vào tham số đầu vào của phương thức.

❖ Khai báo phương thức

Các phương thức được khai báo trong class hoặc struct bằng cách chỉ định mức độ truy xuất là public hoặc private, các bộ nghĩa tùy chọn là abstract hoặc sealed, giá trị trả về, tên của phương thức, và các tham số phương thức.

Lưu ý:

- Kiểu trả về không phải là một phần của khai báo phương thức với mục đích nạp chồng phương thức. Tuy nhiên, nó sẽ là một phần của khai báo phương thức khi xác định sự tương thích giữa một đại diện (delegate) và phương thức mà nó chỉ đến.

- Một phương thức có thể có hoặc không có giá trị trả về.

- Các tham số của phương thức được đặt trong cặp dấu ngoặc tròn và phân cách nhau bởi dấu phẩy “,”. Cặp dấu ngoặc rỗng tương ứng với phương thức không tham số.

Ví dụ 1.46: Lớp `Motorcycle` sau đây có ba phương thức:

```

abstract class Motorcycle
{

```

```

//Bất kỳ ai cũng có thể gọi phương thức này.
public void StartEngine()
{
    /* Các chỉ thị của phương thức */
}
//Chỉ có lớp dẫn xuất mới gọi được phương thức này.
protected void AddGas(int gallons)
{
    /* Các chỉ thị của phương thức */
}
//Lớp dẫn xuất có thể ghi đè sự thi hành của lớp cơ sở
public virtual int Drive(int miles, int speed)
{
    /* Các chỉ thị của phương thức */
    return 1;
}
//Lớp dẫn xuất phải hoàn thành phương thức này.
public abstract double GetTopSpeed();
}

```

Cú pháp khai báo phương thức:

```

<A>[B]<kiểu trả về> <tên phương thức>([<danh sách tham số>])
{
    /*Khởi lệnh thân phương thức*/
}

```

Trong đó:

- + A: chỉ dẫn truy cập (public, internal, protected, private)
- + B: bối nghĩa tùy chọn (static, abstract, virtual…)

Lưu ý: cần thêm từ khóa static vào cú pháp trên để khai báo static method và chỉ dẫn truy cập tương ứng bắt buộc phải là public.

Ví dụ 1.47: Giả sử trong class Maytinh có phương thức Cong được khai báo ở dạng static. Trong phương thức Main của class Program, việc truy xuất phương thức Cong sẽ được thực hiện thông qua tên class Maytinh.Cong(a, b).

```

class Maytinh
{
    //Phương thức Cong được khai báo ở dạng static
    public static int Cong(int so1, int so2)
    {
        return so1 + so2;
    }
}
class Program

```

```

{
    static void Main(string[] args)
    {
        int a, b;
        Console.WriteLine("Nhập số thứ nhất: ");
        a = Convert.ToInt32(Console.ReadLine());
        Console.WriteLine("Nhập số thứ hai: ");
        b = Convert.ToInt32(Console.ReadLine());
        Console.WriteLine("Kết quả {0} + {1} = {2}", a,
        b, Maytinh.Cong(a, b));
        Console.ReadLine();
    }
}

```

Kết quả sau khi chạy chương trình thu được:

```

Nhập số thứ nhất: 7
Nhập số thứ hai: 8
Kết quả 7 + 8 = 15
-
```

Lưu ý: Trong đoạn chương trình trên Convert.ToInt32 được sử dụng cho việc chuyển đổi kiểu của giá trị nhận vào thông qua phương thức Console.ReadLine(), từ kiểu string sang kiểu số nguyên int (Int32) (kiểu của biến nhận giá trị). Chúng ta có thể thay Convert.ToInt32 bằng int.Parse hoặc Int32.Parse cho việc chuyển đổi kiểu này, tuy nhiên cần xem xét cách sử dụng cho từng trường hợp cụ thể. Với các kiểu dữ liệu khác cũng thực hiện tương tự, trừ kiểu dữ liệu char và string.

❖ Truy xuất phương thức

Việc truy xuất phương thức dạng non-static methods của một lớp cần được thực hiện thông qua đối tượng là thể hiện của lớp với cú pháp như sau:

```
<tên đối tượng>.<tên phương thức>([<danh sách tham số>])
```

Việc truy xuất phương thức dạng static methods của một lớp cần được thực hiện trực tiếp thông qua tên lớp theo cú pháp sau:

```
<tên lớp>.<tên phương thức>([<danh sách tham số>])
```

❖ Truyền đối số (Arguments) cho phương thức

Có hai cách truyền đối số cho phương thức trong C#, đó là truyền tham trị và truyền tham chiếu.

Để thực hiện truyền đối số theo kiểu tham chiếu ta sử dụng từ khóa ref cho tham số đó trong cả khai báo và truy xuất phương thức.

Ví dụ 1.48: Truyền theo tham trị

```

class PassingRefByVal
{

```

```

static void Change(int[] pArray)
{
    // This change affects the original element.
    pArray[0] = 888;
    // This change is local.
    pArray = new int[5] { -3, -1, -2, -3, -4 };
    System.Console.WriteLine("Inside the method, the
    first element is: {0}", pArray[0]);
}
static void Main(string[] args)
{
    int[] arr = { 1, 4, 5 };
    System.Console.WriteLine("Inside Main, before
    calling the method, the first element is: {0}",
    arr[0]);
    Change(arr);
    System.Console.WriteLine("Inside Main, after
    calling the method, the first element is: {0}",
    arr[0]);
    System.Console.ReadLine();
}
/* Kết quả thu được là gì????*/

```

Ví dụ 1.49: Truyền theo tham chiếu

```

class PassingRefByRef
{
    static void Change(ref int[] pArray)
    {
        // Both of the following changes will affect the
        // original variables:
        pArray[0] = 888;
        pArray = new int[5] { -3, -1, -2, -3, -4 };
        System.Console.WriteLine("Inside the method, the
        first element is: {0}", pArray[0]);
    }
    static void Main(string[] args)
    {
        int[] arr = { 1, 4, 5 };
        System.Console.WriteLine("Inside Main, before
        calling the method, the first element is: {0}",
        arr[0]);
    }
}

```

```

        Change(ref arr);
        System.Console.WriteLine("Inside Main, after
calling the method, the first element is: {0}",
arr[0]);
        System.Console.ReadLine();
    }
}/* Kết quả thu được là gì????*/

```

c/- Phương thức khởi tạo (Constructors) và phương thức hủy (Destructors)

❖ Phương thức khởi tạo (Constructors)

Phương thức khởi tạo (bộ khởi tạo) là phương thức đặc biệt, được gọi khi tạo đối tượng là thể hiện của lớp, có chức năng khởi tạo các giá trị của thành phần dữ liệu (bao gồm fields và properties) của đối tượng.

Trước khi bộ khởi tạo được thực thi thì đối tượng chưa được cấp phát trong bộ nhớ. Sau khi bộ khởi tạo thực thi hoàn thành thì bộ nhớ sẽ lưu giữ một thể hiện hợp lệ của lớp vừa khai báo.

Phương thức khởi tạo được định nghĩa khi xây dựng lớp. Nếu chúng ta không khai báo các phương thức khởi tạo tường minh thì mặc định CLR sẽ thay mặt chúng ta mà tạo phương thức khởi tạo gọi là phương thức khởi tạo ngầm định.

Phương thức khởi tạo sẽ được khai báo giống như phương thức thông thường của lớp tuy nhiên nó sẽ có các đặc điểm như: không có giá trị trả về, tên phương thức khởi tạo phải trùng với tên lớp. Các phương thức khởi tạo của lớp được khai báo theo cú pháp sau:

```

<chỉ dẫn truy cập><tên lớp>([<danh sách tham số>])
{
    /* Khởi lệnh thân phương thức khởi tạo */
}

```

Ví dụ 1.50:

```

class Point
{
    private int x;
    private int y;
    //Constructor không tham số
    public Point()
    {
        this.x = 0;
        this.y = 0;
    }
    //Constructor hai tham số
    public Point(int oX, int oY)
    {

```

```

        this.x = oX;
        this.y = oY;
    }
    public void Display() {
        Console.WriteLine("Toa do x = {0}, toa do y =
{1}", this.x, this.y);
    }
}

```

❖ Phương thức hủy (Destructors)

Ngược lại với phương thức khởi tạo, phương thức hủy được gọi khi đối tượng tương ứng bị xóa khỏi bộ nhớ. Phương thức hủy có cùng tên với tên lớp tuy nhiên phía trước có dấu “~”.

Phương thức hủy có các đặc điểm sau đây:

- + Không cho phép kế thừa và không cho phép nạp chồng;
- + Không được gọi tường minh;
- + Không có phạm vi truy cập và không có tham số truyền vào;
- + Chỉ được dùng với class, không được định nghĩa trong struct.

Cú pháp khai báo phương thức hủy:

```

~ <tên lớp>()
{
    /* Khởi lệnh thân phương thức hủy */
}

```

Khi làm việc với các đoạn mã không được quản lý thì cần phải khai báo tường minh các phương thức hủy để giải phóng các tài nguyên.

C# cung cấp ngầm định một phương thức để thực hiện điều khiển công việc này, phương thức đó là Finalize() hay còn gọi là bộ kết thúc.

Phương thức Finalize này sẽ được gọi bởi cơ chế thu dọn khi đối tượng bị hủy và phương thức này được gọi trong nội dung thực hiện của phương thức hủy.

1.6.3 Kế thừa

a/- Kế thừa trong lập trình hướng đối tượng

Kế thừa là khái niệm then chốt của các ngôn ngữ lập trình hướng đối tượng dùng để mô tả hiện tượng một lớp sử dụng lại một số thuộc tính (fields và properties) hoặc phương thức của một lớp khác.

- + Lớp sử dụng các thuộc tính của lớp khác được gọi là lớp dẫn xuất hay lớp con.
- + Lớp cho phép lớp khác sử dụng các thuộc tính và phương thức được gọi là lớp cơ sở hay lớp cha.

Kế thừa trong hướng đối tượng được chia thành 2 loại: đơn kế thừa (single inheritance) và đa kế thừa (multiple inheritance). Việc sử dụng đa kế thừa một cách hợp lý

giúp cho chương trình tối ưu hơn về mặt kích thước, tuy nhiên việc xử lý lỗi (nếu có) sẽ vô cùng phức tạp. Vì vậy mặc dù C# được phát triển từ ngôn ngữ lập trình C++, hỗ trợ cả 2 dạng kế thừa trên, tuy nhiên trong thiết kế của mình, ngôn ngữ C# chỉ hỗ trợ đơn kế thừa.

b/- Đơn kế thừa trong C#

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

Cú pháp khai báo kế thừa trong C#:

```
class <tên lớp dẫn xuất> : <tên lớp cơ sở>
{
    //Các thành phần của lớp dẫn xuất
}
```

Ví dụ 1.51: Khai báo một class Cat và các class dẫn xuất từ nó là Balinese và Birman

```
class Cat
{
    private string Name;
    private float Weight;
    public Cat()
    {
        this.Name = "";
        this.Weight = 0f;
    }

    //Constructors
    public Cat(string name)
    {
        this.Name = name;
        this.Weight = 0f;
    }

    //methods
    public void Speak()
    {
        Console.WriteLine("Meo meo, Everyone calls me {0}",
            this.Name);
    }

    public void DrinkWater()
    {
        Console.WriteLine("Nhao, I would like a little of
water");
```

```

    }
}
```

Lớp Balinese được dẫn xuất từ lớp cơ sở Cat với thành phần riêng của nó là Communicate() được thêm vào.

```

class Balinese : Cat {
    public void Communicate() {
        Console.WriteLine("I am sensitive to your moods and
feelings");
    }
}
```

Lớp Birman là một lớp dẫn xuất khác của lớp cơ sở Cat với thành phần Greeting() được thêm vào.

```

class Birman : Cat
{
    public void Greeting()
    {
        Console.WriteLine("I will generally greet
visitors with curiosity rather than fear");
    }
}
```

Trong một ứng dụng, việc tận dụng tính năng kế thừa của hướng đối tượng làm cho chương trình trở nên ngắn gọn, dễ hiểu. Trong tình huống chương trình có chứa nhiều lớp tương tự nhau trong đó có rất nhiều thành phần (thuộc tính, phương thức) giống nhau thì việc xây dựng một lớp cơ sở chứa các thành phần chung làm cho việc cập nhật, chỉnh sửa được thuận lợi hơn.

c/- Lớp dẫn xuất truy xuất thành phần của lớp cơ sở

Một lớp dẫn xuất không thể truy xuất các thành phần private của lớp cơ sở. Tuy nhiên, các thành phần private vẫn hiện diện trong lớp dẫn xuất và có thể làm những việc chúng đã làm trong chính lớp cơ sở.

Khi một phương thức được ghi đè trong lớp dẫn xuất, việc triệu gọi phương thức thực thi sẽ được thực hiện trên phiên bản của lớp dẫn xuất. Tuy nhiên, để cho phép trong lớp dẫn xuất có thể triệu gọi phương thức với phiên bản được cài đặt ở lớp cơ sở, C# có một cú pháp đặc biệt với từ khóa base: **base.<MethodName>()**

Ví dụ 1.52: Lớp ITStudent được dẫn xuất từ lớp Student với phương thức Display() của lớp cơ sở được truy xuất thông qua từ khóa Base.

```

class Student
{
    public virtual void Display()
    {
```

```

        Console.WriteLine("General information of
student...");
    }
}

class ITStudent : Student
{
    public override void Display()
    {
        base.Display();
        Console.WriteLine("Special information of IT
student...");
    }
}

```

1.6.4 Tính đa hình

a/- Phương thức ảo (virtual methods) và phương thức trừu tượng (abstract methods)

Khi lớp cơ sở khai báo một phương thức ảo với bộ nghĩa tùy chọn `virtual` thì lớp dẫn xuất có thể ghi đè (overriding) phương thức với những xử lý riêng của chính lớp dẫn xuất.

Nếu lớp cơ sở có định nghĩa một phương thức trừu tượng với bộ nghĩa tùy chọn `abstract` thì phương thức đó phải được ghi đè trong bất kỳ lớp dẫn xuất kế thừa trực tiếp từ lớp cơ sở và lớp dẫn xuất không được khai báo trừu tượng. Nếu lớp dẫn xuất là lớp trừu tượng thì nó sẽ không thực thi các thành phần trừu tượng kế thừa.

Thành phần trừu tượng và thành phần ảo là nền tảng của tính đa hình của lập trình hướng đối tượng.

b/- Lớp cơ sở trừu tượng (abstract base classes)

Có thể khai báo một lớp trừu tượng nếu muốn ngăn chặn việc tạo trực tiếp đối tượng là thể hiện của lớp (instantiation) với từ khóa `new`. Khi khai báo lớp cơ sở trừu tượng thì nó chỉ có thể được sử dụng khi có một lớp được dẫn xuất từ nó.

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

Ví dụ 1.53:

```

abstract class SinhVien
{
    // a field
    private bool damaged = false;
    //an abstract method
    public abstract decimal DiemTrungBinh();
}

```

Khi một lớp được khai báo ở dạng trừu tượng nó không cần có các thành phần trừu tượng. Nhưng ngược lại, nếu một lớp có chứa thành phần trừu tượng thì bắt buộc nó phải được khai báo trừu tượng.

Các lớp dẫn xuất không được khai báo dưới dạng trừu tượng phải cung cấp sự thực thi cho bất kỳ phương thức trừu tượng nào thuộc lớp cơ sở trừu tượng.

c/- *Ghi đè phương thức (Method overriding)*

Ghi đè (Overriding) và nạp chồng (overloading) là hai kỹ thuật tạo nên tính đa hình trong lập trình hướng đối tượng. Trong lập trình cấu trúc C, chúng ta không thể nào khai báo các phương thức (method) trùng tên, nhưng trong OOP chỉ cần áp dụng hai kỹ thuật này, chúng ta hoàn toàn có thể làm được điều đó.

Chúng ta sử dụng ghi đè phương thức (overriding) khi muốn thay đổi hành vi của phương thức mà nó ghi đè. Nói cách khác, phương thức ghi đè (trong lớp dẫn xuất) sẽ ghi đè nội dung của nó lên phương thức bị ghi đè (trong lớp cơ sở). Phương thức ghi đè phải được khai báo giống hệt phương thức bị ghi đè cho cả phần tham số và kiểu trả về.

Overriding thường được sử dụng trong phương thức ở lớp dẫn xuất, bằng cách khai báo một phương thức ở lớp cơ sở là `virtual`, chúng ta có thể ghi đè phương thức đó ở lớp dẫn xuất của lớp này với bổ nghĩa tùy chọn `override`. Điều này có nghĩa là có thể cài đặt lại phương thức `VirtualMethod()` (với cùng chữ ký phương thức) trong lớp dẫn xuất `MyDerivedClass` của `MyBaseClass` như trong ví dụ sau.

Ví dụ 1.54:

```
class MyBaseClass
{
    public virtual void MyVirtualMethod()
    {
        Console.WriteLine("Phuong thuc nay duoc khai bao trong
        lop co so MyBaseClass");
    }
}

class MyDerivedClass : MyBaseClass
{
    public override void MyVirtualMethod()
    {
        Console.WriteLine("Phuong thuc nay duoc dinh nghia de
        trong lop dan xuuat MyDerivedClass");
    }
}
```

Khi gọi phương thức `MyVirtualMethod` từ một thể hiện của lớp dẫn xuất, thì phương thức của lớp dẫn xuất sẽ được triệu gọi, không phải là phương thức của lớp cơ sở.

d/- Che giấu phương thức

Nếu một phương thức có chữ ký phương thức được khai báo trong cả lớp cơ sở và lớp dẫn xuất, nhưng các phương thức không được khai báo tương ứng với các bô nghĩa tùy chọn là `virtual` và `override` thì phiên bản phương thức ở lớp dẫn xuất được gọi là đã che giấu phiên bản ở lớp cơ sở.

Phiên bản của phương thức được truy xuất sẽ tùy thuộc vào kiểu dữ liệu của biến được sử dụng để tham chiếu đến đối tượng chứ không phải là chính đối tượng.

Tại thời điểm biên dịch, trình biên dịch sẽ đưa ra cảnh báo về việc phương thức bị che giấu khi đó cần thêm từ khóa `new` khi khai báo phương thức được định nghĩa lại trong lớp dẫn xuất.

e/- Nạp chồng phương thức (Method overloading)

Nạp chồng phương thức (overloading) đơn giản chỉ để tạo ra các phương thức cùng tên trong cùng một lớp. Tuy nhiên, các phương thức đó phải khác nhau về tham số đầu vào (parameter) hoặc kiểu trả về.

Các phương thức với kỹ thuật nạp chồng sẽ không ghi đè lên nhau thay vào đó chúng sẽ cùng tồn tại song song. C# hỗ trợ nạp chồng phương thức, cho phép có nhiều phiên bản cho một phương thức có các chữ ký khác nhau.

Ví dụ 1.55: Lớp `Student` sau đây sẽ có 2 phương thức `Display()` và `Display(string Content)` được khai báo theo dạng nạp chồng phương thức.

```
class Student
{
    public void Display()
    {
        //Implementation of method
    }
    public void Display(string Content)
    {
        //Implementation of method
    }
}
```

f/- Nạp chồng toán tử (Operator overloading)

Nạp chồng toán tử là việc thực hiện định nghĩa lại một số toán tử mà toán tử này làm việc với dữ liệu là đối tượng thuộc lớp đang xây dựng.

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 <kiểu trả về> operator <toán tử> ([<danh sách tham số>])
{
    //cài đặt mã lệnh
}
```

Ví dụ 1.56:

```
public static Complex operator +(Complex c1, Complex c2)
{
    return new Complex(c1.real + c2.real, c1.imaginary +
        c2.imaginary);
}
```

Lưu ý:

- C# yêu cầu các toán tử phải là static method với chỉ dẫn truy cập public
- Trong C#, tất cả các lớp đều dẫn xuất từ lớp cơ sở System.Object và lớp này vốn định nghĩa sẵn phương thức Equals và toán tử gán do đó không cần thiết phải định nghĩa lại toán tử gán.

CÂU HỎI VÀ BÀI TẬP

Bài 1. Tạo một project ứng dụng Console mới với tên Baitap1. Trong phương thức main () nhập vào đoạn code sau, biên dịch và chạy chương trình. Hãy cho biết chương trình làm điều gì?

```
int x = 0;
for(x = 1; x < 10; x++)
{
    System.Console.Write("{0:03}",x);
}
```

Bài 2. Trong phương thức main() của Bài 1. Nhập vào đoạn code sau, tìm lỗi của chương trình? sửa lỗi và biên dịch chương trình.

```
for(int i=0;i < 10 ; i++)
System.Console.WriteLine("so :{1}", i);
```

Bài 3. Tạo project ứng dụng Console mới với tên Baitap3. Trong phương thức main () nhập vào đoạn code sau. Tìm lỗi của chương trình, sửa lỗi và biên dịch lại chương trình?

```
double myDouble;
decimal myDecimal;
myDouble = 3.14;
myDecimal = 3.14;
Console.WriteLine("My Double: {0}", myDouble);
Console.WriteLine("My Decimal: {0}", myDecimal);
```

Bài 4. Tạo project ứng dụng Console mới với tên Baitap4. Trong phương thức main () nhập vào đoạn code sau. Tìm lỗi của chương trình, sửa lỗi và biên dịch lại chương trình?

```
int value;
if (value > 100);
System.Console.WriteLine("Number is greater than 100");
```

Bài 5. Viết chương trình nhập 3 số a, b, c hãy cho biết 3 số trên có thể là độ dài 3 cạnh của một tam giác không?

Điều kiện: $(a+b) > c$; $(a+c) > b$; $(b+c) > a$; $a, b, c > 0$

Bài 6. Viết chương trình hiển thị ra màn hình các mẫu sau:

*	\$ \$ \$ \$ \$ \$	*
* *	\$ \$ \$ \$ \$	* * *
* * *	\$ \$ \$ \$	* * * *
* * * *	\$ \$ \$	* * * * * *
* * * * *	\$ \$	* * * * * * *
* * * * * *	\$	* * * * * * * *

Bài 7. Viết chương trình nhập vào điểm cơ bản (đcb) và điểm nâng cao (đnc) cho 1 học viên. Cho biết học viên này được xếp loại gì, với cách xếp loại dựa trên điểm trung bình ($\text{đtb} = (\text{đcb} + \text{đnc}) / 2$) như sau:

Nếu $\text{đtb} \geq 9$ và không có điểm nào dưới 8 thì được xếp loại xuất sắc.

Nếu $\text{đtb} \geq 8$ và không có điểm nào dưới 7 thì được xếp loại giỏi.

Nếu $\text{đtb} \geq 7$ và không có điểm nào dưới 6 thì được xếp loại khá.

Nếu $\text{đtb} \geq 5$ vào không có điểm nào dưới 3 thì được xếp loại trung bình.

Còn lại thì ghi không đat.

Bài 8. Viết chương trình làm việc như 1 máy tính bỏ túi:

- Nhập vào 2 số.
 - Hỏi toán tử là +, -, * hay /, tương ứng in ra tổng, hiệu, tích, thương của 2 số.
 - Nếu không phải là các toán tử trên thì kết thúc chương trình.

Bài 9. Viết chương trình in ký tự số (0..9) và ký tự chữ (a..z) với mã ký tự tương ứng của từng

ký tú:

'0':48

'1': 49

• • • •

Bài 10. Viết chương trình giải phương trình bậc nhất với các giá trị a, b được nhập từ bàn phím

Bài 11. Viết chương trình giải phương trình bậc hai với các giá trị a, b, c được nhập từ bàn phím

Bài 12. Viết chương trình tính chu vi và diện tích của các hình sau: đường tròn, hình chữ nhật, hình thang, tam giác.

Bài 13. Viết chương trình tính tổng $S = 1/2 + 2/3 + 3/4 + \dots + n/(n+1)$, với n là số nguyên được nhập từ bàn phím.

Bài 14. Viết chương trình tính tổng $S=1 + 2 - 3 + 4 + 5 - 6 + 7 + 8 - 9 + \dots + /-n$, với n là số nguyên được nhập từ bàn phím.

Bài 15. Viết chương trình tính tổng $S= -1 + 2 - 3 + 4 - \dots + (-1)n n$, với n là số nguyên dương nhập từ bàn phím.

Bài 16. Viết chương trình tính tổng $S = \sum_{i=1}^n \sum_{j=1}^m i + 2j$ với n, m là số nguyên dương được nhập từ bàn phím.

Bài 17. Viết chương trình tính $e = 1 + \frac{x}{1!} - \frac{x^2}{2!} + \frac{x^3}{3!} - \frac{x^4}{4!} + \dots + (-1)^{n+1} \frac{x^n}{n!}$, với n là số nguyên dương, x là số thực được nhập từ bàn phím.

Bài 18. Viết chương trình tính $e = 1 + 1/1! + 1/2! + 1/3! + \dots + 1/n!$, cho đến khi $1/(n+1)!$ nhỏ hơn Epsilon với Epsilon là 1 lượng khá nhỏ được nhập từ bàn phím.

Bài 19. Viết chương trình hiển thị ra màn hình như sau:

```

    1
    2 3 2
    3 4 5 4 3
    4 5 6 7 6 5 4
    5 6 7 8 9 8 7 6 5
    6 7 8 9 0 1 0 9 8 7 6
    7 8 9 0 1 2 3 2 1 0 9 8 7
    8 9 0 1 2 3 4 5 4 3 2 1 0 9 8
    9 0 1 2 3 4 5 6 7 6 5 4 3 2 1 0 9
    0 1 2 3 4 5 6 7 8 9 8 7 6 5 4 3 2 1 0
  
```

Bài 20. Viết chương trình nhập vào 1 số nguyên dương, in ra ước số lẻ lớn nhất của nó. Ví dụ ta nhập vào số 60 thì in ra là 15.

Bài 21. Viết chương trình in ra tất cả các số nguyên tố < N, với N được nhập từ bàn phím.

Bài 22. Viết chương trình tính số hạng thứ n của dãy Fibonaci.

Hướng dẫn: Dãy Fibonaci là dãy số gồm số hạng $F(n)$ với $F(n)=F(n-1)+F(n-2)$ và $F(1)=F(2)=1$.

Bài 23. Viết chương trình nhập vào mảng 1 chiều gồm n phần tử kiểu số nguyên. In ra mảng vừa nhập theo thứ tự ngược. Cho biết có bao nhiêu phần tử có nội dung là số nguyên tố. Tính tích các phần tử là ước số của k, với k được nhập từ bàn phím. Cho biết phần tử X xuất hiện ở lần thứ m tại vị trí thứ mấy, với X và m được nhập từ bàn phím. Sắp xếp mảng theo thứ tự giảm dần. In ra mảng sau khi sắp xếp.

Bài 24. Viết chương trình thực hiện các công việc sau:

- Nhập mảng 1 chiều gồm n phần tử kiểu số nguyên.

- In ra mảng vừa nhập.
- In ra vị trí của các phần tử lớn nhất có trong dãy.
- Tính trị trung bình của các phần tử dương có trong dãy.
- Đếm số phần tử là lũy thừa của K, với K nhập từ bàn phím

Bài 25. Viết chương trình thực hiện các công việc sau:

- Nhập mảng 1 chiều gồm n phần tử kiểu số nguyên.
- In ra các phần tử nằm ở các vị trí chẵn.
- Kiểm tra xem dãy có thứ tự hay không?
- Kiểm tra xem dãy có đối xứng không?
- Tạo ra 1 mảng mới được copy từ mảng nhập gồm M phần tử bắt đầu từ phần tử thứ K, với M và K được nhập từ bàn phím.
- In ra mảng vừa tạo.

Bài 26. Viết chương trình nhập vào ma trận a gồm m hàng, n cột, các phần tử kiểu số thực. In ra ma trận vừa nhập. Cho biết trong ma trận có bao nhiêu phần tử có phần nguyên là chẵn. Tính tích các phần tử dương nằm trên hàng h, với h nhập từ bàn phím. Sắp xếp các phần tử nằm trên cột c theo thứ tự tăng dần, với c nhập từ bàn phím. In ra ma trận sau khi sắp xếp

Bài 27. Viết chương trình nhập vào ma trận vuông cấp n, các phần tử kiểu ký tự. In ra ma trận vừa nhập. Cho biết trong ma trận có bao nhiêu ký tự ‘T’. In ra các phần tử nằm trên đường chéo phụ. Cho biết ký tự lớn nhất nằm trên đường chéo chính là gì. Cho biết ma trận có hàng nào có thứ tự tăng dần không?

Bài 28. Viết chương trình nhập vào ma trận vuông cấp n, các phần tử kiểu số nguyên. In ra ma trận vừa nhập. In ra các phần tử nằm trên đường đường biên của ma trận. Cho biết phần tử nhỏ nhất nằm ở vị trí nào. Cho biết hàng nào có tổng các phần tử là lớn nhất. Tính giá trị trung bình của tất cả các phần tử dương có trong ma trận

Bài 29. Viết chương trình nhập một số nguyên N từ bàn phím, sau đó in ra màn hình N dòng đầu tiên của tam giác Pascal sử dụng mảng.

Bài 30. Một phân số gồm 2 thành phần tử số và mẫu số. Anh (Chị) hãy thực hiện các công việc sau:

- Viết phương thức cho phép nhập vào một phân số bất kỳ.
- Một phân số được gọi là tối giản khi và chỉ khi UCLN của tử số và mẫu số có giá trị bằng 1. Anh (Chị) hãy xây dựng phương thức tìm UCLN của 2 số nguyên dương m và n, sau đó sử dụng hàm này cho việc kiểm tra phân số người dùng nhập vào có dạng tối giản hay chưa, nếu chưa thực hiện việc rút gọn phân số bằng cách chia tử số và mẫu số cho UCLN của chúng.

- Viết phương thức cho phép in ra màn hình phân số vừa nhập ở dạng tối giản và theo đúng định dạng (Ví dụ: tử số = 8, mẫu số = -9. Phân số vừa nhập là -8/9).

d/- Viết phương pháp có tên tinhtoan thực hiện tính toán trên 2 phần số vừa nhập, với phép toán cần thực hiện (cộng: +; trừ: -; nhân: *; chia: /) được nhập từ bàn phím. Trong trường hợp phép toán không hợp lệ thì yêu cầu người dùng nhập lại.

e/- Nạp chồng các phép toán so sánh trên hai phân số.

f/- Viết chương trình kiểm tra tính đúng đắn của các hàm vừa xây dựng.

Bài 31. Viết chương trình quản lý sách cho 1 thư viện. Thông tin về mỗi quyển sách gồm: tựa sách, tên tác giả, tên nhà xuất bản, năm xuất bản và số trang của nó. Nhập danh sách n quyển sách, với n nhập từ bàn phím. In danh sách các quyển sách đã nhập. Cho biết có bao nhiêu quyển sách của tác giả X, với X nhập từ bàn phím. Tính tổng số trang của các quyển sách được xuất bản vào năm 2013. Cho biết quyển sách dày nhất có bao nhiêu trang. In ra tựa sách, tên tác giả của những quyển sách có số trang là P, với P nhập từ bàn phím.

Bài 32. Viết chương trình nhập vào danh sách sinh viên của 1 lớp, có tối đa 80 sinh viên. Mỗi sinh viên gồm các thông tin: mã số, họ tên, quê quán, giới tính, năm sinh, điểm trung bình và xếp loại. Kiểm tra dữ liệu nhập: mã số phải đủ 6 ký tự trong đó 3 ký tự đầu là chữ cái và 3 ký tự cuối là chữ số, họ tên chỉ toàn chữ cái và khoảng trắng nhưng không toàn là khoảng trắng, năm sinh từ 1985 đến 1995, điểm trung bình từ 0.0 đến 4.0 và xếp loại dựa vào điểm trung bình. In ra danh sách các sinh viên đã nhập. Cho biết trong lớp có bao nhiêu sinh viên nữ. In ra họ tên các sinh viên có điểm trung bình cao nhất. Sắp xếp danh sách sinh viên giảm dần theo điểm trung bình.

Bài 33. Quản lý sinh viên

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

- + Sinh viên khoa Công nghệ Thông tin phải học 3 môn Pascal, C# và SQL.

- + Sinh viên khoa Vật lý phải học 4 môn: Cơ học, Điện học, Quang học, Vật lý hạt nhân.

- + Sinh viên khoa Văn phải học 2 môn Văn học cổ điển và Văn học Hiện đại

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

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

Chương 2

WINDOWS FORMS VÀ CONTROLS

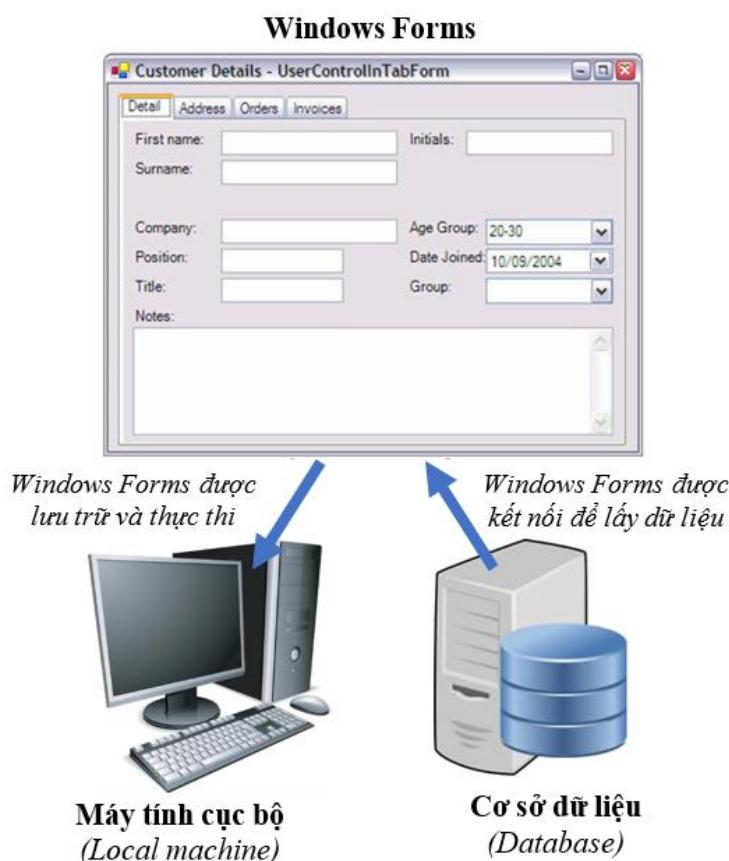
2.1 WINDOWS FORMS

2.1.1 Tổng quan về Windows Forms

Windows Forms là một phần của .NET Framework; là môi trường đa ngôn ngữ, đa nền tảng cho việc xây dựng, triển khai và chạy các ứng dụng. Windows Forms cung cấp mô hình lập trình chuẩn cho phép các nhà phát triển tạo ra các ứng dụng hạn chế tối đa lỗi (bugs) và mâu thuẫn (inconsistencies). Trong .NET Framework các Form được thực thi bởi CLR.

Windows Forms cho phép các nhà phát triển tạo ra những ứng dụng có giao diện người dùng với đa dạng các thành phần được dựng sẵn. Những thành phần này được gọi là các điều khiển (controls), chúng cho phép nhận và hiển thị thông tin theo yêu cầu thông qua Form. Nhờ vào giao diện đồ họa, dễ tương tác, giúp cho các ứng dụng trở nên thân thiện hơn với người dùng.

Sau khi xây dựng giao diện, nhà phát triển có thể sử dụng bất kỳ ngôn ngữ nào được hỗ trợ bởi .NET Framework để cung cấp các chức năng được yêu cầu, từ đó có thể tạo ra các ứng dụng hoạt động dựa trên nền tảng Windows.



Hình 2.1: Vai trò của Windows Forms

Các ứng dụng tạo bằng Windows Forms được thực thi trên máy tính cục bộ (local machine). Các Form có thể truy xuất đến các nguồn tài nguyên của máy cục bộ như bộ nhớ,

máy in, thư mục và các tập tin vì vậy nó phù hợp cho việc tạo ra các ứng dụng desktop và quản lý các nguồn tài nguyên một cách kín đáo.

Vai trò của Windows Forms bao gồm: chứa các điều khiển (controls), xử lý dữ liệu đầu vào do người dùng cung cấp, hiển thị thông tin đến người dùng, kết nối cơ sở dữ liệu (có thể tồn tại trên một máy khác).

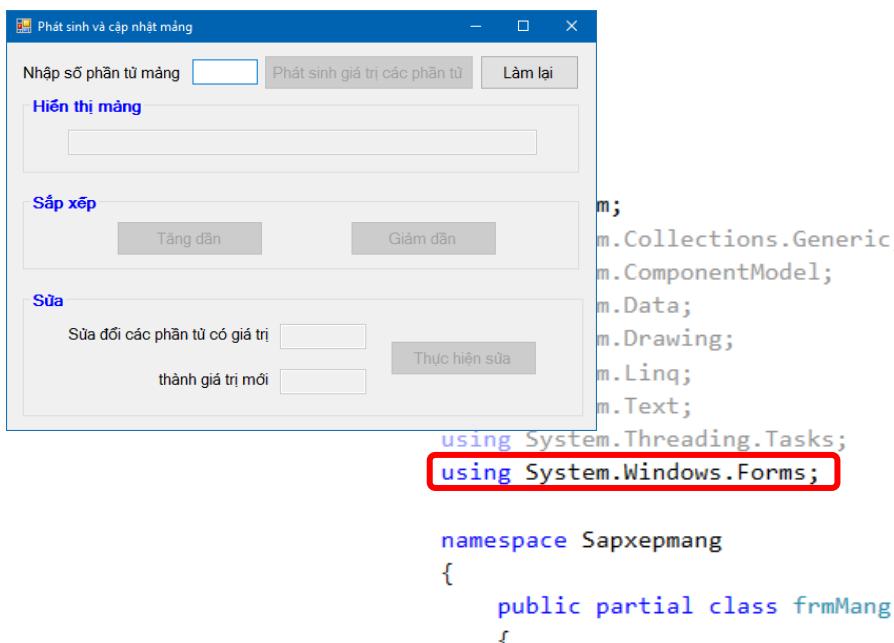
2.1.2 Lớp cơ sở: Form class

Visual studio cung cấp môi trường phát triển tích hợp (Integrated Development Environment, IDE) giữa mã lệnh (code) và giao diện thiết kế với tập các điều khiển phong phú có sẵn. Bằng cách soạn thảo mã lệnh bổ sung cho các chức năng của các điều khiển lập trình viên dễ dàng xây dựng và nhanh chóng đưa ra những giải pháp cho bài toán cần xử lý, từ đó tạo ra các ứng dụng Windows Forms mạnh mẽ.

Form là đơn vị cơ bản của một ứng dụng Windows Forms. Để giải quyết yêu cầu của bài toán lập trình viên cần xác định rõ chức năng và diện mạo cần thiết kế cho Form. Có thể hình dung Form như một phiên bản trắng và lập trình viên, nhà phát triển là các nghệ nhân điều khắc, họ cần có biện pháp “chạm trổ phiên bản” của mình sao cho phù hợp với yêu cầu được đặt ra về cả chức năng và tính thẩm mỹ.

Để tạo một Form hoàn chỉnh, các nhà phát triển ứng dụng sẽ sử dụng các điều khiển cho việc tạo giao diện người dùng cũng như sẽ soạn thảo mã lệnh đáp ứng các chức năng tương tác và thao tác với cơ sở dữ liệu của Form.

Một ứng dụng Windows Forms (WinForms) sẽ có ít nhất một Form (window) được dùng cho việc tương tác giữa ứng dụng và người dùng. Các ứng dụng lớn có thể chứa nhiều Form và các Form sẽ được điều hướng với các điều khiển nhất định. Một Form được tạo ra là một đối tượng của lớp được dẫn xuất từ lớp cơ sở Form class (nằm trong namespace System.Windows.Forms).

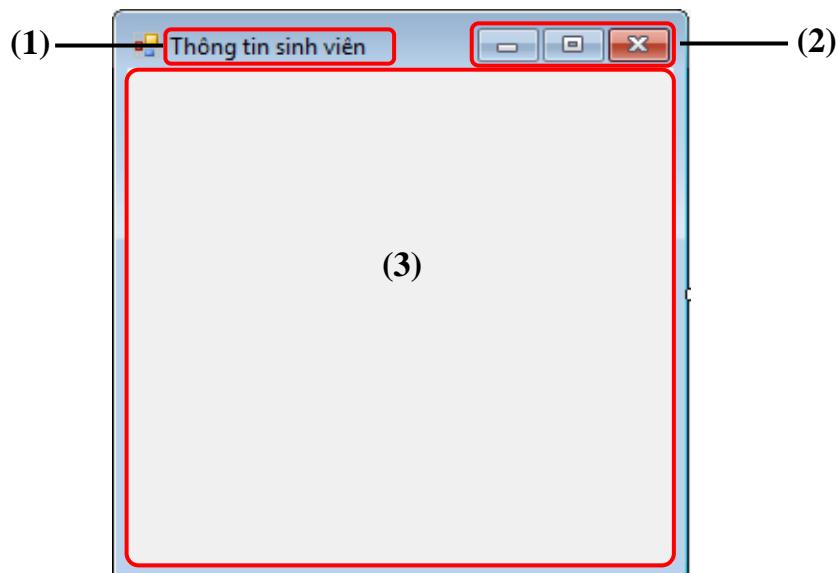


Hình 2.2: Mỗi Form được tạo ra là một đối tượng của lớp dẫn xuất từ Form class

2.1.3 Cấu trúc và thành phần của Form

a/- Cấu trúc Form

Cấu trúc cơ bản của Form bao gồm: (1) thanh tiêu đề hiển thị chức năng của Form; (2) hộp điều khiển chứa Minimize Box, Maximize Box và Close Button (có thể đóng hoặc mở hộp điều khiển dựa trên các properties tương ứng); (3) nội dung Form, nơi chứa các điều khiển (controls)



Hình 2.3: Cấu trúc của Form

b/- Thuộc tính (Properties)

Các thuộc tính (Properties) của Form cho phép điều chỉnh vẻ bề ngoài của Form như kích thước, màu sắc, tên Form, tiêu đề của Form, vị trí Form xuất hiện khi chạy ứng dụng...

Bảng 2.1: Một số thuộc tính Form thường được thao tác

Properties	Mô tả
AcceptButton	Chỉ định button được click trên Form khi người dùng nhấn phím ENTER
BackColor	Chỉ định màu nền của Form
BackgroundImage	Chỉ định hình nền cho Form
ForeColor	Chỉ định màu chữ của các điều khiển trên Form
Location	Chỉ định tọa độ góc trên bên trái của Form (chỉ định vị trí Form xuất hiện)
Modal	Không cho phép các Form khác hoạt động khi form còn đang mở
Name	Chỉ định tên của Form. Lưu ý, tên Form nên bắt đầu với tiền tố frm
Text	Chỉ định nội dung được hiển thị trên thanh tiêu đề của Form
Size	Chiều dài và chiều rộng của Form, tính theo pixel
WindowState	Chỉ định trạng thái của Form khi hiển thị

Ví dụ sau đây sẽ minh họa cách thức các thuộc tính của Form được sử dụng.

Ví dụ 2.1:

```
Form frmStudent = new Form();
```

```

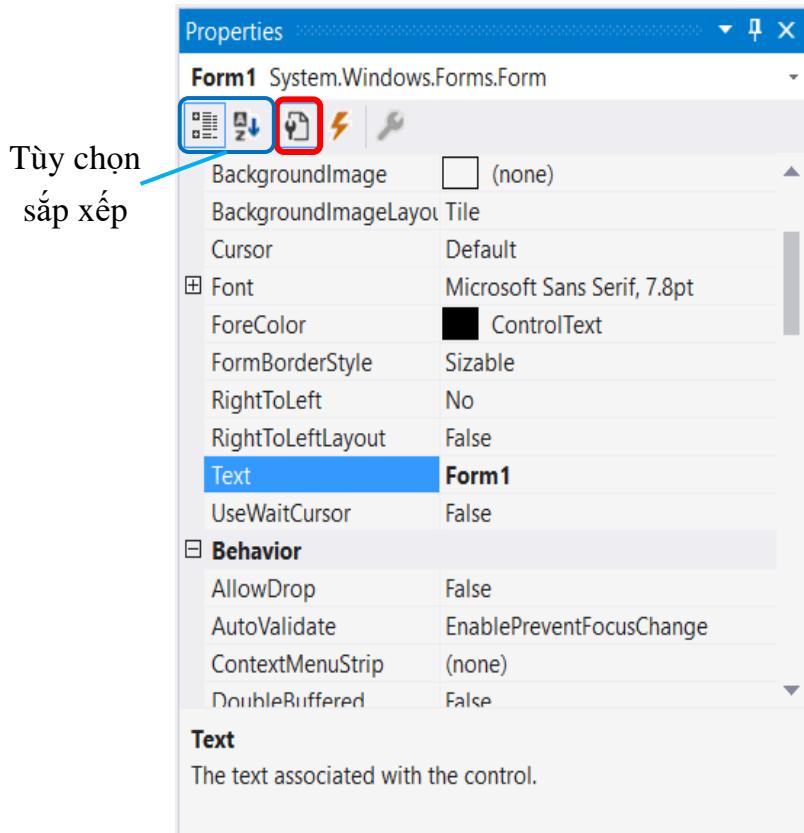
frmStudent.Text = "Thông tin sinh viên";
frmStudent.ForeColor = Color.White;
frmStudent.BackColor = Color.DarkBlue;
frmStudent.WindowState = FormWindowState.Maximized;

```

Đoạn mã nguồn trên tạo ra một đối tượng frmStudent thuộc Form class. Thuộc tính Text của đối tượng cho phép hiển thị nội dung Thông tin sinh viên trên thanh tiêu đề của Form. Thuộc tính ForeColor và BackColor lần lượt cho phép thiết lập màu văn bản của các điều khiển trên Form (nếu không có điều chỉnh đặc biệt ở các điều khiển) và màu nền của Form. Với thuộc tính WindowState được gán giá trị FormWindowState.Maximized cho phép thiết lập trạng thái hiển thị đầy màn hình (full screen) khi Form hiển thị.

Lưu ý: Việc đặt tên Form và các điều khiển trên Form nên tuân thủ theo quy tắc đặt tên với các tiền tố được gợi ý nhằm giúp dễ dàng quản lý mã nguồn.

Trong khung thiết kế (design view), chúng ta có thể thao tác với properties của Form hoặc các điều khiển nhờ vào cửa sổ **Properties** (mặc định nằm ở góc dưới bên phải) với **tùy chọn hiển thị Properties**.



Hình 2.4: Cửa sổ Properties với tùy chọn hiển thị Properties

c/- Phương thức (Methods)

Các phương thức (Methods) của Form cho phép điều chỉnh các hành vi của nó.

Bảng 2.2: Một số phương thức Form được sử dụng phổ biến

Methods	Mô tả
Activate	Kích hoạt Form và đưa focus cho Form
Close	Đóng Form
Focus	Đặt con trỏ nhập liệu vào điều khiển trên Form
Hide	Ẩn Form với người dùng
Show	Hiển thị Form với người dùng

d/- Sự kiện (Events)

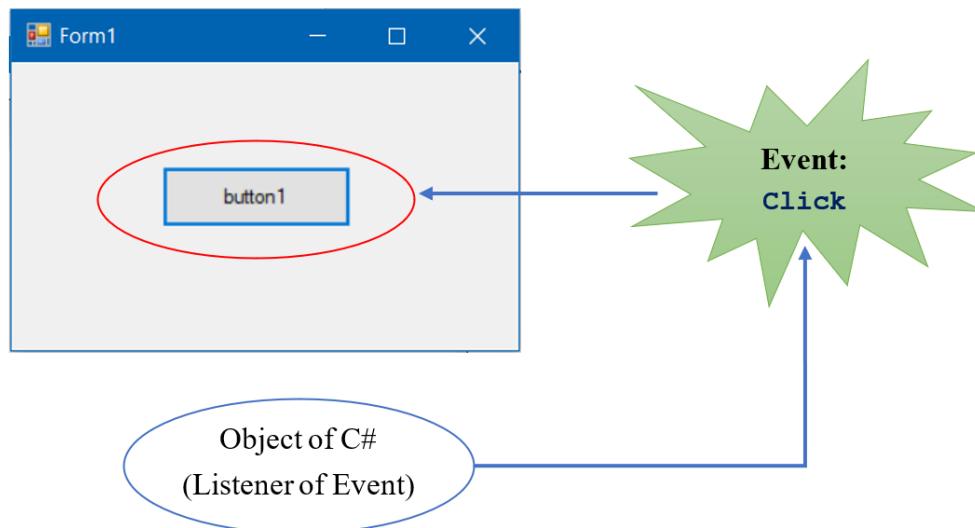
Một sự kiện (Event) cho phép một lớp hoặc một đối tượng thông báo cho các lớp hoặc các đối tượng khác biết khi có điều gì đó liên quan xảy ra.

- Lớp gửi hoặc bật (raise) sự kiện được gọi là các “Publisher”
- Lớp nhận hoặc xử lý sự kiện được gọi là các “Subscriber”

Bảng 2.3: Một số sự kiện thường được sử dụng của Form

Events	Mô tả
Activated	Diễn ra khi Form được kích hoạt trong mã nguồn hoặc bởi người dùng
Click	Diễn ra khi Form được nhấp
Deactivate	Diễn ra khi focus trên Form bị loại bỏ và Form không còn ở trạng thái hoạt động (active)
FormClosed	Diễn ra sau khi Form bị đóng
FormClosing	Diễn ra trước khi Form bị đóng
GotFocus	Diễn ra khi Form nhận được focus
Load	Diễn ra khi Form hiển thị lần đầu tiên
Resize	Diễn ra khi Form bị thay đổi kích thước
Validated	Xảy ra khi Form hoàn thành việc xác thực
Validating	Diễn ra khi Form đang xác thực

Ví dụ phổ biến nhất đó là sự kiện click vào một nút trên Form (sự kiện của nút, không phải của Form).



Hình 2.5: Sự kiện nhấp một nút (Button) trên Form

Sự kiện của Form class cho phép Form đáp lại hành động mà người dùng đã thực hiện trên Form. Để xử lý (handle) các sự kiện, các bộ xử lý (handlers) phải được định nghĩa và các phương thức sẽ được gọi thực thi khi sự kiện được kích hoạt. Như trong ví dụ minh họa trên, bộ xử lý button1_Click sẽ được định nghĩa để xử lý sự kiện click trên button1.

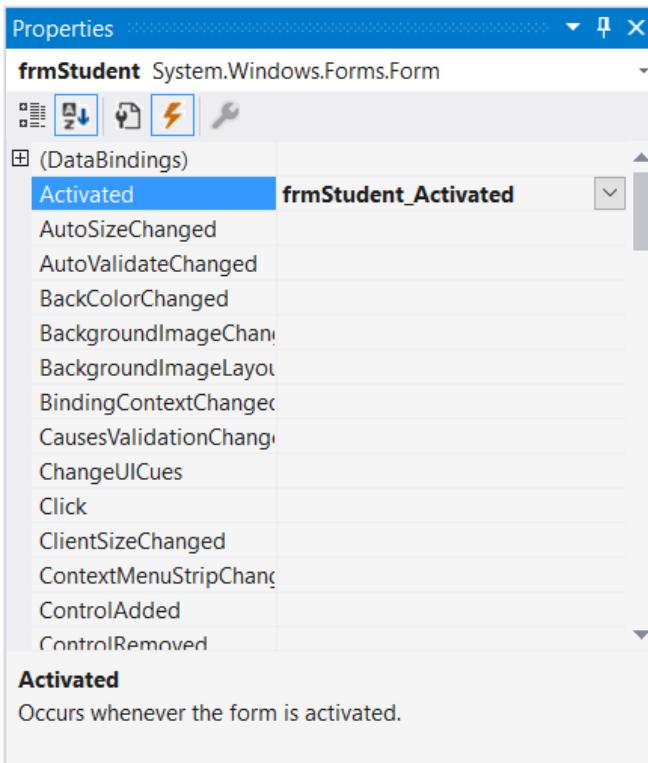
Ví dụ 2.2:

