

.NET FRAMEWORK y C# Clase 1

¿Qué es un Framework?

- Marco de trabajo
- Puede estar compuesto por:
 - Librerías de Clases
 - Documentación
 - Ayuda
 - Ejemplos
 - Tutoriales
 - Etc.

.NET Framework

- **Plataforma .NET** (en español)
- El **.NET Framework** constituye las bases sobre las que, tanto **aplicaciones** como servicios, son **construidas** y **ejecutadas**.
- .NET Es una plataforma **multilenguaje**: C#, C++, VB.NET, F#, Pascal, Python, Rubi, etc.

.NET Framework

- Permite el desarrollo de todo tipo de funcionalidades:
 - Programas de consola
 - Servicios Windows
 - Aplicaciones para dispositivos móviles
 - Desarrollos de escritorio o para Internet
 - Aplicaciones distribuidas
 - Aplicaciones concurrentes/paralelas
 - Etc.

.NET Framework

.NET Framework

Common
Language
Runtime

Base Class
Library

CLR – Common Language Runtime

- El CLR es el **motor de ejecución** (runtime) del .NET Framework.
- Ofrece **servicios automáticos** tales como:
 - Administración de la memoria
 - Seguridad del código
 - Administra la conversión de tipos
 - Inicialización de variables
 - Control de overflows
 - Permite que convivan diferentes versiones de una misma dll, sin que se generen conflictos

CLR – Common Language Runtime

- El CLR define **un entorno de ejecución** virtual independiente en el que trabajan las aplicaciones escritas con cualquier lenguaje .NET.
- En teoría cualquier aplicación escrita para .NET puede ejecutarse **en cualquier tipo de arquitectura de hardware**.
- Microsoft dispone de implementación de .NET para Windows de 32 bits, Windows de 64 bits e incluso para Windows Phone

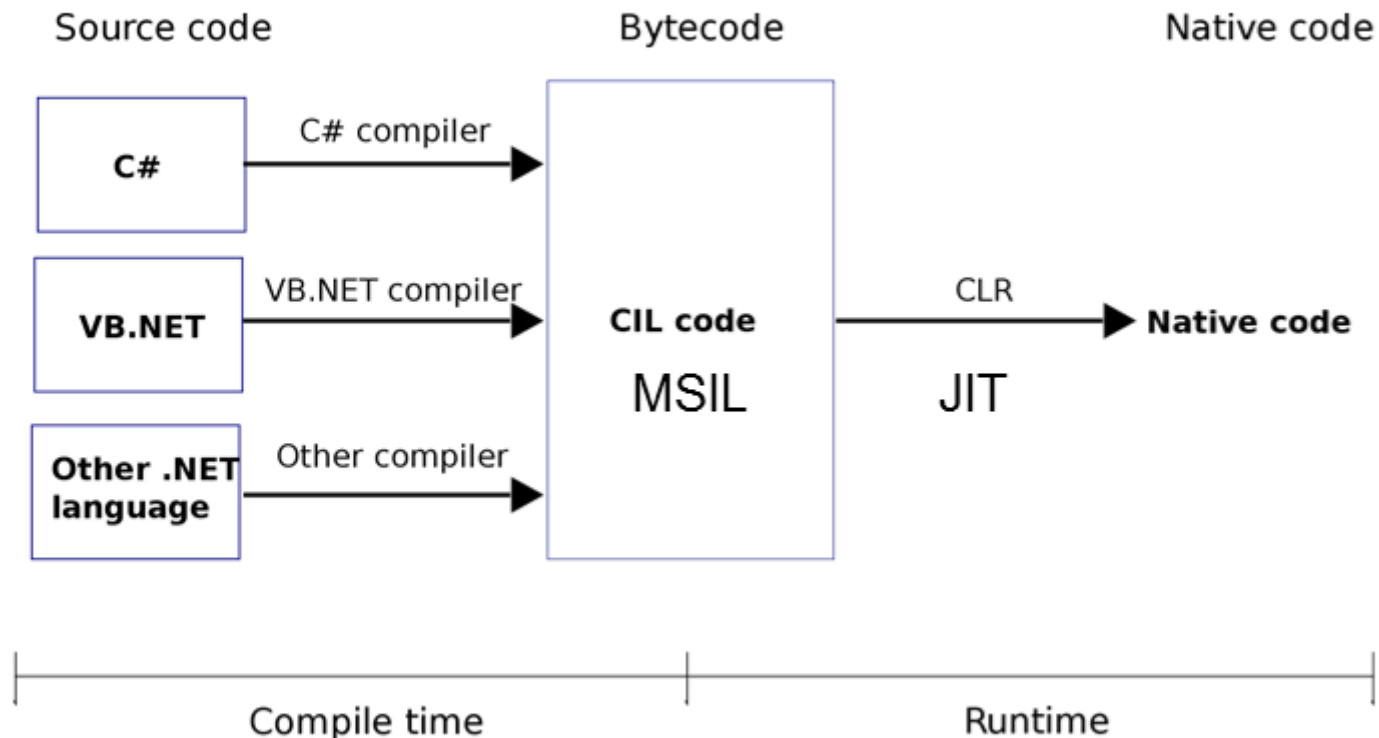
Microsoft Intermediate Language (MSIL)

