

# ALGORITMOS Y PROGRAMACIÓN

---

Clase 1

Lenguaje C#

# Temario

## Lenguaje C#

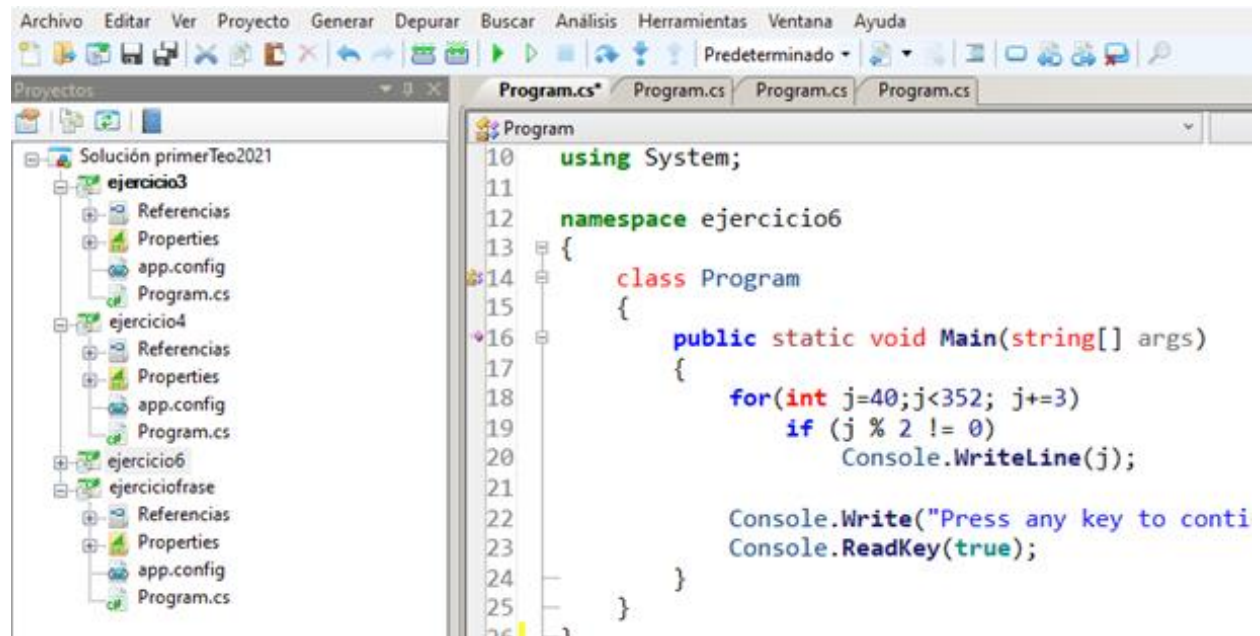
- Estructura de un programa. Espacios de nombres.
- Tipos de datos
- Conversiones
- Operadores aritméticos y lógicos
- Sentencias
- Estructuras de control

# C# - Estructura de un programa

Como vimos en la introducción del ambiente, toda aplicación que desarrollemos en C# va a estar definida como un proyecto, adentro de una solución.

Un proyecto contiene varios archivos. En particular, contiene al archivo `program.cs` que es donde se codifica nuestro programa de aplicación.

Dentro de esa plantilla hay una zona de inclusión (cláusula `using`) y uno o varios espacios de nombres (namespace).



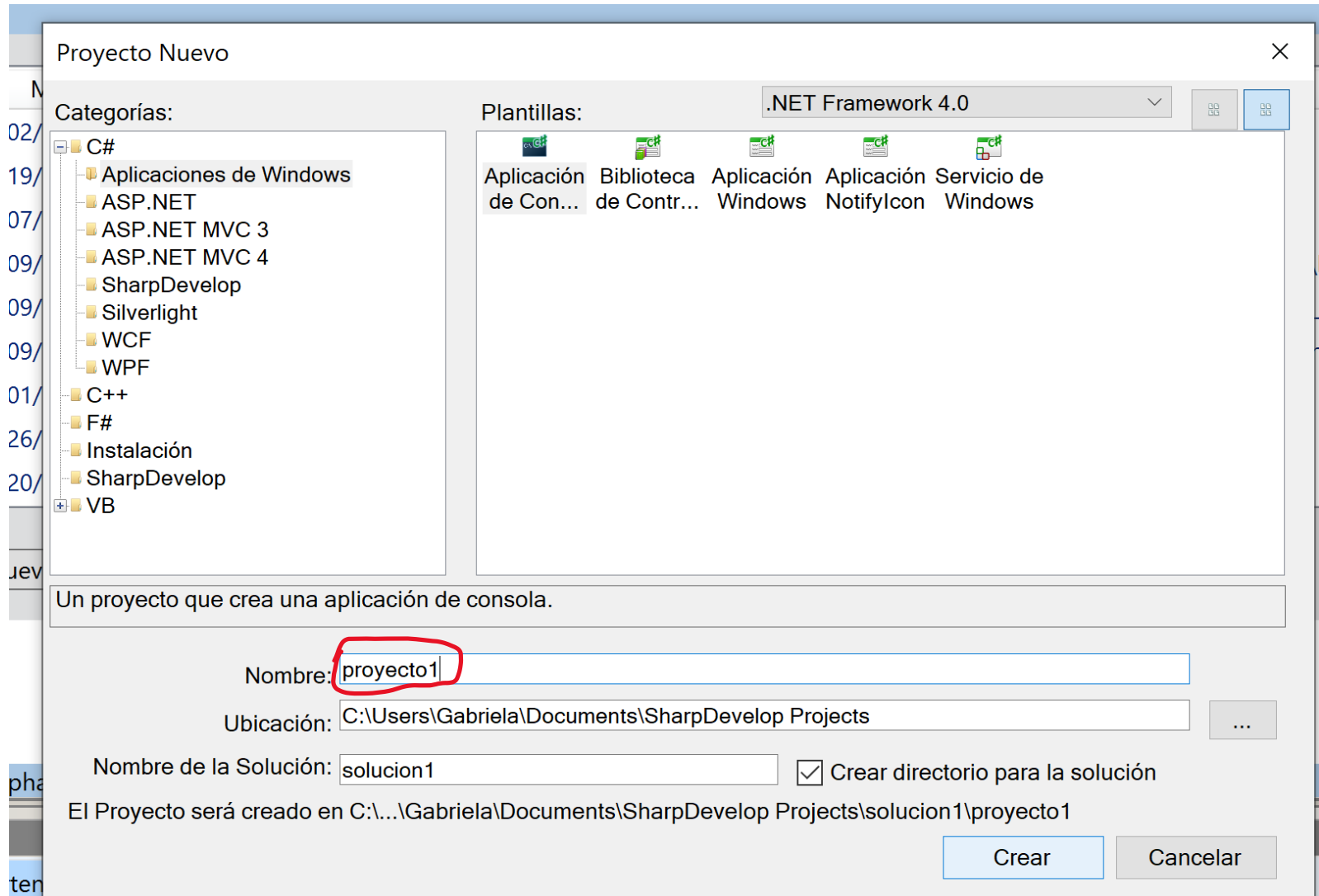
# C# - Estructura de un programa

Un *namespace* o espacio de nombres se usa como un sistema de organización interno, ya que agrupa al conjunto de elementos que componen a un proyecto: clases, estructuras, interfaces, etc..