```
private void frmStudent_Activated(object sender, EventArgs e)
{
    this.BackColor = Color.Coral;
}

private void frmStudent_FormClosed(object sender,
FormClosedEventArgs e)
{
    MessageBox.Show("Closing the form");
}
```

Đoạn mã nguồn trong ví dụ 2.2 định nghĩa hai bộ xử lý sự kiện cho các sự kiện Activated và FormClosed. Khi Form được kích hoạt bởi người dùng, bộ xử lý sự kiện frmStudent_Activated được gọi và màu nền của Form được thiết lập là Coral. Khi Form bị đóng, bộ xử lý sự kiện frmStudent_FormClosed được gọi và một hộp thông báo (MessageBox) được hiển thị tuyên bố Form đang đóng.

Để tạo những bộ xử lý cho các sự kiện của Form từ giao diện đồ họa của Form, chúng ta có thể sử dụng **cửa sổ Properties** với tùy chọn hiển thị **Events** như trong hình sau.



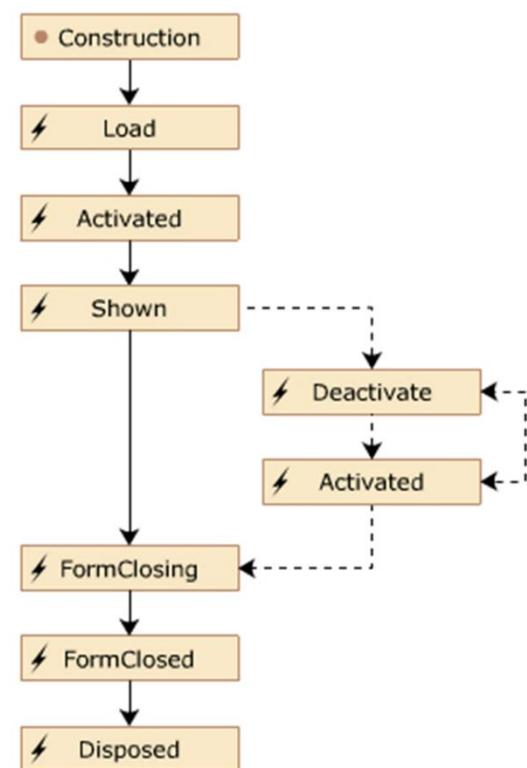
Hình 2.6: Tùy chọn Events của cửa sổ Properties

e/- Vòng đời của Form

“Sự sống” của một Form bắt đầu từ khi một đối tượng thuộc Form class hoặc lớp dẫn xuất từ Form class được tạo ra. Sau đó Form sẽ có các thay đổi trạng thái khác nhau dựa trên sự tương tác với người dùng. Từ đó sinh ra một chuỗi tuần tự các sự kiện diễn ra và được xem như vòng đời của Form. Các sự kiện này được dùng để quản lý và điều khiển các hành vi của Form.

Chuỗi tuần tự các sự kiện bao gồm:

- Khởi tạo (Construction)
- Tải (Load)
- Được kích hoạt (Activated)
- Hiển thị (Shown)
- Vô hiệu hóa (Deactivate)
- Trước khi bị đóng (Closing)
- Sau khi bị đóng (Closed)
- Vứt bỏ (Disposed)



Hình 2.7: Vòng đời của Form

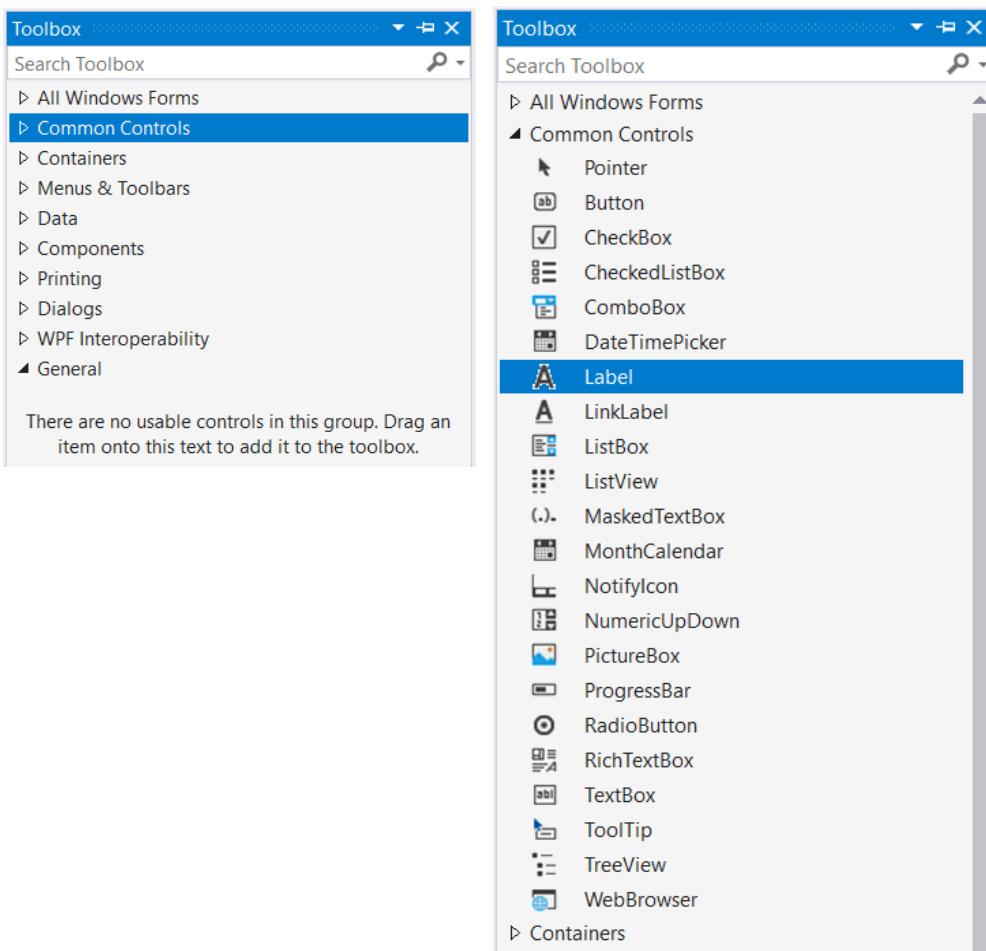
2.2 CÁC ĐIỀU KHIỂN (CONTROLS)

Một điều khiển (control) là một đối tượng trực quan được thêm vào Form để tạo nên giao diện người dùng đồ họa của ứng dụng, được sử dụng với mục đích hiển thị thông tin hoặc nhận thông tin từ người dùng. Ví dụ, Label được dùng để hiển thị thông tin, còn TextBox được dùng để nhận thông tin từ người dùng. Các điều khiển được đặt vào Form với cách thức nhà phát triển ứng dụng đã định trước tạo điều kiện tương tác dễ dàng hơn giữa ứng dụng Windows Forms và người dùng.

2.2.1 Hộp công cụ (Toolbox)

Hộp công cụ (Toolbox) chứa icon của các điều khiển (controls) và các mục (items) khác, các nhà phát triển ứng dụng có thể dễ dàng thêm các điều khiển vào dự án của mình bằng cách kéo thả các icon tương ứng vào Form trong khung thiết kế hoặc dán vào khung soạn thảo mã lệnh.

Để mở Toolbox, click vào **Toolbox** trên menu **View**. Toolbox được tổ chức thành các nhóm (group) chức năng với các điều khiển tương ứng, và tùy theo mục đích sử dụng người dùng có thể lấy điều khiển từ các nhóm này. Lưu ý, để thao tác nhanh với các điều khiển, chúng ta có thể sử dụng khung tìm kiếm điều khiển của Toolbox.



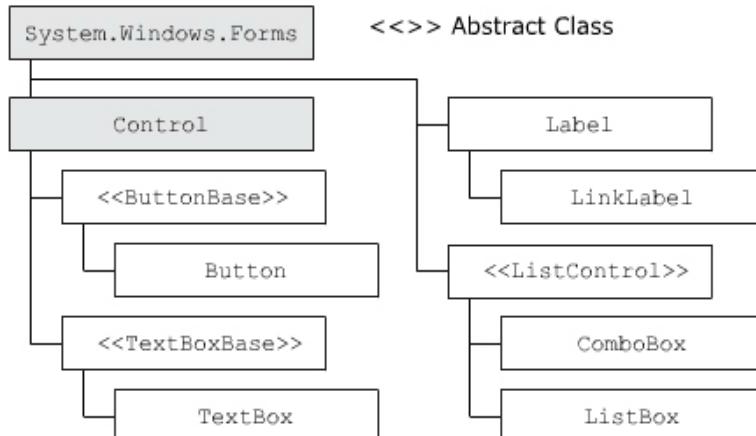
Hình 2.8: Toolbox được tổ chức thành các nhóm chức năng với các điều khiển tương ứng

2.2.2 Các điều khiển cơ bản

Control class được xem là lớp cơ sở của tất cả các điều khiển, được dùng để tùy chỉnh diện mạo của các điều khiển và xử lý dữ liệu nhận được thông qua các điều khiển.

Control class được định nghĩa trong namespace System.Windows.Forms với đa dạng các thuộc tính (properties), phương thức (methods) và sự kiện (events). Các loại điều khiển khác nhau có thể có các thuộc tính, phương thức và sự kiện giống hoặc khác nhau.

Lưu ý, chỉ có các properties và methods của Form mới có thể đọc và được thiết lập bằng từ khóa this. Từ khóa this dùng biểu diễn cho Form hoạt động hiện thời.

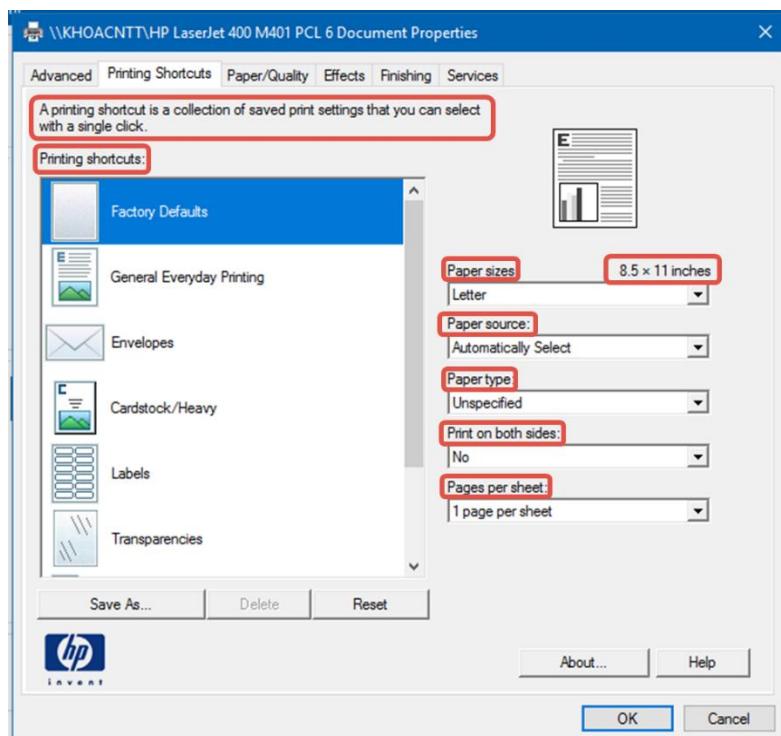


Hình 2.9: “Control” class và các lớp tương ứng được dẫn xuất từ nó.

Các điều khiển được sử dụng phổ biến: Label, TextBox, Button, ListBox, ComboBox, LinkLabel.

Sau đây là một số điều khiển cơ bản trong C#:

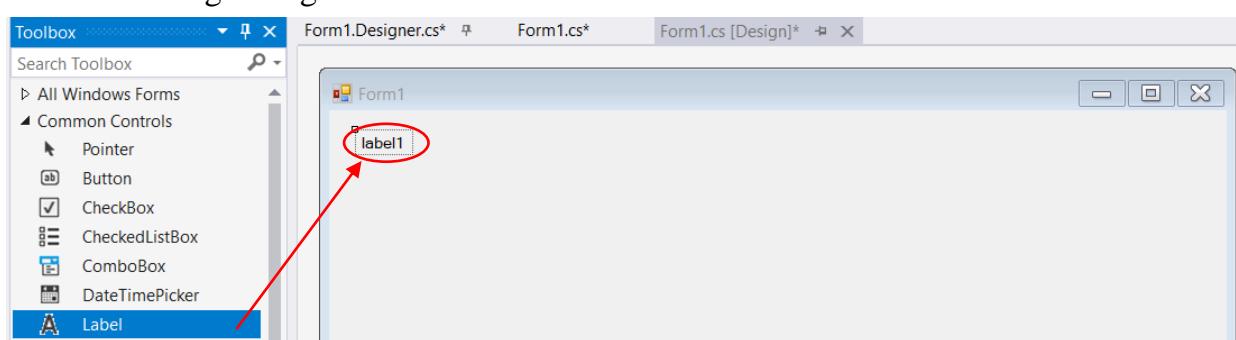
a/- Label



Hình 2.10: Label được sử dụng nhiều trong các ứng dụng Windows Forms

Label (nhãn) được sử dụng cho việc hiển thị thông tin chi tiết mà người dùng không thể thay đổi. Mục đích phổ biến nhất của Label đó là cung cấp chủ thích, tiêu đề cho các điều khiển. Ví dụ, tạo Label “Student Name” làm tiêu đề của TextBox tương ứng cho phép nhập tên của sinh viên. Label thông báo cho người dùng biết họ phải nhập tên sinh viên vào TextBox tương ứng. Label được sử dụng rất nhiều trong các ứng dụng Windows Forms.

Để tạo một đối tượng Label, kéo thả biểu tượng Label tương ứng từ **Toolbox** vào **Form** trong khung thiết kế.



Hình 2.11: Kéo thả Label từ Toolbox vào Form

Tương ứng với thao tác kéo thả thêm Label vào Form, một thể hiện của “Label” class sẽ được tạo ra. Câu lệnh sau đây được dùng để tạo Label.

```

Label <tên đối tượng> = new Label();

private void InitializeComponent()
{
    this.lblTitle = new System.Windows.Forms.Label();
    this.lblUserName = new System.Windows.Forms.Label();
    this.lblPassword = new System.Windows.Forms.Label();
    this.txtUserName = new System.Windows.Forms.TextBox();
    this.txtPassword = new System.Windows.Forms.TextBox();
}

```

Hình 2.12: Các thẻ hiện của “Label” class sẽ được tạo khi kéo thả các Label vào Form

Properties của Label class cho phép chỉ định một số thuộc tính nhất định như caption, alignment, access key của Label

Bảng 2.4: Một số properties của Label class

Properties	Mô tả
Name	Chỉ định hoặc lấy tên của Label. <i>Lưu ý, tên Label nên bắt đầu với tiền tố lbl</i>
Text	Chỉ định hoặc lấy văn bản của Label
TextAlign	Chỉ định hoặc lấy vị trí nội dung văn bản trong Label
Font	Chỉ định phông chữ để hiện thị nội dung văn bản của Label
ForeColor	Chỉ định màu chữ để hiện thị nội dung văn bản của Label
BackColor	Chỉ định màu nền cho Label

Hai **phương thức (methods)** Show () và Hide () được sử dụng nhiều nhất đối với Label, cho phép hiện và ẩn Label tương ứng.

Sự kiện (events) được sử dụng phổ biến với Label đó là Click

❖ LinkLabel

Tương tự Label, nhưng LinkLabel cho phép hiển thị nhãn của nó như một hyperlink. LinkLabel được dùng cung cấp liên kết đến một trang Web hoặc các Form khác. Ví dụ, có thể sử dụng LinkLabel cho việc mở file help để giải thích cho người dùng về cách thức sử dụng ứng dụng.

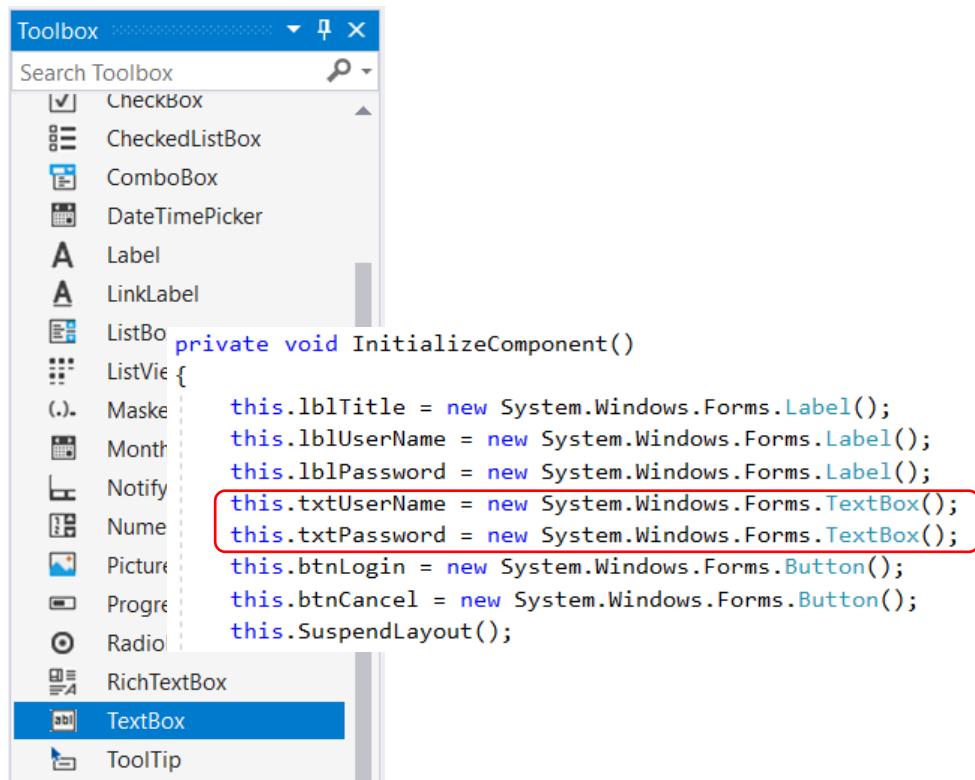
LinkLabel có tất cả các **properties, methods** và **events** của Label class. Ngoài ra LinkLabel cũng sẽ có các thành phần riêng của mình.

b/- TextBox

TextBox nhận thông tin từ người dùng, các thông tin này có thể thay đổi. Ví dụ, tạo TextBox tên txtStudentName để nhận vào tên của sinh viên.

Tương tự như Label, để tạo một TextBox chúng ta thực hiện kéo thả biểu tượng  **TextBox** từ Toolbox vào Form. Kết quả của việc kéo thả này là các thẻ hiện tương ứng của các điều khiển sẽ được tạo ra. Câu lệnh tạo thẻ hiện tương ứng của “TextBox” class.

```
TextBox <tên đối tượng> = new TextBox();
```



Hình 2.13: Thêm TextBox vào Form.

Bảng sau đây thể hiện một số **properties**, **methods** và **events** cần chú ý khi thao tác với điều khiển TextBox.

Bảng 2.5: Một số properties, methods và events của TextBox

Thành phần	Tên	Mô tả
Properties	Name	Chỉ định hoặc lấy tên của TextBox. <i>Lưu ý, tên TextBox nên bắt đầu với tiền tố txt</i>
	Text	Chỉ định hoặc lấy văn bản của TextBox
	CharacterCasing	Chỉ định nội dung nhập vào TextBox sẽ được chuyển sang dạng uppercase hoặc lowercase
	MaxLength	Chỉ định hoặc lấy số ký tự tối đa được nhập vào TextBox
	MultiLine	TextBox có nhận nội dung trên nhiều dòng hay không
	PasswordChar	Chỉ định hoặc lấy ký tự đặc biệt thay thế (mặt nạ) cho các ký tự người dùng nhập vào TextBox (mật khẩu)
	ReadOnly	Nội dung của TextBox có thể điều chỉnh không
Methods	AppendText	Thêm văn bản vào nội dung của TextBox
	Clear	Xóa nội dung văn bản của TextBox
	Focus	Chuyển input focus đến TextBox để nhập liệu
	Copy	Copy nội dung đã chọn trong TextBox vào Clipboard
	Paste	Thay thế nội dung đã chọn trong TextBox bằng nội dung trong Clipboard

Events	KeyPress	Điễn ra khi một phím được nhấn. Sự kiện này sẽ nhận các đối số theo kiểu KeyPressEventEventArgs. Property KeyChar của KeyPressEventEventArgs trả về ký tự tương ứng với phím đã được nhấn. Property Handled của KeyPressEventEventArgs nhận biết sự kiện đã được xử lý hay chưa
	Leave	Xảy ra khi input focus rời khỏi TextBox
	TextChange	Điễn ra khi Text của TextBox bị thay đổi

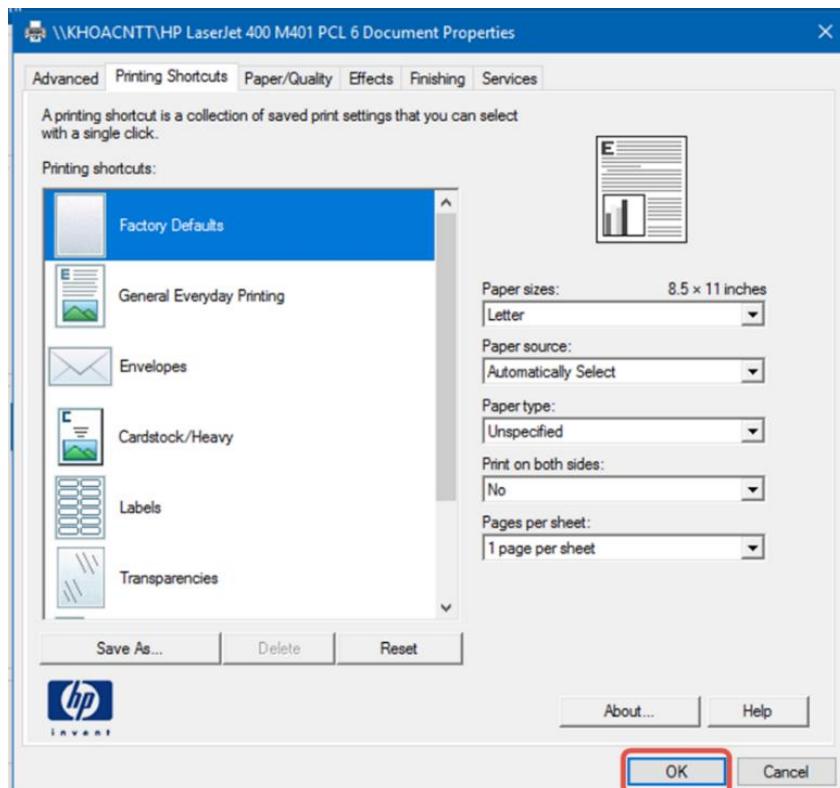
❖ MaskedTextBox

MaskedTextBox được mở rộng từ TextBox cho phép thực hiện việc kiểm tra tính hợp lệ của dữ liệu nhập vào như: tổng số ký tự nhập vào, ký tự số và ký tự chữ cái, theo mẫu, các ký tự đặc biệt như ‘/’ hoặc ‘-’.

Khi một kiểm chứng hợp lệ được chỉ định sử dụng MaskedTextBox, điều khiển sẽ hiện thị mặt nạ yêu cầu số lượng ký tự tương ứng.

c/- Button

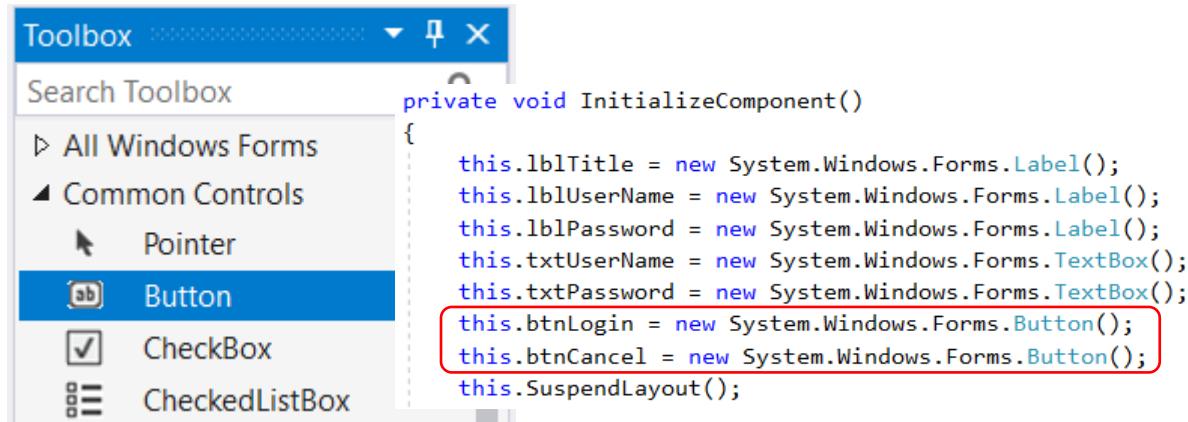
Button (nút) giúp cho việc tương tác giữa người dùng và ứng dụng trở nên dễ dàng hơn. Người dùng có thể nhấp vào Button để yêu cầu ứng dụng thực hiện các hành động được yêu cầu. Ví dụ, nút OK trong hình dưới đây sẽ cho phép người dùng ban hành lệnh in tài liệu đã được chỉ định, hay nút Cancel sẽ cho phép hủy thao tác thiết lập in ấn của người dùng.



Hình 2.14: nút OK trong hộp thoại in ấn

Để tạo một Button trên Form, ta thực hiện kéo thả biểu tượng Button từ Toolbox vào Form. Câu lệnh được sử dụng để tạo Button:

```
Button <tên đối tượng> = new Button();
```



Hình 2.15: Thêm Button vào Form

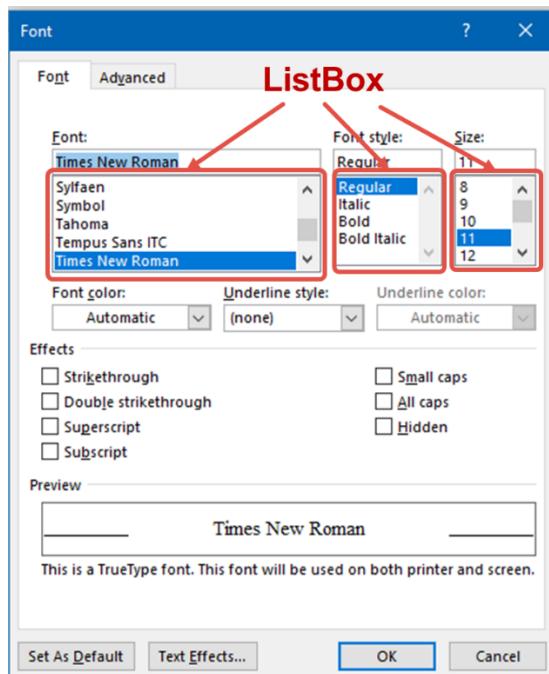
Bảng sau đây thể hiện một số **properties, methods** và **events** cần quan tâm đối với điều khiển Button.

Bảng 2.6: Một số properties, methods và events của Button

Thành phần	Tên	Mô tả
Properties	Name	Chỉ định hoặc lấy tên của Button. <i>Lưu ý, tên Button nên bắt đầu với tiền tố btn</i>
	DialogResult	Chỉ định hoặc lấy giá trị được gửi đến Form cha khi Button được Click
	Enabled	Chỉ định hoặc lấy giá trị nhận định liệu Button có thể phản ứng với hoạt động của người dùng
	Image	Chỉ định hoặc lấy giá trị ảnh của Button
	Text	Chỉ định hoặc lấy văn bản của Button
	Focus	Thiết lập focus trên Button (Button được chọn)
Methods	PerformClick	Tạo sự kiện Click trên Button
Events	Click	Diễn ra khi Button được nhấp
	Enter	Diễn ra khi Button trở thành điều khiển đang hoạt động của Form
	MouseClick	Diễn ra khi Button được nhấp bằng chuột

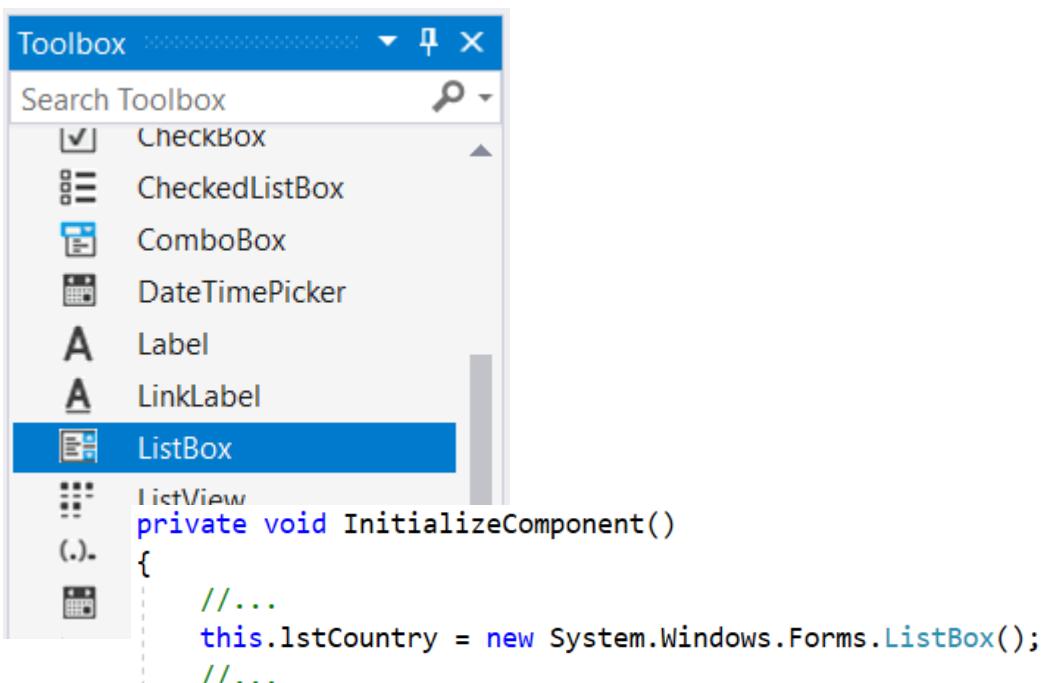
d/- ListBox

ListBox được dùng để chọn một giá trị đơn hoặc nhiều giá trị từ danh sách các giá trị đã cho. Nó được dùng rộng rãi khi người dùng phải chọn một giá trị từ một số lượng lớn các giá trị trong danh sách.



Hình 2.16: Hình ảnh của ListBox trong ứng dụng Windows Forms

ListBox được dùng khi muốn bắt buộc người dùng lựa chọn giá trị trong danh sách đã cho và người dùng không được phép nhập thêm bất kỳ chọn lựa nào khác. Để thêm một ListBox ta thực hiện kéo thả biểu tượng **ListBox** từ Toolbox vào Form.



Hình 2.17: Kéo thả ListBox từ Toolbox vào Form để tạo ListBox mới

Câu lệnh cho phép tạo đối tượng ListBox:

```
ListBox <tên đối tượng> = new ListBox();
```

Bảng sau đây biểu diễn một số **properties, methods** và **events** của ListBox

Bảng 2.7: Một số properties, methods và events của ListBox

Thành phần	Tên	Mô tả
Properties	Name	Chỉ định hoặc lấy tên của ListBox. <i>Lưu ý, tên ListBox nên bắt đầu với tiền tố lst</i>
	Text	Chỉ định hoặc lấy văn bản của item được chọn
	Items	Tập các item của ListBox
	SelectedItem	Chỉ định hoặc lấy item được chọn
	SelectedIndex	Chỉ định hoặc lấy chỉ mục của item được chọn (bắt đầu bởi số 0)
	SelectionMode	None: không cho phép chọn bất kỳ item nào One: cho một lựa chọn duy nhất MultiSimple: cho nhiều lựa chọn MultiExtended: cho nhiều lựa chọn sử dụng Shift + Mouse Click, Shift+Arrow keys, Ctrl + Mouse Click
	DisplayMember	Cho biết thuộc tính của các item được chọn hiển thị trong ListBox.
	ValueMember	Cho biết thuộc tính được dùng như giá trị thực tế của các item trong ListBox
Methods	ClearSelected	Bỏ chọn các item đã chọn trong ListBox
	GetItemText	Trả về nội dung hiển thị của item được chỉ định.
	GetSelected	Trả về giá trị true nếu item được chỉ định đã được chọn và trả về giá trị false nếu item được chỉ định không được chọn
	SetSelected	Thiết lập item chỉ định là được chọn hay không được chọn.
Events	SelectedIndexChanged	Điễn ra khi giá trị của SelectedIndex thay đổi. Mặc định, tất cả các item bị bỏ chọn, giá trị của SelectedIndex ban đầu bằng -1. Khi một item được chọn, giá trị index của nó sẽ được gán vào cho SelectedIndex
	SelectedValueChanged	Điễn ra khi giá trị của thuộc tính SelectedValue bị thay đổi
	ValueMemberChanged	Điễn ra khi giá trị của ValueMember bị thay đổi

Ví dụ 2.3:

```

lstCountry.Items.Add("U.S");
lstCountry.Items.Add("Japan");
lstCountry.Items.Add("Vietnam");
lstCountry.SelectionMode = SelectionMode.MultiExtended;
flag = lstCountry.GetSelected(2);

```

```

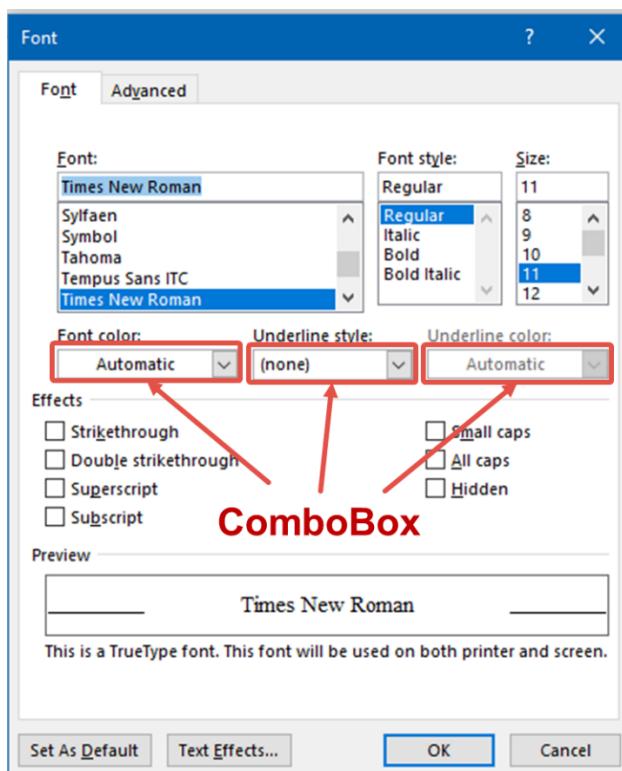
private void lstCountry_SelectedIndexChanged(object sender,
EventArgs e)
{
    MessageBox.Show(lstCountry.SelectedItem.ToString());
}

```

Trong ví dụ trên, phương thức `Add()` được sử dụng để thêm phần tử cho tập `Items` của `lstCountry`. Thuộc tính `SelectionMode` được thiết lập là `MultiExtended` để cho phép người dùng chọn nhiều item cùng lúc. Phương thức `GetSelected()` được dùng để xác định xem liệu item tại vị trí chỉ mục 2 có được chọn hay không. Khi người dùng chọn bất kỳ item nào từ danh sách, giá trị của `SelectedIndex` sẽ thay đổi và sự kiện `SelectedIndexChanged` sẽ được kích hoạt. Kết quả là một hộp thông báo (`MessageBox`) bật lên và hiển thị giá trị văn bản của item đã chọn.

e/- ComboBox

`ComboBox` tương tự như `ListBox`, nhưng chỉ cho phép chọn duy nhất một giá trị.Thêm vào đó, người dùng có thể nhập giá trị mới không có trong danh sách item tại vùng điều chỉnh của `ComboBox`.

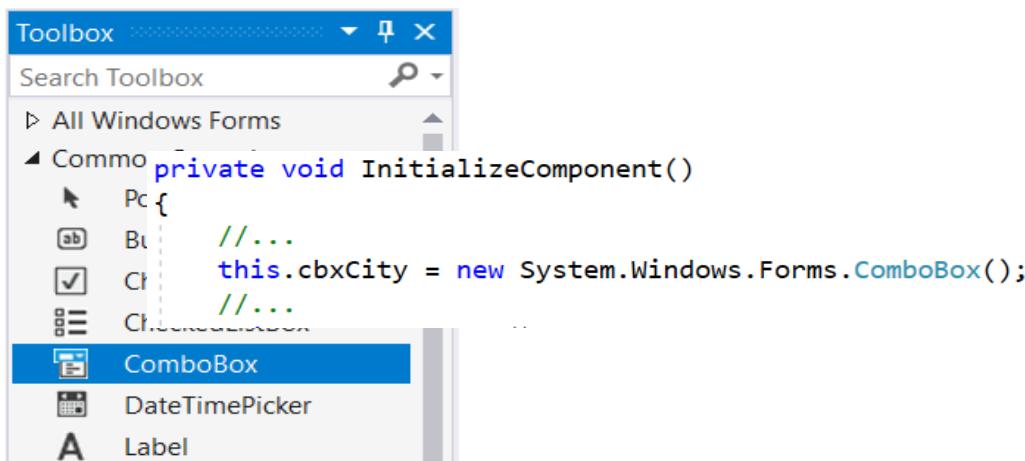


Hình 2.18: Hình ảnh của `ComboBox` trong các ứng dụng Windows Forms.

`ComboBox` là sự kết hợp giữa `ListBox` (vùng thả xuống) và `TextBox` (vùng điều chỉnh). Để thêm một `ComboBox` vào Form ta thực hiện kéo thả biểu tượng `ComboBox` từ Toolbox và Form hoặc dùng lệnh để tạo theo cú pháp sau:

```
ComboBox <tên đối tượng> = new ComboBox();
```

Ví dụ 2.4: ComboBox cbxCity = new ComboBox();



Hình 2.19. Kéo thả ComboBox từ Toolbox vào Form để tạo ComboBox mới
Một số **Properties, methods** và **events** phổ biến của ComboBox được thể hiện trong bảng dưới đây.

Bảng 2.8: Một số properties, methods, events của ComboBox

Thành phần	Tên	Mô tả
Properties	Name	Chỉ định hoặc lấy giá trị tên của ComboBox. Lưu ý, tên ComboBox nên bắt đầu với tiền tố cbx
	Text	Chỉ định hoặc lấy văn bản của Combobox
	Items	Tập chứa các item của ComboBox
	DisplayMember	Cho biết thuộc tính của các item được chọn hiển thị trong ComboBox.
	ValueMember	Cho biết thuộc tính được dùng như giá trị thực tế của các item trong ComboBox
	DropDownStyle	Kiểm soát hiển thị và chức năng của ComboBox (được định nghĩa trong enum ComboBoxStyle)
	MaxDropDownItems	Chỉ định hoặc lấy số lượng phần tử tối đa được hiển thị ở phần thả xuống của ComboBox
	SelectedItem	Chỉ định hoặc lấy item được chọn
	SelectedIndex	Chỉ định hoặc lấy chỉ mục của item được chọn (bắt đầu bởi số 0)
Methods	GetItemText	Lấy văn bản của item được chỉ định
	SelectAll	Chọn tất cả văn bản trong vùng điều chỉnh của ComboBox
	Select	Chọn một phần văn bản trong vùng điều chỉnh của ComboBox
Events	DropDown	Điễn ra khi vùng thả xuống của ComboBox hiển thị
	SelectedIndexChanged	Tương tự ListBox
	SelectedValueChanged	Tương tự ListBox
	ValueMemberChanged	Tương tự ListBox

Ví dụ 2.5:

```

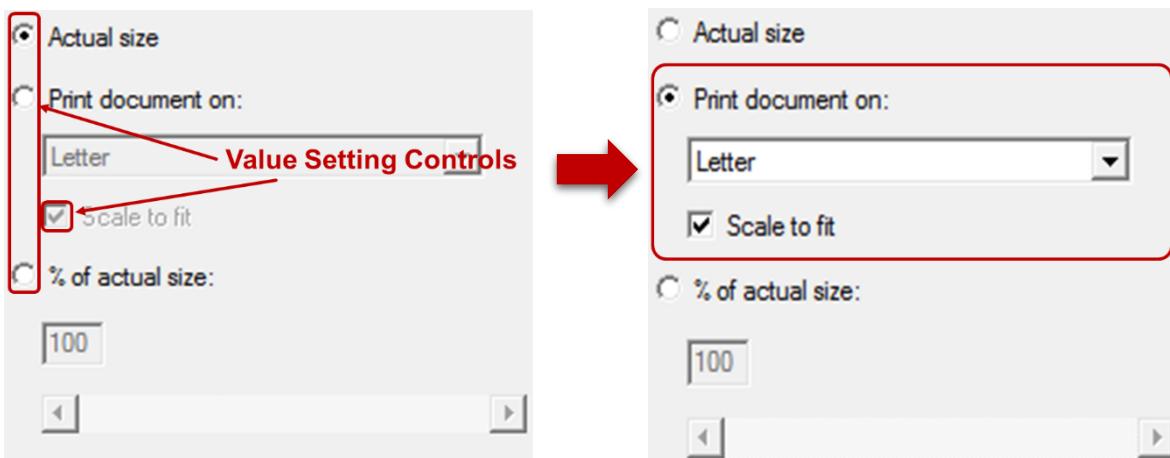
cbxCity.Text = "(Select)";
cbxCity.Items.Add("Ha Noi");
cbxCity.Items.Add("Da Nang");
cbxCity.Items.Add("HCM city");
cbxCity.Items.Add("Can Tho");
cbxCity.MaxDropDownItems = 3;
cbxCity.DropDownStyle= ComboBoxStyle.DropDownList;
private void cbxCity_SelectedIndexChanged(object sender,
EventArgs e)
{
    MessageBox.Show("You have selected " +
    cbxCity.SelectedItem.ToString());
}

```

Trong ví dụ, thuộc tính `Text` được dùng để thiết lập văn bản (text) của `ComboBox` `cbxCity` với nội dung (Select). Tương tự trong `ListBox`, phương thức `Add()` cho phép thêm phần tử cho tập `Items` của `cbxCity`. Giá trị của `MaxDropDownItems` được thiết lập là 3 để chỉ ra rằng chỉ có 3 item được hiển thị trong phần thả xuống của `cbxCity`. Kiểu của điều khiển được thiết lập là `DropDownList`. Khi người dùng chọn vào một item của danh sách, sự kiện `SelectedValueChanged` sẽ được kích hoạt và một hộp thông báo (`MessageBox`) bật lên và hiển thị giá trị văn bản của item đã chọn.

f/- Các điều khiển thiết lập giá trị (Value Setting Controls)

Các điều khiển thiết lập giá trị cho phép chọn một hoặc nhiều giá trị yêu cầu từ danh sách các giá trị đã có.



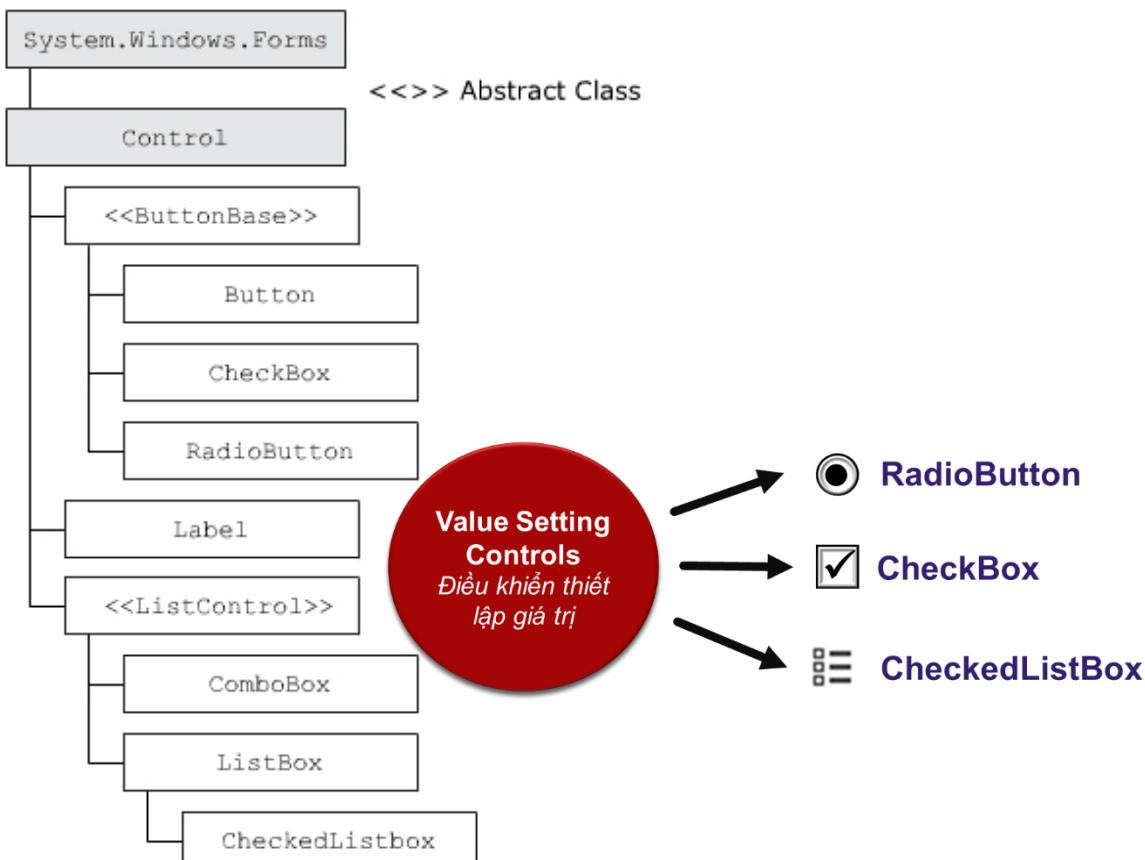
Hình 2.20: Các điều khiển thiết lập giá trị

Xem xét kịch bản nhà phát triển ứng dụng muốn tạo ra một ứng dụng cho việc lưu các thông tin cá nhân của nhân viên tại một công ty. Ứng dụng phải cho phép giám đốc nhân sự chọn giới tính và trình độ của nhân viên từ hai danh sách tách biệt. Với yêu cầu

đó, cần phải có các điều khiển cho phép người dùng chọn các giá trị từ tập các giá trị đã cho.

Trong trường hợp này các điều khiển thiết lập giá trị sẽ được chọn vì chúng cho phép chọn các giá trị mong muốn từ danh sách các giá trị đã có.

Có 3 loại thuộc nhóm điều khiển thiết lập giá trị: RadioButton, CheckBox, CheckedListBox.

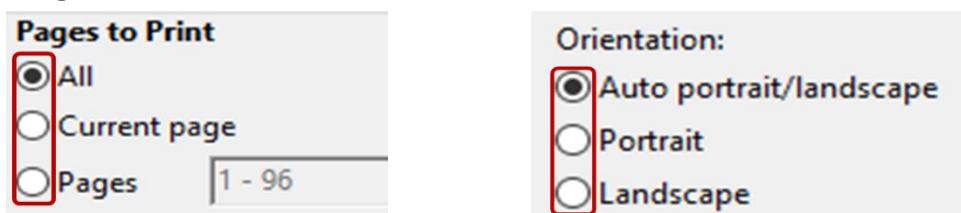


Hình 2.21: Các điều khiển thuộc nhóm Điều khiển thiết lập giá trị

❖ RadioButton

`RadioButton` cho phép người dùng chọn một giá trị đơn từ tập các giá trị đã cho, có nghĩa là chỉ được phép chọn một giá trị ở mỗi lần chọn.

`RadioButton` được sử dụng cho các tùy chọn mang tính loại trừ lẫn nhau. Ví dụ, giới tính có 2 tùy chọn loại trừ nhau: **Nam** và **Nữ**; kết quả thi có 2 tùy chọn loại trừ nhau: **Đạt** và **Không đạt**.



Hình 2.22: Điều khiển RadioButton được sử dụng cho các tùy chọn mang tính loại trừ nhau

Để thêm RadioButton vào Form ta có thể kéo thả biểu tượng **RadioButton** từ Toolbox hoặc sử dụng lệnh.

```
RadioButton <tên đối tượng> = new RadioButton();
```

Ví dụ 2.6:

```
RadioButton rdbNam = new RadioButton();
RadioButton rdbNu = new RadioButton();
RadioButton rdbAll = new RadioButton();
```

Sau đây là một số **properties, methods, và events** của RadioButton.

Bảng 2.9: Một số properties, methods và events của RadioButton

Thành phần	Tên	Mô tả
Properties	Appearance	Chỉ định hoặc nhận cách thức hiển thị của điều khiển. RadioButton sẽ xuất hiện ở dạng mặc định hay trong hình dạng của một Button.
	AutoCheck	Chỉ định hoặc nhận giá trị thuộc tính Checked và vẻ bề ngoài của RadioButton có tự động thay đổi không
	Checked	Chỉ định hoặc nhận giá trị RadioButton đã được check hay chưa.
	Image	Chỉ định hoặc nhận về giá trị hình ảnh xuất hiện trên điều khiển
	Name	Chỉ định hoặc lấy giá trị tên của RadioButton. Lưu ý, tên RadioButton nên bắt đầu với tiền tố rdb
Methods	PerformClick	Tạo sự kiện Click trên RadioButton
	Select	Kích hoạt RadioButton
	Show	Hiển thị RadioButton
Events	CheckedChanged	Điễn ra khi có sự thay đổi giá trị của thuộc tính Checked
	Click	Điễn ra khi điều khiển RadioButton được click

Ví dụ 2.7: Thao tác với properties, methods và events của RadioButton

```
rbNam.Text = "Nam";
rbNam.Checked = false;
rbNam.Show();
rbNu.Text = "Nữ";
rbNu.Checked = false;
rbNu.Show();

private void rbNam_CheckedChanged(object sender, EventArgs e)
{
    if (rbNam.Checked == true) {
```

```

        MessageBox.Show("Bạn đã chọn: " + rdbNam.Text);
        rdbNu.Checked = false;
    }
}

private void rdbNu_CheckedChanged(object sender, EventArgs e)
{
    if (rdbNu.Checked == true) {
        MessageBox.Show("Bạn đã chọn: " + rdbNu.Text);
        rdbNam.Checked = false;
    }
}

```

Lưu ý: Chỉ những RadioButton trong cùng một nhóm mới loại trừ nhau. Nếu không gom nhóm các điều khiển, các RadioButton trên Form sẽ loại trừ lẫn nhau.

❖ CheckBox



Hình 2.23: Hình ảnh của điều khiển CheckBox
trong các ứng dụng Windows Forms

Điều khiển CheckBox cho phép nhận một hoặc nhiều giá trị từ danh sách các giá trị đã cho. Chúng ta có thể sử dụng CheckBox khi muốn người dùng chọn một hoặc nhiều tùy chọn cùng lúc.

Một CheckBox mới được thêm vào Form bằng cách kéo thả biểu tượng từ Toolbox hoặc soạn thảo lệnh.

```
CheckBox <tên đối tượng> = new CheckBox();
```

Sau đây là một số properties, methods và events của điều khiển CheckBox

Bảng 2.10: Một số properties, methods và events của CheckBox

Thành phần	Tên	Mô tả
Properties	Checked	Tương tự RadioButton
	CheckState	Chỉ định hoặc nhận về trạng thái của điều khiển. CheckBox sẽ có 3 trạng thái Checked, Indeterminate và Unchecked.
	Name	Chỉ định hoặc lấy giá trị tên của CheckBox. Lưu ý, tên CheckBox nên bắt đầu với tiền tố chb
	ThreeState	Có cho phép CheckBox nhận 3 trạng thái check thay vì 2 trạng thái check hay không

Methods	Select	Tương tự RadioButton
	Show	Tương tự RadioButton
Events	CheckedChanged	Tương tự RadioButton
	CheckStateChanged	Điễn ra khi thuộc tính CheckState thay đổi giá trị
	Click	Tương tự RadioButton

Ví dụ 2.8: Thao tác với properties, methods và events của CheckBox

```
CheckBox chbCD = new CheckBox();
CheckBox chbVCD = new CheckBox();
chbCD.Text = "CD";
chbCD.Checked = false;
chbCD.CheckState = CheckState.Indeterminate;
chbCD.Show();
chbVCD.Text = "VCD";
chbVCD.Checked = false;
chbVCD.CheckState = CheckState.Unchecked;
chbVCD.Show();
private void chbCD_CheckedChanged(object sender, EventArgs e)
{
    if (chbCD.Checked == false)
    {
        MessageBox.Show("Chỉ yêu cầu CD", "Chi tiết khách hàng",
            MessageBoxButtons.OK,
            MessageBoxIcon.Information);
        chbCD.Checked = false;
        chbCD.CheckState = CheckState.Indeterminate;
    }
}
```

❖ CheckedListBox

CheckedListBox cho phép nhận nhiều giá trị từ một danh sách. Tương tự như ListBox nhưng mỗi giá trị trong danh sách là một CheckBox có thể được check (Checked) hoặc không (Unchecked). Được sử dụng khi muốn hiển thị thêm thông tin với người dùng.

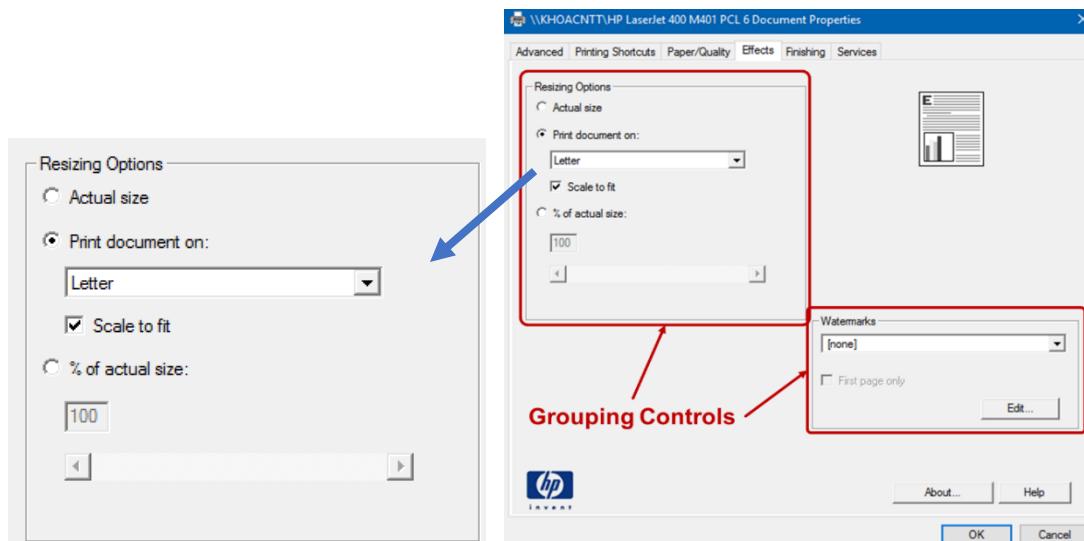
Người dùng có thể thấy tất cả giá trị trong danh sách bằng cách sử dụng thanh cuộn xuất hiện ở bên phải của CheckedListBox.

Ví dụ 2.9: khi cài đặt các bộ ứng dụng (MS Office, SQL Server, Visual Studio), CheckedListBox được sử dụng hiển thị các thành phần của ứng dụng mà người dùng có thể lựa chọn để cài đặt.

g/- Các điều khiển gom nhóm (Grouping Controls)

Các điều khiển gom nhóm có chức năng chứa đựng các điều khiển khác, cho phép tổ chức các điều khiển trong một nhóm. Việc gom nhóm cần thiết khi nhà phát triển muốn đưa ra các thông tin riêng biệt dựa trên kiểu hoặc loại nhất định. Ví dụ, có thể dùng các điều khiển gom nhóm để hiển thị danh sách các thành viên của công ty theo chi nhánh hoặc phòng ban.

Có hai loại điều khiển gom nhóm được sử dụng phổ biến: Panel và GroupBox.



Hình 2.24: Hình ảnh của GroupBox trong các ứng dụng Windows Forms

❖ Panel

Panel cho phép đặt những điều khiển có liên quan nhau cho ra khung nhìn hợp logic với người dùng, nó hoạt động như một vật để chứa một nhóm riêng biệt các điều khiển. Mặc định, Panel không có bất kỳ thẻ hiện nào trên Form, để làm cho Panel xuất hiện nhà phát triển cần thiết lập đường viền cho nó. Tên của các điều khiển Panel nên bắt đầu với tiền tố **pnl**.

❖ GroupBox

GroupBox tương tự như Panel và được dùng để gom nhóm các điều khiển, khi thêm GroupBox vào Form sẽ xuất hiện một khung bao quanh các điều khiển mà nó giữ. Có thể cấp một nhãn cho GroupBox như một tiêu đề chỉ định loại thông tin được nhận hoặc hiện thị. Ví dụ, có thể tạo ra một GroupBox để chọn giới tính và trình độ giáo dục của các nhân viên. Tên của các điều khiển GroupBox nên bắt đầu bằng tiền tố **grp**.

❖ So sánh giữa GroupBox và Panel

Bảng 2.11: So sánh giữa GroupBox và Panel

GroupBox	Panel
Có nhãn	Không có nhãn
Hiển thị trên Form với đường viền	Không có thẻ hiện trên Form
Không có hỗ trợ các thanh cuộn (scroller)	Hỗ trợ các thanh cuộn (scroller)

❖ SplitContainer

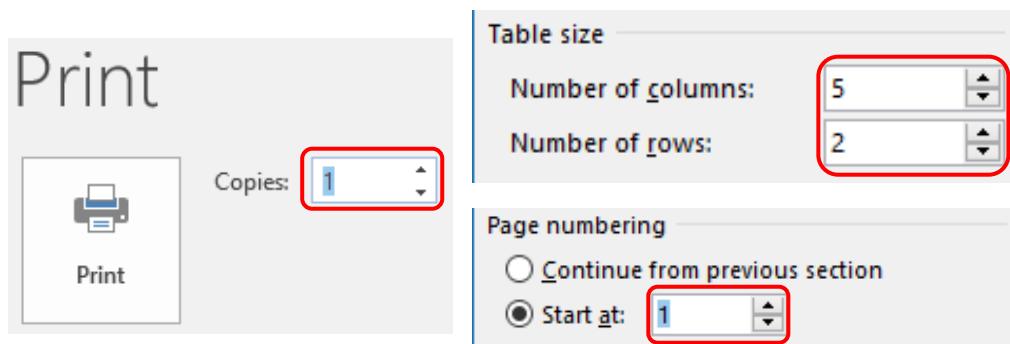
SplitContainer cho phép tạo ra các giao diện người dùng phức tạp bằng cách chia Form thành hai Panel có thể điều chỉnh kích thước, hoặc là theo chiều ngang hoặc là theo chiều dọc. Hai Panel được tách biệt bởi một thanh có thể di chuyển gọi là thanh splitter (splitter bar). Than này được dùng để thay đổi kích thước của vùng hiển thị của từng Panel riêng lẻ.

SplitContainer được sử dụng khi muốn hiển thị thông tin theo dạng thứ bậc trong một Panel và các nội dung con của nó trong một Panel khác. Xuất hiện trong Form tương tự như cửa sổ Windows Explorer, gồm 2 Panel.

2.2.3 Các điều khiển nâng cao

a/- NumericUpDown

NumericUpDown là một điều khiển danh sách lựa chọn cho phép chọn các giá trị số từ một loạt các giá trị. NumericUpDown chỉ cho phép chọn một giá trị ở mỗi lần chọn, ví dụ như tùy chọn số bản sao trong hộp thoại in ấn. Điều khiển này hữu ích đối với dữ liệu tập hợp như số lượng nhân viên, số lượng sinh viên trong một lớp và số ngày.



Hình 2.25:Hình ảnh của NumericUpDown trong các ứng dụng Windows Forms

Để tạo một NumericUpDown trên Form chúng ta có thể thực hiện kéo thả biểu tượng

 **NumericUpDown** từ Toolbox vào Form hoặc sử dụng lệnh. Tên của các đối tượng NumericUpDown nên bắt đầu với tiền tố **upd**.

