

CHƯƠNG I. GIỚI THIỆU C# VÀ .NET FRAMEWORK

1.1. Nền tảng .NET (.NET platform)

.NET Platform là nền tảng phát triển:

-  Cung cấp giao diện lập trình (API) cho các dịch vụ (services) và các hàm API truyền thống của hệ điều hành Windows.
-  Cung cấp một nền tảng phát triển chung cho nhiều ngôn ngữ lập trình khác nhau của Microsoft: C#, Visual J#, Visual Basic...

.NET Platform bao gồm các sản phẩm:

-  Tập hợp các ngôn ngữ (C#, VB...), một tập hợp các công cụ phát triển bao gồm Visual Studio .NET, thư viện để phát triển các ứng dụng web (web and web services), các ứng dụng Windows cũng như môi trường thực thi chung (Common Language Runtime (CLR)).
-  Nền tảng .NET 2.0 không chỉ hỗ trợ các ứng dụng trên máy tính cá nhân, máy chủ mà còn hỗ trợ các thiết bị nhúng (Cell phones, game boxes...).

1.2. Kiến trúc .NET (.NET Framework)

Microsoft .NET hỗ trợ nhiều ngôn ngữ khác nhau nhờ hệ thống - Common Type System (CTS).

Quy định những yêu cầu tối thiểu để một ngôn ngữ lập trình có thể tích hợp vào .NET – Common Language Specification (CLS).

Kiến trúc .NET là tầng ngay trên hệ điều hành, bao gồm:

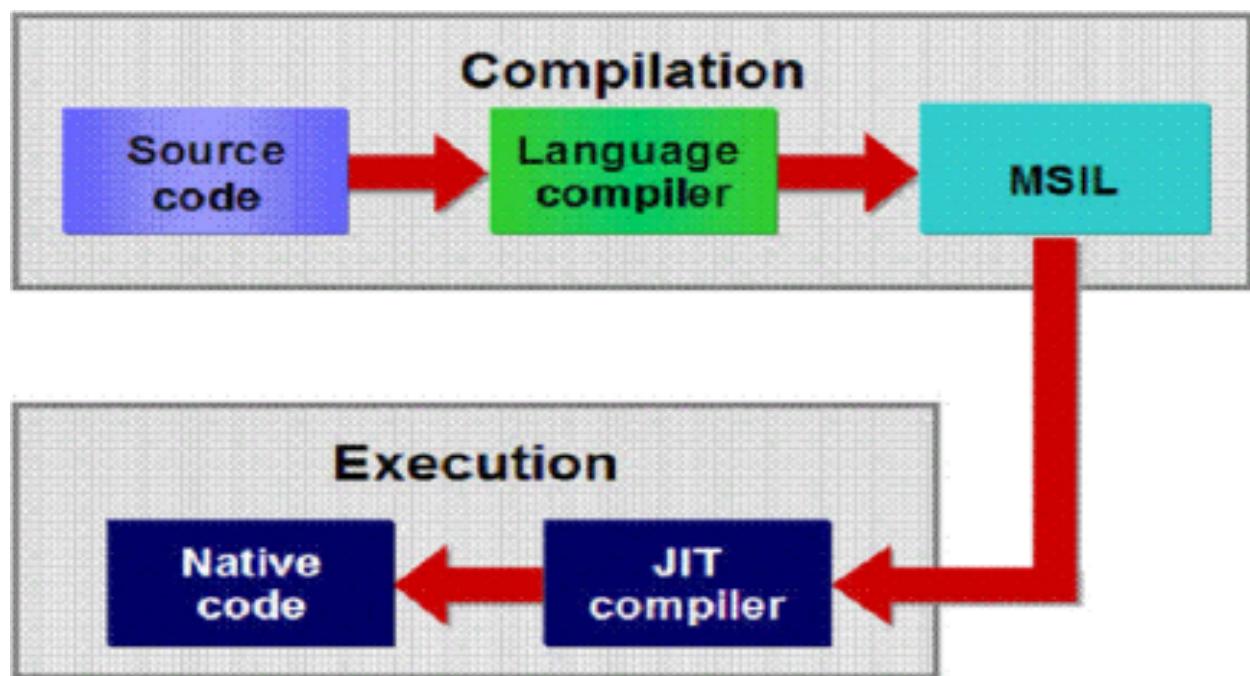
-  5 ngôn ngữ lập trình chính thức: C#, VB, C++, Visual J# và Jscript.NET.
-  CLR (Common Language Runtime).
-  Các thư viện lập trình.

1.3.Biên dịch trong .NET

.NET không biên dịch trực tiếp các chương trình thành file thực thi.

.NET biên dịch các chương trình thành các assembly, chứa các mã chương trình trung gian của Microsoft (Microsoft Intermediate Language - MSIL).

CLR sẽ dịch một lần nữa, sử dụng chương trình biên dịch Just In Time (JIT) chuyển các mã MSIL sang mã máy và thực thi.



Hình 1.1: Biên dịch mã nguồn trong .NET

1.4.Ngôn ngữ lập trình C#

C# được phát triển bởi nhóm tác giả điều hành bởi Anders Hejlsberg và Scott Wiltamuth, tác giả của Turbo Pascal và Borland Delphi.

C# là ngôn ngữ lập trình hướng đối tượng (Object Oriented Language):

-  Hỗ trợ định nghĩa và làm việc với lớp (class).
-  Hỗ trợ đầy đủ ba cơ chế đặc trưng của lập trình hướng đối tượng: Đóng gói, kế thừa và đa hình.

CHƯƠNG II. GIỚI THIỆU MÔI TRƯỜNG PHÁT TRIỂN ỨNG DỤNG VISUAL STUDIO .NET 2012

2.1. Môi trường phát triển ứng dụng .NET

Môi trường thực thi ứng dụng .NET: Microsoft .NET Framework

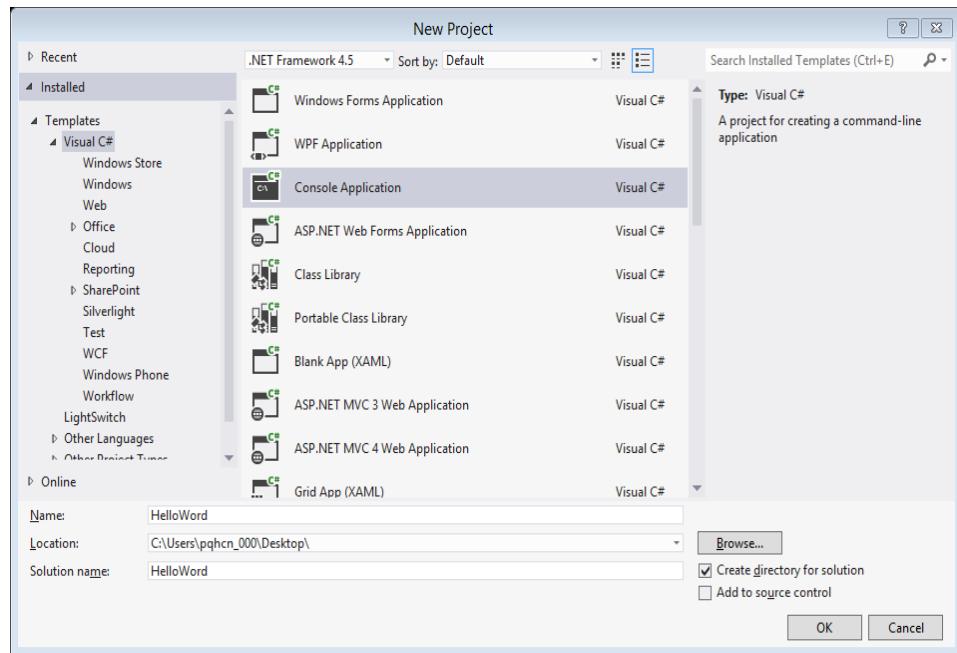
<http://www.microsoft.com/downloads/>

Trình soạn thảo và biên dịch:

-  Visual studio .NET IDE
-  Trình soạn thảo văn bản (Notepad, UltraEdit ...) và trình biên dịch bằng dòng lệnh (Command – line compiler).

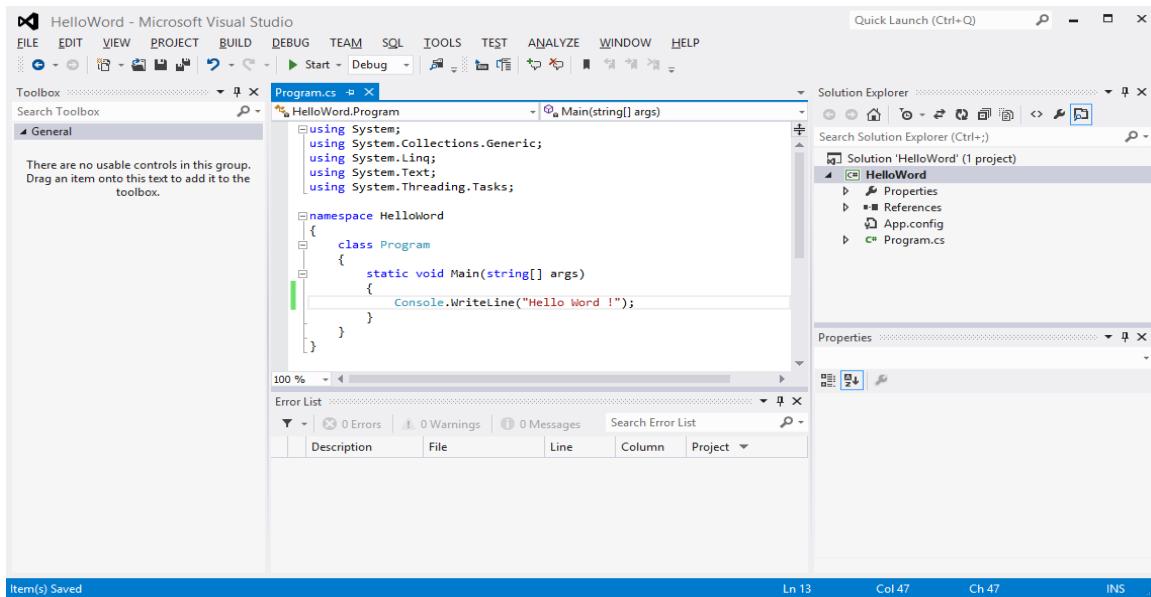
2.2. Chương trình hello Word

- 1. Mở chương trình Visual Studio .NET 2012*
- 2. Tạo một dự án mới với kiểu ứng dụng console (Hello Word)*



Hình 2.1: Tạo ứng dụng Console application đầu tiên

-  Chương trình sẽ tự tạo ra một khung dự án.
-  Thêm một dòng code đơn giản hiển thị dòng: “Hello Word” ra màn hình.
-  Bấm F6 để biên dịch chương trình, F5 để chạy.



Hình 2.2: Biên dịch chương trình

Kết quả sau khi chạy chương trình:



Hình 2.3: Kết quả khi biên dịch chương trình

3. Một số khái niệm cơ bản trong C#



Chú thích:

- ❖ Chú thích trên một dòng: //

VD: // đây là chú thích trên một dòng

- ❖ Chú thích trên nhiều dòng: /* */

VD: /* Đây là chú thích trên nhiều dòng

Dòng chú thích đầu tiên bắt đầu sau dấu “/*”

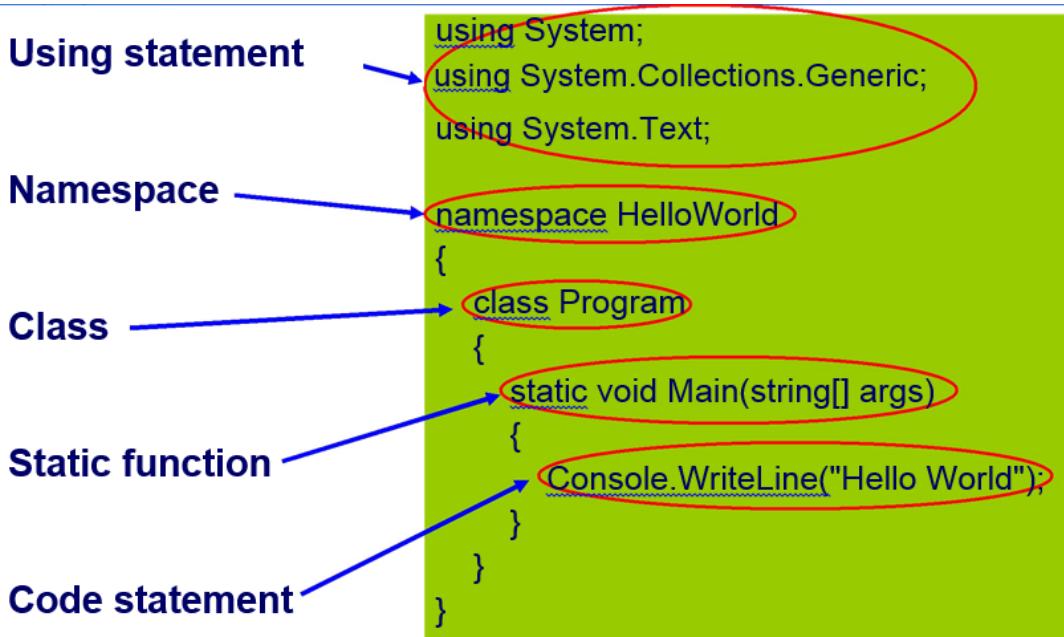
Và kết thúc đoạn chú thích khi gặp dấu “*/” */

- ❖ Trình biên dịch bỏ qua chủ thích.

- ❖ Chỉ dùng cho người đọc.

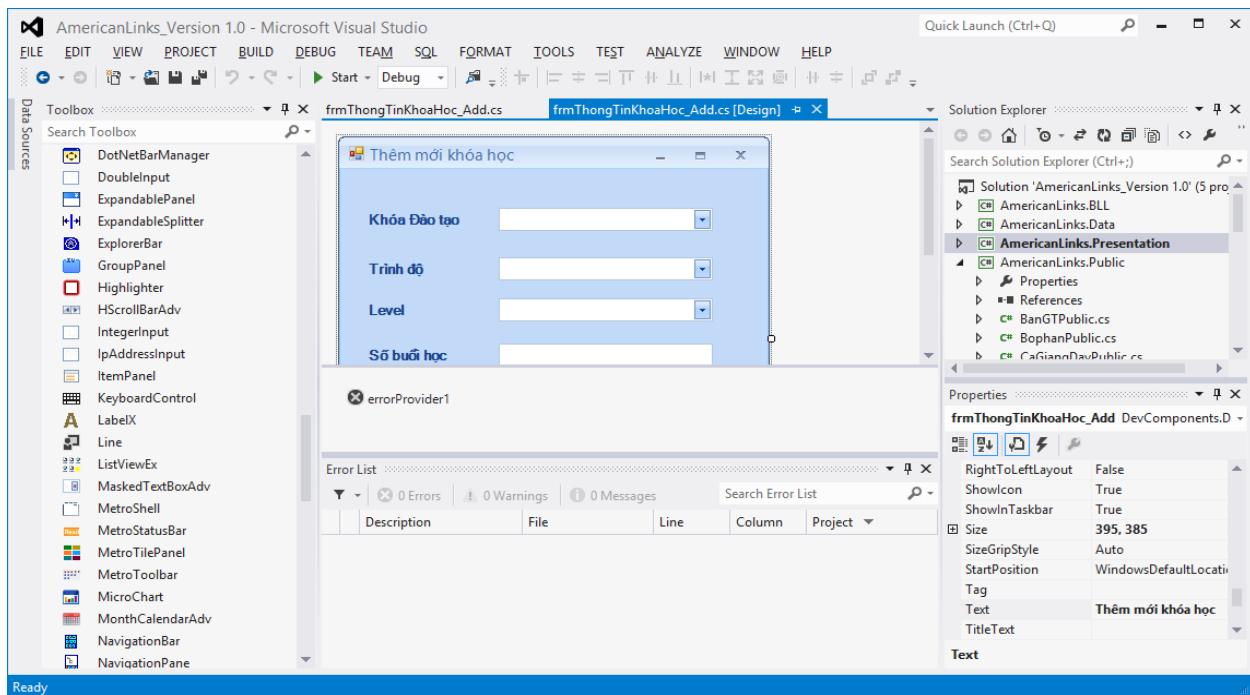
-  Namespaces (không gian tên)
 - ❖ Nhóm các tính năng có liên quan của c# vào một loại.
 - ❖ Cho phép dễ dàng tái sử dụng mã.
 - ❖ Trong thư viện .NET Framework có nhiều không gian tên.
 - ❖ Phải tham chiếu tới để sử dụng.
-  Ký tự cách trắng: chứa các khoảng trắng, ký tự xuống dòng và tabs.
-  Từ khóa (keywords)
 - ❖ Các từ không được dùng làm tên biến, tên lớp hay bất kỳ thứ gì khác.
 - ❖ Có các chức năng đặc biệt không thể thay đổi trong ngôn ngữ.
 - ❖ Tất cả các từ khóa đều được viết thường.
-  Lớp (Class): định nghĩa một kiểu dữ liệu, mô tả một nhóm các đối tượng với các phương thức và thuộc tính.
 - ❖ Phương thức (Method).
 - ❖ Thuộc tính (Property).

Ví dụ: Chương trình “Hello word” thể hiện các khái niệm cơ bản trong c#.



Hình 2.4: Cấu trúc một class trong C#

2.3. Môi trường phát triển ứng dụng Visual Studio .NET



Hình 2.5: Giao diện làm việc trong visual studio

Giao diện làm việc chính.

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

namespace HelloWorld
{
    class Program
    {
        static void Main(string[] args)
        {
            int i;
            for (i = 0; i <= 10; i++)
            {
                Console.WriteLine("Gia tri hien tai: {0}", i);
            }
        }
    }
}
```

Hình 2.6: Chạy chương trình ở chế độ debug

- Nhấn F9 hoặc Click chuột trái vào bên lề trái của hàng cần đặt Breakpoint
- Bắt đầu quá trình Debug: nhấn F5 (Debug => Start debugging)

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

namespace HelloWorld
{
    class Program
    {
        static void Main(string[] args)
        {
            int i;
            for (i = 0; i <= 10; i++)
            {
                Console.WriteLine("Gia tri hien tai: {0}", i);
            }
        }
    }
}
```

Hình 2.7: Gỡ lỗi sử dụng VS.NET 2012

- ⊕ Xem giá trị của các biến, đối tượng

```
static void Main(string[] args)
{
    int i;
    for (i = 0; i <= 10; i++)
    {
        i
        Console.WriteLine("Gia tri hien tai: {0}", i);
    }
}
```

Hình 2.8: Xem giá trị của biến khi chạy debug chương trình 1

- ⊕ Xem giá trị các biến, đối tượng tại cửa sổ local

Name	Value	Type
args	{Dimensions:[0]}	string[]
i	1	int

Locals Watch 1

Hình 2.9: Xem giá trị của biến khi chạy debug chương trình 2

CHƯƠNG III. C# CĂN BẢN

3.1. Kiểu dữ liệu (Type)



C# là ngôn ngữ định kiểu mạnh

- ❖ Phải định nghĩa kiểu dữ liệu của mỗi đối tượng (ví dụ: integers, buttons ...).
- ❖ Trình biên dịch sẽ kiểm tra và đảm bảo ngăn chặn các lỗi do các phép gán sai kiểu dữ liệu.



Phân loại dữ liệu

- ❖ Phân loại theo phương thức định nghĩa: kiểu dữ liệu có sẵn (build - in) và kiểu dữ liệu do người dùng tự định nghĩa (user - defined).
- ❖ Phân theo các thức lưu trữ: tham trị (value) và tham chiếu (reference).
- ❖ Kiểu tham trị: lưu trữ trong vùng nhớ ngăn xếp (stack).
- ❖ Kiểu tham chiếu: địa chỉ được lưu trữ trong vùng nhớ ngăn xếp, dữ liệu thực được lưu trữ trong vùng nhớ Heap → cho phép lưu trữ đối tượng với kích thước lớn.



C# hỗ trợ một số kiểu dữ liệu có sẵn, mỗi kiểu dữ liệu này tương ứng với một kiểu dữ liệu hỗ trợ bởi .NET CLS (Common Language System).



C# có thể sử dụng đối tượng do các ngôn ngữ khác trong bộ .NET tạo ra và ngược lại (ví dụ VB.NET).



Mỗi kiểu dữ liệu có kích thước xác định.

Type	Size	.NET type	Description
Byte	1	byte	Unsigned (value 0 - 255)
Char	2	Char	Unicode characters
Bool	1	Boolean	True or False
sbyte	1	sbyte	Signed (values -128 to 127)
short	2	Int16	Sign (short) (value -32768 to 32767)

Ushort	2	Uint16	UnSign (short) (value 0 to 65535)
Int	4	Int32	Signed integer value between -2,147,483,648 and 2,147,483,647
Uint	4	uint32	Unsigned integer value between 0 and 4,294,967,295
Float	4	single	Floating – point number. Holds the values from approximately $+/-1.5*10^{-45}$ to approximately $+/-3.4*10^{38}$
double	8	double	Double – precision floating point. Holds the values from approximately $+/-5.0*10^{-324}$ to approximately $+/-1.8*10^{308}$ with 15 -16 significant figures.
decimal	16	decimal	Fixed – precision up to 28 digits and the position of the decimal point. This is typically used in financial caculation. Requires the suffix “m” or “M”
Long	8	int64	Signed integers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
ulong	8	uint64	Unsigned integers ranging form 0 to 0xffffffffffffffff



Escape character

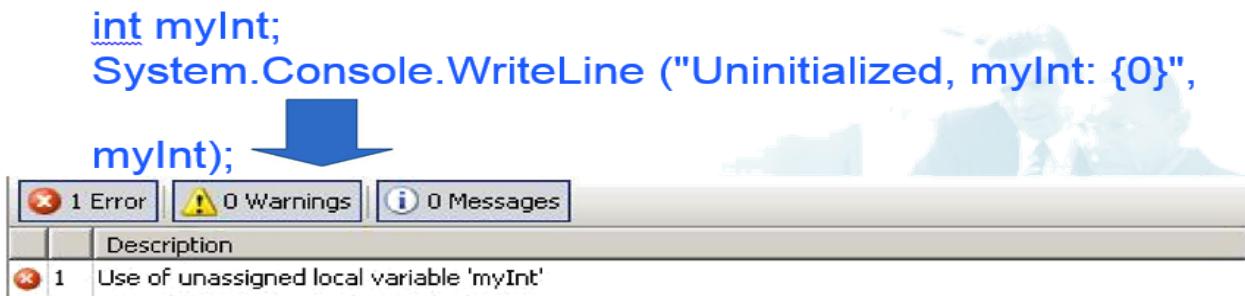
Char	Meaning
\'	Single quote
\”	Double quote
\\	Backslash
\0	Null
\a	Alert
\b	Backspace
\f	Form feed
\n	New line
\r	Carriage return
\t	Horizontal tab

\v	Vertical tab
----	--------------

- ❖ Chuyển đổi kiểu dữ liệu: có 2 cách chuyển đổi kiểu dữ liệu có sẵn:
- ❖ Chuyển đổi ngầm (implied conversion): quá trình chuyển đổi diễn ra tự động và đảm bảo không mất mát dữ liệu. VD: `int x = 2; short y = 3;`
 - ❖ Chuyển đổi tường minh (explicit conversion): sử dụng toán tử chuyển đổi (cast operator). VD: `(char) 65 => chuyển đổi từ số sang ký tự`

3.2. Biến và hằng số (Variables and Constants)

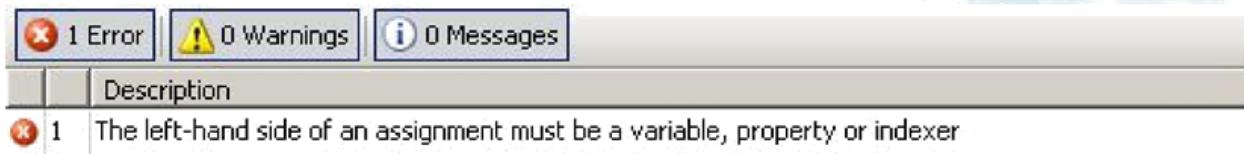
- ❖ Biến (variable):
- ❖ Một vùng nhớ có định kiểu.
 - ❖ Có thể gán và thay đổi được giá trị.
 - ❖ Các biến phải được khởi gán trước khi được sử dụng, nếu không trình biên dịch sẽ báo lỗi.



Hình 3.1: Thông báo lỗi của hệ thống

- ❖ Hằng số (constant): là biến số nhưng không thể thay đổi giá trị sau khi khởi gán.

```
const int myConst=32;  
myConst = 30;
```



 Kiểu liệt kê: chứa một tập hợp các hằng số (enumerator list). Lợi ích của enumerator:

- ❖ Cho phép một nhóm tập hợp các hằng số có liên quan với nhau => tránh phải khai báo nhiều hằng số và tăng mối quan hệ giữa các hằng số.
- ❖ Làm đơn giản hóa mã chương trình.

```
enum GoodWeather : uint
```

```
{
```

```
temperature=28,  
moisture=80,  
wind=5
```

```
}
```



Chuỗi (string) và cách đặt tên trong c#:

- ❖ Chuỗi là một mảng các ký tự.
- ❖ Khai báo và khởi gán giá trị cho chuỗi: `string str = "Hello";`

- ❖ Trong C#, chúng ta nên đặt tên các biến, các phương thức, các lớp ... theo khuyến cáo của Microsoft: Tên biến bắt đầu bằng chữ thường, tên phương thức và các thành phần khác bắt đầu bằng chữ hoa

3.3. Các câu lệnh và cấu trúc điều khiển (Statements)

Một chương trình C# là một dãy các câu lệnh (statements).

Mỗi một câu lệnh kêu thúc bởi dấu “;”.

Các câu lệnh được xử lý theo chiều từ trên xuống dưới (trừ các câu lệnh điều khiển: lệnh nhảy, lệnh lặp ...).

Int x; // a statement

x = 23; // another statement

Int y = x; yet another statement



Lệnh nhảy không điều kiện: có hai trường hợp phát sinh lệnh nhảy không điều kiện:

- ❖ Có lời gọi một phương thức: dùng phương thức hiện tại và nhảy sang (nhảy không điều kiện) thực hiện phương thức vừa triệu gọi, sau khi triệu gọi xong, trở về phương thức trước đó.
- ❖ Sử dụng một trong số các lệnh nhảy không điều kiện: goto, break; continue, return, throw.



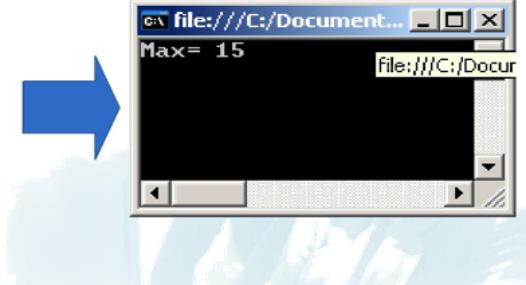
Các lệnh nhảy có điều kiện:

- ❖ Cấu trúc: `if ... else`
- ❖ Cấu trúc: `if ... else` lồng nhau
- ❖ Cấu trúc: `switch ... case`

a. Cấu trúc `if ... else`

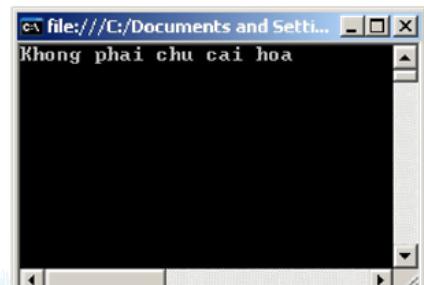
if (Biểu thức điều kiện) Công việc 1 [else Công việc 2]

```
int valueOne = 10;
int valueTwo = 15;
if(valueOne>=valueTwo)
{
    Console.WriteLine("Max= {0}",valueOne);
}
else
{
    Console.WriteLine("Max= {0}",valueTwo);
}
```



- b. Cấu trúc `if ... else` lồng nhau: có thể được sử dụng trong những câu lệnh điều kiện phức tạp, điều kiện lồng nhau.

```
char charTest = 'z';
if (charTest < 'A')
{
    Console.WriteLine("Khong phai chu cai hoa");
}
else
{
    if (charTest > 'Z')
    {
        Console.WriteLine("Khong phai chu cai hoa");
    }
    else
    {
        Console.WriteLine("La chu cai hoa");
    }
}
```



- c. Cấu trúc `switch ... case`

switch (biểu thức cần kiểm tra)

{

case trường_hợp:

Các câu lệnh

Lệnh nhảy (break, continue)

[default: Các câu lệnh cho trường hợp mặc định]

}

Ví dụ: khai báo một biến kiểu integer ứng với mỗi giá trị của biến là các ngày trong tuần, với mỗi giá trị của biến sẽ đưa ra các thông báo khác nhau.

```
Int thu = 2;  
Switch (Thu)  
{  
    Case 2:  
        MessageBox.Show("Thứ 2");  
        Break;  
    Case 3:  
        MessageBox.Show("Thứ 3");  
        Break;  
    Case 4:  
        MessageBox.Show("Thứ 4");  
        Break;  
    Case 5:  
        MessageBox.Show("Thứ 5");  
        Break;
```

Case 6:

```
    MessageBox.Show("Thứ 6");  
    Break;
```

Case 7:

```
    MessageBox.Show("Thứ 7");  
    Break;
```

Case 8:

```
    MessageBox.Show("Chủ nhật");  
    Break;
```

Default:

```
    MessageBox.Show("Không phải ngày trong tuần");  
    Break;
```

```
}
```



Các vòng lặp: `for`, `while`, `do ... while`, `foreach`

a. Vòng lặp For:

for ([initializers]; [expression]; [iterators]) statement

Khởi tạo

Biểu thức kiểm tra

Lệnh lặp

Công việc cần thực thi

```
for (int i=0; i<10; i++) Console.WriteLine("Current  
value = {0}",i);
```

Ví dụ:

```
static void Main(string[] args)
{
    for (int i = 0; i < 25; i++)
    {
        if (i % 5 == 0)
        {
            Console.WriteLine("\n");
            Console.WriteLine("\t{0}", i);
        }
        else
        {
            Console.WriteLine("\t{0}", i);
        }
    }
}
```

```
file:///C:/Documents and Settings/Pham Van Thua...
0      1      2      3      4
5      6      7      8      9
10     11     12     13     14
15     16     17     18     19
20     21     22     23     24
```



b. Vòng lặp while

while (*expression*) statement

Biểu thức kiểm tra

Công việc cần thực thi

```
static void Main(string[ ] args)
{
    int iTesT=0 ;
    while(iTesT<10)
    {
        Console.WriteLine("{0}",iTesT);
        iTesT++;
    }
}
```

```
file:///C:/Doc...
0123456789_
```



c. Vòng lặp do ... while

do *statement* while *expression*

Công việc cần thực thi

Biểu thức kiểm tra

```
static void Main(string[ ] args)
{
    int i = 11;
    do
    {
        Console.WriteLine("i: {0}",
                          i); i++;
    } while (i < 10);
}
```



- d. Vòng lặp foreach: sử dụng lặp qua các phần tử của một mảng hay của một tập hợp.



3.4. Các toán tử (Operators)

Toán tử được phân thành nhiều kiểu khác nhau:

-  Toán tử gán (assignment operator): =
-  Toán tử toán học (arithmetic operators): +, -, * ...
-  Toán tử tăng, giảm (increment and decrement operators): ++, --, -=, *= ...
-  Toán tử quan hệ (relational operators): ==, != ...
-  Toán tử logic (logical operators): &&, || ...
-  Toán tử ba thành phần (Ternary Operator)

 Các toán tử cơ bản: Cộng (+), trừ (-), nhân (*), chia(/)

 Toán tử chia lấy phân dư: %

```
myInt=myInt+10  
myInt=myInt-10  
myInt=myInt*10  
myInt=myInt/10
```

```
myInt+=10  
myInt-=10  
myInt*=10  
myInt/=10
```

Postfix: myInt++
 myInt--
Prefix : ++myInt
 --myInt

Gán trước, biến đổi sau

VD: myInt=1;
 newInt=myInt++;

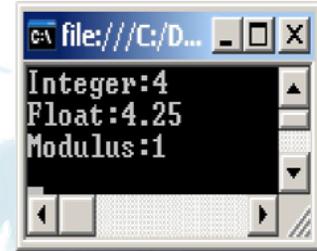
newInt=1

Biến đổi trước, gán sau

VD: myInt=1;
 newInt=++myInt

newInt=2

```
static void Main(string[] args)  
{  
    int i1 = 17;  
    int i2 = 4;  
    Console.WriteLine("Integer:{0} \nFloat:{1}  
    \nModulus:{2}", i1 / i2, (float)i1 / i2, i1 % i2);  
}
```



Toán tử	Tên
==	So sánh bằng
!=	So sánh khác nhau
>	So sánh hơn
>=	So sánh lớn hơn hoặc bằng
<	So sánh nhỏ hơn
<=	So sánh nhỏ hơn hoặc bằng
&&	Toán tử “và” logic

	Toán tử “hoặc” logic
!	Toán tử “phủ định”



Toán tử 3 thành phần: *conditional-expression ? expression1 : expression2*

- ❖ Conditional-expression: biểu thức kiểm tra điều kiện.
- ❖ Expression1: biểu thức 1, thực hiện khi biểu thức kiểm tra trả về true.
- ❖ Expression2: biểu thức 2, thực hiện khi biểu thức kiểm tra trả về false.