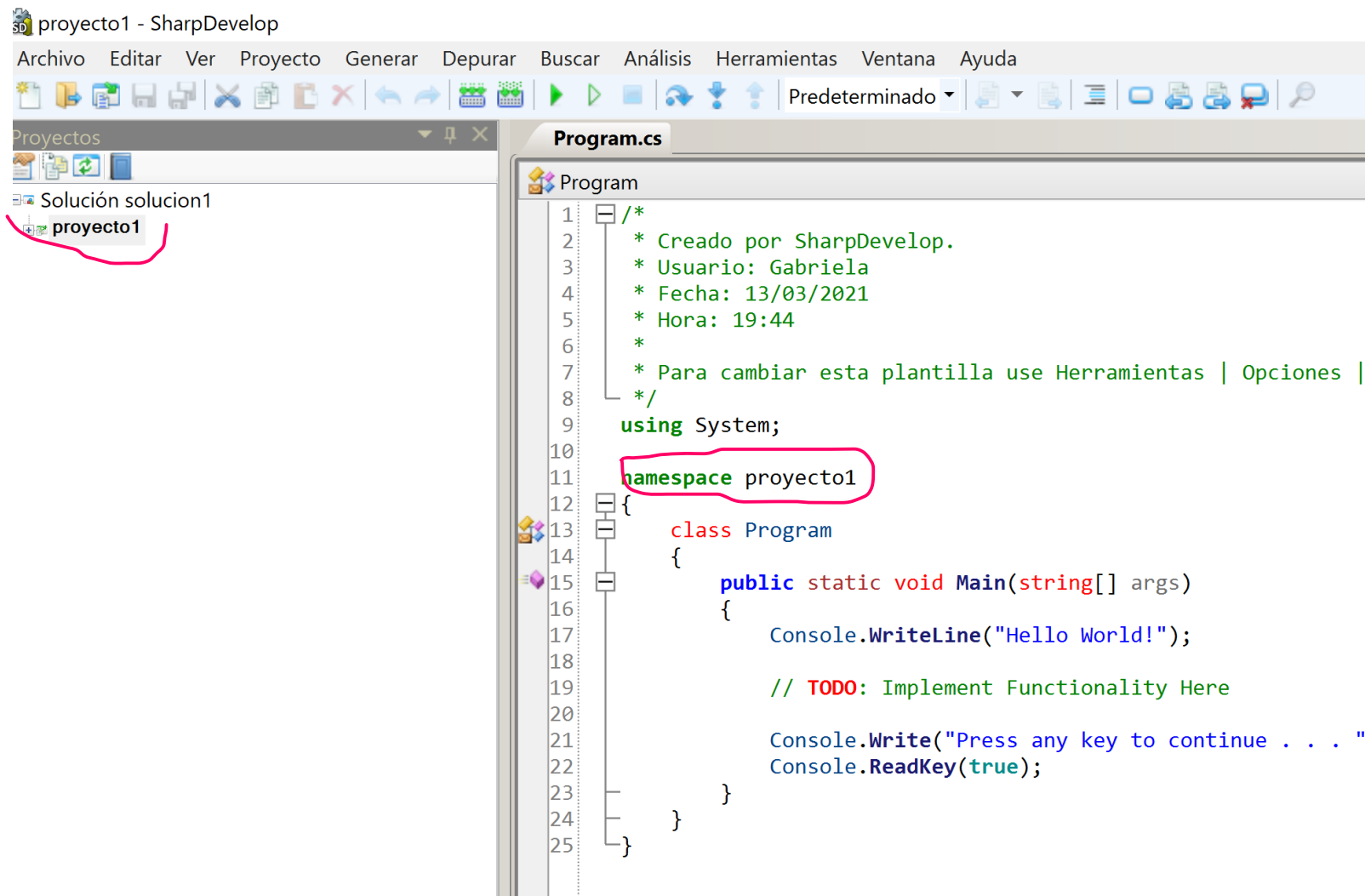
Cuando se crea un nuevo proyecto dentro de una solución, se crea un espacio de nombres nuevo con el mismo nombre del proyecto.

Todo elemento que agreguemos a dicho proyecto, estará comprendido dentro del mismo espacio de nombres.

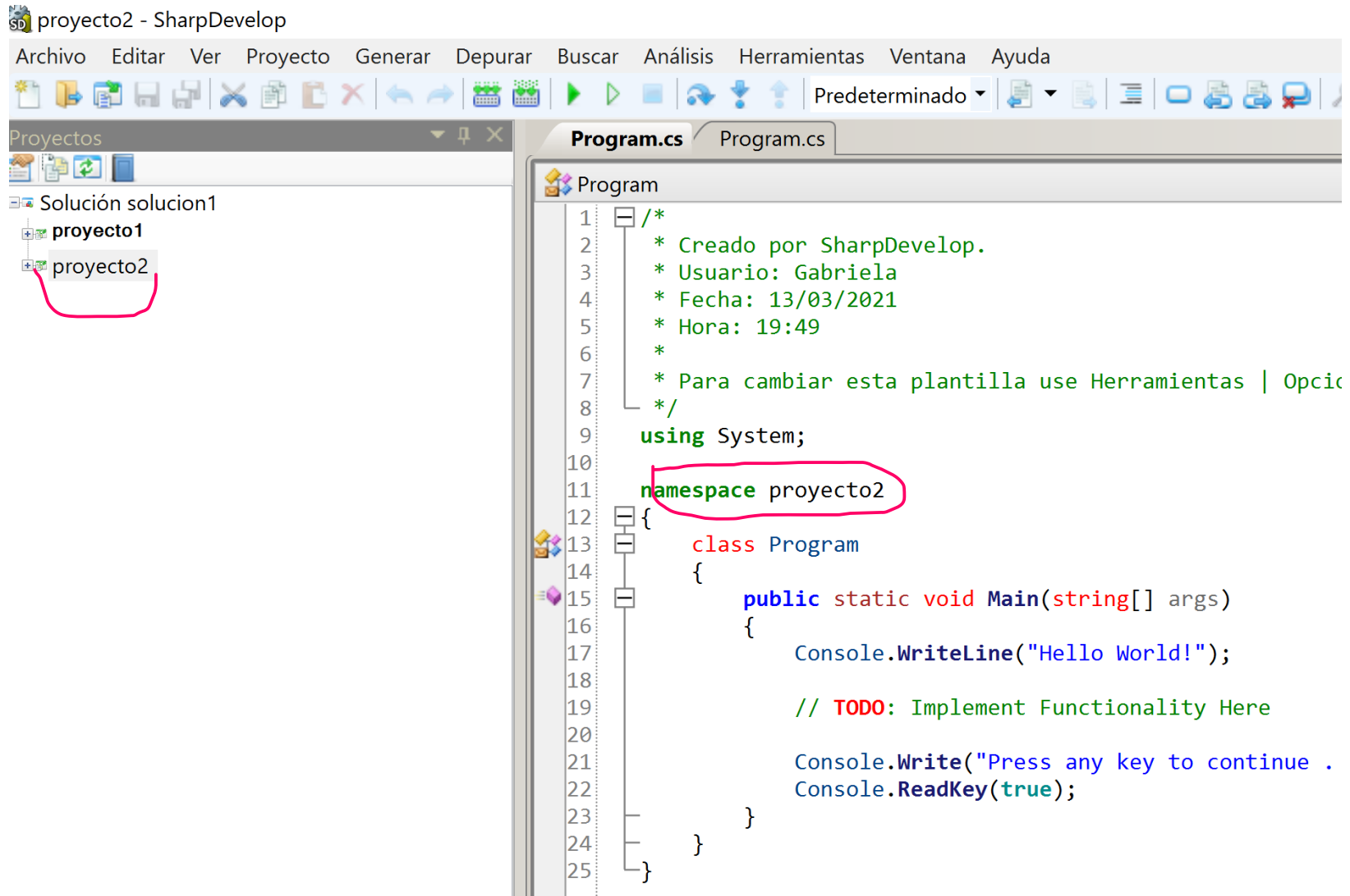
# Agregando una nueva solución y proyecto