```
NumericUpDown <đối tượng> = new NumericUpDown();
```

Ví dụ 2.10:

```
NumericUpDown updPrint = new NumericUpDown();
updPrint.Minimum = 0;
updPrint.Maximum = 100;
updPrint.Increment = 1;
updPrint.UpButton();
private void updPrint_ValueChanged(object sender, EventArgs e)
{
    MessageBox.Show("The value of NumericUpDown was changed
to " + updPrint.Value);
}
```

b/- DomainUpDown

DomainUpDown là một điều khiển danh sách lựa chọn cho phép chúng ta chọn các giá trị văn bản (text) từ tập các giá trị đã có. Tương tự như NumricUpDown, DomainUpDown cũng chỉ cho phép chọn một giá trị ở mỗi lần chọn và việc chọn giá trị có thể được thực hiện bằng cách sử dụng các mũi tên lên xuống của điều khiển. Nhìn chung, điều khiển này được sử dụng cho dãy các giá trị có thứ tự như ngày trong tuần hoặc tháng trong năm.

DomainUpDown được thêm vào Form với thao tác kéo thả biểu tượng DomainUpDown từ Toolbox hoặc sử dụng lệnh. Các đối tượng DomainUpDown nên được đặt tên với tiền tố **dud**.

```
DomainUpDown <tên đối tượng> = new DomainUpDown();
```

Ví dụ 2.11:

```
DomainUpDown dudMonth = new DomainUpDown();
dudMonth.Items.Add("January");
dudMonth.Items.Add("February");
dudMonth.Items.Add("March");
dudMonth.Items.Add("April");
dudMonth.MaximumSize = new Size(100, 100);
dudMonth.MinimumSize = new Size(50, 50);
dudMonth.DownButton();
private void dudMonth_SelectedIndexChanged(object sender,
EventArgs e)
{
    MessageBox.Show("The selected item of DomainUpDown was
changed to " + dudMonth.Text);
}
```

c/- ListView

Điều khiển ListView được dùng để hiển thị tập các mục trong một danh sách, cho phép thêm các mục vào tập hợp và hiển thị các mục với các biểu tượng của chúng. Các biểu tượng này có thể có kích thước nhỏ hoặc lớn. Điều khiển này xuất hiện tương tự như biểu tượng Views trên thanh công cụ chuẩn của Windows Explorer. ListView có 5 khung nhìn (views) chính bao gồm: Tile, List, Detail, SmallIcon, LargeIcon.

Ngoài thao tác kéo thả từ Toolbox vào Form, ListView còn có thể được tạo ra bằng câu lệnh theo cú pháp sau:

```
ListView <tên đối tượng> = new ListView();
```

Tên của các đối tượng ListView nên được đặt bắt đầu với tiền tố **lvw**

Ví dụ 2.12:

```
ListView lvwStaff = new ListView();
```

```

lvwStaff.Items.Add("Adam");
lvwStaff.Items.Add("Nicole");
lvwStaff.Items.Add("Maria");
lvwStaff.Items.Add("Stephen");
lvwStaff.MultiSelect = true;

private void lvwStaff_Click(object sender, EventArgs e)
{
    if (lvwStaff.View.Equals(View.LargeIcon))
        lvwStaff.View = View.List;
    else if (lvwStaff.View.Equals(View.List))
        lvwStaff.View = View.LargeIcon;
}

```

d/- TreeView

Điều khiển TreeView hiển thị dữ liệu theo phương pháp cấp bậc. Điều khiển này tương tự như cửa sổ (pane) bên trái của Windows Explorer. Điều khiển TreeView sẽ có ba loại nút: Root, Parent, Leaf.

Để thêm một TreeView trên Form, chúng ta có thể kéo thả nó từ Toolbox hoặc dùng lệnh để tạo ra một TreeView mới theo cú pháp sau. Lưu ý, các đối tượng TreeView nên được đặt tên với tiền tố tvw.

```
TreeView <tên đối tượng> = new TreeView();
```

Ví dụ 2.13:

```

TreeView tvwProjects = new TreeView();
tvwProject.Nodes.Add("Projects");
tvwProjects.Nodes[0].Nodes.Add("IT");
tvwProjects.Nodes[0].Nodes[0].Nodes.Add("Hi-Fly AirLines");
tvwProjects.Nodes[0].Nodes[0].Nodes.Add("e-Commerce");
tvwProjects.Nodes[0].Nodes[1].Nodes.Add("Inova Bank");
tvwProjects.SelectedNode = tvwProject.Nodes[0];
tvwProjects.Sort();
private void tvwProjects_AfterExpand(object sender,
TreeViewEventArgs e)
{
    tvwProjects.SelectedNode =
    tvwProjects.SelectedNode.Nodes[0];
}

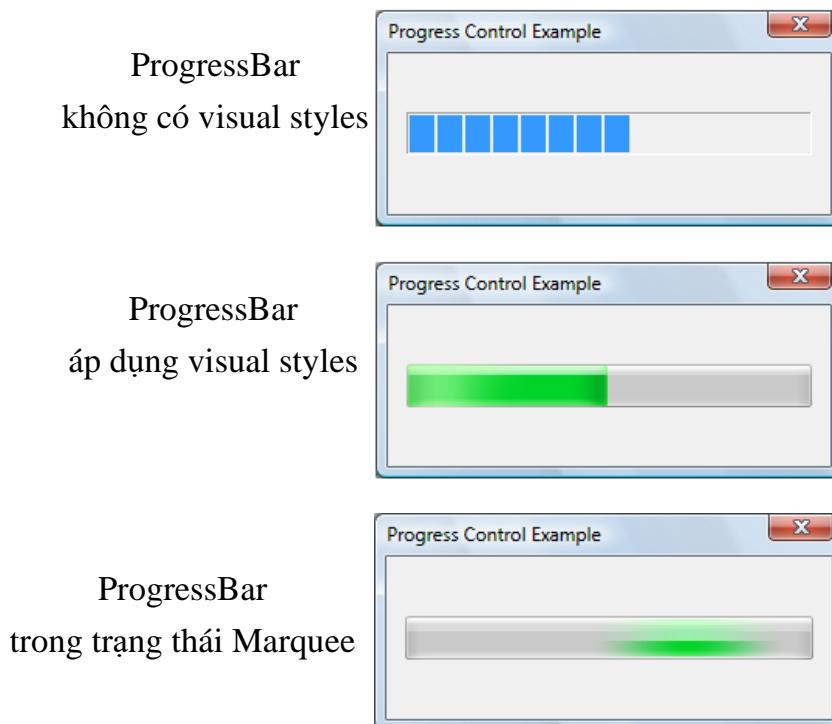
```

Mỗi nút trong TreeView là một thể hiện của nút TreeNode class. Đây là class được dùng để quản lý các nút trong điều khiển. Ví dụ, chúng ta có thể điều hướng thông qua các nút và sử dụng các thuộc tính của TreeNode class để xác định trạng thái cũng như vị trí của một nút.

e/- RichTextBox

Điều khiển RichTextBox dùng để hiển thị, nhập và thao tác với dữ liệu văn bản. Điều khiển này tương tự như ứng dụng WordPad, nó cho phép định dạng văn bản như in đậm văn bản, áp dụng bullets và links, thay đổi phông chữ của văn bản. RichTextBox cũng cho tải văn bản và nhúng hình ảnh từ tập tin, cung cấp chức năng undo và redo, và cho phép chúng ta tìm những ký tự đặc biệt.

RichTextBox mặc định sẽ hiển thị các thanh cuộn ngang và dọc. Văn bản trong RichTextBox hoặc là được truy xuất dưới dạng định dạng text hoặc rich text (.rtf). Các đối tượng RichTextBox nên được đặt tên bắt đầu với tiền tố **rtb**.

f/- ProgressBar

Hình 2.26: Hình ảnh của ProgressBar với các kiểu chỉ định khác nhau.

Điều khiển ProgressBar là một cửa sổ được dùng để biểu thị tiến trình của một hoạt động trong ứng dụng. Cửa sổ này sẽ bao gồm một vùng hình chữ nhật hình động như một tiến trình hoạt động, qua đó cho chúng ta biết mất bao lâu để tiến trình có thể hoàn thành.

ProgressBar có thể hiển thị có hoặc không có kiểu trực quan (visual styles). Đối với các kiểu trực quan, nền và hiệu ứng trực quan của ProgressBar sẽ thực thi theo chủ đề (theme) hiện thời của hệ điều hành.

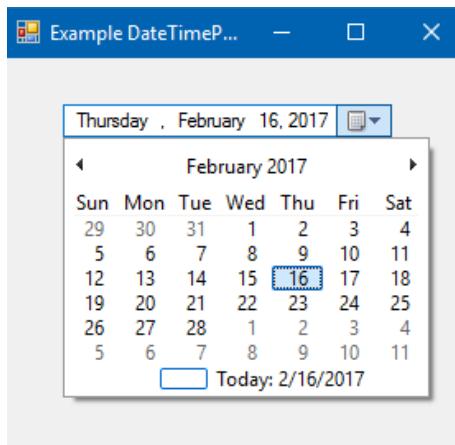
Mặc định ProgressBar sẽ chuyển động theo tỉ lệ hoàn thành tác vụ. Chúng ta có thể thay đổi Style của ProgressBar thành Marquee để nó hiển thị hoạt động nhưng không biểu thị tỉ lệ tác vụ được hoàn thành. Phần được làm nổi bật của ProgressBar sẽ di chuyển qua lại dọc theo chiều dài của thanh ProgressBar.

2.2.4 Các điều khiển Date và thành phần Timer

Các điều khiển date và time được dùng cho việc chọn ngày tháng năm và thời gian. Chúng cung cấp một “quyển lịch” có định dạng đồ họa, cho phép người dùng dễ dàng chọn một ngày cụ thể. Từ đó, giúp tránh được những lỗi có thể xảy ra do nhập ngày tháng năm hoặc thời gian bằng tay, đồng thời cũng ngăn chặn được việc người dùng nhập những giá trị ngày không hợp lệ.

a/- DateTimePicker

Điều khiển DateTimePicker cho phép chọn một mục từ danh sách ngày tháng năm và thời gian. Nó xuất hiện trên Form như một TextBox cùng với một drop-down calendar và sẽ hiển thị ngày hiện tại với một định dạng cụ thể. Calendar sẽ cho phép xem các ngày trong tuần và tìm duyệt qua các tháng, như thể đang lật các trang của một “quyển lịch”.



Hình 2.27: Điều khiển DateTimePicker

Ngoài việc chọn ngày từ drop-down calendar, điều khiển còn cho phép người dùng nhập ngày vào hộp TextBox của nó. Với cả hai cách nhập ngày, dữ liệu vào luôn luôn được kiểm tra tính hợp lệ.

Thuộc tính CustomFormat của lớp DateTimePicker được dùng để chỉ định những định dạng nhà phát triển tự định nghĩa cho việc hiển thị ngày tháng năm và thời gian. Các chuỗi định dạng ngày tháng năm và thời gian rất đa dạng cho phép định dạng ngày tháng năm và thời gian theo các mục đích khác nhau.

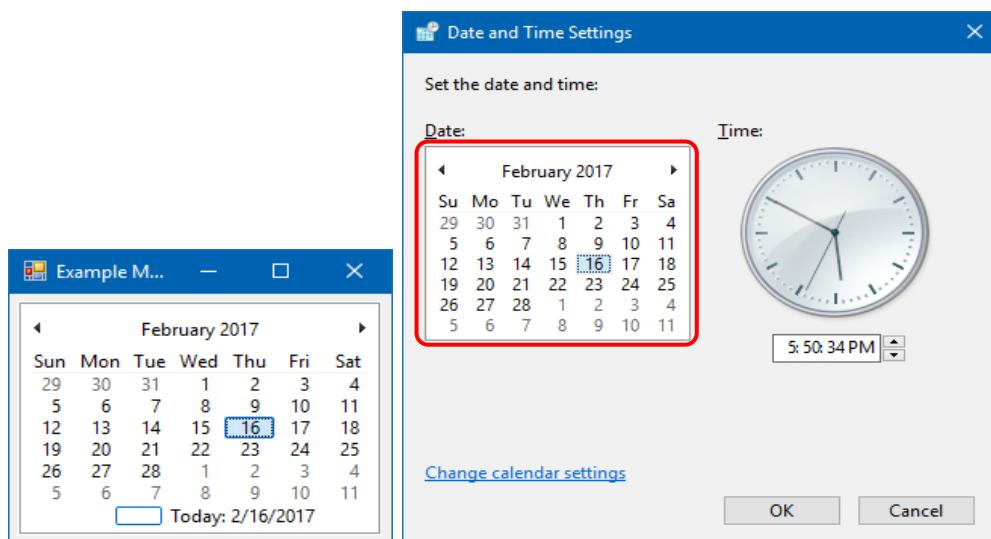
Bảng 2.12: Các chuỗi định dạng cho thuộc tính CustomFormat

Chuỗi định dạng	Mô tả
d	Hiển thị ngày một hoặc hai chữ số
h	Hiển thị giờ một hoặc hai chữ số ở định dạng 12 giờ
H	Hiển thị giờ một hoặc hai chữ số ở định dạng 24 giờ
m	Hiển thị phút một hoặc hai chữ số
M	Hiển thị tháng một hoặc hai chữ số
s	Hiển thị giây một hoặc hai chữ số
y	Hiển thị năm một chữ số

Lưu ý: cần phải thiết lập thuộc tính Format của điều khiển DateTimePicker thành DateTimePickerFormat.Custom để thuộc tính CustomFormat có thể tác động đến định dạng hiển thị của ngày tháng năm và thời gian. Thuộc tính Format gồm các giá trị sau: Custom, Long, Short, Time.

b/- MonthCalendar

Điều khiển MonthCalendar cho phép xem và chọn ngày với sự hỗ trợ của một graphical calendar. Calendar xuất hiện như một lưới của các hàng và cột có thể thả xuống (drop-down), lưới này sẽ hiển thị các cái ngày trong tháng được chỉ định. MonthCalendar tương tự như DateTimePicker, nhưng sẽ cho phép chọn nhiều ngày cùng lúc. Không giống DateTimePicker, MonthCalendar sẽ không thể hiển thị thời gian và không cho phép nhập ngày tháng năm.



Hình 2.28: Hình ảnh của MonthCalendar trong ứng dụng Windows Forms

Lưu ý: Chúng ta có thể thay đổi dáng vẻ bề ngoài của điều khiển MonthCalendar bằng cách điều chỉnh các thuộc tính sau: TitleBackColor, TitleForeColor, TrailingForeColor và ShowTodayCircle. Tuy nhiên, vẻ bề ngoài này vẫn sẽ bị chi phối bởi chủ đề (theme) đang áp dụng của hệ điều hành.

Điều khiển MonthCalendar mặc định chỉ hiển thị một tháng. Tuy nhiên chúng ta có thể chỉ định số tháng sẽ được hiển thị trong một lần và cách sắp xếp chúng trong điều khiển. Để hiển thị nhiều tháng, chúng ta có thể sử dụng thuộc tính CalendarDimensions và lưu ý cần phải điều chỉnh kích thước Form cho phù hợp khi thay đổi các chiều của calendar cũng như thay đổi kích thước của điều khiển.

c/- Thành phần Timer

Timer là một thành phần tạo ra một sự kiện một cách đều đặn. Nó làm việc tương tự như đồng hồ bấm giờ (stopwatch). Ví dụ, chúng ta có thể sử dụng thành phần Timer để thiết lập thời gian người dùng phải hoàn thành câu trả lời cho các câu hỏi đã yêu cầu. Khi vượt quá thời gian đã chỉ định, chúng ta có thể thực hiện các tác vụ nhất định như hỏi

người dùng ngưng việc trả lời câu hỏi. Có rất nhiều cách sử dụng thành phần Timer như thiết lập khoảng thời gian giữa hai sự kiện, giám sát thời gian bỏ ra để hoàn thành một tác vụ cụ thể, hủy một hành động cụ thể sau thời gian đã cho.

Thành phần Timer có hai phương thức khởi tạo có cú pháp như sau:

```
Timer <tên đối tượng> = new Timer();
```

```
Timer <tên đối tượng> = new Timer(Icontainer container);
```

Trong đó: Icontainer: chỉ định nơi (container) chứa thành phần Timer. Ví dụ, nếu kéo thả thành phần Timer vào Form thì Form chính là container của điều khiển này. Khi đó sẽ tương ứng với câu lệnh như sau:

```
Timer tmrCounter = new Timer(this);
```

Timer class được dùng để tạo ra thành phần Timer. Timer class được định nghĩa namespace System.Windows.Forms với nhiều **properties**, **methods** và **events** dùng để thiết lập khoảng thời gian và hành vi của điều khiển.

Bảng 2.13: Một số properties, methods và events của thành phần Timer

Thành phần	Tên	Mô tả
Properties	Enable	Chỉ định hoặc nhận giá trị chỉ ra rằng liệu Timer có đang chạy hay không.
	Interval	Chỉ định hoặc nhận thời gian được tính bằng mili giây (milliseconds) giữa các lần tick của Timer.
Methods	Start	Bắt đầu Timer.
	Stop	Dừng Timer.
Events	Tick	Điễn ra khi một khoảng thời gian cụ thể trôi qua và thuộc tính Enable của thành phần có giá trị là True.

❖ Hạn chế của thuộc tính Interval

Thuộc tính Interval của thành phần Timer được dùng để thiết lập số mili giây để bật sự kiện Tick của điều khiển lên. Tuy nhiên, thuộc tính này có một vài hạn chế như sau:

- Nếu một ứng dụng tạo các yêu cầu quá nặng đến hệ thống như là lặp trong thời gian dài hay các tính toán chuyên sâu, ứng dụng không thể nhận các sự kiện Timer một cách tuần tự như trong chỉ định của thuộc tính Interval.
- Thuộc tính Interval có giá trị lớn nhất là 64.767 mili giây (khoảng 64.8 giây).
- Khoảng thời gian (interval) có thể trôi đi không chính xác. Vì vậy, để đảm bảo độ chính xác, Timer nên kiểm tra đồng hồ hệ thống.

- Hệ thống chỉ có thể tạo ra 18 clock ticks cho mỗi giây. Thậm chí nếu như giá trị của thuộc tính `Interval` là mili giây, thì độ chính xác của thành phần cũng sẽ chỉ xấp xỉ 1/18 giây.

2.3 DIALOGBOX

Các ứng dụng điện hình có một cửa sổ chính, cửa sổ này được sử dụng hiển thị dữ liệu trong quá trình ứng dụng vận hành đồng thời cũng phơi bày các chức năng để xử lý dữ liệu thông qua cơ chế giao diện người dùng (UI) như thanh menu, thanh công cụ và thanh trạng thái. Tuy nhiên một ứng dụng không chỉ tầm thường như thế, nó cũng có thể hiển thị thêm các cửa sổ khác để hiển thị thông tin đặc biệt đến người dùng, thu nhận thông tin từ người dùng, hay cả hai công việc hiển thị và thu nhận thông tin. Các kiểu cửa sổ này được biết đến như các hộp hội thoại (dialog boxes).

2.3.1 Các thành phần và đặc trưng của DialogBox

a/- Thành phần của một DialogBox

Có thể sử dụng DialogBox để nhận dữ liệu vào hoặc hiển thị một vài thông tin quan trọng đến người dùng khi họ tương tác với ứng dụng. Một DialogBox bao gồm:



Hình 2.29: Các thành phần của một DialogBox

- (1) Thanh tiêu đề (Title bar) hiển thị nhãn của DialogBox và biểu tượng close.
- (2) Văn bản chỉ dẫn (Instruction text) thông báo cho người dùng nên làm gì (tùy chọn).
- (3) Nội dung (Content) với các điều khiển để chọn hoặc hiển thị thông tin.
- (4) Vùng hoạt động (Action area) có chứa các nút (Buttons) và các LinkLabel
- (5) Vùng chú thích (Footnote area) giải thích về cửa sổ hay dialog box (tùy chọn).

b/- Đặc trưng của DialogBox

Một DialogBox là một cửa sổ đặc biệt, luôn được triệu gọi và truy xuất thông qua các Form. Nó được gọi là DialogBox vì là tầng trung gian nơi diễn ra cuộc đối thoại giữa người dùng và ứng dụng.

DialogBox khác với Form thông thường do các tính đặc trưng cơ bản sau:

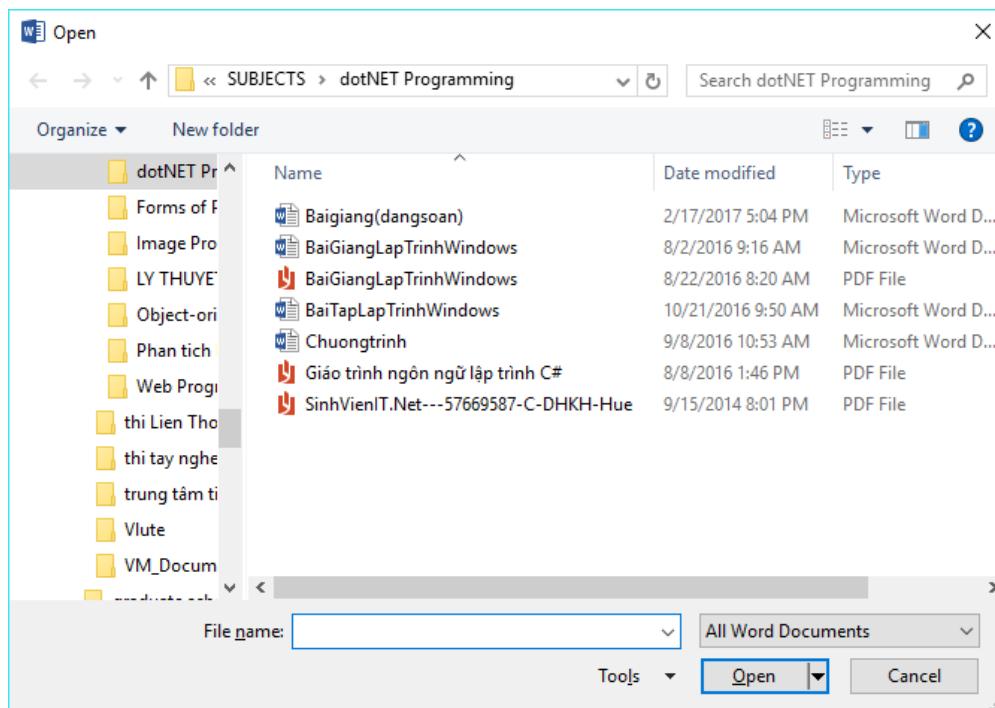
- Không thể thay đổi kích thước cũng như không thể phóng to (maximized) hay thu nhỏ (minimize) nó.
- Thường sẽ có mô tiệp (modal).
- Sử dụng tập các nút (buttons) và các biểu tượng (icons) chung. Nhìn chung, một dialog box sẽ có một vài nút chuẩn như OK và Cancel, nó cũng có thể có các biểu tượng Help và Close ở góc trên bên phải.

c/- Phân loại DialogBox

DialogBox được chia thành hai loại đó là modal và modeless.

❖ Modal DialogBox

Một DialogBox ở dạng modal cho phép thao tác với chức năng cần thêm dữ liệu từ người dùng để có thể tiếp tục. Bởi vì chức năng phụ thuộc vào DialogBox để thu nhận dữ liệu. Modal DialogBox cũng ngăn chặn người dùng kích hoạt các cửa sổ khác trong ứng dụng khi nó còn đang mở. Trong hầu hết các trường hợp, một modal DialogBox cho phép người dùng báo hiệu khi họ hoàn thành thao tác trên DialogBox bằng cách nhấn vào nút OK hoặc Cancel. Nhấn nút OK để chỉ rằng người dùng đã nhập dữ liệu và muốn chức năng được tiếp tục xử lý với dữ liệu. Nhấn nút Cancel có nghĩa rằng người dùng muốn dừng chức năng và quay lại cửa sổ trước đó. Ví dụ phổ biến nhất cho loại DialogBox này đó là hiển thị để mở (open) tập tin mới, lưu (save) hoặc in (print) dữ liệu.

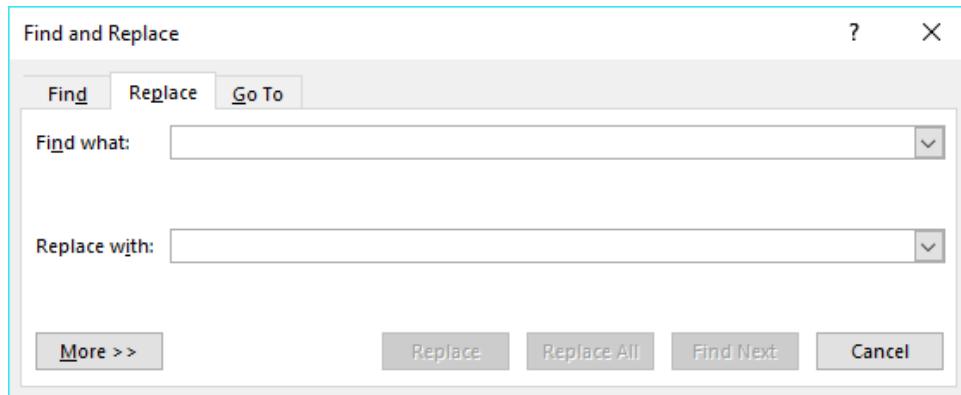


Hình 2.30: Open File Dialog, ví dụ điển hình của dialog boxes kiểu modal

❖ Modeless DialogBox

Trái ngược với dạng modal, một modeless DialogBox không ngăn chặn việc thao tác với các cửa sổ khác khi nó đang mở. Ví dụ, nếu một người dùng muốn tìm sự hiện diện của một từ cụ thể trong một tài liệu, thường cửa sổ chính sẽ mở một DialogBox để hỏi

người dùng từ gì họ muốn tìm kiếm. Khi đó, việc tìm kiếm một từ sẽ không ngăn người dùng điều chỉnh tài liệu vì vậy DialogBox không cần ở dạng modal. Một modeless DialogBox phải cung cấp ít nhất một nút Close để đóng nó lại, và có thể cung cấp thêm các nút để thực thi các chức năng đặc biệt như nút Find Next để tìm các từ tiếp theo khớp với tiêu chí tìm kiếm từ.



Hình 2.31: Find and Replace DialogBox, ví dụ điển hình của modeless DialogBox
DialogBox cũng được phân thành 2 loại Common DialogBox và Custom DialogBox

❖ Common DialogBox (System-defined dialog boxes)

Windows cung cấp rất đa dạng các DialogBox có thể sử dụng chung cho tất cả các ứng dụng, bao gồm các DialogBox cho việc mở tập tin, lưu tập tin và in ấn. Khi đó các DialogBox này được thực thi bởi hệ điều hành và chúng có thể được chia sẻ giữa tất cả các ứng dụng chạy trên hệ điều hành, từ đó tạo tính nhất quán trong trải nghiệm của người dùng. Khi người dùng quen với việc sử dụng các DialogBox do hệ thống cung cấp họ không cần phải học lại việc sử dụng DialogBox trên các ứng dụng khác. Bởi vì các DialogBox này có giá trị trong tất cả các ứng dụng, và bởi vì chúng giúp cung cấp trải nghiệm nhất quán của người dùng nên chúng được biết đến với tên Common DialogBox.

❖ Custom DialogBox (User-defined dialog boxes)

Common DialogBox được sử dụng rộng rãi trong các trường hợp thông dụng, nhưng chúng lại không hỗ trợ các yêu cầu cụ thể của nhà phát triển và người dùng. Đối với các trường hợp này, chúng ta cần tạo ra các DialogBox hỗ trợ các yêu cầu cụ thể được đưa ra. Để làm được điều đó, chúng ta có thể chuyển một Form thành một user-defined DialogBox.

User-defined dialog box được tạo như sau:

- Thiết lập giá trị `FixedDialog` cho thuộc tính `FormBorderStyle` của Form. Thuộc tính này sẽ thay đổi đường viền của Form từ dạng đường viền của Form sang dạng đường viền chuẩn của DialogBox.
- Các thuộc tính `MinimizeBox` và `MaximizeBox` nên được thiết lập là `False`. Việc thiết lập này sẽ làm cho cửa sổ chỉ hiển thị nút Close do hệ thống định nghĩa.
- Form phải có các nút `OK` và `Cancel`. Khi nhấp vào nút `OK` sẽ cho phép người dùng đóng DialogBox, trở lại chức năng chính và tiếp tục xử lý kết quả trả về.

Khi nút Cancel được nhấp, DialogBox sẽ đóng và chức năng cần xử lý trong DialogBox sẽ bị dừng cho những bước thực hiện tiếp theo.

- Thuộc tính AcceptButton của Form nên được thiết lập là nút OK.
- Thuộc tính CancelButton của Form nên được thiết lập là nút Cancel.

Lưu ý: để hiển thị một Form dưới dạng DialogBox, ta sử dụng phương thức ShowDialog() thay cho phương thức Show().

2.3.2 Nhận thông tin từ DialogBox

Một Form gọi DialogBox có thể nhận thông tin từ DialogBox khi nó đóng bằng cách tham chiếu đến thuộc tính DialogResult. Chúng ta cũng có thể nhận kết quả này bằng cách tham chiếu đến giá trị trả về của việc gọi đến phương thức ShowDialog().

Form hiển thị DialogBox sau đó sẽ phản ứng theo giá trị trả về này. Ví dụ, một MessageBox trong các ứng dụng Windows Forms là một modal DialogBox, nó có các nút OK và Cancel. Form hiển thị MessageBox có thể theo vết nút mà người dùng đã nhấp và có thể thực hiện các tác vụ được yêu cầu.

❖ Kiểu dữ liệu liệt kê DialogResult (“DialogResult” enumeration)

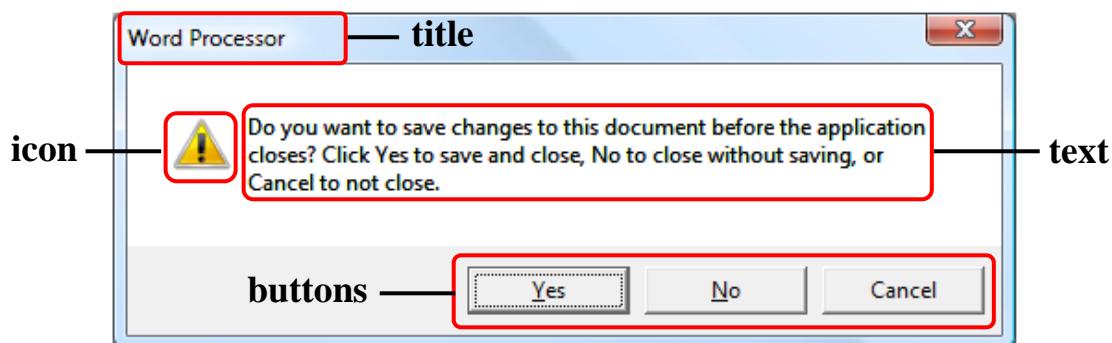
Bảng 2.14: DialogResult enum

Chỉ định	Mô tả
Abort	Giá trị trả về của dialog box là Abort
Cancel	Giá trị trả về của dialog box là Cancel
Ignore	Giá trị trả về của dialog box là Ignore
No	Giá trị trả về của dialog box là No
None	Giá trị trả về của dialog box là None. Điều này có nghĩa là modal dialog box đang chạy.
OK	Giá trị trả về của dialog box là OK
Retry	Giá trị trả về của dialog box là Retry
Yes	Giá trị trả về của dialog box là Yes

Thuộc tính DialogResult được dùng để nhận thông tin từ DialogBox. Chúng ta có thể chỉ định giá trị cho thuộc tính DialogResult bằng cách sử dụng DialogResult enum. Enum này định nghĩa đa dạng các chỉ định để nhận giá trị từ DialogBox. Bảng sau đây sẽ biểu diễn các chỉ định được định nghĩa trong DialogResult enum.

2.3.3 MessageBox

Dùng để hiển thị thông tin và cho phép người dùng đưa ra quyết định với các nút (button). Trong hình sau là một MessageBox hiển thị thông tin một câu hỏi và cung cấp cho người dùng ba nút để trả lời câu hỏi.



Hình 2.32: Minh họa message box

Để tạo MessageBox, chúng ta sử dụng `MessageBox` class. `MessageBox` sẽ cho phép cấu hình văn bản (text), tiêu đề (title), biểu tượng (icon) và các nút (buttons) của nó.

Ví dụ 2.14:

```
// Cấu hình message box cần hiển thị
string messageBoxText = "Do you want to save changes?";
string caption = "Word Processor";
MessageBoxButtons button = MessageBoxButtons.YesNoCancel;
MessageBoxIcon icon = MessageBoxIcon.Warning;
// Hiển thị message box
MessageBox.Show(messageBoxText, caption, button, icon);
```

Khi hiển thị MessageBox, chúng ta cần viết mã lệnh để phát hiện và xử lý quyết định của người dùng (khi các nút được nhấp).

Ví dụ 2.15:

```
// Hiển thị message box
DialogResult result = MessageBox.Show(messageBoxText, caption,
button, icon);
// Xử lý các kết quả của messagebox
switch (result)
{
    case DialogResult.Yes:
        // xử lý khi người dùng nhấp nút Yes
        // ...
        break;
    case DialogResult.No:
        // xử lý khi người dùng nhấp nút No
        // ...
        break;
    case DialogResult.Cancel:
        // xử lý khi người dùng nhấp nút Cancel
        // ...
        break;
}
```

2.4 PHÁT TRIỂN ÚNG DỤNG MDI (MULTIPLE-DOCUMENT INTERFACE)

2.4.1 Úng dụng MDI là gì?

Một ứng dụng có chứa nhiều Form được dùng cho việc thu thập và hiển thị các thông tin khác nhau. Cách thức hiển thị và sử dụng những Form này được cung cấp bởi việc thiết kế document interface. Document interface là cách bố trí của các cửa sổ ứng dụng, nó được dùng cho việc mở và tổ chức các Form trong một cách thức mong muốn và phù hợp.

Có hai loại document interfaces cơ bản đó là Single Document Interface (SDI) và Multiple Document Interface (MDI).

a/- *Multiple Document Interfaces*

Xem xét kịch bản người dùng đang làm việc với một ứng dụng. Khi thao tác, người dùng có thể mở nhiều tài liệu cùng lúc trong cùng một thể hiện của ứng dụng, nhiều cửa sổ con sẽ được hiển thị trong vùng làm việc của nó. Khi đó, người dùng có thể làm việc với bất kỳ cửa sổ con nào bằng cách lựa chọn giữa chúng với nhau. Một ứng dụng MDI cho phép người dùng có thể mở nhiều tài liệu cùng lúc.

b/- *Thuận lợi của ứng dụng MDI*

Việc sử dụng ứng dụng MDI có nhiều thuận lợi như:

- *Tổ chức có tính hệ thống*: một ứng dụng MDI giúp sắp xếp tất cả các Form có liên quan theo một phương thức có tổ chức.

- *Tăng hiệu quả*: ứng dụng MDI cho phép dễ dàng truy xuất nhiều Form thông qua một thanh công cụ đơn. Từ đó, làm tăng tốc độ và hiệu quả của việc truy xuất nhiều Form

- *Sử dụng ít bộ nhớ*: ứng dụng MDI có khả năng mở nhiều Form trong một cửa sổ đơn. Điều này giúp giảm việc sử dụng bộ nhớ, gần như chống lại SDI khi mà nó cần nhiều bộ nhớ hơn để mở nhiều thể hiện để làm việc với nhiều tập tin.

- *Thực hiện dễ dàng*: ứng dụng MDI tạo điều kiện điều chỉnh dễ dàng và cùng lúc nhiều Form.

c/- *Hạn chế của ứng dụng MDI*

Một ứng dụng MDI phải có ít nhất một Form con. Trong trường hợp nhiều Form con, nó sẽ trở nên khó khăn cho người dùng khi làm việc với nhiều Form và chọn lựa giữa các Form để có các thay đổi mong muốn. Điều này làm tăng mối nguy hại của dữ liệu không đồng nhất. Người dùng cũng có thể bị lẫn lộn và mất dấu vết của Form đã truy xuất để tạo ra các thay đổi.

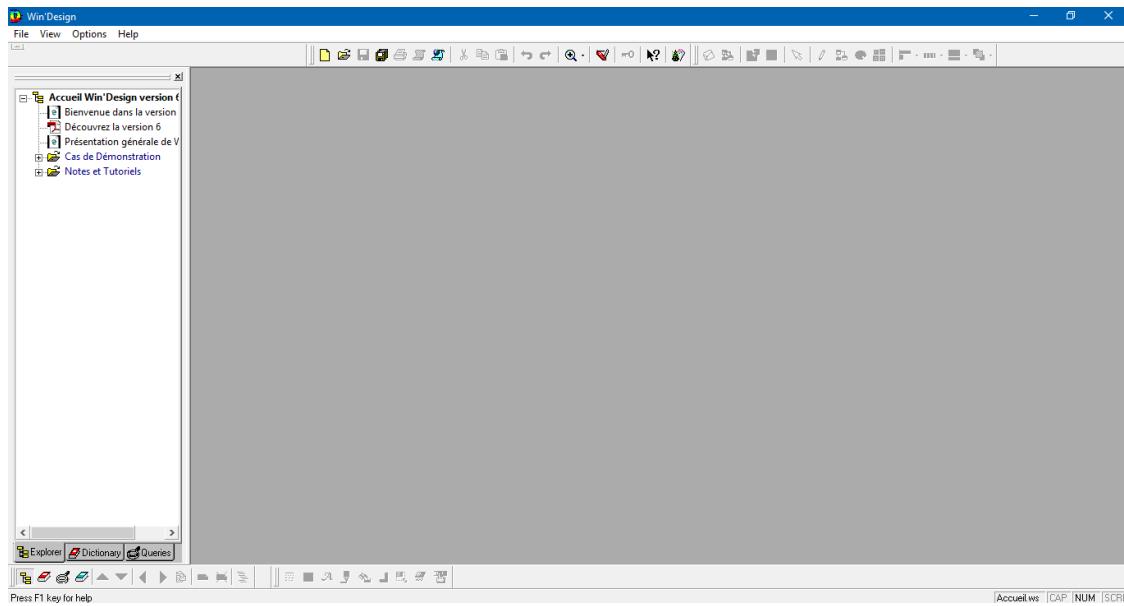
2.4.2 Các thành phần của ứng dụng MDI

Một ứng dụng MDI bao gồm một Form cha (parent form) và nhiều Form con (child forms).

a/- *MDI Parent Form*

Trong ứng dụng MDI, Form cha (MDI Parent Form) hoạt động như một cửa sổ nền, thông qua đó người dùng có thể tương tác với ứng dụng MDI. Tất cả các Form con được

mở trong Form này. Vì vậy, nó hoạt động như một vùng chứa đựng, giữ tất cả các Form con. Mỗi ứng dụng MDI chỉ có một Form cha.



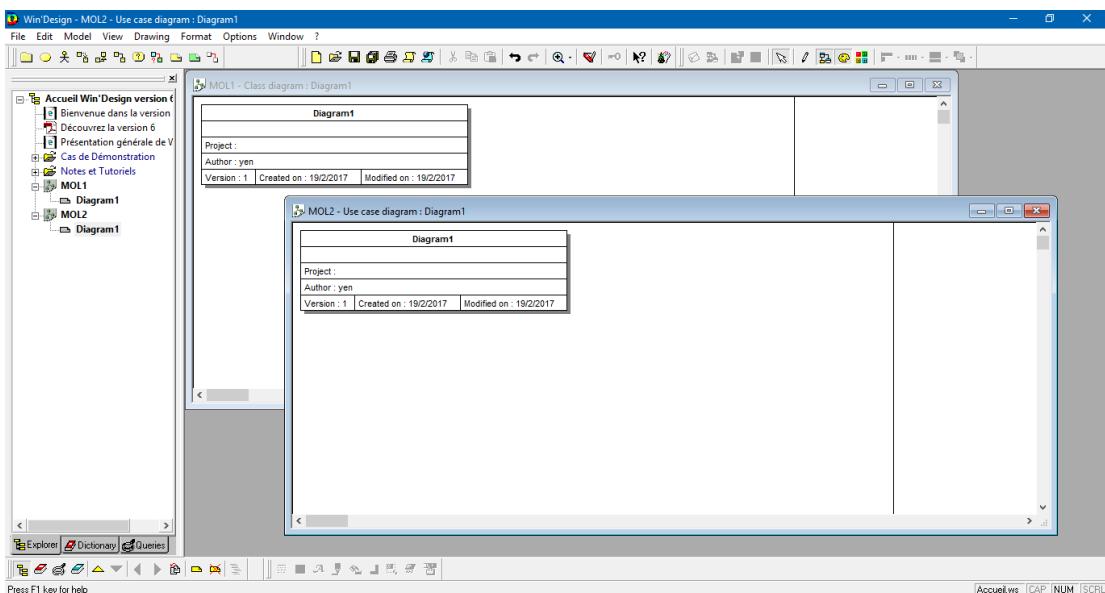
Hình 2.33: Hình ảnh của MDI Parent Form trong ứng dụng Win'Design

❖ Tạo MDI Parent Form khi thiết kế

- Bước 1: Tạo dự án Windows Forms Application
- Bước 2: Trong cửa sổ Properties của Form, thiết lập giá trị true thuộc tính IsMDIContainer. Việc này sẽ chỉ định Form là một MDI container cho các Form con.
- Bước 3: Chạy ứng dụng từ MDI Parent Form.

b/- MDI Child Forms

MDI Child Forms là những Form con được mở trong Form cha. Người dùng làm việc thực sự với những Form con này, khi đó Form cha đóng vai trò trung gian giúp người dùng truy xuất các Form con. Khi Form cha đóng thì tất cả Form con liên kết với nó cũng sẽ đóng. Tuy nhiên, nếu tắt cả các Form con đóng thì Form cha cũng sẽ không tự động đóng.



Hình 2.34: Hình ảnh của MDI Child Forms trong ứng dụng Win'Design

❖ Tạo MDI Child Forms

Ví dụ sau đây sẽ minh họa cách tạo một MDI Child Forms

Ví dụ 2.16:

- Tao một Project với tên Example_MDI
- Trong cửa sổ Properties của Form mặc định, thiết lập giá trị cho các thuộc tính như sau:

- + Name: frmParent
- + Text: MDI Parent Form
- + IsMdiContainer: true
- + WindowState: maximized

- Trong cửa sổ Solution Explorer, nhấp chuột phải lên Project chọn Add/Windows Form...

- Tạo một Form mới với tên frmChild

- Trong cửa sổ Properties của frmChild, thiết lập giá trị cho các thuộc tính như sau:

- + Text: MDI Child Form
- + WindowState: maximized

- Xử lý sự kiện load của frmParent với đoạn mã lệnh sau:

```
frmChild frm = new frmChild();
frm.MdiParent = this;
frm.Show();
```

- Chạy ứng dụng và kiểm tra kết quả.

2.4.3 Menu

a/- Hệ thống menu trong Windows Forms

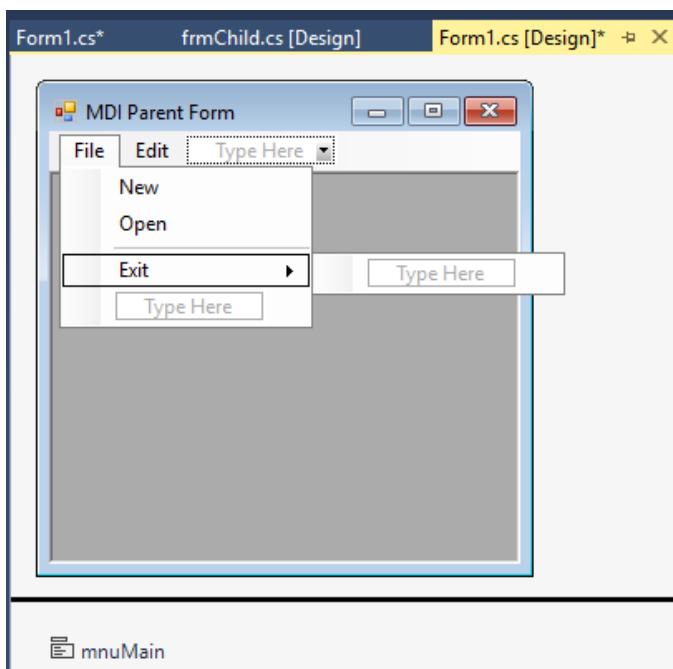
Xem xét kịch bản nhà phát triển muốn tạo ra một ứng dụng Chi tiết nhân viên. Ứng dụng phải cho phép người dùng truy xuất nhiều Form. Nó phải cho phép người dùng truy xuất thông tin các nhân, chi tiết lương, chi tiết giấy phép,... Ứng dụng cũng phải cho phép người dùng thoát khỏi ứng dụng. Các yêu cầu này có thể đạt được bằng cách tạo ra các trình đơn (menus) cho phép người dùng chọn lựa giữa các Form chức năng.

Hệ thống menu trong Windows Forms là một bảng điều khiển menu chính (main menu panel). Bảng điều khiển menu chính bao gồm các thanh menu và nhiều menu con.

b/- Các cấu trúc menu

Có hai loại menu trong Windows Form đó là: main menu và context menu.

- Main menu còn được biết đến là anchored menu, xuất hiện trong thanh menu của Form. Main menu gồm một tập các mục hiển thị theo chiều ngang dọc theo trên đỉnh của hầu hết các ứng dụng.
- Context menu, còn được gọi là popup menu, là các menu xuất hiện khi người dùng nhấp phải chuột trên một điều khiển hoặc trên Form. Context menu là các menu tắt chứa các lệnh được thao tác thường xuyên.

c/- MenuStrip và ToolStripMenuItem

Hình 2.35: Thêm MenuStrip vào Form

Điều khiển MenuStrip cho phép thêm các menu mới, điều chỉnh và sắp xếp lại các menu đã tồn tại, hoặc xóa các menu đã có. Chúng ta cũng có thể sử dụng các access keys, check marks, images và thanh phân cách (separator bar) để cải tiến tính khả dụng của menu.

MenuStrip hỗ trợ MDI, gộp menu, và tool tips.

Để thêm MenuStrip cho Form thực hiện kéo thả biểu tượng MenuStrip từ Toolbox vào Form. Sau đó lần lượt thêm các mục tương ứng cho các sub-menu (ToolStripMenuItem).

ToolStripMenuItem class cung cấp nhiều thuộc tính cho phép nhà phát triển có thể điều chỉnh dáng vẻ bề ngoài cũng như chức năng của các menu item. Class này cũng là thành phần của namespace System.Windows.Forms.

d/- ContextMenuStrip

Điều khiển ContextMenuStrip được dùng để cung cấp đường tắt cho việc truy xuất các tùy chọn menu hiển thị trên thanh menu. Nó tương tự như context menu xuất hiện trong ứng dụng Microsoft Word khi người dùng nhấp chuột phải trên tài liệu. Nó cho phép nhà phát triển gom nhóm hợp lý các tùy chọn có liên quan từ menu, từ đó cho phép người dùng thực hiện nhanh các tác vụ.

Điều khiển này hữu dụng khi chúng ta muốn thực hiện các tác vụ nhất định một cách thường xuyên.

e/- ToolStrip (tạo tool bar)

Xem xét ngữ cảnh chúng ta muốn định dạng nội dung của một tài liệu. Ứng dụng cung cấp tất cả các tùy chọn định dạng trên một menu và trong thanh công cụ (tool bar). Tuy nhiên, chúng ta có thể tìm thấy nó một cách nhanh chóng và thuận tiện hơn bằng cách

dùng các biểu tượng trên toolbar thay vì phải sử dụng menu. Điều khiển ToolStrip là điều khiển phục vụ cho ý định toolbar đã đề cập, nó hoạt động như một vật chứa đựng để nhóm các menu item và các điều khiển khác do người dùng định nghĩa.

Điều khiển ToolStrip có các đặc trưng sau:

- Cung cấp giao diện dùng chung giữa các Form
- Cho phép kéo thả item từ một điều khiển ToolStrip này đến một điều khiển ToolStrip khác.

- Hỗ trợ trật tự các item theo thời gian thực.

- Chịu sự chi phối của hệ điều hành về ngoài cũng như hành vi của nó.

f/- StatusStrip (tạo status bar)

Xem xét ngữ cảnh khi chúng ta làm việc với tài liệu Microsoft Word có 150 trang. Chúng ta có thể nhận định số trang và số từ bằng cách nhìn vào status bar nằm ở góc dưới của cửa sổ.

Điều khiển StatusStrip được dùng để hiển thị các thông tin hữu ích về các tác vụ thực hiện hoặc các điều khiển đang dùng. Nó hiển thị ở đáy của Form. Nó cũng có thể được dùng để hiển thị progress bar.

2.5 TÍCH HỢP TRỢ GIÚP NGƯỜI DÙNG

2.5.1 Vai trò của trợ giúp người dùng (User help) trong các ứng dụng Windows Forms

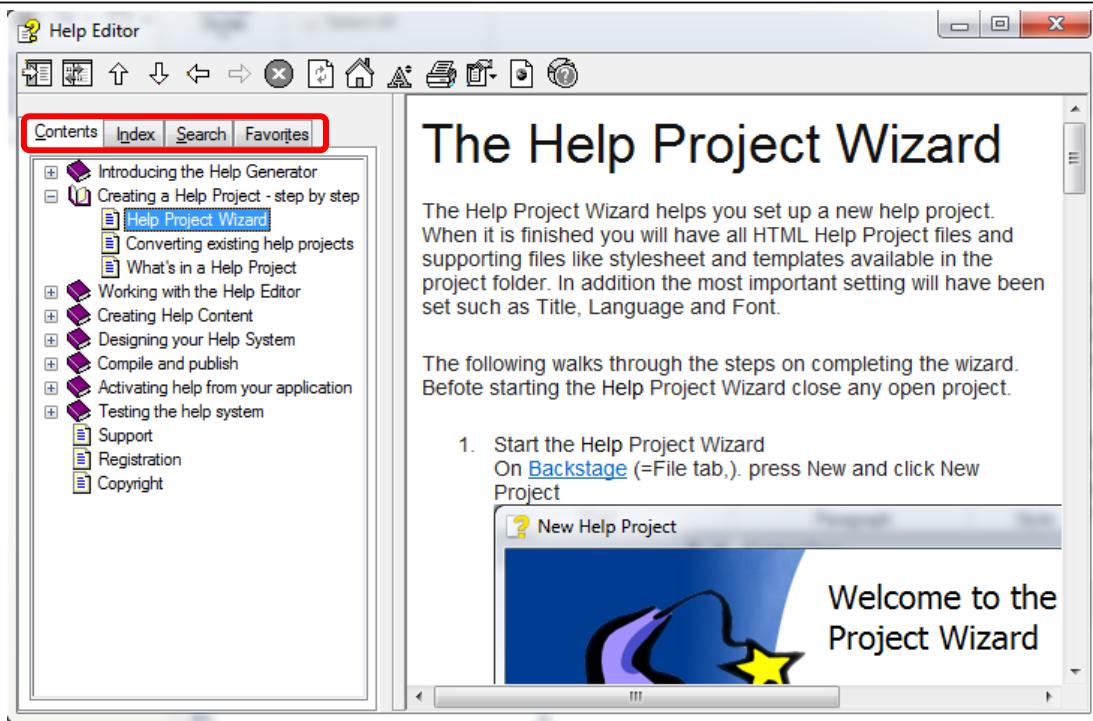
a/- Tập tin trợ giúp (help file) trong các ứng dụng Windows Forms

Xem xét kịch bản một công ty có sử dụng ứng dụng quản lý thông tin nhân viên cho phép nhân viên của họ tự cập nhật thông tin cá nhân của mình. Khi có một nhân viên mới đến công ty làm việc họ sẽ được cấp tài khoản để thao tác với ứng dụng, tuy nhiên họ phải thực hiện các thao tác đó như thế nào. Và nhân viên mới này cũng muốn hiểu rõ hơn về cách thức làm việc của ứng dụng.

Để hỗ trợ cho chức năng này chúng ta có thể tạo ra các tập tin trợ giúp (help files). Khi một ứng dụng cung cấp tập tin trợ giúp cho người dùng, thì yêu cầu tập tin trợ giúp này phải đầy đủ các hướng dẫn và mô tả cách thức thực hiện các tác vụ cụ thể. Windows Forms cho phép tạo ra các tập tin tương tự như các tập tin trợ giúp của các ứng dụng Windows chuẩn như Internet Explorer hay Microsoft Word.

b/- Cấu trúc của tập tin trợ giúp

Một tập tin trợ giúp cho phép chúng ta xem các nội dung trợ giúp, tìm một tác vụ hoặc khái niệm cụ thể... Các chức năng này của tập tin trợ giúp được cung cấp bởi bốn thành phần khác nhau của nó, đó là: Content, Index, Search và Favorites.



Hình 2.36: Cấu trúc của file help

- Content: cửa sổ Content hiển thị cấu trúc thông tin cung cấp bởi tập tin trợ giúp. Các nội dung các chủ đề riêng biệt có thể được nhìn thấy bằng cách định hướng thông qua chủ đề và nhấp vào nó. Người dùng cũng có thể áp dụng thêm các tiêu chuẩn lọc để thấy các chủ đề cụ thể.

- Index: cửa sổ Index hiển thị danh sách các chủ đề được sắp xếp theo thứ tự alphabet. Nó tương tự như trang chỉ mục của một quyển sách và giúp cho việc tìm kiếm các từ khóa cũng như các chủ đề liên quan để từ khóa diễn ra dễ dàng, nhanh và hiệu quả hơn.

- Search: hộp thoại Search tìm các từ hoặc cụm từ khóa đặc biệt. Nó được dùng khi người dùng không biết nơi để tìm thông tin mình cần. Người dùng có thể sử dụng các tùy chọn khác nhau khi tìm kiếm. Một vài tùy chọn được hỗ trợ là tìm, lọc và so khớp từ.

- Favorites: cửa sổ Favorites ghi vào danh sách các đường dẫn đến các mục như các chủ đề hoặc trang Web trợ giúp. Người dùng cũng có thể tổ chức các mục vào các thư mục, đổi tên hoặc xóa chúng.

2.5.2 Tạo tập tin .chm

a/- HTML Help Workshop

Microsoft HTML Help Workshop (Hhw.exe) được sử dụng cho việc thiết kế một tập tin trợ giúp bằng cách tạo và tích hợp các HTML file như các file nguồn. HTML help workshop cho phép tạo ra một dự án độc lập cho một tập tin trợ giúp. Dự án này có chứa tất cả các tập tin khác nhau tạo nên hệ thống trợ giúp và được lưu lại như một tập tin .hhp. Nó cũng cho phép tạo và edit Table of Contents (.hhc) và Index (.hhk).

HTML Help Workshop bao gồm HTML Help compiler, cho phép người dùng xem các tập tin trợ giúp, được tạo ra khi biên dịch dự án, thông qua HTML Help Viewer.

b/- Các kiểu trợ giúp

Một tập tin trợ giúp được tạo ra bởi HTML Help Workshop có thể được hiển thị trong một ứng dụng Windows, và cũng có thể được triển khai như một hệ thống độc lập trên Website. HTML Help Workshop có thể tạo ra hai loại help:

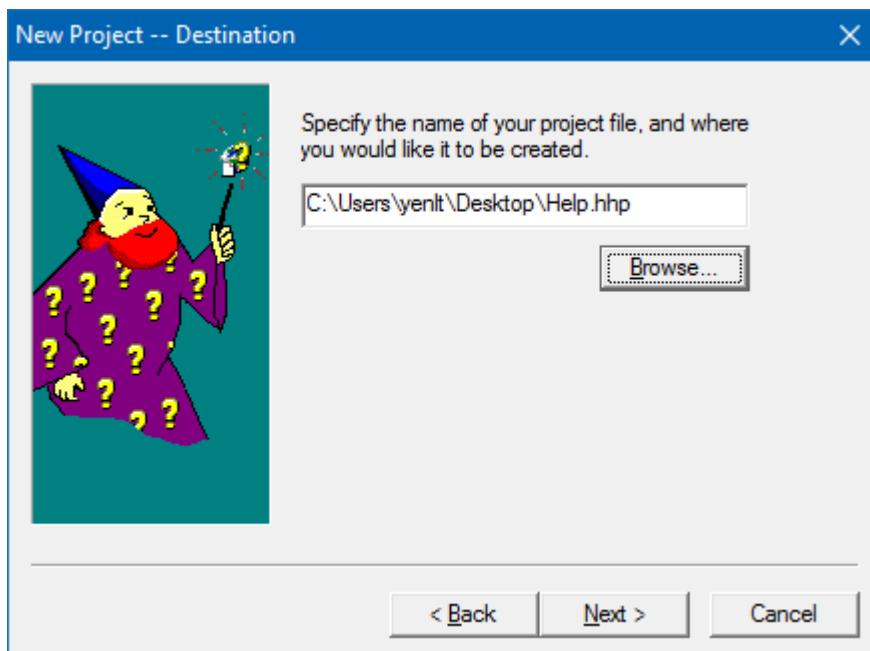
Window-Level Help: các chủ đề help được yêu cầu xuất hiện trong một cửa sổ mới. Nội dung của chủ đề help trong một file HTML sẽ được hiển thị trong cửa sổ. Cách thức này tương tự như khi ta nhấn phím F1 trong các ứng dụng chuẩn trên hệ điều hành Windows.

Context-Sensitive Help: chủ đề help được yêu cầu xuất hiện trong một cửa sổ pop-up. Nội dung của chủ đề help được lưu trong tập tin .txt sẽ được hiển thị trong cửa sổ pop-up.

c/- Các bước tạo tập tin .chm

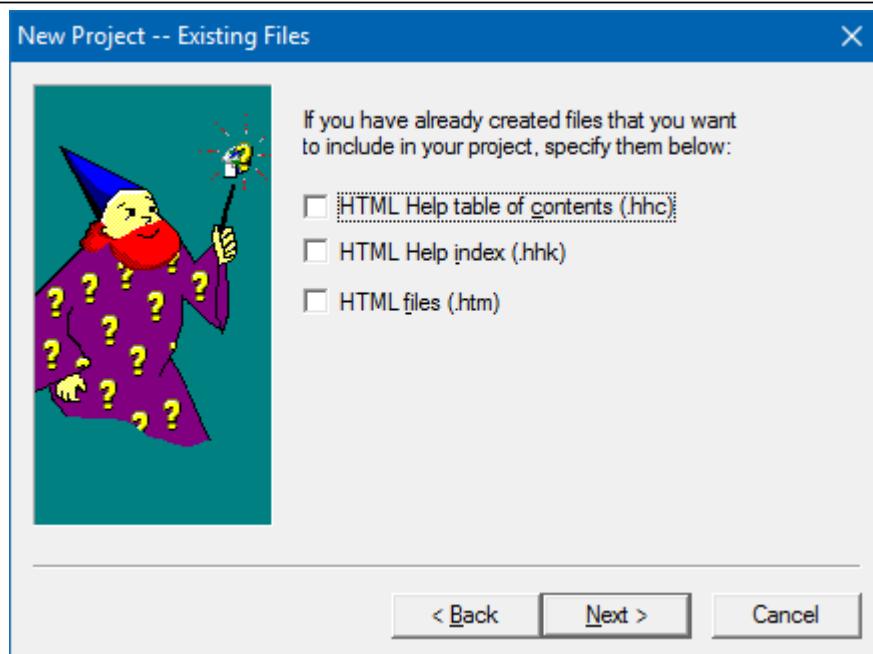
Mỗi chủ đề help được tạo tách biệt và cuối cùng chúng sẽ được tích hợp lại trong cùng một HTML Help Workshop. Sau đây sẽ là cách tạo một tập tin trợ giúp sử dụng HTML Help Workshop:

- Bước 1: Mở HTML Help Workshop
- Bước 2: Chọn mục New từ menu File. New dialog box xuất hiện.
- Bước 3: Chọn OK. New Project wizard xuất hiện
- Bước 4: Chọn Next để mở trang New Project – Destination. Tại đây, chúng ta nên chọn vị trí lưu và tên cho tập tin Project.



Hình 2.37: New Project -- Destination (HTML Help Workshop)

- Bước 5: Chọn Next để đến trang New Project – Existing Files để thêm những tập tin trợ giúp đã tạo trước đó.



Hình 2.38: New Project -- Existing Files

- Bước 6: Chọn Finish

Và bây giờ chúng ta có thể chọn tùy chọn HTML File trong New dialog box. Chúng ta cũng có thể sử dụng các tab Content và Index cùng với các tập tin này. Cuối cùng, project cần được biên dịch (compile) để tạo ra tập tin trợ giúp (sử dụng nút Compile trên toolbar).

2.5.3 Các thành phần và điều khiển trợ giúp người dùng

a/- *Help class*

Help class được dùng để truy xuất và hiển thị nội dung các tập tin trợ giúp HTM hoặc CHM, các tập tin này bao gồm các thành phần Content, Index và Search. Help class nằm trong namespace System.Windows.Forms. Chúng ta không thể tạo đối tượng của Help class. Trong Help class có định nghĩa các phương thức tĩnh như ShowHelp() và ShowHelpIndex() cho phép hiển thị tập tin trợ giúp.

Bảng 2.15: Các methods của Help class hỗ trợ hiển thị tập tin trợ giúp

Methods	Mô tả
ShowHelp	Hiển thị cửa sổ Contents của tập tin trợ giúp
ShowHelpIndex	Hiển thị cửa sổ Index của tập tin trợ giúp
ShowPopup	Hiển thị cửa sổ pop-up help

❖ Phương thức ShowHelp()

Phương thức ShowHelp() của Help class nhận vào tham số thứ 3 là HelpNavigator. Các giá trị của HelpNavigator enum trong Bảng 2.16 hỗ trợ thiết lập các thành phần của tập tin trợ giúp khi hiển thị.

Ví dụ 2.17:

