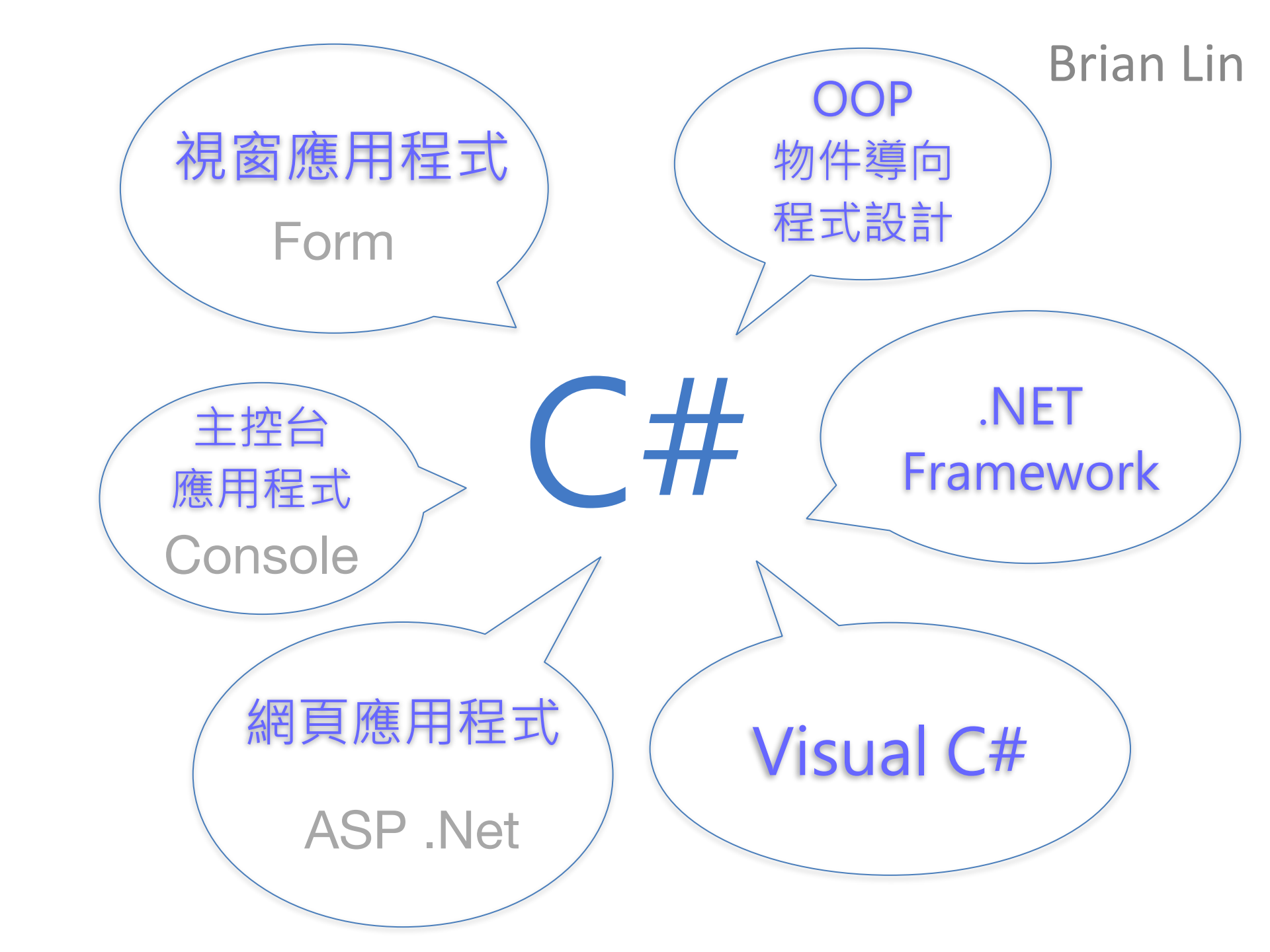