Ví dụ: khai báo hai biến là valueOne và valueTwo và gán giá trị cho 2 biến đó. Sau đó khai báo biến maxValue, gán giá trị cho biến maxValue bằng biểu thức tìm giá trị lớn nhất trong 2 biến valueOne và valueTwo.

```
int valueOne = 10;
int valueTwo = 20;
int maxValue = valueOne > valueTwo ? valueOne:valueTwo
```



Bài tập : em sẽ cho một số bài tập vào phần này sau (có thể cho làm trên console để cho sinh viên làm quen với các cú pháp lệnh trong C# hoặc có thể làm trên Visual Studio luôn để làm quen luôn với các control + cú pháp lệnh).

CHƯƠNG IV. HƯỚNG ĐỐI TƯỢNG TRONG C#

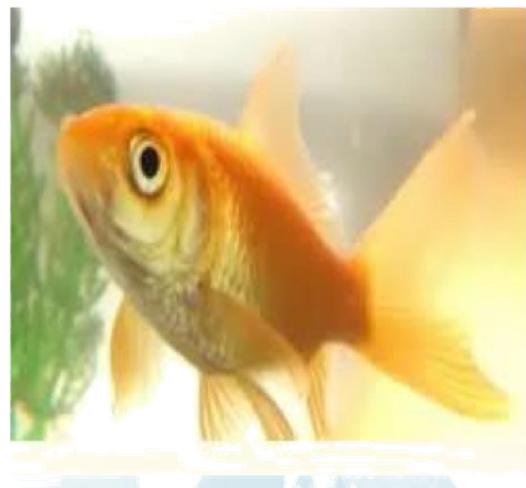
4.1. Cơ bản về lập trình hướng đối tượng



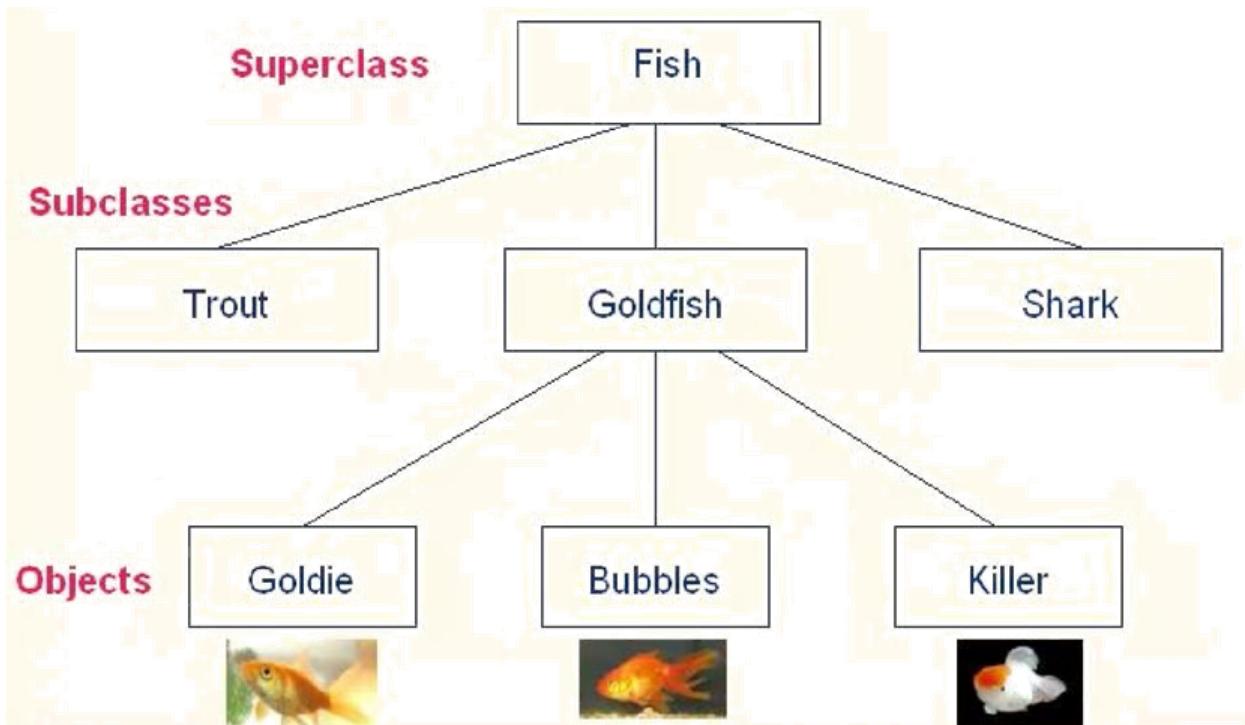
a. Thuộc tính và phương thức

Lớp và các đối tượng thuộc lớp đó có chung các thuộc tính (properties) và phương thức (methods)

- Thuộc tính (Property): kích thước, màu, giới tính...
- Phương thức (Method): bơi, thở, ăn...



b. Kế thừa



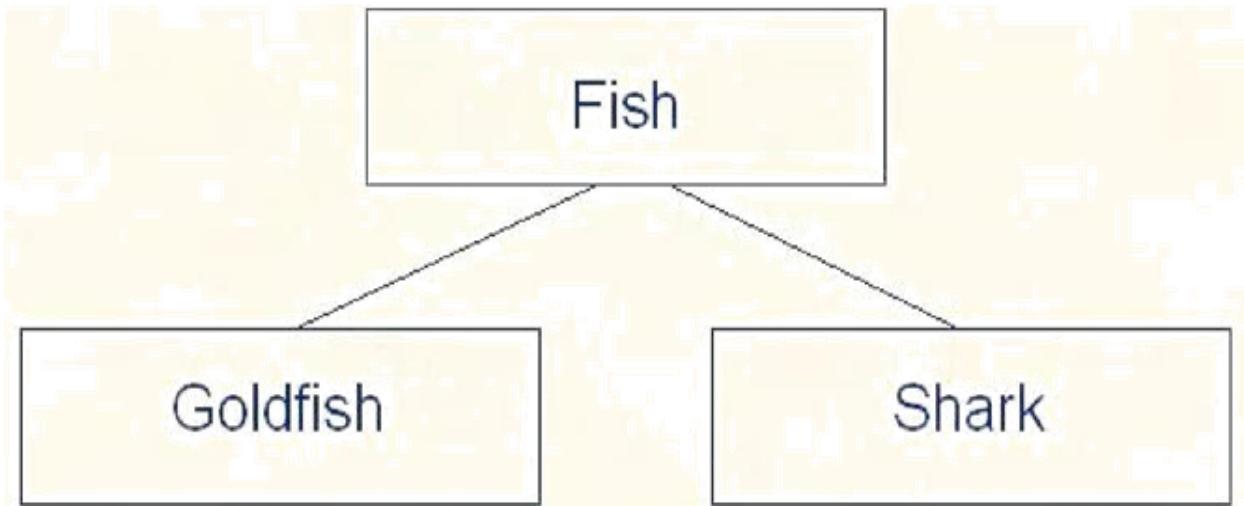
c. Các quan hệ

Các đối tượng không tồn tại một cách độc lập, luôn tồn tại mối quan hệ giữa nó và các đối tượng khác: tổng quát hóa (generalization), cộng tác (asscociation)...

- **Các đồ vật nằm trong phòng**
- **Lọ hoa ở trên bàn**
- **Bàn và ghế thuộc nhóm đồ nội thất**



Tổng quát hóa:



- ❖ Lớp cha (fish) tổng quát hóa các thuộc tính, phương thức chung của các lớp con (goldfish và shark).
- ❖ Các lớp con cũng có những thuộc tính và phương thức riêng.

 *Quan hệ cộng tác:* Các lớp có liên kết (connected) hoặc liên quan (related) với nhau:



**Cd player + Speaker
(Collaboration)**



Shark eat fish



Quan hệ kết hợp: Thể hiện mối quan hệ giữa tổng thể (the whole), và bộ phận (parts): tổng thể (đàn cá vàng), bộ phận (từng con cá vàng).



Quan hệ cấu thành: là một dạng của quan hệ kết tập, tuy nhiên yêu cầu chặt chẽ hơn.

❖ Mỗi bộ phận (part) chỉ nằm trong một tổng thể duy nhất. (ví dụ: phòng ngủ nhà bạn phải nằm trong ngôi của bạn mà không thể nằm trong ngôi nhà nào khác).

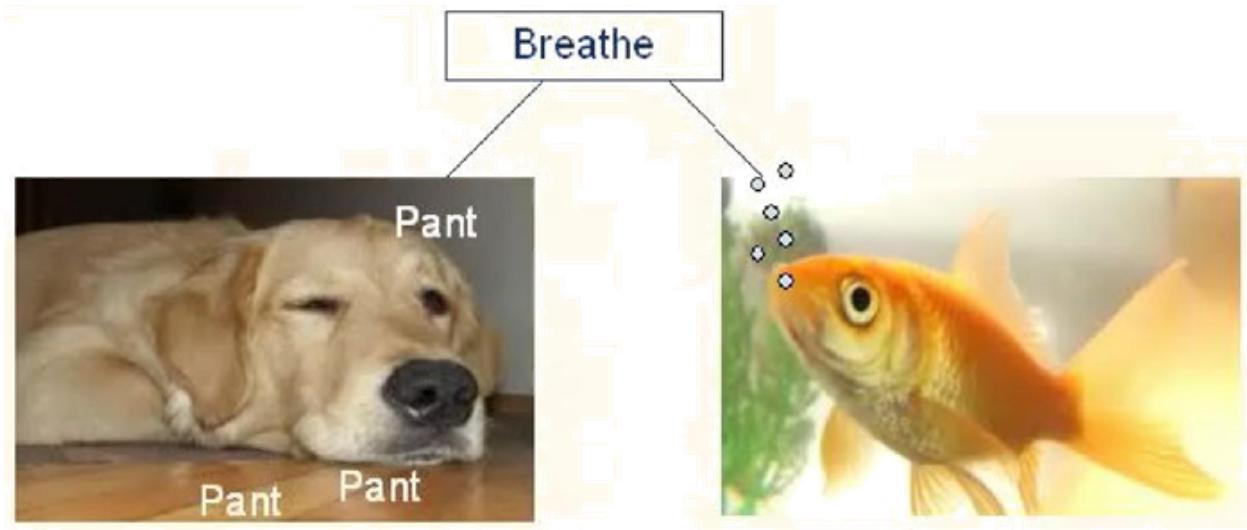


Quan hệ cấu thành: Khi phá hủy cái tổng thể thì đồng thời cái bộ phận cũng bị phá hủy.



d. Quan hệ

- ❖ Số lượng đối tượng tham gia vào quan hệ:
 - ❖ Quan hệ 1 – 1: Một bộ môn trong một phòng.
 - ❖ Quan hệ 1 – nhiều: một giáo viên có thể dạy nhiều lớp.
 - ❖ Quan hệ nhiều – nhiều: một giáo viên dạy nhiều sinh viên, một sinh viên học nhiều giáo viên.
- ❖ Tính đa hình: là khả năng có thể thực hiện cùng một hành động (phương thức) với nhiều cách thức khác nhau không phụ thuộc vào đối tượng cụ thể thuộc lớp nào.



4.2. Lớp và đối tượng (class and Object)

a. Định nghĩa lớp

[*attributes*] [*access-modifiers*] class *identifier* [:*base-class*[, *interface(s)*]]{*class-body*}

- ❖ Access-modifiers: quyết định phạm vi truy cập tới các thuộc tính và phương thức của lớp.
- ❖ Identifier : tên lớp
- ❖ Base-class : lớp cơ sở (lớp cha)



Class-body: định nghĩa các thuộc tính và phương thức của lớp.

b. Access – modifier

Access-modifier	Giới hạn truy nhập
Public	Không giới hạn. Các thành phần public có thể được truy xuất bởi bất cứ phương thức của bất kỳ lớp nào khác
Private	Chỉ có thể truy xuất bởi các phương thức của chính lớp đó
Protected	Có thể được truy xuất bởi các phương thức của chính lớp đó và các lớp dẫn xuất (derived) từ nó
Internal	Có thể được truy xuất bởi các phương thức của các lớp trong cùng Assembly
internal protected	Có thể được truy xuất bởi các phương thức của lớp đó, lớp dẫn xuất từ lớp đó và các lớp trong cùng Assembly với nó

c. Ví dụ định nghĩa lớp

```

public class Time
{
    // Các thuộc tính
    private int Year;
    int Month;
    int Date; int
    Hour; int
    Minute; int
    Second;
    // Các phương thức public
    void DisplayCurrentTime()
    {
        Console.WriteLine("Ngay, gio hien tai");
    }
}

```

Khai báo lớp

Các thuộc tính
(phương thức
truy cập mặc
định là private)

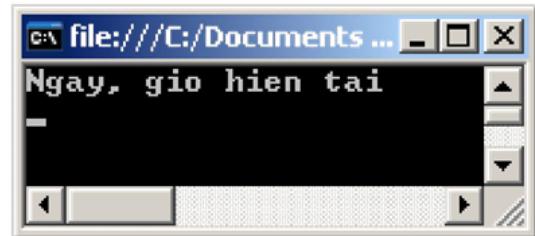
Phương
thức của lớp

d. Khai báo và sử dụng đối tượng của lớp

```

public class Tester
{
    static void Main()
    {
        Time t = new Time();
        t.DisplayCurrentTime();
    }
}

```



e. Tạo đối tượng



Sử dụng từ khóa “new”

`Class_name Object_name = new Class_name();`

Ví dụ: `Time t = new Time();`



Đối tượng là biến kiểu tham chiếu, không phải tham trị:

- ❖ Biến t không chứa giá trị của đối tượng.
- ❖ Biến t chứa địa chỉ của đối tượng được tạo ra trong bộ nhớ Heap

f. *Hàm tạo (constructor)*



Khởi tạo một đối tượng => gọi hàm tạo (constructor)

- ❖ Hàm tạo mặc định: sẽ được CLR cung cấp nếu người lập trình không định nghĩa.
- ❖ Hàm tạo do người lập trình định nghĩa.



Hàm tạo có chức năng:

- ❖ Tạo một đối tượng của lớp và chuyển nó sang trạng thái xác định.
- ❖ Hàm tạo thường được dùng để khởi gán các thuộc tính của đối tượng.



Hàm tạo mặc định:

- ❖ Hàm tạo mặc định tạo ra đối tượng của lớp ngoài ra không làm gì khác.
- ❖ Các thuộc tính được khởi gán các giá trị mặc định.

Kiểu dữ liệu	Giá trị mặc định
Numeric (int, long ...)	0
Bool	False
Char	'\0' null
Enum	0
Reference	Null



Xây dựng hàm tạo:

- ❖ Hàm tạo có tên hàm trùng tên đối tượng, không có kiểu dữ liệu trả về và thường có phạm vi là Public.
- ❖ Hàm tạo có thể có tham số hoặc không.

Public Class_Name()

Public Class_Name(argument list)

- ❖ Ví dụ:

Public Time ()

Public Time (Datetime t)

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

```
    public void DisplayCurrentTime()  
    {  
        System.Console.WriteLine("{0}/{1}/{2}  
        {3}:{4}:{5}", Date, Month, Year, Hour,  
        Minute, Second);  
    }  
}
```

Point
-x : int
-y : int
+Point(in x : int, in y : int)
+Display() : void

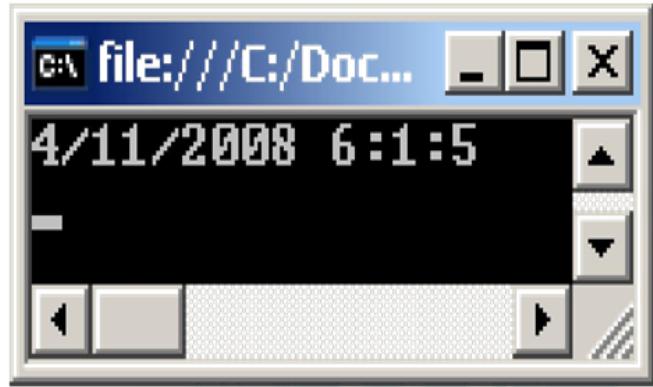


Kiểm tra hàm tạo

```

public class Tester
{
    static void Main()
    {
        System.DateTime currentDate = System.DateTime.Now;
        Time t = new Time(currentDate); t.DisplayCurrentTime();
    }
}

```



g. Hàm tạo sao chép

 Để hỗ trợ chức năng hàm tạo sao chép. Net định nghĩa một interface Icloneable.

 Class hỗ trợ hàm tạo sao chép phải.

❖ Implement interface Icloneable

```

public Object Clone()
{
    return MemberwiseClone(); // shallow copy
}

```

❖ Tự xây dựng hàm tạo sao chép:

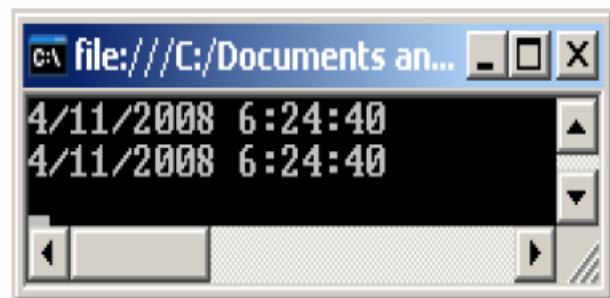
```
public Time(Time previousTime)
{
    Year = previousTime.Year;
    Month = previousTime.Month;
    Date = previousTime.Date;
    Hour = previousTime.Hour;
    Minute = previousTime.Minute;
    Second = previousTime.Second;
}
```

❖ Gọi hàm tạo sao chép:

```

public class Tester
{
    static void Main()
    {
        System.DateTime currentDate = System.DateTime.Now;
        Time t = new Time(currentDate);
        Time t2 = (Time)t.Clone();
        t2.DisplayCurrentTime();
        Time t3 = new Time(t2);
        t3.DisplayCurrentTime();
    }
}

```



h. Từ khóa “This”

- Từ khóa this trả đến thẻ hiện tại (current instance) của đối tượng.
- Từ khóa this rất hữu ích trong một số trường hợp:
 - ❖ Chỉ rõ thành phần, thuộc tính của đối tượng, tránh nhầm lẫn với tên biến, tránh sự nhập nhằng về tên.

```

public void SomeMethod (int hour)
{
    this.hour = hour;
}

```

- ❖ Dùng làm tham số truyền vào cho một phương thức của một đối tượng khác, cho phép phương thức đó có thể tác động đến các thành phần của phương thức hiện tại.

```
class myClass
{
    public void Foo(OtherClass otherObject)
    {
        otherObject.Bar(this);
    }
}
```

- ❖ Gọi một hàm tạo từ hàm tạo khác của lớp.

```
class myClass
{
    public myClass(int i) { //... }
    public myClass( ) : this(42) { //... }
}
```

- ❖ Gọi tương ứng các phương thức, thuộc tính của lớp.

```
public void MyMethod(int y)
{
    int x = 0;
    x = 7;          // assign to a local variable
    y = 8;          // assign to a parameter
    this.z = 5;     // assign to a member variable
    this.Draw( );   // invoke member method
}
```

i. Sử dụng các thành phần tĩnh

 Thành phần tĩnh là các thành phần chung (thuộc tính, phương thức) của lớp.

- ❖ Truy xuất các thành phần tĩnh thông qua tên lớp `Class_name.Static_Member`.
- ❖ C# không cho phép truy xuất các thành phần tĩnh thông qua đối tượng (thể hiện của lớp).
- ❖ Các thành phần tĩnh có thể được truy nhập, triệu gọi trước khi các đối tượng của lớp đó được tạo ra.
- ❖ Các phương thức tĩnh không thể truy xuất trực tiếp các thuộc tính, phương thức không tĩnh (nonstatic)

 Ví dụ:

```

namespace StaticFields
{
    public class Cat
    {
        private static int instances = 0;
        public Cat()
        {
            instances++;
        }
        public static void HowManyCats()
        {
            Console.WriteLine("{0} cats adopted",instances);
        }
    }
}

public class Tester
{
    static void Main()
    {
        Cat.HowManyCats();
        Cat frisky = new Cat();
        Cat.HowManyCats();
        Cat whiskers = new Cat();
        Cat.HowManyCats();
    }
}

```

file:///C:/Documents and Settings/Pham Van Thuan/My Documents/Visual Studio 2005/Projects/Hell...
0 cats adopted
1 cats adopted
2 cats adopted

j. Hủy đối tượng

-  C# hỗ trợ cơ chế tự động thu gom rác (garbage collector) -> người lập trình không phải hủy đối tượng một cách tƣờng minh.
-  Nếu đối tượng có sử dụng các tài nguyên khác (files...) -> người lập trình phải tự xây dựng hàm hủy (Destructor).
-  Bộ thu gom rác tự động gọi hàm hủy, các phương thức không gọi một cách tƣờng minh.
-  Về mặt cú pháp, hàm hủy C# giống với C++

k. Truyền tham số cho phương thức

-  Mặc định, tham số truyền cho phương thức là kiểu tham trị:

- ❖ Một bản sao của tham số đó được tạo ra.
- ❖ Bản sao đó sẽ bị hủy khi kết thúc phương thức.
- ❖ Giá trị của tham số được truyền không thay đổi sau khi kết thúc phương thức.

```
Public int AddValue (int value1, int value2)
```

```
{
```

```
    Return value1 + value2;
```

```
}
```



Truyền tham chiếu: C# hỗ trợ truyền tham chiếu sử dụng các từ khóa:

- ❖ Ref: Truyền tham chiếu, biến được tham chiếu phải được khởi gán trước khi truyền.
- ❖ Out: truyền tham chiếu, biến được tham chiếu không cần khởi gán trước khi truyền.



Truyền tham chiếu với từ khóa Ref:

Định nghĩa phương thức

```
public void GetTime( ref int h, ref int m, ref int s )  
{  
    h = Hour;  
    m = Minute;  
    s = Second;  
}
```

Gọi phương thức

```
System.DateTime currentTime = System.DateTime.Now;  
Time t = new Time( currentTime );  
int theHour = 0;  
int theMinute = 0;  
int theSecond = 0;  
t.GetTime( ref theHour, ref theMinute, ref theSecond );
```



Truyền tham chiếu với từ khóa Out:

Định nghĩa phương thức

```
public void GetTime( out int h, out int m, out int s )  
{  
    h = Hour;  
    m = Minute;  
    s = Second;  
}
```

Gọi phương thức

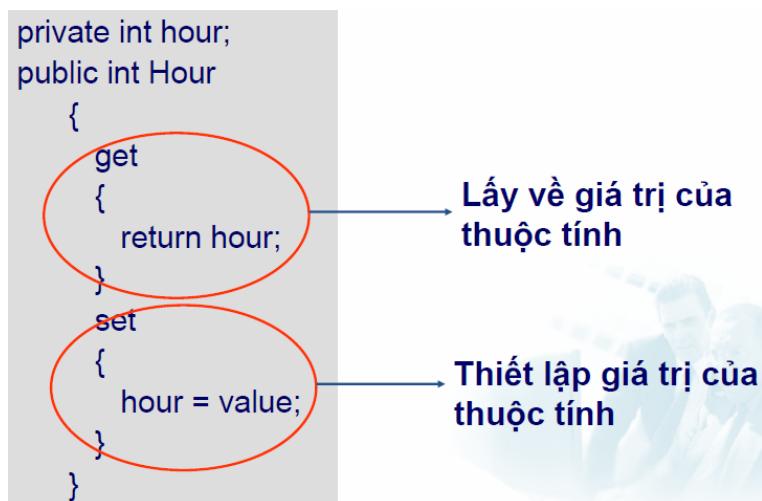
```
System.DateTime currentTime = System.DateTime.Now;  
Time t = new Time( currentTime );  
int theHour;  
int theMinute;  
int theSecond;  
t.GetTime( out theHour, out theMinute, out theSecond );
```

l. Đóng gói dữ liệu thành các thuộc tính



- Đóng gói dữ liệu thành các thuộc tính đem lại lợi ích:
- ❖ Vẫn đảm bảo một giao diện đơn giản cho các đối tượng khác giao tiếp.
 - ❖ Dễ dàng thay đổi mã mà không ảnh hưởng tới giao diện với các đối tượng khác.
 - ❖ Tăng tính an toàn dữ liệu, kiểm tra được dữ liệu khi gán.

m. Sử dụng các thuộc tính



n. Thuộc tính chỉ đọc

```
public static readonly int Year;
public static readonly int Month;
public static readonly int Date;
public static readonly int Hour;
public static readonly int Minute;
public static readonly int Second;
```

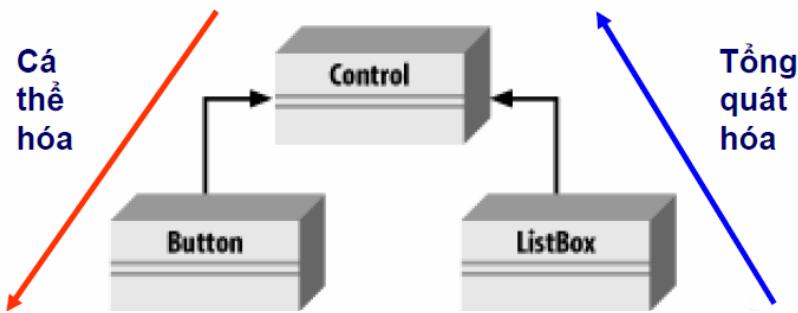
-> Chỉ có thể gán giá trị sử dụng hàm tạo tĩnh (static constructor) hoặc khởi gán ngay từ ban đầu.

4.3. Kỹ thuật kế thừa và đa hình (Inheritance and polymorphism)

Gồm 4 kỹ thuật chính:

- Tổng quát hóa và cá thể hóa (generalization and specialization).
- Ké thừa (inheritance).
- Đa hình (polymorphism).
- Lớp trừu tượng (abstract class).

a. Tổng quát hóa và cá thể hóa

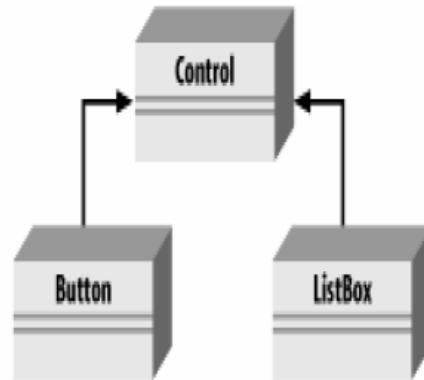


- Tổng quát hóa: các lớp đối tượng có những thuộc tính, phương thức chung được tổng quát thành các lớp cha.
- Cá thể hóa: các lớp con kế thừa các phương thức, thuộc tính của lớp cha và bổ sung thêm thuộc tính, phương thức của riêng nó.

b. Ké thừa

Ké thừa (inheritance) là kỹ thuật thể hiện quan hệ cá thể hóa (specialization).

```
public class ListBox : Control  
public class Button : Control
```



Ưu điểm của kỹ thuật kế thừa:

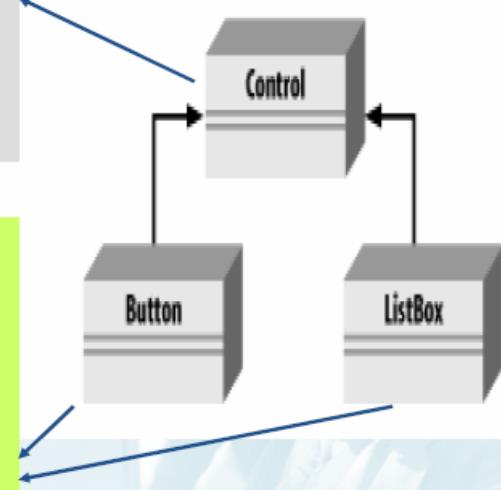
- ❖ Khả năng tái sử dụng mã. VD: lớp ListBox có thể sử dụng một số phương thức, thuộc tính của lớp Control
- ❖ Khả năng sử dụng kỹ thuật đa hình (Polymorphism). VD: lớp Control có phương thức Draw(), lớp Listbox và Button cũng có phương thức Draw().



c. Tạo phương thức hỗ trợ đa hình

K

```
public virtual void Draw( )  
{  
    //Code statements  
}
```



```
public override void Draw( )  
{  
    //Code statements  
}
```

d. Lớp trừu tượng



Sử dụng lớp trừu tượng khi:

- ❖ Yêu cầu tất cả các lớp con, kế thừa từ nó phải cài đặt một hoặc một vài phương thức nào đó.



Đặc điểm của lớp trừu tượng:

- ❖ Lớp trừu tượng là lớp tồn tại ít nhất một phương thức trừu tượng.
- ❖ Không thể tạo đối tượng của lớp trừu tượng



Khai báo lớp trừu tượng:

```
abstract public class Control
{
    protected int top;
    protected int left;
    // constructor
    public Control( int top, int left )
    {
        this.top = top;
        this.left = left;
    }
    abstract public void DrawWindow( );
}
```

Khai báo lớp
trừu tượng

Khai báo
phương thức
trùu tượng



Kế thừa lớp trừu tượng:

```

public class Button : Control
{
    public Button(int top,int left ):base(top, left)
    {
    }
    // implement the abstract method
    public override void DrawWindow( )
    {
        Console.WriteLine( "Drawing a button at {0},
{1}\n",top, left );
    }
}

```

4.4. Xử lý chuỗi



Khai báo:

// Khai báo biến string có tên là str1, sau đó gán giá trị cho biến

String str1; str1 = “Xin chào lớp tin kinh tế k55”;

// Khai báo và khởi tạo giá trị cho biến str2

String str2 = “Xin chào lớp tin kinh tế k55”;

// Cú pháp:

String [tên biến] = [giá trị];

String [tên biến];



Thao tác với chuỗi:

Thuộc tính, phương thức	Mô tả
Empty	Thuộc tính cho biết chuỗi trống

Compare()	Phương thức so sánh hai chuỗi
Concat()	Phương thức nối hai chuỗi
Copy()	Tạo bản sao của một chuỗi
Equals	Kiểm tra hai chuỗi có giống nhau không
Format()	Định dạng một chuỗi
SubString()	Cắt lấy chuỗi con từ một chuỗi mẹ
Length()	Lấy độ dài của chuỗi
Remove()	Xóa bỏ một chuỗi con trong chuỗi mẹ
Replace()	Thay thế một chuỗi vào một chuỗi con trong chuỗi mẹ
...	



Ví dụ:

```
static void Main()
{
    string str1 = "Hanoi";
    string str2 = "La thu do cua Viet Nam";
    string str3 = "HANOI";
    //So sanh chuoi, tra ve 0 neu giuong nhau
    int result = string.Compare(str1, str3);
    Console.WriteLine("Ket qua so sanh hai chuoi {0} va {1} la {2}", str1, str3, result);
    Console.ReadLine();
    //Noi chuoi
    string str4 = string.Concat(str1, " ", str2);
    string str5 = str1 + " " + str2;
    Console.WriteLine("Ket qua noi chuoi {0} va {1}: \n + Su dung phuong thuc
    Concat: {2} \n + Su dung ham chong toan tu +: {3}", str1, str2, str4, str5);
    Console.ReadLine();
    //Lay ve chieu dai chuoi
    Console.WriteLine("Chieu dai cua chuoi \"{0}\" la {1}", str2, str2.Length);
    Console.ReadLine();
}
```

4.5. Bắt ngoại lệ (Handling Exception)

C# sử dụng kỹ thuật Handling Exception để bắt và xử lý lỗi cũng như các ngoại lệ phát sinh trong quá trình thực thi chương trình.

Phân loại bug, error và exception:

-  Bug: lỗi do người lập trình, cần loại bỏ trước khi hoàn thành phần mềm.
-  Error: lỗi phát sinh do người dùng (người dùng nhập sai dữ liệu => phải kiểm tra dữ liệu đầu vào).
-  Exception: lỗi bất thường (hết bộ nhớ, chia cho 0...)

Để báo hiệu một điều kiện bất thường xuất hiện trong quá trình thực thi, thực hiện ném ngoại lệ.

```
Throw new System.Exception();
```

Khi một ngoại lệ được tung ra, chương trình sẽ ngay lập tức dừng lại và CLR sẽ tìm, kiểm tra chương trình bắt ngoại lệ, nếu không tìm thấy nó sẽ kết thúc chương trình.