```
private void btnHelp_Click(System.Object sender, EventArgs e)
{
    Help.ShowHelp(txtCharMap, "file:///c:\\charmap.chm",
    HelpNavigator.Index);
}
```

Bảng 2.16: Các giá trị của HelpNavigator

Giá trị	Mô tả
AssociateIndex	Help file mở ra các chỉ mục theo ký tự đầu tiên của chủ đề
Find	Help file mở trang tìm kiếm
Index	Help file mở cửa sổ index
KeywordIndex	Help file mở chủ đề với các chỉ mục chỉ định nếu có; nếu không, các chỉ mục gần với từ khóa chỉ định sẽ được hiển thị
TableOfContents	Help file mở ở cửa sổ Contents
Topic	Help file mở một chủ đề được chỉ định, nếu chủ đề tồn tại
TopicId	Help file mở một chủ đề dựa trên số định danh của nó

b/- HelpProvider

Thành phần HelpProvider được dùng để liên kết tập tin trợ giúp đến một ứng dụng Windows Forms. Nó cũng hỗ trợ cho context-sensitive help trên các điều khiển (controls) và các DialogBox. Thành phần này có khả năng mở các phần được chỉ định của help file như **Table of Contents**, **Index** và **Search**.

HelpProvider vận hành phối hợp với thuộc tính của các điều khiển trên Form. Để hiển thị tập tin trợ giúp, người dùng phải chọn (focus on) điều khiển muốn yêu cầu trợ giúp và nhấn phím **F1** để hiển thị tập tin trợ giúp.

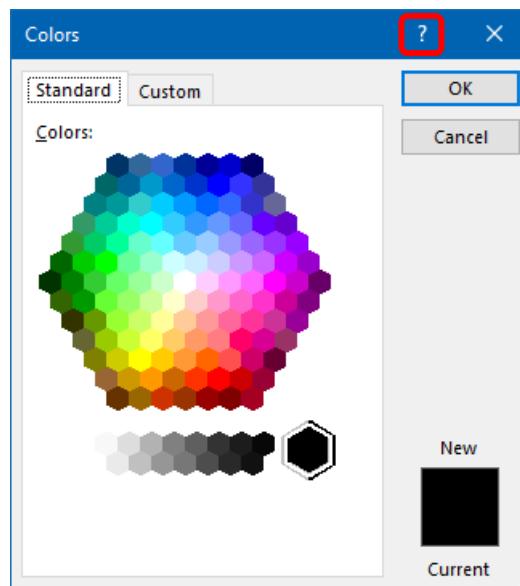
HelpProvider class được sử dụng để hiển thị trợ giúp dưới dạng pop-up hoặc trực tuyến. HelpProvider class nằm trong namespace `System.Windows.Forms`, định nghĩa đa dạng các thuộc tính và phương thức được dùng để hiển thị file help.

Bảng 2.17: Một số properties và methods của HelpProvider

Thành phần	Tên	Mô tả
Properties	HelpNamespace	Nhận hoặc thiết lập giá trị tên của help file được liên kết với đối tượng HelpProvider
	Tag	Nhận hoặc thiết lập giá trị của đối tượng chứa dữ liệu bổ sung về HelpProvider

Methods	GetHelpKeyword	Trả về giá trị từ khóa trợ giúp cho các điều khiển
	GetHelpNavigator	Trả về HelpNavigator hiện thời thiết lập cho điều khiển
	GetShowHelp	Trả về giá trị liệu có nên hiển thị trợ giúp của điều khiển
	SetHelpKeyword	Chỉ định từ khóa được dùng để nhận về sự trợ giúp khi người dùng gọi trợ giúp cho điều khiển
	SetHelpNavigator	Chỉ định lệnh trợ giúp để sử dụng khi nhận sự trợ giúp từ help file cho điều khiển.
	SetShowHelp	Chỉ định sự trợ giúp sẽ được hiển thị cho điều khiển hay không
	ToString	API này hỗ trợ cơ sở hạ tầng sản phẩm và không được sử dụng trực tiếp trong mã nguồn. Trả về một chuỗi biểu diễn HelpProvider hiện thời (Ghi đè Component.ToString())

c/- HelpButton



Hình 2.39: Hình ảnh của HelpButton trong Form

HelpButton là một thuộc tính của Form, mặc định thuộc tính này có giá trị `false`. Khi thuộc tính `HelpButton` được thiết lập giá trị `true`, Form sẽ hiển thị nút Help với dấu chấm hỏi trên thanh tiêu đề ở bên trái của nút close.

Để hiển thị nút Help, thuộc tính `MaximizeBox` và `MinimizeBox` của Form phải được chỉ định là `false`. Người dùng có thể hiển thị tập tin trợ giúp bằng cách click vào nút Help hoặc chọn một điều khiển đặc biệt trên Form. Khi người dùng chọn một điều

khiển, sau đó nhấp vào nút Help, sự kiện HelpRequested của Form sẽ được gọi. Chúng ta có thể viết mã lệnh trong bộ xử lý sự kiện tương ứng để hiển thị tập tin trợ giúp.

d/- Pop-up Help

Pop-up help hiển thị bên dưới các điều khiển khi ta drag chuột lên điều khiển. Nó là dạng context-sensitive help. Loại help này hữu ích cho các modal DialogBox bởi vì chúng không cho phép truy xuất bất cứ tập tin hay cửa sổ nào khác của ứng dụng cho đến khi bị đóng. Điều này có nghĩa là cả help file cũng không thể sử dụng.

Các bước hiển thị pop-up help:

- Bước 1: kéo thả thành phần HelpProvider vào Form
- Bước 2: thiết lập giá trị true cho thuộc tính HelpButton
- Bước 3: để hiển thị HelpButton, các thuộc tính MaximizeBox và MinimizeBox phải được thiết lập là false, thuộc tính ControlBox thiết lập giá trị true và thuộc tính FormBorderStyle phải có 1 trong các giá trị sau: FixedSingle, Fixed3D, FixedDialog, Sizable.
- Bước 4: Chọn điều khiển trên form muốn hiển thị trợ giúp và thiết lập giá trị cho thuộc tính HelpString của nó. Điều này cho phép chuỗi văn bản là giá trị của HelpString sẽ được hiển thị trong cửa sổ tương tự như ToolTip.
- Bước 5: chạy ứng dụng.
- Bước 6: nhấp vào nút Help trên thanh tiêu đề và nhấn chọn điều khiển bạn đã thiết lập chuỗi trợ giúp.

e/- Thành phần ToolTip

Thành phần ToolTip được sử dụng để thêm các chú thích, các chỉ dẫn cho các điều khiển trên Form. Nó cung cấp thông tin trợ giúp ngắn gọn đến người dùng khi con trỏ chuột chỉ đến một điều khiển cụ thể. Thành phần cần được liên kết đến các điều khiển yêu cầu có chú thích. ToolTip class cung cấp một cửa sổ pop-up hình chủ nhật chứa thông tin trợ giúp.

ToolTip class có nhiều properties, methods và events để cung cấp chú thích cho các điều khiển. ToolTip class thuộc namespace System.Windows.Forms.

Bảng 2.18: Một số properties, methods và events của ToolTip

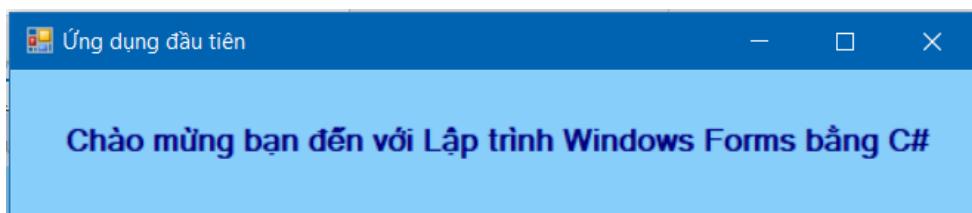
Thành phần	Tên	Mô tả
Properties	Active	ToolTip có đang được kích hoạt hay không
	IsBalloon	ToolTip có ở dạng Balloon hay không
	ShowAlways	ToolTip có được hiển thị, thậm chí khi điều khiển cha của nó không được kích hoạt hay không
	ToolTipIcon	Xác định biểu tượng được hiển thị bên cạnh ToolTip text
	ToolTipTitle	Nhận hoặc thiết lập tiêu đề cho cửa sổ ToolTip

Methods	GetToolTip	Nhận ToolTip text liên kết với điều khiển cụ thể
	Hide	Ẩn cửa sổ ToolTip được chỉ định
	RemoveAll	Hủy tất cả ToolTip text đang liên kết với thành phần ToolTip
	SetToolTip	Liên kết ToolTip text với điều khiển chỉ định
	Show	Thiết lập ToolTip text với điều khiển chỉ định, và hiển thị ToolTip
	ToString	Trả về chuỗi đại diện cho ToolTip
Events	Draw	Diễn ra khi ToolTip được vẽ (draw) và thuộc tính OwnerDraw có giá trị true và thuộc tính IsBalloon có giá trị false.
	Popup	Diễn ra trước khi ToolTip được hiển thị lúc đầu. Đây là sự kiện mặc định của ToolTip class.

CÂU HỎI VÀ BÀI TẬP

Bài 1. Ứng dụng LTW_Starters

- Tạo project có tên LTW_Starters, chọn vị trí lưu project.
- Thêm một thư mục mới có tên Bai1 vào project (Trong cửa sổ Solution Explorer, nhấp phải chuột trên project chọn Add/New Folder) để lưu trữ các Form của Bài 1.
 - Di chuyển Form mặc định lúc tạo project vào thư mục trên. Đặt tên form là frm1, tiêu đề Form là “Cửa sổ 1”.
 - Lần lượt thêm 2 Form vào project với tên frm2 và frm3. Tiêu đề Form tương ứng lần lượt là “Cửa sổ 2” và “Cửa sổ 3”
 - Chạy ứng dụng với Form bắt đầu là frm2.
 - Xóa frm2 ra khỏi project.
 - Thiết lập ứng dụng thực thi bắt đầu từ frm3.
 - Thiết kế lại frm3 như sau:

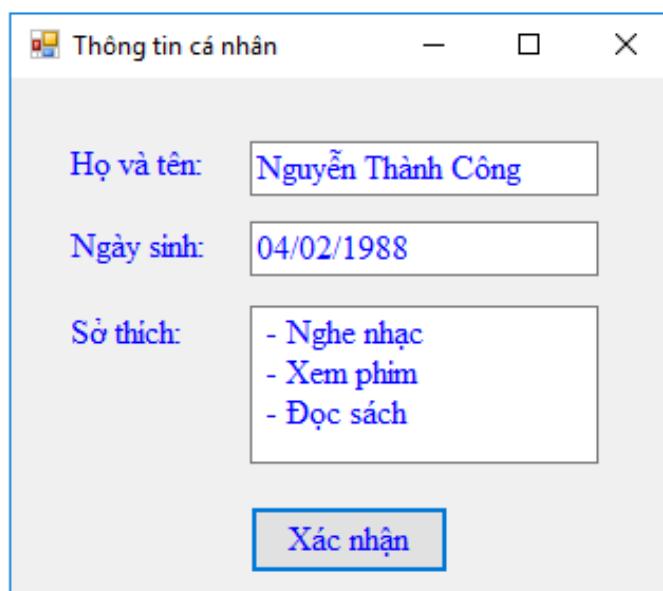


Bài 2.

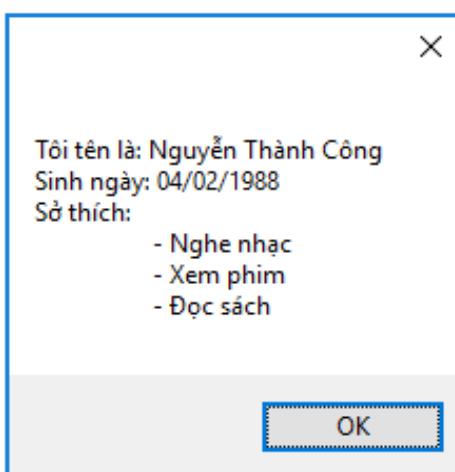
- Thiết kế Form “Chào” như sau:



- Xử lý sự kiện Click của Button “Chào”. Khi người dùng click vào Button, nội dung của TextBox bên trên sẽ hiển thị xuống Label bên dưới theo mẫu “Chào bạn...”.

Bài 3. Thiết kế Form “Thông tin cá nhân” theo mẫu sau:

Xử lý sự kiện Click của Button “Xác nhận”. Khi người dùng click vào Button sẽ xuất hiện MessageBox theo mẫu sau:



Bài 4.

- Thiết kế Form “Diện tích & Chu vi hình tròn” theo mẫu sau:

- Xử lý các sự kiện sau:

+ Khi người dùng nhập vào nút **Tính**, thực hiện kiểm tra giá trị bán kính đã nhập. Nếu giá trị hợp lệ, thực hiện tính và trả kết quả về các TextBox tương ứng. Ngược lại, hiển thị thông báo lỗi giá trị nhập liệu. Biết rằng, chu vi và diện tích hình tròn được tính như sau:

$$\text{Chu vi} = 2 * \pi * \text{Bán kính}$$

$$\text{Diện tích} = \pi * (\text{Bán kính})^2$$

Gợi ý:

- ✓ Các giá trị có kiểu số thực
- ✓ Sử dụng cấu trúc bẫy lỗi try..catch cho việc kiểm tra giá trị bán kính nhập vào.
- ✓ Sử dụng Math class cho việc lấy giá trị PI và giá trị bình phương bán kính.

+ Khi người dùng nhập vào nút **Tiếp tục**, thực hiện xóa rỗng nội dung các TextBox, đặt con trỏ tại TextBox nhập bán kính.

Bài 5.

- Thiết kế Form “Thanh toán tiền điện” theo mẫu sau:

- + Ngày thu được lấy từ ngày hệ thống nhờ vào DateTime class và được hiển thị theo định dạng dd/MM/yyyy (sử dụng phương thức Format () của String class).
- + TextBox thành tiền luôn luôn bị vô hiệu hóa, không cho nhập liệu.
- + Khi chạy ứng dụng, Form xuất hiện ở giữa màn hình.
- Xử lý các sự kiện sau:
 - + Khi Form load, ngày thu được tự động cập nhật theo yêu cầu như trên.
 - + Các TextBox chỉ số cũ, chỉ số mới, đơn giá chỉ cho phép thao tác với các phím số.
 - + Khi nhập vào nút **Tính tiền**:
 - ✓ Kiểm tra tính hợp lệ của các giá trị chỉ số cũ, chỉ số mới, đơn giá. Hiển thị các thông báo lỗi khi dữ liệu nhập không hợp lệ.
 - ✓ Tính và xuất kết quả cho TextBox Thành tiền theo công thức:

$$\text{Thành tiền} = (\text{Chỉ số mới} - \text{Chỉ số cũ}) * \text{Đơn giá}$$
 - ✓ Các TextBox nhập liệu sẽ tự động bị vô hiệu hóa, không cho phép người dùng nhập thông tin mới.
 - + Khi nhập vào nút **Làm mới**:
 - ✓ Xóa rỗng nội dung các TextBox.
 - ✓ Kích hoạt lại các TextBox để nhập liệu (trừ TextBox Thành tiền).
 - ✓ Đặt con trỏ nhập liệu tại TextBox Khách hàng.

Bài 6.

- Thiết kế Form “Thực hiện phép tính” như sau:

- + Các nút +, −, ×, / là các RadioButton ở hình dạng Button.
- + Thiết lập giá trị cho các thuộc tính của Form sao cho: khi nhấn phím Enter tương ứng với nhấp vào Button Tính, khi nhấn phím ESC tương ứng với nhấp vào Button Thoát.
- Xử lý các sự kiện:
 - + Các TextBox Số a và Số b chỉ cho phép nhập giá trị số.
 - + Khi người dùng nhấp vào nút **Tính**, ứng dụng sẽ thực hiện kiểm tra tính hợp lệ của các giá trị nhập vào và sau đó sẽ thực hiện phép toán đã chọn. Các thông

báo lỗi được bật lên nếu người dùng chưa chọn phép toán hoặc có lỗi nhập liệu xảy ra.

+ Khi người dùng nhấp vào nút **Làm mới**, thực hiện xóa rỗng các TextBox và bỏ chọn RadioButton tương ứng với phép toán đã chọn.

+ Thực hiện đóng Form hiện thời khi người dùng nhấp vào nút **Thoát**.

Bài 7. Thiết kế và xử lý các sự kiện cho Form “Chào theo giờ” theo mẫu sau:



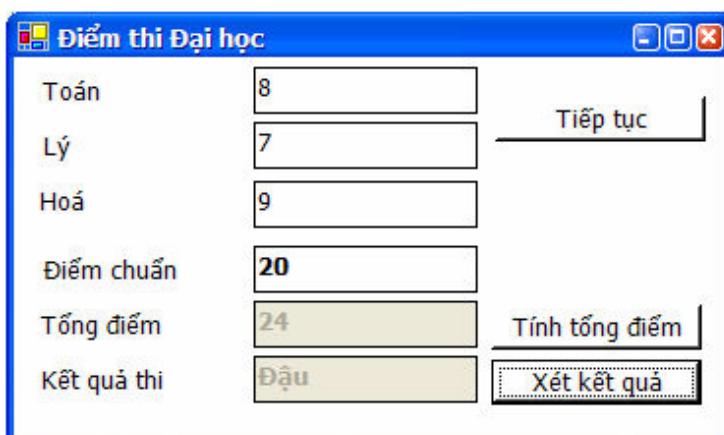
Biết rằng:

- + 5:00-11:59: Good Morning.
- + 12:00-17:59: Good Afternoon.
- + 18:00-21:59: Good Evening.
- + 22:00-4:59: Good Night.
- + Người dùng có thể nhập cả giờ và phút, phân cách nhau bằng dấu “:”.

Bài 8. Thiết kế và xử lý các sự kiện cho Form “Tìm thứ trong tuần” theo mẫu sau. (Gợi ý: sử dụng *DateTime class*)



Bài 9. Thiết kế và xử lý các sự kiện cho Form “Điểm thi Đại học” theo mẫu:



Nếu Tổng điểm lớn hơn Điểm chuẩn thì Kết quả thi được ghi nhận là Đậu, ngược lại thì ghi nhận là Rớt.

Bài 10. Thiết kế ứng dụng MDI với tên Toán học vui:

Với các menu chức năng cụ thể như sau:

- **Thao tác với số nguyên**

- + *Đọc số*: liên kết đến form “Đọc số” theo mẫu sau:

- + *Bảng cửu chương*: liên kết đến form “Bảng cửu chương” theo mẫu sau:

- + *Tìm số lớn nhất*: liên kết đến form “Tìm số lớn nhất” theo mẫu sau:

- + *Chẵn lẻ, âm dương*: liên kết đến một form cho phép kiểm tra tính chẵn, lẻ, âm, dương của số nguyên n. Form liên kết được thiết kế tùy ý đáp ứng yêu cầu.
- + *Số nguyên tố, số chính phương*: liên kết đến một form cho phép kiểm tra số nguyên dương n có phải là số nguyên tố, số chính phương hay không. Form liên kết được thiết kế tùy ý đáp ứng yêu cầu.
- + *Tính BCNN và UCLN*: liên kết đến một form cho phép tìm bội chung nhỏ nhất và ước chung lớn nhất của 2 số nguyên dương m và n. Form liên kết được thiết kế tùy ý đáp ứng yêu cầu.

- **Thao tác với dãy số nguyên**

- + *Tính toán trên dãy số*: liên kết đến form “Tính toán trên dãy số” theo mẫu sau:

- + *Min, Max, Avg và AvgMul*: sau đó có thể xác định các giá trị nhỏ nhất, lớn nhất, giá trị trung bình cộng, trung bình nhân của dãy số vừa nhập.
Form liên kết được thiết kế tùy ý, đáp ứng yêu cầu.

Bài 11. Thiết kế giao diện như sau:

Yêu cầu:

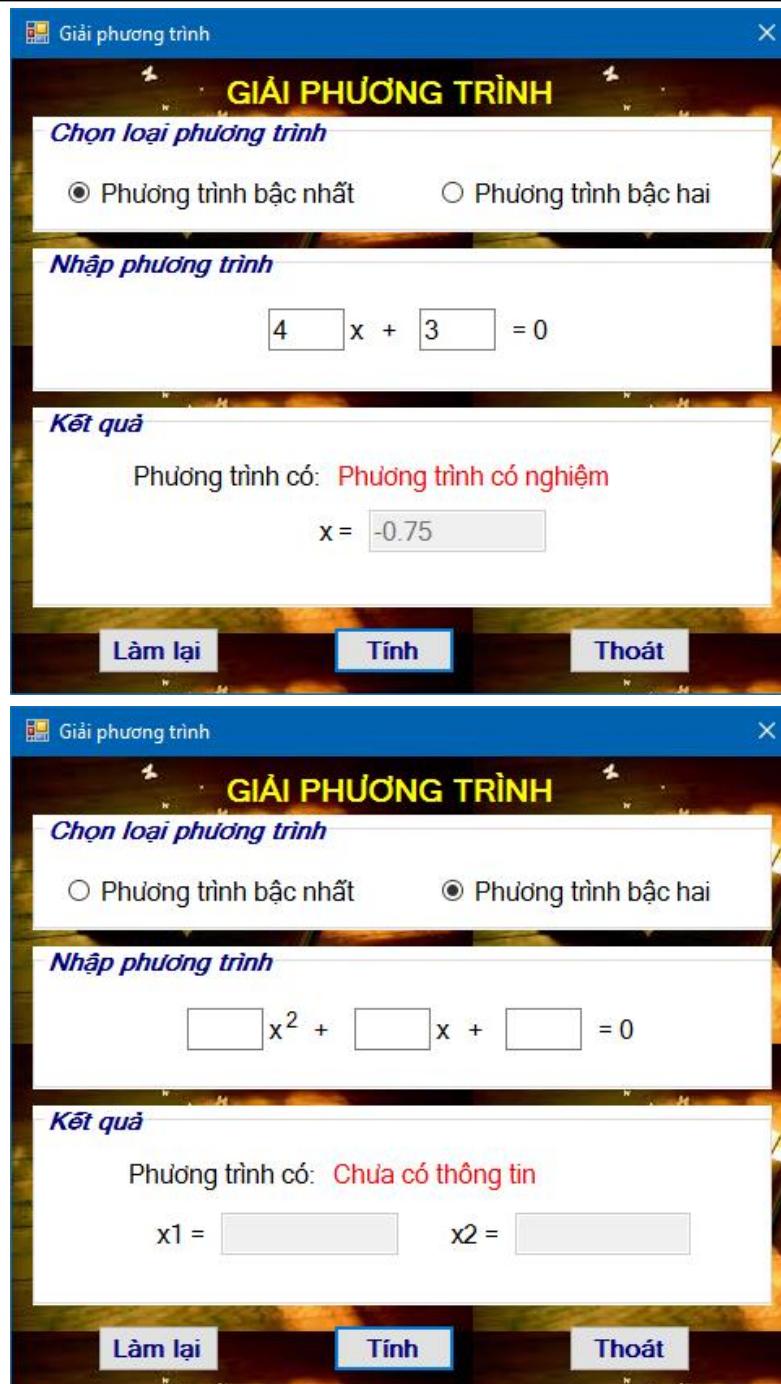
- Khi Form vừa hiện lên, các Textbox, Combobox, Listbox chưa có dữ liệu, con trỏ đặt tại Textbox (thiết lập Tab Order hợp lý).
- Nhấn nút “Cập nhật” hoặc Enter: thêm số vừa nhập ở Textbox vào Combobox (nhớ kiểm tra dữ liệu nhập), đồng thời xóa nội dung Textbox và đặt con trỏ lại Textbox.
- Khi chọn 1 số trên Combobox thì danh sách các ước số của số này sẽ hiển thị vào Listbox bên phải tương ứng.
- Khi nhấn các nút: “Tổng các ước số”, “Số lượng các ước số chẵn”, “Số lượng các ước số nguyên tố” thì sẽ hiển thị thông tin tương ứng vào Messagebox dựa vào các ước số trên Listbox.

Bài 12. Thiết kế giao diện như sau:**Yêu cầu:**

- Khi Form vừa hiện lên, các Textbox, Listbox đều trống, con trỏ đặt tại Textbox (thiết lập Tab Order hợp lý).
- Khi người sử dụng nhập một số vào Textbox rồi Enter hoặc nhấn nút "Nhập" thì số đó được thêm vào Listbox, đồng thời nội dung trong Textbox bị xóa và con trỏ được chuyển về Textbox.
- Người dùng nhấn vào nút nào thì thực hiện chức năng tương ứng của nút đó. Hiện kết quả ra Messagebox (nếu có).
- Thiết lập thuộc tính Anchor hợp lý cho các control.
- Thiết lập MinimumSize cho form.

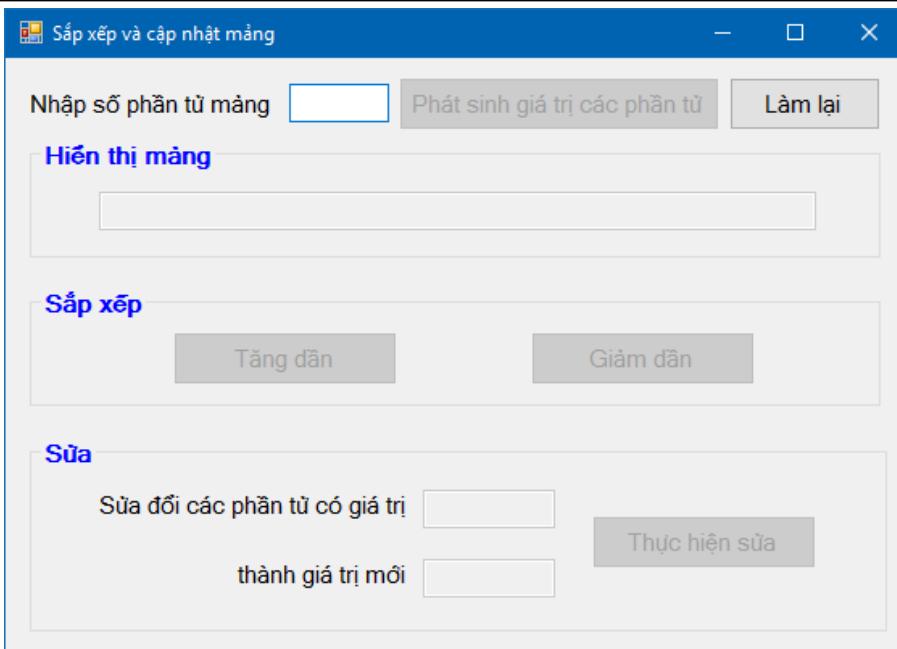
Bài 13. Thiết kế và xử lý sự kiện cho Form “Giải phương trình” theo mẫu sau:**Yêu cầu:**

- Khi người dùng chọn loại phương trình, thông tin ở mục Nhập phương trình và Kết quả sẽ được hiển thị tương ứng như trong các hình.
- Chèn hình nền cho Form.
- Xử lý sự kiện khi người dùng nhấp vào các nút Làm lại, Tính và Thoát.

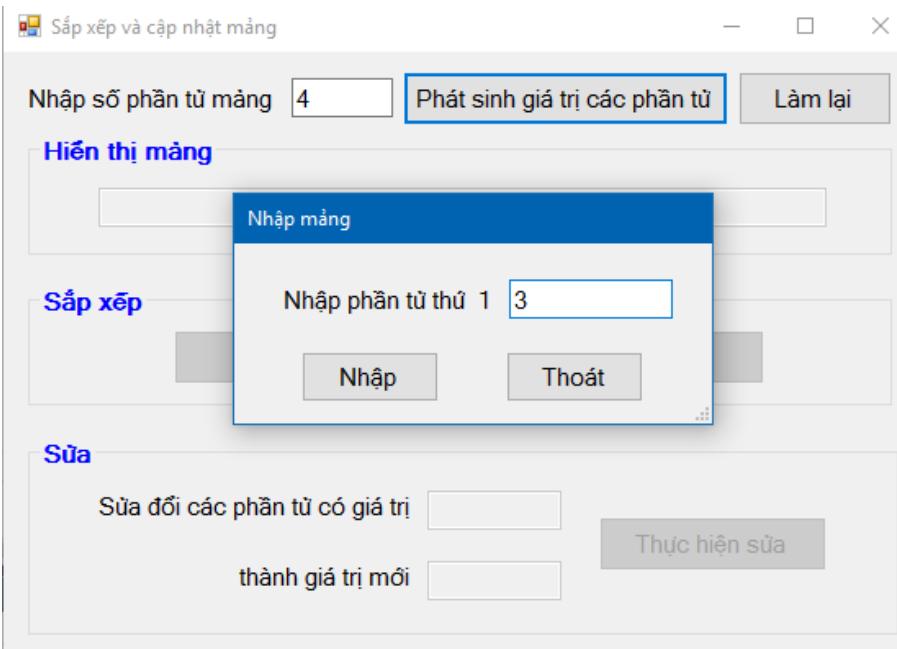


Bài 14. Xây dựng ứng dụng cho phép nhập vào một mảng sau đó:

- + Thực hiện sắp xếp các phần tử mảng theo thứ tự tăng dần và giảm dần.
- + Thực hiện thay đổi giá trị của các phần tử trong mảng.
- Bước 1: Thiết kế Form “Sắp xếp và cập nhật mảng” theo mẫu.
- Bước 2: Sau khi người dùng nhập số phần tử của mảng, nếu số phần tử hợp lệ, nút Phát sinh giá trị các phần tử sẽ được kích hoạt.



- Bước 3: Khi người dùng nhập vào nút Phát sinh giá trị các phần tử mảng thì Form “Nhập mảng” sẽ được hiển thị dưới dạng Modal Dialog Box để người dùng có thể lần lượt nhập giá trị của các phần tử trong mảng. Lưu ý, số lượng giá trị nhập vào phải bằng số phần tử của mảng ở Form chính.



- Bước 4: Quá trình nhập mảng hoàn tất, quay về cửa sổ chính. Lúc này các điều khiển thuộc nhóm Sắp xếp và Sửa sẽ được bật lên để người dùng thao tác.

- Bước 5: Khi người dùng nhấp vào nút Tăng dần hoặc nút Giảm dần, nội dung mảng trong TextBox ở phần hiển thị mảng sẽ được sắp xếp theo trật tự tương ứng.

- Bước 6: Thao tác với nhóm Sửa, người dùng thực hiện nhập các giá trị được yêu cầu sau đó nhấp vào nút Thực hiện sửa. Nếu tìm thấy phần tử có giá trị cần sửa trong mảng thì thay giá trị của nó bằng giá trị mới. Ngược lại, hiển thị thông báo không tìm thấy phần tử trong mảng.

- Bước 7: xử lý sự kiện khi người dùng nhấp vào nút Làm lại, thiết lập lại trạng thái ban đầu của Form “Sắp xếp và cập nhật mảng”.

Bài 15. Thiết kế và xử lý các sự kiện cho Form “Xử lý mảng” theo các hình minh họa sau:

The screenshot shows a Windows application window titled "Xử lý mảng". Inside, there are several sections:

- Nhập số phần tử mảng**: An input field with value "5" and a text box containing "2,3,4,5,6,7,8".
- Hiển thị mảng**: A button labeled "Xuất mảng" followed by a list box showing the array elements: "2 3 4 5 6".
- Tính tổng**: Three input fields showing totals: "Tổng các phần tử trong mảng" (20), "Tổng các phần tử chẵn trong mảng" (12), and "Tổng các phần tử lẻ trong mảng" (8). Next to each is a "Tính tổng" button.
- Tìm Min/Max**: Two input fields showing the minimum and maximum values: "Phần tử nhỏ nhất trong mảng" (2) and "Phần tử lớn nhất trong mảng" (6). Next to each is a "Tìm Min" or "Tìm Max" button.
- Tìm kiếm**: An input field with value "7" and a "Tìm kiếm" button.
- Buttons at the bottom**: "Làm lại" (Reset) and "Mở rộng" (Expand).

Bài 16. Khách sạn WINGS cần quản lý các phòng nghỉ với các thông tin như sau: Mã số phòng, tầng, số thứ tự phòng theo tầng, mã loại phòng, VIP, đơn giá ngày. Trong đó:

+ Mã số phòng được tạo từ mã loại phòng, tầng, số thứ tự phòng theo tầng. Nếu là phòng VIP, phía sau mã phòng sẽ có thêm từ VIP.

Ví dụ: STD1S.02.007 (STD1S.02.007.VIP) là mã số phòng của phòng Standard 1 giường đơn, phòng số 007 ở tầng 2 (VIP).

+ Mã loại phòng:

Mã loại phòng	Mô tả
STD1S	Phòng Standard 1 giường đơn
STD1L	Phòng Standard 1 giường đôi
SUP1S	Phòng Superior 1 giường đơn
SUP1L	Phòng Superior 1 giường đôi
SUP2S	Phòng Superior 2 giường đơn
SUP2L	Phòng Superior 2 giường đôi

Giả sử bài toán không quan tâm đến ràng buộc mã số phòng là duy nhất.

Yêu cầu:

1. Xây dựng class với các thuộc tính, phương thức phù hợp với mô tả bài toán
2. Thiết kế form với các chức năng sau:
 - Cho phép nhập thông tin của N phòng, N không quá 100 và hiển thị các thông tin vừa nhập ra màn hình. Lưu ý, mã số phòng phải được tổng hợp từ các thông tin còn lại

(không nhập mã số phòng); số khách tối đa của phòng được tính dựa trên số khách theo giường có trong phòng cộng thêm 1. Trong đó, giường đơn tương ứng với 1 khách, giường đôi tương ứng với 2 khách.

- Cho phép tìm thông tin phòng nghỉ theo mã loại phòng cho 2 trường hợp gần đúng và chính xác.

Bài 17. Trung tâm thông tin thư viện trường Đại học SPKT Vĩnh Long cần quản lý độc giả là cán bộ viên chức nhà trường với các thông tin sau: MSCB, họ tên, ngày sinh, phái, chức vụ. Trong đó, MSCB là một chuỗi gồm 9 ký tự (XXX.XX.XX) gồm:

+ 3 ký tự đầu thể hiện tên đơn vị:

Ký hiệu	Tên Đơn vị
CTM	Khoa Cơ khí chế tạo máy
CTP	Khoa Công nghệ thực phẩm
DDT	Khoa Điện-Điện tử
FIT	Khoa Công nghệ thông tin
OTO	Khoa Cơ khí động lực

+ 2 ký tự giữa thể hiện năm bắt đầu công tác;

+ 2 ký tự cuối thể hiện số thứ tự của cán bộ trong đơn vị

Ví dụ: **FIT.13.13.** Cán bộ thuộc khoa Công nghệ thông tin, bắt đầu công tác từ năm 2013 và STT là 13.

Giả sử, trong bài toán không quan tâm đến ràng buộc MSCB là duy nhất.

Yêu cầu:

1. Xây dựng class với các thuộc tính và phương thức phù hợp với mô tả bài toán
2. Thiết kế form với các chức năng sau:

- Cho phép nhập thông tin của N độc giả, N không quá 1000 và hiển thị các thông tin vừa nhập ra màn hình. Lưu ý cần thể hiện đầy đủ thông tin về đơn vị công tác, năm bắt đầu công tác thông qua MSCB của cán bộ viên chức.

Cho phép tìm thông tin độc giả theo mã đơn vị công tác (3 ký tự đầu của MSCB)

Chương 3

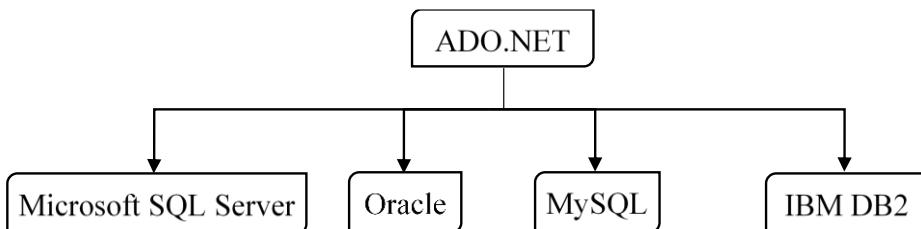
THAO TÁC DỮ LIỆU VỚI ADO.NET

3.1 Kiến trúc tổng quan của ADO.NET

3.1.1 ADO.NET

ADO.NET (ADO là viết tắt của cụm từ ActiveX Data Objects) là một tập hợp các lớp dịch vụ truy cập dữ liệu cho các nhà lập trình .NET Framework. ADO.NET cung cấp một tập phong phú các thành phần cho phép tạo các ứng dụng phân tán, chia sẻ dữ liệu. ADO.NET hỗ trợ đa dạng những nhu cầu phát triển, bao gồm tạo các máy khách cơ sở dữ liệu front-end và các đối tượng nghiệp vụ tầng giữa được sử dụng bởi các ứng dụng, công cụ, ngôn ngữ hoặc các trình duyệt Internet.

ADO.NET là một phần không thể thiếu của .NET Framework, cung cấp các truy cập đến cơ sở dữ liệu quan hệ (ví dụ như SQL Server), XML và các nguồn dữ liệu khác thông qua các trình điều khiển CSDL như OLE DB (Object Linking and Embedding for Databases) và ODBC (Open Database Connectivity). Các ứng dụng người dùng chia sẻ dữ liệu có thể sử dụng ADO.NET cho việc kết nối đến nguồn dữ liệu (tên máy chủ thiết lập kết nối) để truy xuất, xử lý và cập nhật dữ liệu trong CSDL. ADO.NET hỗ trợ hai môi trường lập trình khác nhau đó là **mô hình kết nối** và **mô hình ngắt kết nối**.



Hình 3.1: ADO.NET cung cấp truy cập đến các hệ quản trị cơ sở dữ liệu khác nhau

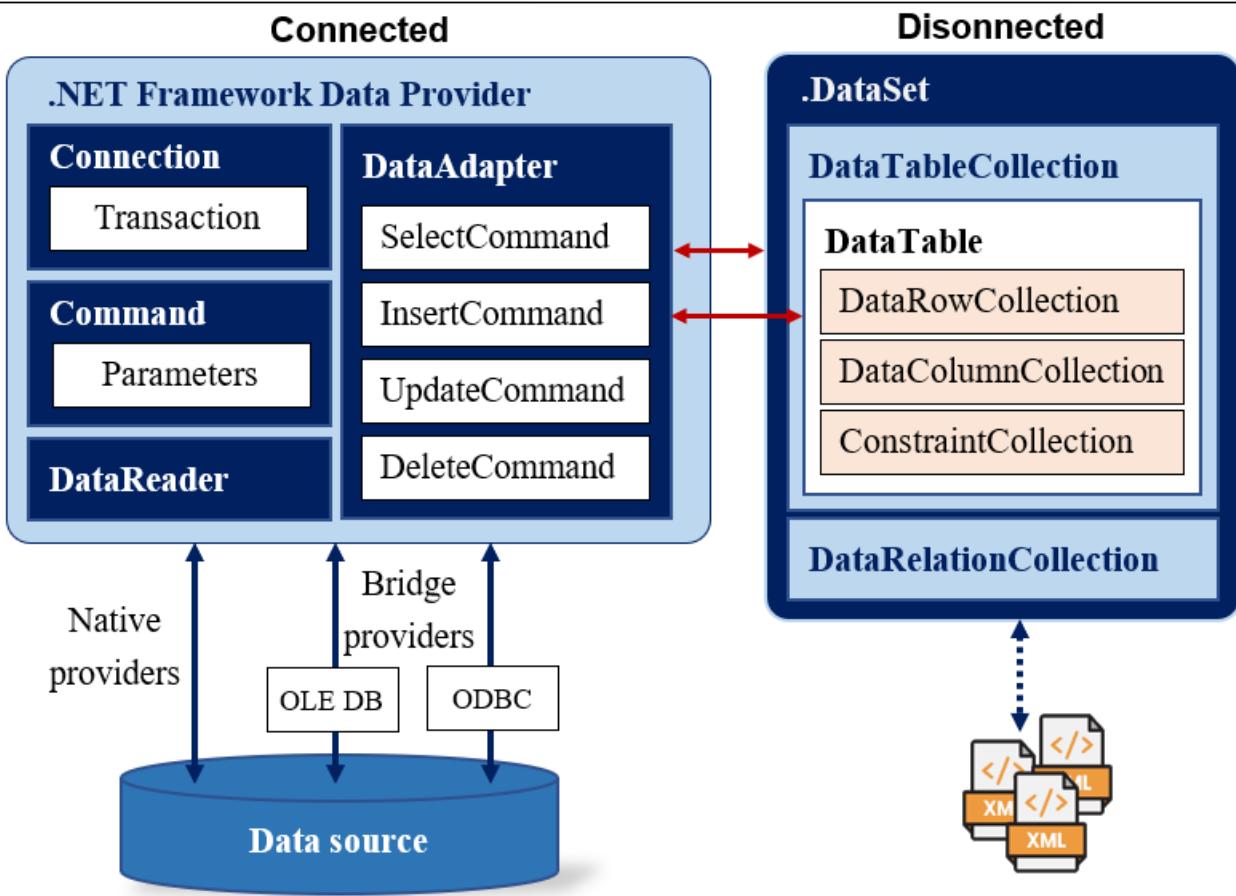
3.1.2 Các thành phần của ADO.NET

Việc xử lý dữ liệu truyền thống chủ yếu phụ thuộc vào mô hình 2 tầng dựa trên kết nối. Khi việc xử lý dữ liệu gia tăng nhu cầu sử dụng các kiến trúc đa tầng, các lập trình viên có xu hướng chuyển sang cách tiếp cận ngắt kết nối để các ứng dụng của họ có khả năng mở rộng tốt hơn.

ADO.NET có 2 thành phần chính hỗ trợ cho việc truy cập và thao tác dữ liệu đó là **.NET Framework Data Provider** và **DataSet**.

a/- .NET Framework Data Provider

.NET Framework Data Provider được sử dụng để kết nối cơ sở dữ liệu, thực thi các lệnh và nhận kết quả. Các kết quả này hoặc sẽ được xử lý trực tiếp, đặt vào DataSet để hiển thị với người dùng khi cần, kết hợp với dữ liệu từ nhiều nguồn khác nhau, hoặc được điều khiển từ xa giữa các tầng. .NET Framework Data Provider có dung lượng nhẹ, tạo ra một lớp nhỏ giữa nguồn dữ liệu và mã nguồn giúp tăng hiệu suất.



Hình 3.2: Các thành phần của ADO.NET

Những đối tượng cốt lõi của .NET Framework Data Provider

Connection: cung cấp các kết nối đến nguồn dữ liệu. Lớp cơ sở cho các đối tượng Connection là `DbConnection`.

Command: cho phép truy cập các lệnh cơ sở dữ liệu để trả về dữ liệu, sửa đổi dữ liệu, chạy các stored procedures và gửi hoặc nhận thông tin các tham số (parameter). Lớp cơ sở của các đối tượng Command là `DbCommand`.

DataReader: cung cấp luồng dữ liệu hiệu suất cao từ nguồn dữ liệu (data source), đọc luồng dữ liệu chỉ chuyển tiếp (forward-only) và chỉ đọc (read-only) từ nguồn dữ liệu. Lớp cơ sở cho các đối tượng DataReader là `DbDataReader`.

DataAdapter: cung cấp cầu nối giữa đối tượng DataSet và nguồn dữ liệu. DataAdapter sử dụng những đối tượng Command để thực thi các lệnh SQL ở nguồn dữ liệu để tải dữ liệu lên DataSet và điều chỉnh những thay đổi trong DataSet về lại nguồn dữ liệu. DataAdapter chủ yếu thực hiện các thao tác như `SELECT`, `INSERT`, `UPDATE` và `DELETE`. Lớp cơ sở của các đối tượng DataAdapter là `DbDataAdapter`.

Về mặt thực tế, .NET Framework Data Provider cung cấp giao diện lập trình chung để làm việc với các nguồn dữ liệu. Mỗi “nhà cung cấp” đặc thù sẽ đưa ra một loại data provider riêng. Dưới đây là bảng mô tả các data provider mà ADO.NET cung cấp sẵn:

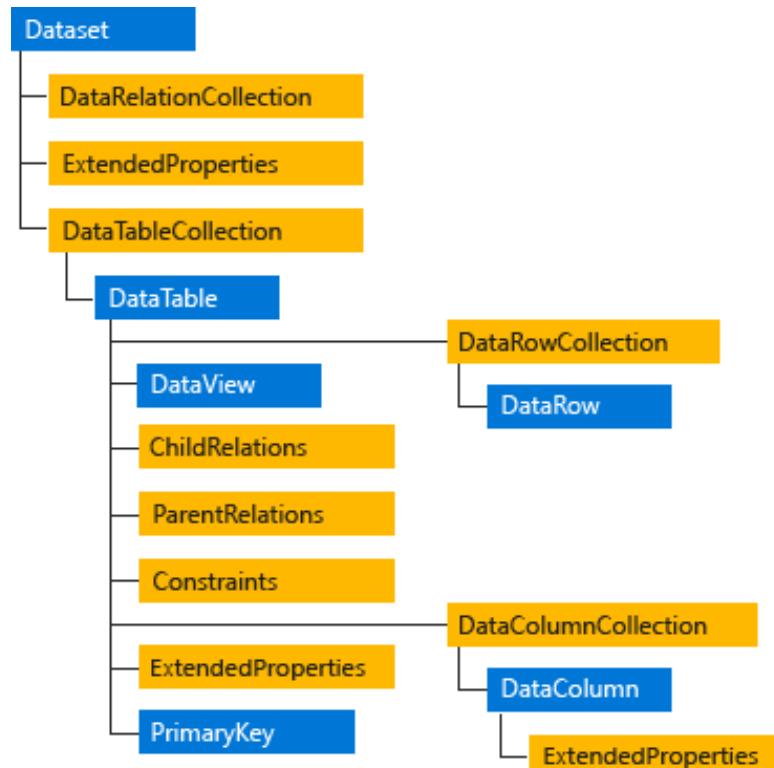
Bảng 3.1: Mô tả các data provider được ADO.NET cung cấp sẵn

Data provider	Connection	Command	DataReader	DataAdapter
SQL Server Provider	SqlConnection	SqlCommand	SqlDataReader	SqlDataAdapter
Oracle Provider	OracleConnection	OracleCommand	OracleDataReader	OracleDataAdapter
OLEDB Provider	OleDbConnection	OleDbCommand	OleDbDataReader	OleDbDataAdapter
ODBC Provider	OdbcConnection	OdbcCommand	OdbcDataReader	OdbcDataAdapter

b/- DataSet

DataSet được thiết kế cho việc truy cập dữ liệu hoàn toàn độc lập với bất kỳ nguồn dữ liệu nào. Vì vậy, nó có thể được sử dụng với nhiều nguồn dữ liệu khác nhau, được sử dụng với dữ liệu XML hay được sử dụng cho việc quản lý dữ liệu cục bộ của ứng dụng.

DataSet như một CSDL thu nhỏ tại máy khách (client) chứa tập của một hoặc nhiều đối tượng DataTable bao gồm các dòng và cột dữ liệu, cũng có khóa chính, khóa ngoại, ràng buộc toàn vẹn và thông tin quan hệ dữ liệu trong các đối tượng DataTable. Tất cả dữ liệu từ nguồn dữ liệu thực sẽ được nạp vào DataSet dưới dạng các DataTable, đó là một “ảnh chụp” của nguồn dữ liệu. Khỏi dữ liệu này sẽ được chỉnh sửa độc lập, sau đó nếu cần sẽ được cập nhật trở lại nguồn dữ liệu. Theo nguyên tắc này, chúng ta không cần duy trì kết nối liên tục không cần thiết với nguồn dữ liệu trong suốt quá trình thao tác với nó. Vì vậy, thành phần này còn được gọi là tầng không kết nối (disconected layer).



Hình 3.3: cấu trúc DataSet

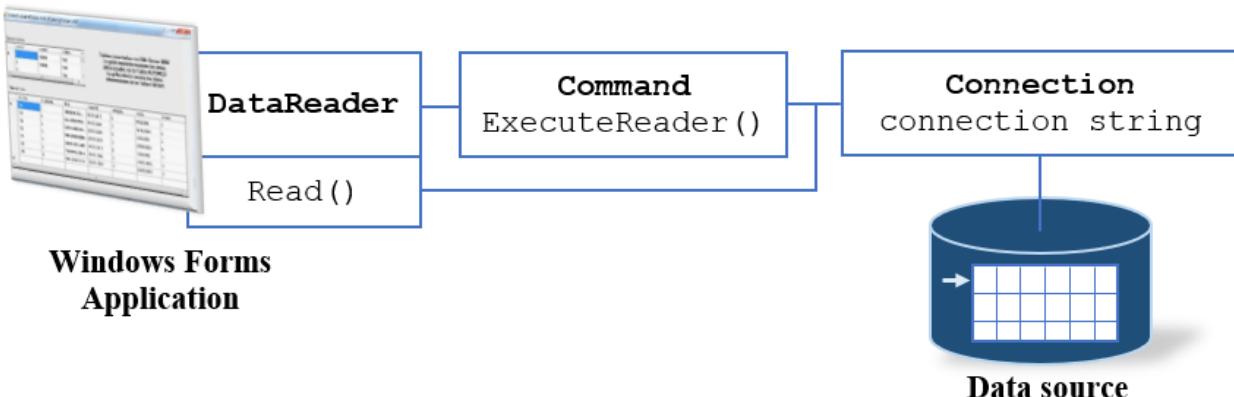
Để làm việc với DataSet, chúng ta có thể áp dụng độc lập hoặc kết hợp các cách sau đây:

- Lập trình để tạo DataTable, DataRelation và Constraint trong DataSet, sau đó điền dữ liệu vào các bảng.
- Điền DataSet với các bảng dữ liệu từ nguồn dữ liệu quan hệ đã có bằng cách sử dụng DataAdapter.
- Sử dụng XML để tải và duy trì nội dung DataSet.

3.2 CÁC MÔ HÌNH TRUY XUẤT DỮ LIỆU

3.2.1 Mô hình kết nối

Khi làm việc với ADO.NET theo mô hình kết nối (connected model), kết nối giữa đối tượng DataReader của ứng dụng với nguồn dữ liệu (data source) sẽ được duy trì hoạt động. Một mẫu tin (record) sẽ được trả về từ nguồn dữ liệu thông qua việc gọi phương thức Read () của đối tượng DataReader. Điểm quan trọng nhất của mô hình kết nối đó là dữ liệu được lấy từ tập dữ liệu trả về (các mẫu tin được trả về bởi một lệnh SQL) theo kiểu chỉ đọc (read-only) và chỉ chuyển tiếp (forward-only) từng mẫu tin cho một lần đọc. Hình dưới đây mô tả cách thức hoạt động của DataReader trong chế độ kết nối.



Hình 3.4: Cách thức hoạt động của DataReader trong chế độ kết nối

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

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

Ví dụ 3.1: minh họa cách bước điền hình làm việc với đối tượng DataReader

```

...
using System.Data.SqlClient;
...
// (1) Tao Connection
SqlConnection conn = new SqlConnection(chuoiKetNoi);
conn.Open();
// (2) Chuoi SQL thuc hien lay danh sach ten cac sinh vien
//      xep tang dan theo NgaySinh
string sql = "SELECT HoTen FROM SinhVien ORDER BY NgaySinh";
// (3) Tao doi tuong Command
SqlCommand cmd = new SqlCommand(sql, conn);
SqlDataReader dr;
// (4) Tao doi tuong DataReader
dr = cmd.ExecuteReader(CommandBehavior.CloseConnection);
while (dr.Read())
{
    listBox1.Items.Add(dr["HoTen"]); // Fill ListBox
}
dr.Close(); // Dong datareader sau khi da su dung xong

```

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

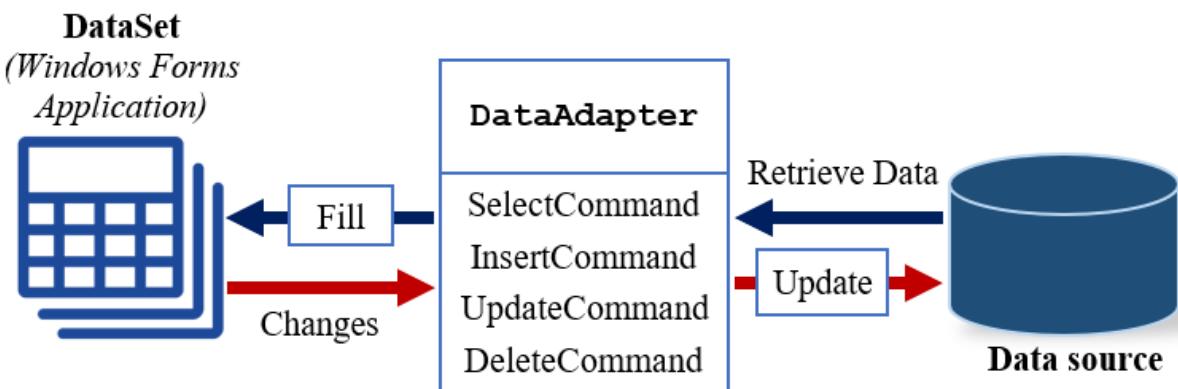
3.2.2 Mô hình ngắt kết nối

Triết lý của mô hình Ngắt kết nối đó là: Dữ liệu được nạp thông qua một lệnh SQL từ nguồn dữ liệu thực vào bộ nhớ đệm tại máy khách (client) - xử lý cục bộ tập dữ liệu kết quả tại client - mọi thay đổi trên tập dữ liệu cục bộ sau đó sẽ được cập nhật trở lại nguồn dữ liệu thực.

Mô hình được gọi là “*ngắt kết nối*” bởi vì kết nối chỉ được mở đủ lâu để đọc dữ liệu từ nguồn dữ liệu hoặc tiến hành các thao tác cập nhật lại nguồn dữ liệu thực. Bằng cách đưa dữ liệu về phía 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 nhiên, mô hình này cũng có nhược điểm về thời gian cần để nạp tập dữ liệu cũng như dung lượng bộ nhớ dùng để chứa dữ liệu tại 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 cập nhật 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 client, chứa một hay nhiều DataTables, giữa các DataTable này cũng có thể có các mối quan hệ (DataRelation) 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 3.5: Các thành phần chính của mô hình ngắt kết nối

Trong các thành phần của DataAdapter, Fill() và Update() được xem là hai phương thức quan trọng nhất. Fill() chuyển một query đến cơ sở dữ liệu và lưu tập kết quả trả về trong một DataTable của DataSet; phương thức Update() thực hiện một thao tác thêm, xóa, cập nhật cho nguồn dữ liệu 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 quan hệ, và đưa nó vào một DataSet.

Ví dụ 3.2:

```

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

1. Tạo 1 đối tượng SqlDataAdapter bằng cách truyền một câu lệnh SELECT và chuỗi kết nối cho bộ khởi tạo của lớp. DataAdapter sẽ tự tạo ra đối tượng Connection cũng như tự quản lý việc mở, đóng Connection khi cần thiết.