- MSIL (Microsoft *Intermediate Language*) es un lenguaje parecido al código **ensamblador** pero de más **alto nivel**, creado para un **hipotético procesador virtual** que no está atado a una arquitectura determinada.
- Cuando se compila una aplicación escrita en un lenguaje .NET (VB, C# u otro de los soportados), **el compilador genera código MSIL.**

Microsoft Intermediate Language (MSIL)

- El código MSIL se convierten en código nativo justo antes de ejecutarse.
- Para convertir MSIL a código nativo, se utilizan compiladores llamados **“Just In Time” JIT**.
- **JIT** es un **compilador bajo demanda**. De manera transparente el código escrito en lenguaje intermedio se traduce al lenguaje nativo del procesador físico que va a ejecutar el código.

Microsoft Intermediate Language (MSIL)



http://en.wikipedia.org/wiki/Common_Language_Runtime

Microsoft Intermediate Language (MSIL)

- Los archivos ejecutables están conformados por
 - MSIL
 - Datos Adicionales (Metadata)
- El MSIL es independiente del lenguaje en el que se desarrolla
- Los metadatos junto con el MSIL permiten crear códigos autodescriptivos -Lenguaje de definición de interfaces (IDL) innecesario-.

Common Language Specification (CLS)

- Para conseguir la interoperabilidad entre lenguajes es necesario disponer de un conjunto de características que todos los lenguajes deben incorporar.
- Entre las cuestiones que regula CLS se encuentran la nomenclatura, la forma de definir los miembros de los objetos, los metadatos de las aplicaciones y el conjunto de tipos de datos que debe ser soportado

Common Type System (CTS)

- Define un conjunto común de “tipos” **orientado a objetos**
- Todo lenguaje de programación debe implementar los tipos definidos por el CTS
- Todo tipo hereda directa o indirectamente del tipo **System.Object**
- Tipos **VALOR** y **REFERENCIA**

Base Classes Library (BCL)

- .NET Framework ofrece infinidad de funcionalidades básicas y avanzadas en forma de **bibliotecas de clases** constituyendo la *Base Classes Library*
- La BCL forma **parte integral de la plataforma** .NET, por lo tanto no se trata de añadidos que se deban obtener o adquirir aparte.

Base Class Library (BCL)

- Existen miles de clases en la BCL que se organizan de un modo coherente en **espacios de nombres (namespaces)**
- Un espacio de nombres es un identificador que permite **organizar** de modo estanco las **clases** que estén contenidas en él, así como **otros espacios de nombres**.

Base Class Library (BCL)

- Por ejemplo, todo lo que tiene que ver con el manejo de estructuras de datos XML se encuentra bajo el espacio de nombres ***System.Xml***.
- La funcionalidad fundamental para crear aplicaciones Web está en el espacio de nombres ***System.Web***. Éste a su vez contiene otros espacios de nombres más especializados

Algunos namespaces de la BCL

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

El lenguaje C#

- C# es un lenguaje de propósito general diseñado por Microsoft **específicamente** para su plataforma .NET.
- C# carece de elementos heredados innecesarios en .NET. Por esta razón, se suele decir que C# es el **lenguaje nativo de .NET**

Características de C#

- Sencillez
 - No necesita archivos adicionales como por ejemplo los de cabecera
 - No incluye elementos poco útiles como macros, herencia múltiple
- Modernidad
 - Incluye elementos que se han demostrado útiles como los tipos bool, string, **decimal** (128 bits) la instrucción **foreach**, etc.

Características de C#

- Orientación a objetos
 - En relación a otros lenguajes como C++ o Delphi, la orientación a objetos en C# es más **pura**, no admiten **ni funciones ni variables globales**
 - A diferencia de C++ y al igual que Java, C# sólo admite **herencia simple** de clases
 - A diferencia de Java, en C# todos **los métodos son sellados por defecto**. Los métodos redefinibles hay que marcarlos con el modificador **virtual** (como en C++), lo que permite evitar errores derivados de redefiniciones accidentales

Características de C#

- Orientación a componentes
 - C# permite definir cómodamente **propiedades, eventos y atributos**
- Seguridad de tipos
 - Incluye mecanismos para asegurar el acceso correcto a los tipos de datos
 - Sólo se admiten **conversiones entre tipos compatibles**
 - No se pueden usar **variables no inicializadas**
 - Se controla los **acceso a los elementos de un array**
 - Se puede controlar la producción de **overflow**