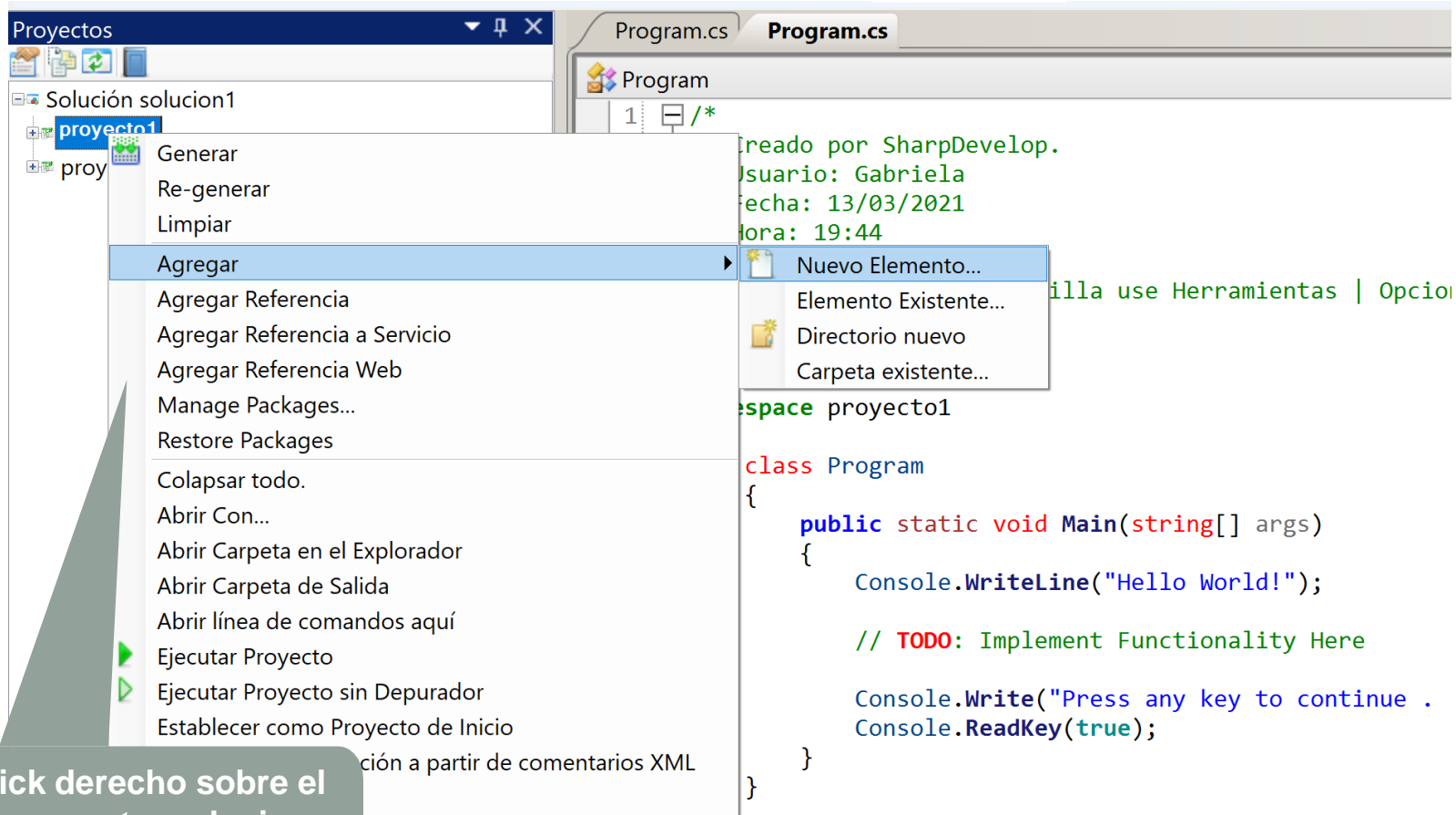
# Agregando una nueva solucion y proyecto



# Agregando un nuevo Proyecto a una solución existente



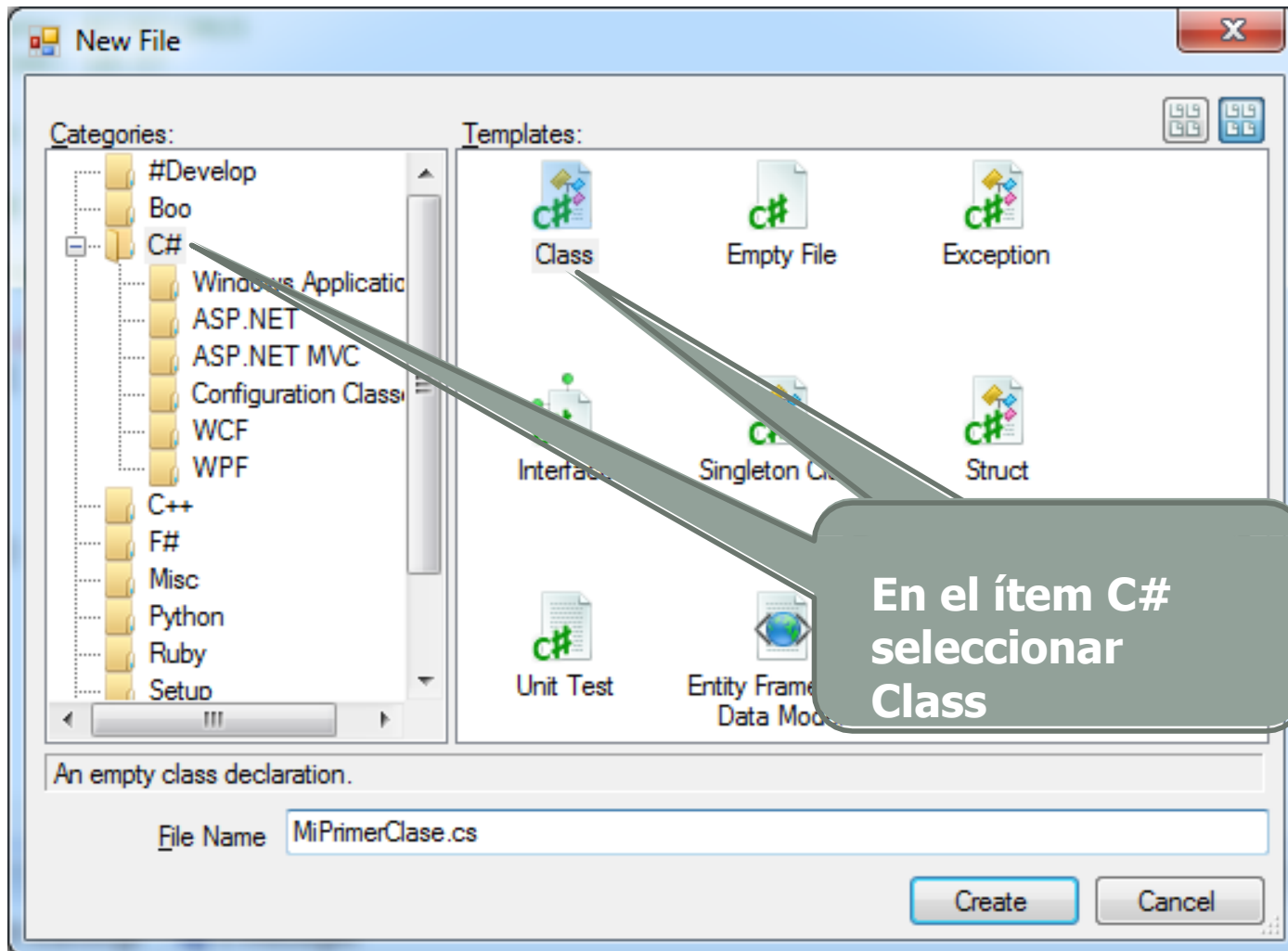
# Agregando un nuevo archivo al proyecto 1