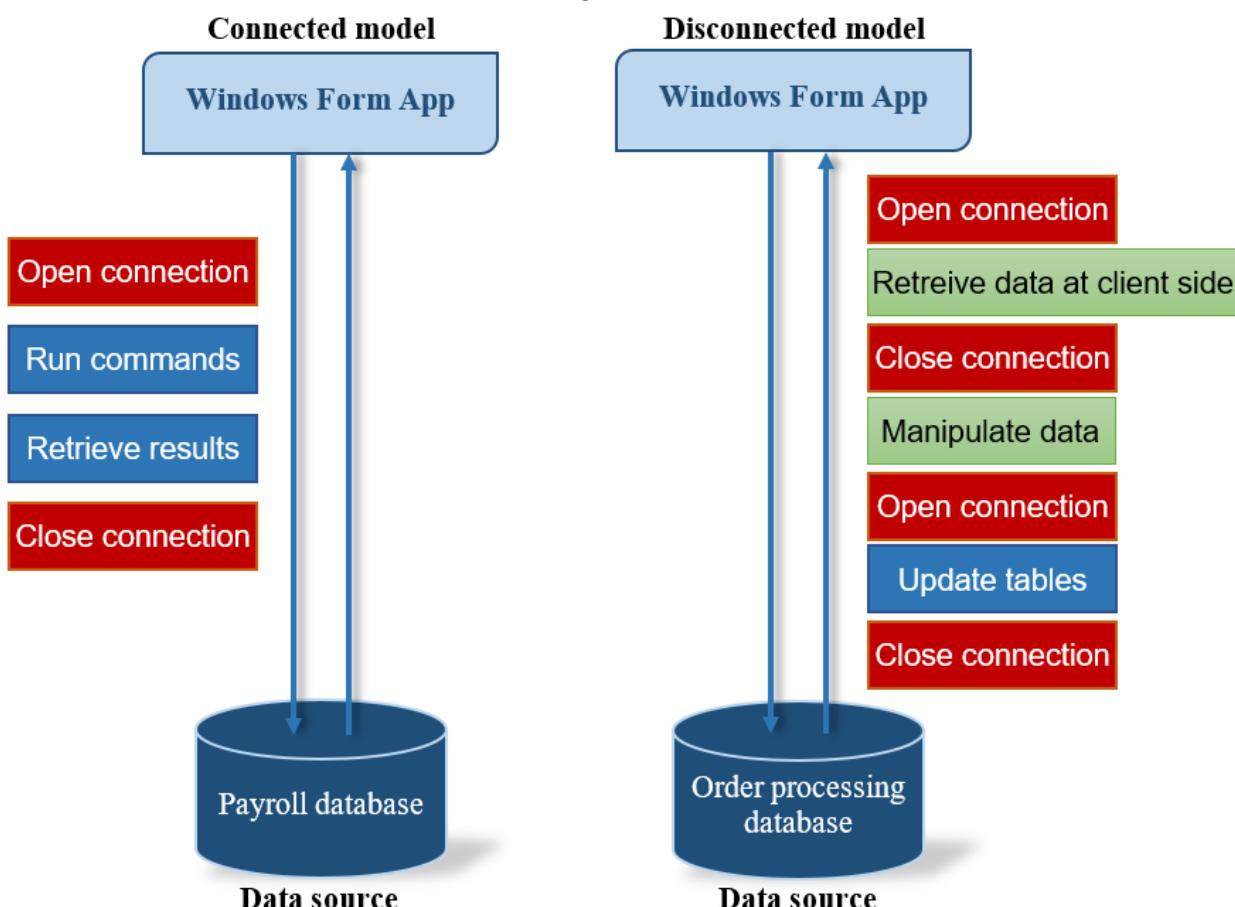
2. Tạo DataSet rỗng

3. Sau khi DataSet rỗng được tạo ra, phương thức Fill() của DataAdapter sẽ tạo trong DataSet một DataTable có tên “**Sinhvien**” và nạp các dòng dữ liệu và DataTable này bằng câu lệnh SQL dạng SELECT của DataAdapter. Mỗi cột của DataTable sẽ tương ứng với một cột trong bảng dữ liệu ở nguồn dữ liệu.

4. Dữ liệu trong DataTable sau đó sẽ được đưa vào một ListBox bằng cách duyệt qua danh sách các dòng của DataTable.

3.2.3 Lựa chọn giữa mô hình kết nối và mô hình ngắt kết nối

DataReader và DataAdapter đưa ra hai cách tiếp cận truy xuất dữ liệu trong ADO.NET: mô hình kết nối và mô hình ngắt kết nối



Hình 3.6: Làm việc với mô hình kết nối và mô hình ngắt kết nối trong ADO.NET

Với DataReader chúng ta có thể nhận luồng dữ liệu chỉ đọc (read-only) và chỉ chuyển tiếp (forward-only). Kết quả trả về khi lệnh truy vấn cơ sở dữ liệu được thực thi, và được lưu trên network buffer tại client cho đến khi chúng ta yêu cầu chúng bằng phương thức Read() của DataReader. Sử dụng DataReader có thể giúp tăng hiệu năng của

ứng dụng cho cả việc nhận dữ liệu ngay khi chúng có giá trị và lưu trữ mỗi lần một mẩu tin trong bộ nhớ vì vậy có thể làm giảm quá tải hệ thống.

Tuy nhiên, theo cách tiếp cận của mô hình kết nối, kết nối đến CSDL được thiết lập theo yêu cầu của ứng dụng. Kết nối sẽ được mở cho đến khi ứng dụng đóng, hay nói cách khác kết nối mở trong suốt “thời gian sống” của ứng dụng hoặc có yêu cầu đóng kết nối. Điều này đặt ra các vấn đề cho sự an toàn CSDL và cũng tạo ra các lưu thông mạng không cần thiết (unnecessary network traffic). Cũng vì thế, hướng tiếp cận này không thể đảm bảo năng xuất tốt nếu số lượng người dùng gia tăng.

Trong mô hình ngắt kết nối, DataAdapter được dùng để nhận dữ liệu từ nguồn dữ liệu và gắn các bảng vào trong một DataSet. DataAdapter cũng cập nhật những thay đổi từ DataSet về lại nguồn dữ liệu. DataAdapter sử dụng đối tượng Connection của .NET Framework Data Provider để kết nối nguồn dữ liệu và dùng đối tượng Command để nhận dữ liệu và giải quyết các thay đổi đối với nguồn dữ liệu. So với mô hình kết nối, DataSet trong mô hình ngắt kết nối cho phép truy xuất theo kiểu read/write nhưng nó lại yêu cầu phải có đủ bộ nhớ để quản lý bản sao dữ liệu nạp về từ nguồn dữ liệu.

Như vậy, chúng ta có thể đưa ra một số quy tắc chung: nếu ứng dụng không cần tính năng cập nhật dữ liệu và hầu như chỉ hiển thị và chọn lọc dữ liệu, DataReader là lựa chọn thích hợp; nếu ứng dụng cần cập nhật dữ liệu, giải pháp sử dụng DataSet nên được xem xét. Tất nhiên, cũng có một số tình huống đi ngược lại các quy tắc này. Chẳng hạn, nếu nguồn dữ liệu chứa số lượng lớn record, khi đó DataSet phải yêu cầu quá nhiều tài nguyên bộ nhớ; hoặc nếu dữ liệu chỉ yêu cầu một vài thao tác cập nhật thì sự kết hợp giữa DataReader và Command để thực thi cập nhật sẽ có ý nghĩa hơn.

Bảng 3.2: So sánh giữa sử dụng DataSet và DataReader

DataSet	DataReader
<ul style="list-style-type: none"> - Dữ liệu cần được serialize (tuần tự hóa) và/hoặc gửi đi bằng HTTP. - Các điều khiển read-only trên Form được buộc (bind) dữ liệu - Một điều khiển trên Form ví dụ như DataGridView được buộc dữ liệu có khả năng cập nhật được. - Một ứng dụng Windows Forms cần thêm, xóa, sửa các dòng dữ liệu 	<ul style="list-style-type: none"> - Cần quản lý một số lượng lớn các mẩu tin, lớn đến mức mà bộ nhớ và thời gian để nạp dữ liệu cho DataSet là phi thực tế. - Dữ liệu read-only và được kết buộc với một điều khiển loại danh sách của Windows Forms hoặc Web Forms - CSDL là không ổn định và thay đổi thường xuyên.

3.3 LÀM VIỆC VỚI MÔ HÌNH KẾT NỐI TRONG ADO.NET

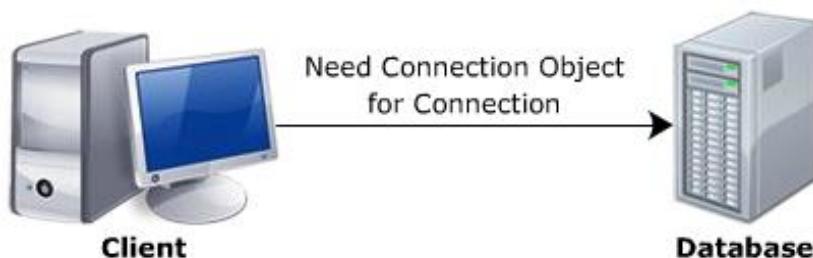
Cho dù làm việc với mô hình kết nối hay mô hình ngắt kết nối, bước đầu tiên trong quá trình truy xuất dữ liệu vẫn là tạo một kết nối làm đường truyền giữa ứng dụng và nguồn dữ liệu thông qua đối tượng của lớp Connection. Khi làm việc với mô hình kết nối, đường

truyền này sẽ được duy trì trạng thái hoạt động cho đến khi thao tác được hoàn tất. Các thao tác này bao gồm thêm, xóa, sửa hay đọc dữ liệu từ nguồn dữ liệu được thực hiện nhờ vào các đối tượng Command.

3.3.1 Đối tượng Connection

Đối tượng Connection cho phép tạo kết nối giữa ứng dụng với CSDL (nguồn dữ liệu). Nó lưu trữ các thông tin được yêu cầu cho việc thiết lập kết nối. Bằng cách thiết lập kết nối, ứng dụng có thể truy xuất và thao tác trực tiếp với dữ liệu trong CSDL.

Chúng ta sử dụng đối tượng Connection nếu muốn ứng dụng truy xuất CSDL nhiều lần. Kết nối giữa ứng dụng và nguồn dữ liệu chỉ được thiết lập cho một lần cụ thể cho việc nhận hoặc cập nhật dữ liệu.



Hình 3.7: Tạo kết nối thông qua đối tượng Connection

Trong ADO.NET có nhiều Connection class tương ứng với các DataProvider (như đã trình bày ở các phần trước). Mặc dù mỗi class có thể có những đặc tính riêng biệt, nhưng các class này phải triển khai giao diện IDbConnection. Bảng dưới đây tóm tắt các thành phần được định nghĩa bởi IDbConnection

Thành phần	Tên	Mô tả
Properties	ConnectionString	Get/Sét chuỗi kết nối đến nguồn dữ liệu
	ConnectionTimeout	Khoảng thời gian tối đa tính bằng giây để chờ thực hiện kết nối đến nguồn dữ liệu
	Database	Tên CSDL ứng với Connection hiện tại
	State	Trạng thái hiện tại của Connection. Trả về một giá trị kiểu enum: Broken, Closed, Connecting, Executing, Fetching hoặc Open.
Methods	Open	Mở một Connection. Roll back mọi thao tác đang làm
	Close	Đóng Connection- trả Connection cho Connection Pool nếu như có sử dụng Connection Pool
	BeginTransaction	Khởi tạo một database transaction
	ChangeDatabase	Thay đổi CSDL hiện tại cho Connection đang mở. Tên CSDL mới được truyền cho phương thức này
	CreateCommand	Tạo một đối tượng Command ứng với Connection
	Leave	Xảy ra khi input focus rời khỏi TextBox
	TextChange	Diễn ra khi Text của TextBox bị thay đổi

Trong ADO.NET, việc thiết lập kết nối đến SQL Server và các nguồn dữ liệu OLE DB được thực hiện bằng các Connection class: SqlConnection và OleDbConnection.

ConnectionString

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, ConnectionString cho SQL Server Provider 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"
```

Trong đó:

- (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=localhost;database=qlsinhvien;uid=sa;pwd=system"  
"server=192.168.0.1;database=qlnhhanvien;uid=k28;pwd=spider"  
hoặc  
"Server=192.168.0.1;Database=qlnhhanvien;Trusted_Connection  
=yes"
```

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

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

3.3.2 Đối tượng Command

Sau khi tạo kết nối đến nguồn dữ liệu, bước tiếp theo trong truy xuất dữ liệu theo mô hình kết nối đó là tạo ra đối tượng Command để gửi một **query** (SELECT) hay một **action**

command (INSERT, UPDATE, DELETE) đến nguồn dữ liệu. Ứng với mỗi Data Provider sẽ có một Command class, các class này đều triển khai giao diện IDbCommand.

a/- Tạo đối tượng Command

Chúng ta có thể sử dụng các hàm khởi tạo để tạo đối tượng Command một cách trực tiếp hoặc có thể sử dụng cách tiếp cận Provider Factory.

Ví dụ 3.3: Đoạn code sau đây minh họa cách tạo ra một đối tượng SqlCommand và thiết lập các thuộc tính của nó:

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

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

Ví dụ 3.4:

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

Lưu ý: DbProviderFactory trong ví dụ trên được khai báo trong namespace System.Data.Common

b/- Thực thi Command

Lệnh SQL được gán trong thuộc tính CommandText của đối tượng Command sẽ được thực thi bằng một trong các phương thức sau:

ExecuteNonQuery: thực thi các action query và trả về số lượng mẫu tin bị ảnh hưởng bởi truy vấn

```
cmd.CommandText = "DELETE SinhVien WHERE MaSinhVien = 12";
int soLuong = cmd.ExecuteNonQuery();
```

ExecuteReader: thực thi một query và trả về đối tượng DataReader để có thể truy cập kết quả trả về. Phương thức nhận tham số tùy chọn kiểu CommandBehavior để có thể tăng hiệu năng thực thi query.

```
cmd.CommandText = "SELECT * FROM SinhVien " +
                  "WHERE YEAR(NgaySinh) > 1981";
SqlDataReader dr = cmd.ExecuteReader();
```

Đây được xem là phương thức quan trọng nhất của đối tượng Command. Tham số tuỳ chọn CommandBehavior cho phép chỉ định hành vi thực thi query. Một số data provider còn sử dụng CommandBehavior để tối ưu hóa quá trình thực thi query. Các giá trị của CommandBehavior có thể được chọn là:

- SingleRow: query chỉ trả về 1 mẫu tin. Mặc định sẽ trả về nhiều mẫu tin.
- SingleResult: query trả về một giá trị duy nhất (single scalar value).
- KeyInfo: trả về thông tin của cột (column) và khóa chính (primary key). Behavior này được sử dụng với phương thức GetSchema của DataReader để lấy thông tin về các cột trong lược đồ (schema).
- SchemaOnly: dùng để lấy về tên của các cột trong tập dữ liệu trả về

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

- SequentialAccess: cho phép dữ liệu trong dòng trả về có thể được truy xuất tuần tự theo cột. Behavior này được dùng cho các trường dữ liệu BLOB hay TEXT.

- CloseConnection: đóng connection khi DataReader được đóng.

ExecuteScalar: Thực thi một query và trả về giá trị của cột đầu tiên trong dòng đầu tiên của tập kết quả

```
cmd.CommandText = "SELECT COUNT(MaSinhVien) " +
                  "FROM SinhVien";
int soSinhVien = (int)cmd.ExecuteScalar();
```

ExecuteXmlReader: chỉ có ở SQL Server Provider. Trả về đối tượng XmlReader dùng để truy xuất tập dữ liệu. Tham khảo XmlReader từ <https://docs.microsoft.com/en-us/c-/C%C3%A1u-h%C3%ADnh-tham-s%C3%B3>

Các đối tượng Command sử dụng các tham số để truyền giá trị cho các lệnh SQL hoặc các thủ tục được lưu trữ (stored procedure), cung cấp việc kiểm tra và xác nhận tính hợp lệ. Không giống với văn bản lệnh, tham số đầu vào được xem như một giá trị chuỗi ký tự chứ không phải mã thực thi. Điều này tạo sự an toàn chống lại các cuộc tấn công “SQL injection”, trong đó kẻ tấn công sẽ chèn thêm một lệnh vào trong một câu lệnh SQL ảnh hưởng đến sự bảo mật trên máy chủ.

Các lệnh được tham số hóa cũng có thể cải tiến hiệu năng thực thi truy vấn, bởi chúng giúp các máy chủ CSDL so khớp chính xác lệnh đến với một kế hoạch truy vấn được lưu trữ đúng. Ngoài các lợi ích về bảo mật và hiệu năng, các lệnh tham số hóa cung cấp phương pháp thuận tiện cho việc tổ chức các giá trị chuyển đến nguồn dữ liệu.

Một đối tượng `DbParameter` có thể được tạo ra bằng hàm khởi tạo của nó và sau đó thêm nó vào `DbParameterCollection` bằng cách gọi phương thức `Add` của `DbParameterCollection`. Phương thức `Add` sẽ nhận input là đối số của hàm khởi tạo hoặc đối tượng tham số hiện có, tùy thuộc và data provider.

Trong SQL Server Provider, cú pháp đặt tên cho tham số là sử dụng tên tham số với tiền tố `@`, `@parametername`.

Ví dụ 3.5:

```
SqlConnection conn = new SqlConnection(strConn);
conn.Open();
string sql = "INSERT INTO Khoa (MaSo, TenKhoa, Viettat) VALUES
(@pMaSo, @pTenKhoa, @pViettat)";
SqlCommand cmd = new SqlCommand();
cmd.Connection = conn;
cmd.CommandText = sql;
cmd.Parameters.AddWithValue("@pMaSo", 04);
cmd.Parameters.AddWithValue("@pTenKhoa", "Công nghệ thông tin");
cmd.Parameters.AddWithValue("@pViettat", "CNTT");
```

Lưu ý, có thể sử dụng tham số trong việc thực thi Stored Procedure trong CSDL bằng đối tượng `Command`.

3.3.3 Đối tượng DataReader

a/- Truy xuất các mẫu tin với DataReader

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

Mặc dù `DataReader` ứng với một `Command` đơn nhưng, `Command` này lại có thể chứa nhiều query trong đó, do đó để có thể trả về nhiều tập dữ liệu kết quả. Ví dụ sau đây sẽ minh họa cách xử lý các dòng dữ liệu trả về bởi 2 query trong cùng một `Command`.

Ví dụ 3.6:

```
SqlConnection conn = new SqlConnection(strConn);
conn.Open();
string q1 = "SELECT * FROM SinhVien WHERE YEAR(NgaySinh) < 1981";
string q2 = "SELECT * FROM SinhVien WHERE YEAR(NgaySinh) > 1990";
SqlCommand cmd = new SqlCommand();
```

```

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

```

Lưu ý: Trong thực tế, giải pháp nối chuỗi ít khi được sử dụng vì lý do an toàn dữ liệu. Hãy xem xét chuỗi: `string q1 = "SELECT * FROM SinhVien WHERE HoTen='"+strHoTen+"'";` Điều gì sẽ xảy ra nếu strHoten được gán bằng `"tnv spider'; DELETE * FROM SinhVien;--"`. Khi đó lệnh SQL được thực thi sẽ là `SELECT * FROM SinhVien WHERE Hoten= 'tnv spider'; DELETE * FROM SinhVien;--`. Lỗi hỏng kiểu này thường được gọi tên là SQL Injection.

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 chỉ chuyển tiếp (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 một hay nhiều dòng để đọc hay không.

b/- Truy xuất giá trị của Column

Khi đọc dữ liệu bằng DataReader, có nhiều cách để truy xuất giá trị Column của mẫu tin hiện tại:

- Truy xuất như mảng, dùng chỉ số cột (bắt đầu từ 0) hoặc tên cột.
- Sử dụng phương thức GetValue bằng cách truyền cho phương thức này chỉ số cột.
- Sử dụng một trong các phương thức định kiểu như GetString, GetInt32, GetDateTime, GetDouble, ...

Ví dụ sau đây minh họa cách thức truy xuất giá trị dữ liệu của các Column.

Ví dụ 3.7:

```

cmd.CommandText = "SELECT MaSinhVien, Hoten, GioiTinh, " +
                  "NgaySinh FROM SinhVien " +
                  "WHERE YEAR(NgaySinh) = 1981";

dr = cmd.ExecuteReader();
dr.Read();
// Các cách để lấy dữ liệu kiểu string ở cột thứ 2 (Hoten)

```

```

string stHoTen;
stHoTen = dr.GetString(1);
stHoTen = (string)dr.GetSqlString(1); // SqlClient provider
stHoTen = (string)dr.GetValue(1);
stHoTen = (string)dr["HoTen"];
stHoTen = (string)dr[1];
// Lấy dữ liệu kiểu DateTime ở cột thứ 4 (NgaySinh) có kiểm
tra giá trị NULL
DateTime dtNgaySinh;
if (!dr.IsDBNull(3))
    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));

```

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:

```

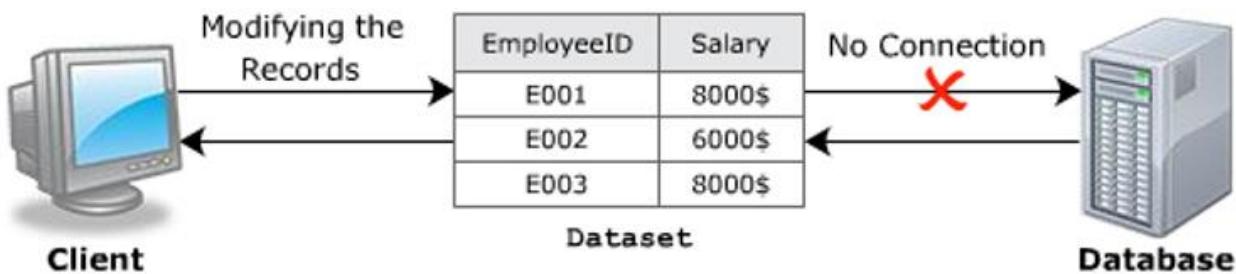
0 ColumnName: movie_ID
1 ColumnOrdinal: 0
... //không liệt kê
12 DataType: System.Int32
... //không liệt kê

```

3.4 LÀM VIỆC VỚI MÔ HÌNH NGẮT KẾT NỐI: DATASET VÀ DATATABLE

3.4.1 DataSet

Một đối tượng `DataSet` là một bản sao của nguồn dữ liệu thực trong bộ nhớ, hay một CSDL trong bộ nhớ. Các đối tượng được lưu trữ trong đó bao gồm các bảng dữ liệu có quan hệ với nhau và các ràng buộc. Một đối tượng `DataSet` không tương tác trực tiếp với nguồn dữ liệu, nó chỉ giữ dữ liệu nhận được từ nguồn dữ liệu. Chúng ta có thể nạp dữ liệu vào `DataSet` bằng `DataAdapter` và sau đó chỉnh sửa dữ liệu này.



Hình 3.8: DataSet

`DataSet` được tạo bằng câu lệnh:

```
DataSet <tên đối tượng> = new DataSet();
```

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 `Copy`, `Merge` và `Clear`.

`DataSet` và `DataTable` là phần lõi của ADO.NET và chúng không là đặc trưng của bất kỳ data provider nào (giống như ở các lớp `Connection`, `Command`, `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 quan hệ) vào `DataSet`.

a/- Đặc trưng của `DataSet`

Làm việc với dữ liệu không kết nối: khi dữ liệu đã được lấy vào `DataSet`, `DataSet` sẽ ngắt kết nối với CSDL. Sau đó, chúng ta có thể thực hiện các thay đổi trên dữ liệu trong `DataSet`. Những thay đổi này có thể được lưu ngược về CSDL bằng cách thiết lập lại kết nối với CSDL.

Làm việc với dữ liệu thứ bậc: các đối tượng DataSet lưu trữ dữ liệu theo kiến trúc thứ bậc. Nó cho phép dễ dàng làm việc với các bảng có quan hệ với nhau từ CSDL.

Thay đổi bộ nhớ đệm: các đối tượng DataSet cho phép chúng ta tạo ra các thay đổi trên một dòng dữ liệu ngay cả khi nó đã ngắt kết nối với CSDL. Các đối tượng DataSet có chứa nhiều phiên bản khác nhau của mỗi dòng trong bộ nhớ đệm. Và các dòng được lưu vào sau cùng sẽ được cập nhật vào CSDL thông qua DataAdapter, khi kết nối được thiết lập lại.

Hỗ trợ tích hợp XML: chúng ta cũng có thể nhận dữ liệu liệu từ một tập tin XML vào DataSet và cập nhật dữ liệu XML từ DataSet vào tập tin XML. Chúng ta cũng có thể tách thông tin bảng, cột và ràng buộc lưu nó vào một giản đồ XML (XML schema).

Thực hiện chức năng thống nhất: đối tượng DataSet cung cấp các chức năng nhất quán không phân biệt kiểu của dữ liệu nguồn và DataAdapter được dùng.

b/- *DataTable*

DataTable được xem là đối tượng trọng tâm của ADO.NET. các đối tượng khác sử dụng DataTable bao gồm DataSet và DataView.

Một DataSet có thể chứa hai đối tượng DataTable có cùng giá trị cho thuộc tính TableName nhưng khác giá trị của thuộc tính Namespace.

Các DataTable trong DataSet mô phỏng các bảng trong CSDL quan hệ (bao gồm các dòng (row) và các cột (column)...). Các thuộc tính quan trọng nhất của lớp DataTable là Columns và Rows định nghĩa cấu trúc và nội dung của bảng dữ liệu.

Trong lập trình, để tạo một DataTable, đầu tiên chúng ta phải định nghĩa lược đồ của nó bằng cách thêm các đối tượng DataColumn vào DataColumnCollection (được truy xuất thông qua thuộc tính Columns của DataTable).

Để thêm các dòng vào DataTable, đầu tiên chúng ta phải sử dụng phương thức NewRow để trả về đối tượng DataRow mới. Phương thức NewRow trả về một dòng theo lược đồ của DataTable, khi nó được định nghĩa bởi DataRowCollection của table. Số dòng tối đa DataTable có thể lưu trữ là 16.777.216.

DataTable cũng chứa tập các đối tượng Constraint được dùng để đảm bảo sự toàn vẹn của dữ liệu.

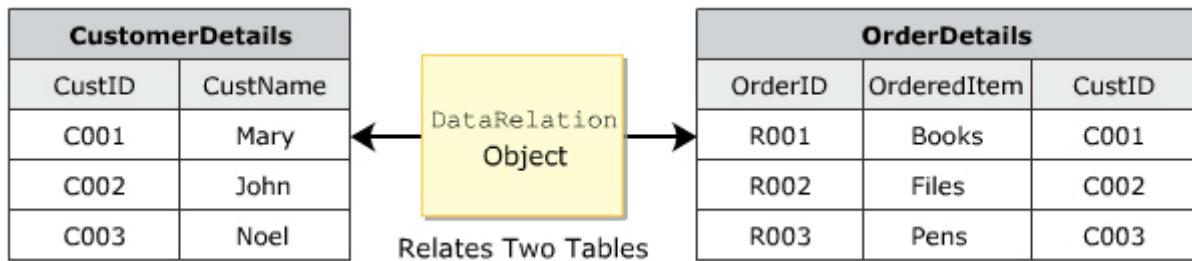
Có nhiều sự kiện của DataTable được dùng để nhận biết các thay đổi được tạo ra trên bảng, bao gồm: RowChanged, RowChanging, RowDeleting và RowDeleted.

Khi một thể hiện của DataTable được tạo, một vài thuộc tính read/write được thiết lập giá trị khởi tạo.

c/- Thiết lập mối quan hệ giữa các bảng

Mỗi quan hệ cha con giữa hai hay nhiều bảng trong DataSet được biểu diễn với một đối tượng DataRelation cho phép định hướng giữa các DataTable trong DataSet. Nó cung cấp các mẫu tin liên quan hiện đang xử lý.

Đối tượng DataRelation duy trì sự toàn vẹn tham chiếu bằng cách tuân theo ràng buộc khóa ngoại. Ngoài ra, đối tượng DataRelation còn tuân theo ràng buộc duy nhất để đảm bảo không có sự trùng lặp trong một cột của bảng. Nó cũng hữu ích cho việc thực hiện các hoạt động update và delete cascade.



Hình 3.9: Quan hệ giữa các bảng

Để thêm quan hệ giữa hai bảng, chúng ta phải chỉ định cột chung tồn tại trong các bảng. Sau đó, gọi phương thức Add() để thêm quan hệ giữa hai bảng. Những thuộc tính của cột được sử dụng cho việc lấy ràng buộc khóa ngoại và ràng buộc khóa duy nhất.

d/- DataColumn

Thuộc tính Columns của DataTable chứa một tập các đối tượng DataColumn biểu diễn các trường dữ liệu của DataTable.

Lớp DataColumn được dùng để tạo ra lược đồ cho một cột cụ thể, được xem là thành phần cơ bản của DataTable. Chúng ta có thể thêm các đối tượng DataColumn vào DataColumnCollection để tạo ra lược đồ của DataTable.

Các thuộc tính của DataColumn cho phép nhận kiểu dữ liệu của cột và thiết lập độ dài lớn nhất của cột. Bảng sau đây tóm tắt các thuộc tính quan trọng của lớp DataColumn.

Các cột của DataTable được tạo ra một cách tự động khi bảng được nạp dữ liệu từ kết quả của một truy vấn CSDL hoặc từ kết quả đọc đường ở một tập tin XML. Tuy nhiên, chúng ta cũng có thể viết code để tạo các cột.

Bảng 3.3: Thuộc tính của DataColumn

Thuộc tính	Mô tả
ColumnName	Tên cột
DataType	Kiểu của dữ liệu chứa trong cột Ví dụ: col1.DataType=System.Type.GetType("System.String");
MaxLength	Độ dài tối đa của một cột văn bản, -1 nếu không xác định độ dài tối đa.
ReadOnly	Cho biết giá trị của cột có được phép chỉnh sửa hay không

AllowDBNull	Giá trị Boolean cho biết cột này có được chứa giá trị NULL hay không
Unique	Giá trị Boolean cho biết cột này có được chứa 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 cột.

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.

Ví dụ 3.8:

```
DataTable tb = new DataTable("DonHang");
DataColumn dCol = new DataColumn("MaSo",
Type.GetType("System.Int16"));
// Dữ liệu của các dòng không được trùng nhau
dCol.Unique = true;
dCol.AllowDBNull = false;
tb.Columns.Add(dCol);
dCol = new DataColumn("DonGia",
Type.GetType("System.Decimal"));
tb.Columns.Add(dCol);
dCol = new DataColumn("SoLuong",
Type.GetType("System.Int16"));
tb.Columns.Add(dCol);
dCol = new DataColumn("ThanhTien",
Type.GetType("System.Decimal"));
dCol.Expression = "SoLuong*DonGia";
tb.Columns.Add(dCol);

// Liệt kê danh sách các Column trong DataTable
foreach (DataColumn dc in tb.Columns)
{
    Console.WriteLine(dc.ColumnName);
    Console.WriteLine(dc.DataType.ToString());
}
```

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

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

Ví dụ 3.9:

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

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

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

e/- DataRow

Lớp DataRow biểu diễn một dòng trong DataTable. Đối tượng của lớp được xem là thành phần cơ bản của DataTable. Lớp được sử dụng cho việc insert, update và delete các giá trị trong DataTable. Chúng ta có thể dùng các thuộc tính và phương thức của DataRow để chỉ định hoặc nhận dữ liệu của dòng và delete một dòng.

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

Ví dụ 3.10:

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

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

Ví dụ 3.11:

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

```
Console.WriteLine(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:

Ví dụ 3.12:

```
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

f/- DataView

DataView đóng vai trò như tầng hiển thị dữ liệu lưu trữ trong DataTable. DataView được dùng cho việc sắp xếp, lọc, tìm kiếm, điều chỉnh và điều hướng thông qua các mẫu tin của một table. Những chức năng này được thực hiện bằng cách đưa đối tượng DataTable như một tham số của hàm khởi tạo DataView. Đối tượng DataTable liên kết đến table được yêu cầu trong cơ sở dữ liệu. Kỹ thuật liên kết đối tượng đến bảng trong cơ sở dữ liệu như thế được gọi là data binding.

Lớp DataView có thể lấy và sửa đổi những hàng được chỉ định từ DataTable. Vì vậy, nó hoạt động như một tập con của DataTable. Vì vậy, chúng ta có thể làm việc đồng thời với hai phiên bản khác nhau của cùng một table.

Lớp DataView thuộc namespace System.Data.

Employee Table in the DataTable		
Name	Department	Monthly Salary
John Fernandes	HR	2000\$
Scully George	Marketing	900\$
Andrew Scott	Production	1000\$
Daniel George	Marketing	2500\$

Employee Table Filtered Using the DataView Class		
Name	Department	Monthly Salary
Scully George	Marketing	900\$
Daniel George	Marketing	2500\$

Ví dụ 3.13:

```
//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";
```

g/- Các bước diễn hình cho việc tạo và nạp dữ liệu cho DataSet

B1: Xây dựng và làm đầy mỗi DataTable trong DataSet bằng dữ liệu từ nguồn dữ liệu thông qua DataAdapter hoặc DataReader.

B2: Thay đổi dữ liệu trong các đối tượng DataTable bằng cách thêm, cập nhật hoặc xóa các đối tượng DataRow.

B3: Gọi phương thức GetChanges để tạo DataSet thứ hai chỉ có thay đổi với dữ liệu.

B4: Gọi phương thức Update của DataAdapter, truyền DataSet thứ hai như một đối số.

B5: Gọi phương thức Merge trộn những thay đổi từ DataSet thứ hai vào bản đầu tiên.

B6: Gọi AcceptChange trên DataSet để chấp nhận những thay đổi, hoặc gọi RejectChanges để hủy những thay đổi.

3.4.2 Nạp dữ liệu vào DataSet

Trong phần này sẽ trình bày cách nạp dữ liệu cho các DataTable của DataSet với các bảng dữ liệu từ nguồn dữ liệu quan hệ đã có.

a/- Nạp dữ liệu đọc từ DataReader 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).

Ví dụ 3.14:

```
SqlCommand cmd = new SqlCommand();
cmd.CommandText = "SELECT * FROM nhanvien";
CommandBehavior closeBeh = CommandBehavior.CloseConnection;
SqlDataReader dr = cmd.ExecuteReader(closeBeh);
DataTable dt = new DataTable("nhanvien");
dt.Load(dr); // Nạp dữ liệu và lược đồ vào table
Console.WriteLine(dr.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 enum LoadOption gồm 1 trong 3 giá trị OverwriteChanges, PreserveChanges, Upsert tương ứng với tùy chọn ghi đè những thay đổi, cập nhật giá trị nguyên gốc không thay đổi giá trị hiện tại (mặc định), hoặc cập nhật các giá trị hiện tại không thay đổi giá trị nguyên gốc. Đoạn code dưới đây minh họa cách trộn dữ liệu vào các dòng hiện tại theo kiểu ghi đè các giá trị hiện tại:

Ví dụ 3.15:

```
SqlConnection conn = new SqlConnection(chuoiKetNoi);
conn.Open();
string sql = "SELECT * FROM nhanvien WHERE diachi='a'";
SqlCommand cmd = new SqlCommand(sql, conn);
SqlDataReader dr = cmd.ExecuteReader();
DataTable dt = new DataTable("nhanvien");
dt.Load(dr);
Console.WriteLine(dt.Rows[0]["HoTen"]); // giả sử giá trị nhận
được là "tnv spider"
// Gán khóa chính
DataColumn[] col = new DataColumn[1];
col[0] = dt.Columns["Manv"];
```

```

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

```

b/- Nạp dữ liệu vào DataSet sử dụng 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”:

Ví dụ 3.16:

```

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

```

Với một table đang tồn tại, tác dụng của lệnh Fill tùy thuộc vào table có primary key 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.

3.4.3 Cập nhật cơ sở dữ liệu bằng DataAdapter

Sau khi dữ liệu được nạp vào DataSet bằng DataAdapter, kết nối giữa DataSet và nguồn dữ liệu sẽ bị đóng, những thay đổi sau đó tạo ra cho dữ liệu sẽ chỉ có tác động 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 đến nguồn dữ liệu.

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 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 action 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.

a/- 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 tạo 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:

Ví dụ 3.17:

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

```

int nUpdate = da.Update(ds, "khoa");
MessageBox.Show("Số dòng được thay đổi: " +
    nUpdate.ToString()); // 3

```

Có một số hạn chế khi sử dụng CommandBuilder: lệnh 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 lệnh Select ban đầu.

b/- Đồng bộ hóa dữ liệu giữa DataSet và CSDL

Việc sử dụng DataAdapter làm đơn giản hóa và tự động hóa quá trình cập nhật CSDL hoặc bất kỳ nguồn dữ liệu 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 nguồn dữ liệu không bị khóa (lock) giữa thời gian mà chúng được đọc và thời gian mà các cập nhật được áp dụng cho nguồn dữ liệu. Trong khoảng thời gian này, user khác có thể cũng cập nhật nguồn dữ liệu. Nếu có thay đổi xảy ra kể từ lần đọc trước đó thì phương thức Update sẽ nhận biết được và không cho áp dụng thay đổi đối với các dòng dữ liệu đó.

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

c/- Sử dụng Transactions để Roll Back nhiều cập nhật

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

Ví dụ 3.18:

```

SqlDataAdapter da = new SqlDataAdapter();
SqlCommandBuilder sb = new SqlCommandBuilder(da);
SqlTransaction tran;
SqlConnection conn = new SqlConnection(chuoiKetNoi);

```

```
conn.Open();
// Connection phải được dùng với Transaction
// (1) Tạo ra một transaction
tran = conn.BeginTransaction();
// (2) Gắn SelectCommand với transaction
da.SelectCommand = new SqlCommand(sql, conn, tran);
DataSet ds = new DataSet();
da.Fill(ds, "docgia");
//
// Code ở phần này thực hiện các cập nhật lên các dòng dữ
liệu từ DataSet
//
try
{
    int updates = da.Update(ds, "docgia");
    MessageBox.Show("Cập nhật: " + updates.ToString());
}
// (3) Nếu có ngoại lệ xuất hiện, roll back mọi cập nhật
trong transaction
catch (Exception ex)
{
    MessageBox.Show(ex.Message); // Lỗi khi cập nhật
    if (tran != null)
    {
        tran.Rollback(); // Roll back mọi cập nhật
        tran = null;
        MessageBox.Show("Tất cả cập nhật đã bị roll back.");
    }
}
finally
{
    // (4) Nếu không có lỗi, commit tất cả các cập nhật
    if (tran != null)
    {
        tran.Commit();
        MessageBox.Show("Tất cả đã được cập nhật thành công.
");
        tran = null;
    }
}
```

```
conn.Close();
```

d/- Xác định các dòng gây ra lỗi cập nhật

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

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

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

```

        r.RejectChanges();
        string iMadocgia = ((int)r["Madocgia"]).ToString();
        string msg = state + " Madocgia: " + iMadocgia;
        Console.WriteLine(msg);
    }
}
}

```

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

3.4.4 Định nghĩa các Relationship giữa các DataTable

Trong một DataSet với nhiều đối tượng DataTable, chúng ta có thể sử dụng các đối tượng DataRelation để liên kết từ một bảng đến một bảng khác hay điều hướng xuyên qua các bảng và để nhận về các hàng con hoặc cha từ một bảng quan hệ.

Đối số được yêu cầu để tạo một DataRelation là một cái tên cho nó, và một mảng của một hoặc nhiều DataColumn tham chiếu đến các cột được xem như các cột cha và con trong mối quan hệ (Relationship). Sau khi tạo DataRelation, chúng ta có thể sử dụng nó để điều hướng giữa các bảng và nhận các giá trị.

Cú pháp khởi tạo:

```

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

```

Việc thêm một DataRelation vào DataSet, mặc định sẽ thêm một ràng buộc duy nhất (UniqueConstraint) trên DataTable cha và một ràng buộc khóa ngoại (ForeignKeyConstraint) ở DataTable con. Ràng buộc khóa ngoại sẽ quyết định cách thức DataTable con bị ảnh hưởng khi các dòng dữ liệu ở phía DataTable cha bị thay đổi hay bị xóa (giống với CSDL thực tế).

Thuộc tính Relations của DataSet giúp quản lý tập các DataRelation đã được định nghĩa trong DataSet. Phương thức Add của thuộc tính này được sử dụng để thêm các DataRelation vào tập Relations.

Ví dụ 3.19: thiết lập mới quan hệ giữa 2 bảng KHOA và DOCGIA để có thể liệt kê danh sách các độc giả của mỗi khoa

```

string connStr = "Data Source=tên máy chủ;Initial Catalog=quanlythuvien; Trusted_Connection = yes";
DataSet ds = new DataSet();
// (1) Fill bảng docgia

```

```

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

```

3.4.5 Thao tác dữ liệu với DataGridView

a/- Điều khiển DataGridView

Điều khiển DataGridView hiển thị dữ liệu theo dạng tabular. Nó hoạt động như một giao diện của đối tượng DataSet và được sử dụng để hiển thị dữ liệu của một table được yêu cầu từ cơ sở dữ liệu. Nó cũng được dùng cho việc insert, update, và delete dữ liệu từ table trong cơ sở dữ liệu. Điều khiển có thể hiển thị dữ liệu của một bảng đơn, nhiều bảng và các bảng có quan hệ với nhau.

Ví dụ 3.20:

```

SqlDataAdapter sqldaOrders= new SqlDataAdapter("SELECT * FROM Orders",
sqlconOrders);
DataSet dsetOrders = new DataSet("Orders");
sqldaOrders.Fill(dsetOrders, "Orders");
DataGridView dgridOrders = new DataGridView();
this.Controls.Add(dgridOrders);
dgridOrders.DataSource = dsetOrders;
dgridOrders.Dock=DockStyle.Fill;

```

```
dgridOrders.DataMember = "Orders";
```

b/- Điều khiển DataGridView

Điều khiển DataGridView là một điều khiển phức tạp được sử dụng để hiển thị dữ liệu từ một bảng ở dạng tabular từ các loại nguồn dữ liệu khác nhau. Chúng ta có thể định nghĩa những hoạt động tùy chỉnh sử dụng điều khiển này dựa vào tính năng của nó:

- Cung cấp các kiểu cột khác nhau như cột với TextBox và CheckBox hiển thị trong DataGridView.

- Hỗ trợ nhiều nguồn dữ liệu để làm việc với các cơ sở dữ liệu khác nhau. Chúng ta có thể sử dụng điều khiển thậm chí nếu nó không có ràng buộc với bất cứ nguồn dữ liệu nào.

- Thực hiện tính năng chế độ ảo bằng cách cho phép điều khiển sử dụng bộ nhớ cache. Tính năng này cải thiện hiệu năng khi quản lý lượng lớn mẫu tin bởi cache và cho phép chúng ta thực hiện các hoạt động đã được tùy chỉnh.

- Sử dụng thành phần BindingSource để đơn giản hóa quá trình data binding. Điều này là bởi vì thành phần cung cấp tất cả các dịch vụ được yêu cầu cho data binding.

Khi điều khiển DataGridView không ràng buộc với nguồn dữ liệu, chế độ làm việc này được gọi là unbound mode. Trạng thái này thường được sử dụng để quản lý lượng dữ liệu nhỏ.

DataGridView được dùng trong các ngữ cảnh:

- Hiển thị lượng dữ liệu nhỏ với unbound mode
- Xem và cập nhật dữ liệu trong bảng của CSDL sử dụng các đối tượng ADO.NET như DataTable.

- Xử lý lượng dữ liệu lớn sử dụng chế độ ảo cho việc tối ưu hiệu năng.
- Tùy chỉnh các dòng, cột, ô và tiêu đề được cung cấp bởi tool tips và shortcut menu.

Ví dụ 3.21:

```
DataGridView dgvwDepartment = new DataGridView();
this.Controls.Add(dgvwDepartment);
dgvwDepartment.DataSource = dsetDepartment.Tables[1];
dgvwDepartment.AutoGenerateColumns = true;
dgvwDepartment.Columns.Add("dcolName", "Name");
dgvwDepartment.Columns.Add("dcolAge", "Age");
dgvwDepartment.Rows.Add("Harry");
dgvwDepartment.Rows[0].Cells[1].Value= "20";
```

c/- So sánh giữa DataGrid và DataGridView

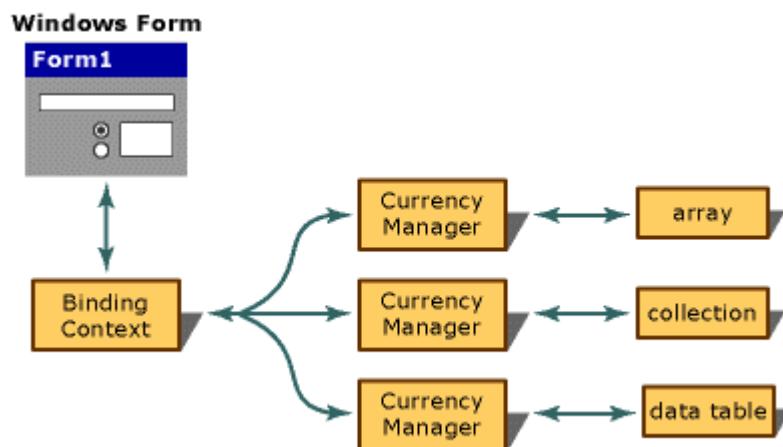
Điều khiển DataGridView mở rộng chức năng của điều khiển DataGrid bằng cách cung cấp các tính năng mới.

DataGridView	DataGrid
Cung cấp nhiều kiểu column dựng sẵn	Cung cấp số lượng kiểu column hạn chế
Hiển thị dữ liệu từ nguồn dữ liệu có ràng buộc và không ràng buộc	Chỉ hiển thị dữ liệu như một sự mở rộng của nguồn dữ liệu
Hỗ trợ nhiều phương pháp để hiển thị và định dạng dữ liệu. Ví dụ, chúng ta có thể thiết lập cách thức hiển thị cho một số ô có loại dữ liệu tương tự nhau	Hỗ trợ hạn chế định dạng dữ liệu
Cung cấp nhiều tùy chọn cho việc thay đổi vẻ ngoài của các ô, dòng, cột và tiêu đề. Ví dụ, chúng ta có thể thay đổi kích thước của chúng, cung cấp tool tips và shortcut menu.	Hỗ trợ hạn chế định dạng cho các ô, dòng, cột và tiêu đề.
Không hỗ trợ hiển thị dạng cấp bậc của dữ liệu từ các bảng	Hỗ trợ hiển thị dữ liệu dạng cấp bậc của dữ liệu từ hai bảng có quan hệ với nhau.

3.5 SỬ DỤNG DATABINDING

3.5.1 DataBinding là gì?

DataBinding là một trong những tính năng mạnh mẽ của .NET Windows Forms. DataBinding cung cấp cách thức cho các nhà phát triển tạo liên kết đọc/ghi giữa các điều khiển trong form và dữ liệu trong ứng dụng của họ (buộc điều khiển với dữ liệu). Về mặt cổ điển, buộc dữ liệu được dùng trong các ứng dụng để tận dụng lợi thế của dữ liệu được lưu trữ trong cơ sở dữ liệu. Nó có ích cho việc xem, thêm mới, cập nhật, tìm kiếm và xóa các mẩu tin trong bảng dữ liệu. Các mẩu tin có thể được lấy từ một bảng đơn hoặc từ nhiều bảng (qua các khung nhìn dữ liệu (View)). Windows Forms DataBinding cho phép chúng ta truy xuất dữ liệu từ các cơ sở dữ liệu cũng như các cấu trúc dữ liệu khác như mảng (array) hay tập hợp (collection).



Hình 3.10: DataBinding trong ứng dụng Windows Forms

3.5.2 Các cách buộc dữ liệu

ADO.NET cung cấp 2 loại Binding:

- **Buộc dữ liệu đơn giản (simple data binding):** Tại một thời điểm, một giá trị đơn trong DataSet có thể bị buộc vào một điều khiển bất kỳ.

Ví dụ 3.22: giả sử đã có 1 DataSet ds chứa dữ liệu của bảng KHOA, cần buộc tên khoa vào TextBox txttenkhoa:

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

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

- **Buộc dữ liệu phức tạp (complex data binding):** Tập dữ liệu chứa trong DataTable hoặc DataView của DataSet được buộc vào điều khiển thay vì chỉ một giá trị đơn (DataColumn). Các điều khiển như DataGridView và ComboBox hỗ trợ *buộc dữ liệu phức tạp*.

Ví dụ 3.23: giả sử đã có 1 DataSet ds chứa dữ liệu của bảng KHOA, cần buộc tên khoa vào ComboBox cmbkhoa và buộc toàn bộ dữ liệu của bảng Khoa vào DataGridView:

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

3.5.3 Các nguồn dữ liệu của DataBinding

Như đã trình bày ở trên, trong Windows Forms, chúng ta có thể *buộc dữ liệu (bind)* với nhiều kiểu cấu trúc dữ liệu khác nhau, từ đơn giản (mảng) đến phức tạp (các DataColumn, DataView,...). Nói một cách đơn giản, các cấu trúc có thể *buộc dữ liệu* phải hỗ trợ *IList Interface*. Do các cấu trúc này cung cấp nhiều tính năng hơn mà chúng ta có thể tận dụng khi buộc dữ liệu.

a/- Mảng (array) hoặc tập hợp (collection)

```
int[] t = new int[4] { 12, 2, 3, 4 };
//buộc dữ liệu đơn giản
txtNumber.DataBindings.Add("Text", t, "");
//buộc dữ liệu phức tạp
cmbNumber.DataSource = t;
```

b/- Các đối tượng dữ liệu ADO.NET

ADO.NET cung cấp một số cấu trúc dữ liệu phù hợp DataBinding:

DataColumn: Mỗi đối tượng DataColumn có một thuộc tính DataType xác định kiểu dữ liệu mà cột chứa. Chúng ta có thể thực hiện *buộc dữ liệu đơn giản* một điều khiển (như thuộc tính Text của TextBox) với một cột trong bảng dữ liệu như sau:

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

DataTable: Một đối tượng DataTable có 2 tập: DataColumn (biểu diễn các cột dữ liệu trong bảng) và DataRow (biểu diễn các dòng dữ liệu trong bảng). Chúng ta có thể thực hiện *buộc dữ liệu* một điều khiển đến thông tin được chứa trong bảng dữ liệu. Tuy nhiên, khi chúng ta *buộc dữ liệu* đến DataTable, thực tế là chúng ta đang *buộc dữ liệu* với khung nhìn mặc định (default View) của bảng dữ liệu.

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

DataView: Một đối tượng DataView là một khung nhìn tùy chỉnh của một bảng dữ liệu đơn hoặc dữ liệu được lấy từ nhiều bảng có thể lọc và sắp xếp. Một DataView là một “snapshot” được dùng bởi các điều khiển được *buộc dữ liệu phức tạp*. Chúng ta có thể thực hiện các *buộc dữ liệu đơn giản hoặc phức tạp* với dữ liệu trong một DataView. Tuy nhiên, hãy lưu ý là chúng ta đang thực hiện các việc này trên một “ảnh” của dữ liệu hơn là trên một nguồn dữ liệu cập nhật và rõ ràng.

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

- **DataSet:** như trong ví dụ 3.23.

3.5.4 Ngữ cảnh buộc dữ liệu (BindingContext)

Mỗi Windows Form có ít nhất một đối tượng BindingContext quản lý các đối tượng CurrencyManager của Form. Tương ứng với mỗi nguồn dữ liệu trên Form sẽ có một đối tượng CurrencyManager riêng biệt. Bởi có nhiều nguồn dữ liệu kết nối với Form nên đối tượng BindingContext sẽ cho phép chúng ta lấy một CurrencyManager riêng biệt bất kỳ kết nối với một nguồn dữ liệu.

Ví dụ, nếu chúng ta thêm một TextBox vào Form và buộc nó với một cột của một bảng dữ liệu (ví dụ: `khoa.tenkhoa`) trong một DataSet (ví dụ: `ds`), TextBox sẽ giao tiếp

với BindingContext của Form. Sau đó, đến lượt của BindingContext, nó sẽ “nói” với một đối tượng CurrencyManager cụ thể của kết nối dữ liệu đó. Nếu chúng ta truy vấn thuộc tính Position của CurrencyManager, nó sẽ báo cáo mẫu tin hiện thời của dữ liệu được buộc vào TextBox.