Características de C#

- Sistema de tipos unificado:
 - A diferencia de C++, en C# todos los tipos de datos que se definan siempre derivarán, de **una clase base común** llamada **System.Object**,
 - A diferencia de Java, en C# esto también es aplicable a los tipos de datos básicos (**boxing** y **unboxing**)

Características de C#

- **Extensibilidad de operadores:** C# permite redefinir operadores incluidos los de conversión de tipos (implícitas y explícitas).
- **Compatible:** Desde código C# se puede acceder a código nativo escrito como funciones sueltas no orientadas a objetos tales como las DLLs de la API Win32. También se puede acceder a objetos COM y controles ActiveX

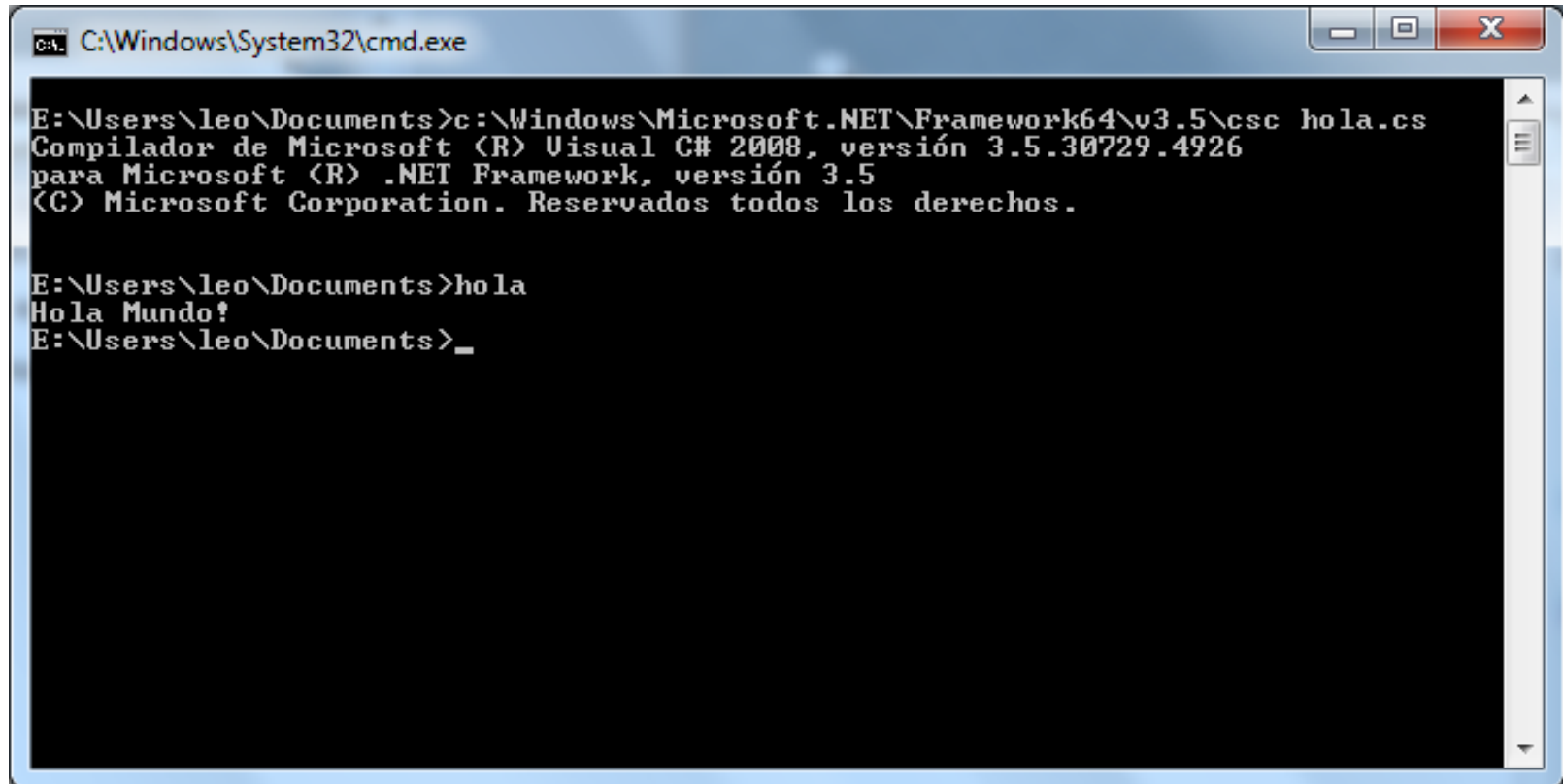
Primera aplicación en C#

```
class HolaMundo
{
    static void Main()
    {
        System.Console.Write("Hola mundo!");
    }
}
```

