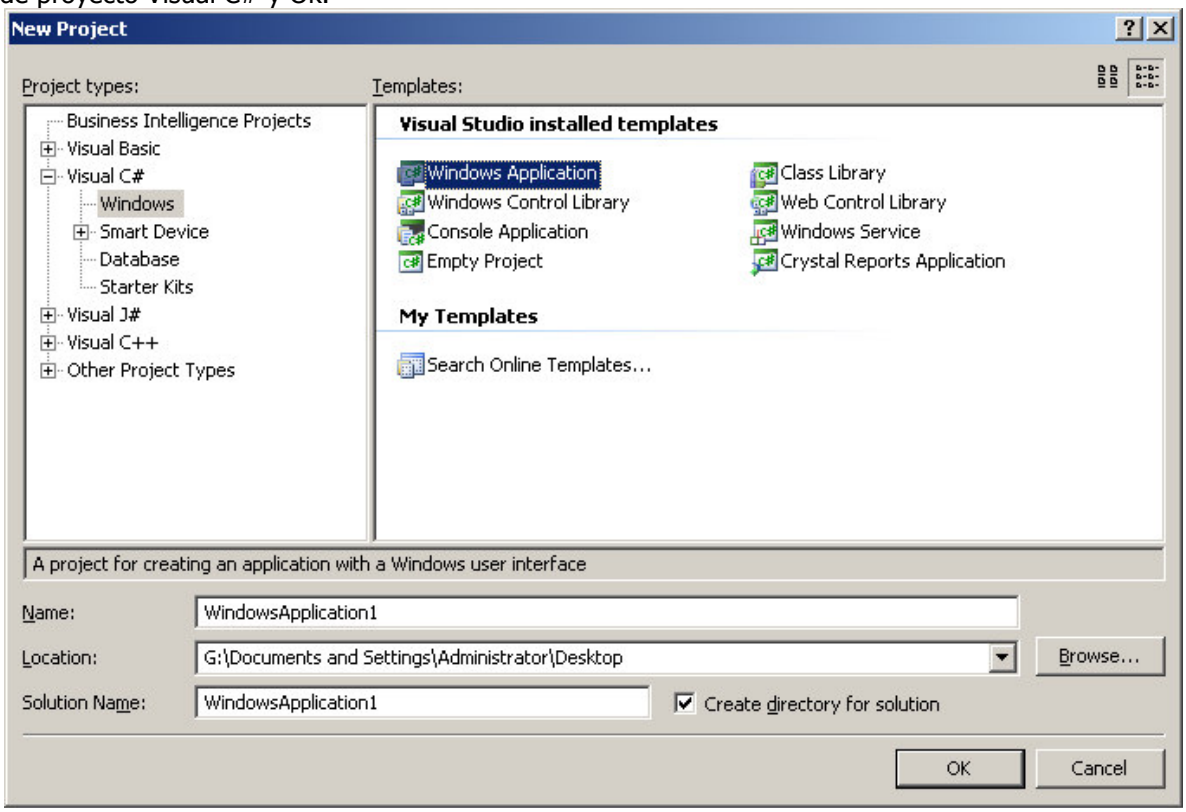
LA BASE DE C# .NET 2005

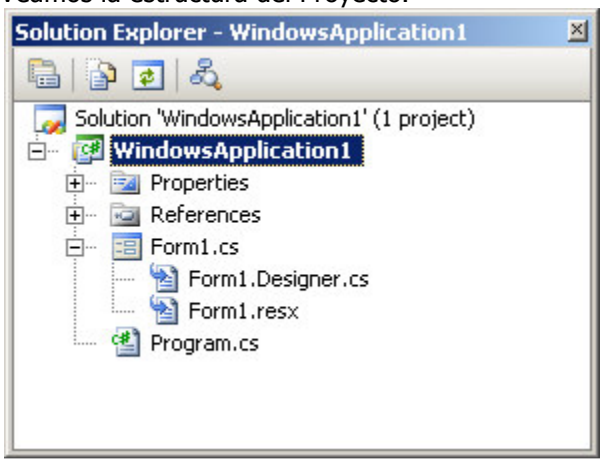
C# es un paquete de Visual Studio .NET 2005, y una aplicación es una colección de una o más clases, estructuras y otros tipos.