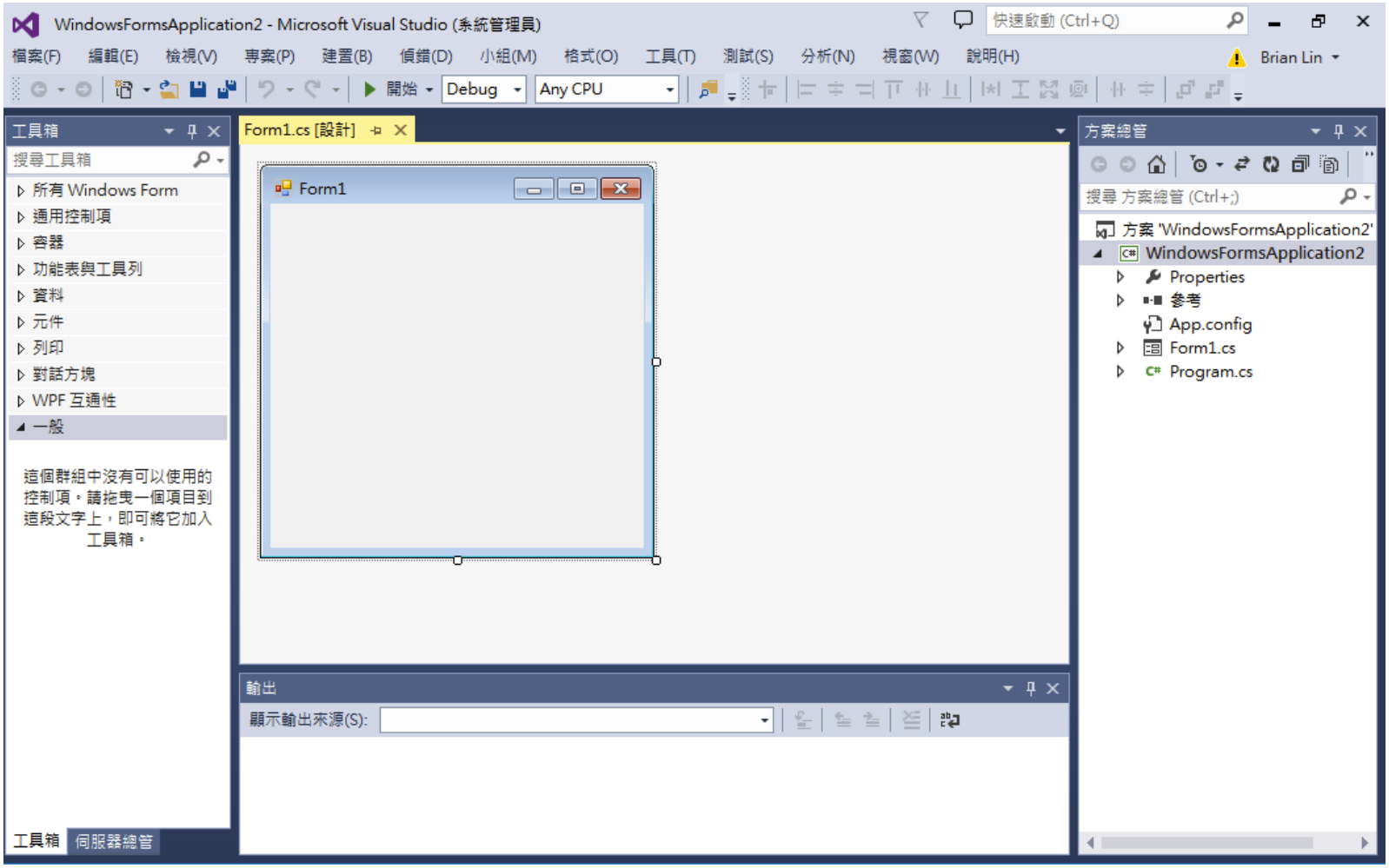
C#程式設計