Ví dụ: ngoại lệ được thông báo trong chế độ Debug:

The screenshot shows a Microsoft Visual Studio IDE window titled "UsingStringBuilder.StringTester". The code editor displays the following C# code:

```

#region Using directives

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

#endregion

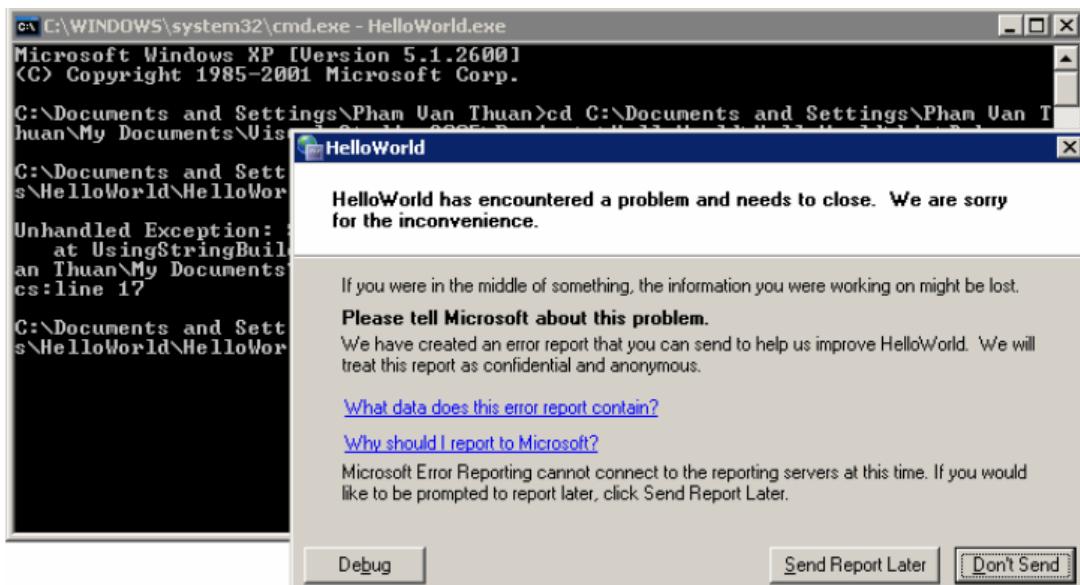
namespace UsingStringBuilder
{
    public class StringTester
    {
        static void Main()
        {
            int a = 19;
            int b = 0;
            int c = a / b;
        }
    }
}

```

A yellow arrow points from the line of code `int c = a / b;` to a "DivideByZeroException was unhandled" dialog box. The dialog box contains the following information:

- Attempted to divide by zero.**
- Troubleshooting tips:**
 - Make sure the value of the denominator is not zero before performing a division operation.
 - Get general help for this exception.
- Actions:**
 - View Detail...
 - Copy exception detail to the clipboard

Lỗi chương trình khi chạy thực tế:

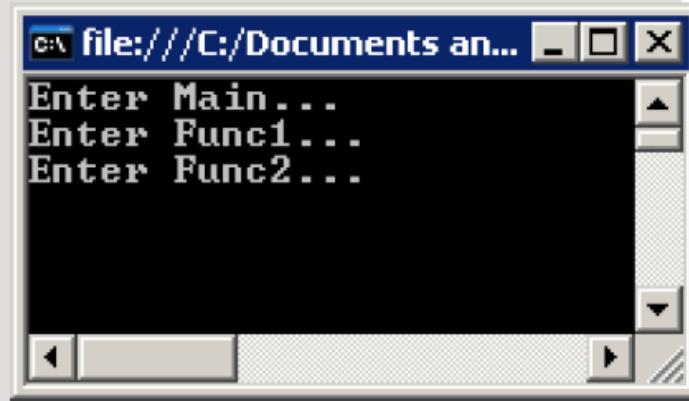


Ví dụ:

```

public class Test
{
    public static void Main()
    {
        Console.WriteLine("Enter Main...");
        Test t = new Test();
        t.Func1();
        Console.WriteLine("Exit Main...");
    }
    public void Func1()
    {
        Console.WriteLine("Enter Func1...");
        Func2();
        Console.WriteLine("Exit Func1...");
    }
    public void Func2()
    {
        Console.WriteLine("Enter Func2...");
        throw new System.Exception();
        Console.WriteLine("Exit Func2...");
    }
}

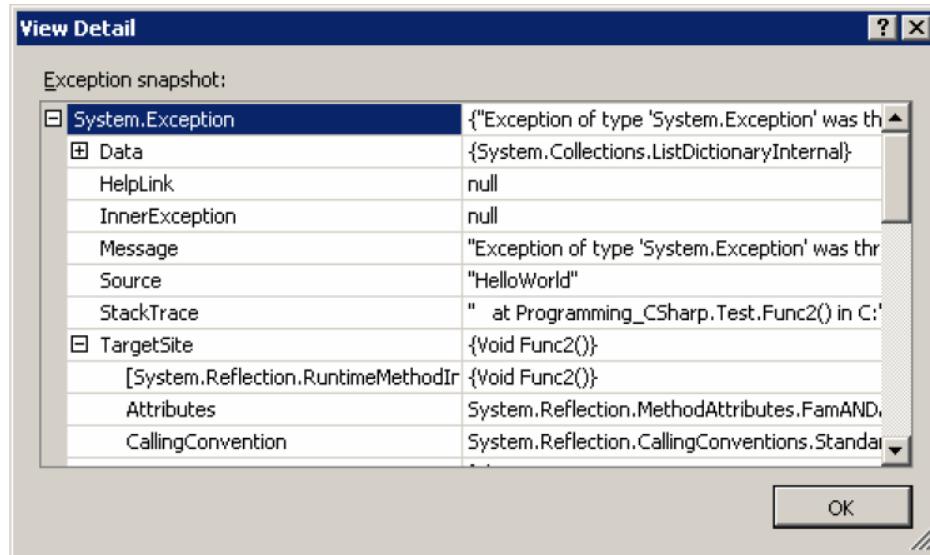
```



Ngoại lệ xuất hiện:



Chi tiết ngoại lệ:



- ❖ Sử dụng try ... catch ... finally để kiểm tra, bắt và xử lý ngoại lệ

```
try
{
    //Lệnh có thể phát sinh ngoại lệ, cần kiểm tra
}
catch(Exception e)
{
    //Bắt và xử lý ngoại lệ nếu có
}
Finally
{
    //Đoạn mã luôn thực thi khi xuất hiện ngoại lệ
}
```

- ❖ Có thể có nhiều đoạn lệnh catch trong một câu lệnh try ... catch tương ứng với nhiều ngoại lệ khác nhau.
- ❖ Đoạn lệnh try ... catch có thể đặt trong phương thức có thể phát sinh ngoại lệ hoặc đặt ở cấp cao hơn, phương thức triệu gọi đoạn mã có thể phát sinh ngoại lệ

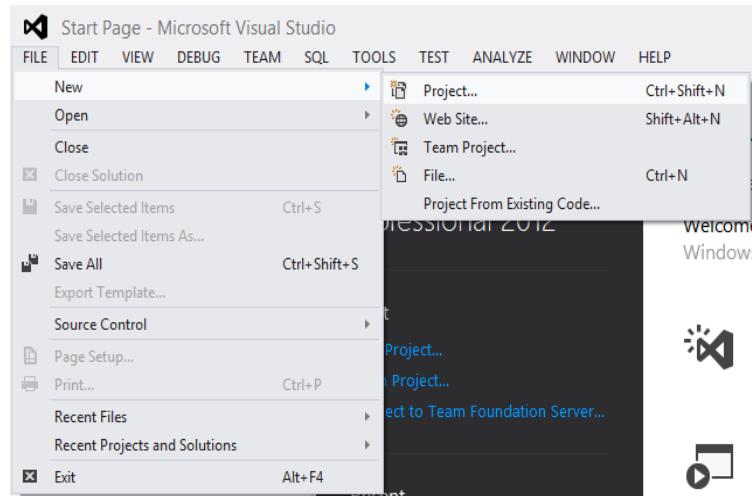
CHƯƠNG V. LẬP TRÌNH ỨNG DỤNG VỚI C#

5.1.Xây dựng chương trình C# đầu tay

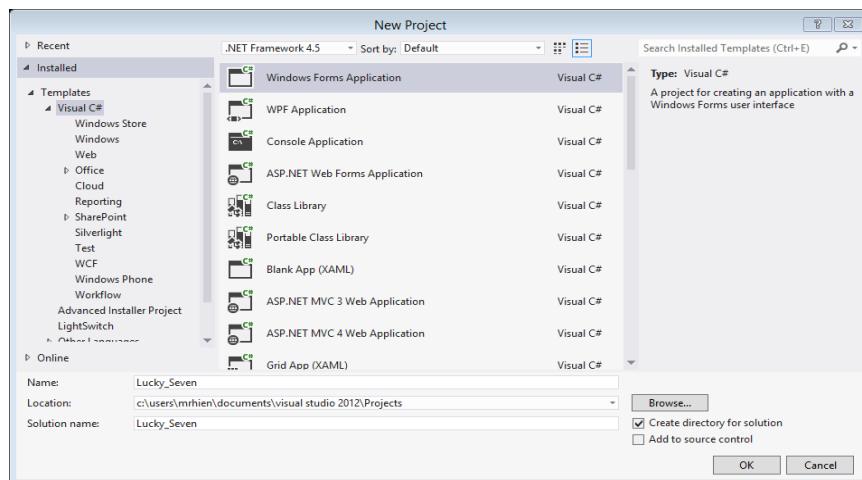
5.1.1. Xây dựng chương trình C# đầu tay

Trong phần này, chúng ta sẽ tiến hành xây dựng một project c# đầu tay. Project mang tên: “Lucky Seven”.

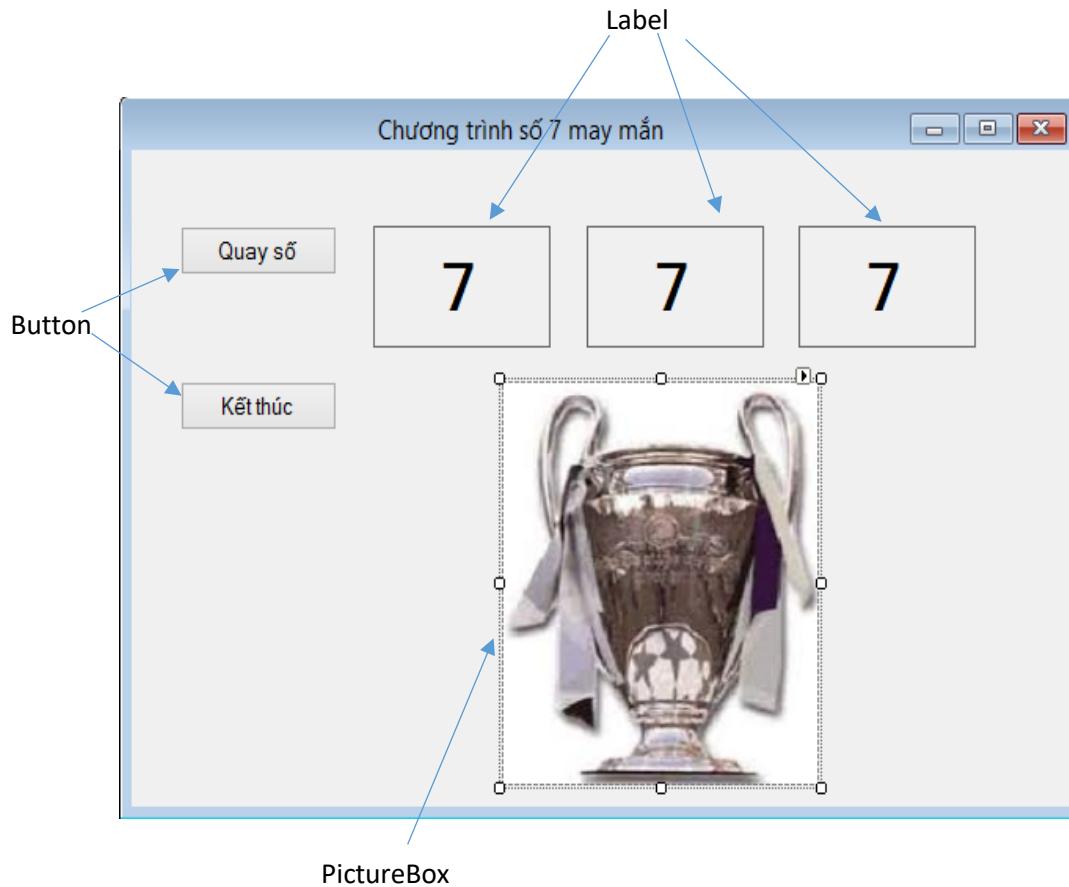
Chạy Visual Studio, chọn File => New => Project (hoặc nhấn tổ hợp phím “Ctrl + Shift + N”) để tạo mới một Project:



Một cửa sổ mới xuất hiện, chọn các thông tin liên quan đến Project (ngôn ngữ lập trình: C#, Kiểu Application: Windows Forms Application, Name: Lucky_Seven, thư mục lưu trữ Project) sau đó “Click” nút lệnh “Ok” để tạo mới project:



Sau khi tạo mới Project, theo mặc định project sẽ có sẵn một form có tên là Form1, sử dụng các control trong thanh toolbox tạo giao diện như sau:



Giao diện chương trình bao gồm: 2 Button, 3 Label, 1 PictureBox. Vào cửa sổ Properties:

- ➡ Đặt thuộc tính “Text” của Form1 là: “Chương trình số 7 may mắn”
- ➡ Đặt thuộc tính “AutoSize” của 3 Label là “False”, thuộc tính “ TextAlign” của 3 Label là Middlecenter, chọn font chữ cho 3 label.
- ➡ Đặt tên cho 2 nút lệnh là “btQuaySo” và “btKetThuc”, đặt thuộc tính “Text” cho 2 nút lệnh là “Quay số” và “Kết thúc”.



Đặt thuộc tính “SizeMode” của PictureBox là “StretchImage”, vào thuộc tính “Image” để chọn ảnh hiển thị trên PictureBox.

Khi bạn click vào nút “Quay số” thì chương trình phát sinh ngẫu nhiên ba số ở ba label. Nếu một trong ba số là chữ số 7 thì hiện ảnh ở đối tượng PictureBox, nếu cả 3 số đều là chữ số 7 thì sẽ hiện ảnh ở đối tượng PictureBox và một thông báo với nội dung “Bạn là người chiến thắng!”.

Các bước thực hiện:

- Tạo giao diện cho chương trình.
- Thiết lập thuộc tính cho các đối tượng trong giao diện.
- Viết mã chương trình.
- Lưu và chạy chương trình.
- Biên dịch file thực thi .exe.

Sau đây chúng ta sẽ viết mã cho chương trình để chương trình chạy theo đúng yêu cầu đề ra. Để tạo sự kiện “click” cho các nút lệnh ta thực hiện theo một trong 2 cách sau: vào cửa sổ Properties => Event => Click hoặc Double Click vào nút lệnh.

Viết mã cho nút lệnh btQuaySo:

```
private void btQuaySo_Click(object sender, EventArgs e)
{
    Random rd = new Random();
    label1.Text = rd.Next(1, 10).ToString();
    label2.Text = rd.Next(1, 10).ToString();
    label3.Text = rd.Next(1, 10).ToString();
    if (label1.Text == "7" && label2.Text == "7" && label3.Text == "7")
    {
        pictureBox1.Visible = true;
        MessageBox.Show ("Bạn là người chiến thắng!");
    }
    else
    {
        if (label1.Text == "7" || label2.Text == "7" || label3.Text == "7")
```

```

        pictureBox1.Visible = true;
    else
        pictureBox1.Visible = false;
    }
}

```

Viết mã cho nút lệnh btKetThuc:

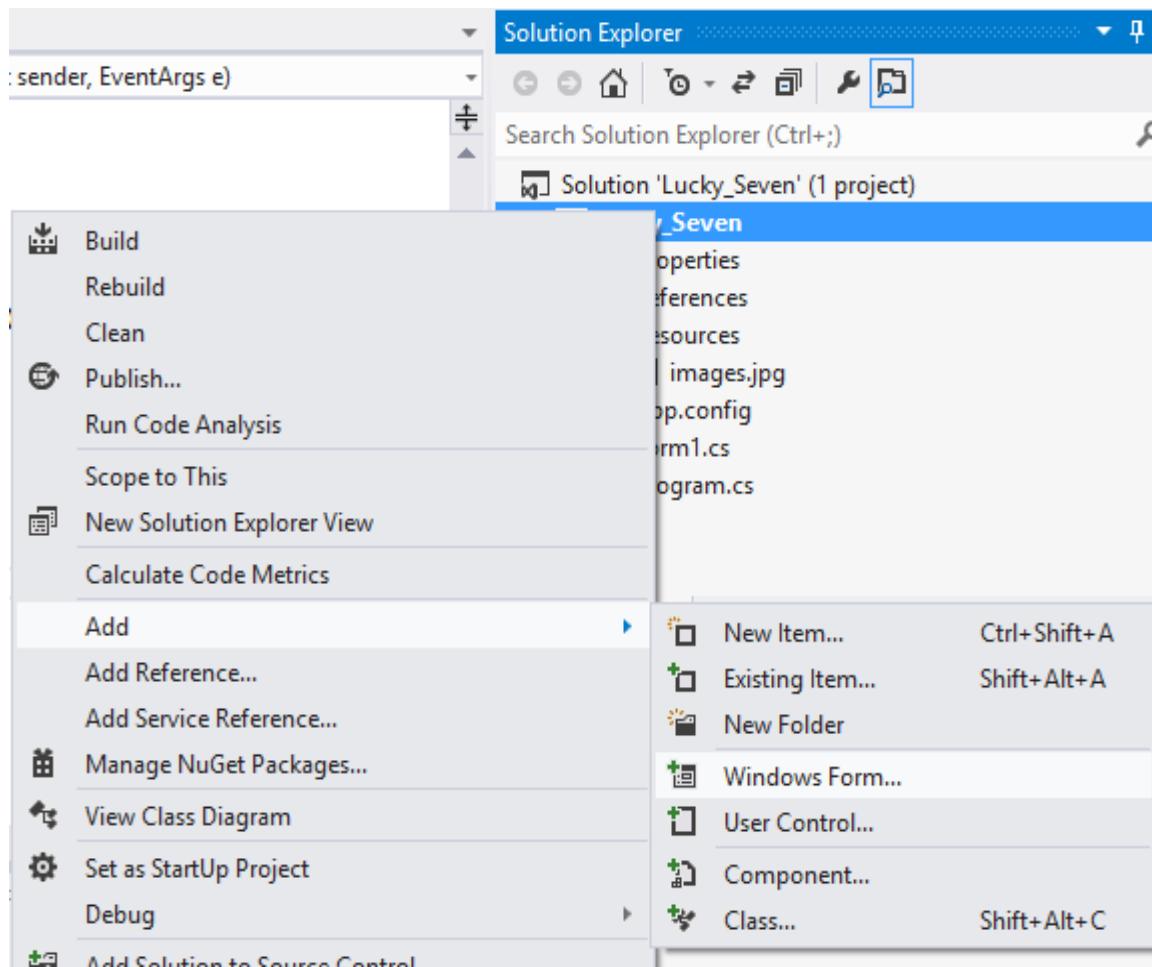
```

private void btKetThuc_Click(object sender, EventArgs e)
{
    Close();
}

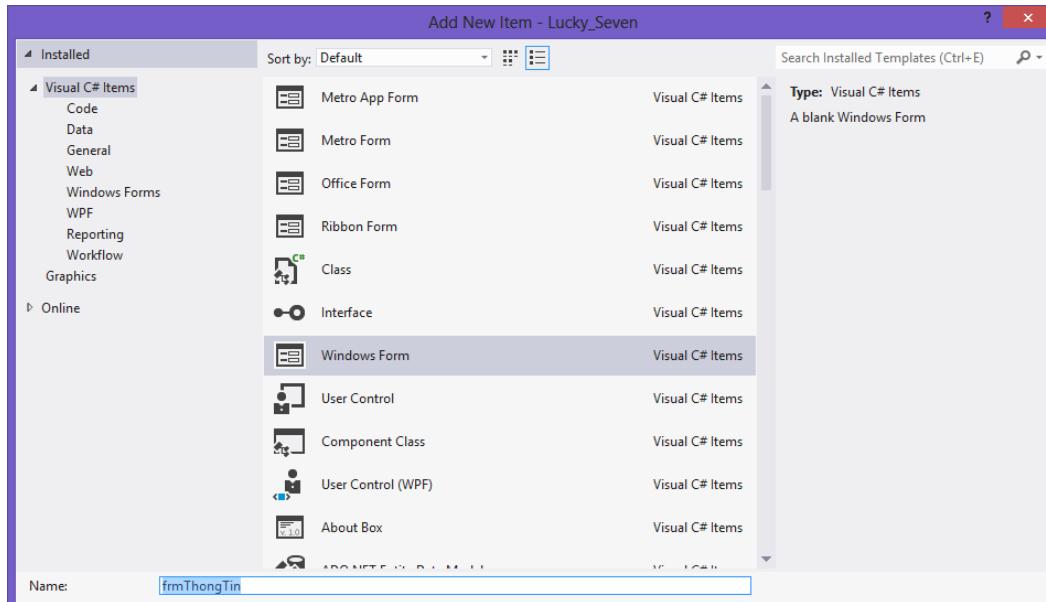
```

Sau khi viết mã cho các nút lệnh xong, chúng ta thêm một form nữa để hiển thị thông tin chương trình LuckySeven mà chúng ta vừa xây dựng.

Trong cửa sổ Solution Explorer click chuột phải vào project => Add => Windows Form để thêm một form mới:

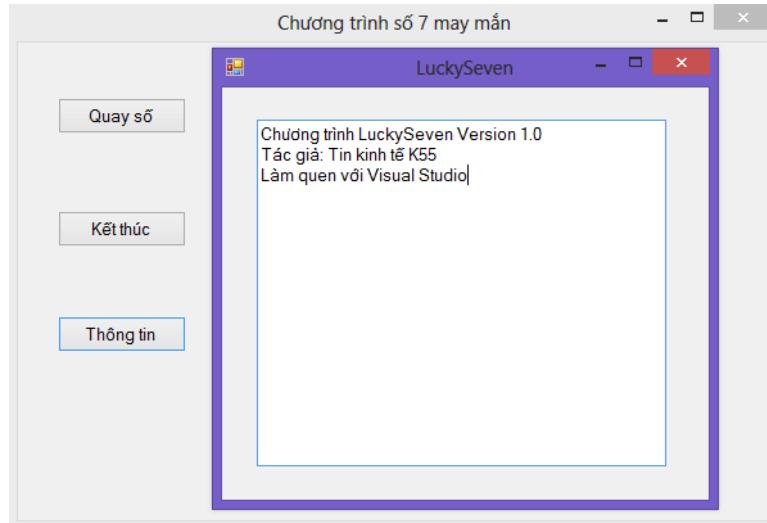


Chọn Windows Form, đặt tên cho Form là frmThongTin, rồi click nút lệnh Add để tạo form:



Đặt thuộc tính Text cho frmThongTin là LuckySeven, thêm một Textbox vào frmThongTin, đặt thuộc name cho textbox là txtThongTin, thuộc tính Multiline là true, thuộc tính Readonly là true, thuộc tính text là thông tin của chương trình. Sau đó thêm một Button vào Form1, đặt thuộc tính Name là btThongTin, thuộc tính text là Thông tin và viết mã cho nút lệnh btThongTin:

```
private void btThongTin_Click(object sender, EventArgs e)
{
    frmThongTin frm = new frmThongTin();
    frm.Show();
}
```



5.1.2. Định vị form trên màn hình Desktop

Bạn có thể định vị form trên màn hình desktop khi nó xuất hiện bằng thuộc tính DesktopBounds. Nó cho phép định vị trí của form với góc phải dưới và góc trái trên. Đơn vị tinh là pixel.

Ngoài ra bạn còn có thể sử dụng thuộc tính StartPosition với các đặc điểm: Manual – bằng tay, CenterScreen – giữa màn hình, WindowsDefaultLocation – vị trí mặc định, WindowsDefaultBound – kích thước mặc định.

a. Sử dụng thuộc tính StartPosition

Bây giờ chúng ta sẽ dùng thuộc tính StartPosition và DestopBounds để định vị trí form qua bài tập *MyDesktopBound* sau đây.

Bạn tạo mới một Project có tên *MyDesktopBound* và làm như sau:



Mở properties của form1.cs.



Thay thuộc tính StartPosition thành CenterScreen và chạy thử. Form sẽ xuất hiện ở chính giữa màn hình.



Đóng chương trình, đặt thuộc tính StarPosition thành Manual. Với thuộc tính này bạn cần đặt lại thuộc tính Location, ta đặt thuộc tính này là 100, 50.



Chạy thử chương trình. Form sẽ hiển thị theo tọa độ ta đã đặt.

b. Sử dụng thuộc tính DesktopBounds

Đặt thêm nút nhấn lên form1, đặt text là “Tạo form mới”.

Tạo thủ tục Button1_Click và nhập mã như sau:

//Tạo form thứ hai có tên Form2

```
Form form2= new Form();
```

//Định nghĩa thuộc tính Text và đường viền cho form

```
form2.Text = "Form mới";
```

```
form2.FormBorderStyle = System.Windows.Forms.FormBorderStyle.FixedDialog;
```

//Chỉ định vị trí của form được đặt giữa màn hình

```
form2.StartPosition = FormStartPosition.CenterScreen;
```

//Khai báo cấu trúc Rectangle nắm giữ kích thước mới

//Góc trái trên (200,100), Chiều dài và cao (300,250)

```
Rectangle _form2 = new Rectangle(200, 100, 300, 250);
```

//Định kích thước của form sử dụng đối tượng rectangle trên

```
form2.DesktopBounds = _form2;
```

//Hiển thị form

```
form2.ShowDialog();
```

Bạn chạy chương trình này bằng cách ấn F5. Nhấn vào nút “tạo form mới” để tạo form thứ hai. Form này có vị trí như ta đã định. Form này không cho phép bạn kéo lại kích thước như các form trước đây do ta đã đặt thuộc tính FormBorderStyle của form thành FixedDialog.

c. *Phóng to, thu nhỏ và khôi phục lại cửa sổ chương trình*

Ngoài ra bạn cũng có thể phóng to, thu nhỏ hay khôi phục lại vị trí mặc định của form. Bạn có thể thực hiện điều này khi thiết kế hay khi chương trình đang chạy.

Để làm điều này trước hết bạn cần cho hiện hai nút Maximize và minimize ở góc phải trên chương trình bằng hai thuộc tính:

```
MaximizeBox = True; MinimizeBox = True;
```

Tiếp đến trong mã chương trình hay trong cửa sổ thuộc tính bạn đặt thuộc tính WindowState như sau:

```
WindowState = FormWindowState.Minimized
```

Nếu bạn muốn kiểm soát kích thước phóng to, thu nhỏ cho phép của form bạn đặt thuộc tính MinimumSize, MaximumSize. Hai thuộc tính này có kiểu cấu trúc Size giống như cấu trúc Rectangle, ví dụ:

```
Size s = New Size(400, 300);
```

```
MaximumSize = s;
```

5.1.3. *Thêm vào các điều khiển lúc form đang chạy*

Ta thường đưa các điều khiển trên Toolbox khi thiết kế form. Bạn cũng có thể đưa chúng vào trong form khi chương trình đang chạy – tạo điều khiển động. Quy trình để đưa như sau:

Khai báo biến đối tượng có kiểu lớp của phần tử giao diện mà bạn muốn đưa vào, ví dụ:

```
Button btnOk = new Button();
```

Thiết lập thuộc tính cho các nút nhấn sau khi đã khai báo như trên:

//Đặt thuộc tính cho nút nhấn

```
btnOk.Text = "OK";
```

```
btnOk.Location = new Point(110, 100);
```

Đưa đối tượng vào form. Để thực hiện điều này, bạn đưa các đối tượng vào tập hợp Controls của form bằng phương thức Add:

```
form2.Controls.Add(btnOk);
```

Bài tập MyAddControls sau đây sẽ minh họa cụ thể hơn:

Bạn tạo một Solution mới và thêm vào một Project có cùng tên như trên. Thiết kế form1 có một nút nhấn với thuộc tính text là “Hiển thị ngày”. Khi người dùng click vào đây thì một form mới sẽ được tạo ra. Khi form này tạo ra thì đồng thời mã chương trình sẽ tạo hai điều khiển là nhãn lblNgay ghi ngày hiện hành và nút nhấn btnOK để đóng form thứ hai này lại.

Bạn tạo thủ tục Button1_Click và nhập mã như sau:

//Khai báo form và các đối tượng điều khiển

```
Form form2 = new Form();
```

```
Label lblNgay = new Label();
```

```
Button btnOK = new Button();
```

//Đặt thuộc tính text cho nhãn là thời gian hiện tại

```
lblNgay.Text = "Hôm nay là: " + DateTime.Now.ToShortDateString();
```

//Đặt thuộc tính Size (kích thước) cho nhãn là 150,50

```
lblNgay.Size = new Size(150, 50);
```

//Đặt thuộc tính Location (vị trí) cho nhãn là 80,50

```
lblNgay.Location = new Point(80, 50);
```

//Đặt thuộc tính text cho nút nhấn là OK

```
btnOK.Text = "OK";
```

//Đặt thuộc tính Location cho nút nhấn btnOK

```
btnOK.Location = new Point(110, 100);
```

//Đặt thuộc tính cho form mới

```
form2.Text = "Ngày hiện hành";
```

```
form2.CancelButton = btnOK;
```

```
form2.StartPosition = FormStartPosition.CenterScreen;
```

//Đưa các đối tượng mới vào tập hợp Controls

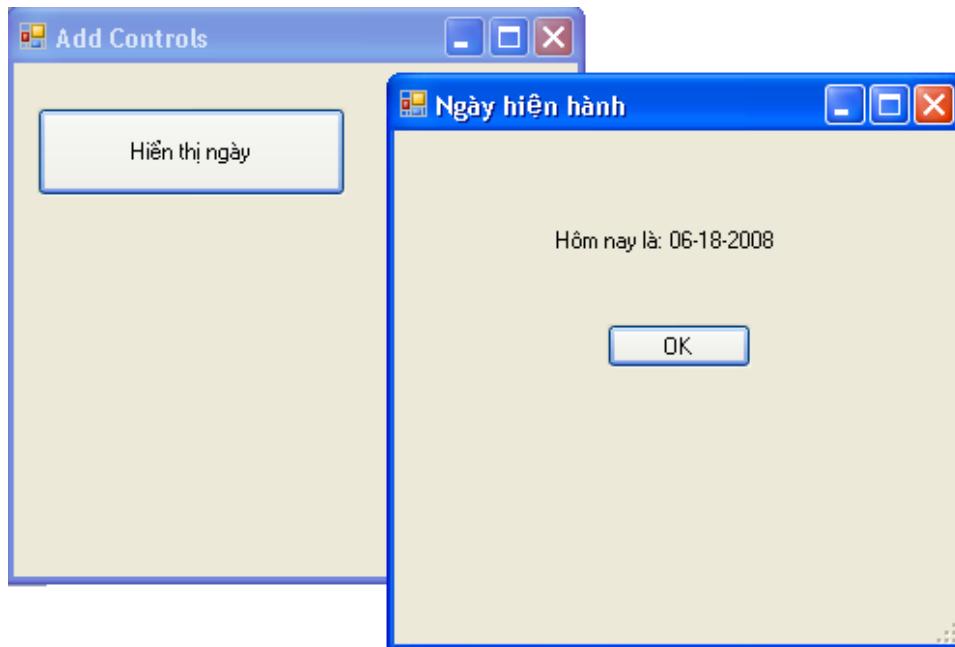
```
form2.Controls.Add(lblNgay);
```

```
form2.Controls.Add(btnOK);
```

//Gọi hiển thị form2

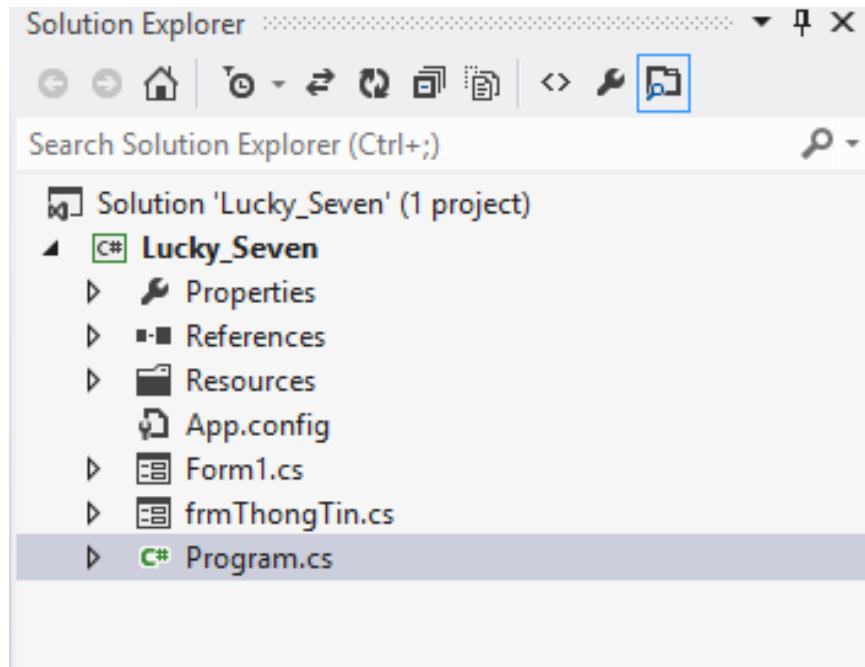
```
form2.ShowDialog();
```

Chạy chương trình và chúng ta sẽ thấy kết quả.