```
// Buộc dữ liệu đơn giản
textBox.DataBindings.Add("Text", ds, "khoa.tenkhoa");
// Lấy vị trí hiện tại
CurrencyManager cm =
(CurrencyManager) this.BindingContext[dsCust, "khoa"];
long rowPosition = (long) cm.Position;
```

CurrencyManager và PropertyManager

CurrencyManager được sử dụng để giữ các điều khiển đã buộc dữ liệu được đồng bộ với nhau (hiển thị dữ liệu của cùng mẫu tin). Để có thể làm được điều đó, CurrencyManager quản lý tập các dữ liệu liên kết được cung cấp bởi nguồn dữ liệu. Với mỗi nguồn dữ liệu được kết nối với Form, Form sẽ duy trì ít nhất một CurrencyManager. Bởi có thể có nhiều hơn một nguồn dữ liệu được kết nối với Form, như đã trình bày ở trên, đối tượng BindingContext sẽ quản lý tất cả các đối tượng CurrencyManager của một Form cụ thể. Nói rộng hơn, tất cả các điều khiển chứa đựng (container controls) có ít nhất một đối tượng BindingContext để quản lý các CurrencyManager của chúng.

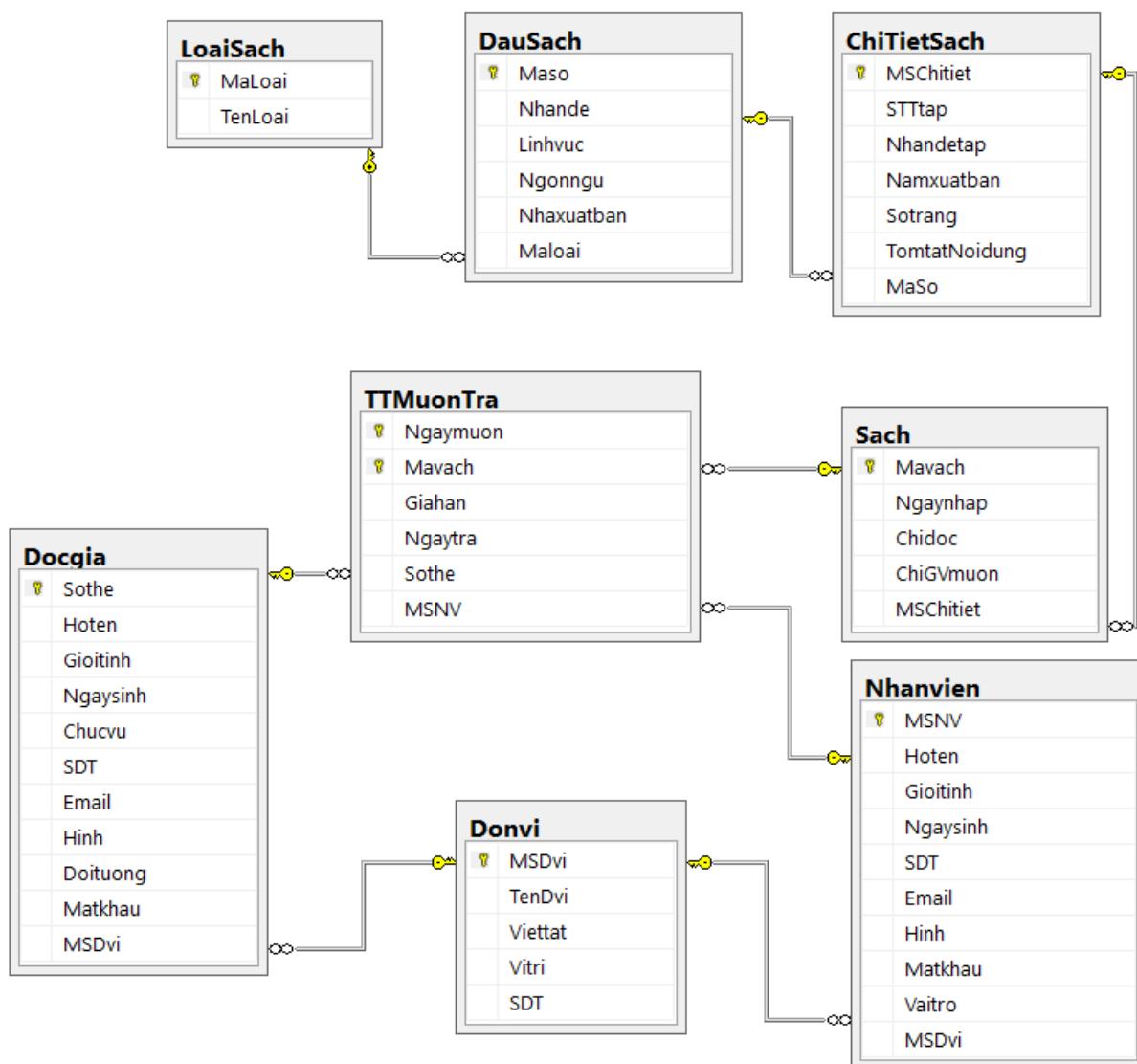
Một thuộc tính quan trọng của CurrencyManager đó là Position. Currency là một thuật ngữ được dùng để chỉ tính hiện tại của vị trí trong cấu trúc dữ liệu. Chúng ta có thể sử dụng thuộc tính Position của CurrencyManager class để xác định vị trí hiện tại của tất cả các điều khiển đã buộc dữ liệu với cùng một CurrencyManager.

Ví dụ, hãy tượng tượng một tập hợp bao gồm hai cột “Madocgia” và “HoTen”. Hai điều khiển TextBox được buộc vào cùng một nguồn dữ liệu. Khi thuộc tính Position của CurrencyManager chung được đặt ở vị trí thứ tư trong danh sách (tương đương với độc giả thứ năm, bởi nó bắt đầu từ 0), cả hai điều khiển hiển thị giá trị tương ứng (“Madocgia” thứ năm, và “HoTen” thứ năm) cho vị trí đó trong nguồn dữ liệu.

Lưu ý: trong trường hợp nếu nguồn dữ liệu chỉ trả về một giá trị đơn khi đó PropertyManager sẽ được lưu trữ bên trong BindingContext thay cho việc sử dụng một CurrencyManager.

Bảng 3.4: Các thuộc tính của CurrencyManager

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

CÂU HỎI VÀ BÀI TẬP**PHẦN 1: XÂY DỰNG ỨNG DỤNG MDI: QUẢN LÝ THƯ VIỆN VLUTE**

Giả sử chúng ta đã có cơ sở dữ liệu QLThuvienVLUTE trong SQL Server có quan hệ như sơ đồ trên.

1. Xây dựng frmMain đóng vai trò là MDI Parent Form, có các menu cho phép liên kết đến các chức năng của hệ thống như sau:

Hệ thống: hỗ trợ các chức năng đăng nhập, đăng xuất, thoát ứng dụng, quản lý tài khoản người dùng sau khi đăng nhập.

Quản lý sách: quản lý các thông tin có liên quan đến sách (đầu sách, chi tiết sách, sách) đồng thời cho phép tất cả người dùng tra cứu thông tin sách có trong thư viện.

Quản lý độc giả: cho phép quản lý thông tin có liên quan đến độc giả (độc giả, đơn vị), cũng như thông tin mượn trả sách của độc giả tại thư viện.

Quản trị hệ thống: quản lý các nhóm người dùng của hệ thống (giảng viên, sinh viên, nhân viên thư viện) và các người dùng của từng nhóm cụ thể.

Thống kê-Báo cáo: phục vụ các chức năng như thống kê danh sách độc giả, tình hình mượn trả sách của độc giả; danh sách nhân viên; báo cáo danh mục sách với số lượt mượn trả tương ứng theo tháng, quý, năm.

Tìm kiếm: cho phép người dùng tìm các thông tin có liên quan đến sách, độc giả, nhân viên... (lưu ý, cần bảo mật các thông tin cá nhân hoặc các thông tin mật khác).

Lưu ý, cần đảm bảo phân quyền chức năng hệ thống cho từng nhóm người dùng cụ thể sau khi đăng nhập, cũng như giới hạn các chức năng đối với nhóm người dùng chưa đăng nhập.

Gợi ý: gom nhóm các ToolStripItem theo từng nhóm người dùng, sau đó dựa vào người dùng đăng nhập sẽ cho hiển thị nhóm ToolStripItem tương ứng với nhóm người dùng của họ. Thuộc tính Doituong (0: giảng viên, 1: sinh viên) của bảng Docgia và bảng Nhanvien được sử dụng cho việc xác định nhóm người dùng.

2. Thiết kế các MDI Child Form đáp ứng từng chức năng cụ thể của hệ thống.
3. Xây dựng lớp Connection phục vụ các chức năng kết nối và truy xuất CSDL.

```
class Connection
{
    //properties
    private string strConn= "server=CANARY-
PC;database=quanlythuvien;uid=sa;pwd=system";
    public SqlConnection conn { get; set; }
    public SqlCommand cmd { get; set; }
    public SqlDataReader drd { get; set; }
    public SqlDataAdapter da { get; set; }
    //constructor
    public Connection()
    {
        conn = new SqlConnection(strConn);
        cmd = null;
        drd = null;
        da = null;
    }

    //methods
    //Mở kết nối đến nguồn dữ liệu
    public bool openConn()
    {
        try
        {
            conn.Open();
            return true;
        }
    }
}
```

```
        }
        catch (Exception)
        {
            return false;
        }
    }

//Đóng kết nối đến nguồn dữ liệu
public bool closeConn()
{
    try
    {
        conn.Close();
        return true;
    }
    catch (Exception)
    {
        return false;
    }
}

////Mô hình kết nối////
//Đỗ dữ liệu từ CSDL -> DataReader -> Client đọc
public SqlDataReader executeSQL(string sql)
{
    cmd = new SqlCommand(sql, conn);
    drd = cmd.ExecuteReader();
    return drd;
}

//Cập nhật dữ liệu theo mô hình kết nối
public int executeUpdate(string sql)
{
    cmd = new SqlCommand(sql, conn);
    return cmd.ExecuteNonQuery();
}

////Mô hình ngắt kết nối////
//Đỗ dữ liệu từ CSDL -> DataAdapter -> DataTable
(DataSet-Client)
public DataTable loadDataTable(string sql)
{
    cmd = new SqlCommand(sql, conn);
    da = new SqlDataAdapter(cmd);
    DataTable dt = new DataTable();
```

```

        da.Fill(dt);
        return dt;
    }
    //Cập nhật dữ liệu từ DataTable vào CSDL qua DataAdapter
    (tương đồng bảng)
    public void UpdateTable(DataTable dt)
    {
        SqlCommandBuilder scb = new SqlCommandBuilder(da);
        da.Update(dt);
    }

}

```

4. Xử lý tác vụ thao tác với cơ sở dữ liệu

B1: Tạo ra một đối tượng thuộc lớp connection

```
Connection ketnoi = new Connection();
```

B2: Mở kết nối

```
//gọi phương thức (method) openConn() của lớp connection
ketnoi.openConn();
```

B3: Xử lý

B3.1: Xử lý theo mô hình kết nối

2 phương thức:

Phương thức executeSQL(String sql): phục vụ cho việc truy vấn (query) dữ liệu thông qua các câu SELECT. (cmd.ExecuteReader(); thực hiện đọc dữ liệu vô DataReader)

Phương thức executeUpdate(String sql): phục vụ cho việc cập nhật dữ liệu (INSERT, UPDATE, DELETE). (cmd.ExecuteNonQuery(); thực hiện cập nhật dữ liệu dựa trên chuỗi sql của cmd)

```

String sql = "SELECT hoten FROM nhanvien";
SqlDataReader drd = ketnoi.executeSQL(sql);
String kq = "";
while (drd.Read()) //Đọc dữ liệu từ DataReader thông qua phương
thức Read
{
    kq += drd["hoten"].ToString() + ", ";
}

```

B3.2: Xử lý theo mô hình ngắn kết nối (load dữ liệu lên combobox hoặc datagridview)

Phương thức loadDataTable(String sql): nạp dữ liệu được lấy dựa trên câu lệnh truy vấn sql vào DataTable thông qua phương thức Fill của DataAdapter

```

//load combobox
String sql = "SELECT makhoa, makhoa + ' - ' + tenkhoa AS
hienthi FROM khoa";
DataTable dt = ketnoi.loadDataTable(sql);
cbxKhoa.DataSource = dt;
cbxKhoa.ValueMember = "makhoa";
cbxKhoa.DisplayMember = "hienthi";
//Load datagridview
sql = "SELECT * FROM khoa";
DataTable dt2 = ketnoi.loadDataTable(sql);
dgvDSKhoa.AutoGenerateColumns = false;
dgvDSKhoa.DataSource = dt2;

```

B4: Đóng kết nối

```
//gọi phương thức (method) closeConn() của lớp Connection
ketnoi.closeConn();
```

2. Lập trình thao tác dữ liệu và xử lý sự kiện cho các điều khiển trong các MDI Child Form đã thiết kế.

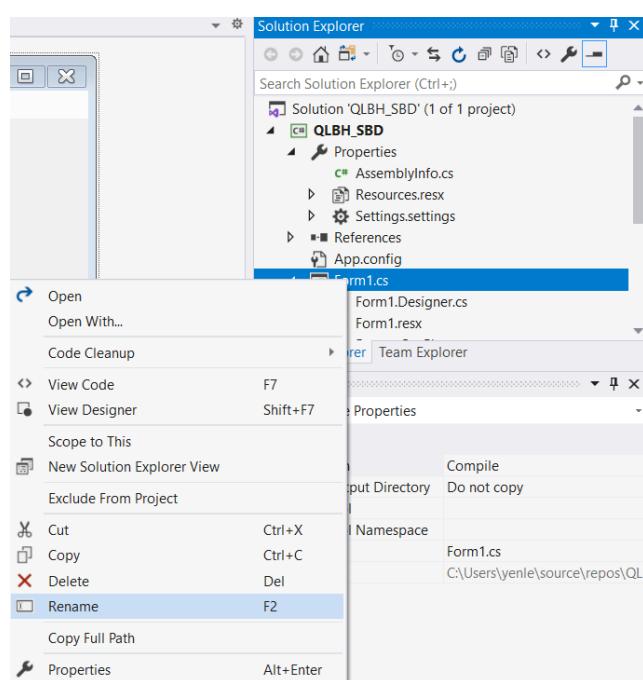
PHẦN 2: THAO TÁC VỚI DATASET VÀ DATABINDING

1. Thiết kế ứng dụng MDI

❖ Thiết kế MDI Parent Form

- (Tùy chọn) Trong cửa sổ Solution Explorer, **đổi tên Form mặc định** của project vừa tạo thành **frmMain**

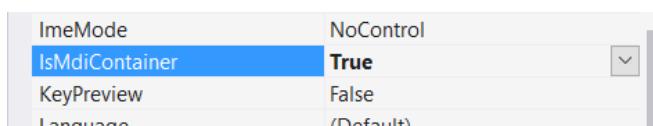
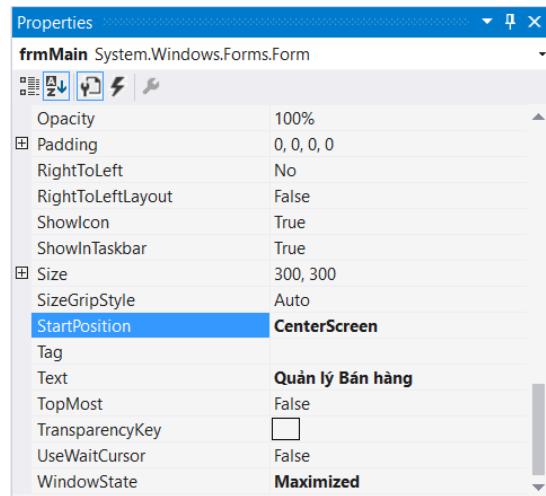
- + Right click trên Form1
- + Chọn Rename
- + Nhập tên mới là frmMain



- Trong cửa sổ **Properties** của **frmMain** lần lượt thay đổi giá trị cho các thuộc tính sau:

- + *Name*: **frmMain** (nếu không đổi tên form ở bước trên).
- + *StartPosition*: **CenterScreen**
- + *Text*: **Quản lý bán hàng**
- + *WindowState*: **Maximized**

- Định **frmMain** trở thành **MDI Parent Form** của ứng dụng MDI bằng cách thiết lập thuộc tính



IsMdiContainer của form bằng **True**.

- Điều chỉnh kích thước frmMain trong khung nhìn thiết kế để có thể dễ dàng thao tác với các điều khiển.

- Thêm **main menu** cho **frmMain**:

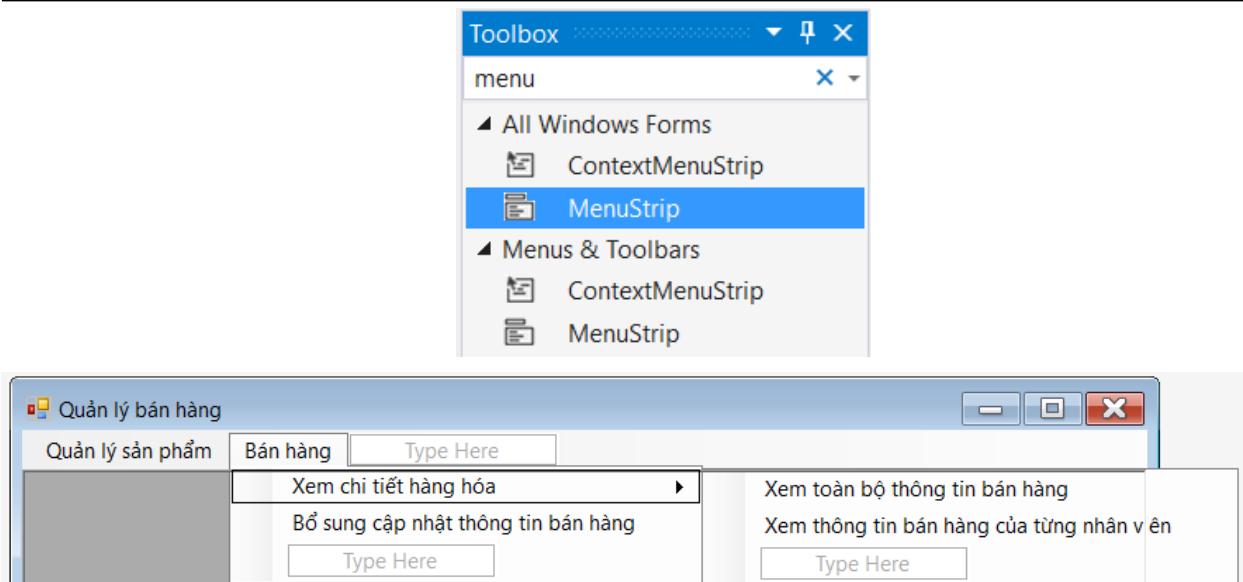
- + Trong trường **Search** của **Toolbox**, nhập từ khóa của điều khiển (control) cần tìm. Ở đây là từ khóa **menu**
- + Kéo thả điều khiển **MenuStrip** tìm được từ **Toolbox** vào **frmMain** để tạo **main menu** cho Form.
- + Lần lượt nhập **Text (Type Here)** và thiết lập lại giá trị cho thuộc tính **Name** trong cửa sổ **Properties** cho các **ToolStripMenuItem** với các mục chọn như sau:

Quản lý sản phẩm

- + *Xem danh sách sản phẩm*
- + *Bổ sung, cập nhật sản phẩm*

Bán hàng

- + *Xem chi tiết hàng bán*
 - * *Xem toàn bộ thông tin bán hàng*
 - * *Xem thông tin bán hàng của từng nhân viên.*
- + *Bổ sung, cập nhật thông tin bán hàng*



❖ Thiết kế các MDI Child Forms

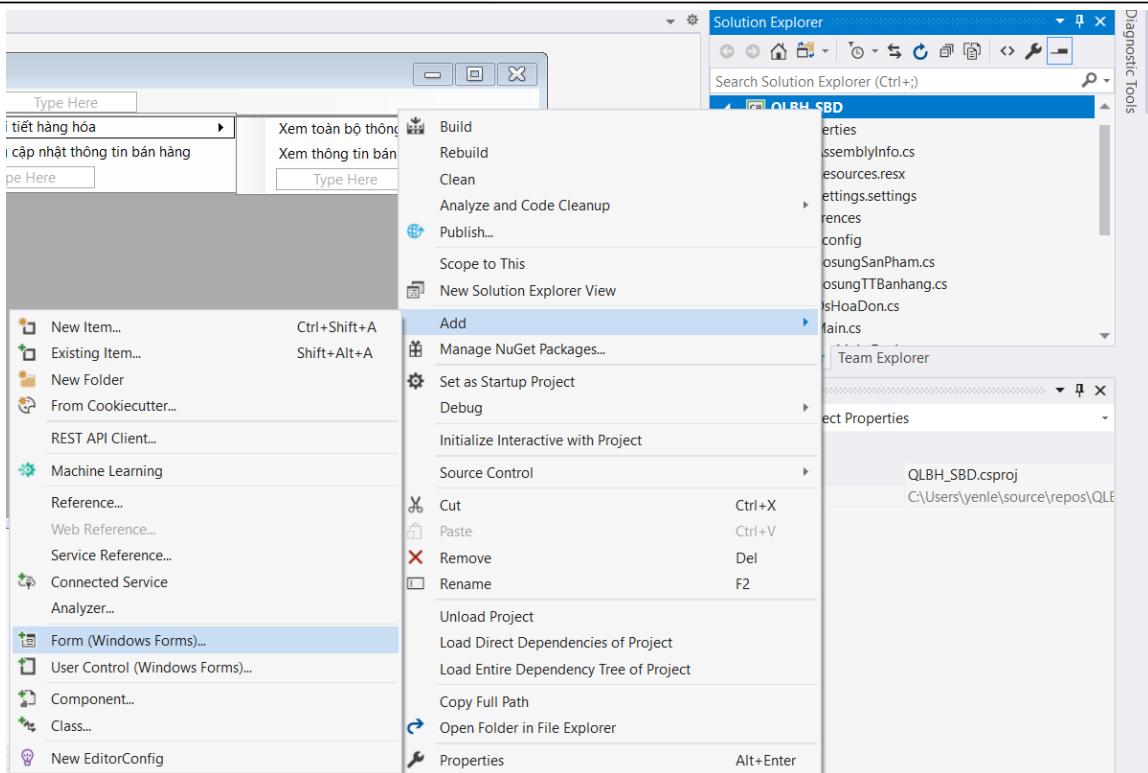
Thiết kế các form con theo từng chức năng trong menu. Yêu cầu:

- + Các form **xem danh sách sản phẩm** và **xem toàn bộ thông tin bán hàng** có các button điều khiển di chuyển bắn ghi như *First, Last, Previous, Next*.
- + Các form **bổ sung cập nhật sản phẩm** và **bổ sung cập nhật thông tin bán hàng** sử dụng công cụ *Binding Navigator* để thao tác trực tiếp trên lưới dữ liệu.
- + Form **xem thông tin bán hàng của từng nhân viên** cho phép chọn tên nhân viên trong một combobox và hiển thị thông tin về các lần bán hàng của nhân viên đó.

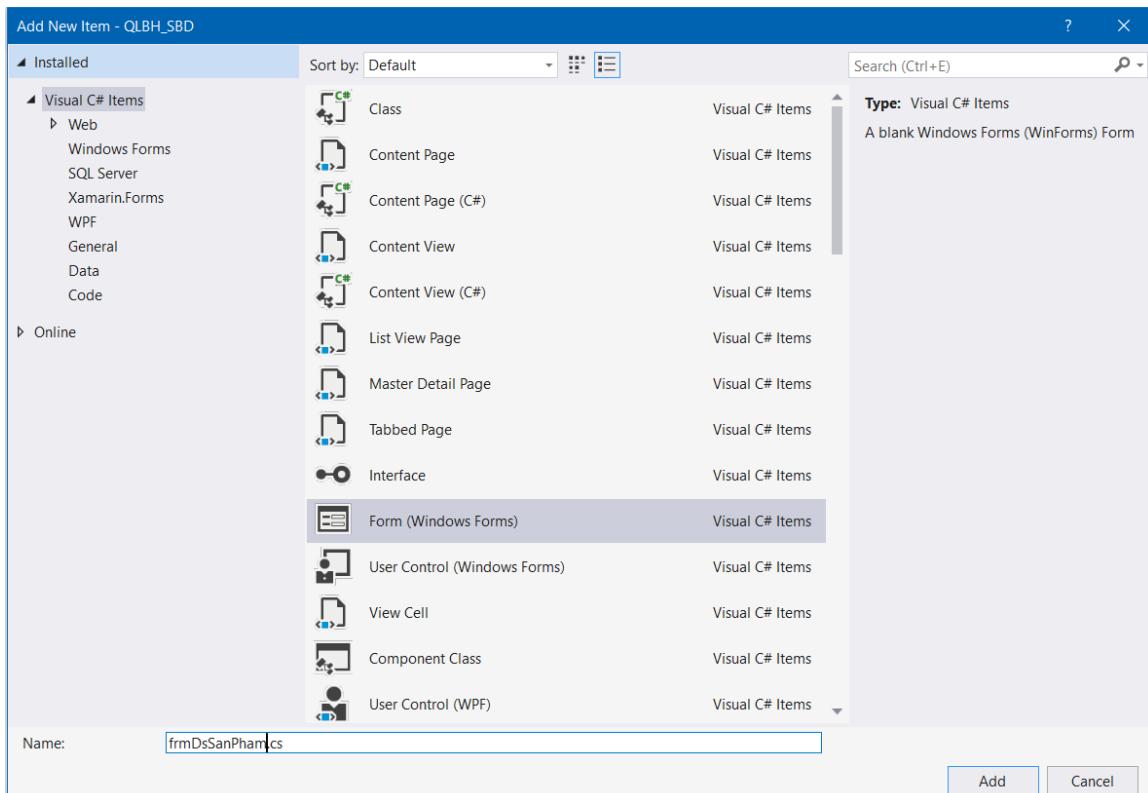
Lần lượt thiết kế các **MDI Child Forms** tương ứng với các chức năng cần xử lý. Ví dụ như thiết kế **frmDsSanPham**

- Thêm form mới vào dự án đã tạo:

- + Right click trên dự án trong cửa sổ Solution Explorer
- + Chọn Add → Form (Windows Forms)...



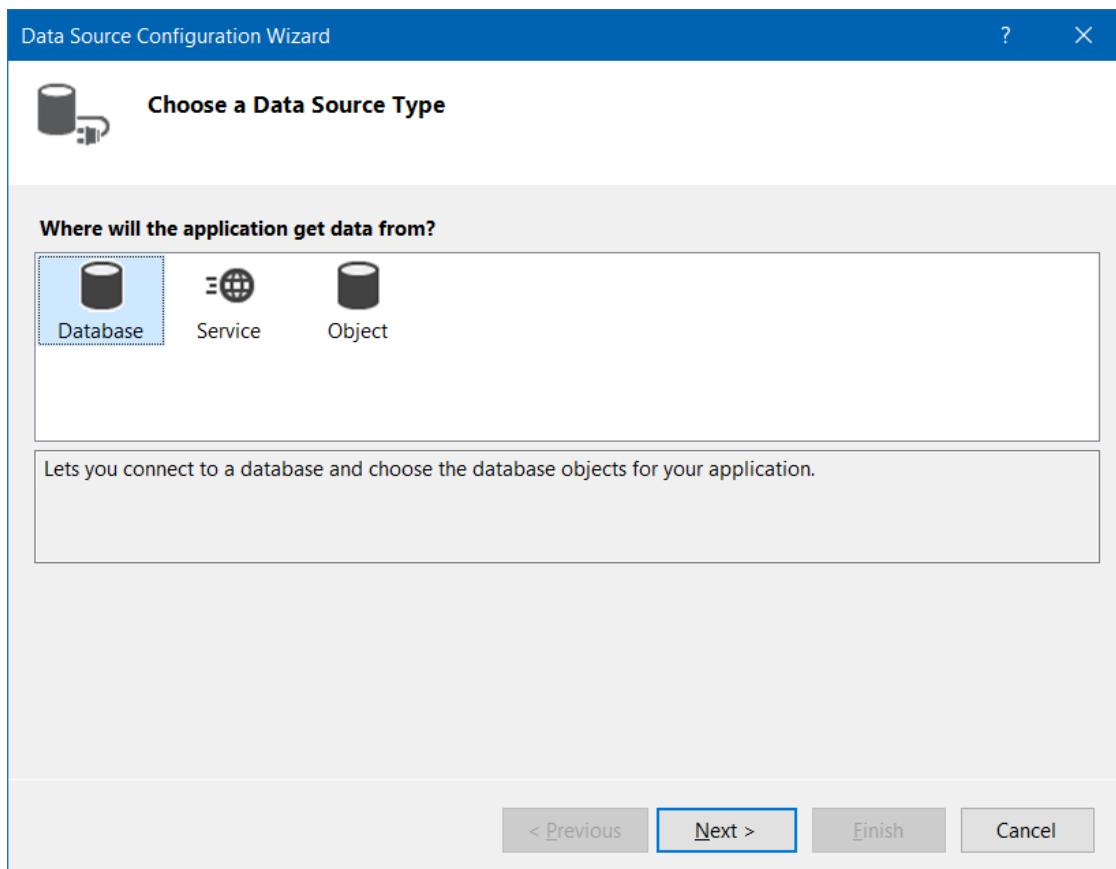
+ Trong cửa sổ Add New Item... nhập tên cho form mới, sau đó nhấn nút Add để thêm form vào dự án.



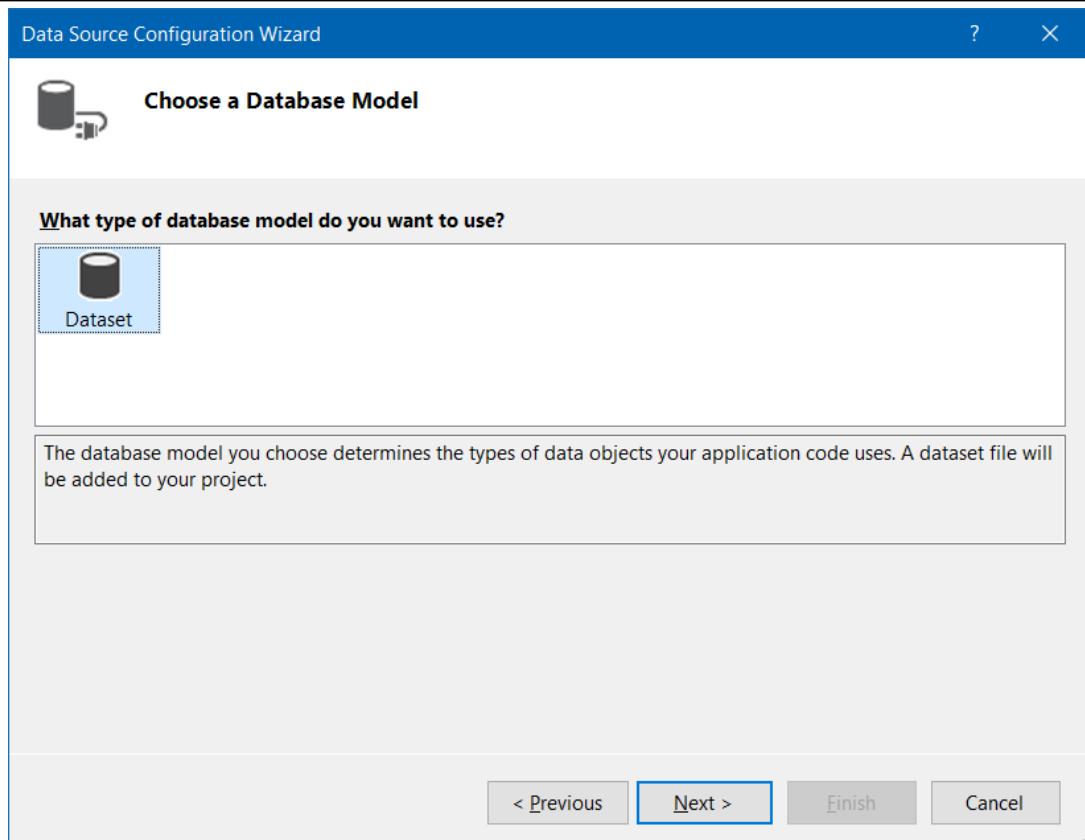
Lưu ý, tên form nên bắt đầu với tiền tố frm đi kèm với cụm từ gợi nhớ về chức năng của form.

1. Tạo DataSet

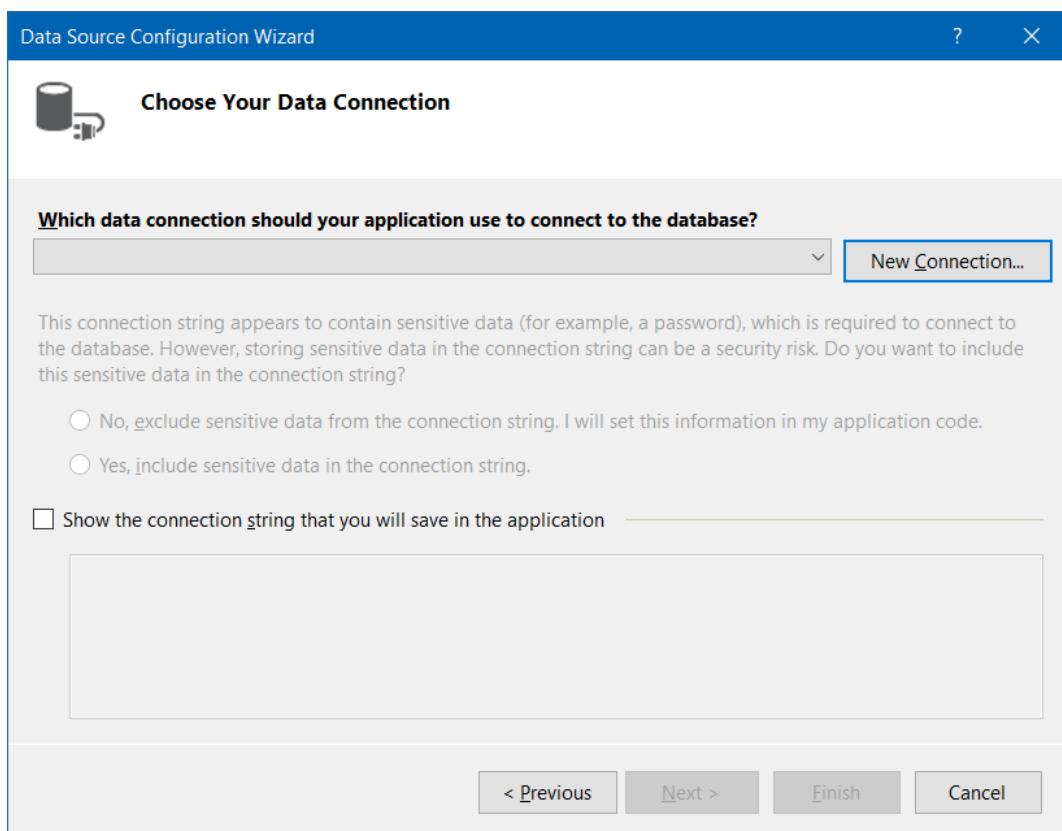
- Tạo cở sở dữ liệu QLBH trong SQL Server theo mô tả sau:
SanPham(hangID, tenhang, dongia)
Hoadon(hoadonID, ngayban, nguoinhan, **hangid**, soluong, thanhtien)
- Từ menu Project chọn **Add New Data Source...**
- Trong cửa sổ **Data Source Configuration Wizard** lần lượt thực hiện các thao tác
 - + **Choose a Data Source Type** (Chọn kiểu nguồn dữ liệu): **Database**. Nhập chọn Next



- + **Choose a Database Model** (chọn mô hình cơ sở dữ liệu): **Dataset**. Nhập chọn Next

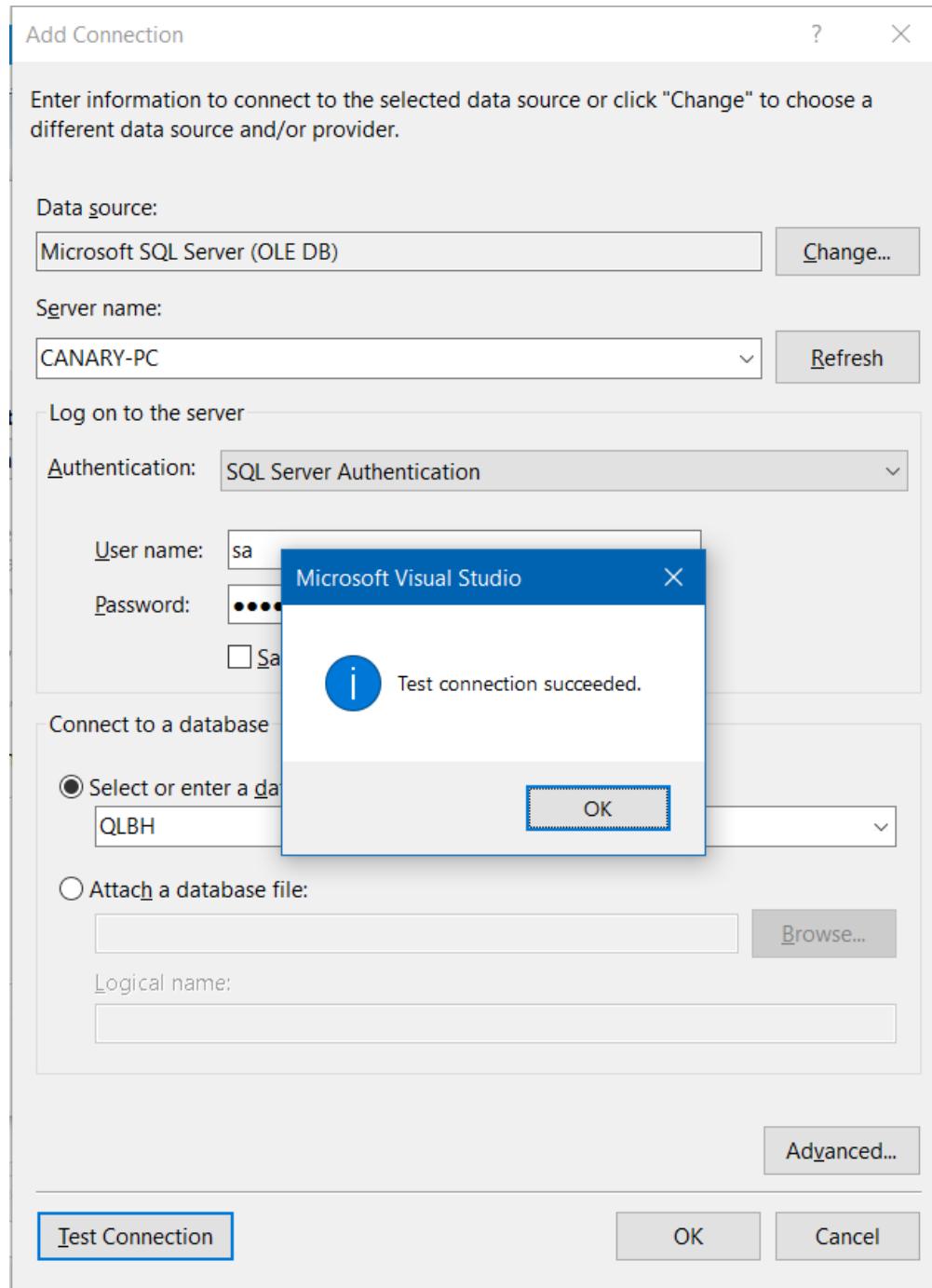


+ **Choose Your Data Connection** (Chọn kết nối dữ liệu): nếu đã có kết nối phù hợp chọn kết nối đó và click Next. Nếu chưa có kết nối hoặc muốn tạo một kết nối mới click chọn **New Connection...**

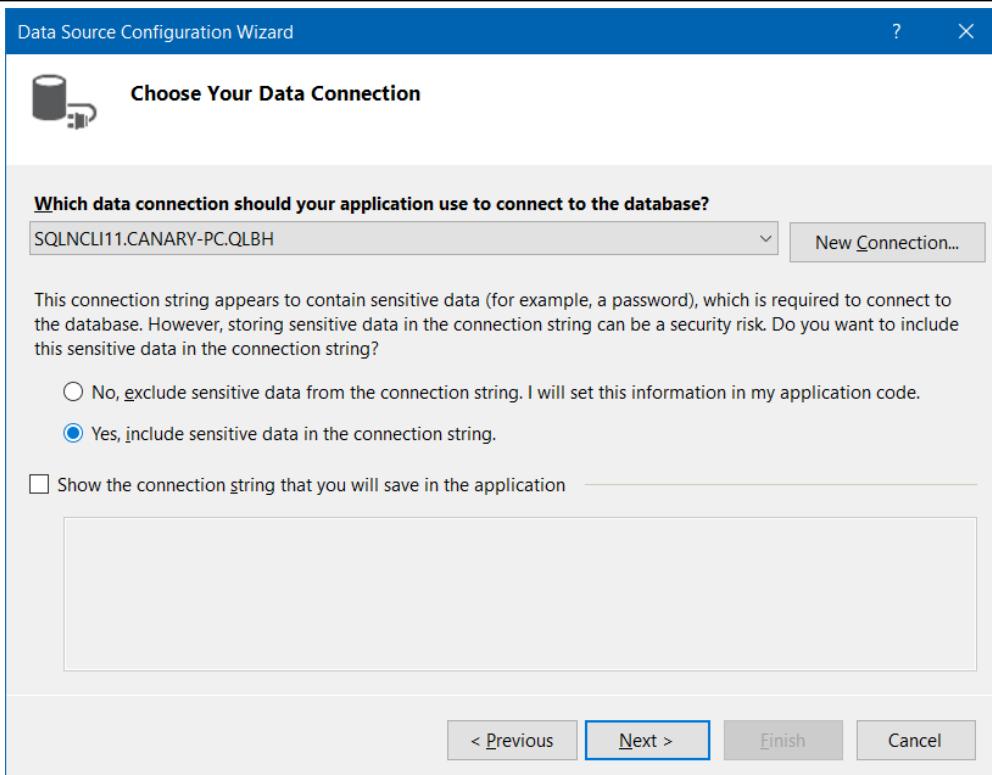


+ **New Connection...**

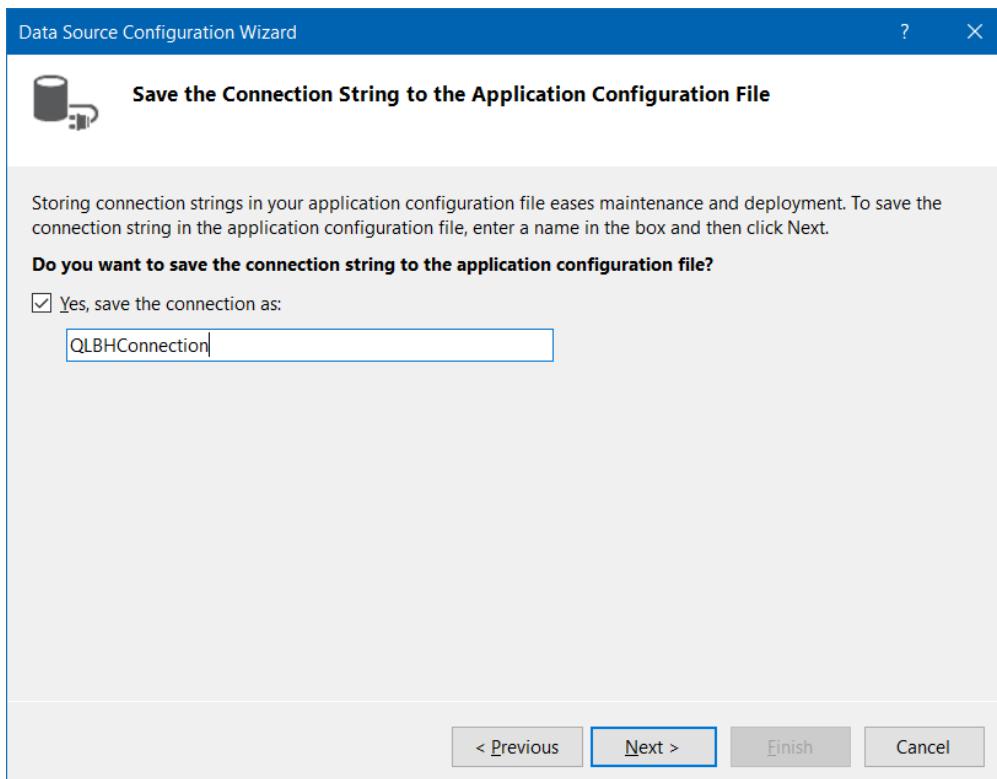
- ✓ Trong cửa sổ **Add Connection** lần lượt chọn hoặc nhập các thông tin tương ứng theo yêu cầu.
- ✓ Có thể chọn Test Connection để kiểm tra kết quả kết nối
- ✓ Click OK để tạo kết nối mới



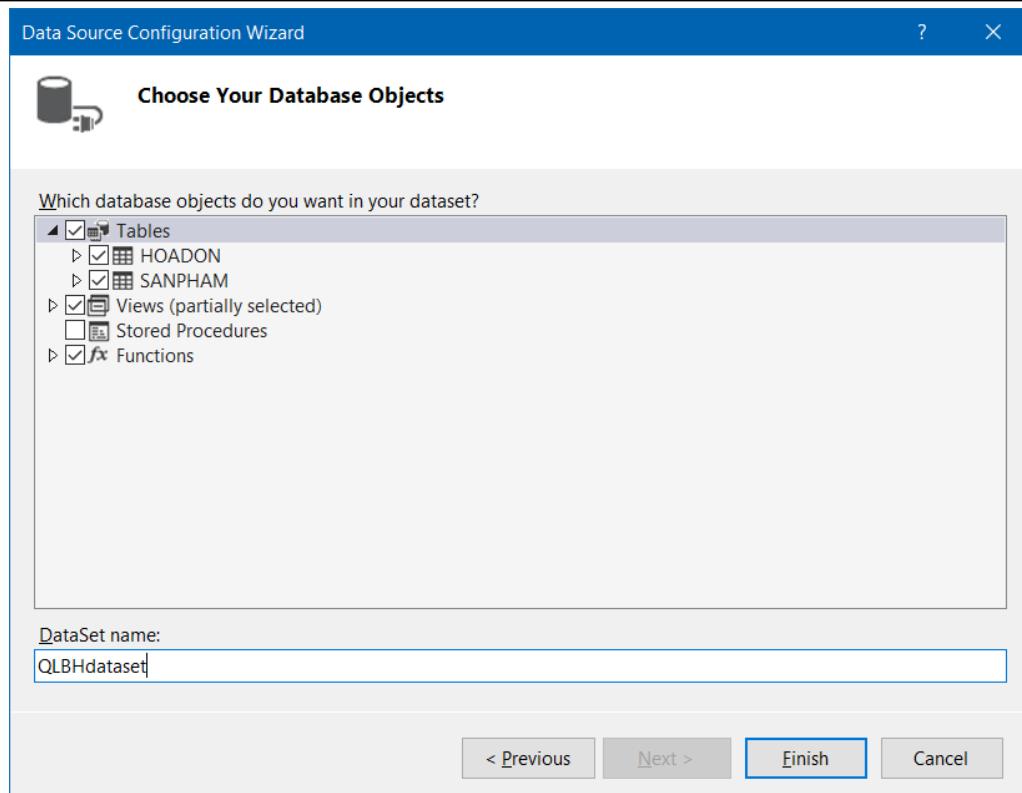
+ Trong cửa sổ **Data Source Configuration Wizard** ở mục **Choose Your Data Connection**, chọn **Yes, include sensitive data in the connection string** để thiết lập lưu trữ password vào kết nối trong Settings của project. Click Next



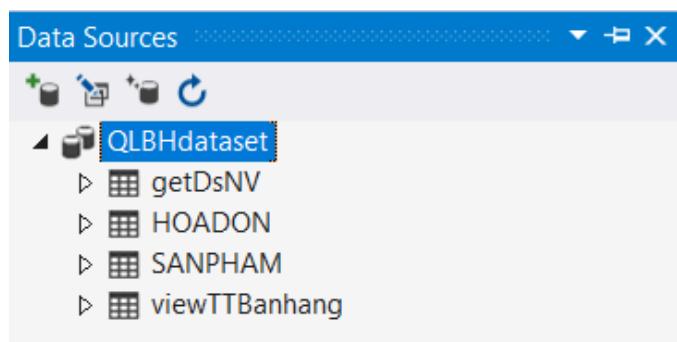
- + Đặt lại tên cho kết nối vừa tạo (để dàng truy cập khi cần trong quá trình soạn thảo mã nguồn). Click Next.



- + **Choose Your Database Objects** (chọn các đối tượng trong CSDL để thêm vào Dataset): thực hiện chọn các đối tượng như Tables, Views, Stored Procedures và đặt tên cho dataset. Lưu ý, chỉ chọn các đối tượng do người dùng tạo.

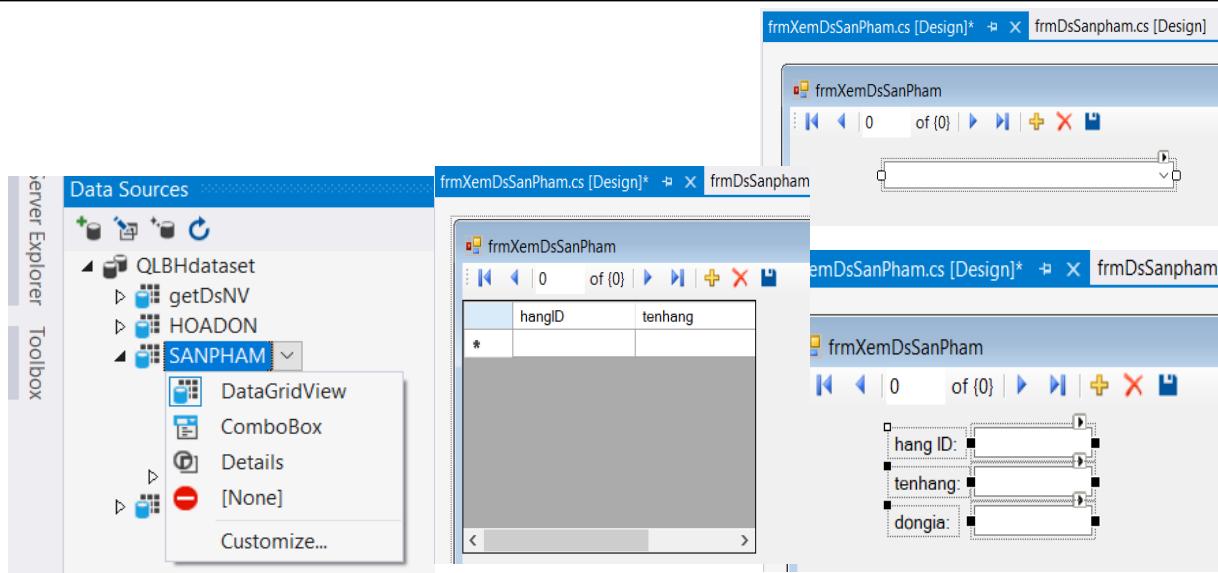


+ Sau khi tạo dataset mới sẽ xuất hiện trong cửa sổ **Data Sources** (View → Other Windows → Data Sources).



2. Kéo thả dữ liệu từ Dataset vào Form (DataBinding)

- Kéo thả dữ liệu từ dataset trong cửa sổ **Data Sources** vào form để tự sinh ra các điều khiển (controls) tương ứng với các tùy chọn: **DataGridView** (hiển thị dưới dạng DataGridView), **Combobox** (hiển thị dưới dạng ComboBox), **Details** (các trường dữ liệu sẽ hiển thị dưới dạng các Label cùng với các điều khiển cho phép nhập liệu như TextBox, DateTimePicker...). Ngoài ra, người dùng có thể chọn Customize để có những tùy chọn khác.

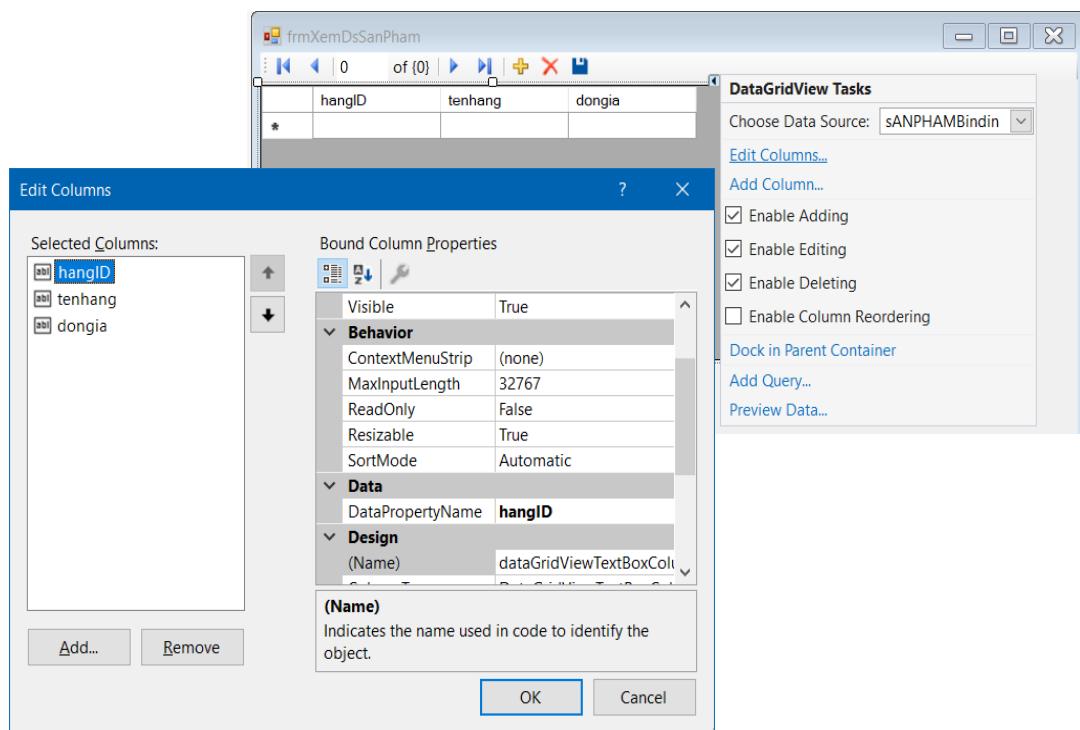


- + Thanh **BindingNavigator** sẽ được tự động sinh ra khi kéo thả dữ liệu từ dataset vào form. Nếu không sử dụng, người dùng có thể xóa thanh này khỏi form.

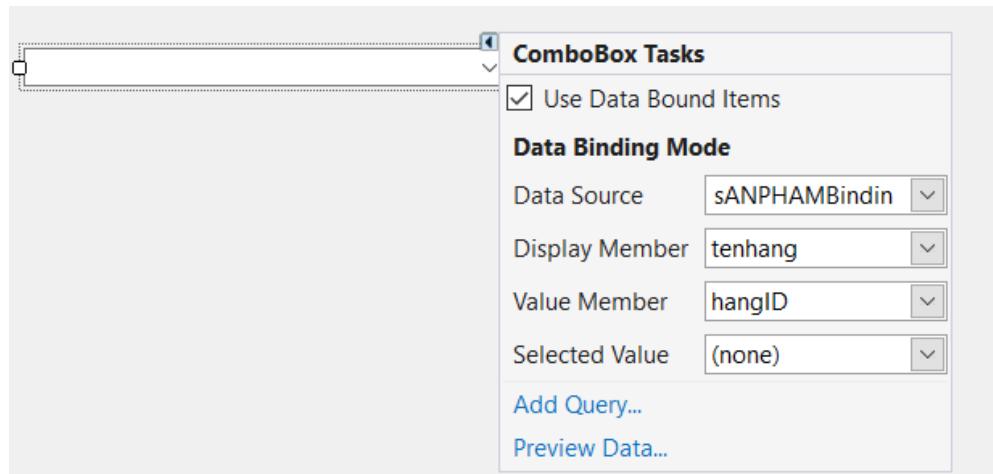


- Lưu ý:

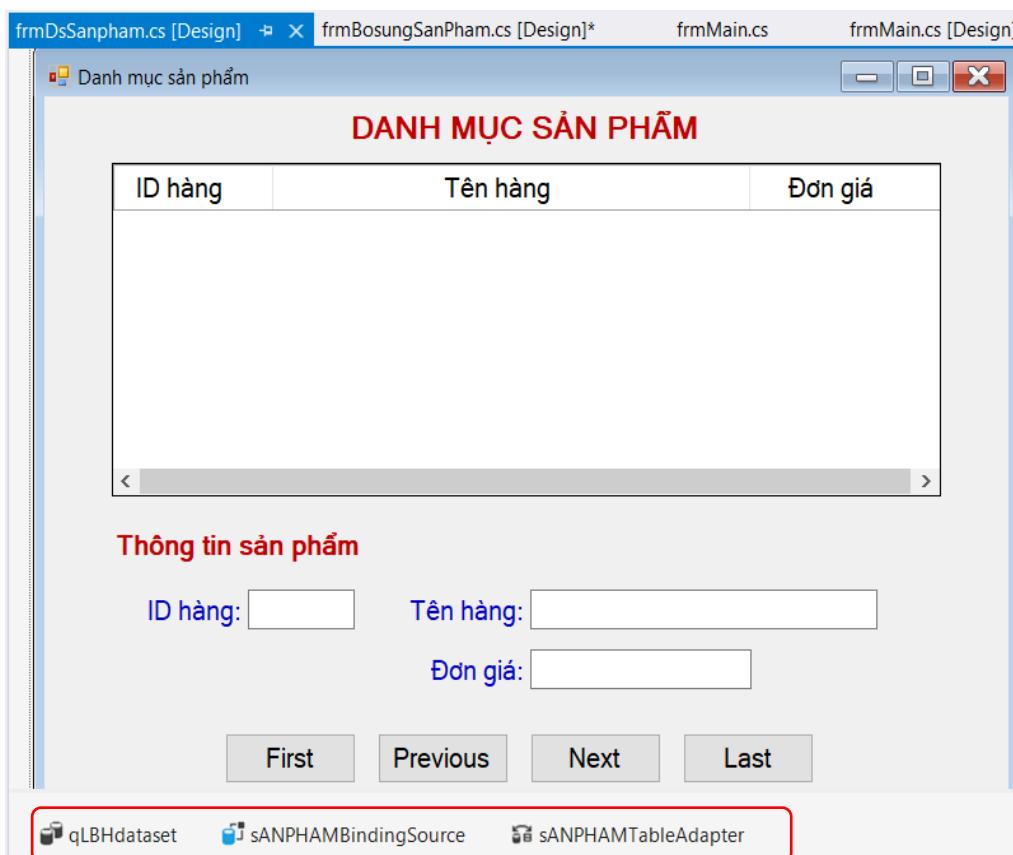
- + **DataGridView**: cần thao tác với Edit Columns để điều chỉnh tiêu đề cột sang tiếng Việt (nếu cần) và thiết lập lại một số định dạng có liên quan đến nội dung hiển thị trong **DataGridView**. Nếu **DataGridView chỉ cho phép người dùng xem dữ liệu** (không thêm, sửa, xóa dữ liệu) cần thay đổi các thuộc tính **Allow...** của **DataGridView** từ **true** về **false**. Ngoài ra, một vài thuộc tính khác của **DataGridView** có thể được thay đổi giá trị theo ý tưởng thiết kế từng cá nhân.



+ **ComboBox:** cần định rõ giá trị cho **Display Member** (thành phần hiển thị ra bên ngoài với người dùng ứng dụng) và **Value Member** (thành phần giá trị dùng để tương tác, khóa truy vấn dữ liệu) của ComboBox. Lưu ý bên cạnh đó cần thêm Label cho ComboBox bằng cách sử dụng điều khiển trong ToolBox.



- + **Details:** cần điều chỉnh các thuộc tính của Label và các điều khiển khác sao cho đảm bảo khả năng tương tác tốt của ứng dụng đối với người dùng
- Sau khi kéo thả dữ liệu từ dataset vào form các thành phần có liên quan hỗ trợ thao tác với dữ liệu cũng sẽ được tự động thêm vào form.



- Sử dụng DataBinding xử lý sự kiện Click của các nút First, Previous, Next và Last. Mỗi Windows Form đều có một thuộc tính BindingContext. Một BindingContext có một tập hợp các BindingManagerBase, các đối tượng này được tạo ra khi dữ liệu buộc vào một điều khiển.

+ Xử lý nút **First** (Về đầu), trong sự kiện **Click** của nút **First** thiết lập vị trí **BindingContext** là **mẫu tin đầu tiên** trong bảng dữ liệu **Sanpham** được lấy từ **sANPHAMBindingSource** (*Đây là một BindingSource được sinh ra khi “buộc” dữ liệu vào các điều khiển trên Form*)