Paso 1: La creación de un proyecto en C#.

- Primero abrimos .NET 2005, nos ubicamos en File – New – Proyect y seleccionamos el tipo de proyecto Visual C# y Ok.



- Veamos la estructura del Proyecto.



Este proyecto no es diferente al Proyecto en Visual Basic.
References : En esta carpeta se hace referencia a todas librerías que son dependientes del formulario.
Program.cs : En esta clase existe un método Main que viene a hacer el punto de inicio del formulario que se va ejecutar dentro del Proyecto.

- Estructura de la clase Program.cs

```
using System;
using System.Collections.Generic;
using System.Windows.Forms;

namespace WindowsApplication1
{
  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());
    }
  }
}
```

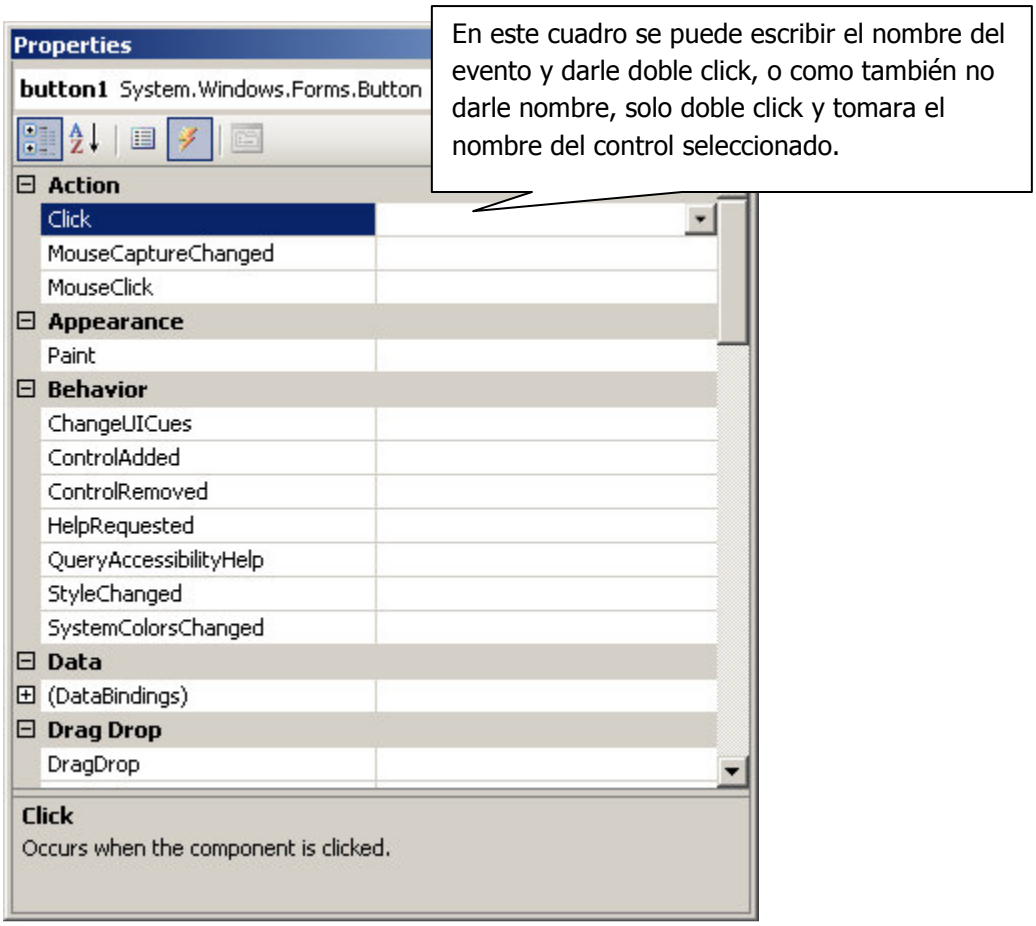
Donde:
Application.Run(new Form1()); => En esta linea nosotros podemos establecer que formulario se va ejecutar en el momento de cargar la Aplicación.
Ejemplo: si agregamos un nuevo formulario llamado **Form2.cs** ,y queremos ejecutar con la tecla de función F5, modificamos la línea.

```
Application.Run(new Form2());
```

Paso 2: Selección de eventos y llamada a controles.