5.1.4. Chọn form khởi động của project

Để chọn form chạy khởi động, chúng ta mở cửa sổ Solution Explorer, mở file Program.cs:



Muốn đặt form nào làm form khởi động chúng ta sử dụng câu lệnh:

Application.Run (new Tên_Form_Khởi_Dộng());

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Lucky_Seven
{
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Form1());
        }
    }
}
```

5.2.Xây dựng chương trình quản lý đầu tiên

Trong phần này, chúng ta sẽ xây dựng một chương trình quản lý bán hàng đơn giản, chương trình sẽ quản lý các thông tin về hàng hóa (nhập, xuất, tồn), khách hàng và nhà cung cấp. Chương trình được xây dựng trên 2 nền tảng winform application và asp.net mvc. Để xây dựng được chương trình chúng ta sẽ thực hiện theo các bước sau:



Xây dựng cơ sở dữ liệu.



Viết mã cho chương trình.

5.3.1. Xây dựng cơ sở dữ liệu cho chương trình

Trong phần này, chúng ta sẽ tiến hành phân tích yêu cầu bài toán, từ đó đưa được các mô hình dữ liệu, và tiến hành xây dựng cơ sở dữ liệu trên hệ quản trị cơ sở dữ liệu SQL Server.

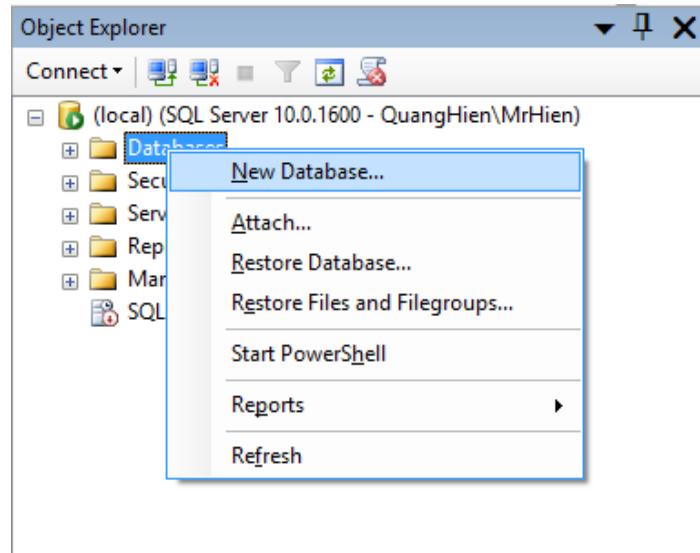
Dựa trên yêu cầu quản lý của bài toán, chúng ta tiến hành xây dựng các table lưu trữ thông tin hàng hóa, chi tiết nhập, chi tiết xuất, thông tin khách hàng, thông tin nhà cung cấp. Bao gồm các table sau:

- tblChiTietNhap: STT int (Is Identity = true), MaPhieuNhap varchar(20), MaHangHoa varchar(20), SoLuong int, NgayNhap date, MaNCC varchar(20)
- tblChiTietXuat: STT int (Is Identity = true), MaPhieuXuat varchar(20), MaHangHoa varchar(20), SoLuong int, NgayXuat date, MaKhachHang varchar(20)
- tblHangHoa: MaHangHoa varchar(20), TenHangHoa nvarchar(50), DonGia decimal(18, 0), DonViTinh nvarchar(50), MaNCC varchar(20)
- tblHoaDon: MaHoaDon varchar(20), NgayTao date, MaKhachHang varchar(20)
- tblKhachHang: MaKhachHang varchar(20), TenKhachHang nvarchar(50), SoDienThoai nvarchar(15)
- tblNhaCC: MaNCC varchar(20), TenNCC nvarchar(50), SoDienThoai nvarchar(15)
- tblNhapXuatTon: MaHangHoa varchar(20), Nhap int, Xuat int, Ton int
- tblPhieuNhap: MaPhieuNhap varchar(20), Ngaytao Date, MaNCC varchar (20)
- tblPhieuXuat: MaPhieuXuat varchar(20), Ngaytao Date, MaKhachHang varchar (20)

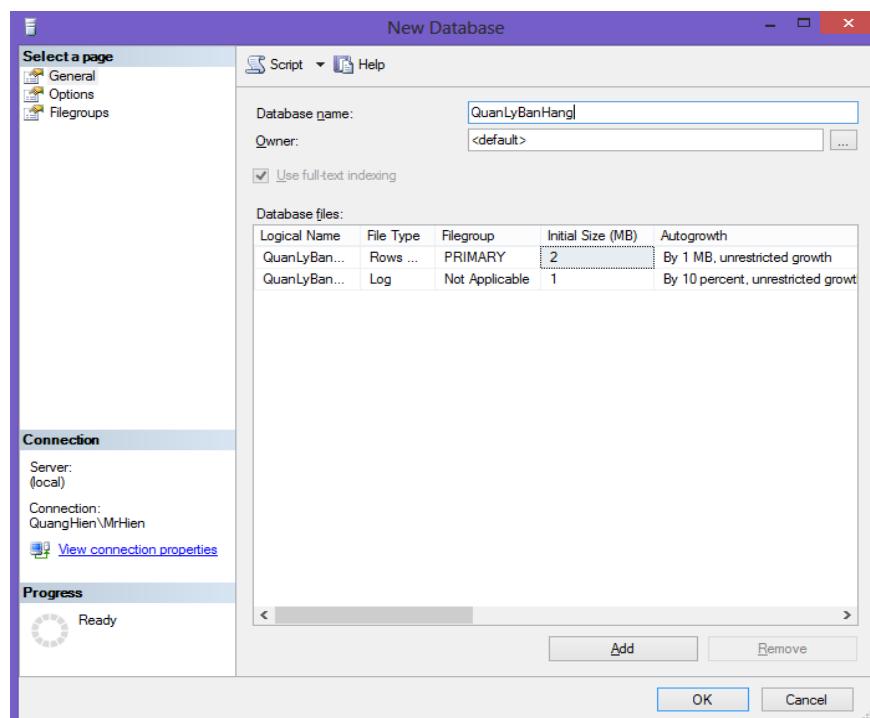
Để xây dựng cơ sở dữ liệu cho chương trình quản lý bán hàng trên SQL Sever ta có thể thực hiện theo 2 cách sau: Xây dựng cơ sở dữ liệu bằng ngôn ngữ SQL hoặc xây dựng cơ sở dữ liệu bằng trình Design của SQL Sever.

a. Xây dựng cơ sở dữ liệu bằng trình Design của SQL Sever

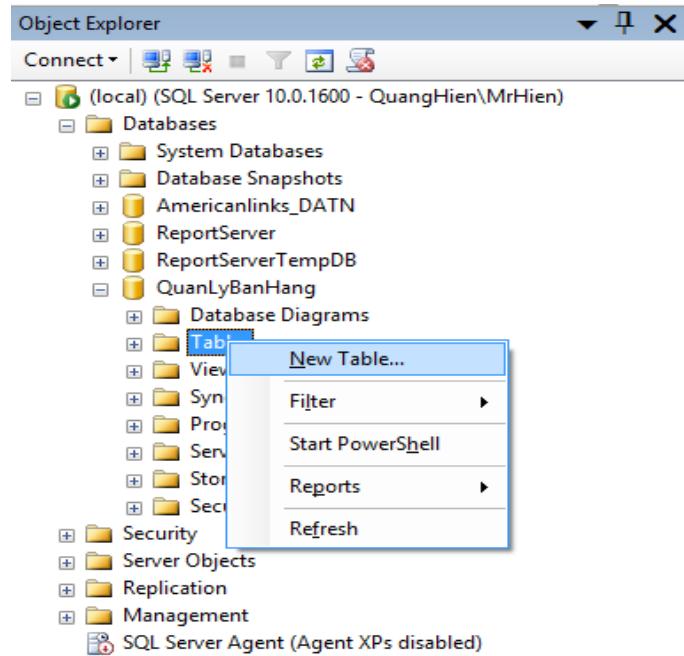
Sau khi cài đặt SQL Sever, chúng ta chạy Microsoft SQL Sever Management Studio, chọn Databases, click chuột phải chọn New Databases:



Đặt tên cho cơ sở dữ liệu là QuanLyBanHang, click và nút lệnh “Ok” để khởi tạo cơ sở dữ liệu:



Để tạo các table, chúng ta mở cơ sở dữ liệu bán hàng ra, click chuột phải vào table, chọn new table:

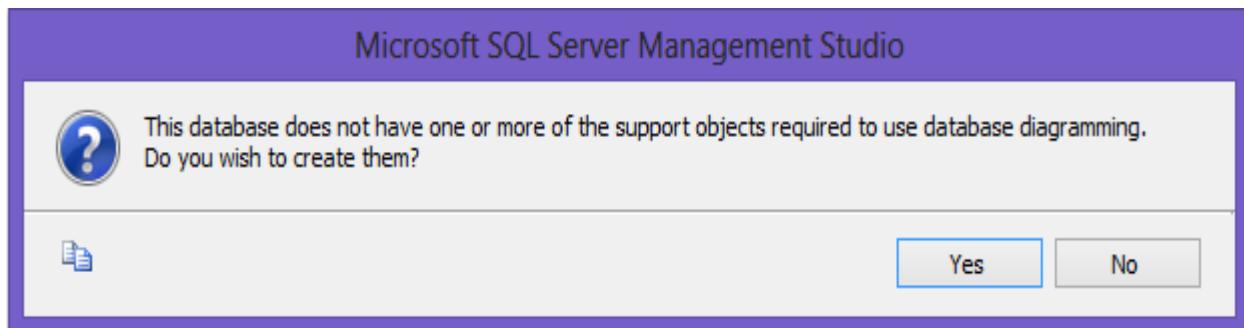


Khi đó sẽ hiện ra một table, chúng ta sẽ điền thông tin các trường trong bảng dữ liệu, ở đây sẽ ví dụ tạo bảng tblHangHoa, các bảng khác sẽ làm tương tự.

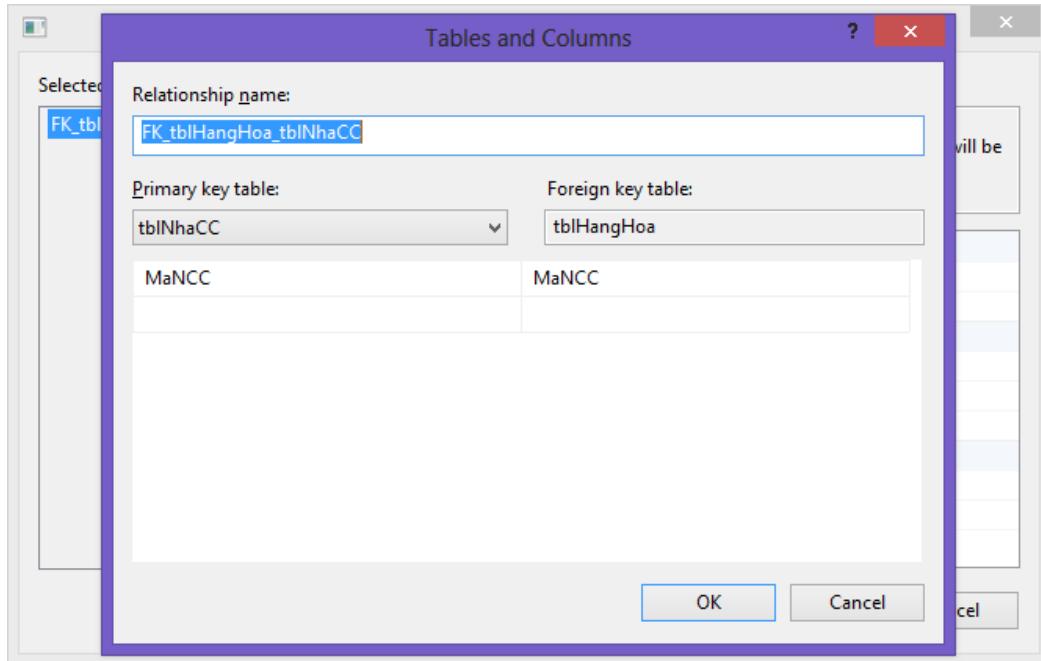
Sau khi nhập đầy đủ thông tin các trường dữ liệu trong bảng (Tên trường, kiểu dữ liệu) , nhấn tổ hợp phím “Ctrl + S”, xuất hiện ra một cửa sổ để tiền tên của bảng dữ liệu, click vào nút lệnh “Ok” để khởi tạo bảng dữ liệu.

A screenshot of the 'Choose Name' dialog box in SSMS. The dialog has a title bar 'Choose Name' and buttons for '?', 'X', 'OK', and 'Cancel'. The main area contains the placeholder text 'Enter a name for the table:' followed by a text input field containing 'tblHangHoa'. In the background, the table creation wizard interface is visible, showing columns MaHangHoa, TenHangHoa, DonGia, DonViTinh, and MaNCC with their respective data types and nullability settings.

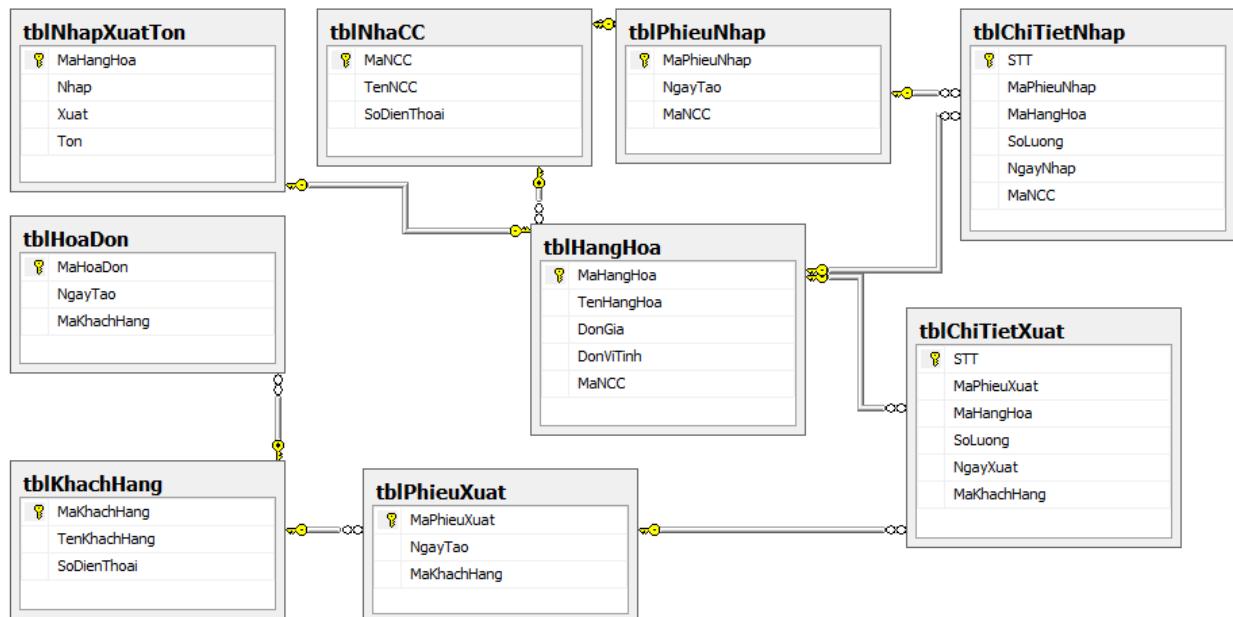
Sau khi tạo xong các table, chúng ta sẽ tạo liên kết giữa các bảng để đảm bảo tính toàn vẹn dữ liệu. Click vào Database Diagrams, sẽ xuất hiện một thông báo hỏi xem bạn có muốn tạo sơ đồ liên kết giữa các bảng hay không? Click vào nút lệnh “Yes” để tạo liên kết giữa các bảng:



Click chuột phải vào Database, chọn new Database Diagrams, hệ thống sẽ đưa ra một cửa sổ chứa các table, chọn các table muốn tạo liên kết rồi click vào nút lệnh Add, để tạo liên kết giữa hai bảng qua một trường dữ liệu, ta sẽ kéo trường dữ liệu đó từ một trong hai bảng muốn liên kết rồi thả vào bảng kia. Lúc này hệ thống sẽ đưa ra một cửa sổ đưa ra các thông tin về liên kết giữa hai bảng chúng ta muốn tạo ra. Click vào nút lệnh Ok để tạo liên kết:

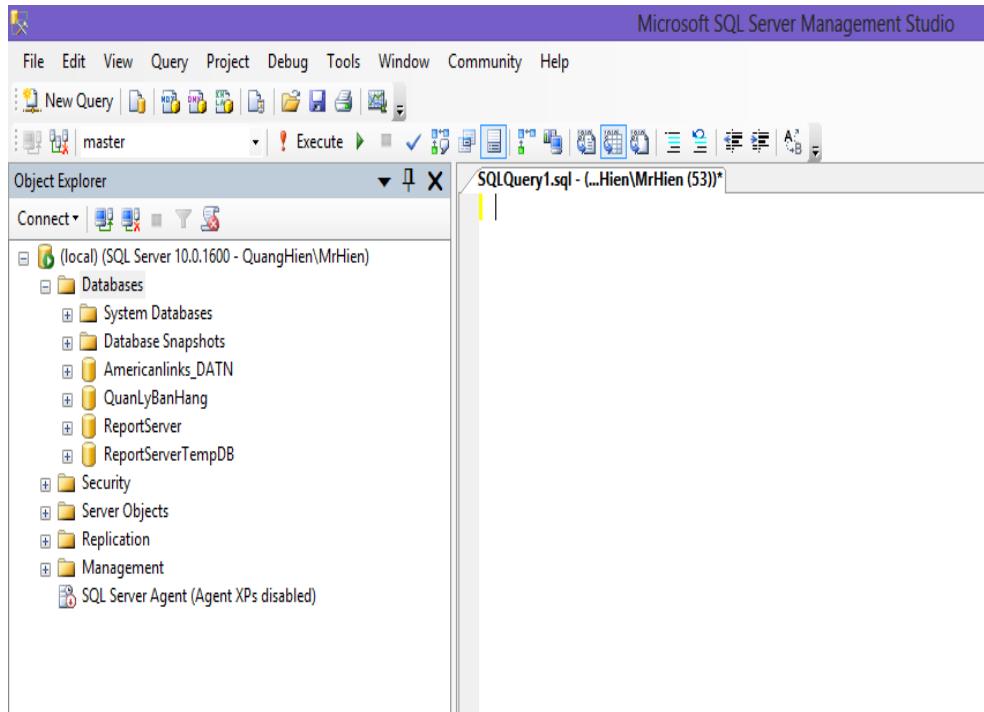


Chúng ta sẽ tạo liên kết như sau:



b. Xây dựng cơ sở dữ liệu bằng ngôn ngữ SQL

Để thực hiện các thao tác với cơ sở dữ liệu, chúng ta cần thực thi những câu lệnh SQL: như Create, Alter, Update, Delete, Insert, Select. Trong cửa sổ làm việc, chọn New Query, xuất hiện một cửa sổ để viết lệnh SQL:



Sau khi gõ lệnh SQL, để thực thi các lệnh đó, chúng ta sẽ click vào nút lệnh “Execute” để thực thi đoạn lệnh đó, hệ thống sẽ đưa ra thông báo xem đoạn lệnh vừa rồi có thực hiện thành công hay lỗi nào để người lập trình biết và sửa đổi.

Đầu tiên chúng ta sẽ tạo mới cơ sở dữ liệu QuanLyBanHang bằng cách thực thi câu lệnh:

CREATE DATABASE QuanLyBanHang

Sau khi tạo thành công cơ sở dữ liệu QuanLyBanHang, chúng ta tiếp tục khởi tạo các table như trên. Ở đây sẽ ví dụ tạo bảng tblChiTietNhap, các bảng khác sẽ làm tương tự:

-- Sử dụng cơ sở dữ liệu QuanLyBanHang để thực hiện các lệnh SQL

Use QuanLyBanHang

-- Tạo bảng tblChiTietNhap

```
create table tblChiTietNhap
(
    -- Identity thiết lập thuộc tính tự động tăng cho trường STT
    STT int IDENTITY PRIMARY KEY,
    MaPhieuNhap varchar (20) not null,
    MaHangHoa varchar (20) not null,
    SoLuong int not null,
    NgayNhap date not null,
    MaNCC varchar (20) not null
)
```

c. Xây dựng các store procedure

Tạo store procedure có tên KhachHang_SelectAll có nhiệm vụ lấy tất cả thông tin từ table tblKhachHang:

```
create proc KhachHang_SelectAll
as
Select * from tblKhachHang
```

Tạo store procedure có tên KhachHang_Add có nhiệm vụ thêm mới dữ liệu vào table tblKhachHang:

```
create proc KhachHang_Add (@MaKhachHang varchar (20),
                            @TenKhachHang nvarchar (50),@SoDienThoai nchar (15))
as
insert into tblKhachHang (MaKhachHang, TenKhachHang, SoDienThoai)
values (@MaKhachHang, @TenKhachHang, @SoDienThoai)
```

Tạo store procedure có tên KhachHang_Update có nhiệm vụ sửa đổi dữ liệu trên table tblKhachhang:

```
create proc KhachHang_Update (@MaKhachHang varchar (20),
                                @TenKhachHang nvarchar (50),@SoDienThoai nchar (15))
as
```

```
update tblKhachHang set TenKhachHang = @TenKhachHang, SoDienThoai  
= @SoDienThoai where MaKhachHang = @MaKhachHang
```

Tạo store procedure có tên KhachHang_Delete có nhiệm vụ xóa dữ liệu trên table tblKhachhang:

```
create proc KhachHang_Delete (@MaKhachHang nchar (10))  
as  
delete from tblKhachHang where MaKhachHang = @MaKhachHang
```

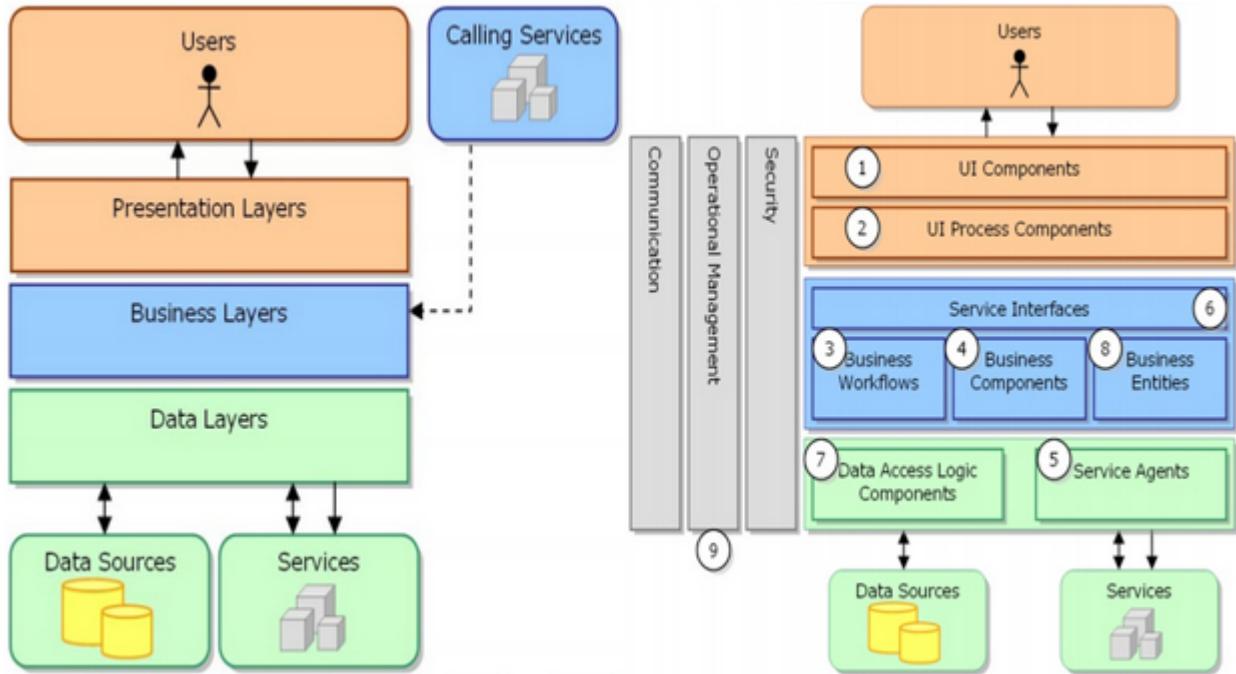
5.3.2. Viết mã cho chương trình trên nền tảng Winform Application

1. Giới Thiệu Về Mô hình 3 lớp

a. Giới thiệu

Khi bạn mới tiếp xúc với Windows Form và ADO.NET, việc lập trình bắt đầu trở lên phức tạp khi dự án lớn dần. Bởi vậy để dễ quản lý các thành phần của hệ thống, cũng như không bị ảnh hưởng bởi các thay đổi, người ta hay nhóm các thành phần có cùng chức năng lại với nhau và phân chia trách nhiệm cho từng nhóm để công việc không bị chồng chéo và ảnh hưởng lẫn nhau. Một trong những mô hình lập trình như vậy đó là Mô hình 3 lớp (Three Layers).

Mô hình 3 lớp được cấu thành từ: **Presentation Layers**, **Business Layers**, và **Data Layers**. Các lớp này sẽ giao tiếp với nhau thông qua các dịch vụ (services) mà mỗi lớp cung cấp để tạo nên ứng dụng, lớp này cũng không cần biết bên trong lớp kia làm gì mà chỉ cần biết lớp kia cung cấp dịch vụ gì cho mình và sử dụng nó mà thôi.



Hình: Kiến trúc mô hình 3 lớp

b. Các thành phần của mô hình 3 lớp

Presentation Layers

Lớp này làm nhiệm vụ giao tiếp với người dùng cuối để thu thập dữ liệu và hiển thị kết quả/dữ liệu thông qua các thành phần trong giao diện người sử dụng. Lớp này sẽ sử dụng các dịch vụ do lớp Business Logic cung cấp. Trong .NET thì bạn có thể dùng **Windows Forms**, **ASP.NET** hay **Mobile Forms** để hiện thực lớp này.

Trong lớp này có 2 thành phần chính là **User Interface Components** và **User Interface Process Components**.

UI Components: là những phần tử chịu trách nhiệm thu thập và hiển thị thông tin cho người dùng cuối. Trong ASP.NET thì những thành phần này có thể là các TextBox, các Button, DataGridView...

UI Process Components: là thành phần chịu trách nhiệm quản lý các qui trình chuyển đổi giữa các UI Components. Ví dụ chịu trách nhiệm quản lý các màn hình nhập dữ liệu trong một loạt các thao tác định trước như các bước trong một Wizard...

Lưu ý : Lớp này không nên sử dụng trực tiếp các dịch vụ của lớp Data Access mà nên sử dụng thông qua các dịch vụ của lớp Business Logic vì khi bạn sử dụng trực tiếp như vậy, chúng ta có thể bỏ qua các ràng buộc, các logic nghiệp vụ mà ứng dụng cần phải có.



Business Logic Layer

Lớp này thực hiện các nghiệp vụ chính của hệ thống, sử dụng các dịch vụ do lớp Data Access cung cấp, và cung cấp các dịch vụ cho lớp Presentation. Lớp này cũng có thể sử dụng các dịch vụ của các nhà cung cấp thứ 3 để thực hiện công việc của mình.

Trong lớp này có các thành phần chính là Business Components, Business Entities và Service Interface.

Service Interface: là giao diện lập trình mà lớp này cung cấp cho lớp Presentation sử dụng. Lớp Presentation chỉ cần biết các dịch vụ thông qua giao diện này mà không cần phải quan tâm đến bên trong lớp này được hiện thực như thế nào.

Business Entities: là những thực thể mô tả những đối tượng thông tin mà hệ thống xử lý. Các Business Entities này cũng được dùng để trao đổi thông tin giữa lớp Presentation và lớp Data Layers.

Business Components: là những thành phần chính thực hiện các dịch vụ mà Service Interface cung cấp, chịu trách nhiệm kiểm tra các ràng buộc logic (constraints), các qui tắc nghiệp vụ (Business Rules), sử dụng các dịch vụ bên ngoài khác để thực hiện các yêu cầu của ứng dụng.



Data Layer

Lớp này thực hiện các nghiệp vụ liên quan đến lưu trữ và truy xuất dữ liệu của ứng dụng. Thường lớp này sẽ sử dụng các dịch vụ của các hệ quản trị cơ sở dữ liệu như SQL Server, Oracle ... để thực hiện nhiệm vụ của mình. Trong lớp này có các thành phần chính là Data Access Logic, Data Sources, Servive Agents).

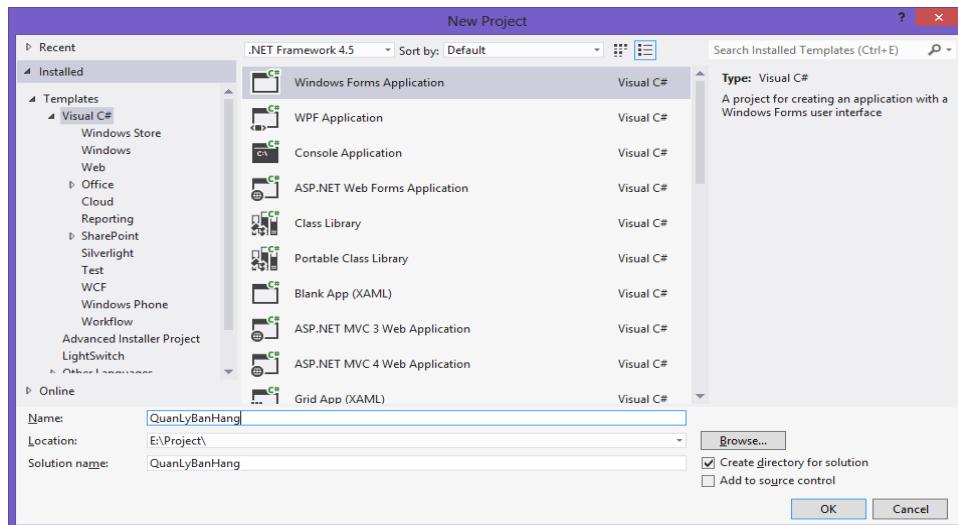
Data Access Logic Components (DAL) là thành phần chính chịu trách nhiệm lưu trữ vào và truy xuất dữ liệu từ các nguồn dữ liệu – Data Sources như RDMBS, XML, File systems.... Trong .NET Các DAL này thường được hiện thực bằng cách sử dụng thư viện ADO.NET để giao tiếp với các hệ cơ sở dữ liệu hoặc sử dụng các O/R Mapping Frameworks để thực hiện việc ánh xạ các đối tượng trong bộ nhớ thành dữ liệu lưu trữ trong CSDL. Chúng ta sẽ tìm hiểu các thư viện O/R Mapping này trong một bài viết khác.

Service Agents: là những thành phần trợ giúp việc truy xuất các dịch vụ bên ngoài một cách dễ dàng và đơn giản như truy xuất các dịch vụ nội tại.

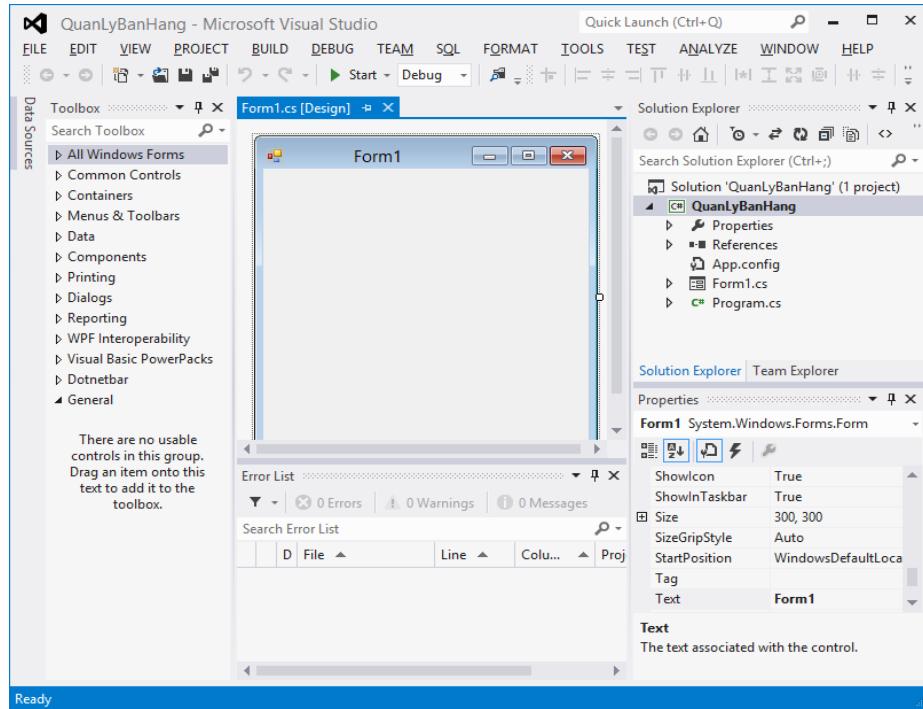
2. Xây dựng giao diện và lập trình



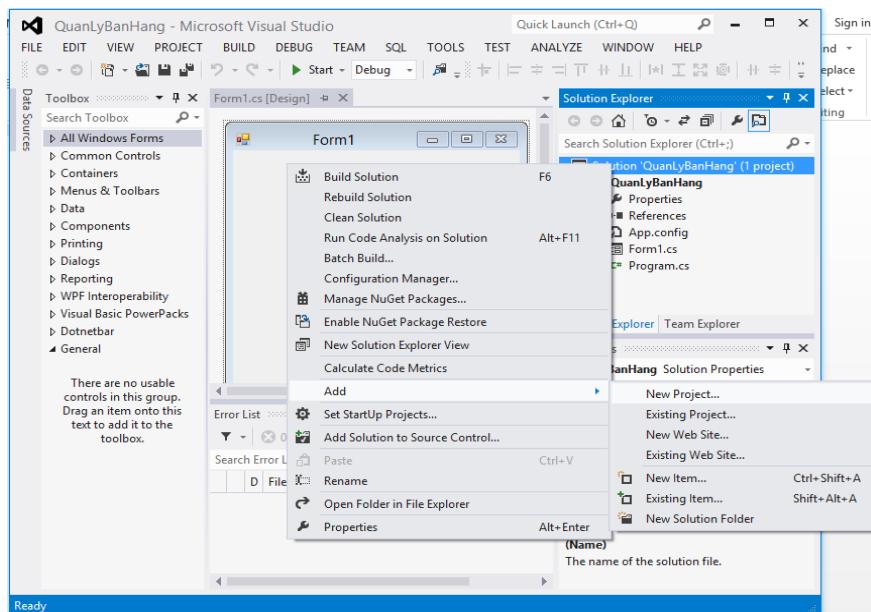
Khởi động visual studio, chọn tạo mới một project (Windows Form Application C#), đặt tên cho project là QuanLyBanHang:



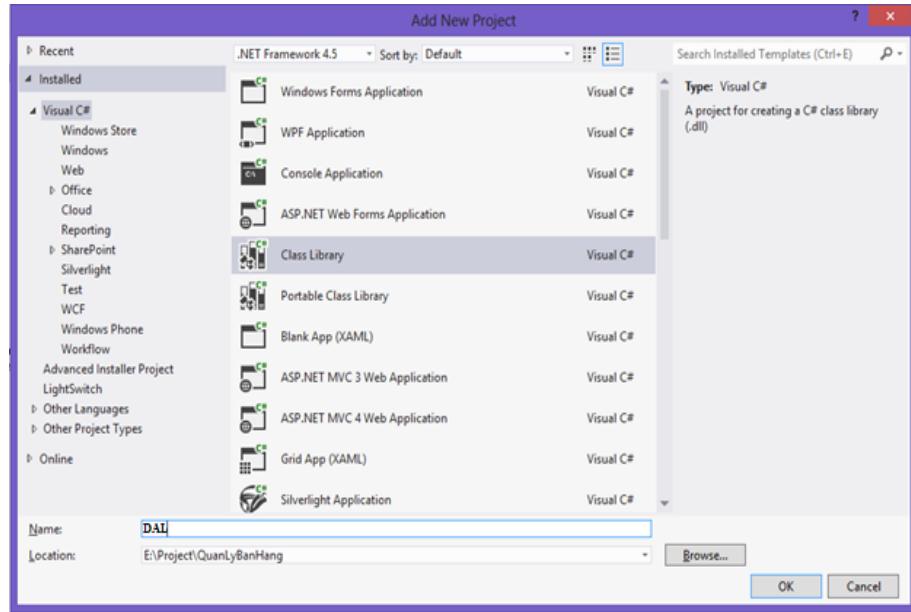
Sau khi khởi tạo thành công, cửa sổ mới chứa project vừa tạo sẽ được mở ra:



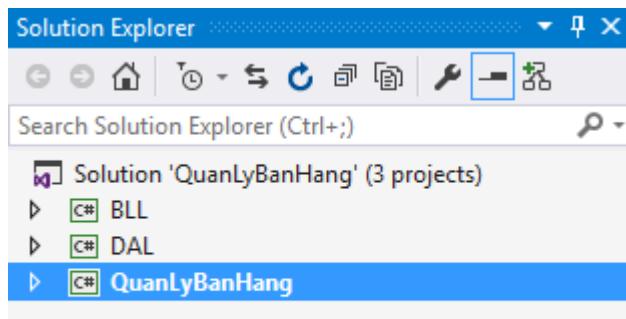
Trong cửa sổ Solution Explorer, click chuột phải vào solution, chọn Add, chọn New Project:



Một cửa sổ mới xuất hiện, Chọn C# Class Library đặt tên cho Project là DAL, sau đó click vào nút lệnh Ok để thêm project vào solution:



Tương tự Add thêm project BLL vào solution:

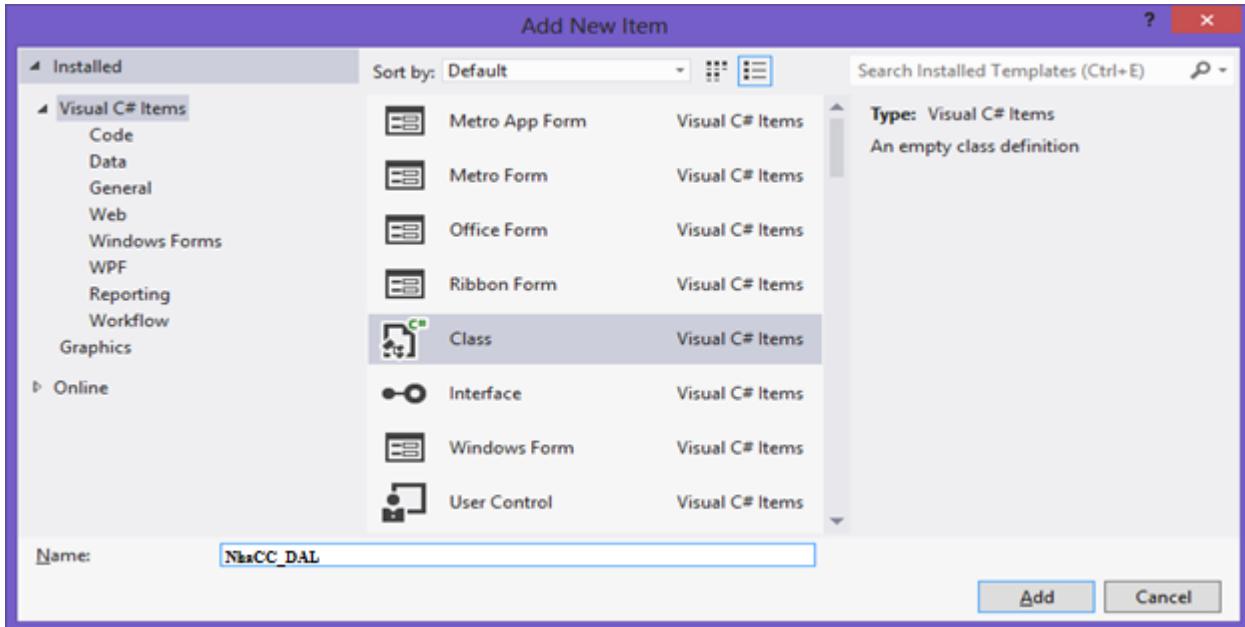


Sau đây chúng ta sẽ làm việc với từ project trong solution: DAL (chứa các khai báo về các trường dữ liệu trong các table), BLL (gọi và truyền tham số cho các store procedure và thực thi các store procedure), QuanLyBanHang (Thiết kế giao diện để giao tiếp với người sử dụng, tương ứng với lớp Presentation).



Xử lý lớp DAL:

Click chuột phải vào lớp DAL chọn Add => New Item (hoặc nhấn tổ hợp phím Ctrl + Shift + A), một cửa sổ mới xuất hiện, chúng ta chọn Class, đặt tên cho Class là NhaCC_DAL, rồi click nút lệnh Add để thêm class vào project DAL:



Mỗi một table chúng ta sẽ tạo một class tương ứng trong lớp DAL.

Sau đó viết mã cho class NhaCC_DAL như sau:

// Gọi các thư viện trong C#