Brian Lin


Brian Lin
視窗應用程式
Form
OOP
物件導向
程式設計
主控台
應用程式
Console
C#
.NET
Framework
網頁應用程式
ASP .Net
Visual C#

Visual Studio 整合開發環境



下載 https://www.visualstudio.com/downloads/

變數的命名規則

- 第一個字元不可為數字，例 0-9。
- 第一個字元可以是大寫或小寫英文字母或是底線 (_)。避免用中文命名變數。
- 變數名稱中間不可以有空白字元。
- 英文字母大小寫視為不同的變數，例 number 和 Number 是不同的變數。
- 對的命名方式：

  total_amount

  _myTotalMoney
- 錯的命名方式：

  7Eleven

  !Waring

  good man

基本的資料形態

- int (整數)32位元，long 64位元，short 16 位元
  - int i = 3;
- float (浮點數)精確度7位小數，double精確度16位小數
  - float a = 3.14f;
  - double b = 3.141592653589793;
- bool (布林值)
  - true 或 false。
- char (字元值)
  - char a = 'm';
- string (字串)
  - string myString = "Hello World !!";

算數運算子

- 加(+)，減(-)，乘(*)，除(/)：
  - 先乘除，後加減。
- 整數運算：
  - int a; a = 3 / 2; // a答案是1，整數運算會去小數點。
- 負數運算：
  - -a; // 加上負號。
- 模(餘)數運算：
  - c = a % b; // c 答案是 a 除以 b 的餘數。
- 整數與浮點運算關係：
  - 浮點數和整數運算，答案沒有影響，小數點會保留。
- 指定運算子 (+=; -= ; *=; /=;)：
  - c += 3 等於 c = c +3 //指定運算子。
- 遞增運算(++; --;)：
  - a++ 等於 a=a+1。

條件判斷式

- if ... else 判斷式，else if 判斷式。

```
if (x<0) { s =-1;} else { s = x*x;};
```

- switch 判斷式。

```
switch (i) {
case 1:
break;
case 2:
break;
default:
break;
}
```

- 條件運算式。

```
s = (x < 0)? -1 : x*x;
```

迴圈 (loop)

- for (int i=0;i<10;i++) { };
  - ◇ for迴圈，巢狀 for 迴圈。
- while (i < 10) {i++;};
  - ◇ while 迴圈。
- do { i++; } while (i<10);
  - ◇ do...while 迴圈。
- break 與 continue 敘述：

關係與條件運算式

==	等於
!=	不等於
>	大於
>=	大於等於
<	小於
<=	小於等於
&&	條件 AND
||	條件 OR

陣列 (Array)

- 陣列是一種資料結構，用來儲存多個相同型別的變數。陣列視為物件。
- 陣列宣告的形式:
  - int[] array1 = new int[6] ; // 一維陣列;
  - int[] array2 = new int[] { 1, 2, 3, 4, 5,6 };
  - int[] array3 = { 1, 2, 3, 4, 5, 6 };
  - int[,] twoDimArray1 = new int[2, 3]; //二維陣列;
  - int[,] twoDimArray2 = { { 1, 2, 3 }, { 4, 5, 6 } };
  - int[, ,] array3D = new int[,,] { { { 1, 2, 3 } }, { { 4, 5, 6 } } }; // 三維陣列;

建立第一個Form程式

- Program.cs
  - ◇ 為程式進入點。
- Form1.cs
  - ◇ 載入第一個表單。
- Form1.Designer.cs
  - ◇ 為自動產生，不要修改。
- Form1 設計檢視
  - ◇ 在空白處，點擊兩下，Forma1.cs 會產生 Form1_Load 方法。

.cs 程式的內容 (Form1.cs為例)

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

namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            Console.WriteLine("Hi, 第一個 C# 程式.");

        }
    }
}
```

引用的命名空間

自身的命名空間

部分類別定義

類別方法

類別方法

命名空間 Namespace

- 命名空間提供一個有效的方法，去組織類別的程式碼，並且避免類別名稱重覆的問題，而且可以有效縮短執行方法的程式碼長度。

- 例如：
  - 在檔頭宣告
  - Using System.Windows.Forms;
  - 原來的程式碼
  - System.Windows.Forms.MessageBox.Show("Hello !!");
  - 可以改寫成
  - MessageBox.Show("Hello !!");

Console.WriteLine 格式化輸出

- Console.WriteLine(“Hi, 第一個 C# 程式.”);
  - ◇ 在輸出視窗檢視。
- Console.WriteLine ("pi = {0}", 3.14);
- Console.WriteLine("Today is {0}-{1}-{2}", 2013, 1, 3);

MessageBox 提示視窗

- MessageBox.Show(); //可以顯示出提示視窗;

  形式1：

  MessageBox.Show ("訊息內容", "標題列", MessageBoxButtons.OK);

  形式２：

  MessageBox.Show ("訊息內容", "標題列", MessageBoxButtons.OK, MessageBoxIcon.Error);

基本的 UI 元件

- Label (文字標籤)
- Button (按鈕)
- TextBox (文字輸入框)
- 各 UI 相對應的事件探討。

型別轉換

- Convert.ToString();
  - ◇ 轉換到字串 String 型別。
- Convert.ToInt32();
  - ◇ 轉換到整數 Int 型別。
- Convert.ToSingle();
  - ◇ 轉換到單精確浮點數 float 型別。
- Convert.ToDouble();
  - ◇ 轉換到雙精確浮點數 double 型別。
- Convert.ToBoolean();
  - ◇ 轉換到布林值 Boolean 型別。

方法 (Methods)

- 方法（類似函式或副程式）提供了有效的方式，將程式碼更有效率的應用 ，可將龐大成拆分成片斷，並重覆使用。

- 方法的幾個特色：
  ① 提供回傳值或不回傳值的功能。
  ② 方法間可以傳遞不同的參數。
  ③ 方法有兩種，一種是靜態方法（Static Methods），一種是非靜態方法或稱動態方法（Non-static Methods）。

方法的參數傳遞

- 無參數傳遞的方法：