- Agregamos un par de controles a en el formulario.

- Para seleccionar un evento se puede hacer de 2 formas:
  - Podemos darle doble click en el control y seleccionar el evento predeterminado del control, como el evento click, textchange, etc…
  - Otra forma y recomendable es seleccionar un evento en el cuadro de la propiedad del control que ha seleccionado.

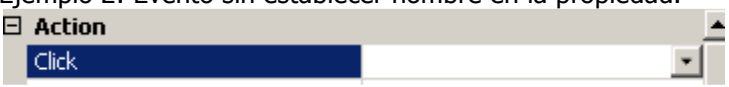


- Ejemplo 1: Evento con un nombre asignado en la propiedad.


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

}
```

- Ejemplo 2: Evento sin establecer nombre en la propiedad.




```
private void button1_Click(object sender, EventArgs e)
{

}
```

- **Código fuente:**

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

namespace WindowsApplication1
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
      }

      private void button1_Click(object sender, EventArgs e)
      {
         label1.Text = "Mi Primer Programa";
      }
   }
}
```

Paso 3: Llamada de un formulario a otro.

- Llamada del formulario Form1 a Form2

```
//Crear un objeto a travez de la clase Form2
 Form2 frm = new Form2();
//utilizando la propiedad Show para mostrar el formulario Form2
 frm.Show();
//ocultar el formulario actual
 this.Hide();
```

- Llamada del formulario Form2 a Form1

```
Form1 frm = new Form1();
frm.Show();
 this.Hide();
```

- Salir de la aplicacion.

```
if (MessageBox.Show("Desea Salir","Aplicacion",
MessageBoxButtons.YesNo,MessageBoxIcon.Question)==DialogResult.Yes)
    Application.Exit();//salir de la aplicacion
```

Tipos de Datos en C#

La siguiente tabla muestra los tamaños e intervalos de los tipos integrales, que constituyen un subconjunto de los tipos simples.

Tipo	Intervalo	Tamaño
sbyte	-128 a 127	Entero de 8 bits con signo
byte	0 a 255	Entero de 8 bits sin signo
char	U+0000 a U+ffff	Carácter Unicode de 16 bits
short	-32.768 a 32.767	Entero de 16 bits con signo
ushort	0 a 65.535	Entero de 16 bits sin signo
int	-2.147.483.648 a 2.147.483.647	Entero de 32 bits con signo
uint	0 a 4.294.967.295	Entero de 32 bits sin signo
long	-9.223.372.036.854.775.808 a 9.223.372.036.854.775.807	Entero de 64 bits con signo
ulong	0 a 18.446.744.073.709.551.615	Entero de 64 bits sin signo

- **bool**

  La palabra clave **bool** es un alias de System.Boolean. Se utiliza para declarar variables que almacenan los valores booleanos, true y false.

  Ejemplo:

  Puede asignar un valor booleano a una variable de tipo **bool**. También es posible asignar una expresión que se evalúa como **bool** a una variable de tipo **bool**.

```
using System;
public class MyClass
{
   static void Main()
   {
      bool i = true;
      char c = '0';
      Console.WriteLine(i);
      i = false;
      Console.WriteLine(i);

      bool Alphabetic = (c > 64 && c < 123);
      Console.WriteLine(Alphabetic);
   }
}
```

Resultado:
True
False
False

- **int**
  La palabra clave **int** denota un tipo integral que almacena valores según el tamaño y el intervalo que se indican en la tabla siguiente.