```
using System;
namespace DAL
{
    public class NhaCC_DAL
    {
        // Khai báo các trường và kiểu dữ liệu tương ứng trong table tblNhaCC

        public string MaNCC{ get; set; }
        public string TenNCC{ get; set; }
        public string SoDienThoai{ get; set; }

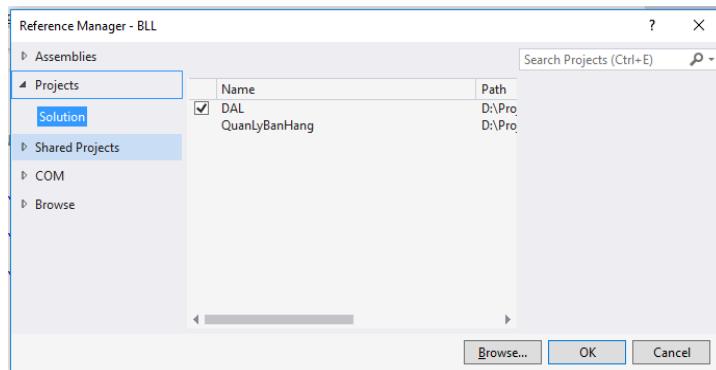
    }
}
```

Tương tự, chúng ta sẽ tạo thêm các class KhachHang_DAL, HangHoa_DAL, ChiTietNhap_DAL, ChiTietXuat_DAL, NhaphXuatTon_DAL trong project DAL.



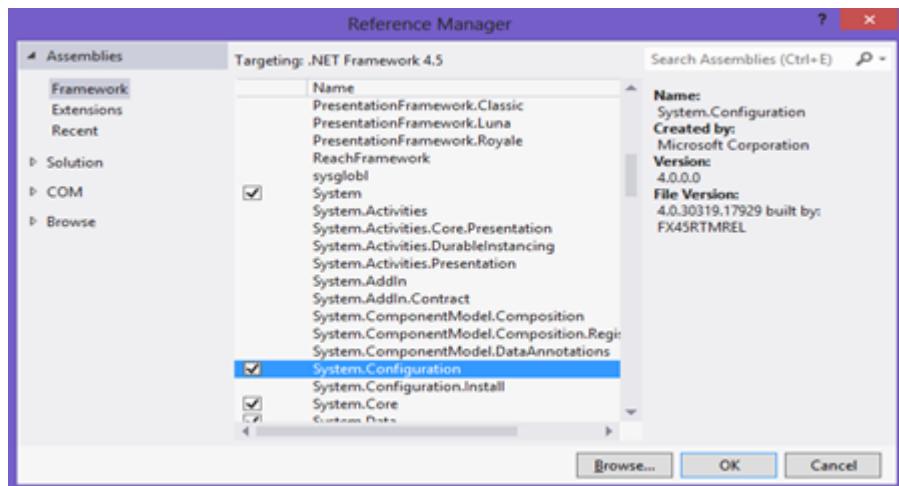
Xử lý lớp BLL:

Lớp BLL sẽ kế thừa lớp từ lớp DAL, để lớp BLL có thể kế thừa lớp DAL chúng ta làm như sau: click chuột phải vào References trong lớp BLL chọn Add Reference:



Vào thẻ Solution chọn DAL rồi click vào nút lệnh Ok (sau khi thêm lớp DAL vào Reference thì lớp BLL mới kế thừa được lớp DAL).

Tiếp theo, vào thẻ Framework Add thêm System.Configuration vào Reference của lớp BLL để sử dụng file App.config (giới thiệu ở phần sau).



Tiếp theo, chúng ta sẽ tạo một class có tên là KetNoi_BLL (chứa các hàm C# để thực thi các câu lệnh), viết mã cho class KetNoi_BLL như sau:

// Gọi các thư viện cần sử dụng trong lớp BLL:

```
using System.Configuration;
using System.Data;
using System.Data.SqlClient;
```

// Khai báo đối tượng SqlConnection để kết nối đến cơ sở dữ liệu

// Truyền đường dẫn đến cơ sở dữ liệu trong file App.Config

```
SqlConnection con = new SqlConnection
```

```
(ConfigurationManager.ConnectionStrings["project"].ConnectionString);
```

// Hàm mở kết nối, kiểm tra xem trạng thái kết nối với cơ sở dữ liệu có phải
là đóng kết nối hay không? Nếu đang đóng kết nối thì mở kết nối ra.

```
public void clsketnoi()
```

```
{
```

```
    if (con.State == ConnectionState.Closed)      con.Open();
```

```
}
```

// Hàm lấy dữ liệu không chứa tham số

```
public DataTable laydulieu(string sql)
```

```
{
```

```
    clsketnoi();
```

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

```
    cmd.CommandType = CommandType.StoredProcedure;
```

```
    SqlDataAdapter adapter = new SqlDataAdapter(cmd);
```

```
    DataTable dt = new DataTable();
```

```
    adapter.Fill(dt);      return dt;
```

```

    }

// Hàm lấy dữ liệu chứa tham số

public DataTable laydulieu(string sql, string[] name, object[] value,
                           int Nparameter)

{
    clsketnoi ();
    SqlCommand cmd = new SqlCommand(sql,con);
    cmd.CommandType = CommandType.StoredProcedure;
    for (int i = 0; i < Nparameter; i++)
    {
        cmd.Parameters.AddWithValue(name[i], value[i]);
    }
    SqlDataAdapter adapter = new SqlDataAdapter(cmd);
    DataTable dt = new DataTable();
    adapter.Fill(dt);
    return dt;
}

```

Tương tự như lớp DAL, tương ứng với mỗi table chúng ta cũng sẽ tạo một class trong lớp BLL. Mỗi class trong lớp BLL sẽ chứa các hàm dùng để thực thi các store procedure tương ứng với mỗi table.

Mã của lớp BLL sẽ được viết như sau (ví dụ viết trên class NhaCC_BLL):

```
// Gọi các hàm thư viện trong C# mà lớp BLL cần sử dụng
```

```

using System.Configuration;
using System.Data;
using System.Data.SqlClient;

```

```
// Khai báo lớp DAL
    using DAL;
// Khai báo class KetNoi_BLL
    KetNoi_BLL cls = new KetNoi_BLL ();
// Khai báo chuỗi kết nối trong file App.Config
    SqlConnection con = new
        SqlConnection(ConfigurationManager.ConnectionStrings["project"].
    ConnectionString);
// Hàm lấy toàn bộ dữ liệu trong table tblNhaCC
// Sau khi gọi hàm giá trị trả về sẽ là một DataTable
    public DataTable NhaCC_SelectAll ()
    {
        // Gọi hàm laydulieu trong class KetNoi_BLL để thực thi store
        procedure NhaCC_SelectAll
        return cls.laydulieu("NhaCC_SelectAll ");
    }
// Hàm thêm mới dữ liệu vào table tblNhaCC
public void NhaCC_Add (NhaCC_DAL d)
{
    // Kiểm tra nếu kết nối đến cơ sở dữ liệu là đóng thì mở kết nối
    if (con.State == ConnectionState.Closed)      con.Open();
    // Khai báo một SqlCommand để thực thi store procedure
    SqlCommand cmd = new SqlCommand ("NhaCC_Add", con);
    cmd.CommandType = CommandType.StoredProcedure;
    // Truyền các tham số đã được khai báo trong store procedure
    cmd.Parameters.Add (new SqlParameter ("@MaNCC", d.MaNCC));
    cmd.Parameters.Add (new SqlParameter ("@TenNCC", d.TenNCC));
```

```

cmd.Parameters.Add (new SqlParameter ("@SoDienThoai", d.SoDienThoai));

    // Thực thi Command

cmd.ExecuteNonQuery ();

    // Ngắt kết nối đến cơ sở dữ liệu

con.Close ();

}

// Hàm cập nhật dữ liệu vào table tblNhaCC

public void NhaCC_Update (NhaCC_DAL d)

{

if (con.State == ConnectionState.Closed)      con.Open();

SqlCommand cmd = new SqlCommand ("NhaCC_Update", con);

cmd.CommandType = CommandType.StoredProcedure;

cmd.Parameters.Add (new SqlParameter ("@MaNCC", d.MaNCC));

cmd.Parameters.Add (new SqlParameter ("@TenNCC", d.TenNCC));

cmd.Parameters.Add (new SqlParameter ("@SoDienThoai", d.SoDienThoai));

cmd.ExecuteNonQuery ();

con.Close ();

}

//Hàm xóa dữ liệu trong table tblNhaCC Biến “Ma” chính là tham số MaNCC

public void NhaCC_Delete (string Ma)

{

if (con.State == ConnectionState.Closed)      con.Open();

SqlCommand cmd = new SqlCommand ("NhaCC_Delete", con);

cmd.CommandType = CommandType.StoredProcedure;

cmd.Parameters.Add (new SqlParameter ("@MaNCC", Ma));

cmd.ExecuteNonQuery ();

con.Close ();

```

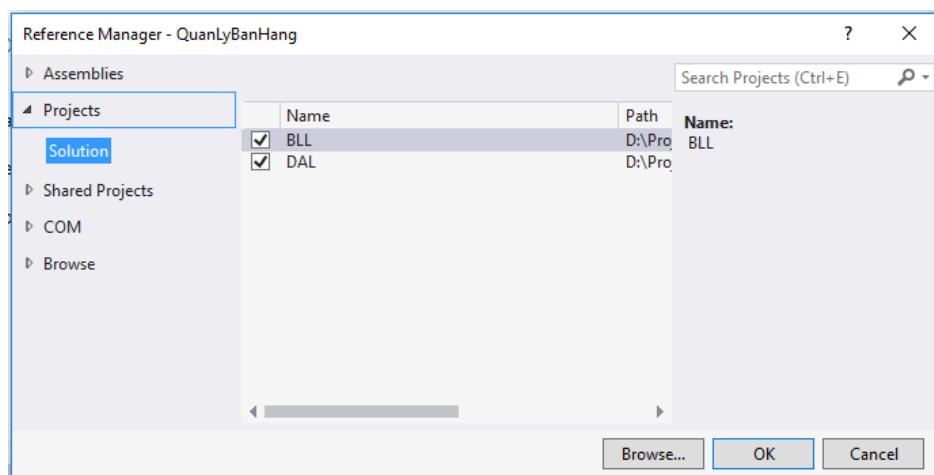
}

Tương tự, chúng ta sẽ tạo thêm các class KhachHang_BLL, HangHoa_BLL, ChiTietNhap_BLL, ChiTietXuat_BLL, NhapXuatTon_BLL trong project BLL.



Xử lý lớp QuanLyBanHang:

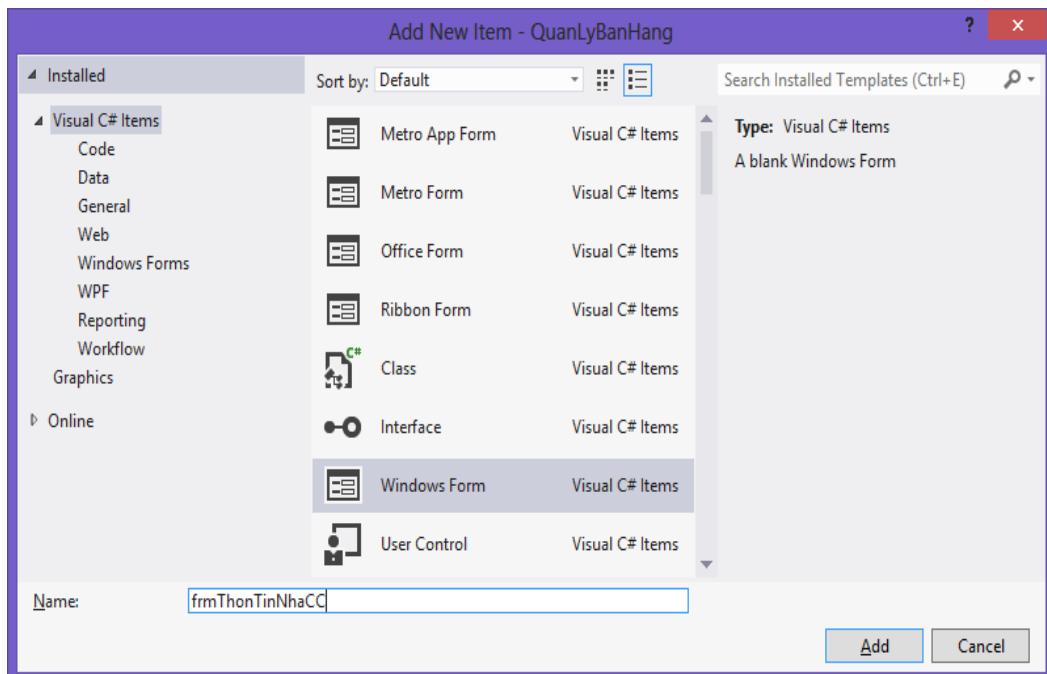
Lớp QuanLyBanHang sẽ kết thừa lớp từ lớp DAL và BLL để lớp QuanLyBanHang có thể kế thừa lớp DAL và BLL chúng ta làm như sau: click chuột phải vào lớp QuanLyBanHang chọn Add Reference:



Tiếp theo, chúng ta vào lớp QuanLyBanHang tìm file App.config và viết mã cho file config đó như sau (bổ sung phần code trong vùng đóng khung):

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <startup>
        <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5"/>
    </startup>
    <connectionStrings>
        <add name="project" connectionString="Data Source=.;Initial Catalog=QuanLyBanHang;Integrated Security=True"
            providerName="System.Data.SqlClient"/>
    </connectionStrings>
</configuration>
```

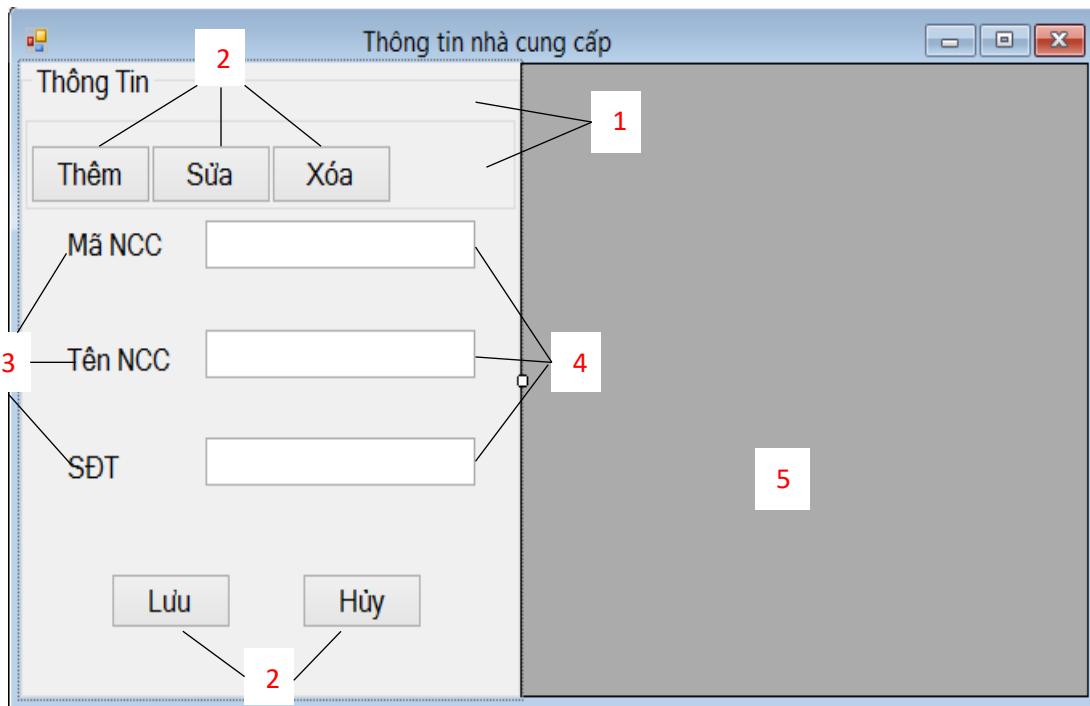
Sau khi thêm file config để lưu đường dẫn kết nối đến cơ sở dữ liệu, chúng ta sẽ Add thêm một form mới tên là frmThongTinNhaCC vào lớp QuanLyBanHang để hiển thị các thông tin về nhà cung cấp cũng như thực hiện các thao tác với về dữ liệu với table tblNhaCC:



Chỉnh các thuộc tính cho form frmThongTinNhaCC:

- Text = “Thông tin nhà cung cấp”
- StartPosition = CenterScreen
- Font Size = 12

Xây dựng giao diện cho frmThongTinNhaCC như sau:



1: Các GroupBox: GroupBox1 và GroupBox2 để gom nhóm các control



Đặt thuộc tính Dock cho GroupBox1 là Left, GroupBox2 là Top

2: các button: btThem, btSua, btXoa, btLuu, btHuy



Đặt thuộc tính Dock cho btThem, btSua, btXoa là Left

3: Các label

4: các textbox: txtMaNCC, txtTenNCC, txtSDT

5: Datagridview: dtgThongTin



Đặt thuộc tính Dock cho dtgThongTin là Fill



Đặt thuộc tính AutoSizeColumnsMode là Fill

Tiếp theo chúng ta sẽ viết mã cho frmThongTin, frmThongTin có nhiệm vụ cập nhật, hiển thị thông tin của các nhà cung cấp:

// Khai báo các thư viện cần sử dụng

```
using System;
using System.Data;
using System.Text;
using System.Windows.Forms;
using BLL;
using DAL;

// Khai báo hai class NhacCC_BLL và NhaCC_DAL

NhaCC_BLL nccbll = new NhaCC_BLL ();
NhaCC_DAL nccdal = new NhaCC_DAL ();

// Khai báo biến kt để kiểm tra xem ta click vào nút lệnh nào

String kt;

// Tạo sự kiện Load frmThongTinNhaCC và viết lệnh như sau:

private void frmThongTinNhaCC_Load(object sender, EventArgs e)
{
    // đổ dữ liệu vào dtgThongTin
    dtgThongTin.DataSource = nccbll.NhaCC_SelectAll ();
    // Gán giá trị cho biến kt bằng rỗng
    kt = "";
}

// Tạo sự kiện CellClick của dtgThongTin và viết lệnh như sau:

private void dtgThongTin_CellClick(object sender, DataGridViewCellEventArgs e)
{
    if (dtgThongTin.Rows.Count > 0)
    {
        txtMaNCC.Text = dtgThongTin.CurrentRow.Cells [0].Value.ToString ();
    }
}
```

```
txtTenNCC.Text = dtgThongTin.CurrentRow.Cells [1].Value.ToString ();  
txtSDT.Text = dtgThongTin.CurrentRow.Cells [2].Value.ToString ();  
}  
}
```

// Tạo sự kiện click của nút lệnh btThem và viết lệnh như sau:

```
private void btThem_Click(object sender, EventArgs e)  
{  
    // Gán giá trị cho biến kt bằng “Them”  
    kt = "Them";  
    // Gán text cho các textbox bằng rỗng  
    txtMaNCC.Text = txtTenNCC.Text = txtSDT.Text = "";  
    dtgThongTin.DataSource = nccbll.NhaCC_SelectAll ();  
}
```

// Tạo sự kiện click của nút lệnh btSua và viết lệnh như sau:

```
private void btSua_Click(object sender, EventArgs e)  
{  
    // Gán giá trị cho biến kt bằng “Sua”  
    kt = "Sua";  
}
```

// Tạo sự kiện click của nút lệnh btXoa và viết lệnh như sau:

```
private void btXoa_Click(object sender, EventArgs e)  
{  
    // Xóa thông tin của nhà cung cấp có mã bằng txtMaNCC.text  
    nccbll.NhaCC_Delete (txtMaNCC.Text);
```

```
// Nếu lệnh xóa thông tin được thực hiện thì thông báo
MessageBox.Show("Xóa thành công");
}

// Tạo sự kiện click của nút lệnh btLuu và viết lệnh như sau:
private void btLuu_Click(object sender, EventArgs e)
{
    // Gán giá trị cho các trường được khai báo trong store procedure
    nccdal.MaNCC = txtMaNCC.Text;
    nccdal.TenNCC = txtTenNCC.Text;
    nccdal.SoDienThoai = txtSDT.Text;
    // Kiểm tra xem có phải click nút thêm hay không
    if (kt == "Them")
    {
        // Thực hiện hàm Add dữ liệu vào tblNhaCC
        nccbll.NhaCC_Add(nccdal);
        MessageBox.Show("Thêm mới thành công!");
    }
    // Kiểm tra xem có phải click nút Sửa hay không
    else if (kt == "Sua")
    {
        // Thực hiện hàm Update dữ liệu vào tblNhaCC
        nccbll.NhaCC_Update(nccdal);
        MessageBox.Show("Sửa thành công!");
    }
    // Gọi sự kiện load form
    frmThongTinNhaCC_Load(sender, e);
}
```

// Tạo sự kiện click của nút lệnh btHuy và viết lệnh như sau:

```
private void btHuy_Click(object sender, EventArgs e)
{
    // Gọi sự kiện load form
    frmThongTinNhaCC_Load (sender, e);
```

```
}
```

Tương tự chúng ta sẽ xây dựng các form để hiện thị, cập nhật thông tin cho các table tblKhachHang, tblHangHoa, tblChiTietNhap, tblChiTietXuat, tblXuatNhapTon.