```
static void showMessage() {
    MessageBox.Show( "Hello!!  你好。");
}
```

- 含參數傳遞的方法：

```
static void sendMessage(string strMessage) {
    string output = string.Format("傳遞的訊息是：{0}", strMessage);
    MessageBox.Show(output );
}
```

靜態與非靜態方法

- 靜態方法（Static）類別不需實體化即可執行：

```
static void printAandB(int a, int b) {
    string output =  string.Format("a是{0}，b是{1}",a,b);
    MessageBox.Show(output);
}
```

- 非靜態方法（Non-static）類別需實體化才可執行：

```
int sum(int a, int b) {
    int c;
    c= a+b;
    return c;
}
```

方法的回傳值

- 無回傳值的方法：

```
void printPrice(float price) {
    string output = string.Format("價格是:{0:C}", price);
    MessageBox.Show(output);
}
```

- 有回傳值的方法：

```
int sum(int a, int b) {
    int c;
    c= a+b;
    return c;
}
```

參數傳遞的方式

- 傳值呼叫 (Call by Value)。
  - ◇ 例：public void call_value(int n);
  - ◇ 此方法傳遞的參數值在方法或函式的內部，會用複製的方式傳遞參數，所以會儲存在不同的記憶體空間。

- 傳址呼叫 (Call by Address 或 Call by Reference)
  - ◇ 例：public void call_reference(ref int n);
  - ◇ 此方法傳遞的參數值在方法或函式的內部，會傳遞原有參數的記憶體位址，所以會共用參數相同的記憶體空間。

陣列的操作

- foreach () {…}; 迴圈讀取所有陣列元素。
- myArray.GetUpperBound(); 取得陣列的上限 index 值。
- Array.Sort(myArray); 單一陣列排序。
- Array.Sort(myKeyArray, myValueArray); 兩個鍵值陣列相關排序;
- Array.IndexOf(myArray, key); 利用鍵值尋找陣列元素，並回傳 index。

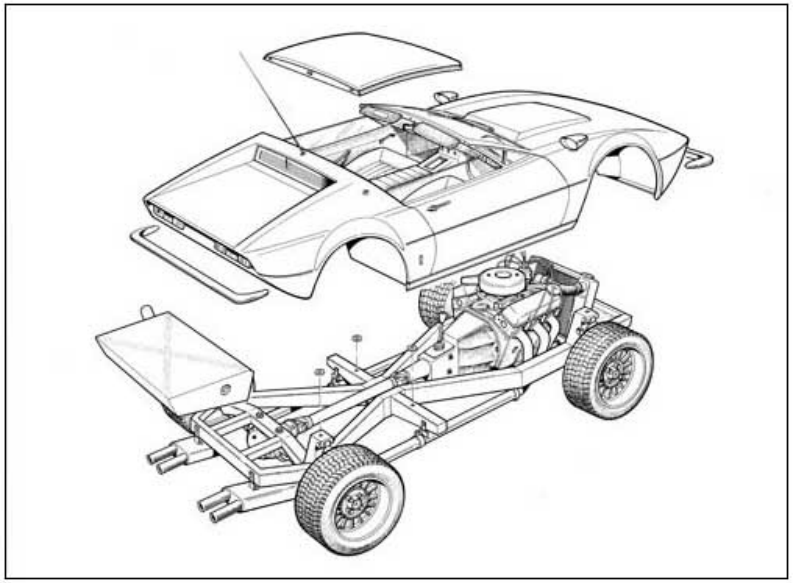
List 與 ArrayList