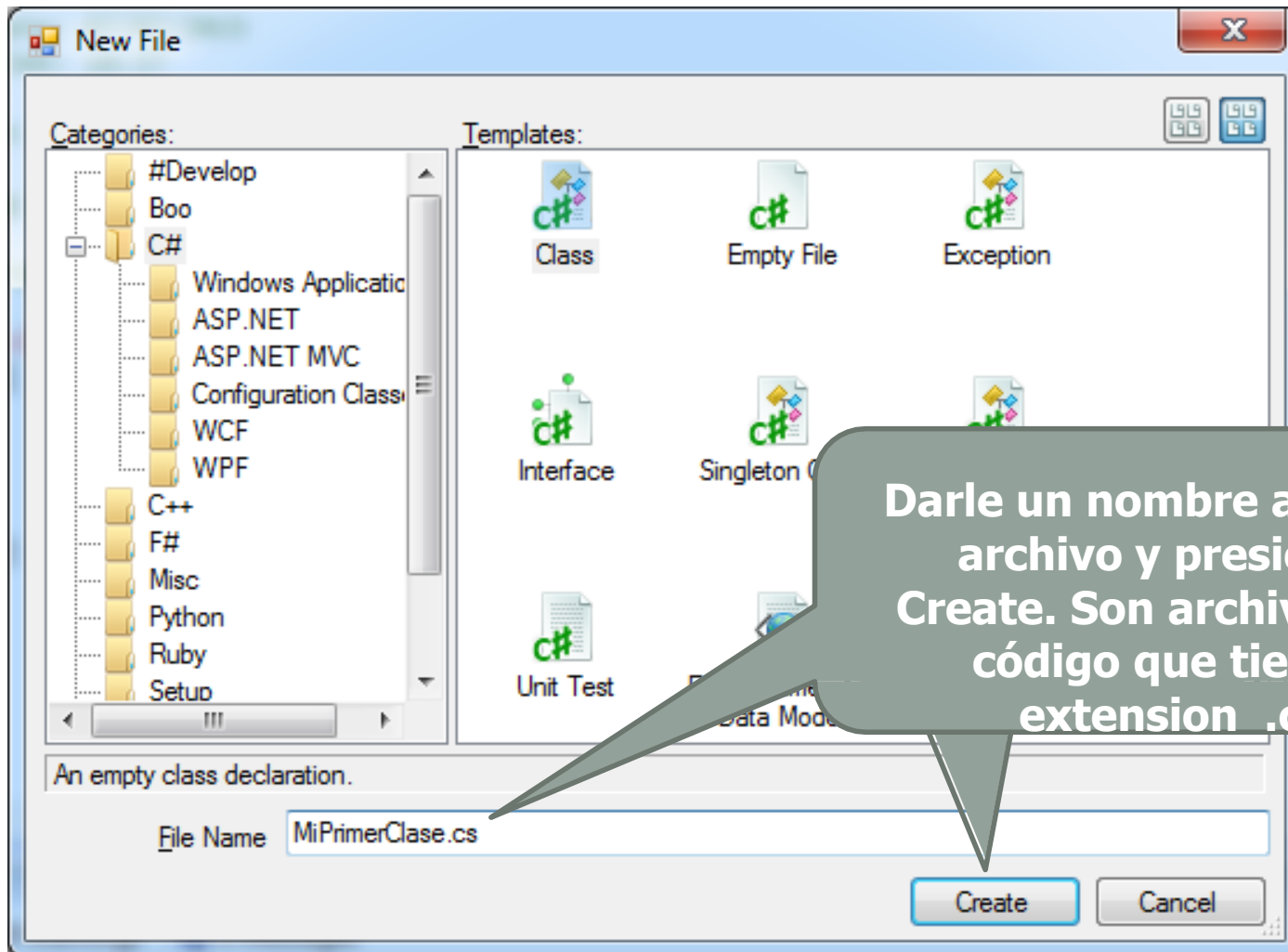
Click derecho sobre el proyecto y elegir opción Agregar Nuevo elemento



# Agregando un nuevo archivo al proyecto 1



# Agregando un nuevo archivo al proyecto 1



# Agregando un nuevo archivo al proyecto 1

The screenshot shows the Visual Studio IDE. On the left, the 'Solución solución1' tree view shows a project named 'proyecto1'. Inside 'proyecto1', the files 'Referencias', 'Properties', 'app.config', 'MiPrimerClase.cs', and 'Program.cs' are listed. 'MiPrimerClase.cs' is highlighted with a blue selection box. A callout bubble points to this file with the text: 'El archivo es agregado al árbol del proyecto1'. Another callout bubble points to the 'MiPrimerClase' namespace in the code editor with the text: 'Por defecto el archivo es creado dentro del mismo namespace del propio proyecto1'. On the right, the code editor shows the content of 'MiPrimerClase.cs'. The code is as follows:

```
1  /*
2     * Creado por SharpDevelop.
3     * Usuario: Gabriela
4     * Fecha: 13/03/2021
5     * Hora: 20:21
6     *
7     * Para cambiar esta plantilla use Her
8  */
9  using System;
10
11  namespace proyecto1
12  {
13      /// <summary>
14      /// Description of MiPrimerClase.
15      /// </summary>
16      public class MiPrimerClase
17      {
18          public MiPrimerClase()
19          {
20          }
21      }
22  }
```

# Entendiendo el uso de namespace

- A pesar de que el programa principal (Program.cs) y la clase creada (MiPrimerClase.cs) están en archivos distintos, ambas están declaradas en un mismo namespace (Proyecto1)
- .NET asocia todos los archivos que estén declarados en un mismo namespace como si fuera un único archivo.
- El uso del mismo namespace permite crear tantos archivos como se deseen permitiendo así una mejor organización del código.

# Entendiendo el uso de namespace

Para usar elementos definidos en otro namespace hay que importar dicho namespace con la cláusula **using**.

```
using System;  
using System.Collections;
```

- Esta cláusula habilita a utilizar todo lo que está incluido dentro de ese namespace.
- Su uso es opcional, si no se agrega en el programa la cláusula **using** correspondiente, en el código del cuerpo se puede utilizar el nombre completo calificando a la instrucción:

```
System.Console.WriteLine("Hola Mundo!");  
System.Collections.ArrayList a;
```

# Base Classes Library (BCL)

- .NET Framework ofrece infinidad de funcionalidades básicas y avanzadas en forma de bibliotecas de clases constituyendo la Base Classes Library
- Existen miles de clases en la BCL que se organizan de un modo coherente en espacios de nombres (namespaces)
  - Algunos incluso contienen otros espacios de nombres más especializados

# Algunos namespaces de la BCL

<b>Espacio de nombres</b>	<b>Utilidad de los tipos de datos que contiene</b>
<b>System</b>	Tipos muy frecuentemente usados, como los tipos básicos, tablas, excepciones, fechas, números aleatorios, recolector de basura, entrada/salida en consola, etc.
<b>System.Collections</b>	Colecciones de datos de uso común como pilas, colas, listas, diccionarios, etc.
<b>System.Data</b>	Manipulación de bases de datos. Forman la denominada arquitectura <b>ADO.NET</b> .
<b>System.IO</b>	Manipulación de ficheros y otros flujos de datos.
<b>System.Net</b>	Realización de comunicaciones en red.
<b>System.Reflection</b>	Acceso a los metadatos que acompañan a los módulos de código.
<b>System.Runtime.Remoting</b>	Acceso a objetos remotos.
<b>System.Security</b>	Acceso a la política de seguridad en que se basa el CLR.
<b>System.Threading</b>	Manipulación de hilos.
<b>System.Web.UI.WebControls</b>	Creación de interfaces de usuario basadas en ventanas para aplicaciones Web.
<b>System.Windows.Forms</b>	Creación de interfaces de usuario basadas en ventanas para aplicaciones estándar.
<b>System.XML</b>	Acceso a datos en formato XML.

# Cómo crear nuestro primer programa de aplicación

Pasos a seguir:

1. Abrir SharpDevelop.
2. Crear una **solución** llamada "Teoria 1".
3. Crear un **proyecto** llamado "Ejercicio 1".
4. Escribir las líneas de código de nuestro Proyecto dentro de la plantilla.