3. Tạo báo cáo với crystal report

Bước 1: Cài đặt crystal report (link download <http://tektutorialshub.com/install-crystal-reports-visual-studio/>)

Bước 2: Tạo một store procedure có nhiệm vụ lấy về tất cả dữ liệu từ table tblNhaCC (sử dụng để tạo báo cáo không có tham số).

```
create proc NhaCC_SelectAll
as
select * from tblNhaCC
```

Bước 3: Tạo một store procedure có nhiệm vụ lấy về dữ liệu từ table tblNhaCC với tham số truyền vào là mã nhà cung cấp (sử dụng để tạo báo cáo có tham số)

```
create proc NhaCC_Select_MaNCC
(@MaNCC varchar(10))
as
select * from tblNhaCC where MaNCC = @MaNCC
```

Chú ý: Phân biệt kiểu dữ liệu char và varchar, nên sửa lại kiểu dữ liệu của mã nhà cung cấp là varchar(10)

Bước 4: Trong class **NhaCC_BLL** viết hàm lấy dữ liệu từ 2 store procedure tạo ở bước 2 và 3:

```
public class NhaCC_BLL
{
    // Khai báo class KetNoi_BLL để sử dụng các hàm lấy dữ liệu
    KetNoi_BLL cls = new KetNoi_BLL();
    // Hàm lấy toàn bộ dữ liệu trong table tblNhaCC
    // Sau khi gọi hàm giá trị trả về sẽ là một DataTable
```

```

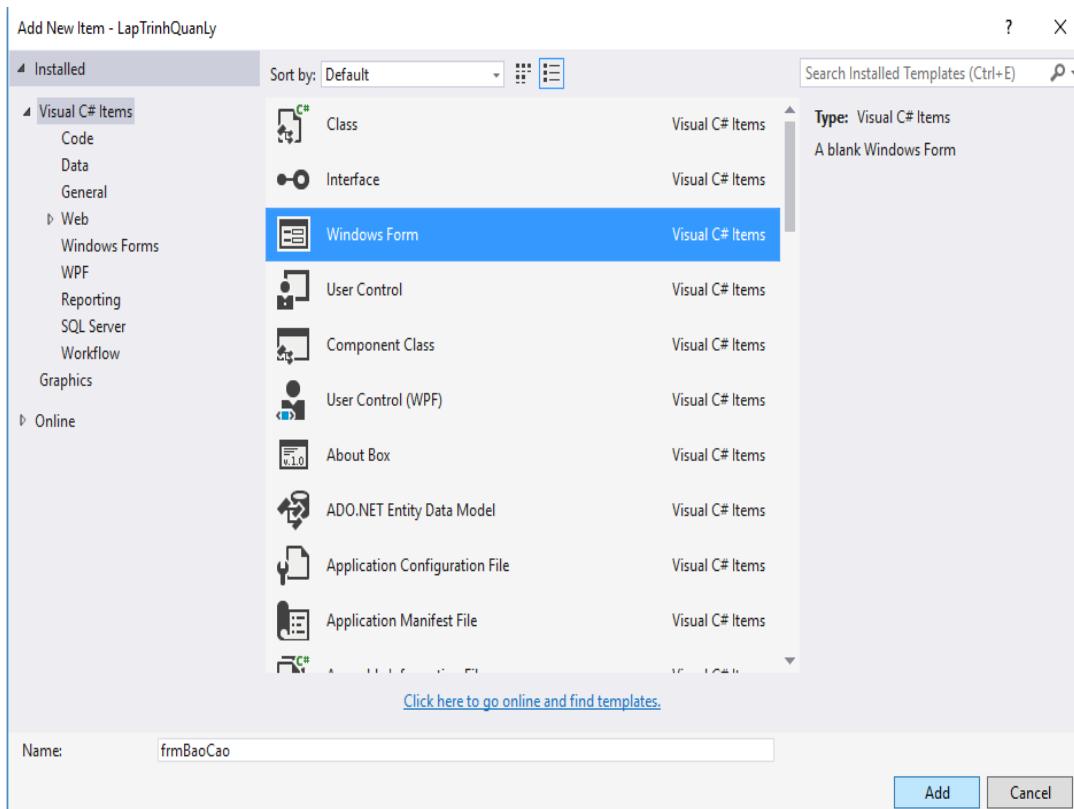
public DataTable NhaCC_SelectAll()
{
    // Gọi hàm laydulieu có tham số truyền vào duy nhất là tên của thủ tục
    return cls.laydulieu("NhaCC_SelectAll ");
}

public DataTable NhaCC_Select_MaNCC(string MaNCC)
{
    // Gọi hàm lấy dữ liệu với 4 tham số truyền vào gồm: tên storeprocedur, mảng
    // chứa tên các tham số, mảng chứa giá trị của các tham số và số tham số truyền vào

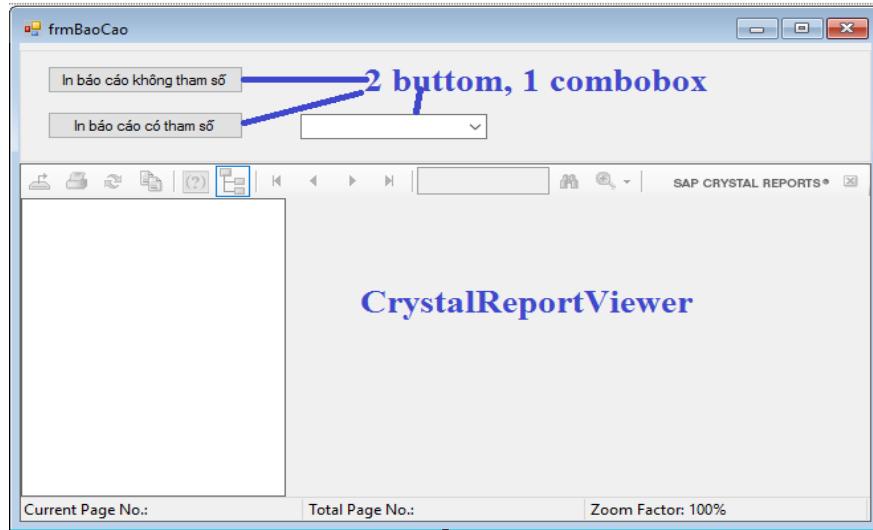
    // Khai báo tham số truyền vào là một mảng 1 phần tử chứa tên tham số là
    // NCC
    string[] DanhSachThamSo = new string[1];
    DanhSachThamSo[0] = "MaNCC";
    // Khai báo tham số truyền vào là một mảng 1 phần tử chứa giá trị tham số
    // là giá trị của biến MaNCC
    object[] GiaTriCacThamSo = new object[1];
    GiaTriCacThamSo[0] = MaNCC;
    // gọi thực thi hàm lấy dữ liệu với 4 tham số truyền vào, với tham số thứ
    // 4 có giá trị bằng 1
    return cls.laydulieu("NhaCC_Select_MaNCC", DanhSachThamSo,
    GiaTriCacThamSo,1);
}
}

```

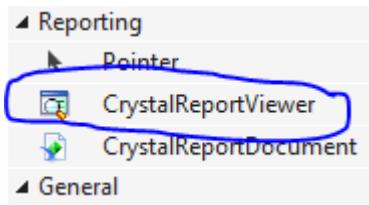
Bước 5: Tạo một form để hiển thị báo cáo đặt tên là frmBaoCao



Bước 6: Tạo giao diện cho frmBaoCao như sau:

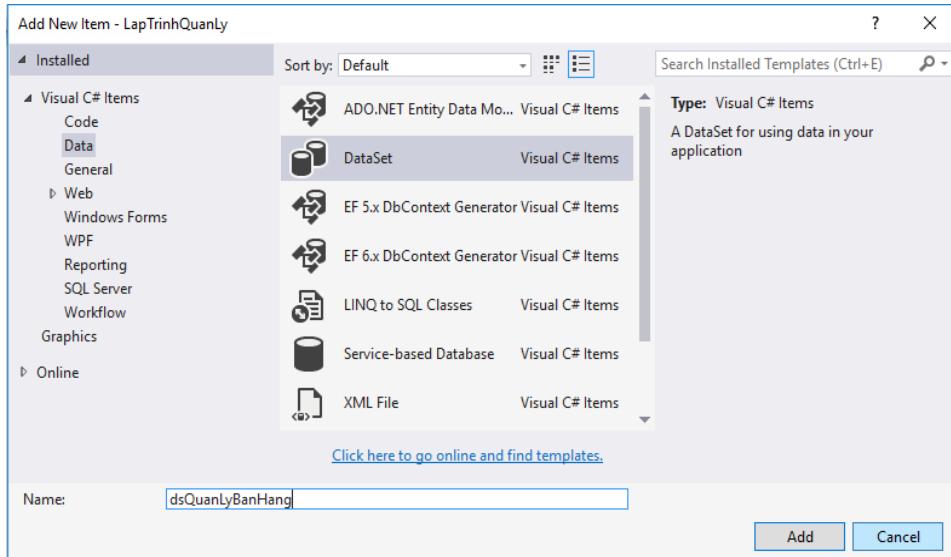


Thêm crystalreportviewer vào form trong Toolbox:

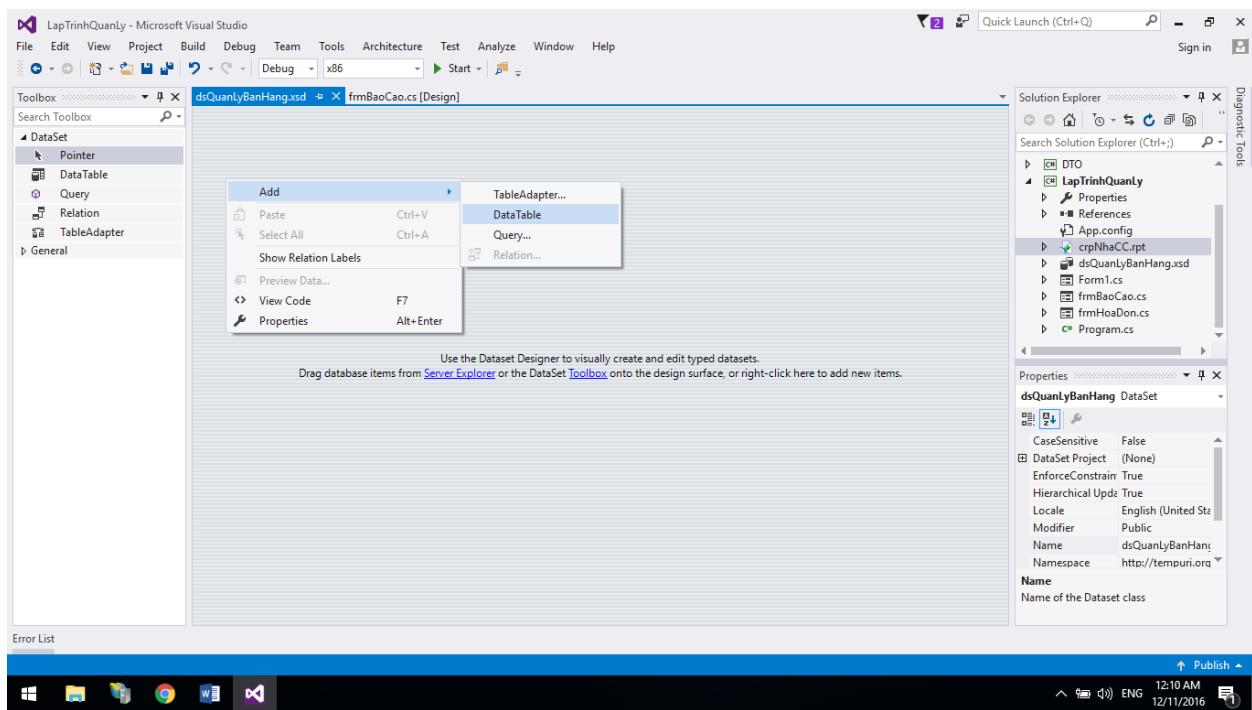


Đặt tên cho các control là: btInKhongThamSo, btInCoThamSo, cbMaNCC, RePort.

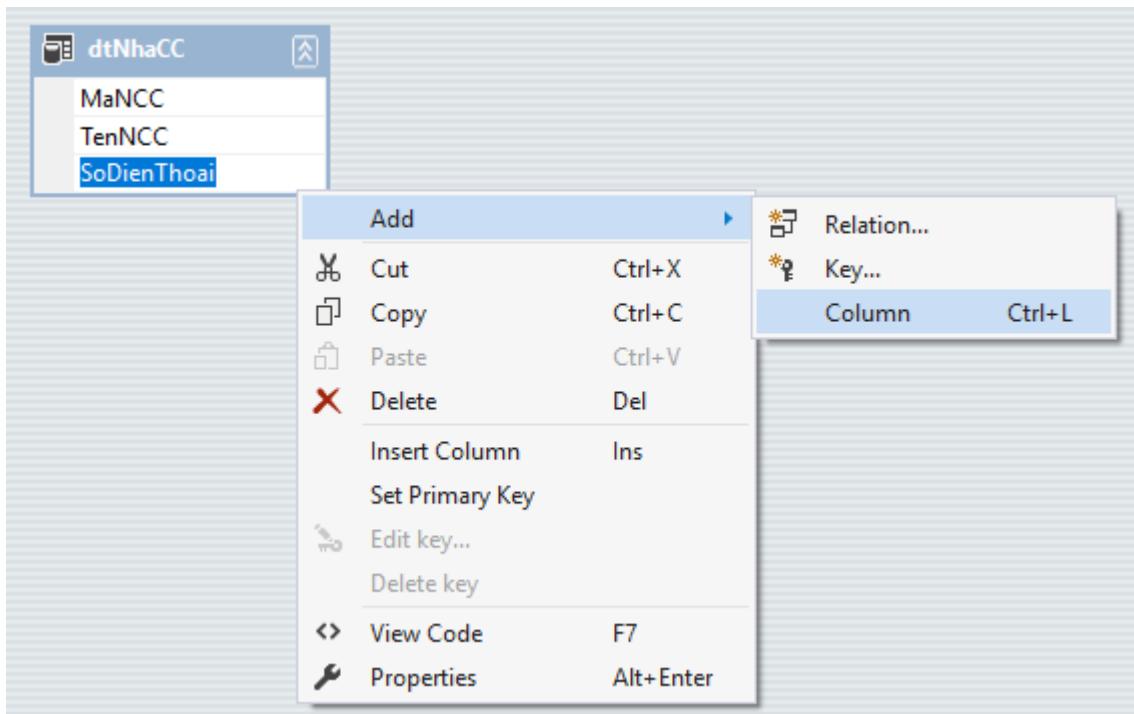
Bước 7: Tạo một dataset trong project QuanLyBanHang có tên là dsQuanLyBanHang



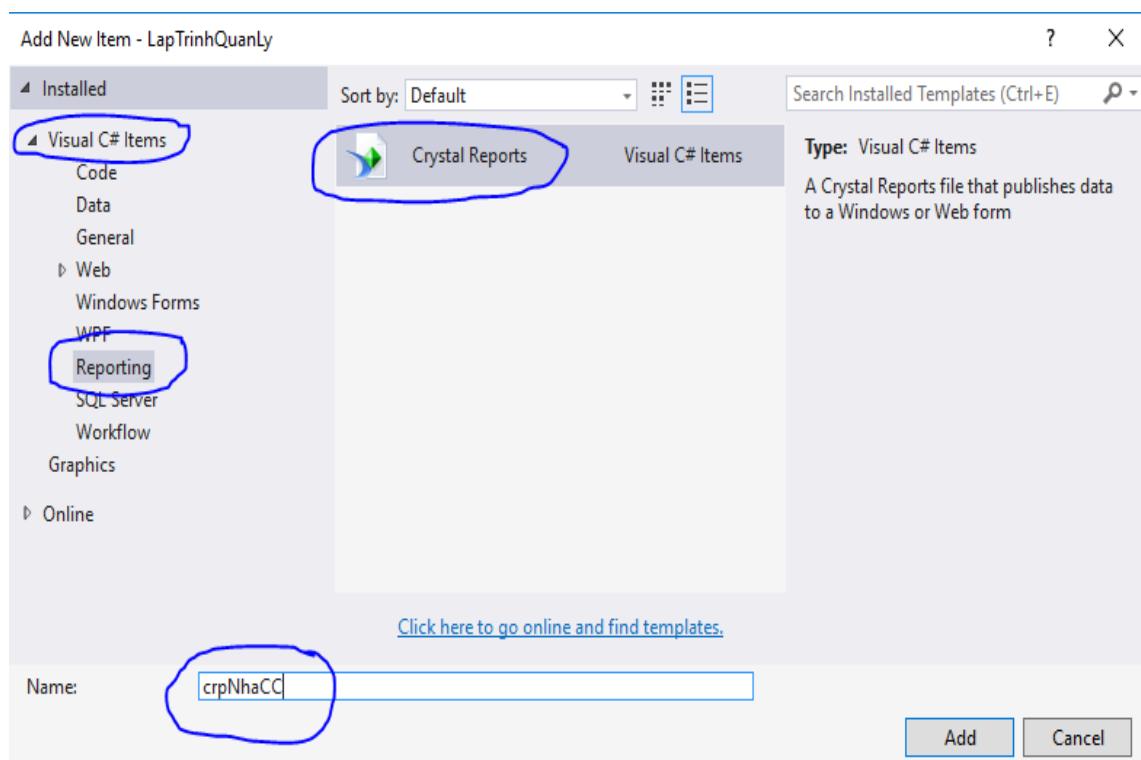
Bước 8: Add vào dsQuanLyBanhang một datatable, đặt tên là dtNhaCC



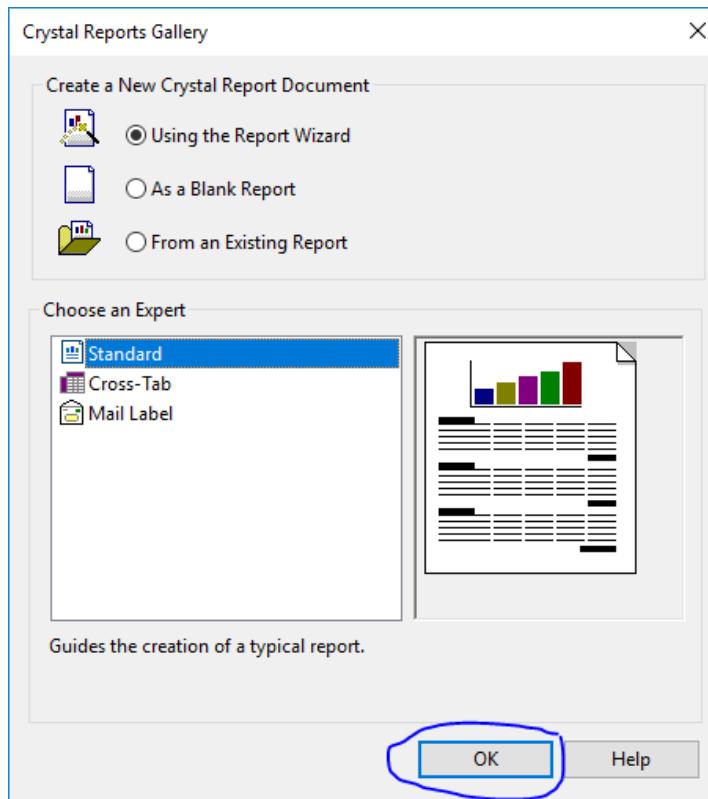
Sau đó add vào dtNhaCC 2 column MaNCC, TenNCC, SoDienThoai (lưu ý:
phải trùng với tên các trường dữ liệu trong cơ sở dữ liệu)



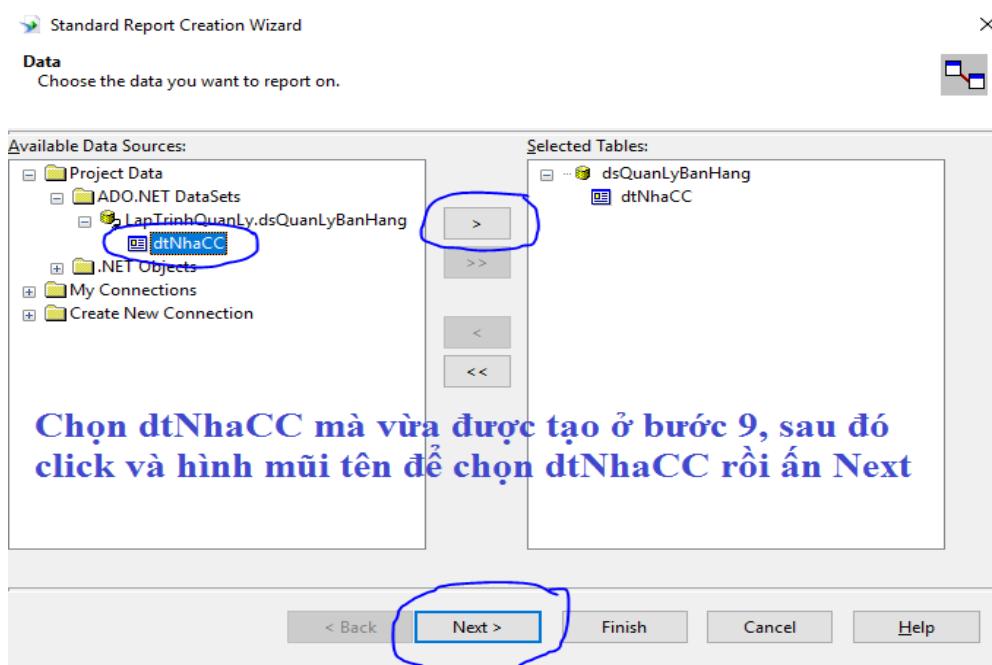
Bước 9: Add vào project QuanLyBanHang một crystal reports đặt tên là crpNhaCC



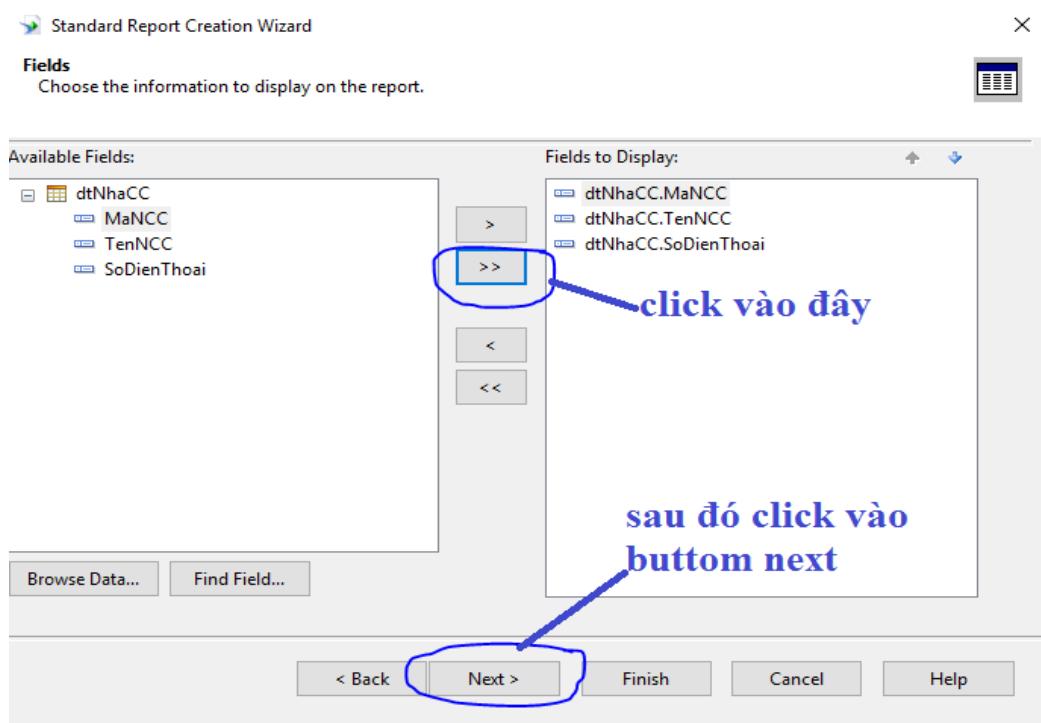
Xuất hiện một cửa sổ click chọn “Ok”



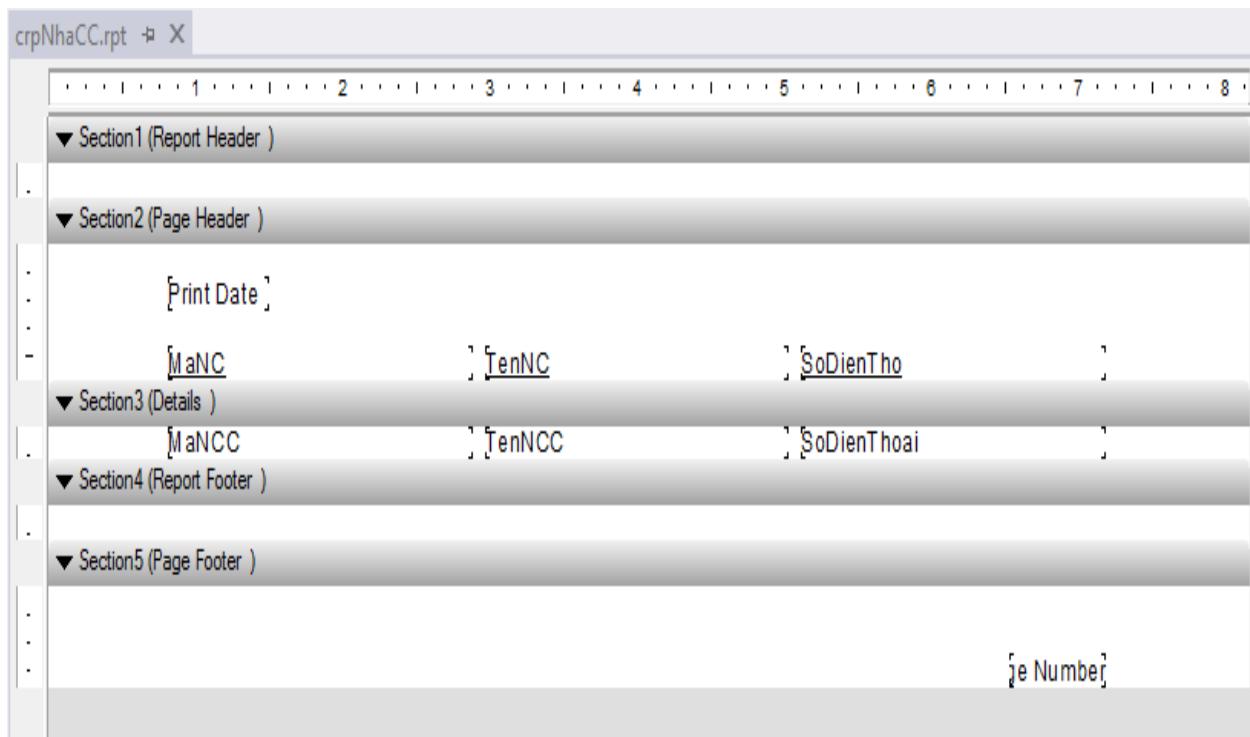
Cửa sổ mới xuất hiện:



Xuất hiện cửa sổ tiếp theo:



Cửa sổ tiếp theo xuất hiện chọn **Finish**, sau khi tạo thành công sẽ xuất hiện:



Bước 10: Viết code cho frmBaoCao:

- Khai báo NhaCC_BLL, crpNhaCC và viết code cho sự kiện load form để dữ liệu và cbNhaCC

```
NhaCC_BLL nccbll = new NhaCC_BLL();
crpNhaCC crp = new crpNhaCC();
1 reference
private void frmBaoCao_Load(object sender, EventArgs e)
{
    cbMaNCC.DataSource = nccbll.NhaCC_SelectAll();
    cbMaNCC.ValueMember = "MaNCC";
    cbMaNCC.DisplayMember = "TenNCC";
}
```

- Viết code cho nút lệnh btInKhongThamSo

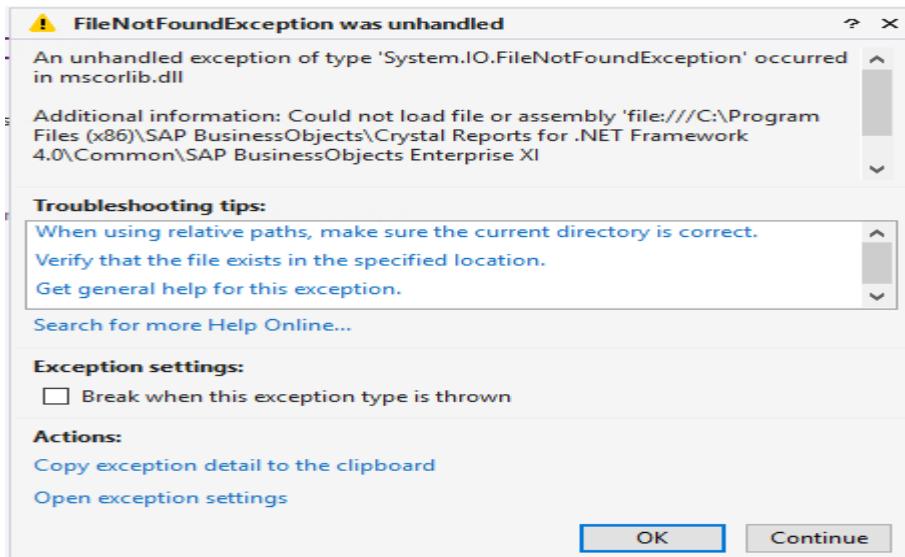
```
private void btInKhongThamSo_Click(object sender, EventArgs e)
{
    // Gán data source cho crpNhaCC
    crp.SetDataSource(nccbll.NhaCC_SelectAll());
    // gán reporrt cho Report (xem bước 7)
    ReReport.ReportSource = crp;
}
```

- Viết code cho nút lệnh btInCoThamSo

```
private void btInCoThamSo_Click(object sender, EventArgs e)
{
    // Gán data source cho crpNhaCC với tham số truyền vào là mã của nhà cung
    // được chọn từ cbNhaCC (xem code sự kiện load form)
    crp.SetDataSource(nccbll.NhaCC_Select_MaNCC(cbMaNCC.SelectedValue.ToString()));
    // gán reporrt cho Report (xem bước 7)
    ReReport.ReportSource = crp;
}
```

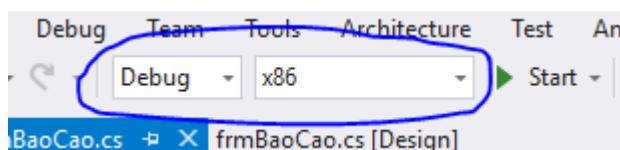
- Chạy chương trình và xem kết quả

Bước 11: Trong trường hợp click vào các button xem báo cáo mà xuất hiện thông báo:



Thì xử lý như sau (2 bước):

- Chọn chế độ chạy x86:

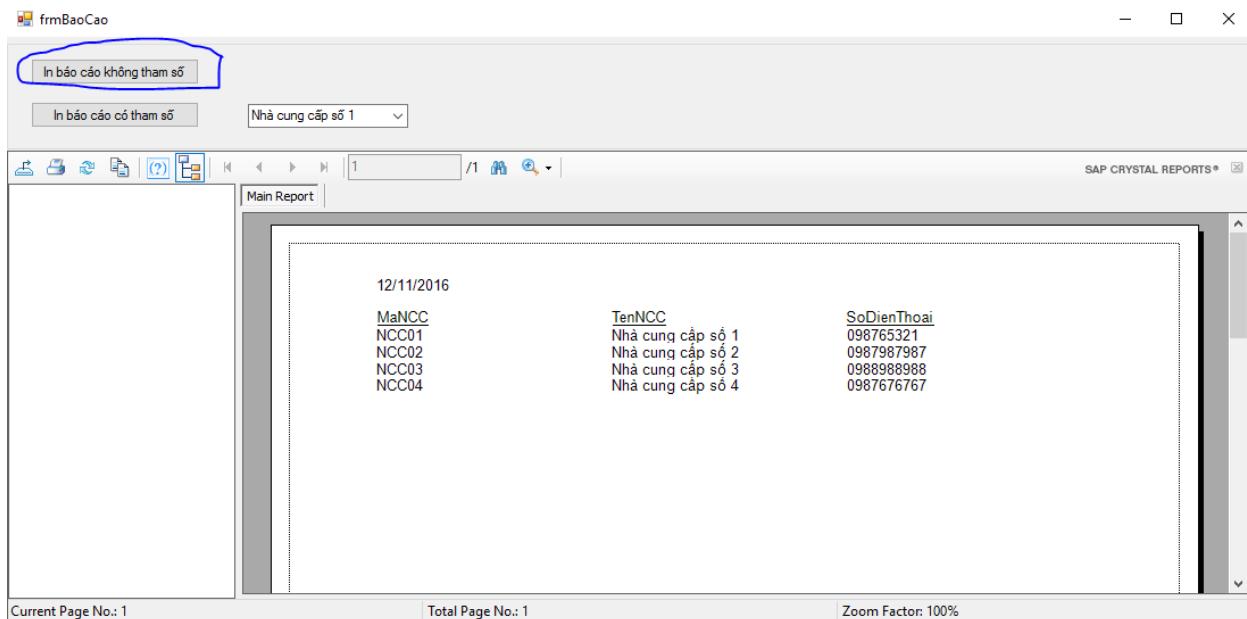


- Thêm vào thẻ startup trong file App.Config:

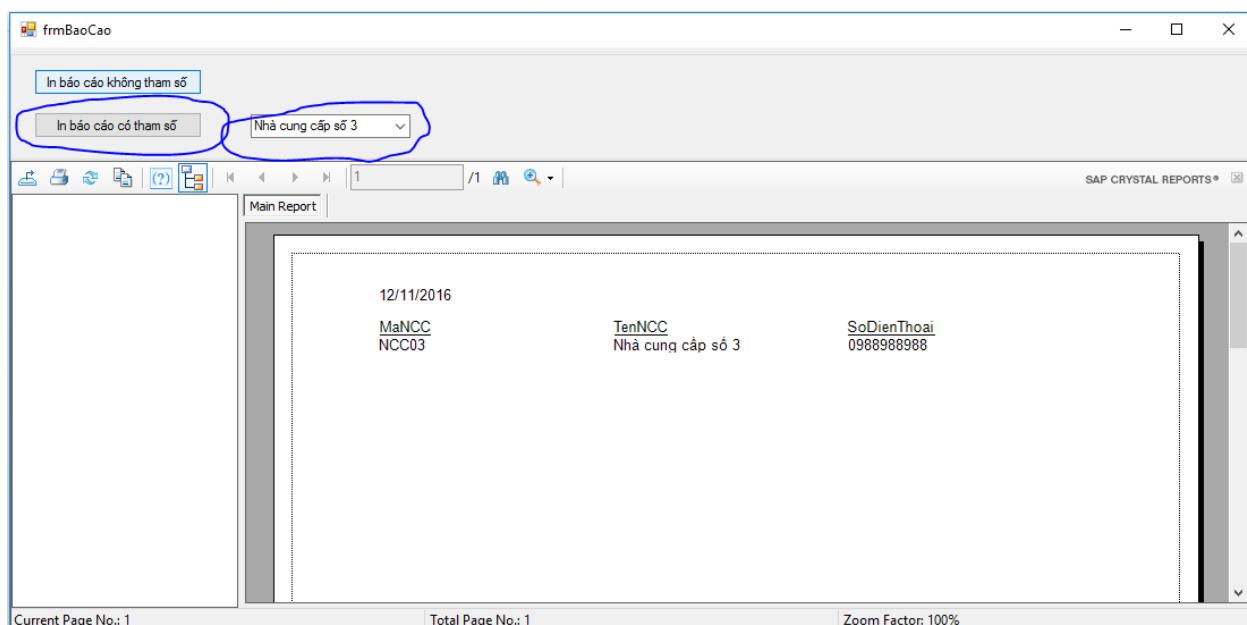
```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <configSections>
  </configSections>
  <startup useLegacyV2RuntimeActivationPolicy="true">
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.6"/>
  </startup>
  <connectionStrings>
    <add name="project" connectionString="Data Source=.;Initial Catalog=QuanLyBanH
      <add name="LapTrinhQuanLy.Properties.Settings.QuanLyBanHangConnectionString" c
    </connectionStrings>
  </configuration>
```

Bước 12: Chạy chương trình kiểm tra kết quả

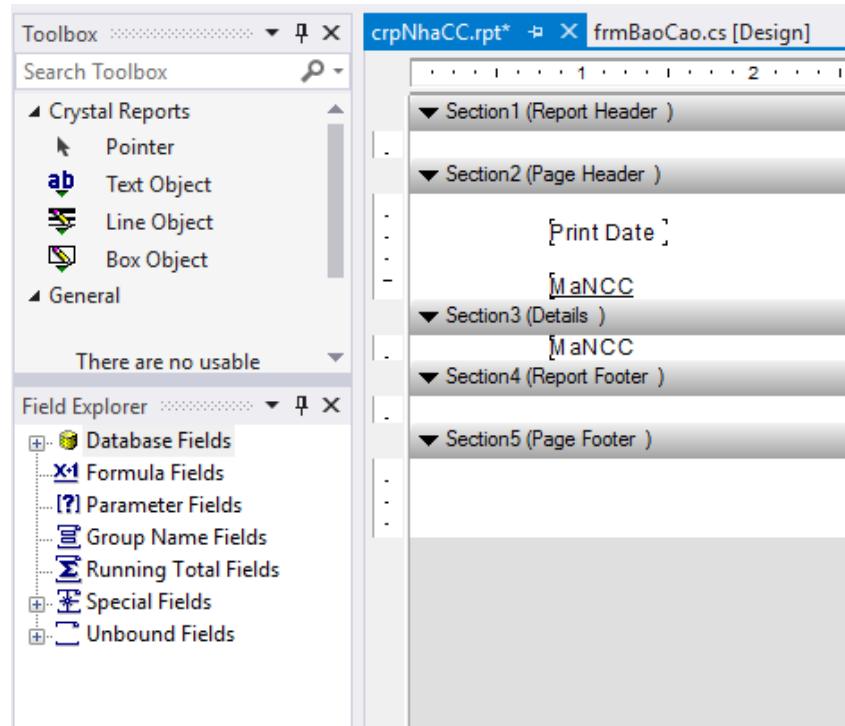
- In báo cáo không chứa tham số



- In báo cáo chứa tham số



Bước 13: Chú ý: cần sử dụng toolbox để thiết kế giao diện của báo cáo đẹp hơn



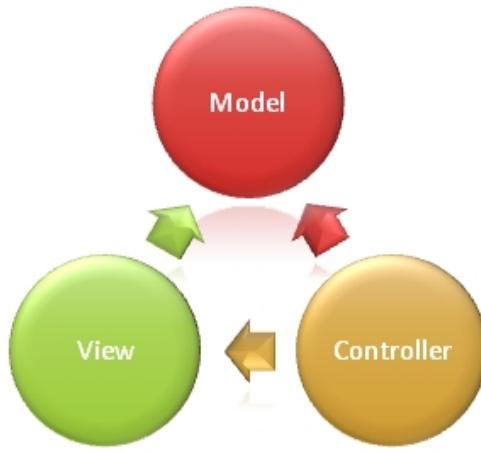
5.3.3. Viết mã cho chương trình trên nền tảng ASP.NET MVC

1. Tổng quan về ASP.MVC

Mẫu kiến trúc Model – View – Controller được sử dụng nhằm chỉ ứng dụng thành ba thành phần chính: model, view và controller. Nền tảng ASP.NET MVC giúp cho chúng ta có thể tạo được các ứng dụng web áp dụng mô hình MVC thay vì tạo ứng dụng theo mẫu ASP.NET Web Forms. Nền tảng ASP.NET MVC có đặc điểm nổi bật là nhẹ (lightweight), dễ kiểm thử phần giao diện (so với ứng dụng Web Forms), tích hợp các tính năng sẵn có sẵn của ASP.NET. Nền tảng ASP.NET MVC được định nghĩa trong namespace System.Web.Mvc và là một phần của namespace System.Web.