Compilación en línea de comando. Una vez guardado el código fuente en un archivo “hola.cs” se compila con el siguiente comando:

```
csc hola.cs
```


Primera aplicación en C#

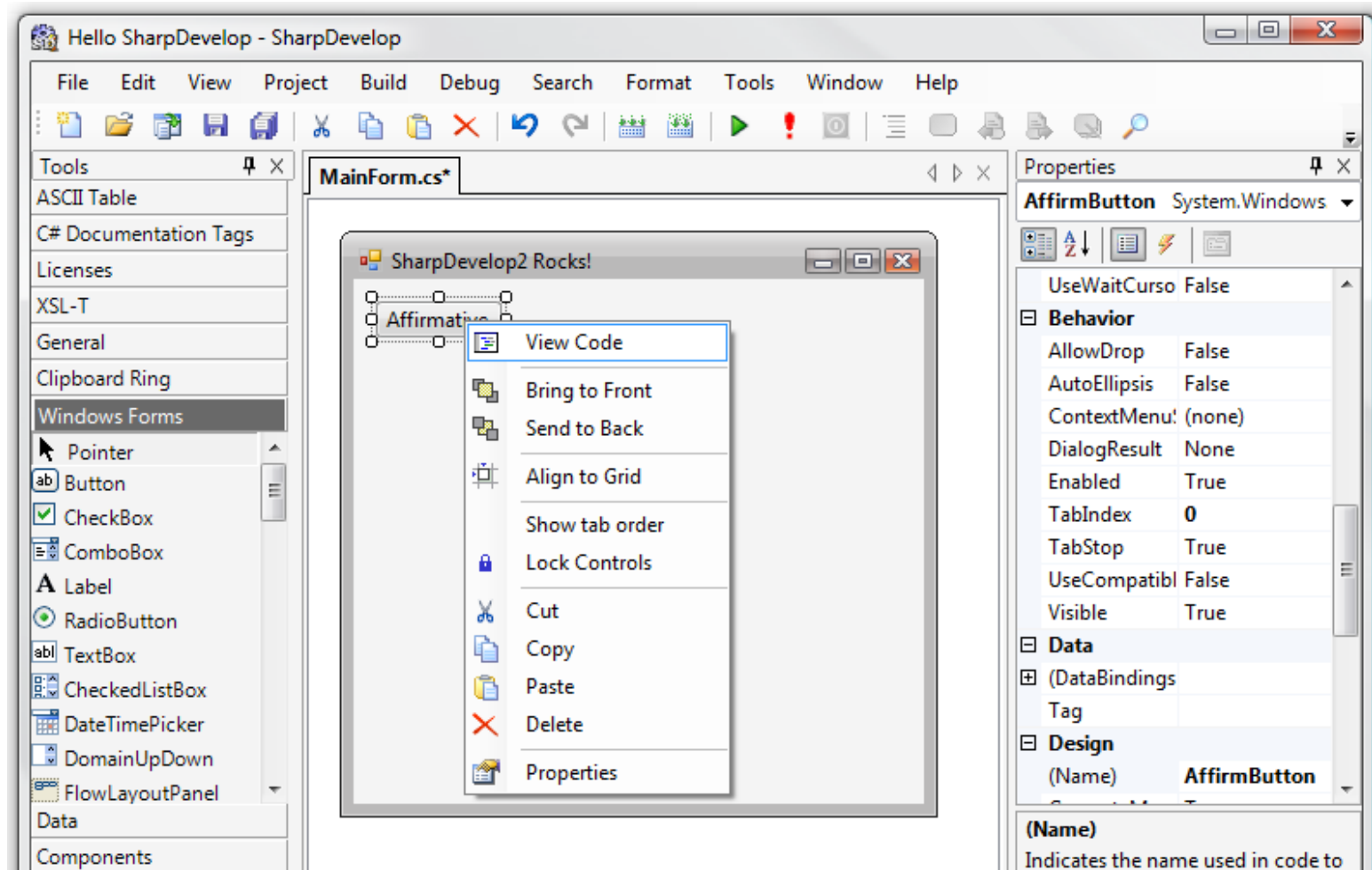


```
C:\Windows\System32\cmd.exe

E:\Users\leo\Documents>c:\Windows\Microsoft.NET\Framework64\v3.5\csc hola.cs
Compilador de Microsoft (R) Visual C# 2008, versión 3.5.30729.4926
para Microsoft (R) .NET Framework, versión 3.5
(C) Microsoft Corporation. Reservados todos los derechos.

E:\Users\leo\Documents>hola
Hola Mundo!
E:\Users\leo\Documents>_
```