Tipo	Intervalo	Tamaño	Tipo de .NET Framework
int	-2.147.483.648 a 2.147.483.647	Entero de 32 bits con signo	System.Int32

*Las variables de tipo **int** se pueden declarar e inicializar como en el siguiente ejemplo:*
int i = 123;
*Existe una conversión implícita predefinida de **int** a long, float, double o decimal. Por ejemplo:*

'123' no es un int, así una conversión implícita tomar lugar aquí.
float f = 123;
*Existe una conversión implícita predefinida de sbyte, byte, short, ushort o char a **int**. Por ejemplo, la instrucción de asignación siguiente genera un error de compilación si no se emplea una conversión explícita:*

```
long aLong = 22;
int i1 = aLong;       // Error: conversión no implícita desde long.
int i2 = (int)aLong;  // OK: conversion explícita.
```

*Observe que no existe conversión implícita de tipos de punto flotante a **int**. Por ejemplo, la instrucción siguiente generará un error de compilación, a menos que se utilice una conversión explícita:*

```
int x = 3.0;          // Error: conversión no implícita desde un tipo double.
int y = (int)3.0;     // OK: conversion explícita.
```

- **double**
  La palabra clave **double** denota un tipo simple que almacena valores de punto flotante de 64 bits. La siguiente tabla muestra la precisión y el intervalo de valores aproximado para el tipo **double**.

Tipo	Intervalo aproximado	Precisión	Tipo de .NET Framework
double	±5.0 × 10.324 a ±1.7 × 10308	15-16 dígitos	System.Double

*De forma predeterminada, un literal numérico real en el lado derecho del operador de asignación se trata como un valor de tipo **double**. No obstante, si desea tratar un número entero como **double**, utilice el sufijo d o D, por ejemplo:*

```
double x = 3D;
```

*En el ejemplo siguiente, se suman valores int, short, float y **double** que dan un resultado **double**.*

```
using System;
class TiposMesclados
{
   static void Main()
```

```
    {
        int x = 3;
        float y = 4.5f;
        short z = 5;
        double w = 1.7E+3;
            // El resultado del Segundo argumento es un tipo double.
          Console.WriteLine("La suma es {0}", x + y + z + w);
    }
}
```

Resultado: La suma es 1712.5

- **float**
  La palabra clave **float** denota un tipo simple que almacena valores de punto flotante de 32 bits. La siguiente tabla muestra la precisión y el intervalo de valores aproximado para el tipo **float**.

Tipo	Intervalo aproximado	Precisión	Tipo de .NET Framework
float	±1,5 × 10–45 a ±3,4 × 1038	7 dígitos	System.Single

De forma predeterminada, un literal numérico real en el lado derecho del operador de asignación se trata como double. Por consiguiente, para inicializar una variable de tipo float, utilice el sufijo f o F de la manera siguiente:

```
float x = 3.5F;
```

*Si no utiliza este sufijo en la declaración anterior, obtendrá un error de compilación, ya que está intentando almacenar un valor de tipo double en una variable de tipo **float**.*

*En el siguiente ejemplo, una expresión matemática incluye valores de tipo int, short y **float**, y proporciona un resultado de tipo **float**. Observe que la expresión no incluye ningún valor de tipo double.*

```
using System;
class TiposMesclados
{
    static void Main()
    {
        int x = 3;
        float y = 4.5f;
        short z = 5;
        Console.WriteLine("El resultado es {0}", x * y / z);
    }
}
```

Resultado:
El resultado es 2.7

- **long**
  La palabra clave **long** denota un tipo integral que almacena valores según el tamaño y el intervalo que se indican en la tabla siguiente.

Tipo	Intervalo	Tamaño	Tipo de .NET Framework

long	–9,223,372,036,854,775,808 a 9,223,372,036,854,775,807	Entero de 64 bits con signo	System.Int64

*Las variables de tipo **long** se pueden declarar e inicializar como en el siguiente ejemplo:*
long long1 = 4294967296;

*Cuando un literal entero no tiene sufijo, su tipo es el primero de estos tipos en el que se puede representar su valor: int, uint, **long**, ulong. En el ejemplo anterior, es del tipo **long**, ya que supera el intervalo de uint.*

*También se puede utilizar el sufijo L con el tipo **long** de este modo:*

long long2 = 4294967296L;