MVC là một mẫu thiết kế (design pattern) chuẩn mà nhiều lập trình viên đã quen thuộc. Một số loại ứng dụng web sẽ thích hợp với kiến trúc MVC. Một số khác vẫn thích hợp với ASP.NET Web Forms và cơ chế postbacks. Đôi khi có những ứng dụng kết hợp cả hai kiến trúc trên.

Nền tảng MVC bao gồm các thành phần dưới đây:



Hình 01: Mô hình Model – View – Controller

Models: Các đối tượng Models là một phần của ứng dụng, các đối tượng này thiết lập logic của phần dữ liệu của ứng dụng. Thông thường, các đối tượng model lấy và lưu trạng thái của model trong CSDL. Ví dụ như, một đối tượng Product (sản phẩm) sẽ lấy dữ liệu từ CSDL, thao tác trên dữ liệu và sẽ cập nhật dữ liệu trở lại vào bảng Products ở SQL Server.

Trong các ứng dụng nhỏ, model thường là chỉ là một khái niệm nhằm phân biệt hơn là được cài đặt thực thụ, ví dụ, nếu ứng dụng chỉ đọc dữ liệu từ CSDL và gởi chúng đến view, ứng dụng không cần phải có tầng model và các lớp liên quan. Trong trường hợp này, dữ liệu được lấy như là một đối tượng model (hơn là tầng model).

Views: Views là các thành phần dùng để hiển thị giao diện người dùng (UI). Thông thường, view được tạo dựa vào thông tin dữ liệu model. Ví dụ như, view dùng để cập nhật bảng Products sẽ hiển thị các hộp văn bản, drop-down list, và các check box dựa trên trạng thái hiện tại của một đối tượng Product.

Controllers: Controller là các thành phần dùng để quản lý tương tác người dùng, làm việc với model và chọn view để hiển thị giao diện người dùng. Trong một ứng dụng MVC, view chỉ được dùng để hiển thị thông tin, controller chịu trách nhiệm quản lý và đáp trả nội dung người dùng nhập và tương tác với người dùng. Ví dụ, controller sẽ quản lý các dữ liệu người dùng gửi lên (query-string values) và gửi các giá trị đó đến model, model sẽ lấy dữ liệu từ CSDL nhờ vào các giá trị này.

Mẫu MVC giúp bạn tạo được các ứng dụng mà chúng phân tách rạch rời các khía cạnh của ứng dụng (logic về nhập liệu, logic xử lý tác vụ và logic về giao diện). Mẫu MVC chỉ ra mỗi loại logic kể trên nên được thiếp lập ở đâu trên ứng dụng. Logic giao diện (UI logic) thuộc về views. Logic nhập liệu (input logic) thuộc về controller. Và logic tác vụ (Business logic – là logic xử lý thông tin, mục đích chính của ứng dụng) thuộc về model. Sự phân chia này giúp bạn giảm bớt được sự phức tạp của ứng dụng và chỉ tập trung vào mỗi khía cạnh cần được cài đặt ở mỗi thời điểm. Ví dụ như bạn chỉ cần tập trung vào giao diện (views) mà không phải quan tâm đến logic xử lý thông tin của ứng dụng.

Để quản lý sự phức tạp của ứng dụng, mẫu MVC giúp cho chúng ta có thể kiểm thử ứng dụng dễ dàng hơn hẳn so với khi áp dụng mẫu Web Forms. Ví dụ, trong một ứng dụng ASP.NET Web Forms, một lớp thường được sử dụng để hiển thị thông tin xuất ra cho người dùng và đồng thời xử lý thông tin người dùng nhập. Việc xây dựng các bộ test tự động cho ứng dụng Web Forms là rất phức tạp, bởi để kiểm thử mỗi trang web, bạn phải khởi tạo đối tượng trang, khởi tạo tất cả các control được sử dụng trong trang và các lớp phụ thuộc trong ứng dụng. Và bởi vì có quá nhiều lớp cần được khởi tạo để chạy được trang, thật khó để có thể viết các test chỉ tập trung vào một khía cạnh nào đó của ứng dụng. Và vì thế, kiểm thử đối với các ứng dụng dựa trên nền tảng Web Forms sẽ khó khăn hơn nhiều so với khi áp dụng trên ứng dụng MVC. Hơn thế nữa, việc kiểm thử trên nền tảng Web Forms yêu cầu phải sử dụng đến web server.

Nền tảng MVC phân tách các thành phần và sử dụng các interface (khái niệm giao diện trong lập trình hướng đối tượng), và nhờ đó có thể kiểm thử các thành phần riêng biệt trong tình trạng phân lập với các yếu tố còn lại của ứng dụng.

Sự phân tách rạch rời ba thành phần của ứng dụng MVC còn giúp cho việc lập trình diễn ra song song. Ví dụ như một lập trình viên làm việc với view, lập trình viên thứ hai lo cài đặt logic của controller và lập trình viên thứ ba có thể tập trung vào logic tác vụ của model tại cùng một thời điểm.

Lựa chọn áp dụng MVC trong xây dựng ứng dụng

Bạn cần phải xem xét kỹ càng việc áp dụng mô hình ASP.NET MVC hay mô hình ASP.NET Web Forms khi xây dựng một ứng dụng. Mô hình MVC không phải là mô hình thay thế cho Web Forms, bạn có thể dùng một trong hai mô hình.

Trước khi quyết định sử dụng MVC hay Web Forms cho một web site cụ thể, bạn cần phải phân tích lợi ích khi chọn một trong hai hướng.

Lợi ích của ứng dụng web dựa trên mô hình MVC

Nền tảng ASP.NET MVC mang lại những lợi ích sau:

- Dễ dàng quản lý sự phức tạp của ứng dụng bằng cách chia ứng dụng thành ba thành phần model, view, controller
- Nó không sử dụng view state hoặc server-based form. Điều này tốt cho những lập trình viên muốn quản lý hết các khía cạnh của một ứng dụng.
- Nó sử dụng mẫu Front Controller, mẫu này giúp quản lý các requests (yêu cầu) chỉ thông qua một Controller. Nhờ đó bạn có thể thiết kế một hạ tầng quản lý định tuyến. Để có nhiều thông tin hơn, bạn nên xem phần Front Controller trên web site MSDN
- Hỗ trợ tốt hơn cho mô hình phát triển ứng dụng hướng kiểm thử (TDD)

- Nó hỗ trợ tốt cho các ứng dụng được xây dựng bởi những đội có nhiều lập trình viên và thiết kế mà vẫn quản lý được tính năng của ứng dụng.

Lợi ích của ứng dụng được xây dựng trên nền tảng Web Forms

- Nó hỗ trợ cách lập trình hướng sự kiện, quản lý trạng thái trên giao thức HTTP, tiện dụng cho việc phát triển các ứng dụng Web phục vụ kinh doanh. Các ứng dụng trên nền tảng Web Forms cung cấp hàng tá các sự kiện được hỗ trợ bởi hàng trăm các server controls.
- Sử dụng mẫu Page Controller. Xem thêm ở mục Page Controller trên MSDN
- Mô hình này sử dụng view state hoặc server-based form, nhờ đó sẽ giúp cho việc quản lý trạng thái các trang web dễ dàng.
- Nó rất phù hợp với các nhóm lập trình viên quy mô nhỏ và các thiết kế, những người muốn tận dụng các thành phần giúp xây dựng ứng dụng một cách nhanh chóng.
- Nói tóm lại, áp dụng Web Forms giúp giảm bớt sự phức tạp trong xây dựng ứng dụng, bởi vì các thành phần (lớp Page, controls,...) được tích hợp chắc chắn và thường thì giúp bạn viết ít code hơn là áp dụng theo mô hình MVC.

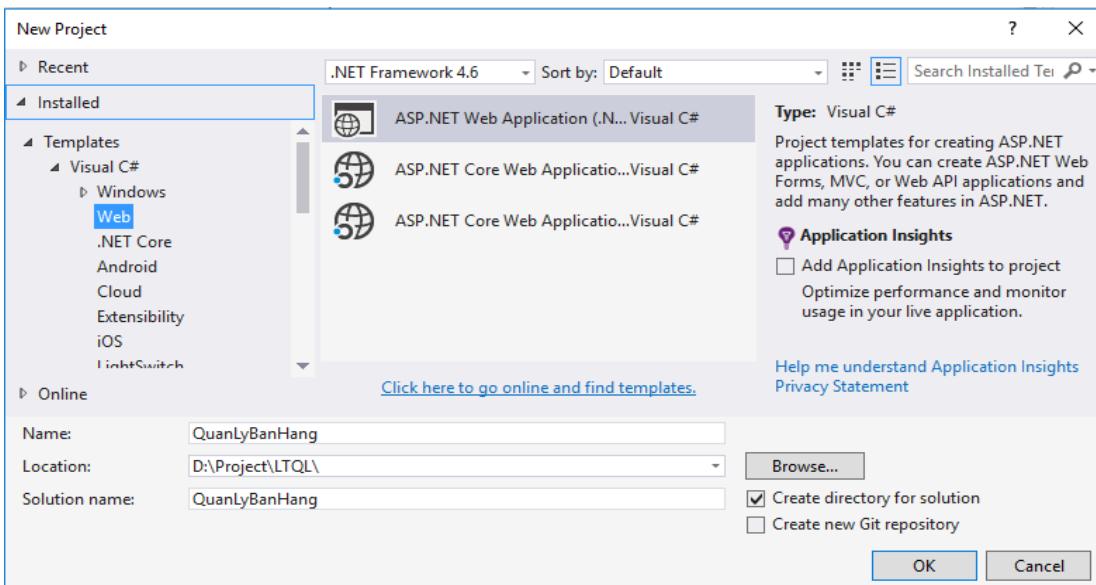
Các tính năng của nền tảng ASP.NET MVC

- Tách bạch các tác vụ của ứng dụng (logic nhập liệu, business logic, và logic giao diện), dễ dàng kiểm thử và mặc định áp dụng hướng phát triển TDD. Tất cả các tính năng chính của mô hình MVC được cài đặt dựa trên interface và được kiểm thử bằng cách sử dụng các đối tượng mocks, mock object là các đối tượng mô phỏng các tính năng của những đối tượng thực sự trong ứng dụng. Bạn có thể kiểm thử unit-test cho ứng dụng mà không cần chạy controller trong tiến trình ASP.NET, và điều đó giúp unit test được áp dụng nhanh chóng và tiện dụng. Bạn có thể sử dụng bất kỳ nền tảng unit-testing nào tương thích với nền tảng .NET.

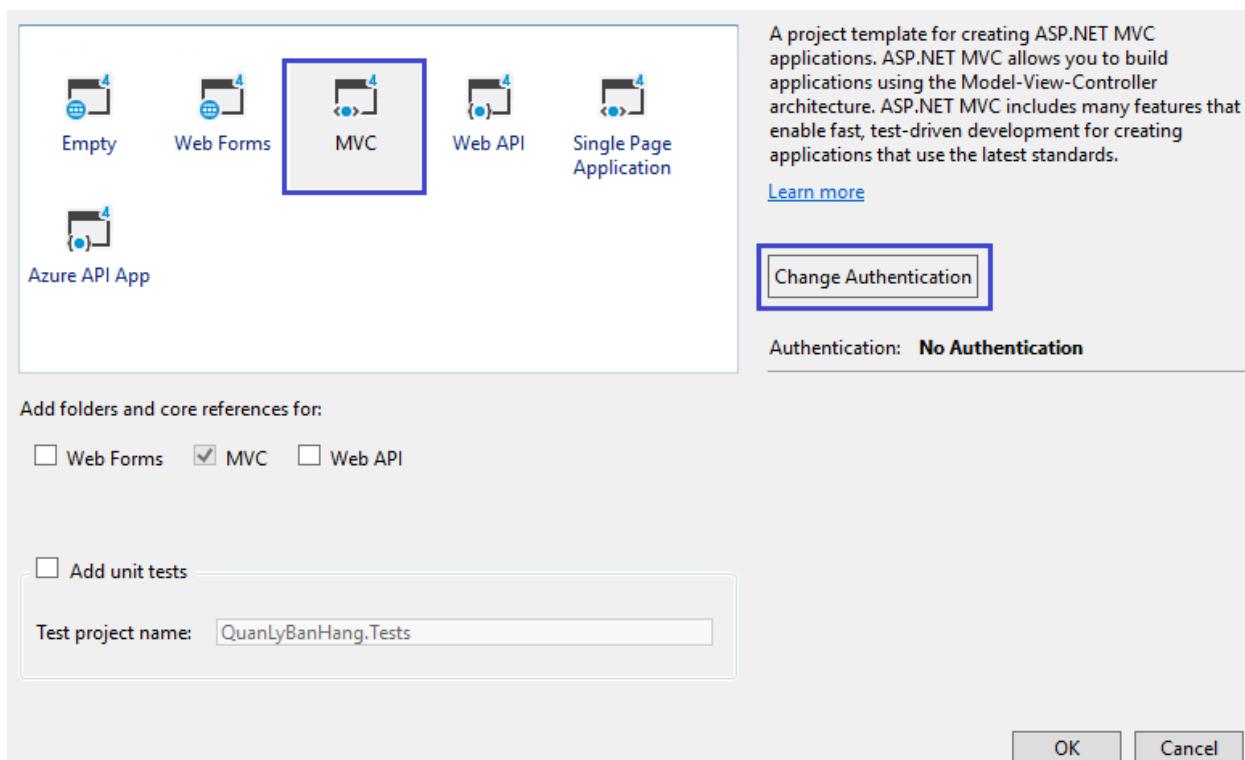
- MVC là một nền tảng khả mở rộng (extensible) & khả nhúng (pluggable). Các thành phần của ASP.NET MVC được thiết kế để chúng có thể được thay thế một cách dễ dàng hoặc dễ dàng tùy chỉnh. Bạn có thể nhúng thêm view engine, cơ chế định tuyến cho URL, cách kết xuất tham số của action-method và các thành phần khác. ASP.NET MVC cũng hỗ trợ việc sử dụng Dependency Injection (DI) và Inversion of Control (IoC). DI cho phép bạn gắn các đối tượng vào một lớp cho lớp đó sử dụng thay vì buộc lớp đó phải tự mình khởi tạo các đối tượng. IoC quy định rằng, nếu một đối tượng yêu cầu một đối tượng khác, đối tượng đầu sẽ lấy đối tượng thứ hai từ một nguồn bên ngoài, ví dụ như từ tập tin cấu hình. Và nhờ vậy, việc sử dụng DI và IoC sẽ giúp kiểm thử dễ dàng hơn.
- ASP.NET MVC có thành phần ánh xạ URL mạnh mẽ cho phép bạn xây dựng những ứng dụng có các địa chỉ URL xúc tích và dễ tìm kiếm. Các địa chỉ URL không cần phải có phần mở rộng của tên tập tin và được thiết kế để hỗ trợ các mẫu định dạng tên phù hợp với việc tối ưu hóa tìm kiếm (URL) và phù hợp với lập địa chỉ theo kiểu REST.
- Hỗ trợ sử dụng đặc tả (các thẻ) của các trang ASP.NET(.aspx), điều khiển người dùng (.ascx) và trang master page (.master). Bạn có thể sử dụng các tính năng có sẵn của ASP.NET như là sử dụng lồng các trang master page, sử dụng in-line expression (<%= %>), sử dụng server controls, mẫu, data-binding, địa phương hóa (localization) và hơn thế nữa.
- Hỗ trợ các tính năng có sẵn của ASP.NET như cơ chế xác thực người dùng, quản lý thành viên, quyền, output caching và data caching, session và profile, quản lý tình trạng ứng dụng, hệ thống cấu hình...
- Bắt đầu từ ASP.NET MVC 3 còn bổ sung một view engine mới là Razor View Engine cho phép thiết lập các view nhanh chóng, dễ dàng và tốn ít công sức hơn so với việc sử dụng Web Forms view engine.

2. Xây dựng chương trình

- Khởi động visual studio, chọn tạo mới một project (ASP.NET Web Application), đặt tên cho project là QuanLyBanHang:

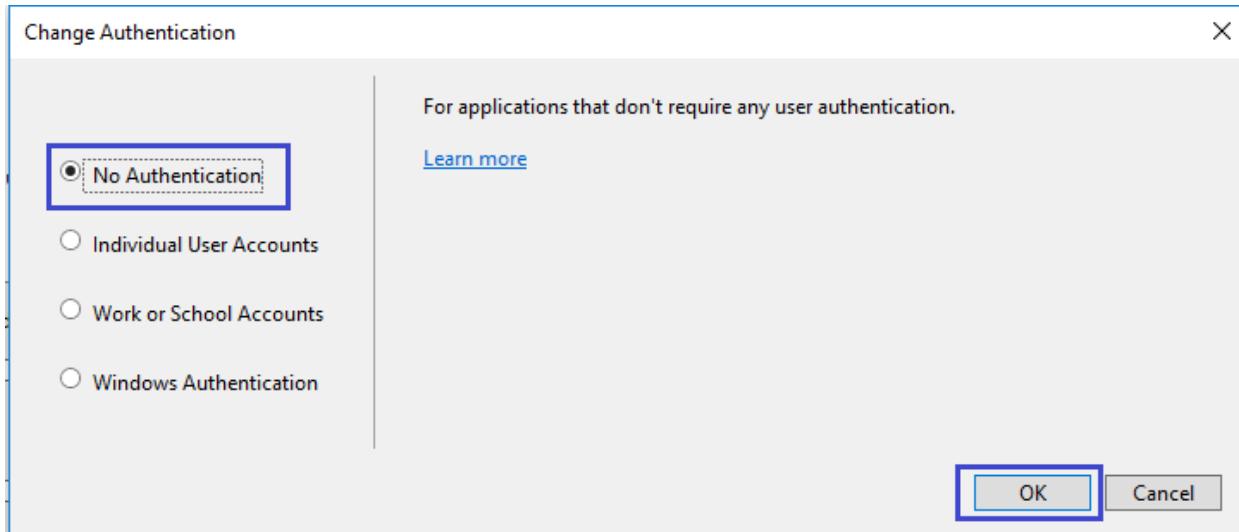


- Cửa sổ mới hiện ra chọn MVC, click vào “Change Authentication”

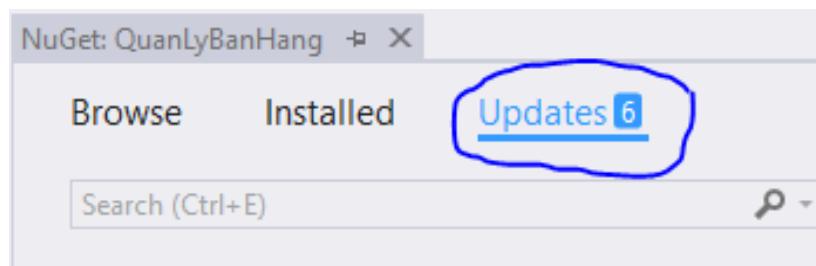




Cửa sổ mới hiện ra chọn “No Authentication” → OK → OK để tạo project.



Sau khi tạo xong project, Click chuột phải vào project → Manage Nuget Packages nếu trong phần “Update” có thông báo cần update các thư viện thì nên update lên phiên bản mới nhất (cần có kết nối internet ở bước này):

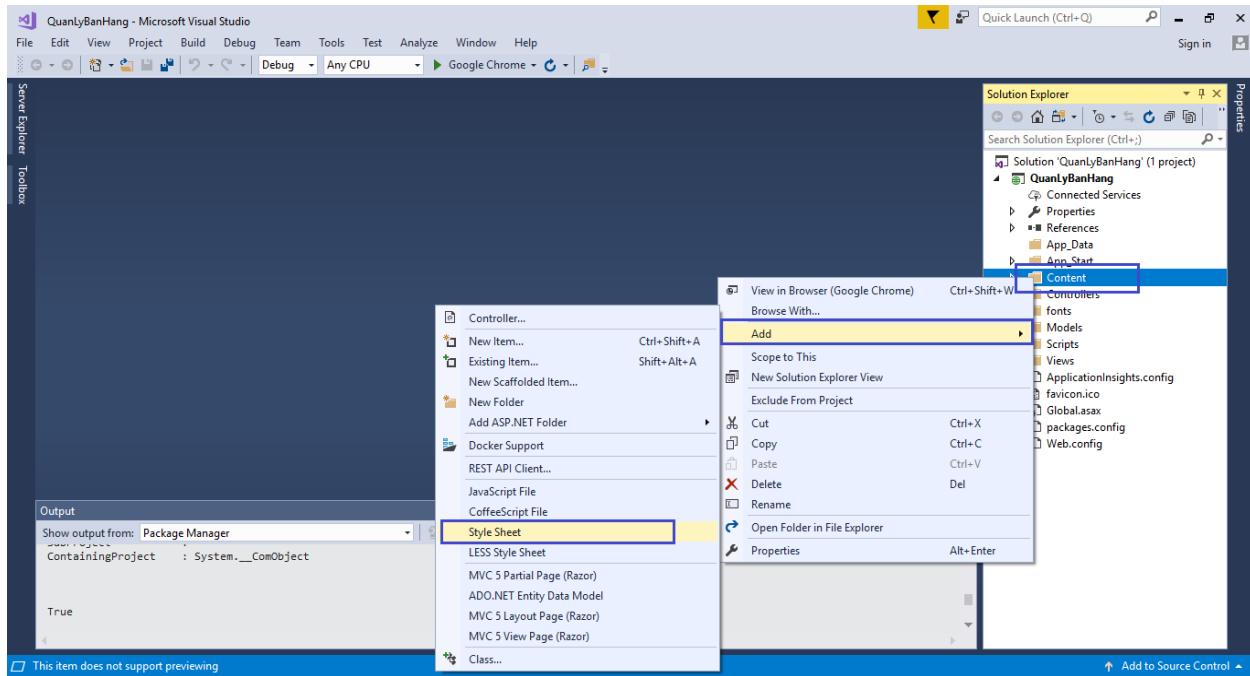


Sau khi update xong, Visual Studio sẽ đưa ra thông báo yêu cầu restart lại project, chọn Restart → ok để hoàn tất việc update.

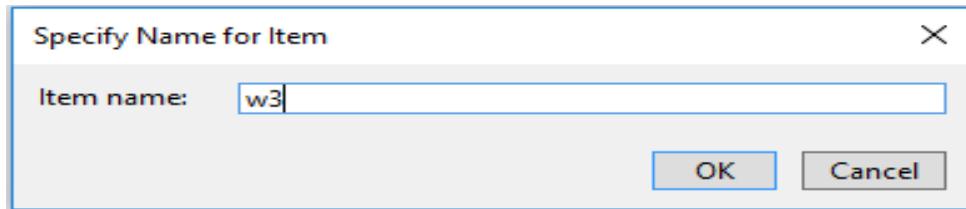


Sử dụng thư viện w3.css để thiết kế giao diện (nếu sử dụng bootstrap thì bỏ qua bước này)

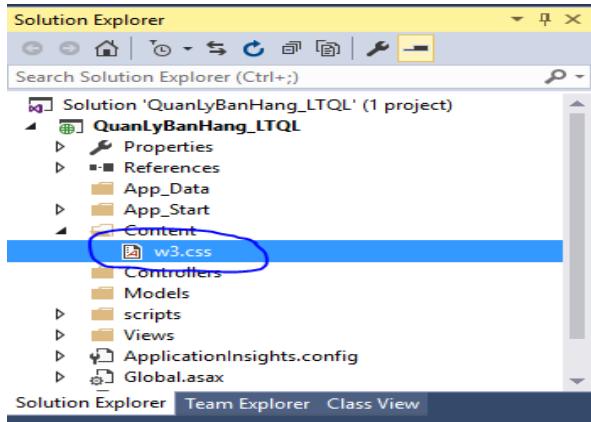
- Truy cập vào địa chỉ sau để download mã nguồn của w3.css
https://www.w3schools.com/w3css/w3css_downloads.asp.
- Click chuột phải vào thư mục Content → Add → Style Sheet:



- Một cửa sổ mới xuất hiện, nhập w3 rồi click “Ok”, sau đó copy mã nguồn vừa download ở bước trên vào file (bước tạo file w3.css để thiết kế giao diện chương trình)



- Sau khi tạo thành công sẽ có file w3.css trong thư mục Content



Cấu hình sử dụng BundleConfig:

- Để cấu hình sử dụng file “W3.css” vừa tạo trong thư mục “Content”, chúng ta viết bổ sung mã nguồn cho file BundleConfig.cs như sau (phân đóng khung màu xanh):

```

public static void RegisterBundles(BundleCollection bundles)
{
    bundles.Add(new ScriptBundle("~/bundles/jquery").Include(
        "~/Scripts/jquery-{version}.js"));

    bundles.Add(new ScriptBundle("~/bundles/jqueryval").Include(
        "~/Scripts/jquery.validate*"));

    bundles.Add(new ScriptBundle("~/bundles/modernizr").Include(
        "~/Scripts/modernizr-*"));

    bundles.Add(new ScriptBundle("~/bundles/bootstrap").Include(
        "~/Scripts/bootstrap.js",
        "~/Scripts/respond.js"));

    bundles.Add(new StyleBundle("~/Content/css").Include(
        "~/Content/bootstrap.css",
        "~/Content/site.css",
        "~/Content/W3.css"));
}

```

- Mở thư mục “View ➔ shared ➔ _layout.cshtml”, code gọi sử dụng (phân đóng khung màu xanh) các file CSS, JavaScript được cấu hình ở bước trên (chú ý phân biệt @Styles.Render và @Scripts.Render và đường dẫn truyền vào so với cấu hình trong file BundleConfig.cs)

```

<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>@ViewBag.Title - My ASP.NET Application</title>
    @Styles.Render("~/Content/css")
    @Scripts.Render("~/bundles/modernizr")
</head>
<body>
    <div class="navbar navbar-inverse navbar-fixed-top">...</div>
    <div class="container body-content">...</div>

    @Scripts.Render("~/bundles/jquery")
    @Scripts.Render("~/bundles/bootstrap")
    @RenderSection("scripts", required: false)
</body>
</html>

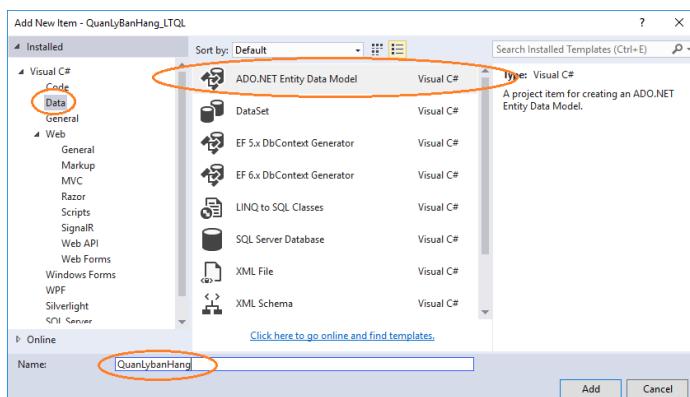
```

- File _layout.cshtml được sử dụng để chứa các thành phần dùng chung cho tất cả các trang sử dụng layout là trang _layout.cshtml (ví dụ: TopMenu, LeftMenu, RightMenu, Liên hệ ...)
- Chú ý: Trong layout dùng chung cho tất cả các trang, phần @RenderBody() là nội dung của các view sử dụng layout, phần còn lại sẽ là nội dung của layout được hiển thị giống nhau ở tất cả các view có sử dụng layout.