Entorno de desarrollo SharpDevelop

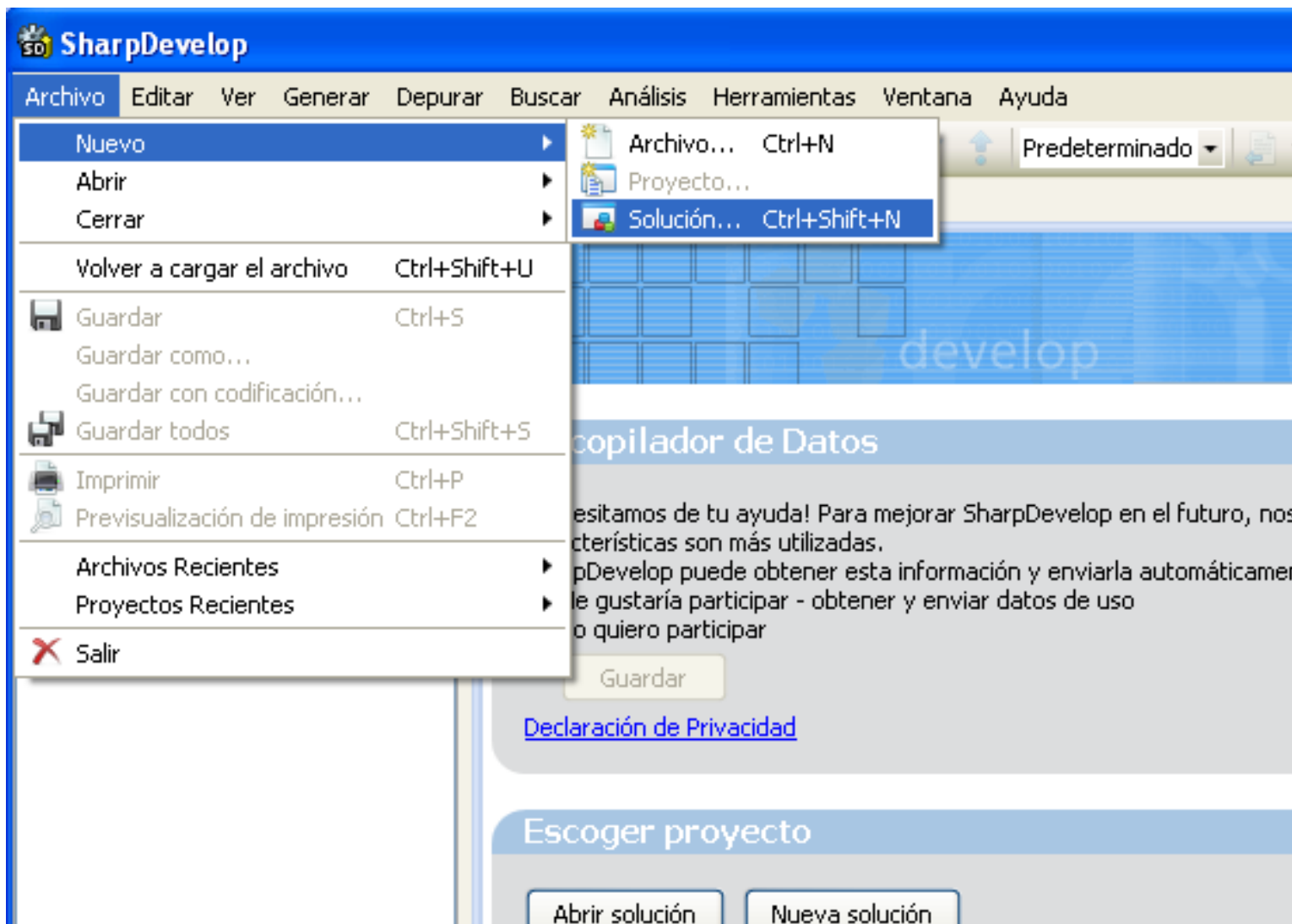


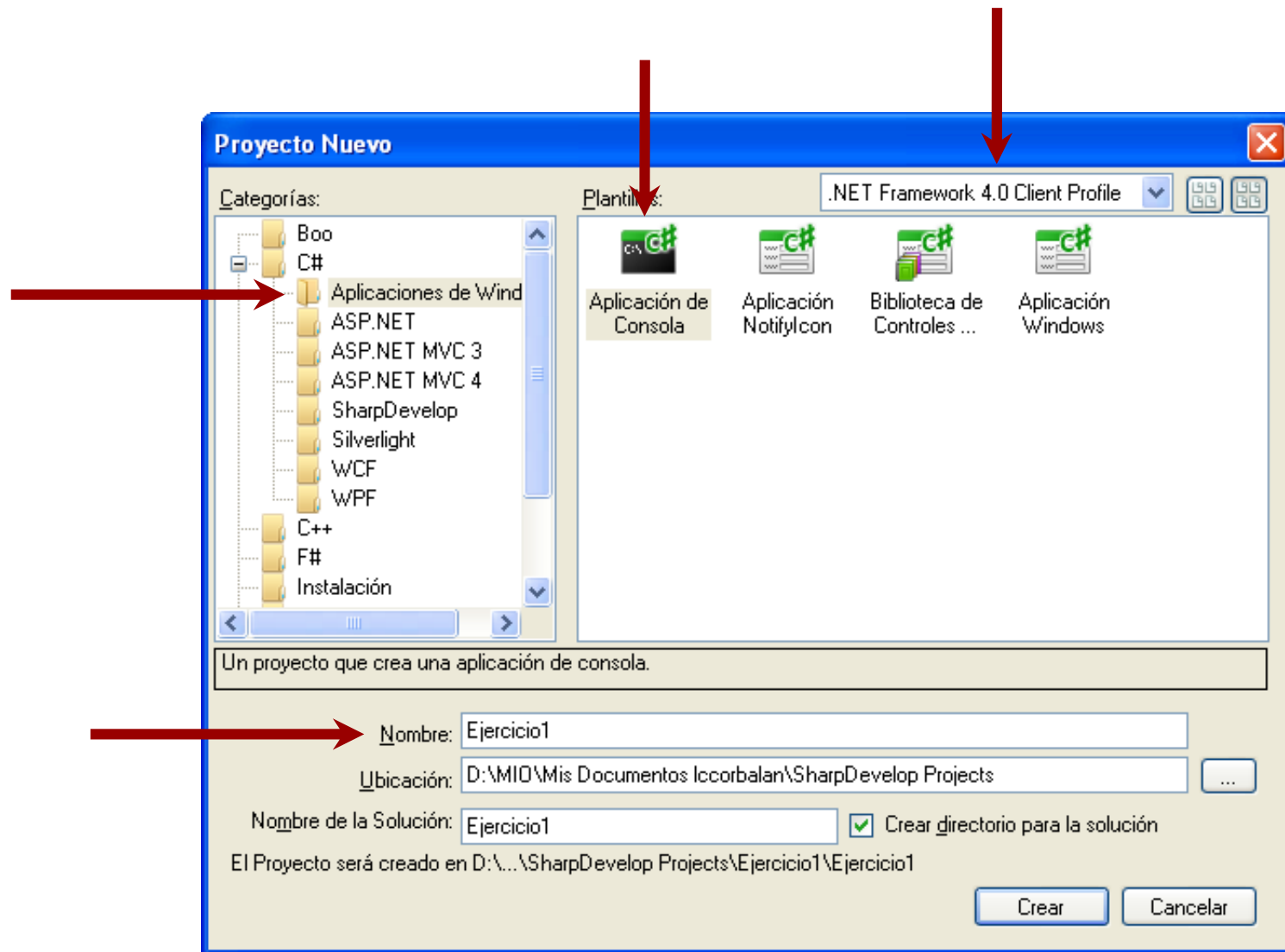
Entorno de desarrollo SharpDevelop

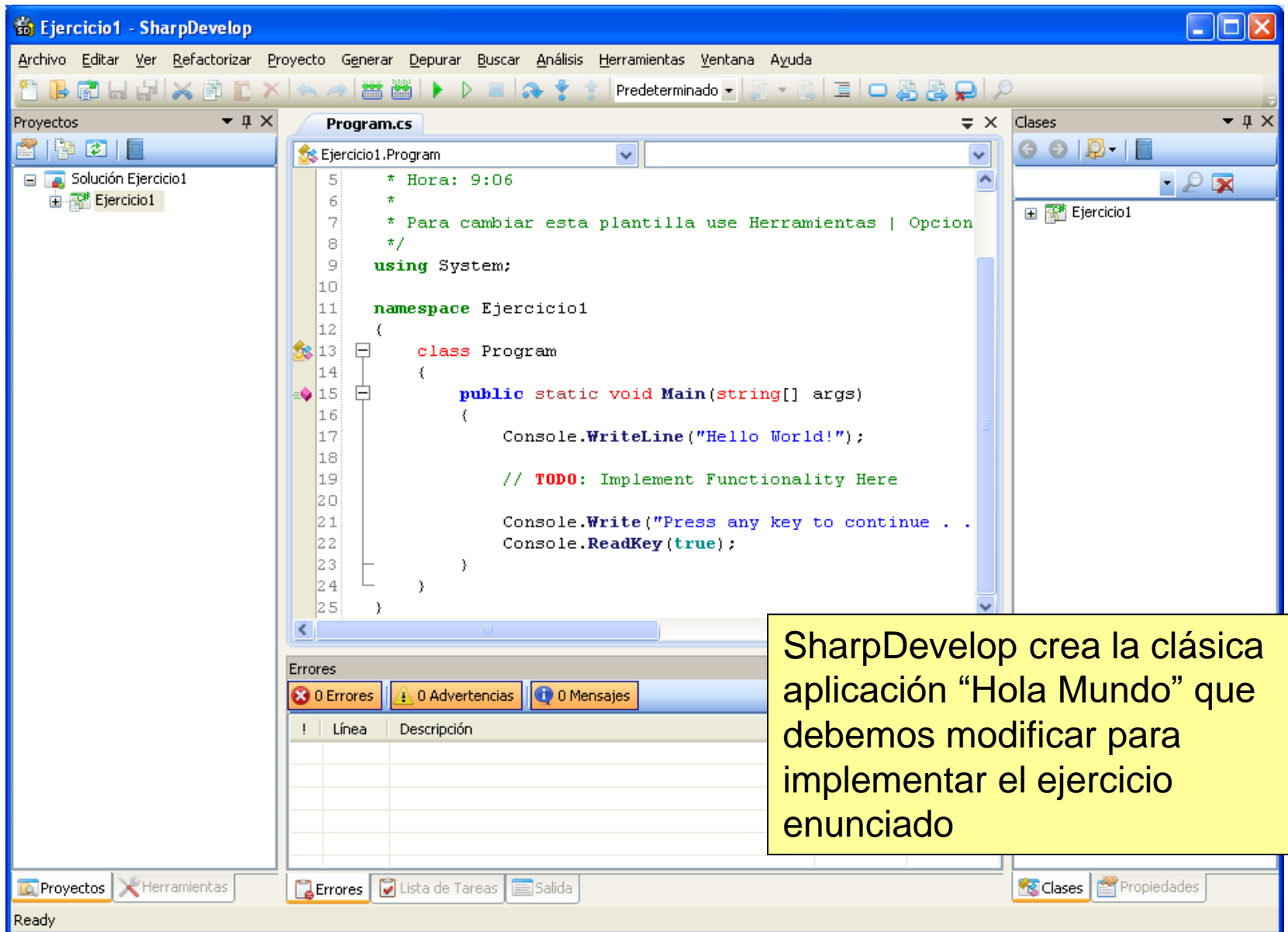
- IDE Open Source para .NET
- Soporta varios lenguajes
- Versión que utilizaremos: 5.1 (soporta Framework 2.0, 3.0, 3.5, 4.0 y 4.5)

Descargar de:

<http://www.icsharpcode.net/OpenSource/SD/Download/>







Sistema de tipos en C#

.NET Framework	Alias en C#
System.Boolean	bool
System.Byte	byte
System.Int16	short
System.Int32	int
System.Int64	long
System.Single	float
System.Double	double
System.Decimal	decimal

.NET Framework	Alias en C#
System.Char	char
System.String	string
System.Object	object
System.DateTime	-----
System.SByte	sbyte
System.UInt16	ushort
System.UInt32	uint