*Cuando se utiliza el sufijo L, el tipo del entero literal será **long** o ulong, según su tamaño. En este caso, es **long**, ya que es menor que el intervalo de ulong.*

*El tipo **long** se puede utilizar con otros tipos integrales numéricos en la misma expresión, en cuyo caso la expresión se evalúa como **long** (o bool en el caso de expresiones relacionales o booleanas). Por ejemplo, la siguiente expresión se evalúa como **long**:*

898L + 88

*Existe una conversión implícita predefinida de **long** a float, double o decimal. En los demás casos, se debe utilizar una conversión explícita. Por ejemplo, la instrucción siguiente genera un error de compilación si no se emplea una conversión explícita:*

```
int x = 8L;        // Error: conversion no implicita desde long para int
int x = (int)8L;   // OK: conversion explicita para int
```

*Existe una conversión implícita predefinida de sbyte, byte, short, ushort, int, uint o char a **long**.*

*Observe que no existe conversión implícita de tipos de punto flotante a **long**. Por ejemplo, la instrucción siguiente generará un error de compilación, a menos que se utilice una conversión explícita:*

```
long x = 3.0;        // Error: conversión no implícita desde double.
long y = (long)3.0;  // OK: conversion explicita.
```

- **byte**
  La palabra clave **byte** denota un tipo integral que almacena valores según se muestra en la tabla siguiente.

Tipo	Intervalo	Tamaño	Tipo de .NET Framework
byte	0 a 255	Entero de 8 bits sin signo	System.Byte

*Las variables de tipo **byte** se pueden declarar e inicializar como en el siguiente ejemplo:*

```
byte myByte = 255;
```

*En la declaración anterior, el literal entero 255 se convierte implícitamente del tipo int al tipo **byte**. Si el literal entero supera el intervalo de valores del tipo **byte**, se producirá un error de compilación.*

```
byte x = 10, y = 20;
```

*La instrucción de asignación siguiente producirá un error de compilación, ya que la expresión aritmética del lado derecho del operador de asignación se evalúa de forma predeterminada como **int**.*

// Error: conversion desde int para byte:
byte z = x + y;
Para solucionar este problema, utilice una conversión explícita:

// OK: explicit conversion:
byte z = (byte)(x + y);
*Sin embargo, no existe conversión implícita de tipos de punto flotante a tipo **byte**. Por ejemplo, la instrucción siguiente generará un error de compilación, a menos que se utilice una conversión explícita:*

// Error: no implicit conversion from double:
byte x = 3.0;
// OK: conversión explicita:
byte y = (byte)3.0;

- **char**
  La palabra clave **char** se utiliza para declarar un carácter Unicode en el intervalo indicado en la siguiente tabla. Los caracteres Unicode son caracteres de 16 bits que se utilizan para representar la mayoría de los lenguajes escritos de todo el mundo.

Tipo	Intervalo	Tamaño	Tipo de .NET Framework
char	U+0000 a U+ffff	Carácter Unicode de 16 bits	System.Char

*Las constantes de tipo **char** se pueden escribir como literales de cadena, secuencias de escape hexadecimales o representaciones Unicode. Los códigos de caracteres integrales se pueden convertir explícitamente al tipo char. En las siguientes instrucciones se declara una variable de tipo **char** y se inicializa con el carácter X:*

char char1 = 'Z'; // Character literal
char char2 = '\x0058'; // Hexadecimal
char char3 = (char)88; // Cast from integral type
char char4 = '\u0058'; // Unicode

- **decimal**
  La palabra clave **decimal** denota un tipo de datos de 128 bits. Comparado con los tipos de punto flotante, el tipo **decimal** tiene una mayor precisión y un intervalo más reducido, lo que lo hace adecuado para cálculos financieros y monetarios. El intervalo aproximado y la precisión para el tipo **decimal** aparecen en la siguiente tabla.