# C# - Primer programa de aplicación

```
using System;
```

```
namespace Ejercicio1
```

```
{
```

```
    class Program
```

```
    {
```

```
        public static void Main(string[] args)
```

```
        {
```

```
            Console.WriteLine("Hola mundo");  
            Console.ReadKey(true);
```

```
        }
```

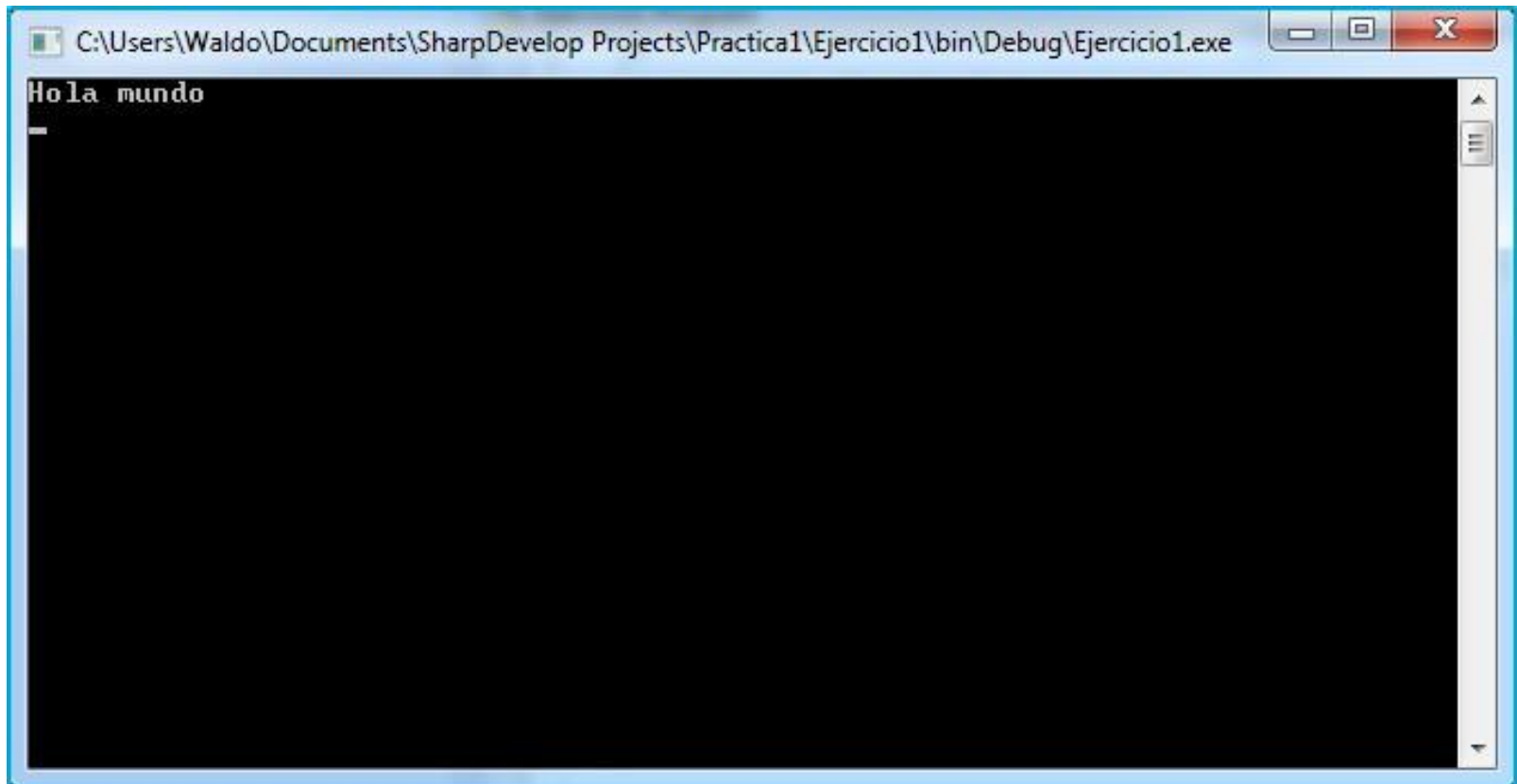
```
    }
```

```
}
```

Plantilla a ser  
reemplazada por el  
código de nuestros  
programas

# Salida obtenida

Al ejecutar el programa obtenemos la siguiente salida por consola



A screenshot of a Windows console window. The title bar at the top shows the file path: C:\Users\Waldo\Documents\SharpDevelop Projects\Practica1\Ejercicio1\bin\Debug\Ejercicio1.exe. The console area is black with white text. The first line of output is "Hola mundo", followed by a blank line. A small cursor is visible on the second line.

```
C:\Users\Waldo\Documents\SharpDevelop Projects\Practica1\Ejercicio1\bin\Debug\Ejercicio1.exe
Hola mundo
_
```

# C# - Primer programa

```
using System;
```

```
namespace Ejercicio1
```

```
{
```

```
    class Program
```

```
    {
```

```
        public static void Main(string[] args)
```

```
        {
```

```
            Console.WriteLine("Hola mundo");
```

```
            Console.ReadKey(true);
```

```
        }
```

```
    }
```

```
}
```

La instrucción  
Console.WriteLine escribe en  
la consola

Para poder ver el resultado  
de una ejecución siempre  
utilizaremos  
Console.ReadKey(true)  
como última instrucción.

# C# - Tipos de datos

C# tiene, entre otros, los siguientes tipos de datos

## Enteros

• byte	1 byte	0..255
• short	2 bytes	-32.768.....32.767
• int	4 bytes	-2.147.483.648.....2.147.483.647
• long	8 bytes	

## Puntos flotantes

• float	4 bytes	Con precisión de 7 dígitos
• double	8 bytes	Con precisión de 15-16 dígitos
• decimal	16 bytes	Con precisión de 28-29 dígitos

## Booleanos

• bool	1 byte	true.....false
--------	--------	----------------

## Caracteres

• char	<b>'H', 'e', 'l', 'o', etc.</b>
--------	---------------------------------

## Strings

• string	$(2 * n)$ bytes	Cualquier secuencia de $n$ caracteres.
----------	-----------------	--

# C# - Declaración de variables

- En C# todas las variables deben ser declaradas antes de poder ser usadas.
- Al **declarar una variable** se debe indicar primero el **tipo de dato** que almacenará y luego el **nombre** de la variable.

```
int var1;  
int VAR1;  
char var2;
```

C# es sensible a las mayúsculas y minúsculas.

**var1 y VAR1 son variables distintas**

- Las constantes se declaran con la palabra clave **const**.  

```
const int lados_cuadrado = 4;  
const char primera_letra = 'A';  
const double pi = 3.14159;
```
- En C# toda sentencia debe finalizar con punto y coma.

# Conversiones de tipo numéricas implícitas

Una variable de tipo	puede ser asignada a otra de tipo
sbyte	short, int, long, float, double, decimal
byte	short, ushort, int, uint, long, ulong, float, double, decimal
short	int, long, float, double, decimal
ushort	int, uint, long, ulong, float, double, decimal
int	long, float, double, decimal
uint	long, ulong, float, double, decimal
long	float, double, decimal
ulong	float, double, decimal
char	ushort, int, uint, long, ulong, float, double, decimal
float	Double

# Conversiones de tipo numéricas implícitas

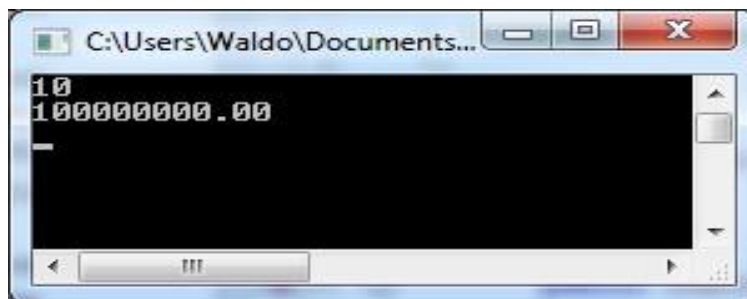
Ejemplo. Pruebe el siguiente código en máquina y observe qué ocurre.

```
byte b = 10;  
int i = b;  
Console.WriteLine(i);
```

Conversión implícita de  
byte a int

```
i = 100000016;  
float f = i;  
Console.WriteLine(f.ToString("#.00"))
```