- List 與 ArrayList 是陣列的更進階型態。

- [] 陣列：宣告的資料物件類型固定，長度也固定。
- List：宣告的資料物件類型固定，但長度不固定，可動態增減。
- ArrayList：宣告的資料物件類型不固定，可混合不同類型的物件，但使用上必須轉型，長度亦不固定，可動態增減。

類別（Class）與物件（Object）

類別 Class



New

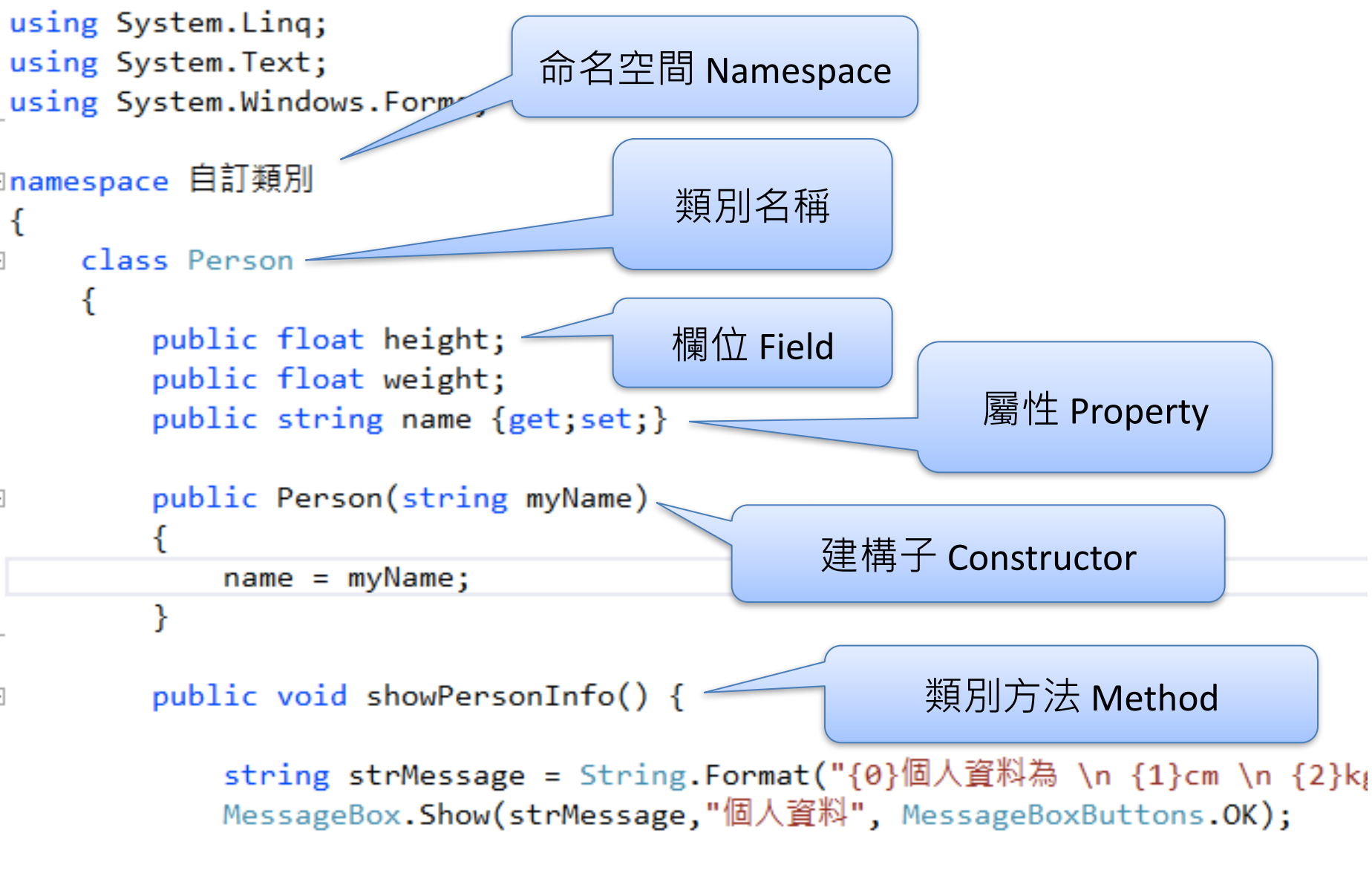
物件 object
Instance

物件，就是類別實體化之後的結果。

類別 Class

- 類別是物件導向開發（OOP）的重要基礎，而程式中最小的運作單元，就是物件（Object），也就是由類別（Class）產生的，提供並組成程式所需要的功能。
- 類別的構成：
  - ✓ C#的一切，都是基於類別。
  - ✓ 類別由方法（Method），欄位（Field），和屬性（Property）組成。
  - ✓ 類別俱有封裝的特性，利用公開（Public）或私有（Private），保護所屬成員的資料。
  - ✓ 利用建構子（Constructor），來初始化物件。但C#亦可不需要使用建構子來初始化。

自定類別 Class


using System.Linq;
using System.Text;
using System.Windows.Form
namespace 自訂類別
{
class Person
{
public float height;
public float weight;
public string name {get;set;}
public Person(string myName)
{
name = myName;
}
public void showPersonInfo() {
string strMessage = String.Format("{0}個人資料為 \n {1}cm \n {2}k
MessageBox.Show(strMessage,"個人資料", MessageBoxButtons.OK);
命名空間 Namespace
類別名稱
欄位 Field
屬性 Property
建構子 Constructor
類別方法 Method

命名空間 Namespace

- 命名空間提供一個有效的方法，去組織類別的程式碼，並且避免類別名稱重覆的問題，而且可以有效縮短執行方法的程式碼長度。

- 例如：
  - 在檔頭宣告
  - Using System.Windows.Forms;
  - 原來的程式碼
  - System.Windows.Forms.MessageBox.Show("Hello !!");
  - 可以改寫成
  - MessageBox.Show("Hello !!");

繼承 Inheritance

- 繼承是物件導向開發最主要的觀念，當需要創建建自己的類別時，可以繼承現有的類別，加以使用其功能，重覆利用已寫過的類別程式碼。
- 例如：

父類別：