System.UInt64	ulong

C# es sensible a las mayúsculas !!!

Tipos numéricos en C#

Categoría	Bits	Tipo	Intervalo/precisión
Enteros con signo	8	sbyte	−128...127
	16	short	−32.768...32.767
	32	int	−2.147.483.648...2.147.483.647
	64	long	−9.223.372.036.854.775.808...9.223.372.036.854.775.807
Enteros sin signo	8	byte	0...255
	16	ushort	0...65.535
	32	uint	0...4.294.967.295
	64	ulong	0...18.446.744.073.709.551.615
Punto flotante	32	float	$1,5 \times 10^{-45}$ a $3,4 \times 10^{38}$, con precision de 7 digitos
	64	double	$5,0 \times 10^{-324}$ a $1,7 \times 10^{308}$, con precisión de 15 dígitos
Decimal	128	decimal	$1,0 \times 10^{-28}$ a $7,9 \times 10^{28}$, con precisión de 28 dígitos

Constantes y variables

- **Declarar constantes**

- Se utiliza la instrucción **const**, seguida del **tipo de datos** y el **valor** que le asignaremos a esa constante:

```
const double pi=3.1416;
```

- **Declarar variables**

- Se declaran indicando el tipo seguida por el identificador (ninguna instrucción especial para declarar variables, simplemente).

```
int i;
```

```
char c;
```

Constantes y variables

- Se pueden declarar más de una variable en la misma sentencia. Por ejemplo, el siguiente código declara tres variables del tipo `int`.

```
int a, b, c;
```

- Se puede asignar un valor inicial a las variables en la misma sentencia en la que se declaran. Por ejemplo:

```
int b = 12, c = 15;
```

```
char c='A'; string st="Hola";
```

Un literal de tipo char se escribe con comillas simples, uno de tipo string se escribe con comillas dobles