Conversión implícita de  
int a float.  
Se pierde precisión,  
pero no magnitud



# Conversiones de tipo numéricas explícitas

- Se utiliza el operador de **cast**.
- La operación de casting consiste en anteponer entre paréntesis el tipo al cual queremos convertir el resultado de una variable o expresión, y aceptamos una eventual pérdida de información.

```
int i = 100;
```

```
short s = (short) i;
```

→ s=100

```
double d = 13.78;
```

```
i = (int) d;
```

→ i=13



# Operadores aritméticos

Operador	Operación
+	Suma
-	Resta
*	Multiplicación
/	División
%	Resto

```
int a = 100, b = 4, c=3;  
double d = 13.78;  
double r = (a+b) / (c*d);
```

# Operadores de asignación

Operador	Operación
++	Incremento
--	Decremento
=	Asignación
*=	Multiplicación seguida de asignación
/=	División seguida de asignación
%=	Resto seguido de asignación
+=	Suma seguido de asignación
-=	Resta seguido de asignación

```
int a = 100, b = 4, c = 3;
```

```
c+= (a-b);
```

```
//c = c + (a-b);
```

```
b*= a;
```

```
//b = b * a;
```

```
a++;
```

```
//a = a + 1;
```

```
int i = 3;
```

```
Console.WriteLine(i);    // output: 3
```

```
Console.WriteLine(i++);  // output: 3
```

```
Console.WriteLine(i);    // output: 4
```

```
double a = 1.5;
```

```
Console.WriteLine(a);    // output: 1.5
```

```
Console.WriteLine(++a);  // output: 2.5
```

```
Console.WriteLine(a);    // output: 2.5
```

# C# - Bloque de sentencias

- Un bloque es una lista de cero o más sentencias encerradas entre llaves “{“ “}”
- El alcance o ámbito de una variable declarada en el bloque es el propio bloque.

{

**Las variables están definidas cada una en un bloque independiente. No hay error.**

```
{  
    int j=2  
    int i=1;  
    Console.WriteLine(i);  
}  
Console.WriteLine(i);
```

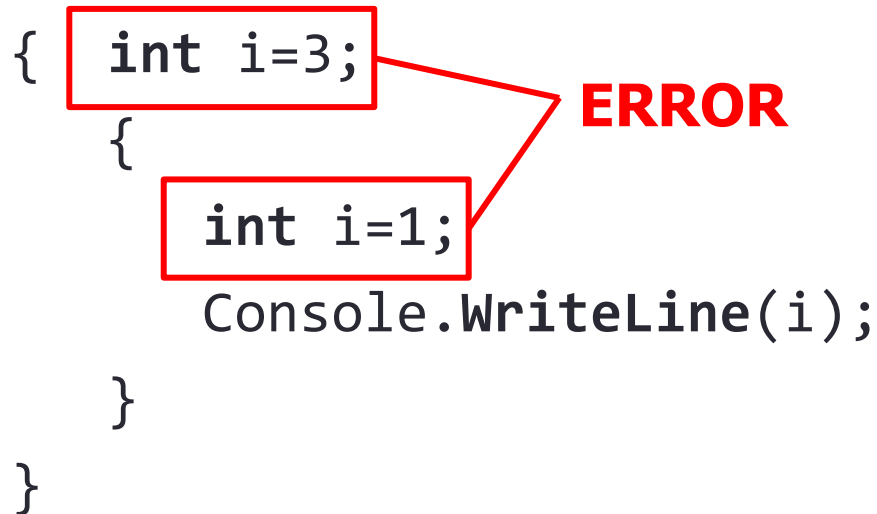
}

```
{ int j=0; }
```

Esta instrucción produce error porque la variable i no está definida en este ámbito

# C# - Bloque de sentencias

```
{ int i=3;
  {
    int i=1;
    Console.WriteLine(i);
  }
}
```



**ERROR**

Los bloques pueden anidarse pero, si dentro de un bloque interno definimos una variable con el mismo nombre que otra definida en un bloque externo, se produce un **ERROR**, ya que no se podrá determinar a cuál de las dos se estará haciendo referencia cada vez que se utilice su nombre en el bloque interno.

# Puesta en común: Ejercicio 3 - Práctica 1

**Escriba un programa que calcule la suma de dos números reales introducidos por teclado y muestre el resultado en la consola.**

- ¿Qué tipos de datos vamos a usar?
- Utilizaremos `Console.ReadLine()`; que lee los caracteres introducidos por el usuario hasta que aprieta ENTER.
- `Console.ReadLine()`; devuelve un string.

**Ayuda: utilice `Double.Parse(st)` para obtener el valor real del string `st`.**

```
using System;
namespace ejercicio3
{
    class Program
    {
        public static void Main(string[] args)
        {
            double suma, num1, num2;

            Console.WriteLine("Ingrese primer valor real (usar coma para los decimales)");
            num1=double.Parse(Console.ReadLine());

            Console.WriteLine("Ingrese segundo valor real (usar coma para los decimales)");
            num2=double.Parse(Console.ReadLine());

            suma=num1 + num2;
            Console.WriteLine("La suma de ambos valores es: {0}", suma);

            Console.Write("Press any key to continue . . . ");
            Console.ReadKey(true);
        }
    }
}
```

# Estructuras de control condicional - IF

- Los bloques verdadero o falso pueden tener más de una instrucción.
- Si un bloque cuenta con una sola instrucción las llaves pueden omitirse.
- El bloque **else** es opcional.

```
if ( <condición> )  
{  
    <bloque V, código si la condición es verdadera>;  
}  
else  
{  
    <bloque F, código si la condición es falsa>;  
}
```

# Estructuras de control - IF

Ejemplos:

```
int a = 3;
if ( a > 0 )    /*decisión con ambas ramas*/
{
    Console.WriteLine("El valor es positivo");
    a = 1;
}
else
{
    Console.WriteLine("El valor es negativo");
    a = -1;}

int b = 3;    /*decisión solamente con rama verdadera*/
if ( b == 0 )
    Console.WriteLine("El valor es nulo");
```



# Estructuras de control - IF

Los IFs pueden estar anidados

```
int a = -1;  
if(a>0)  
    if(a>1)  
        Console.WriteLine(">1");  
else  
    Console.WriteLine("a es negativo");
```

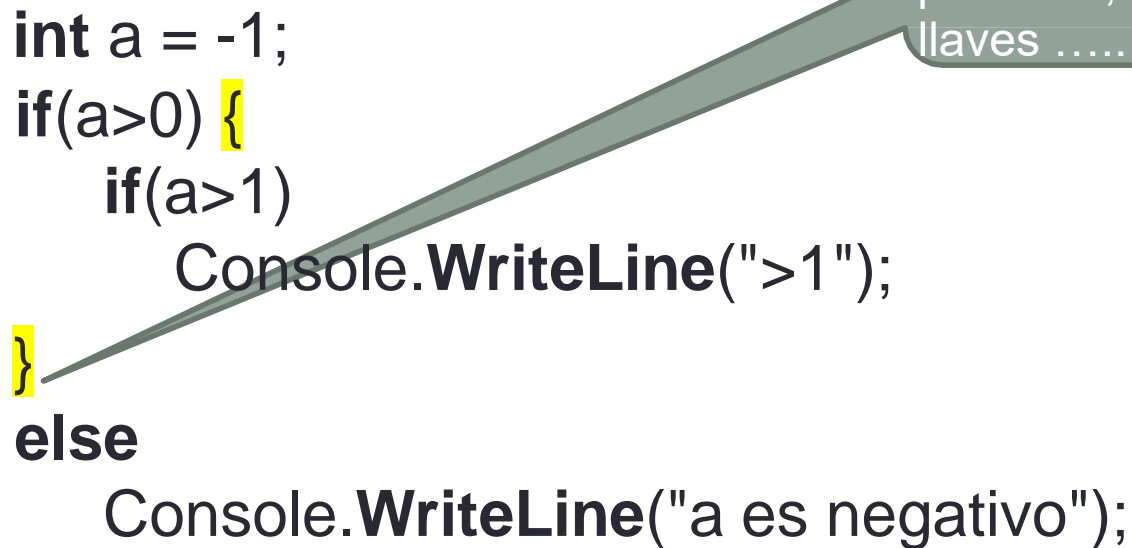
¿Qué imprime si a vale -1?  
Comprobar en máquina.