```
Class Car { … }
```

繼承父類別：

```
Class Porsche : Car { … }
```

使用：

```
Car myCar = new Car();
Porsche myPorsche = new Porsche();
```

多型 Polymorphism

- 多型又可稱同名異式，在類別中建立同名的方法或屬性，來提供不同的功能或傳遞不同的參數，多型主要有兩種型態，多載和覆寫。

- 覆寫 (Override)：宣告相同方法名稱，也使用相同參數，取代原來方法功能。
- 多載：宣告相同方法名稱，但使用不同參數。

列舉 Enumeration

- 列舉(Enumeration)是一系列的整數常數的定義，方法是利用變數的命名，與所需的整數常數對應起來，提供程式碼來存取使用，宣告的關鍵字為enum。

  形式如下：

  enum Days { Sun = 7, Mon = 1, Tue = 2,Wed = 3, Thu = 4, Fri = 5, Sat =6 }

介面 Interface

- 介面(Interface)提供了一種程式設計的模式，利用相同的介面，在不同類別繼承並實作出不同功能的方法，但仍舊可以維持相同的方法或屬性的名稱。

- 介面僅能宣告成員，不能實體化(new)，亦無實作程式碼，必須由類別繼承來實作。
- 介面的成員可以是屬性，方法，事件。
- 類別雖只能繼承單一父類別，但可繼承多個介面。
- 類別繼承了某介面，則該介面的所有成員都必須實作。

泛型 Generics

- 泛型(Generics)的概念是，當我們在定義類別，介面或方法的時候，可以不用先指定其資料型態，之後實體化來使用時，再決定其型別。

```
// 泛型，未指定的資料型態，由 T 關鍵字代替;
public class MyGenericClass<T>
{
    public T myGenericProperty;
    public void MyMethod(T input) { }
}
```

結構

- 結構(Struct)是一種值型態(Value Type)的資料結構，定義方式跟類別有點類似，但不完全一樣，以下是結構的特點。

- 結構是值型態(Value Type)，類別是參考型態(Reference Type)。
- 結構不支援繼承，也就是不能繼承，也不能被繼承。
- 結構的建構子，一定要有參數。
- 結構的欄位，屬性，不能在宣告的時候賦值，可以在建構式的階段賦值。

委派 Delegate

- 委派（Delegate），就是將方法（Method）物件化，於是可以將物件化後的方法，代入方法的參數，然後傳遞呼叫使用。

- Delegate 實作的流程：
  1. 定義委派（Delegate），目標（Target）方法相關參數，需對應。
  2. 指定委派的目標方法（Target Method）。
  3. 調用委派執行動作。