Tipo	Intervalo aproximado	Precisión	Tipo de .NET Framework
decimal	±1.0 × 10e−28 a ±7.9 × 10e28	28-29 dígitos significativos	System.Decimal

*Si desea tratar un literal numérico real como **decimal**, utilice el sufijo m o M; por ejemplo:*

decimal myMoney = 300.5m;

Sin el sufijo m, el número se trata como double, lo cual genera un error del compilador.

*Los tipos integrales se convierten implícitamente a **decimal** y el resultado se evalúa como **decimal**. Por consiguiente, es posible inicializar una variable de tipo decimal mediante un literal entero sin el sufijo, de la manera siguiente:*

decimal myMoney = 300;

*No existe conversión implícita entre tipos de punto flotante y el tipo **decimal**; por lo tanto, se debe utilizar una conversión de tipos para convertir estos dos tipos. Por ejemplo:*

```
decimal myMoney = 99.9m;
double x = (double)myMoney;
myMoney = (decimal)x;
```

*Además, se pueden combinar tipos integrales numéricos y **decimal** en la misma expresión. No obstante, si se combinan tipos de punto flotante con el tipo **decimal** sin realizar una conversión de tipos, se producirá un error de compilación.*

*En este ejemplo, se combina un tipo **decimal** y un tipo int en la misma expresión. El resultado se evalúa como tipo **decimal**.*

*Si intenta agregar las variables double y **decimal** mediante una instrucción como ésta:*

```
double x = 9;
Console.WriteLine(d + x); // Error
```

Obtendrá el siguiente mensaje de error:

Operator '+' cannot be applied to operands of type 'double' and 'decimal'

```
// keyword_decimal.cs
// conversion a decimal
using System;
public class TestDecimal
{
   static void Main ()
   {
      decimal d = 9.1m;
      int y = 3;
      Console.WriteLine(d + y);
  }
}
```

Resultados

```
  12.1
```

- **sbyte**
  La palabra clave **sbyte** denota un tipo integral que almacena valores según el tamaño y el intervalo que se indican en la tabla siguiente.

Tipo	Intervalo	Tamaño	Tipo de .NET Framework
sbyte	–128 a 127	Entero de 8 bits con signo	System.SByte

*Las variables de tipo **sbyte** se pueden declarar e inicializar como en el siguiente ejemplo:*

```
sbyte sByte1 = 127;
```

*En la declaración anterior, el literal entero 127 se convierte implícitamente del tipo int al tipo **sbyte**. Si el literal entero supera el intervalo de valores del tipo **sbyte**, se producirá un error de compilación.*

*Considere, por ejemplo, las dos variables siguientes de tipo **sbyte**, x e y:*

```
sbyte x = 10, y = 20;
```

La instrucción de asignación siguiente producirá un error de compilación, ya que la expresión aritmética del lado derecho del operador de asignación se evalúa como int de forma predeterminada.

sbyte z = x + y; // Error: conversion from int to sbyte

Para corregir este problema, convierta la expresión del modo siguiente:

sbyte z = (sbyte)(x + y); // OK: explicit conversión

Sin embargo, es posible utilizar las instrucciones siguientes, donde la variable de destino tiene un tamaño de almacenamiento igual o superior:

```
sbyte x = 10, y = 20;
int m = x + y;
long n = x + y;
```

*Observe que no existe conversión implícita de tipos de punto flotante a **sbyte**. Por ejemplo, la instrucción siguiente generará un error de compilación, a menos que se utilice una conversión explícita:*

```
sbyte x = 3.0;        // Error: no implicit conversion from double
sbyte y = (sbyte)3.0;  // OK: explicit conversión
```

- **short**

  La palabra clave **short** denota un tipo de datos integral que almacena valores según el tamaño y el intervalo que se indican en la tabla siguiente.