```
sANPHAMBindingSource.MoveFirst();
```

+ Xử lý nút **Previous** (Lùi, Về trước), trong sự kiện **Click** của nút **Previous** thiết lập vị trí **BindingContext** là **mẫu tin trước** mẫu tin hiện tại đang hiển thị từ bảng dữ liệu **Sanpham** được lấy từ **sANPHAMBindingSource**

```
sANPHAMBindingSource.MovePrevious();
```

+ Xử lý nút **Next** (Tới, Về sau), trong sự kiện **Click** của nút **Next** thiết lập vị trí **BindingContext** là **mẫu tin tiếp theo** mẫu tin hiện tại đang hiển thị từ bảng dữ liệu **Sanpham** được lấy từ **sANPHAMBindingSource**

```
sANPHAMBindingSource.MoveNext();
```

+ Xử lý nút **Last** (Về cuối), trong sự kiện **Click** của nút **Last** thiết lập vị trí **BindingContext** là **mẫu tin cuối cùng** mẫu tin hiện tại đang hiển thị từ bảng dữ liệu **Sanpham** được lấy từ **sANPHAMBindingSource**

```
sANPHAMBindingSource.MoveLast();
```

3. Cập nhật dữ liệu sử dụng BindingNavigator

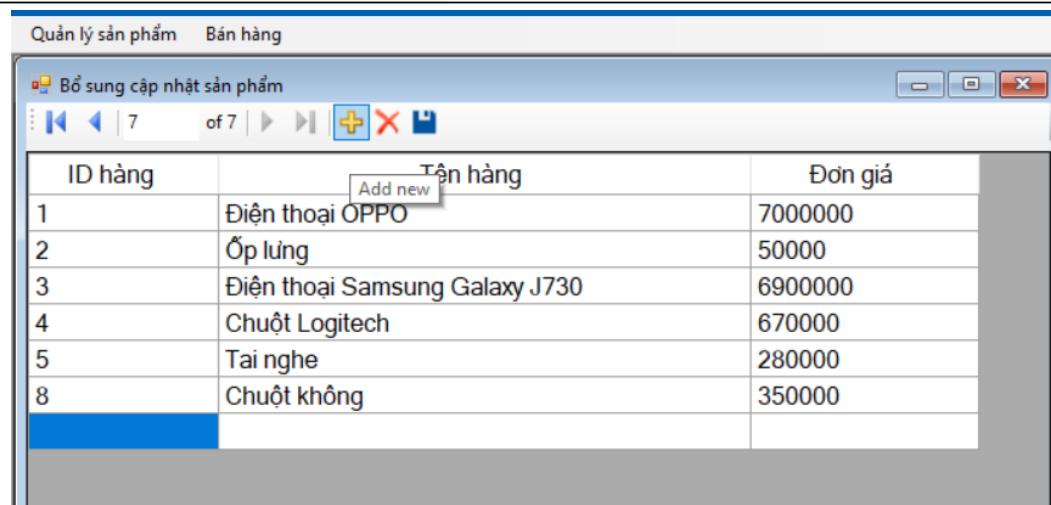


- **bindingNavigatorAddNewItem** : cho phép thêm một mẫu tin trên BindingSource, từ đó người dùng có thể thêm dữ liệu cho mẫu tin mới (thêm dòng trên DataGridView, các trường nhập dữ liệu được xóa trắng cho phép nhập dữ liệu mới).

- **bindingNavigatorDeleteItem** : cho phép xóa một hoặc nhiều mẫu tin đã chọn trên BindingSource.

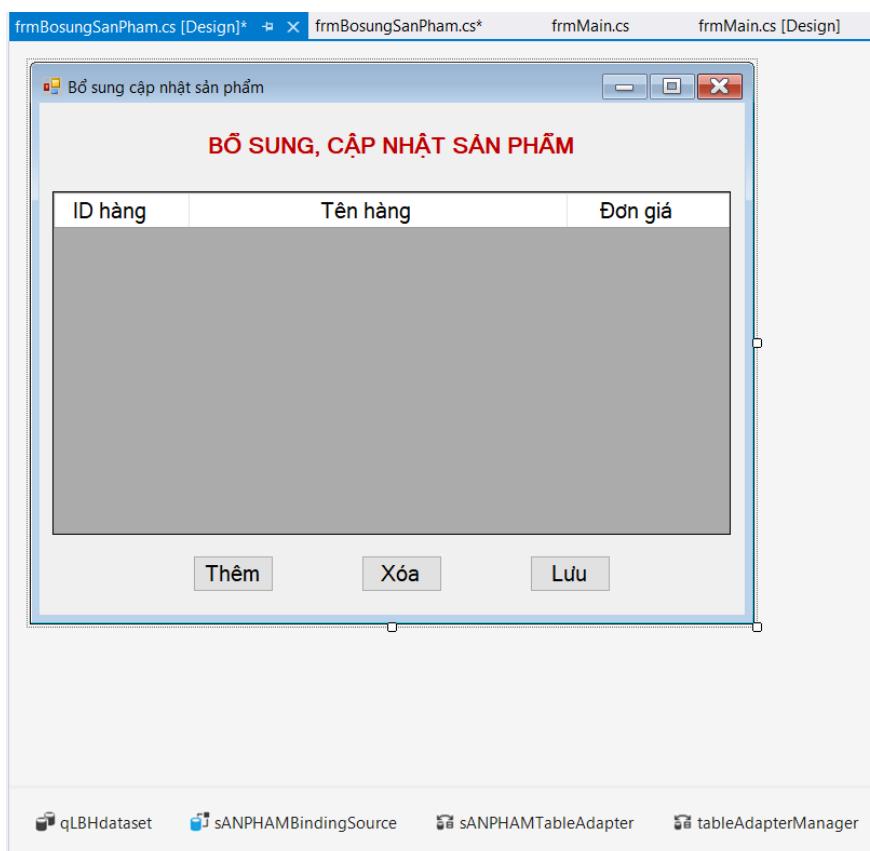
- **BindingNavigatorSaveItem** : lưu việc cập nhật dữ liệu trên BindingSource xuống database vật lý thực tế.

- Ví dụ minh họa Form cho phép Bổ sung cập nhật thông tin sản phẩm sử dụng BindingNavigator



4. Cập nhật dữ liệu không sử dụng BindingNavigator

- Tạo và thiết kế một Form mới **frmBoSungSanPham** cho phép cập nhật dữ liệu trực tiếp từ **DataGridView**



- Thêm mã tin trên DataGridView với nút **Thêm**: xử lý sự kiện **Click** của nút **Thêm** cho phép thêm 1 mẩu tin mới trên **sANPHAMBindingSource** thông qua phương thức **AddNew()**. Do **DataGridView** (dgvSanPham) đã được buộc dữ liệu vô **sANPHAMBindingSource**, nên khi Click và nút Thêm một dòng mới sẽ được sinh ra trên **DataGridView**.

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

```
sANPHAMBindingSource.AddNew();
```

```
}
```

- Xử lý **nút Xóa**, trong sự kiện **Click** của **nút Xóa**, viết lệnh cho phép người dùng xóa dòng hiện tại được chọn trên **DataGridView**

```
private void btnDel_Click(object sender, EventArgs e)
{
    //lấy dòng hiện tại đang được chọn của DataGridView
    int vt = dgvSanpham.CurrentRow.Index;
    //xóa mẫu tin tương ứng với dòng vt
    sANPHAMBindingSource.RemoveAt(vt);
}
```

- Xử lý **nút Lưu**, trong sự kiện **Click** của **nút Lưu**, viết lệnh cho phép cập nhật lại dữ liệu của **bảng Sanpham** trong **SQL Server** thông qua **DataSet** đã được thiết lập.

```
private void btnSave_Click(object sender, EventArgs e)
{
    SANPHAMTableAdapter.Update(qlBHdataset.SANPHAM);
}
```

- Kết quả khi **frmBoSungSanPham** thực thi



5. Lọc dữ liệu cho DataGridView dựa trên BindingSource

```
< tên BindingSource>.Filter = "Chuỗi điều kiện lọc";
```

- Ví dụ 1: Lọc dữ liệu cho **NhaCCSourceBindingSource** có **Địa Chỉ** ở **Vĩnh Long**

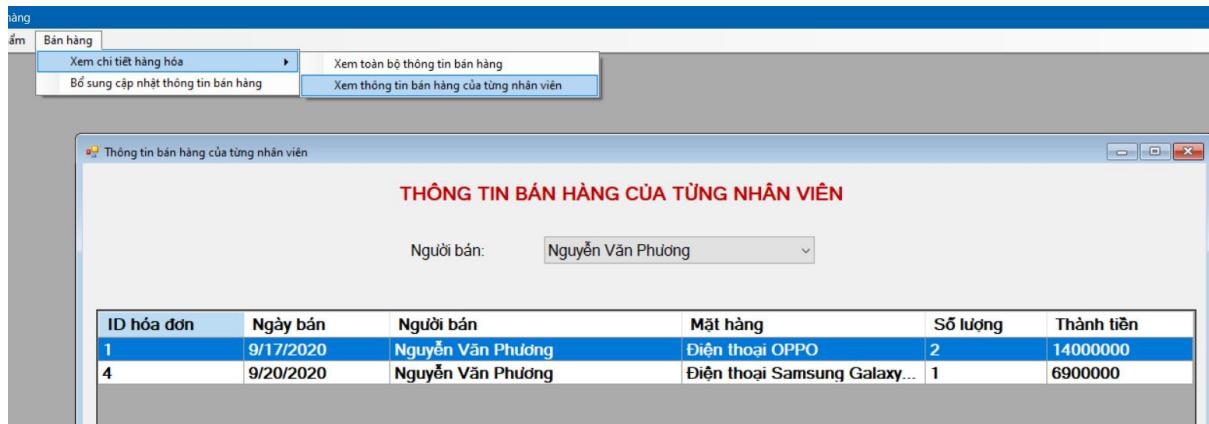
```
NhaCCBindingSource1.Filter = "DiaChi ='Vĩnh Long'";
```

- Ví dụ 2: Lọc dữ liệu cho **NhaCCSourceBindingSource** có **Địa Chỉ** là **giá trị** của combobox **Cmb_DiaChi**

```
NhaCCBindingSource1.Filter = "DiaChi ='" + Cmb_DiaChi.Text + "'";
```

- Ví dụ 3: Lọc dữ liệu **hOADONBindingSource** có **nguoiban** là **SelectedValue** (ValueMember) của ComboBox **cbxNgB**

```
hOADONBindingSource.Filter= "nguoiban=' " + cbxNgB.SelectedValue + " '";
```



PHẦN 3: VÍ DỤ MINH HỌA

I. Trong SQL Server xây dựng CSDL QLKHHT (Quản lý kế hoạch học tập) như sau:

HOCPHAN (MSHP, TenHP, TinchiLT, TinchiTH)

HOCKYNAMHOC(MSHK, HocKy, Namhoc)

HOC (#MSHP, #MSHK)

Giả sử mỗi năm học chỉ có 2 học kỳ chính I và II, hãy nhập dữ liệu cho bảng học kỳ năm học với số lượng 4 mẫu tin.

II. Trong Visual Studio, xây dựng Ứng dụng Quản lý kế hoạch học tập với các yêu cầu:

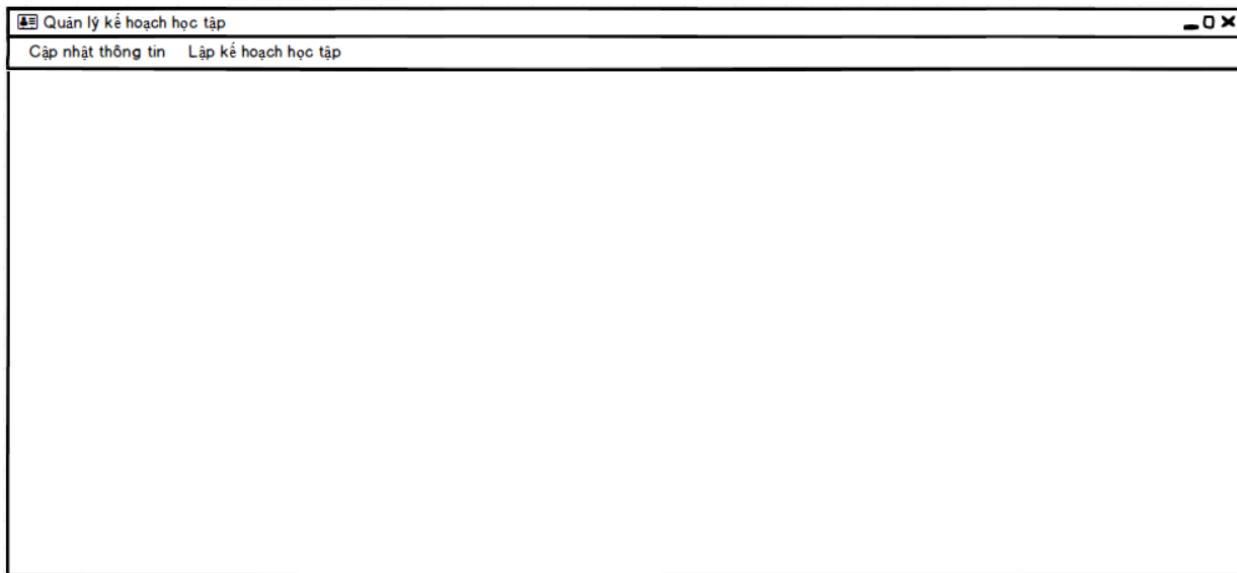
1. Xây dựng form chính, cho phép người dùng truy cập các chức năng của ứng dụng:

- *Cập nhật thông tin*

+ *Học phần*

+ *Học kỳ năm học*

- *Lập kế hoạch học tập*



2. Xây dựng form cập nhật thông tin học phần cho phép người dùng thêm, sửa, xóa thông tin của một học phần. *Lưu ý, nếu học phần đã được lập kế hoạch học tập thì người dùng không thể sửa hoặc xóa thông tin của học phần đó.*

3. Xây dựng form Lập kế hoạch học tập cho phép người dùng chọn những học phần sẽ học trong một học kỳ năm học cụ thể của họ. Lưu ý, số tín chỉ tối đa có thể học trong một học kỳ là 25 tín chỉ bao gồm cả tín chỉ lý thuyết và tín chỉ thực hành.

Lưu ý: Sinh viên có thể thiết kế giao diện các form khác với yêu cầu nhưng cần đảm bảo các tính năng được yêu cầu theo giao diện đã phát họa.

Chương 4

KẾT XUẤT BẢO BIỂU (REPORT)

4.1 GIỚI THIỆU CÁC CÔNG CỤ HỖ TRỢ TẠO BẢO BIỂU

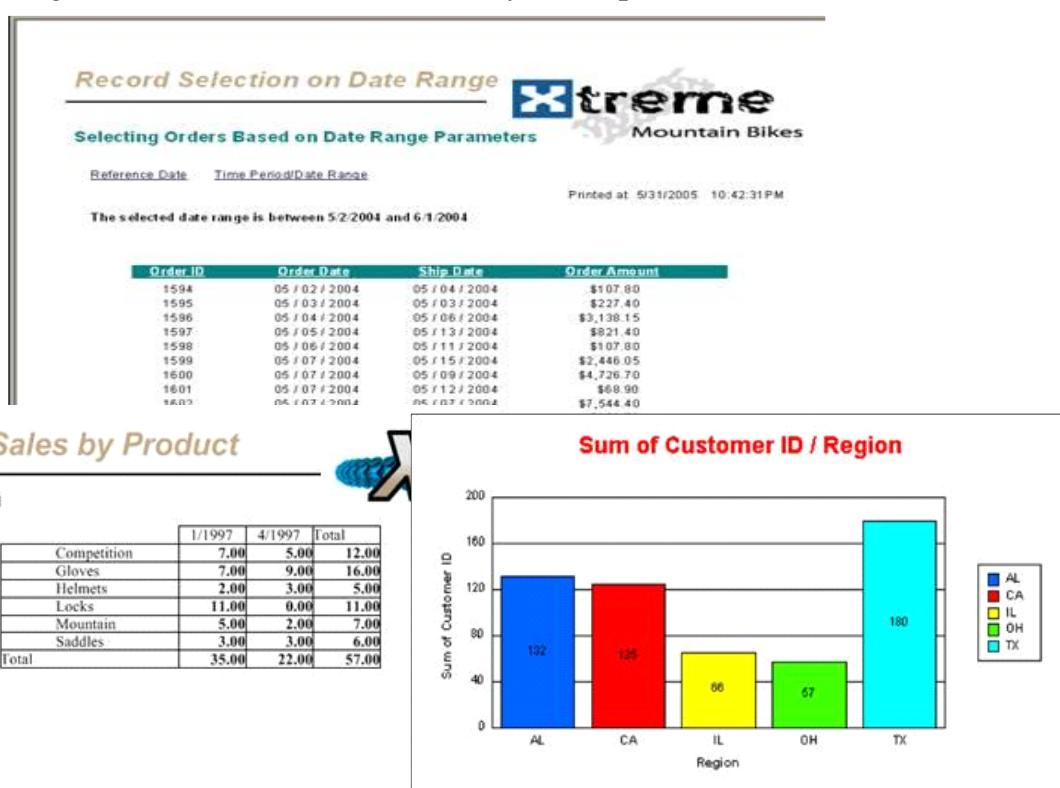
4.1.1 SAP Crystal Reports

a/- Giới thiệu

SAP Crystal Reports: là công cụ thiết kế báo cáo cho phép tạo ra những báo cáo và định dạng dữ liệu từ những dữ liệu nguồn khác nhau.

Ngoài việc đọc từ một dữ liệu nguồn, SAP Crystal Reports có riêng một ngôn ngữ công thức (formula language) để tính toán và một số tính năng khác như kèm theo biểu đồ (chart, graph) hoặc bảng cross-tab, ...

Một số dạng bảo biểu có thể tạo bởi SAP Crystal Reports:



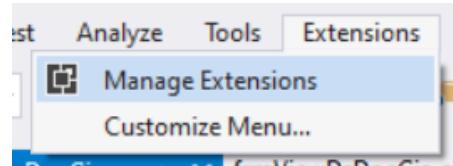
Hỗ trợ các chức năng in ấn, kết xuất sang các định dạng khác : PDF, Excel

b/- Cài đặt SAP Crystal Reports cho Visual Studio

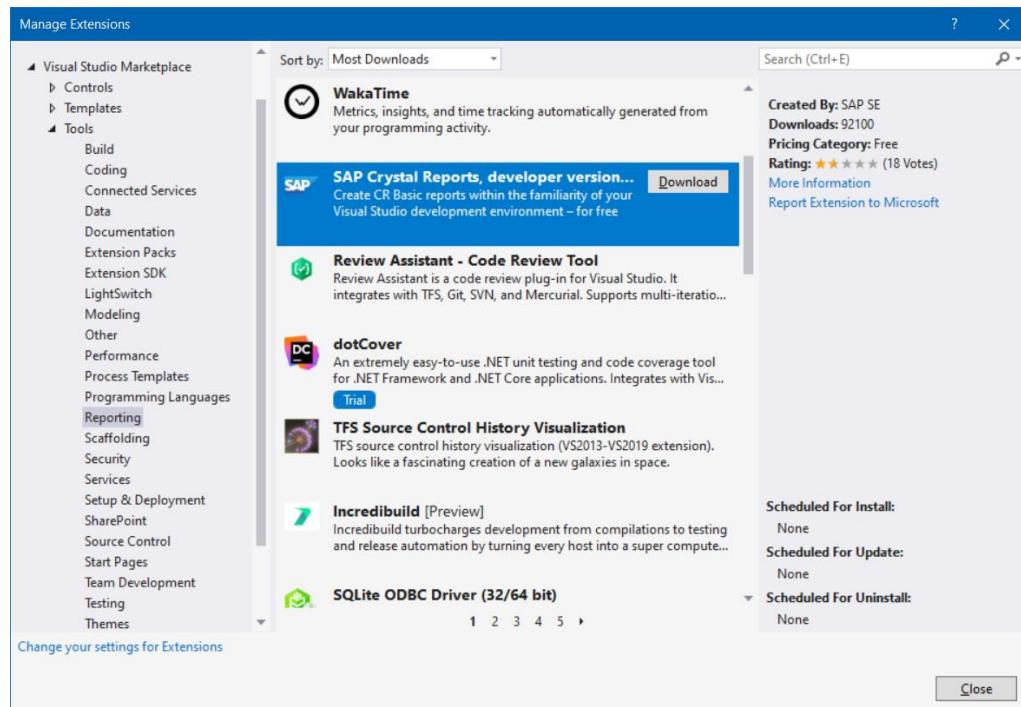
Từ phiên bản Visual Studio 2010 trở đi, Crystal Report không còn được tích hợp sẵn trong bộ Visual Studio mà đã tách ra thành một công cụ độc lập SAP Crystal Report, có thể được sử dụng như một phần mềm độc lập hoặc cài đặt như một phần tích hợp vào Visual Studio.

Các bước cài đặt SAP Crystal Report:

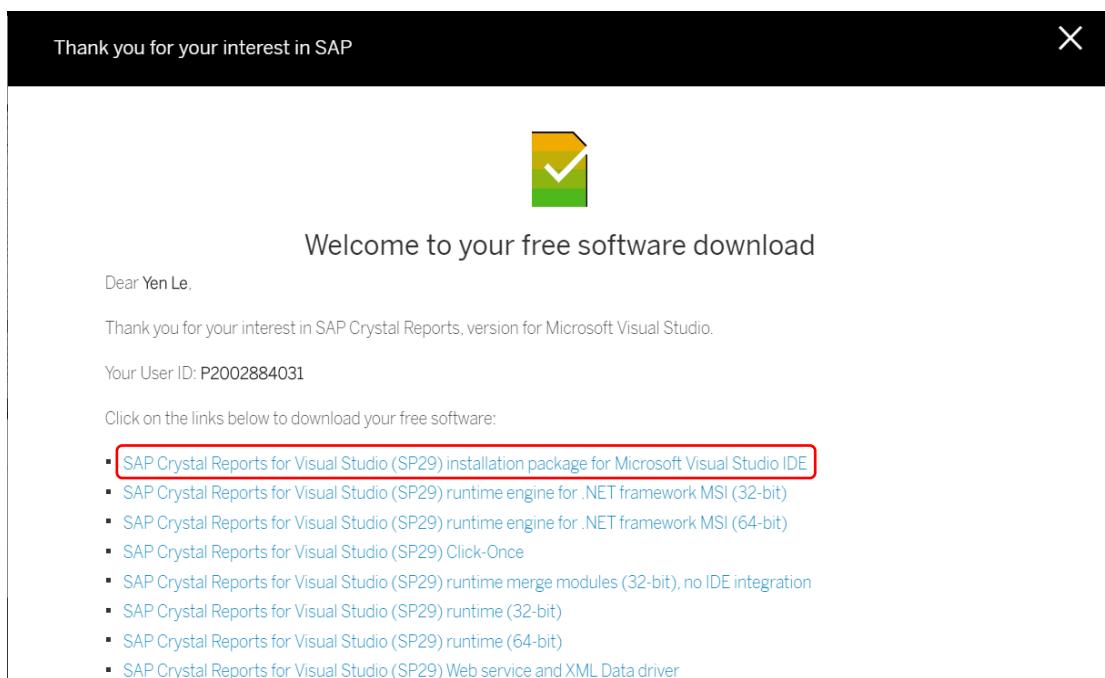
Bước 1: Tải bộ cài đặt CRforVS13SP29_0-10010309.EXE từ Manage Extensions của Visual Studio như sau:



Trong khung nhìn bên trái của cửa sổ Manage Extensions, chọn Online > Visual Studio Marketplace > Tools > Reporting. Sau đó chọn download SAP Crystal Reports trong khung nhìn giữa như sau:



SAP sẽ yêu cầu đăng ký tài khoản và việc này là hoàn toàn miễn phí. Sau khi đăng ký tài khoản, sử dụng tài khoản đăng nhập vào hệ thống, chọn gói sản phẩm cần tải xuống như sau:



Bước 2: Tắt ứng dụng Visual Studio nếu đang chạy.

Bước 3: Thực thi bộ cài đặt.

Bước 4: Sau khi hoàn thành, kiểm tra lại trong Visual Studio (Add New Item), thấy đã có Crystal Report.

4.1.2 Các công cụ khác

a/- Microsoft RDLC Report Designer

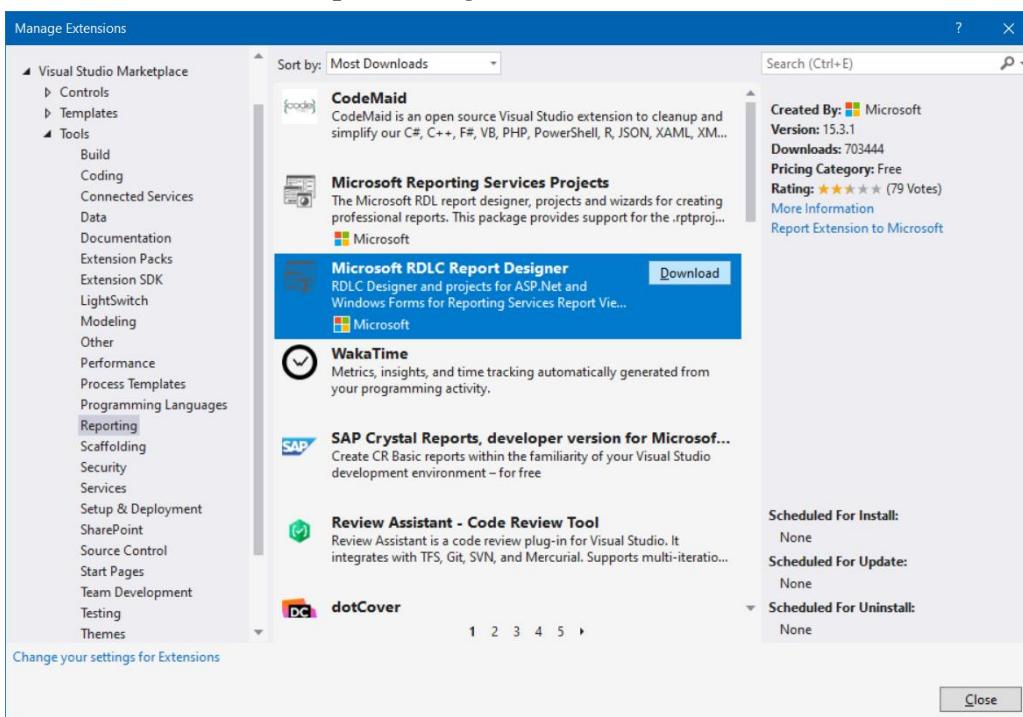
RDLC là viết tắt của Definition Language Client Side. Nó được dùng cho việc tạo các báo biểu sử dụng kỹ thuật reporting của Microsoft. Nó không phải là một báo biểu của bên thứ ba mà là một dịch vụ reporting tích hợp trong Microsoft Visual Studio.

Ưu điểm:

- Chạy phía client
- Chạy dựa trên chỉ một DataSet
- Chạy dựa trên một nguồn dữ liệu
- Không cần cài đặt bất kỳ dịch vụ reporting nào

Được hiển thị thông qua điều khiển ReportViewer. ReportViewer là một điều khiển phân phối miễn phí cho phép những báo biểu vào các ứng dụng được phát triển trên nền .NET Framework. Các báo biểu được thiết kế với những thao tác kéo thả đơn giản sử dụng Report Designer, một extension của visual studio.

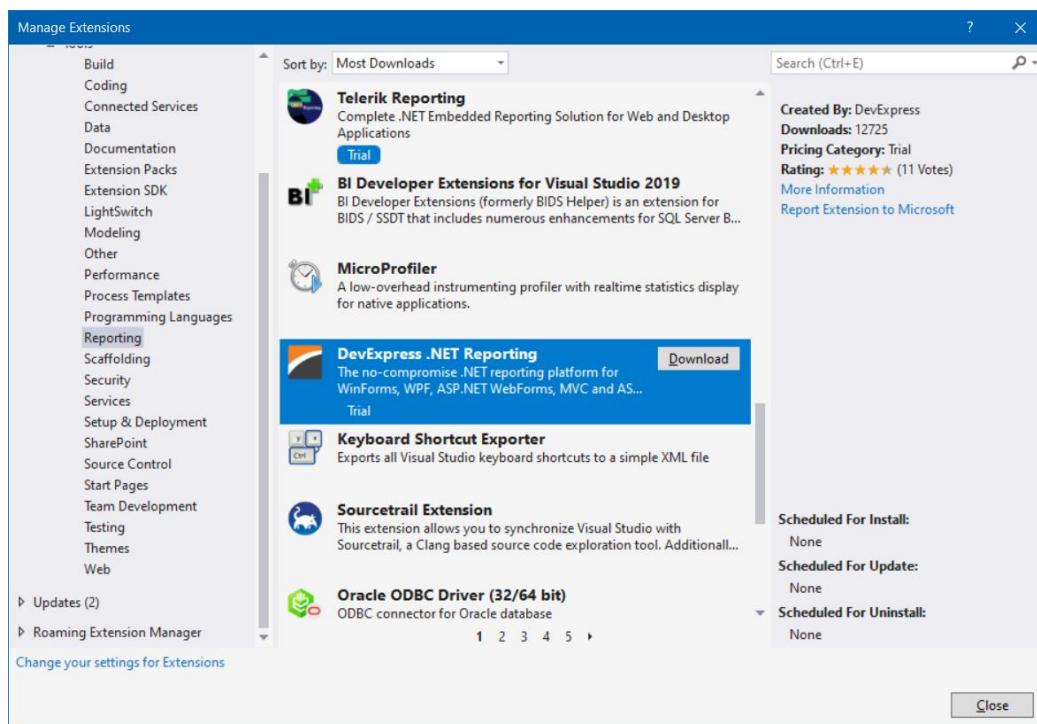
Để thêm Microsoft RDLC Report Designer vào visual studio, trong cửa sổ Manage Extensions chọn Online > Visual Studio Marketplace > Tools > Reporting, sau đó chọn Microsoft RDLC Report Designer để tải về.



b/- DevExpress .NET Reporting

DevExpress Reporting là một nền tảng reporting đầy đủ tính năng cho WinForms, WPF, ASP.NET Web Forms, ASP.NET MVC, ASP.NET Core và Blazor cho phép chúng

ta tạo các ứng dụng reporting. Tuy nhiên, DevExpress .NET Reporting yêu cầu phải trả phí để sử dụng.



4.2 PHÂN TÍCH YÊU CẦU VÀ THIẾT KẾ BÁO BIỂU

4.2.1 Phân tích yêu cầu kết xuất báo biểu

Tổng hợp dữ liệu là việc xác định các thông tin cần hiển thị, cách bố trí, tổ chức thông tin trên báo biểu

Báo biểu được thiết kế theo mẫu báo biểu hoặc các từ mô tả yêu cầu của khách hàng.

Một số tiêu chí khi thiết report:

- Đáp ứng mục tiêu nghiệp vụ, phù hợp với người sử dụng
- Số lượng vừa đủ, sắp xếp, gom nhóm hợp lý
- Trình bày dữ liệu đúng vị trí

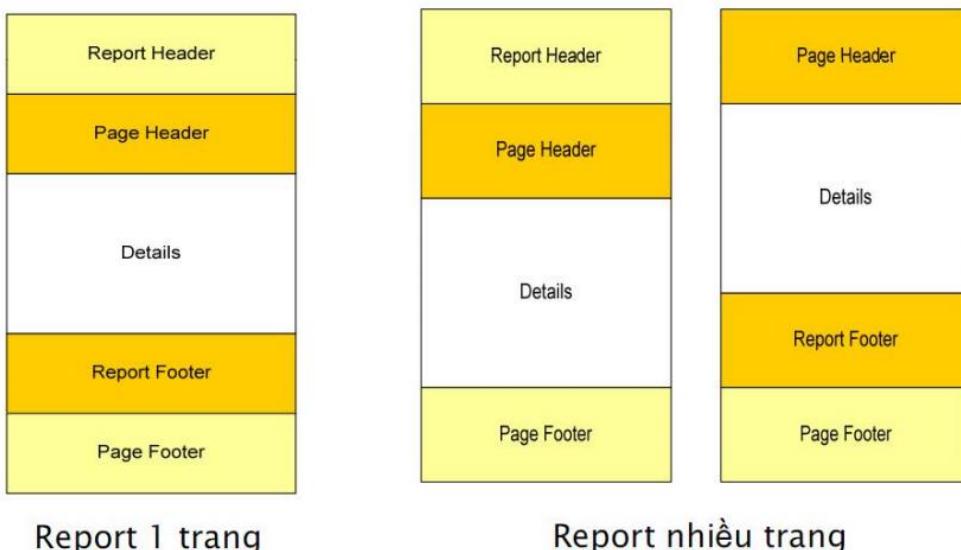
Ví dụ 4.1: Báo biểu thể hiện danh sách nhân viên của từng phòng, số nhân viên và lương lớn nhất của từng phòng

DANH SÁCH NHÂN VIÊN

<u>HONV</u>	<u>TENLOT</u>	<u>LUONG</u>	<u>NamSinh</u>
Phòng Phạm	Văn	55,000.00	1965
		Số nhân viên trong phòng	1
		Lương lớn nhất của phòng	55,000.00
Phòng Lê	Quỳnh	43,000.00	1967
Bùi	Ngọc	25,000.00	1954
Trần	Hồng	25,000.00	1967
		Số nhân viên trong phòng	3
		Lương lớn nhất của phòng	43,000.00
Phòng Trần	Thanh	25,000.00	1957
Nguyễn	Mạnh	38,000.00	1967
Nguyễn	Thanh	40,000.00	1962
Đinh	Bá	30,000.00	1960
		Số nhân viên trong phòng	4
		Lương lớn nhất của phòng	40,000.00
		<u>Tổng số nhân viên</u>	<u>8</u>

4.2.2 Phát thảo báo biểu

a/- Các thành phần của báo biểu trong SAP Crystal Reports



Report 1 trang

Report nhiều trang

Hình 4.1: Các thành phần của báo biểu trong SAP Crystal Reports

- Report Header: xuất hiện một lần ở trang đầu tiên của báo biểu (tên cơ quan, đơn vị, tiêu đề báo biểu).

- Page Header, Page Footer: xuất hiện lặp đi lại trên mỗi trang báo biểu (tiêu đề cột của các bảng dữ liệu, số thứ tự trang).
- Detail: nội dung chính của báo biểu, trình bày dữ liệu đã được tổng hợp, thống kê.
- Report Footer: xuất hiện một lần duy nhất ở trang cuối cùng của báo biểu (thông tin ký xác nhận)

Ví dụ 4.2: Xác định các thành phần báo biểu trong mẫu sau:

DANH SÁCH NHÂN VIÊN				
Trang	1			
Họ nhân viên	Tên lót	Tên nhân viên	Tên phòng	Tên trưởng phòng
Le	Quynh	Nhu	Điều hành	Quang
Trần	Thanh	Tâm	Nghiên cứu	Tùng
Nguyễn	Mạnh	Hùng	Nghiên cứu	Tùng
Nguyễn	Thanh	Tùng	Nghiên cứu	Tùng
Phạm	Văn	Vinh	Quản lý	Vinh
Bùi	Ngọc	Hằng	Điều hành	Quang
Trần	Hồng	Quang	Điều hành	Quang
Đinh	Bá	Tiến	Nghiên cứu	Tùng

Thành phố Hồ Chí Minh, Ngày	12/10/2007
Công ty TNHH ABC	

b/- Phát thảo báo biểu

Dựa trên kết quả phân tích yêu cầu kết xuất báo biểu, xác định thông tin và dữ liệu tương ứng cho từng thành phần của báo biểu.

Ví dụ 4.3: Kết xuất thống kê danh sách độc giả của thư viện

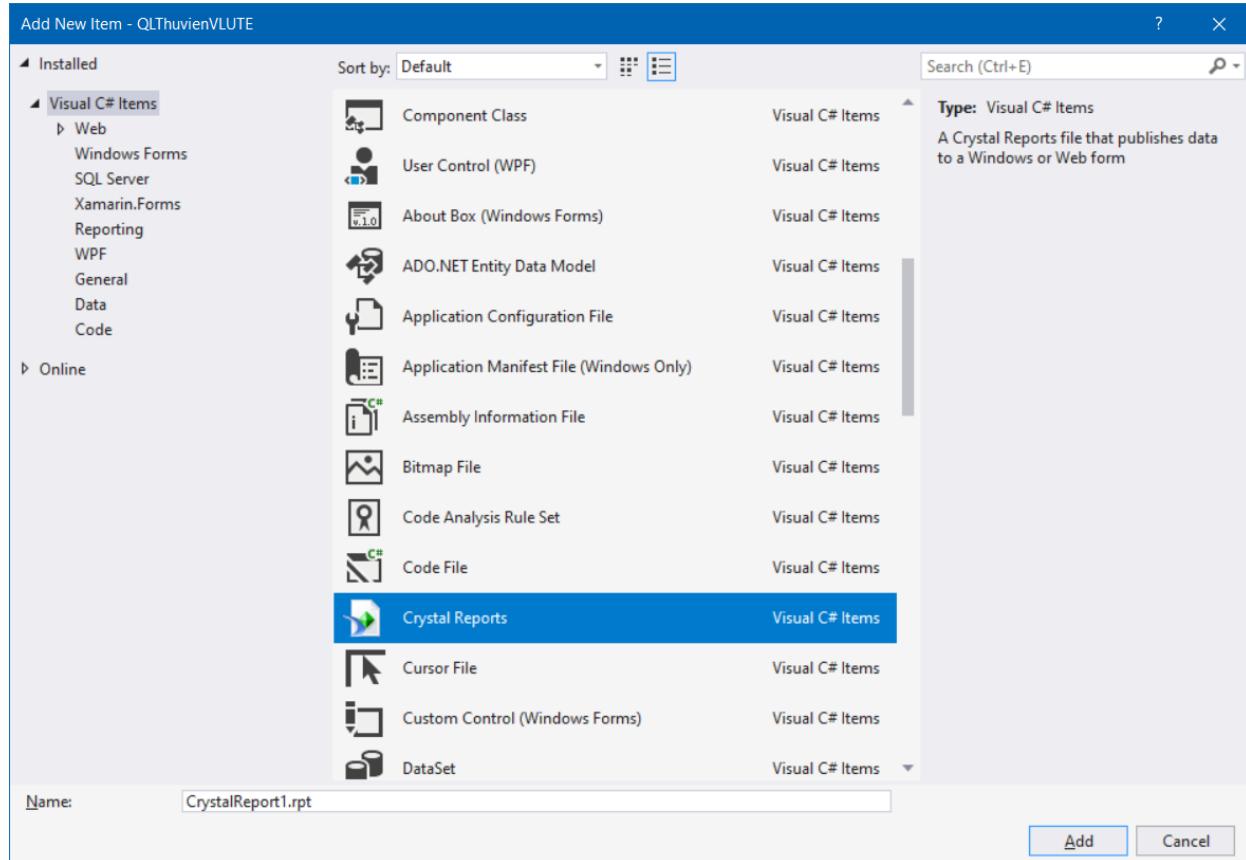
- Report Header: thông tin thư viện, tiêu đề báo biểu (DANH SÁCH ĐỘC GIẢ).
- Page Header: tiêu đề cột của bảng thống kê thông tin độc giả (Số thẻ, Họ và tên, Giới tính, Ngày sinh, SĐT, Email, Đơn vị).
- Detail: dữ liệu chi tiết tương ứng về thông tin độc giả.
- Page Footer: số thứ tự trang/ tổng số trang, thông tin thư viện làm footer của trang.
- Report Footer: ký xác nhận của quản lý thư viện.

4.2.3 Thiết kế báo biểu dựa trên bản phát thảo

a/- Tạo mới một báo biểu với SAP Crystal Reports

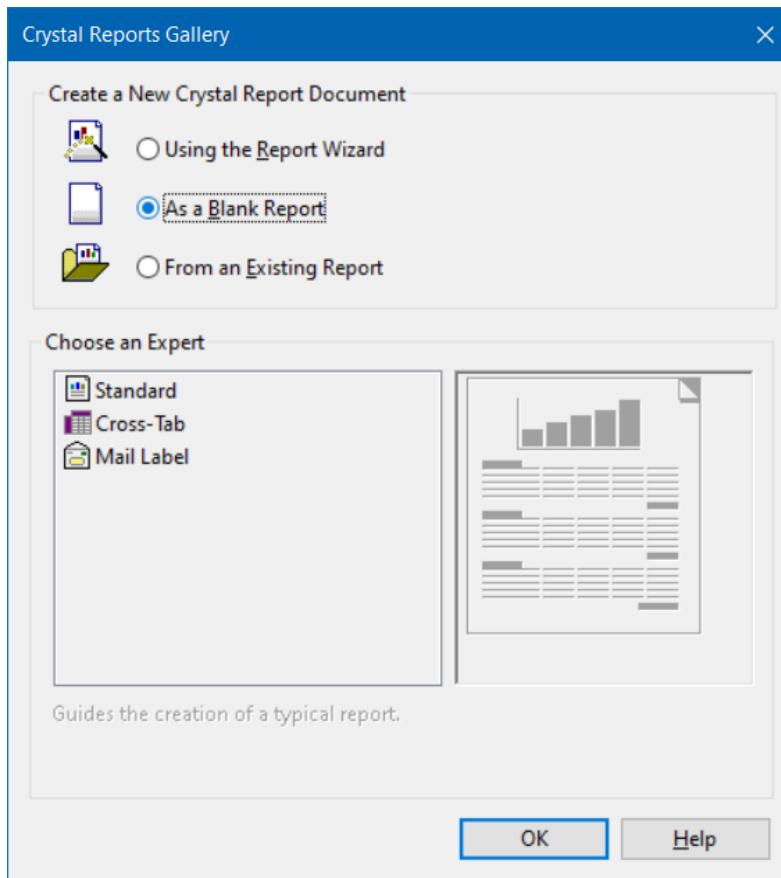
Bước 1: Nhấp chuột phải trên Project trong khung Solution Explorer, chọn Add/New Item... (hoặc nhấn tổ hợp phím Ctrl+Shift+A)

Bước 2: Trong cửa sổ Add New Item, chọn Crystal Reports.

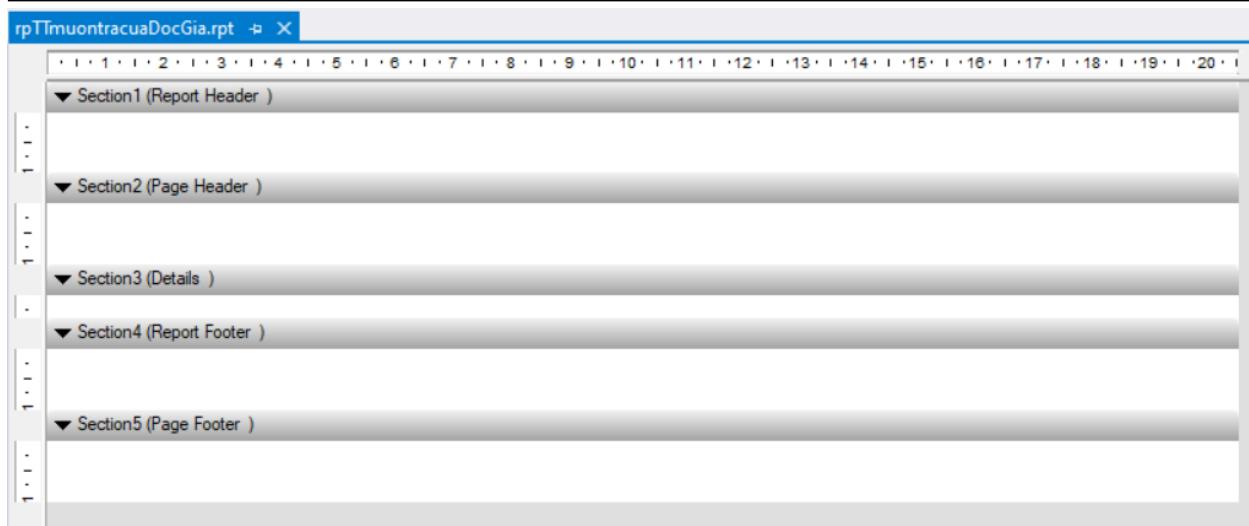


Bước 3: Trong trường Name, đặt tên cho báo cáo cần tạo. Sau đó nhấn Add.

Bước 4: Trong hộp thoại Crystal Report Gallery, chọn As a Blank Report, nhập OK.



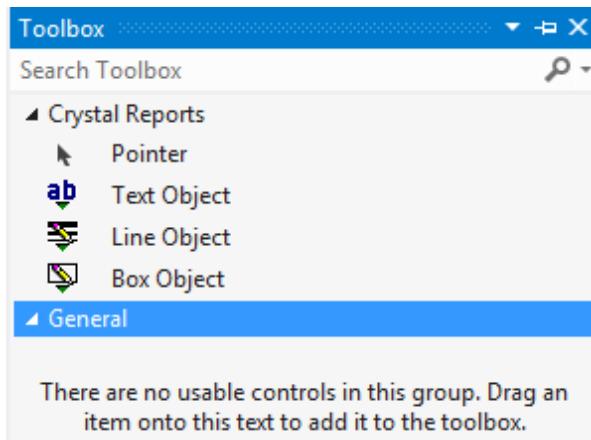
Hoàn thành Bước 4, báo cáo mới được tạo ra với các thành phần rỗng như sau:



b/- Thêm các đối tượng dữ liệu cho báo biểu

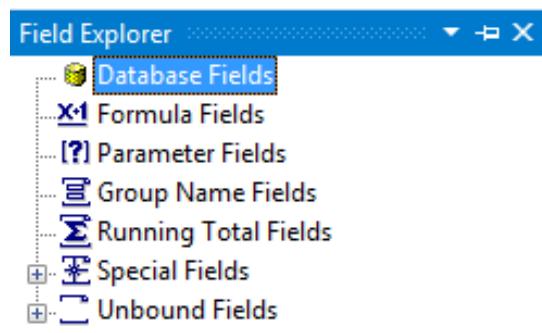
❖ Toolbox

Visual Studio, hỗ trợ một số công cụ cho việc thiết kế một báo biểu theo yêu cầu người dùng.



Ví dụ, để thêm một đối tượng văn bản như tiêu đề báo biểu hay dòng ghi chú chúng ta có thể sử dụng Text Object.

❖ Field Explorer

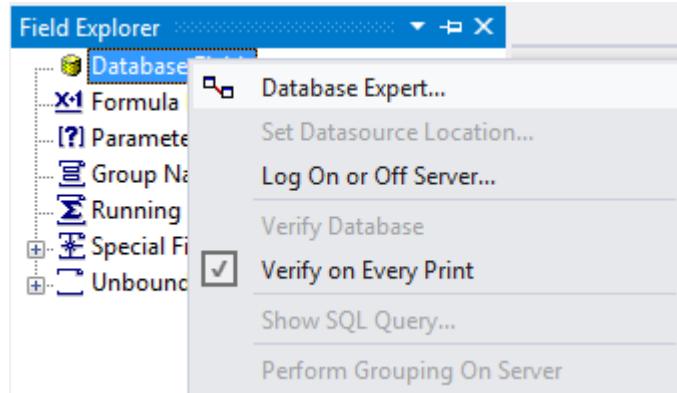


Ngoài các công cụ trong Toolbox, Visual Studio còn cung cấp Field Explorer hỗ trợ việc thêm mới các đối tượng dữ liệu cho một báo biểu.

Database Fields: Thêm các đối tượng dữ liệu được lấy từ một CSDL, có thể là table, stored procedure, SQL command). Thông thường các trường này sẽ được hiển thị trong phần detail của báo biểu.

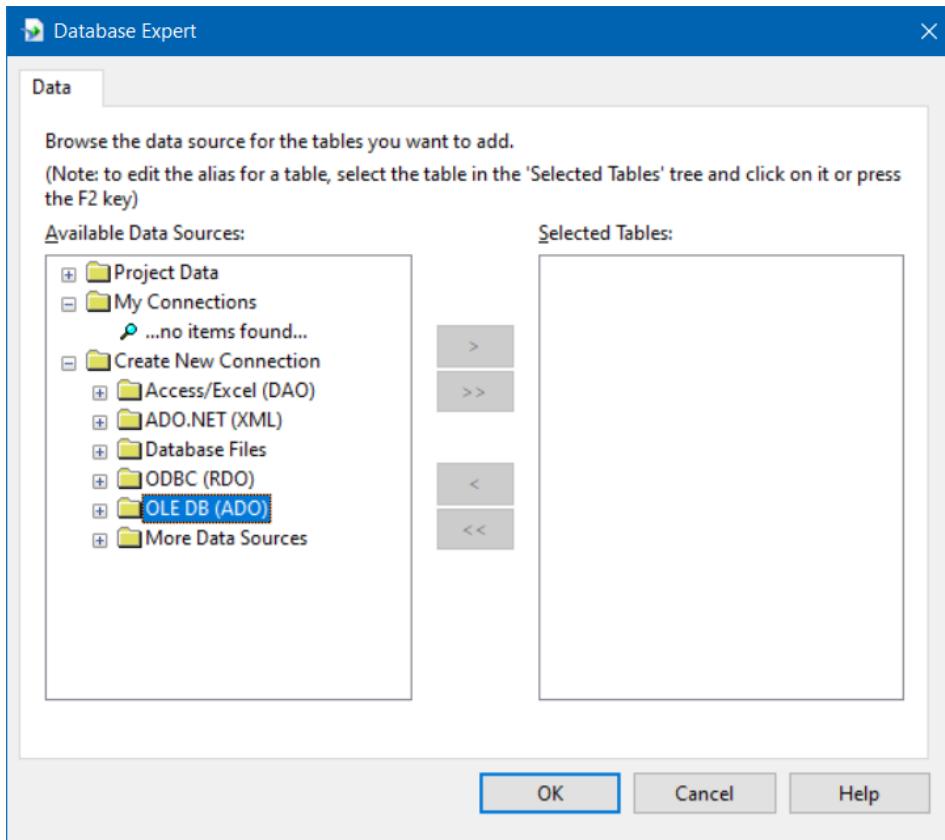
Thêm đối tượng Database Fields:

Bước 1: R-Click Database Fields, chọn Database Expert...

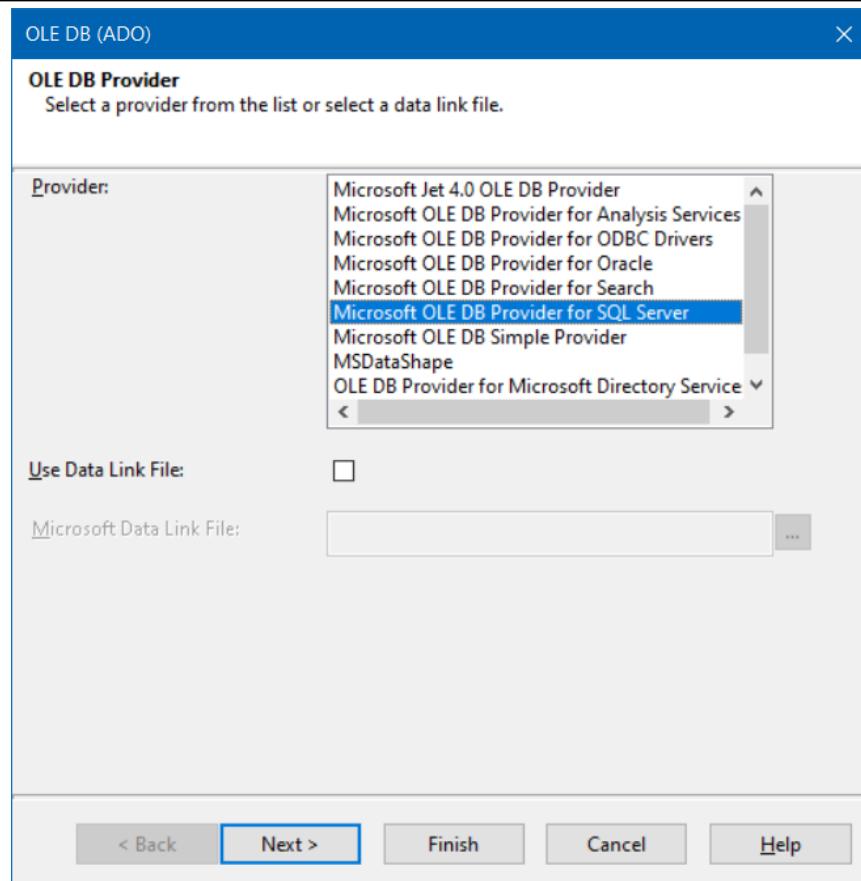


Bước 2: Trong hộp thoại Database Expert, tạo kết nối mới đến CSDL nếu cần hoặc trong My Connections chưa có thành phần con (...no items found...).

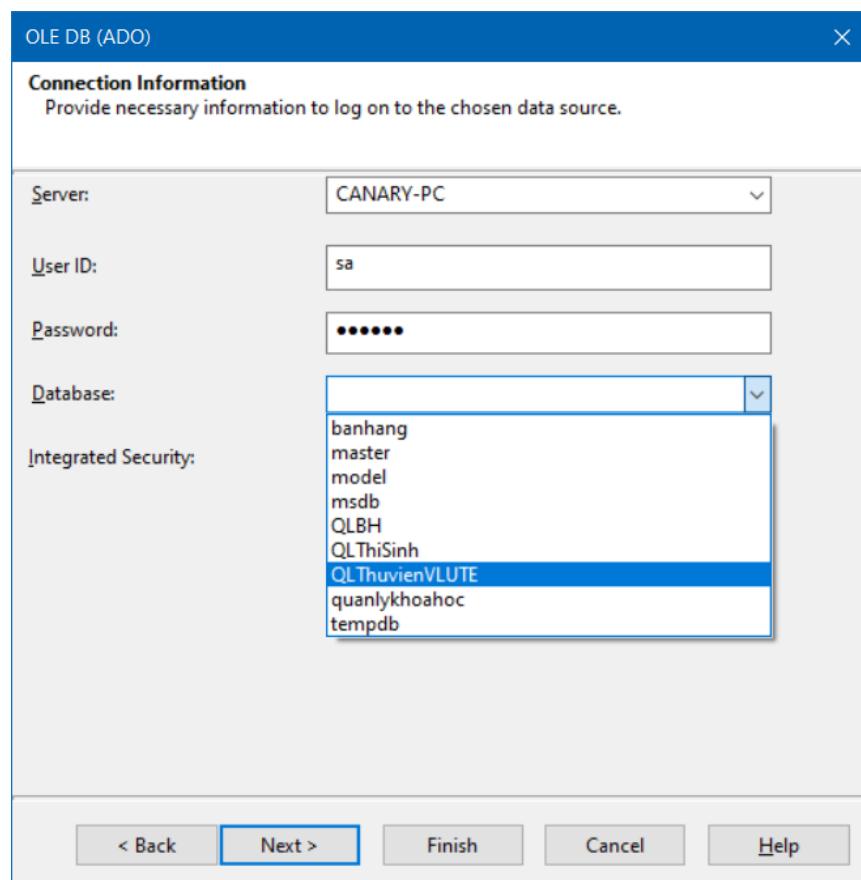
Bước 2.1: Chọn Create New Connection\OLE DB (ADO).



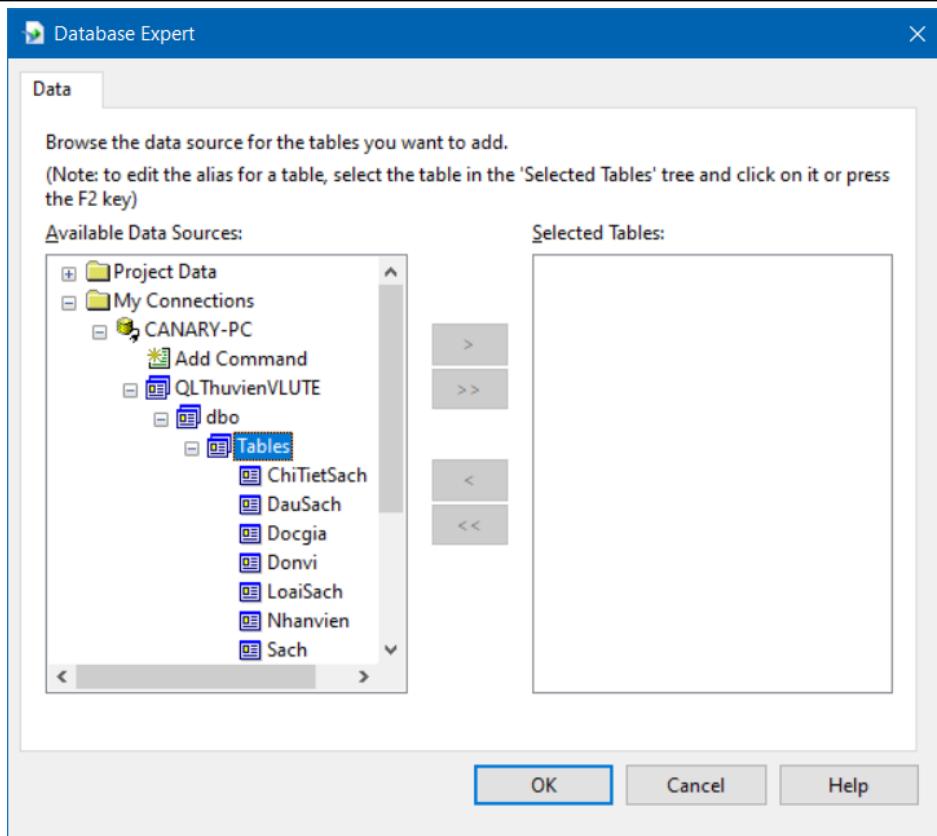
Bước 2.2: Hộp thoại OLE DB (ADO) xuất hiện. Trong Provider chọn Microsoft OLE DB Provider for SQL Server. Sau đó click Next.



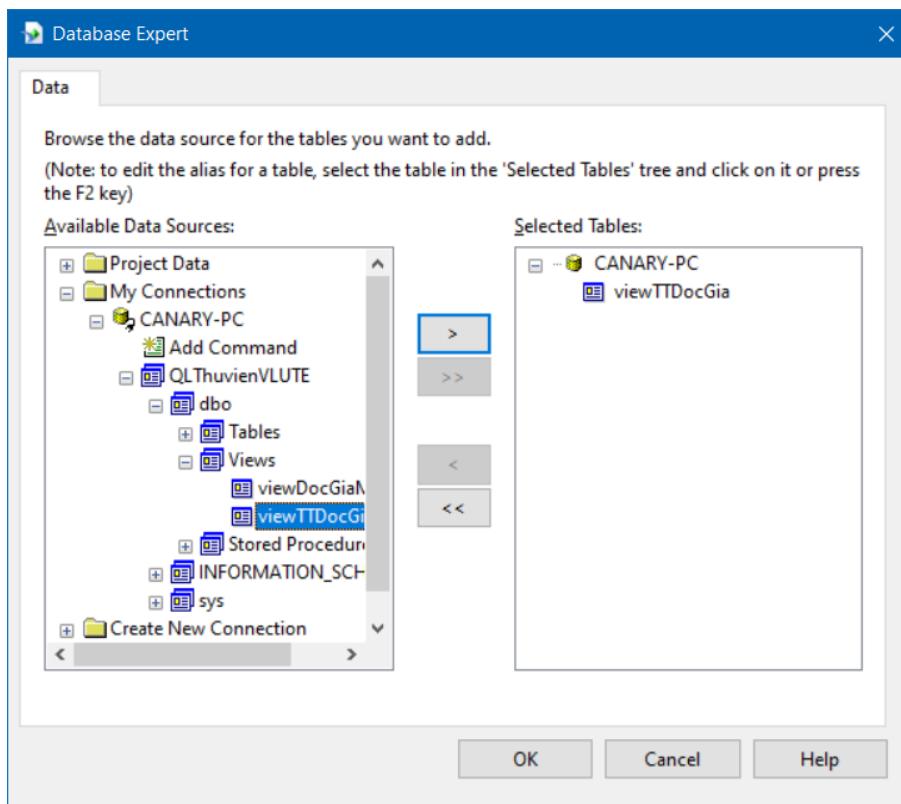
Bước 2.3: Nhập thông tin kết nối CSDL cần thiết để thêm đối tượng dữ liệu vào báo biểu (Server, User ID, Password, Database). Sau đó click Finish.



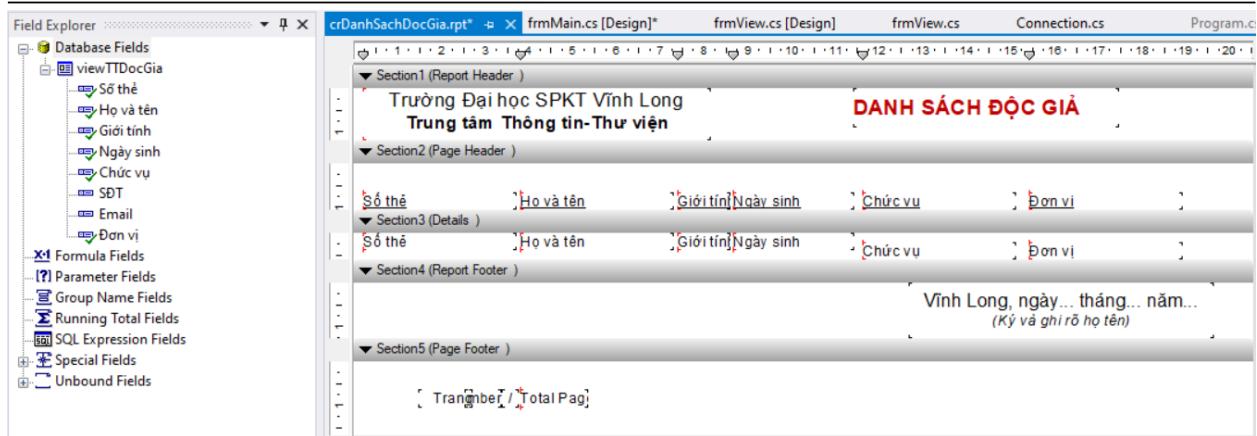
Kết thúc Bước 2.3, refresh My Connections, xuất hiện kết nối và CSDL đã chọn.



Bước 3: Trong CSDL chọn đối tượng dữ liệu cần thêm vào báo biểu (có thể là table, view, store procedure hoặc là một sql command). Sau đó nhấp vào nút ‘>’ để đưa đối tượng vào khung bên phải. Sau đó click OK.



Bước 4: Kéo thả các trường (đối tượng dữ liệu) cần thêm vào trong phần detail của báo biểu. Các Text Objects tương ứng sẽ được tự sinh ra trong phần Page Header.



Formula Fields: đây là các trường tạo thành từ việc thiết lập các công thức. Ví dụ: trường điểm TB của thí sinh = (điểm LT + điểm TH)/2. Tạo mới đối tượng Formula bằng cách sử dụng Formula Editor hoặc Formula Expert.

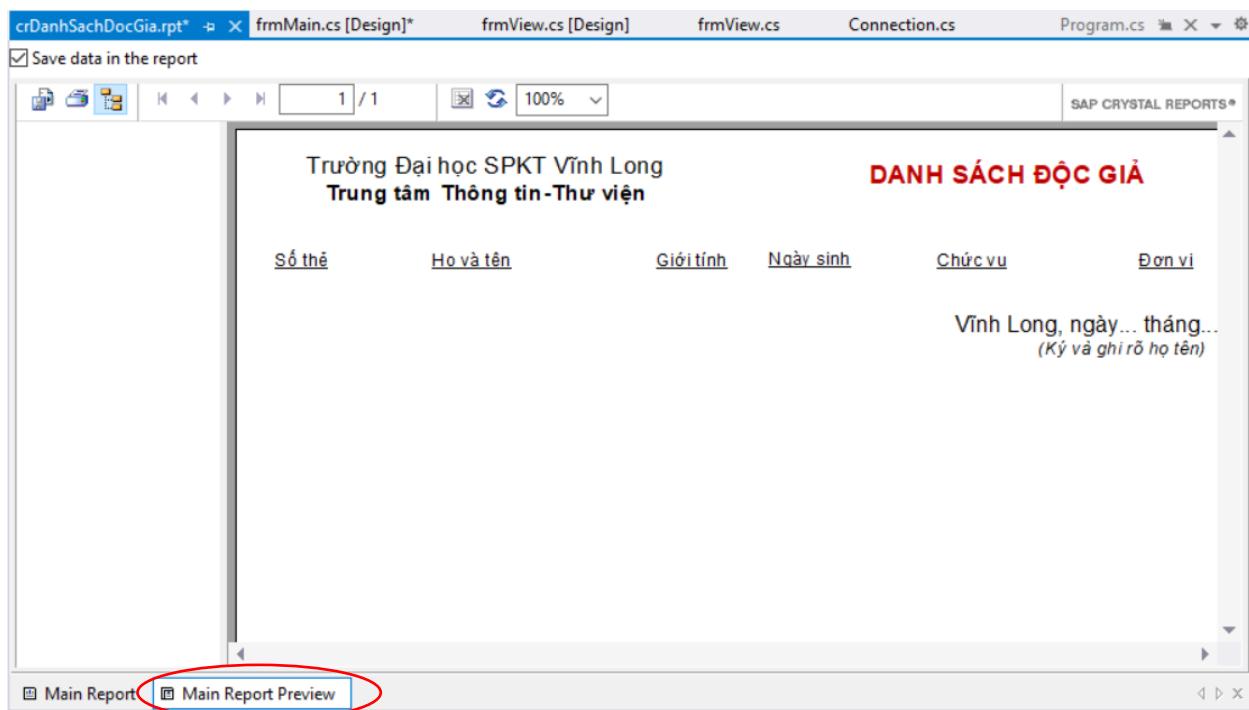
Group Name Fields: xác định dữ liệu trong Detail của báo biểu được gom nhóm theo thuộc tính nào nào.

Special Fields: cho phép chèn thêm các đối tượng như Page N of M, Page Number, Total Page Count, Print Date, Print Time...

Ngoài ra còn một số trường khác như: Parameter Fields, Running Total Fields, Unbound Fields.

c/- Xem trước báo biểu trong quá trình thiết kế

Trong khung thiết kế báo biểu chọn Main Report Preview để xem trước nội dung báo biểu đã thiết kế.



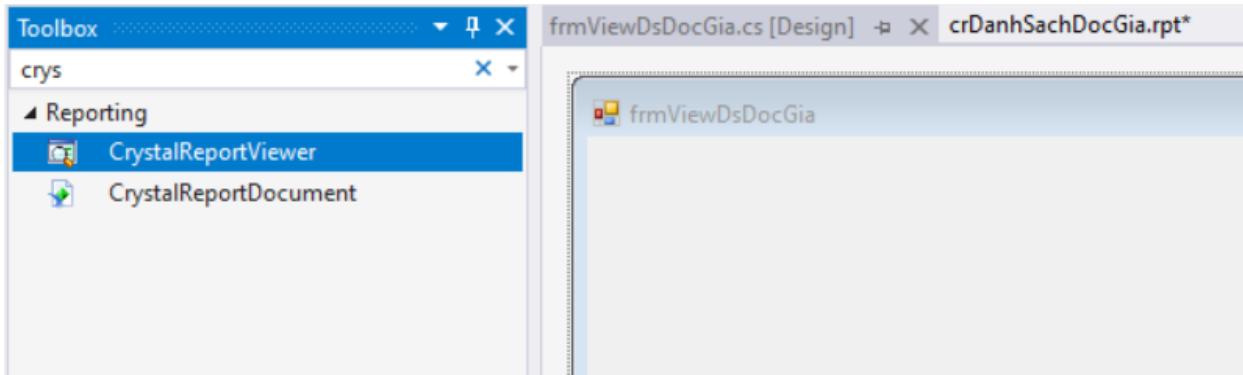
4.3 KẾT NỐI BÁO BIỂU VỚI FORMS

a/- Sử dụng điều khiển CrystalReportViewer

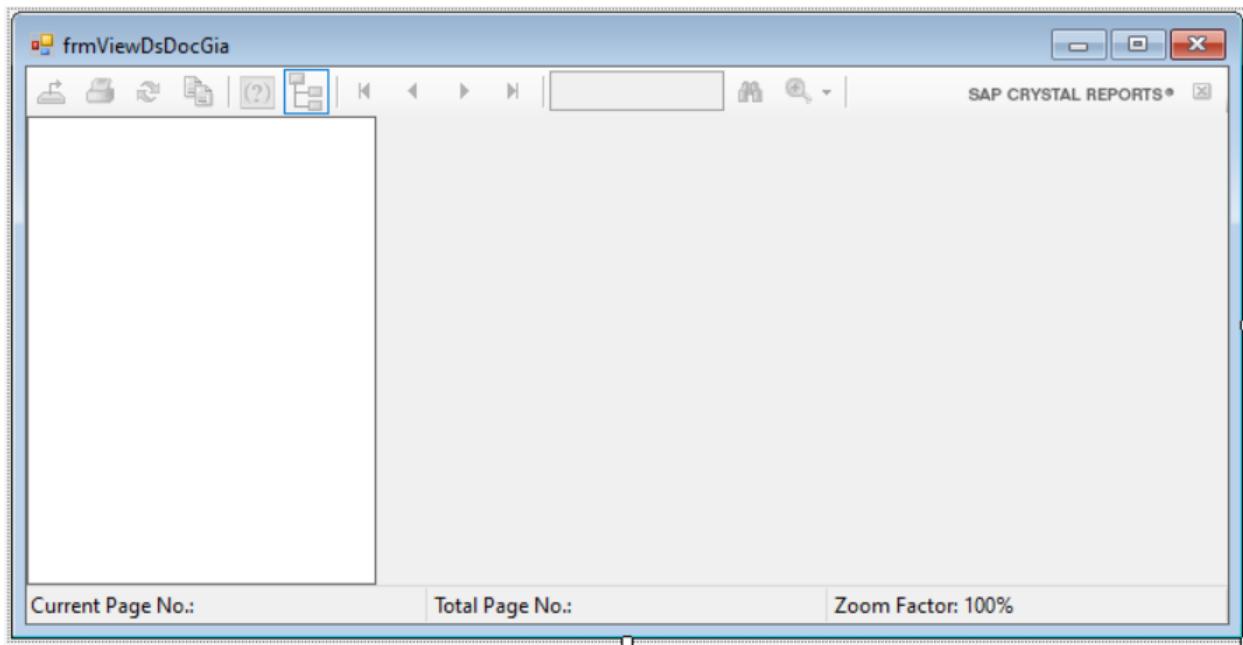
CrystalReportViewer giúp hiển thị các báo biểu được tạo trong ứng dụng.

Bước 1: Trong ứng dụng (dự án), tạo một form mới và đặt lại tên mới frmViewDsDocGia dùng để hiển thị báo biểu.

Bước 2: Trong Toolbox, kéo thả CrystalReportViewer vào form vừa tạo, đổi tên thành crViewer.



Hình 4.2: Kéo thả CrystalReportViewer từ Toolbox vào frmViewDsDocGia



Hình 4.3: Giao diện frmViewDsDocGia sau khi kéo thả CrystalReportViewer vào Form

b/- Tích hợp hiển thị báo biểu trong ứng dụng

Một báo biểu trong ứng dụng được hiển thị thông qua frmView tương ứng bằng cách click vào một nút (Button), một liên kết (LinkLabel), một nhãn (Label) hoặc từ một mục của ToolStrip. Để tích hợp việc hiển thị một báo biểu trong ứng dụng chúng ta sẽ tiến hành xử lý các sự kiện tương ứng với từng điều khiển (control) khác nhau theo yêu cầu.

Ví dụ 4.4: Giả sử có một frmMain với hệ thống menu đã được thiết kế có mục chọn Danh sách độc giả. Xử lý sự kiện click mục chọn Danh sách độc giả cho phép hiển thị frmViewDsDocGia. Trong sự kiện load của frmViewDsDocGia thiết lập nguồn của CrystalReportViewer