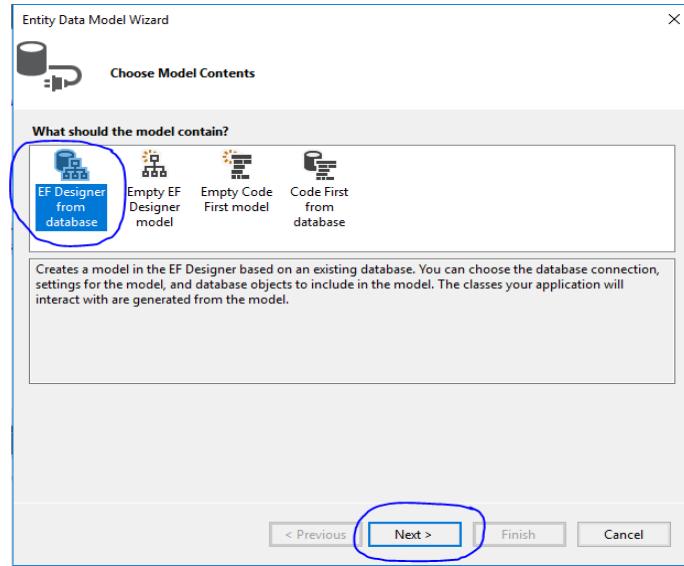


Sử dụng data entity framework trong xây dựng website:

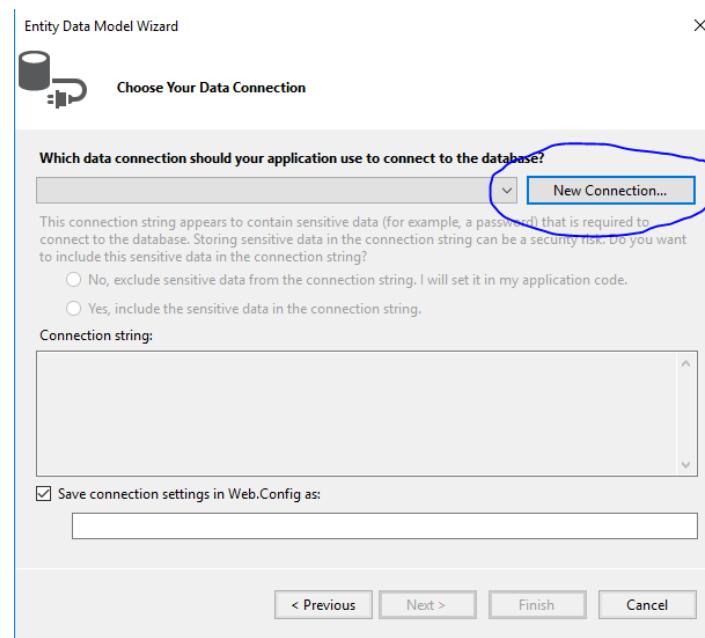
- Trong thư mục “**Models**”, Add đối tượng ADO.Net Entity Data Model đặt tên là QuanLybanHang:



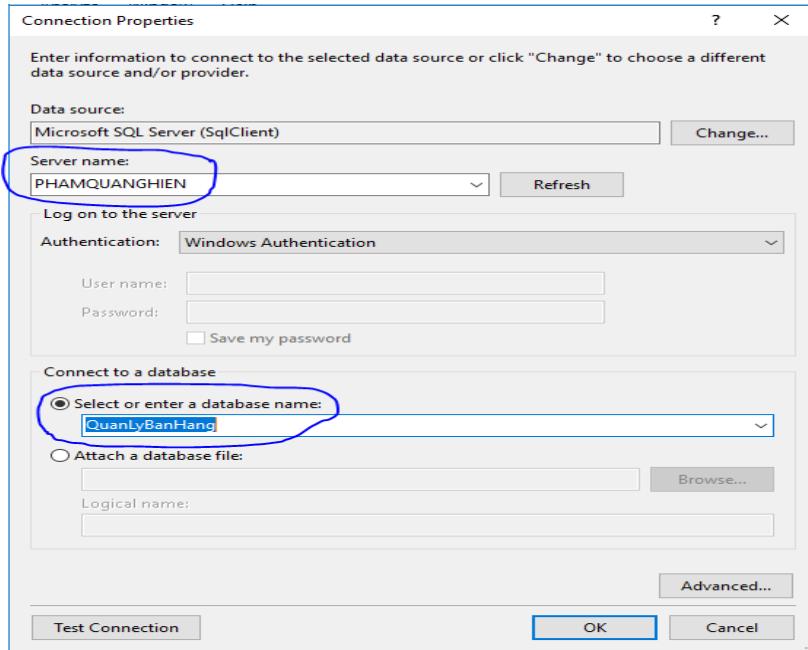
- Một cửa sổ mới xuất hiện, click EF Designer from database ➔ Next:



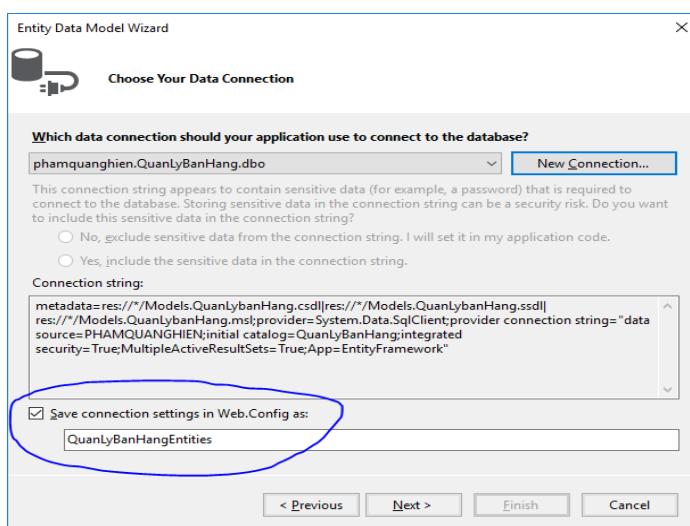
- Xuất hiện cửa sổ tiếp theo, chọn New Connection để xuất hiện cửa sổ thiết lập kết nối với database:



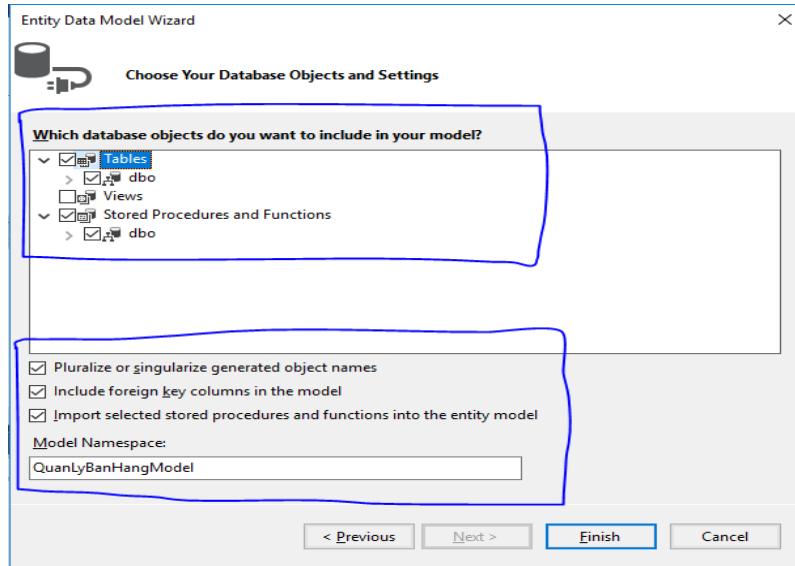
- Trong cửa sổ mới xuất hiện, điền đầy đủ thông tin Server Name và Database Name để thiết lập kết nối với database:



- Kết nối thành công với database sẽ xuất hiện cửa sổ, chú ý phần “Save connection setting in Web.Config as” đây chính là tên của đối tượng được lưu trữ trong file web.config hỗ trợ cho việc trao đổi dữ liệu với database:



- Tiếp theo, chọn các table, views, store procedure and function mà bạn muốn sử dụng khi làm việc với database và chọn finish (trước đó có thể có bước chọn phiên bản data entity framework để làm việc, lưu ý nên chọn phiên bản mới nhất):

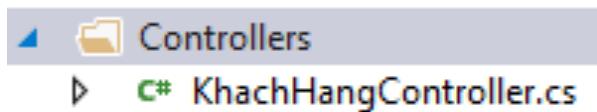


- Sau đó build lại project.



Tạo controller

Click chuột phải vào thư mục Controllers → Add → Controller → MVC5 Controller - Empty đặt tên là KhachHangController (lưu ý: phần hậu tố Controller là bắt buộc trong tên của các controller, ví dụ tạo một controller có nhiệm vụ xử lý thông tin hàng hóa thì đặt tên là HangHoaController), sau khi tạo thành công, trong thư mục Controllers sẽ xuất hiện file KhachHangController.cs, đồng thời sẽ xuất hiện thư mục KhachHang bên trong thư mục Views (để chứa tất cả các view liên quan đến KhachHangcontroller):



Trước hết, chúng ta sẽ khai báo để sử dụng data entity model trong bước trước (mặc định khi tạo controller chúng ta sẽ có sẵn phương thức Index):

```

using QuanLyBanHang_LTQL.Models;
using System.Web.Mvc;

namespace QuanLyBanHang_LTQL.Controllers
{
    public class KhachHangController : Controller
    {
        QuanLyBanHangEntities entity = new QuanLyBanHangEntities();
        // GET: KhachHang
        public ActionResult Index()
        {
            return View();
        }
    }
}

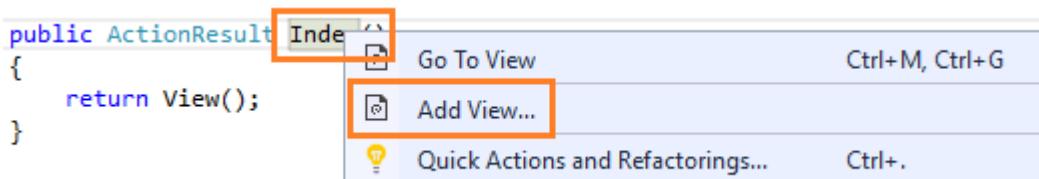
```

Sau đây chúng ta sẽ tiến hành viết code cho các phương thức Index (lấy dữ liệu từ database và hiển thị cho người dùng), Create (thêm dữ liệu vào database), Edit (sửa thông tin trong database), Details (xem thông tin chi tiết một bản ghi trong database), Delete (xóa dữ liệu trong database).

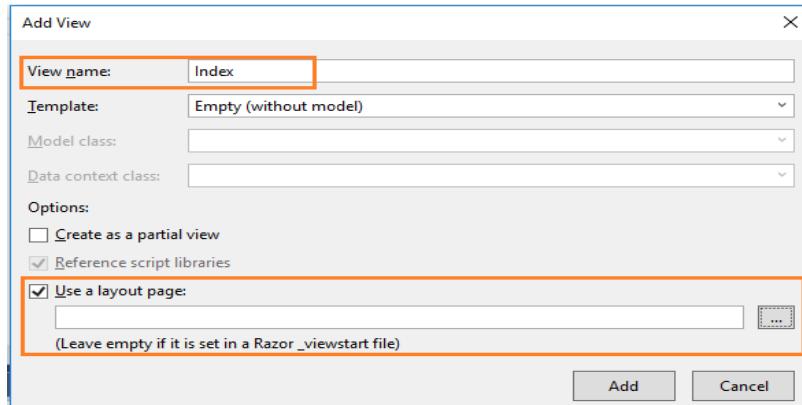


Phương thức Index (trong bước này chúng ta sẽ nhập trước dữ liệu vào trong database để demo):

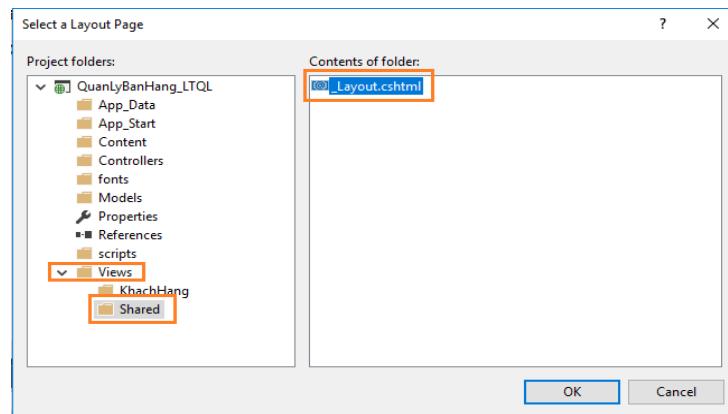
- Trước tiên chúng ta sẽ tạo view để hiển thị thông tin khách hàng bằng cách click chuột phải vào Index → Add View:



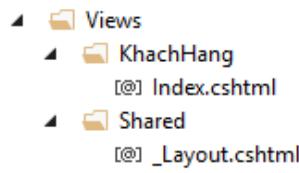
- Trong cửa sổ mới xuất hiện, click vào nút lệnh “...” trong phần use layout page để chọn layout cho trang index:



- Cửa sổ mới xuất hiện, chúng ta chọn file _layout.cshtml trong thư mục Shared đã tạo ở bước 6 sau đó click Ok ➔ Add để tạo View Index:



- Tạo mới thành công sẽ xuất hiện file Index.cshtml bên trong thư mục KhachHang:



- Tiếp tục viết code cho file KhachHangController như sau:

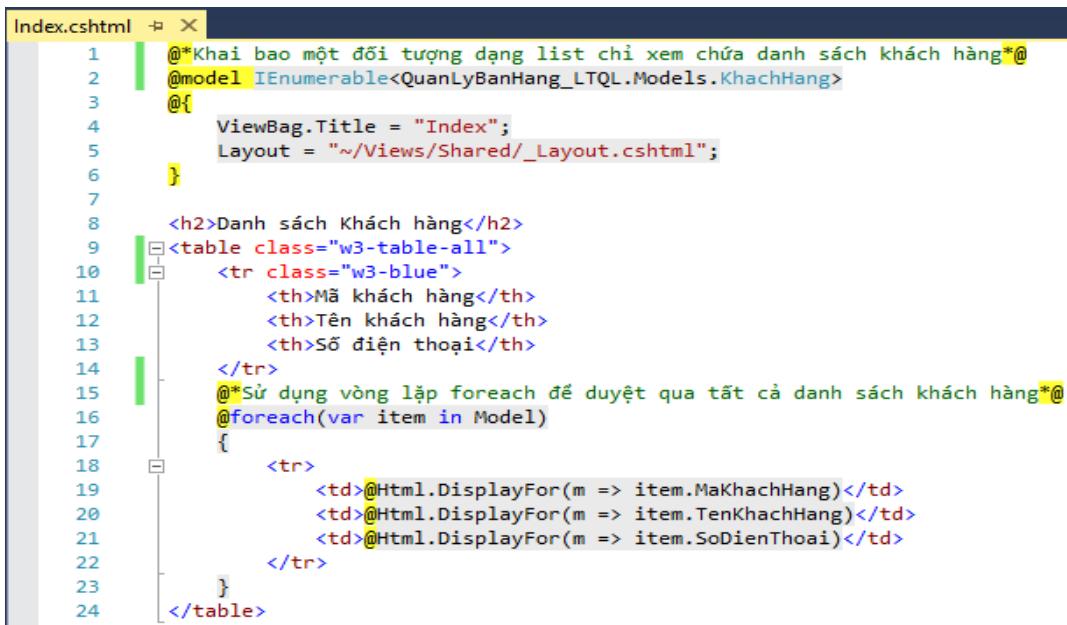
```

using QuanLyBanHang_LTQL.Models;
using System.Web.Mvc;
using System.Linq;

namespace QuanLyBanHang_LTQL.Controllers
{
    public class KhachHangController : Controller
    {
        private QuanLyBanHangEntities entity = new QuanLyBanHangEntities();
        // GET: KhachHang
        public ActionResult Index()
        {
            //Khai báo một biến để chứa dữ liệu lấy từ table khách hàng
            var list_KH = entity.KhachHangs.ToList();
            //trả về view biến vừa khai báo ở trên
            return View(list_KH);
        }
    }
}

```

- Sau đó, chúng ta đi xây dựng view để hiển thị danh sách khách hàng:



```

Index.cshtml
1  /*Khai bao một đối tượng dạng list chỉ xem chứa danh sách khách hàng*/
2  @model IEnumerable<QuanLyBanHang_LTQL.Models.KhachHang>
3  @{
4      ViewBag.Title = "Index";
5      Layout = "~/Views/Shared/_Layout.cshtml";
6  }
7
8  <h2>Danh sách Khách hàng</h2>
9  <table class="w3-table-all">
10     <tr class="w3-blue">
11         <th>Mã khách hàng</th>
12         <th>Tên khách hàng</th>
13         <th>Số điện thoại</th>
14     </tr>
15     /*Sử dụng vòng lặp foreach để duyệt qua tất cả danh sách khách hàng*/
16     @foreach(var item in Model)
17     {
18         <tr>
19             <td>@Html.DisplayFor(m => item.MaKhachHang)</td>
20             <td>@Html.DisplayFor(m => item.TenKhachHang)</td>
21             <td>@Html.DisplayFor(m => item.SoDienThoai)</td>
22         </tr>
23     }
24 </table>

```

Sau đó, build lại project, chạy ứng dụng và truy cập vào trình duyệt theo đường dẫn localhost:port/KhachHang/Index (chú ý: port là số cổng mà ứng dụng chạy, có thể khác nhau ở mỗi máy tính khác nhau, ở đây là cổng 81; “KhachHang” là tên controller, “Index” là tên của action). Kết quả hiển thị như sau:

Mã khách hàng	Tên khách hàng	Số điện thoại
KH01	Steven Gerrard	0123456789
KH02	Philippe Coutinho	0987654321
KH03	Jamie Carragher	0999999999

Chú ý: Dữ liệu hiển thị có thể khác nhau ở mỗi máy tùy theo dữ liệu nhập vào Database trên mỗi máy.

Tiếp theo bổ sung code cho file Index.html như dưới đây để tạo liên kết với các phương thức Create Edit, Details, Delete

```

8      <h2>Danh sách Khách hàng</h2>
9      <table class="w3-table-all">
10     <tr class="w3-blue">
11       <th>Mã khách hàng</th>
12       <th>Tên khách hàng</th>
13       <th>Số điện thoại</th>
14       <th></th>
15     </tr>
16     /* Sử dụng vòng lặp foreach để duyệt qua tất cả danh sách khách hàng*/
17     @foreach(var item in Model)
18     {
19       <tr>
20         <td>@Html.DisplayFor(m => item.MaKhachHang)</td>
21         <td>@Html.DisplayFor(m => item.TenKhachHang)</td>
22         <td>@Html.DisplayFor(m => item.SoDienThoai)</td>
23         <td>
24           @Html.ActionLink("Edit", "Edit", new { id = item.MaKhachHang }) |
25           @Html.ActionLink("Details", "Details", new { id = item.MaKhachHang }) |
26           @Html.ActionLink("Delete", "Delete", new { id = item.MaKhachHang })
27         </td>
28       </tr>
29     }
30   </table>
31
32   <p>
33     @Html.ActionLink("Create New", "Create")
34   </p>

```

Truy cập vào đường dẫn <http://localhost:81/KhachHang/Index> chúng ta sẽ thấy kết quả như sau:



Danh sách Khách hàng

Mã khách hàng	Tên khách hàng	Số điện thoại	
KH01	Steven Gerrard	0123456789	Edit Details Delete
KH02	Philippe Coutinho	0987654321	Edit Details Delete
KH03	Jamie Carragher	0999999999	Edit Details Delete

[Create New](#)



Phương thức create:

Đối với phương thức create chúng ta tiến hành viết code như sau:

```
// GET: KhachHangs/Create
public ActionResult Create()
{
    return View();
}

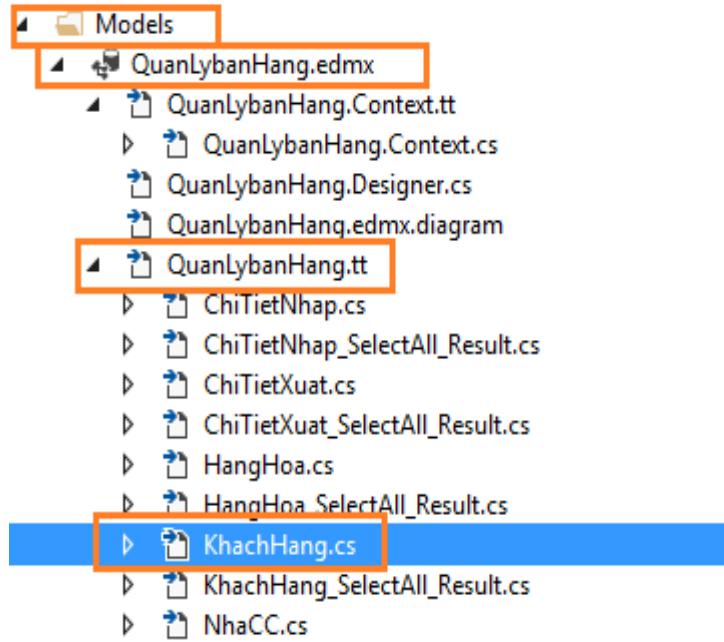
// POST: KhachHangs/Create
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult Create([Bind(Include = "MaKhachHang,TenKhachHang,SoDienThoai")] KhachHang khachHang)
{
    if (ModelState.IsValid)
    {
        entity.KhachHangs.Add(khachHang);
        entity.SaveChanges();
        return RedirectToAction("Index");
    }

    return View(khachHang);
}
```

Sau đó chúng ta đi tạo view create (tương tự như tạo view index ở bước trên) và code như sau:

```
1 @model QuanLyBanHang_LTQL.Models.KhachHang
2 @{
3     ViewBag.Title = "Create";
4     Layout = "~/Views/Shared/_Layout.cshtml";
5 }
6 @using (Html.BeginForm())
7 {
8     @Html.AntiForgeryToken()
9     <div class="w3-margin w3-card-4">
10         <div class="w3-blue w3-container"> <h3>Create KhachHang</h3> </div>
11         @*Thông báo lỗi sử dụng models validation*@
12         @Html.ValidationSummary()
13         @*Thông báo lỗi trong models validation Có thể sử dụng như sau
14         @Html.ValidationMessageFor(model => model.MaKhachHang, "", new { @class = "w3-text-blue" })*@
15         <div class="w3-container">
16             <form>
17                 <p>
18                     <label>Mã khách hàng</label>
19                     @Html.EditorFor(model => model.MaKhachHang, new { htmlAttributes = new { @class = "w3-input" } })
20                 </p>
21                 <p>
22                     <label>Tên Khách hàng</label>
23                     @Html.EditorFor(model => model.TenKhachHang, new { htmlAttributes = new { @class = "w3-input" } })
24                 </p>
25                 <p>
26                     <label>Số điện thoại</label>
27                     @Html.EditorFor(model => model.SoDienThoai, new { htmlAttributes = new { @class = "w3-input" } })
28
29                 </p>
30                 <p> <button class="w3-button w3-white w3-border w3-border-blue w3-round-large">Create</button> </p>
31             </form>
32         </div>
33     </div>
34 }
35 <div class="w3-margin">
36     @Html.ActionLink("Back to List", "Index")
37 </div>
38 @section Scripts { @Scripts.Render("~/bundles/jqueryval") }
```

Vào thư mục Models mở file KhachHang.cs và viết code như sau:



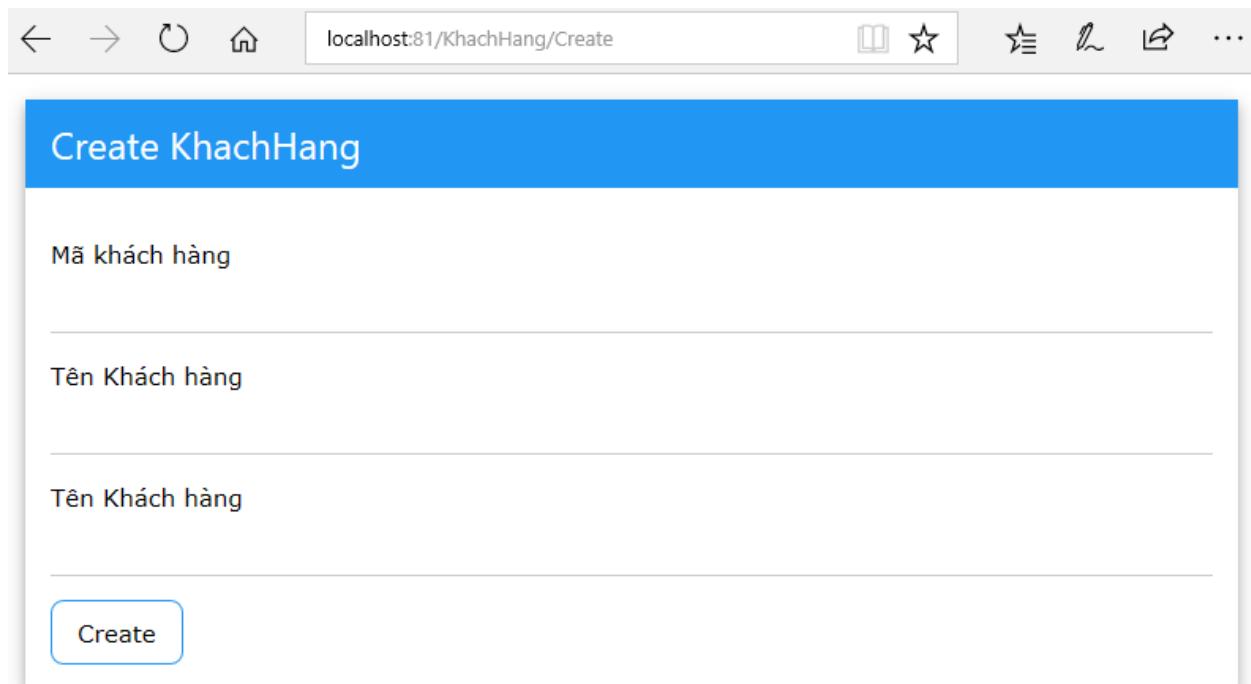
```
namespace QuanLyBanHang_LTQL.Models
{
    using System.Collections.Generic;
    using System.ComponentModel.DataAnnotations;

    public partial class KhachHang
    {
        [System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2214:DoNotCallOverridableMethodsInConstructors")]
        public KhachHang()
        {
            this.PhieuXuats = new HashSet<PhieuXuat>();
        }

        [Required(ErrorMessage = "MaKhachHang is required")]
        public string MaKhachHang { get; set; }
        [Required(ErrorMessage = "TenKhachHang is required")]
        public string TenKhachHang { get; set; }
        [Required(ErrorMessage = "SoDienThoai is required")]
        public string SoDienThoai { get; set; }

        [System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage", "CA2227:CollectionPropertiesShouldBeReadOnly")]
        public virtual ICollection<PhieuXuat> PhieuXuats { get; set; }
    }
}
```

Truy cập vào đường dẫn localhost/**port**/KhachHang/Create để xem kết quả trả về:



Mã khách hàng

Tên Khách hàng

Tên Khách hàng

Create

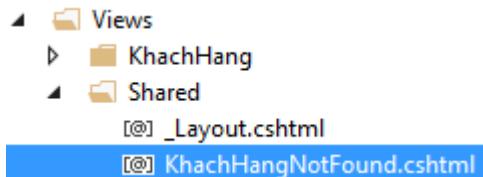
[Back to List](#)

Chú ý, Khi chưa nhập thông tin vào các trường dữ liệu bắt buộc như mã khách hàng, tên khách hàng và click vào nút lệnh Create thì sẽ có thông báo đưa ra là “MaKhachHang is required” và “TenKhachHang is required”. Nhập đầy đủ thông tin và quan sát kết quả trả về.



Phương thức Edit:

Đầu tiên chúng ta sẽ add một MVC 5 View Page (Razor) vào trong thư mục shared và đặt tên là KhachHangNotFound (view được trả về khi không tìm thấy thông tin của khách hàng)



Sau đó viết code cho file KhachHangNotFound như sau:

```
@{
    ViewBag.Title = "KhachHangNotFound";
    Layout = "~/Views/Shared/_Layout.cshtml";
}


Không tìm thấy thông tin của khách hàng theo yêu cầu của bạn. Vui lòng kiểm tra lại.



w3-text-blue">
Click vào đây, "Index", "KhachHang")
    để quay về danh sách khách hàng


```

Tiếp theo tạo action Edit trong controller KhachHang và viết code như sau:

```
public ActionResult Edit(string id)
{
    if (id==null)
    {
        return RedirectToAction("Index", "KhachHang");
    }
    else
    {
        KhachHang khachHang = entity.KhachHangs.Find(id);
        if (khachHang == null)
        {
            //return HttpNotFound();
            return View("KhachHangNotFound");
        }
        return View(khachHang);
    }
}
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult Edit([Bind(Include = "MaKhachHang,TenKhachHang,SoDienThoai")] KhachHang khachHang)
{
    if (ModelState.IsValid)
    {
        entity.Entry(khachHang).State = EntityState.Modified;
        entity.SaveChanges();
        return RedirectToAction("Index");
    }
    return View(khachHang);
}
```

Tiếp tục tạo view Edit trong thư mục view (xem lại cách tạo view Index và view Create ở trên) và viết code như sau:

```
@model QuanLyBanHang_LTQL.Models.KhachHang
@{
    ViewBag.Title = "Edit";
    Layout = "~/Views/Shared/_Layout.cshtml";
}

@using (Html.BeginForm())
{
    @Html.AntiForgeryToken()

    <div class="w3-margin w3-card-4">
        <div class="w3-blue w3-container"> <h3>Cập nhật thông tin khách hàng</h3> </div>
        @Html.ValidationSummary(true)
        @Html.HiddenFor(model => model.MaKhachHang)
        <div class="w3-container">
            <form>
                <p>
                    <label class="w3-text-blue">Tên khách hàng:</label>
                    @Html.EditorFor(model => model.TenKhachHang, new { htmlAttributes = new { @class = "w3-input" } })
                </p>
                <p>
                    <label class="w3-text-blue">Số điện thoại:</label>
                    @Html.EditorFor(model => model.SoDienThoai, new { htmlAttributes = new { @class = "w3-input" } })
                </p>
                <p>
                    <button class="w3-button w3-white w3-border w3-border-blue w3-round-large">Update</button>
                </p>
            </form>
        </div>
    </div>
}
<div class="w3-margin">
    @Html.ActionLink("Back to List", "Index")
</div>

@section Scripts {
    @Scripts.Render("~/bundles/jqueryval")
}
```

Sau đó rebuild lại project rồi chạy kiểm tra kết quả.



Phương thức Details:

Tương tự như trên, chúng ta tạo action Details trong KhachHangController rồi viết code như sau:

```
public ActionResult Details(string id)
{
    if (id == null)
    {
        return RedirectToAction("Index", "KhachHang");
    }
    KhachHang khachHang = entity.KhachHangs.Find(id);
    if (khachHang == null)
    {
        return View("KhachHangNotFound");
    }
    return View(khachHang);
}
```

Sau đó tạo view Details và viết code như sau:

```
@model QuanLyBanHang_LTQL.Models.KhachHang
@{
    ViewBag.Title = "Details";
    Layout = "~/Views/Shared/_Layout.cshtml";
}

<div class="w3-card-4 w3-margin">
    <div class="w3-blue w3-container">
        <h3>Thông tin chi tiết khách hàng</h3>
    </div>
    <ul class="w3-ul w3-border">
        <li>
            <label class="w3-text-blue">Mã khách hàng:</label>
            @Html.DisplayFor(model => model.MaKhachHang)
        </li>
        <li>
            <label class="w3-text-blue">Tên khách hàng:</label>
            @Html.DisplayFor(model => model.TenKhachHang)
        </li>
        <li>
            <label class="w3-text-blue">Số điện thoại:</label>
            @Html.DisplayFor(model => model.SoDienThoai)
        </li>
    </ul>
</div>
<p class="w3-margin">
    @Html.ActionLink("Edit", "Edit", new { id = Model.MaKhachHang })
    @Html.ActionLink("Back to List", "Index")
</p>
```

Sau đó rebuild lại project rồi chạy kiểm tra kết quả.



Phương thức Delete:

Tương tự như trên, chúng ta tạo action Delete trong KhachHangController rồi viết code như sau:

```
public ActionResult Delete(string id)
{
    if (id == null)
    {
        return RedirectToAction("Index", "KhachHang");
    }
    KhachHang khachHang = entity.KhachHangs.Find(id);
    if (khachHang == null)
    {
        return View("KhachHangNotFound");
    }
    return View(khachHang);
}

[HttpPost, ActionName("Delete")]
[ValidateAntiForgeryToken]
public ActionResult DeleteConfirmed(string id)
{
    KhachHang khachHang = entity.KhachHangs.Find(id);
    entity.KhachHangs.Remove(khachHang);
    entity.SaveChanges();
    return RedirectToAction("Index");
}
```

Sau đó tạo view Delete và viết code như sau:

```
@model QuanLyBanHang_LTQL.Models.KhachHang
@{
    ViewBag.Title = "Delete";
    Layout = "~/Views/Shared/_Layout.cshtml";
}
<div class="w3-card-4 w3-margin">
    <div class="w3-blue w3-container">
        <h3>Bạn có chắc chắn muốn xóa thông tin của khách hàng này không?</h3>
    </div>
    <ul class="w3-ul w3-border">
        <li>
            <label class="w3-text-blue">Mã khách hàng:</label>
            @Html.DisplayFor(model => model.MaKhachHang)
        </li>
        <li>
            <label class="w3-text-blue">Tên khách hàng:</label>
            @Html.DisplayFor(model => model.TenKhachHang)
        </li>
        <li>
            <label class="w3-text-blue">Số điện thoại:</label>
            @Html.DisplayFor(model => model.SoDienThoai)
        </li>
    </ul>
    @using (Html.BeginForm())
    {
        @Html.AntiForgeryToken()
        <div class="form-actions no-color w3-padding">
            <button class="w3-button w3-white w3-border w3-border-blue w3-round-large">Delete</button>
            @Html.ActionLink("Back to List", "Index")
        </div>
    }
</div>
```

Sau đó rebuild lại project rồi chạy kiểm tra kết quả.

3. Tạo báo cáo