Tipo	Intervalo	Tamaño	Tipo de .NET Framework
short	–32.768 a 32.767	Entero de 16 bits con signo	System.Int16

*Las variables de tipo **short** se pueden declarar e inicializar como en el siguiente ejemplo:*

short x = 32767;

*Considere, por ejemplo, las dos variables siguientes de tipo **short**, x e y:*

short x = 5, y = 12;

La instrucción de asignación siguiente producirá un error de compilación, ya que la expresión aritmética del lado derecho del operador de asignación se evalúa como int de forma predeterminada.

short z = x + y; // Error: no conversion from int to short

Para solucionar este problema, utilice una conversión explícita:

short z = (**short**)(x + y); // OK: explicit conversion

Sin embargo, es posible utilizar las instrucciones siguientes, donde la variable de destino tiene un tamaño de almacenamiento igual o superior:

```
int m = x + y;
long n = x + y;
```

*No existe conversión implícita de tipos de punto flotante a tipo **short**. Por ejemplo, la instrucción siguiente generará un error de compilación, a menos que se utilice una conversión explícita:*

```
short x = 3.0;        // Error: no implicit conversion from double
short y = (short)3.0;  // OK: explicit conversion
```

- **string**
  El tipo **string** representa una cadena de caracteres Unicode. **string** es un alias para String en .NET Framework. Las cadenas son *inmutables*: no se puede cambiar el contenido de un objeto de tipo string después de crearlo.
  *Aunque **string** es un tipo de referencia, los operadores de igualdad (**==** y **!=**) se definen para comparar los valores de objetos **string**, no las referencias. De esta forma, es más intuitivo comprobar la igualdad entre cadenas. Por ejemplo:*

```
string a = "hello";
string b = "h";
// Append to contents of 'b'
b += "ello";
Console.WriteLine(a == b);
Console.WriteLine((object)a == (object)b);
```

  Esto presenta "True" y, después, "False" porque el contenido de las cadenas es equivalente, pero a y b no hacen referencia a la misma instancia de cadena.

  El operador + concatena cadenas:

```
    string a = "good " + "morning";
```

  Esto crea un objeto de tipo string que contiene "good morning".

  *El operador [] se puede utilizar para tener acceso a caracteres individuales de un objeto **string**:*

```
    string str = "test";
char x = str[2];  // x = 's';
```

  *Los literales de cadena son objetos de tipo **string** que se pueden escribir de dos formas: entre comillas o entre comillas y precedidos de @. Los literales de cadena se deben encerrar entre comillas ("):*

```
"good morning"  // a string literal
```

  Los literales de cadena pueden contener cualquier literal de carácter, incluidas las secuencias de escape:

```
    string a = "\\\u0066\n";
```

  Esta cadena contiene una barra diagonal inversa, la letra f y el carácter de nueva línea.

  Los literales de cadena entrecomillados y precedidos por @ empiezan con este símbolo y se encierran entre comillas. Por ejemplo:

```
@"good morning"  // a string literal
```

  La ventaja de utilizar la combinación del signo @ y el entrecomillado es que las secuencias de escape no se procesan, lo que facilita la escritura, por ejemplo, de un nombre de archivo completo:

```
@"c:\Docs\Source\a.txt"  // rather than "c:\\Docs\\Source\\a.txt"
```

  Para incluir comillas tipográficas en una cadena precedida del símbolo @, escriba las comillas dos veces:

```
@""Ahoy!"" cried the captain." // "Ahoy!" cried the captain.
```

- **params**
  La palabra clave **params** permite especificar un parámetro de método que acepta un número variable de argumentos.
  *No se permiten parámetros adicionales después de la palabra clave **params**, ni varias palabras clave **params** en una misma declaración de método.*

```
// cs_params.cs
using System;
public class MyClass
{

   public static void UseParams(params int[] list)
   {
      for (int i = 0 ; i < list.Length; i++)
      {
         Console.WriteLine(list[i]);
      }
      Console.WriteLine();
   }

   public static void UseParams2(params object[] list)
   {
      for (int i = 0 ; i < list.Length; i++)
      {
         Console.WriteLine(list[i]);
      }
      Console.WriteLine();
   }

   static void Main()
   {
      UseParams(1, 2, 3);
      UseParams2(1, 'a', "test");

      // An array of objects can also be passed, as long as
      // the array type matches the method being called.
      int[] myarray = new int[3] {10,11,12};
      UseParams(myarray);
   }
}
```

Resultado:

```
1
2
3

1
a
test

10
11
12
```