**Recordar que en C# no existe el concepto de indentación. Si no se usan bloques de llaves, el ELSE se asocia al último IF**

# Estructuras de control - IF

Los IFs pueden ser anidados

```
int a = -1;  
if(a>0) {  
    if(a>1)  
        Console.WriteLine(">1");  
}  
else  
    Console.WriteLine("a es negativo");
```



The diagram illustrates a nested if-else structure. A large grey arrow on the left points from the opening brace of the first if statement down to the else keyword, indicating the flow of execution. A smaller grey arrow points from the closing brace of the first if statement to the else keyword, indicating the end of the first if block. A callout box points to the closing brace of the first if statement.

Para lograr que el ELSE se asocie al primer IF, usamos llaves .....

# Puesta en común:

**Dados dos números enteros ingresados por el usuario determinar cuál de ellos es el mayor.**

Repasemos:

Qué tipos de datos vamos a ingresar?

Necesitamos alguna conversión?

Qué estructura de control vamos a utilizar para determinar cuál es el mayor valor ingresado?

Cómo mostramos el resultado?

```
Console.WriteLine("Ingrese un número");  
int num1 = int.Parse(Console.ReadLine());  
Console.WriteLine("Ingrese otro número");  
int num2 = int.Parse(Console.ReadLine());
```

```
//Determinación del máximo
```

```
Console.WriteLine("El número más grande es");  
if(num1 > num2)  
    Console.WriteLine(num1);  
else  
    { if (num1 < num 2)  
        Console.WriteLine(num2);  
    else  
        Console.WriteLine("son iguales");  
    }  
Console.Readkey(true);
```

# Estructuras de control condicional – SWITCH

- Es una estructura condicional que simplifica el uso de IFs anidados.
- Se aplica sobre una variable que puede tomar varios valores distintos.

```
switch ( <testVar> ){  
    case <valor1>:  
        <código si <testVar> == <valor1> >  
        break;  
  
    ...  
    case <valorN>:  
        <código si <testVar> == <valorN> >  
        break;  
    default:  
        <código si no entró por ningún case>  
        break;  
}
```

# Estructuras de control – SWITCH

## Ejemplo

```
int a = 3;  
switch (a) {  
    case 1:  
        Console.WriteLine("uno");  
        break;  
    case 2:  
        Console.WriteLine("dos");  
        break;  
    default:  
        Console.WriteLine("otro");  
        break;  
}
```

Pueden haber tantas sentencias CASE como se necesiten.

El uso de la condición default es opcional

# Puesta en común Ejercicio 4 - TP1

**Escriba un programa de aplicación que solicite al usuario que ingrese un número de mes (1 a 12) e imprima el nombre del mes correspondiente. Si el valor ingresado no está en ese rango debe imprimir “Mes inválido”. Utilice la sentencia switch.**

- ¿Tipo de dato de mi variable evaluada en el switch?
- ¿Es necesaria la conversión string al tipo de mi variable?
- ¿Cuántas opciones voy a definir?

```
public static void Main(string[] args)
{
    int mes;

    Console.WriteLine("ingrese un valor entre 1 y 12 referido a un mes del año");
    mes=int.Parse(Console.ReadLine());

    switch (mes) {
        case 1:
            Console.WriteLine("enero"); break;
        case 2:
            Console.WriteLine("febrero");break;
        case 3:
            Console.WriteLine("marzo"); break;
        case 4:
            Console.WriteLine("abril"); break;
        case 5:
            Console.WriteLine("mayo");break;
        case 6:
            Console.WriteLine("junio");break;
        /* continúa en la diapositiva sgte*/
    }
```



```
case 7:
    Console.WriteLine("julio"); break;
case 8:
    Console.WriteLine("agosto"); break;
case 9:
    Console.WriteLine("septiembre"); break;
case 10:
    Console.WriteLine("octubre"); break;
case 11:
    Console.WriteLine("noviembre"); break;
case 12:
    Console.WriteLine("diciembre"); break;
default:
    Console.WriteLine("error de tipeo"); break;
}
```

```
Console.Write("Press any key to continue . . . ");
Console.ReadKey(true);
}
```

# Estructuras de control repetitivas - WHILE

- El bloque del while puede tener más de una instrucción.
- Si solo contiene una sentencia, se omiten las llaves.

```
while ( <condicion> )  
{  
    <Código mientras condicion sea verdadera>;  
}
```

Ejemplo

**¿Qué  
imprime?**

```
int a = 1;  
while ( a <= 5 ) {  
    Console.WriteLine(a++);
```

```
}
```

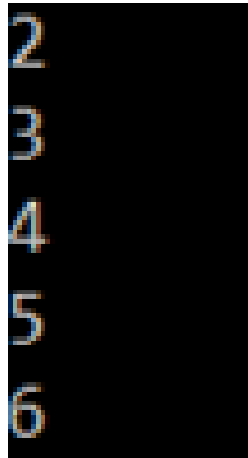
**Se  
pueden  
omitir**

```
1  
2  
3  
4  
5
```

Ejemplo

**¿Qué  
imprime?**

```
int a = 1;  
while ( a <= 5 )  
    Console.WriteLine(++a);
```



2  
3  
4  
5  
6

# Estructuras de control repetitivas – DO WHILE

- A diferencia del WHILE, el DO WHILE ejecuta el código de su bloque al menos una vez. Primero ejecuta el bloque de instrucciones y luego evalúa la condición.

```
do {  
    <Código hasta que la condición sea falsa>;  
}  
while ( <condición> );
```

# Estructuras de control – DO WHILE

Ejemplo

**¿Qué  
imprime?**

```
int a = 1;  
do {  
    Console.WriteLine("Hola");  
}  
while ( a++ != 1 );
```

Un solo Hola y  
corta la ejecución.

# Puesta en común.

**Solicitar al usuario que ingrese una sucesión de letras  
(termina al ingresar un punto).**

**Luego informar la cantidad de vocales leídas.**

Qué estructura de control nos conviene utilizar para  
ingresar las letras?

```
int voc=0;
Console.WriteLine("Ingrese una letra o punto
                    para terminar");
string letra = Console.ReadLine();
while(letra != ".")
{
    if ((letra=="a")|| (letra=="e")|| (letra=="i")|| (letra=="o")||
        (letra=="u") )
        voc++;

    Console.WriteLine("Ingrese otra letra o
                    punto para terminar");
    letra = Console.ReadLine();
}
Console.WriteLine("La cantidad de vocales es: " + voc);
Console.Readkey(true);
```



# Estructuras de control repetitivas - FOR

- El FOR tiene tres partes separadas por ";" :
  - Valor inicial de la variable iteradora
  - Condición de fin
  - Modificación de la variable iteradora

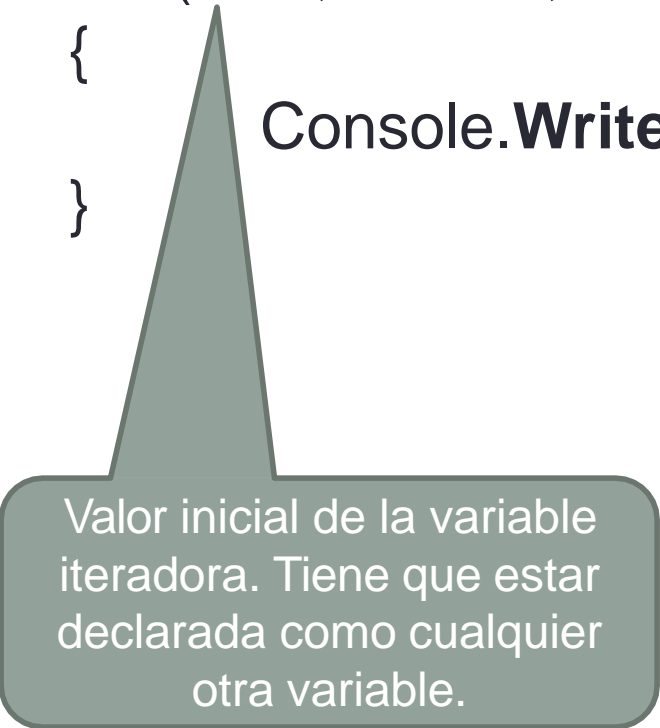
```
for ( <inicialización> ; <condición> ; <modificación> )  
{  
    <Código mientras condición sea verdadera>;  
}
```

El bloque del for puede tener más de una instrucción. Si tiene una sola se pueden omitir las llaves de apertura y cierre.