```
// Mở frmViewDsDocGia từ menu của frmMain
private void itemDSDocgia_Click(object sender, EventArgs e)
{
```

```
frmViewDsDocGia ob = new frmViewDsDocGia();
ob.MdiParent = this;
ob.Show();
}
// Thiết lập nguồn của CrystalReportViewer trong xự kiện load
của frmViewDsDocGia
private void frmViewDsDocGia_Load(object sender, EventArgs e)
{
    Connection conn = new Connection();
    if (conn.openConn())
    {
        rpDanhSachDocGia rp = new rpDanhSachDocGia();
        string sql = "select * from viewTTDocGia";
        DataTable dt = conn.loadDataTable(sql);
        rp.SetDataSource(dt);
        crvDsDocGia.ReportSource = rp;
    }
    else
    {
        MessageBox.Show("Không thể kết nối CSDL");
    }
}
```

CÂU HỎI VÀ BÀI TẬP

Thiết kế các báo biểu cho mục Thông kê-Báo cáo theo yêu cầu của Hệ thống QUẢN LÝ THƯ VIỆN VLUTE.

Chương 5

HOÀN THIỆN ỨNG DỤNG QUẢN LÝ THÔNG TIN

5.1 THIẾT LẬP AN NINH HỆ THỐNG

5.1.1 Thiết lập thông tin kết nối cơ sở dữ liệu

Bảo vệ truy cập đến nguồn dữ liệu là một trong những mục tiêu quan trọng nhất khi bảo mật một ứng dụng. Một chuỗi kết nối có một lỗ hổng tiềm ẩn nếu nó không được bảo mật. Việc lưu trữ thông tin kết nối dưới văn bản thuần túy hoặc duy trì nó trong bộ nhớ có nguy cơ ảnh hưởng đến toàn bộ hệ thống đã xây dựng. Các chuỗi kết nối được nhúng trong mã nguồn có thể được đọc bằng lldasm.exe (IL Disassembler) để xem ngôn ngữ trung gian của Microsoft (MSIL) trong một mã hợp ngữ đã được biên dịch.

Các lỗ hổng bảo mật liên quan đến các chuỗi kết nối có thể phát sinh dựa trên loại xác thực được sử dụng, cách các chuỗi kết nối được duy trì trong bộ nhớ và trên đĩa cũng như các kỹ thuật được sử dụng để xây dựng chúng tại thời gian chạy. Chúng ta có thể bảo vệ truy cập nguồn dữ liệu với các lưu ý như sau:

Sử dụng Windows Authentication: người dùng được Windows xác thực, truy cập đến các nguồn tài nguyên máy chủ và cơ sở dữ liệu được xác định bằng cách cấp quyền cho nhóm người dùng và người dùng Windows.

Không sử dụng các tập tin liên kết dữ liệu dùng chung (Universal Data Link, UDL): tránh lưu trữ các chuỗi kết nối cho một OleDbConnection trong một tập tin UDL. Các tập tin UDL lưu trữ ở dạng văn bản rõ ràng và không thể được mã hóa. Tập tin UDL là một tài nguyên dựa trên tập tin bên ngoài ứng dụng, và không thể được bảo mật hoặc mã hóa bằng .NET Framework.

Tránh các tấn công Injection (Injection Attack) bằng Connection String Builders: một tấn công Injection có thể xảy ra khi nối chuỗi động được sử dụng để xây dựng chuỗi kết nối dựa trên dữ liệu đầu vào của người dùng. Nếu dữ liệu đầu vào không được xác thực và văn bản hoặc ký tự độc hại không được để ý, kẻ tấn công có thể truy cập dữ liệu nhạy cảm hoặc các tài nguyên khác trên máy chủ. Để xử lý vấn đề này, ADO.NET giới thiệu các lớp connection string builder để xác thực cú pháp chuỗi kết nối và đảm bảo các thông số bổ sung không được đưa vào.

Sử dụng Persist Security Info = False: đây là chế độ bảo mật và được khuyến nghị sử dụng bởi Microsoft để tránh các thông tin nhạy cảm như ID người dùng và mật khẩu có thể bị lấy từ kết nối sau khi nó được mở (khi thiết lập Persist Security Info bằng True hoặc Yes). Khi thiết lập bằng False hoặc No, các thông tin bảo mật được loại bỏ sau khi chúng được dùng cho việc mở kết nối, đảm bảo rằng nguồn không đáng tin cậy không có quyền truy cập vào thông tin nhạy cảm về bảo mật.

Mã hóa các tập tin cấu hình: chúng ta có thể lưu các chuỗi kết nối vào các tập tin cấu hình (configuration files), điều này giúp loại bỏ sự cần thiết phải nhúng chúng vào mã

nguồn ứng dụng. Các tập tin cấu hình là các tập tin XML chuẩn mà .NET Framework đã xác định tập các phần tử thông dụng. Các chuỗi kết nối trong các tập tin cấu hình thường được lưu trữ trong thành phần `<connectionStrings>` trong `app.config` cho ứng dụng Windows và `web.config` trong ứng dụng ASP.NET.

5.1.2 Xây dựng Form xác thực người dùng

Xây dựng Form xác thực người dùng là một yêu cầu không thể thiếu đối với các hệ thống quản lý thông tin. Form xác thực người dùng giúp xác nhận người dùng nào đã đăng nhập vào hệ thống và quyền hạn của họ trong việc tương tác với các chức năng hệ thống.

Các thông tin xác thực người dùng được xem là các dữ liệu nhạy cảm có thể ảnh hưởng đến an ninh hệ thống. Bởi vì, khi thông tin người dùng bị đánh cắp, kẻ tấn công có thể sử dụng chúng cho việc đột nhập vào hệ thống và thực hiện các tác vụ mà người dùng đó được cấp quyền. Chúng ta có thể áp dụng phương pháp mã hóa chuỗi cho việc bảo mật các thông tin này.

5.1.3 Phân chia chức năng của các nhóm người dùng

Việc phân chia chức năng cho từng nhóm người dùng cụ thể được xem là một trong những biện pháp giúp đảm bảo an ninh hệ thống. Một người dùng hay nhóm người dùng sẽ chịu trách nhiệm về những thay đổi trong dữ liệu, các nguồn tài nguyên có liên quan đến các chức năng mà họ được cấp quyền. Đặc biệt đối với các chức năng liên quan đến nghiệp vụ chuyên môn yêu cầu phải có những kiến thức, kỹ năng và kinh nghiệm chuyên biệt trong việc xử lý. Phân chia chức năng không đúng đối tượng hoặc nhóm đối tượng có thể gây nguy hại cho bảo mật ứng dụng, và xa hơn sẽ ảnh hưởng đến quy trình nghiệp vụ cũng như đem đến những thiệt hại không hề nhỏ cho các cơ quan, tổ chức chủ quản.

5.1.4 Phân quyền truy xuất trong cơ sở dữ liệu

Phân quyền truy xuất trong cơ sở dữ liệu là một giải pháp cho phép:

- Ngăn chặn các truy cập bất hợp pháp
- Hạn chế tối đa những sai sót của người dùng
- Đảm bảo dữ liệu không bị mất và thay đổi ngoài ý muốn
- Không tiết lộ nội dung dữ liệu cũng như chương trình xử lý

Quản trị viên cơ sở dữ liệu có thể phân quyền hạn cho người dùng hoặc nhóm người dùng với các chức năng `INSERT`, `UPDATE`, `DELETE` hoặc `SELECT` dữ liệu trên các bảng cụ thể tương ứng với các chức năng ứng dụng mà họ đã được phân chia.

Hiện nay các hệ quản trị cơ sở dữ liệu đều có hỗ trợ chức năng phân quyền truy cập cho các nhóm người dùng hoặc người dùng nhằm giúp tăng cường bảo mật và truy vết tác động của người dùng đối với dữ liệu.

5.2 MENU VÀ PHÂN QUYỀN AN NINH TRÊN MENU

5.2.1 Xây dựng hệ thống menu

Các ứng dụng thông thường, đặc biệt là các ứng dụng quản lý hệ thống đều có nhu cầu phân quyền cho các người dùng (user) hay các nhóm người dùng (group) nghĩa là người dùng sau khi đăng nhập vào ứng dụng sẽ chỉ được phép sử dụng một số Form nhất định (Phân quyền theo Form) và những chức năng cụ thể Form đó (phân quyền theo chức năng).

Việc xây dựng hệ thống menu cần đảm bảo gồm nhóm các chức năng chính của hệ thống. Từ đó có thể thực hiện phân quyền an ninh trên menu cho các người dùng hay các nhóm người dùng tương ứng với từng chức năng. Ví dụ, với quản trị viên của hệ thống thông tin quản lý đào tạo họ sẽ có quyền truy cập vào các chức năng quản trị hệ thống như thêm người dùng, phân nhóm người dùng, cấp quyền truy cập. Tuy nhiên, quản trị viên sẽ không được quyền truy cập đến các nghiệp vụ trong quản lý đào tạo như xếp thời khóa biểu, quản lý điểm của người học.

5.2.2 Phân quyền theo Forms

Một người dùng hay một nhóm người dùng thường được phân quyền sử dụng một số chức năng nào đó của ứng dụng, các chức năng này thường được kích hoạt bởi các mục chọn trong menu và mở các Form cụ thể tương ứng. Thông thường sau khi người dùng đăng nhập chỉ những mục chọn tương ứng với chức năng được cấp cho người dùng mới được hiển thị, các mục chọn khác có thể được ẩn đi hoặc vô hiệu hóa. Trong trường hợp tốt nhất, các mục chọn không được cấp cho người dùng nên bị ẩn đi để tránh khơi gợi sự tò mò của người dùng từ đó có thể dẫn đến những mối nguy hiểm ngoài ý muốn cho ứng dụng cũng như hệ thống quản lý.

5.2.3 Phân quyền theo chức năng

Trong một số trường hợp khác, các người dùng hay các nhóm người dùng có thể truy cập đến cùng một chức năng (form) nhưng trong đó họ sẽ được phép thao tác với các phương thức khác nhau, khi đó việc phân quyền trên menu sẽ trở nên bất tiện vì cứ mỗi người dùng hoặc nhóm người dùng chúng ta phải tạo một form riêng cho họ nhưng trong thực tế thì không cần phải như thế. Chúng ta có thể cho tất cả các người dùng hoặc nhóm người dùng này có thể vào cùng một form nhưng chỉ những phương thức tương ứng với người dùng mới được kích hoạt, các phương thức khác sẽ bị vô hiệu hóa hoặc ẩn dấu đi.

5.3 ĐÓNG GÓI VÀ TRIỂN KHAI

Chúng ta có thể sử dụng Windows Application Packing Project trong Visual Studio để đóng gói một ứng dụng desktop. Sau đó, có thể phân phối gói này lên Microsoft Store, Web trong công ty hoặc bất kỳ cơ chế phân phối nào khác.

Windows Application Packing Project có giá trị trong các bản Visual Studio 2019 và Visual Studio 2017 15.5 trở lên.

Để thấy Windows Application Pakaging Project trong “Add New Project”, chúng ta cần cài đặt ít nhất một trong các Visual Studio workload sau:

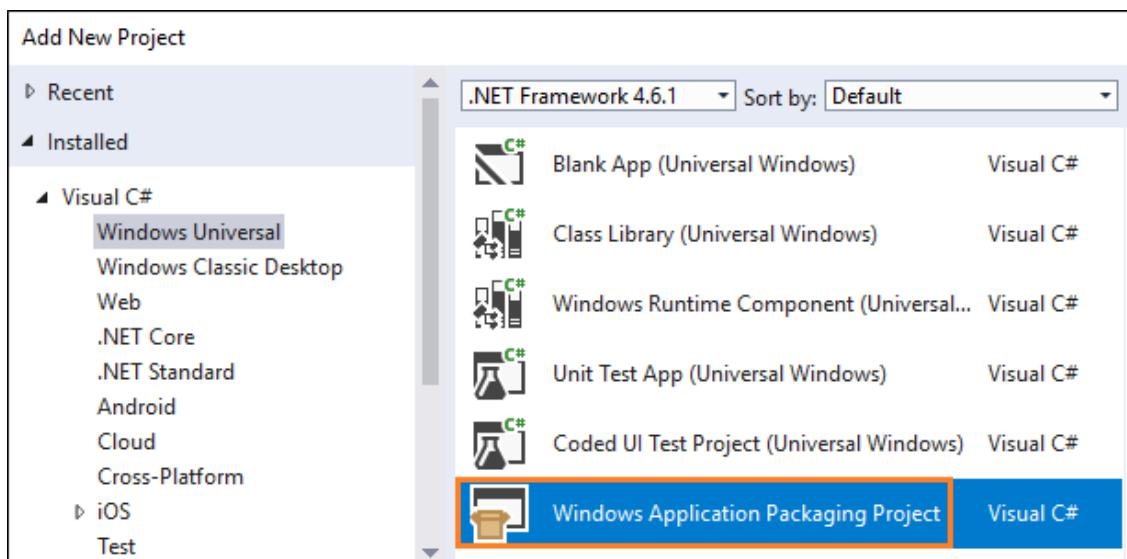
- Universal Windows Platform development workload
- Thành phần tùy chọn “MSIX Packing Tools” trong .NET Core workload
- Thành phần tùy chọn “MSIX Packing Tools” trong .NET desktop development workload.

a/- Thiết lập Windows Application Packaging Project trong Solution đã có

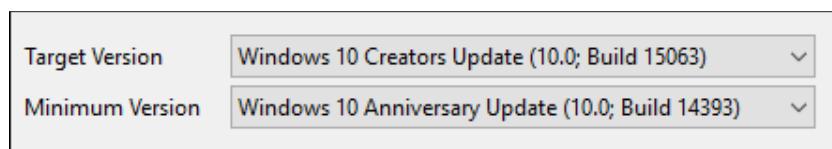
Bước 1: Trong Visual Studio, mở Solution có chứa Project ứng dụng.

Bước 2: Tạo mới 1 project có kiểu Windows Application Packaging Project trong Solution.

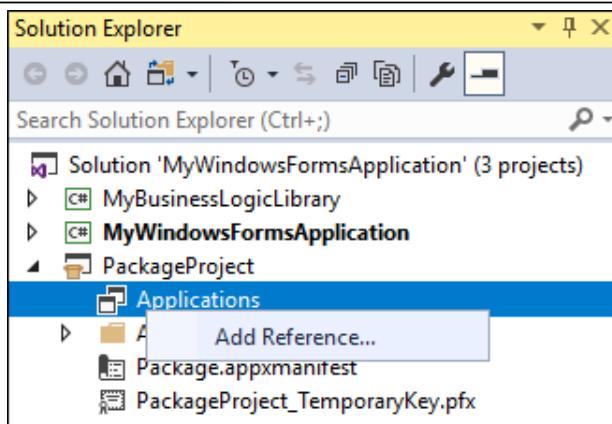
Lưu ý, chúng ta sẽ không phải thêm bất kỳ mã nguồn nào vào project này. Nó chỉ ở đó để tạo ra gói cài đặt cho ứng dụng của chúng ta và được xem là một “packaging project”).



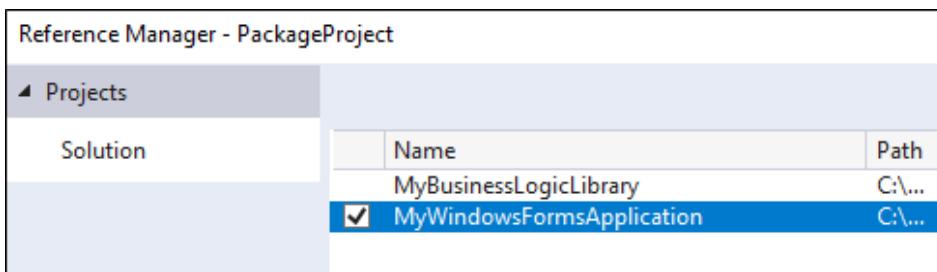
Bước 3: Thiết lập Target Version của project này thành bất cứ phiên bản nào mà chúng ta muốn, nhưng phải đảm bảo thiết lập Minimum Version là Windows 10 Anniversary Update.



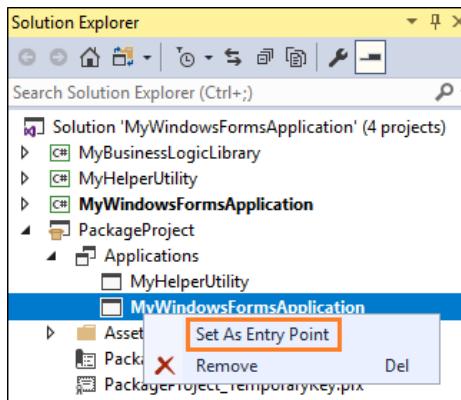
Bước 4: Trong Solution Explorer, nhấp chuột phải trên thư mục Applications trong packing project và chọn Add Reference.



Bước 5: Chọn project tương ứng với ứng dụng cần đóng gói, và sau đó chọn nút OK.



Chúng ta có thể gom nhiều ứng dụng vào trong gói cài đặt, nhưng chỉ có một trong số chúng sẽ khởi động khi người dùng chọn chạy ứng dụng. Trong nút Applications, nhấp chuột phải trên ứng dụng muốn thiết lập thành ứng dụng khởi động mặc định và sau đó chọn Set as Entry Point.



Bước 6: Chính sửa tập tin kê khai (manifest editor), visual studio cung cấp trình chỉnh sửa tập kê khai được tích hợp sẵn giúp chúng ta thiết lập danh tính, khả năng... của ứng dụng bằng giao diện trực quan.

Để truy cập vào trình chỉnh sửa tập kê khai, chúng ta chỉ cần nhấp đúp vào tập tin Package.appxmanifest có trong dự án Windows Application Packing.

Nếu chúng ta đóng gói các ứng dụng Windows cổ điển, các phần có liên quan là:

Visual Asset, giúp tạo các icon phù hợp để sử dụng trong Windows 10.

Packaging, dùng để thiết lập danh tính bằng các xác định tên gói, tên nhà xuất bản...

Declarations, cho phép chúng ta thêm các điểm mở rộng để tích hợp ứng dụng với Windows 10.

Application **Visual Assets** **Capabilities** **Declarations** **Content URLs** **Packaging**

Use this page to set the properties that identify and describe your app.

Display name: Expenselt-Test

Entry point: \$targetentrypoint\$

Default language: en-US [More information](#)

Description: ExpenseltDemo.Package

Supported rotations: An optional setting that indicates the app's orientation preferences.

Landscape Portrait Landscape-flipped Portrait-flipped

Lock screen notifications: (not set)

Resource group:

Tile Update:

Updates the app tile by periodically polling a URI. The URI template can contain "{language}" and "{region}" tokens that will be replaced at runtime to generate the URI to poll.

[More information](#)

Recurrence: (not set)

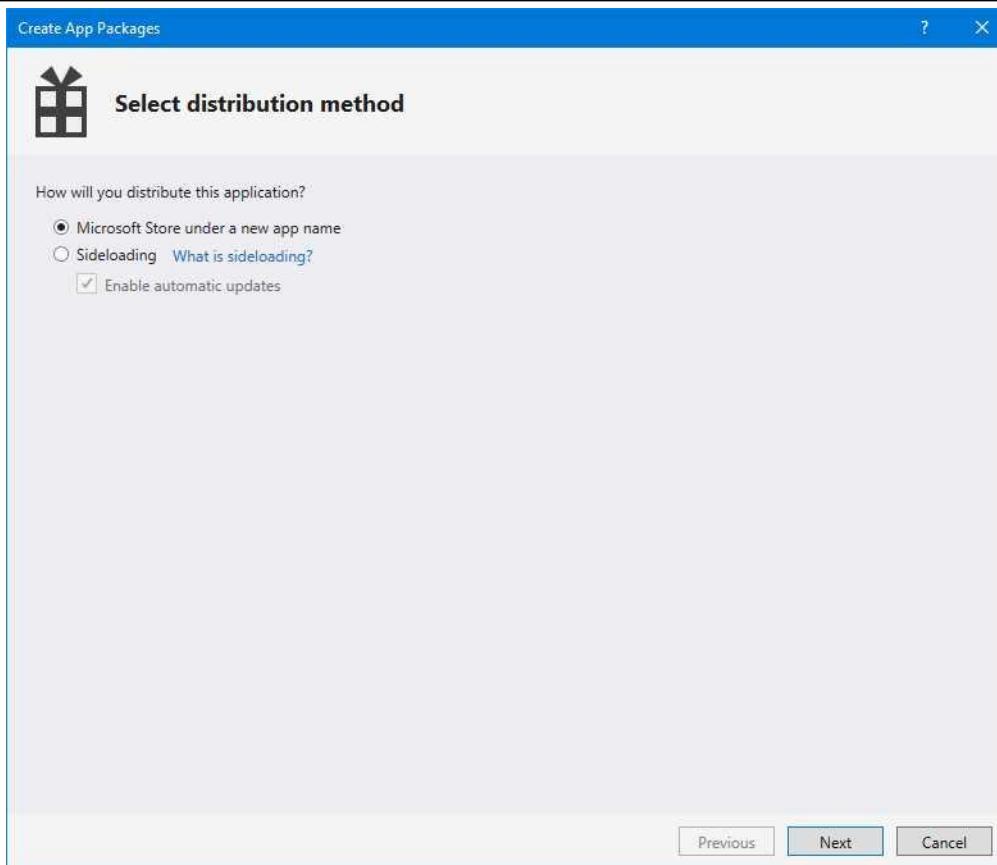
URI Template:

Bước 7: Build packing project để đảm bảo không có lỗi xảy ra. Nếu có lỗi xuất hiện, mở Configuration Manager và đảm bảo các project nhắm tới cùng một platform.

Configuration Manager				
Active solution configuration:		Active solution platform:		
Project contexts (check the project configurations to build or deploy):				
Project	Configuration	Platform	Build	Deploy
Landmarks	Debug	x86	<input checked="" type="checkbox"/>	<input type="checkbox"/>
LandmarksPackage	Debug	x86	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
LandmarksRuntimeComponent	Debug	x86	<input checked="" type="checkbox"/>	<input type="checkbox"/>
LandmarksUWP	Debug	x86	<input checked="" type="checkbox"/>	<input type="checkbox"/>

Bước 8: Dùng Create App Packages wizard để tạo MSIX package/bundle hoặc 1 tập tin .msixupload/.appxupload (để xuất bản lên Store).

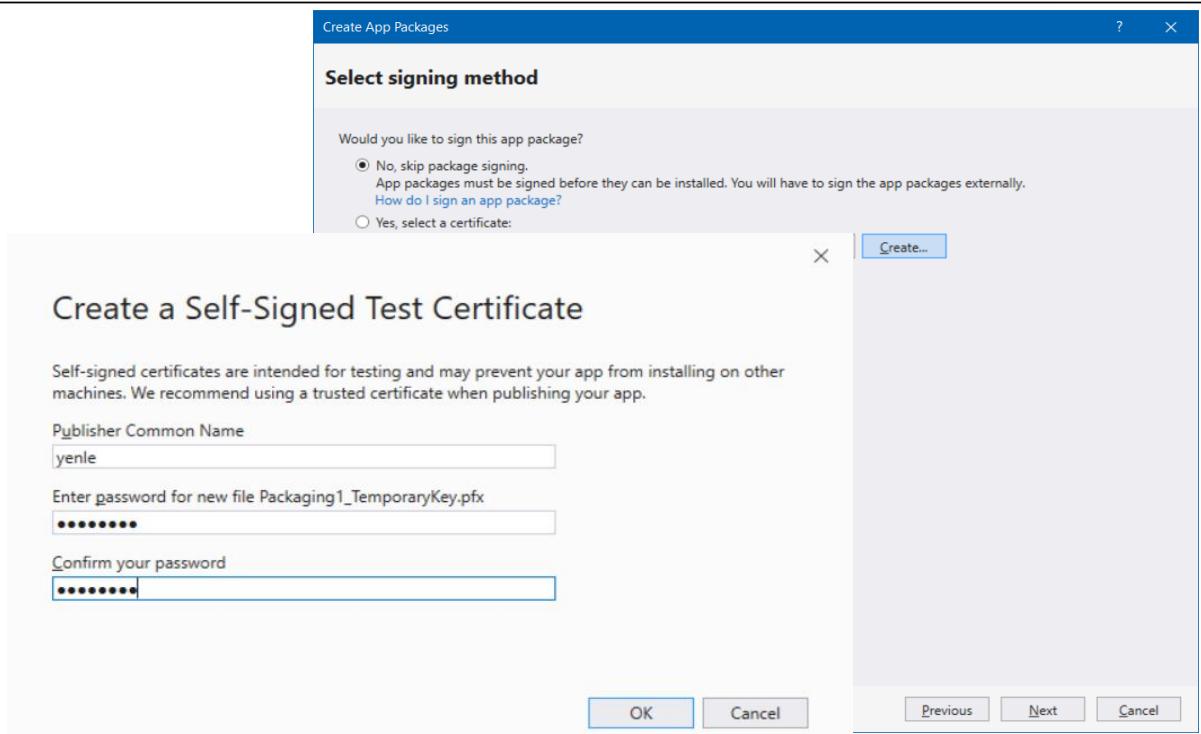
Nhấn chuột phải trên Windows Application Packaging Project và chọn Publish (Store) > Create app packages



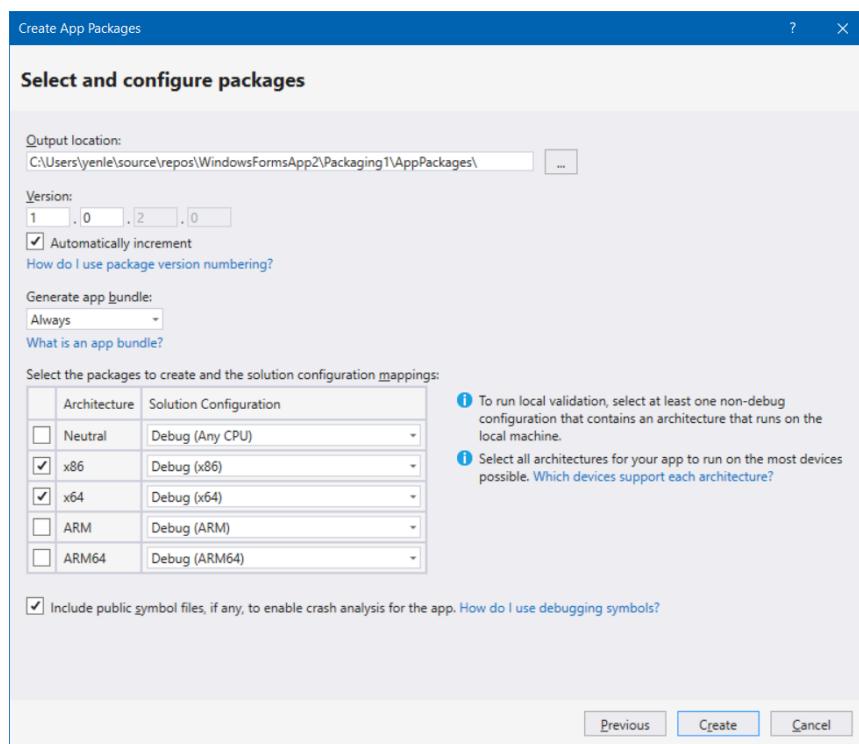
Trước khi bắt đầu tiến trình, chúng ta phải chọn phương thức ứng dụng được phân phối: xuất bản (publish) ứng dụng lên Microsoft Store hoặc phân phối sideload (cài đặt phần mềm ngoài store) cho việc triển khai thử công hoặc cài đặt tự do.

Nếu chúng ta chọn Sideload, ở bước tiếp theo, Create App Packages wizard sẽ yêu cầu chúng ta chọn phương pháp ký cho ứng dụng nếu chúng ta muốn ký gói cài đặt với một trusted certificate.

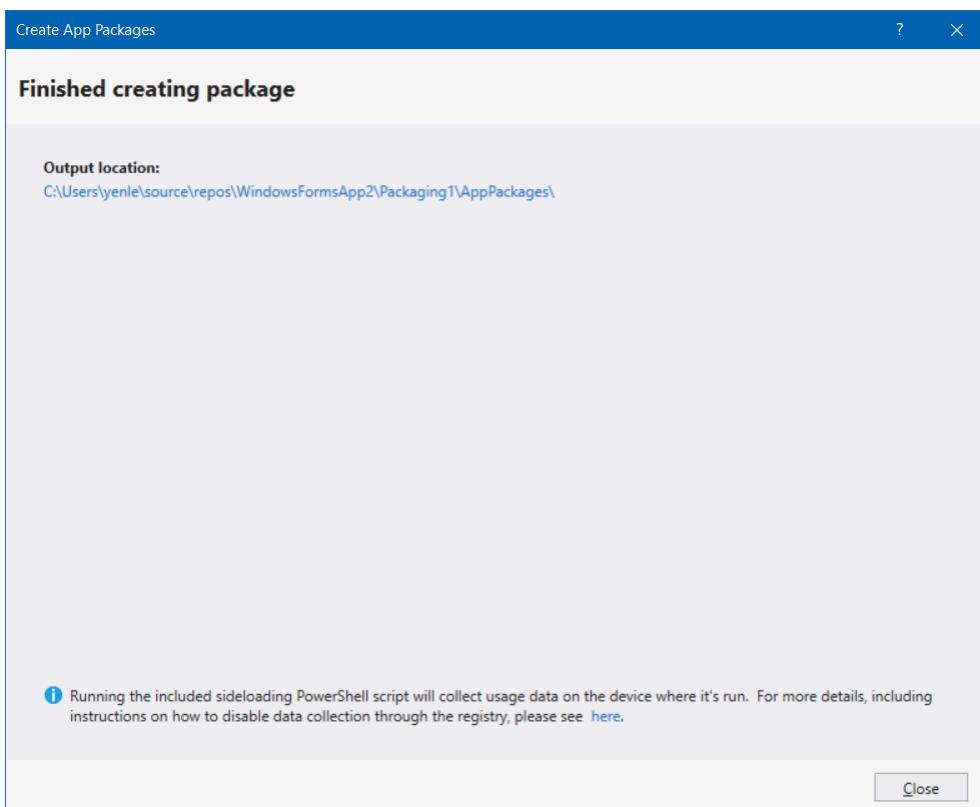
- Chọn Yes, select a certificate
- Nhấn Create để tạo certificate mới (nếu chưa có)
- Sau đó nhập mật khẩu để tạo certificate mới.



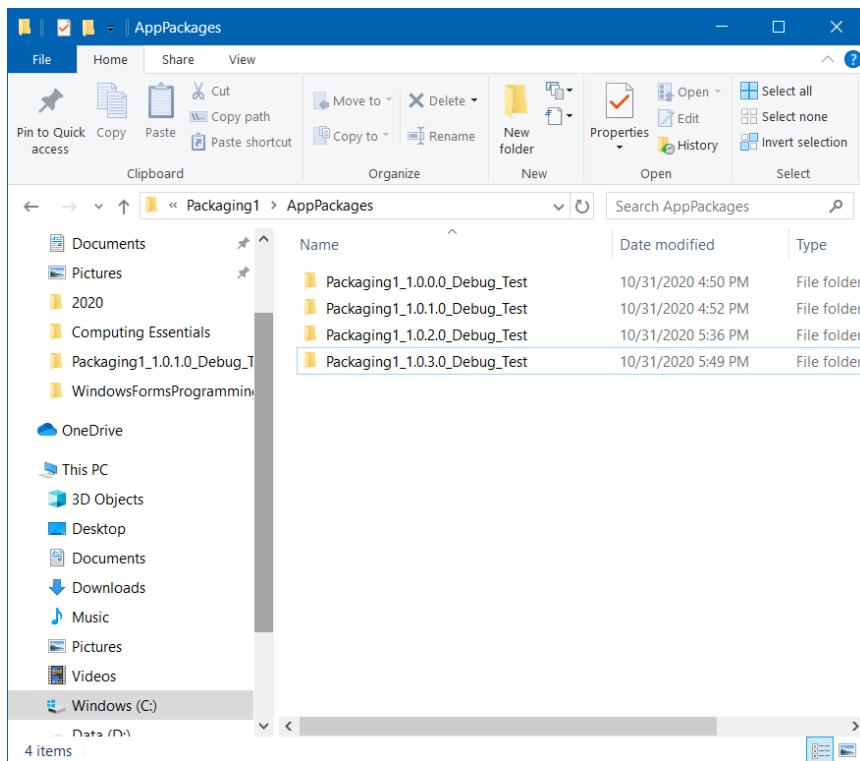
Ở phần tiếp theo, chúng ta sẽ có tùy chọn để cấu hình gói cài đặt. Đầu tiên, chúng ta sẽ xác định thư mục chứa gói MSIX sau khi tạo. Mặc định, nó sẽ là thư mục AppPackages trong Windows Application Packaging Project. Tiếp theo, chúng ta có thể định nghĩa số thứ tự phiên bản và chọn có muốn tạo app bundle không (khuyến nghị nên dùng Always). Cuối cùng, chúng ta sẽ định nghĩa các gói cài đặt muốn tạo dựa trên các kiến trúc khác nhau mà chúng ta muốn hỗ trợ. Khi chúng ta đóng gói một ứng dụng desktop cổ điển, chúng ta có thể chọn hỗ trợ các kiến trúc x86 và x64.



Nhấn chọn Create để tiến hành đóng gói ứng dụng. Kết quả sau khi hoàn thành sẽ như sau.



Kết quả thu được trong thư mục của Windows Application Packing Project.



CÂU HỎI VÀ BÀI TẬP

Tăng cường bảo mật cho hệ thống **QUẢN LÝ THƯ VIỆN VLUTE**.

Đóng gói và triển khai hệ thống **QUẢN LÝ THƯ VIỆN VLUTE**