# Estructuras de control - FOR

## Ejemplo

```
int i;  
for (i = 1; i <= 10; i++)  
{  
    Console.WriteLine(i);  
}
```

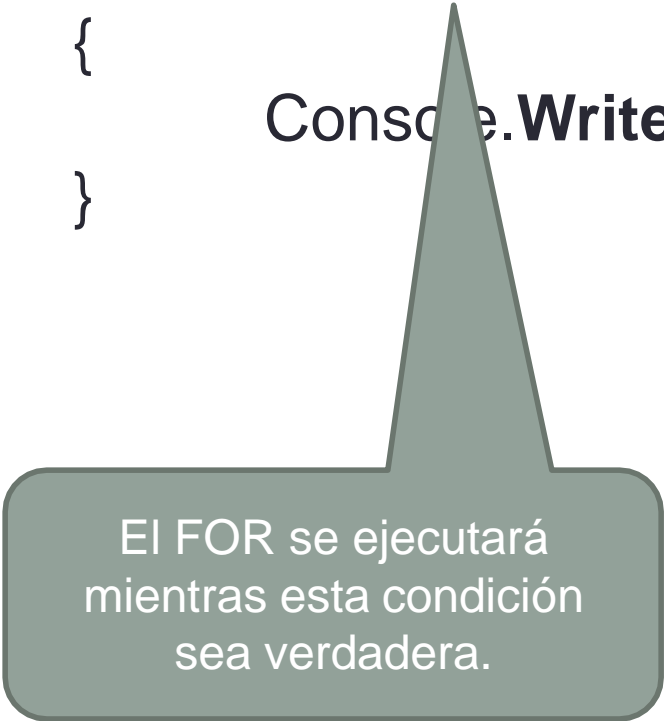


Valor inicial de la variable iteradora. Tiene que estar declarada como cualquier otra variable.

# Estructuras de control - FOR

## Ejemplo

```
int i;  
for (i = 1; i <= 10; i++)  
{  
    Console.WriteLine(i);  
}
```

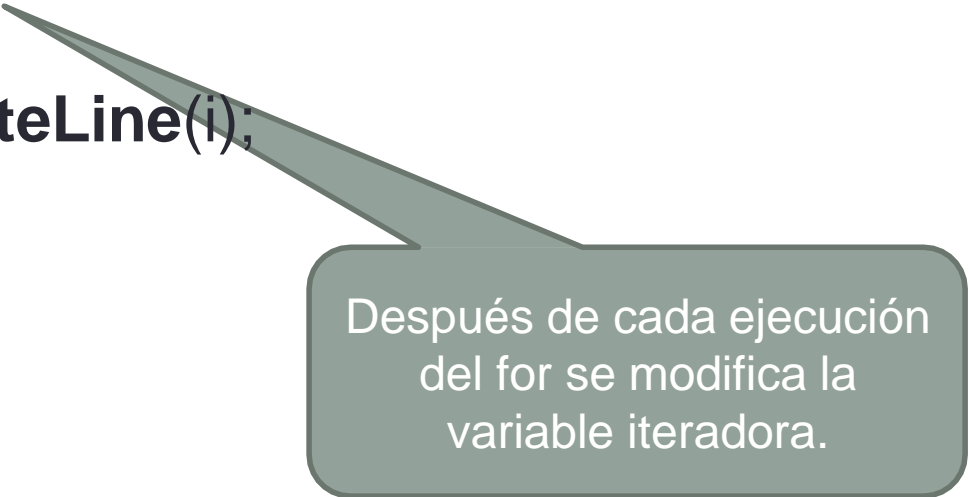


El FOR se ejecutará  
mientras esta condición  
sea verdadera.

# Estructuras de control - FOR

## Ejemplo

```
int i;  
for (i = 1; i <= 10; i++)  
{  
    Console.WriteLine(i);  
}
```



Después de cada ejecución del for se modifica la variable iteradora.

Imprime los números del 1 al 10.

# Estructuras de control - FOR

## Ejemplos

```
for (i = 10; i >= 0; i--)  
    Console.WriteLine(i);
```

De 10 a 0,  
decrementando en uno

```
for (i = 1; i <= 10; i += 2)  
    Console.WriteLine(i);
```

De 1 a 10,  
incrementando de a dos

```
for (i = 100; i > 0; i /= 2)  
    Console.WriteLine(i);
```

De 100 a 1, dividiendo  
por dos.

# Puesta en común

**Solicitar al usuario que ingrese por teclado un número n y calcular la sumatoria desde 1 hasta n.**

```
Console.WriteLine("Ingrese un número");  
int n = int.Parse(Console.ReadLine());
```

```
int suma = 0;  
for(int i = 1; i <= n; i++)  
    suma += i;
```

```
Console.WriteLine("El resultado es " + suma);
```

```
Console.Readkey(true);
```

# Operadores booleanos

Operador	Operación
<	Menor que
>	Mayor que
<=	Menor o igual que
>=	Mayor o igual que
!=	Distinto de
==	Igual a

```
if (a >= b) { .... }  
if (a != b) { .... }  
if (a == b) { .... }
```

# Operadores lógicos

Operador	Operación	Observación:
&	And	Evalúa siempre ambos operandos
	Or	Evalúa siempre ambos operandos
!	NOT	Negación
^	XOR	Evalúa siempre ambos operandos
&&	And en cortocircuito	Evalúa el operando derecho solo si es necesario
	Or en cortocircuito	Evalúa el operando derecho solo si es necesario

```
if ((a >= b) & (b < 0)) { .... }  
if ((a != 3) | (a != 4)) { .... }  
if ( ! (a == b) && (b > 9)) { .... }
```

And de circuito corto, si la primer condición es falsa, la segunda condición NO se evalúa (El resultado del And ya se sabe que es false)



# Interrupción de bucles

- Interrupción de bucles. Pueden ser utilizados para cualquier estructura de bucle: FOR, WHILE y DO WHILE.
  - **break** – El bucle termina inmediatamente
  - **continue** – Termina el ciclo corriente inmediatamente (la ejecución continua con el próximo ciclo)
  - **goto** – Permite saltar fuera del bucle (no se recomienda)
  - **return** – Salta fuera del loop y de la función que lo contiene

# Interrupción de bucles

Interrupción de bucles. Ejemplo:

```
int i;  
for(i=1; i<=10; i++) {  
    if (i==5)  
        break;  
    if (i % 2 == 0)  
        continue ;  
    Console.WriteLine(i);  
}
```

¿Qué imprime?  
Probar en máquina

¿imprime 1,3 y  
corta el proceso  
con el break.

# Puesta en común Ejercicio 6 -TP 1

Escriba un programa de aplicación que imprima en la consola todos los números impares del intervalo [40,352] que además sean múltiplos de 3.

Para verificar que los números sean impares y múltiplos de 3  
¿Qué operación aritmética deberíamos usar?

¿Qué estructura de control deberíamos usar?

Existen diversas maneras de programarlo.

```
using System;
```

```
namespace ejercicio6
```

```
{
```

```
    class Program
```

```
    {
```

```
        public static void Main(string[] args)
```

```
        {
```

```
            for(int j=40;j<352; j++)
```

```
                if ((j % 2 != 0) && (j % 3 == 0) )
```

```
                    Console.WriteLine(j);
```

```
                Console.Write("Press any key to continue . . . ");
```

```
                Console.ReadKey(true);
```

```
            }
```

```
        }
```

```
    }
```

# Muchas gracias