Conversiones de tipo numéricas implícitas

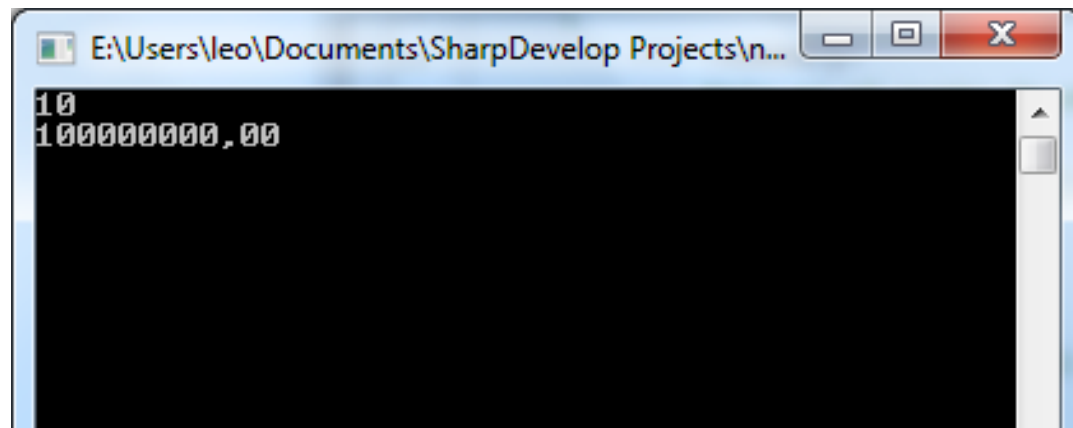
```
byte b=10;  
int i=b;  
Console.WriteLine(i);  
i = 1000000015;  
float f = i;  
System.Console.WriteLine(f.ToString("#.00"));
```

Conversión implícita de
byte a int

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

Conversiones de tipo numéricas implícitas

```
byte b=10;  
int i=b;  
Console.WriteLine(i);  
i = 1000000015;  
float f = i;  
System.Console.WriteLine(f.ToString("#.00"));
```



Conversiones de tipo numéricas implícitas

- De sbyte a short, int, long, float, double o decimal.
- De byte a short, ushort, int, uint, long, ulong, float, double o decimal.
- De short a int, long, float, double o decimal.
- De ushort a int, uint, long, ulong, float, double o decimal.
- De int a long, float, double o decimal.
- De uint a long, ulong, float, double o decimal.
- De long a float, double o decimal.
- De ulong a float, double o decimal.
- De char a ushort, int, uint, long, ulong, float, double o decimal.
- De float a double.

Las conversiones de int, uint, long o ulong a float y de long o ulong a double pueden producir una pérdida de precisión, pero no una pérdida de magnitud.

Conversiones de tipo numéricas explícitas

```
int x = 100;  
short z = x;  
double d=13.45;  
x=d;
```

Error de compilación

Error de compilación

Conversiones de tipo numéricas explícitas

SOLUCIÓN

```
int x = 100;  
short z = (short)x;  
double d=13.45;  
x=(int)d;
```

Conversión explícita
utilizando el operador
de Cast

Conversión explícita
utilizando el operador
de Cast. La variable x
es asignada con el
número 13

Conversiones de tipo numéricas explícitas

Puede haber pérdida de información o incluso excepciones

- De sbyte a byte, ushort, uint, ulong o char.
- De byte a sbyte y char.
- De short a sbyte, byte, ushort, uint, ulong o char.
- De ushort a sbyte, byte, short o char.
- De int a sbyte, byte, short, ushort, uint, ulong o char.
- De uint a sbyte, byte, short, ushort, int o char.
- De long a sbyte, byte, short, ushort, int, uint, ulong o char.
- De ulong a sbyte, byte, short, ushort, int, uint, long o char.
- De char a sbyte, byte o short.
- De float a sbyte, byte, short, ushort, int, uint, long, ulong, char o decimal.
- De double a sbyte, byte, short, ushort, int, uint, long, ulong, char, float o decimal.
- De decimal a sbyte, byte, short, ushort, int, uint, long, ulong, char, float o double.

Operadores aritméticos

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

Operadores relacionales

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

Operadores lógicos

Operador	Operación
&	AND
	OR
!	NOT
^	XOR
& &	AND en cortocircuito
	OR en cortocircuito

Operadores de asignación

Operador	Operación
++	Incremento
--	Decremento
=	Asignación simple
*=	Multiplicación más asignación
/=	División más asignación
%=	Residuo más asignación
+=	Suma más asignación
-=	Resta más asignación

Espacios de nombres

- .NET Framework utiliza los espacios de nombres para organizar sus múltiples clases. Ejemplo:

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

- `System` es un espacio de nombres y `Console` es una clase de ese espacio de nombres.

Espacios de nombres

- Se puede utilizar la palabra clave `using` para que no se requiera el nombre completo. Ejemplo :

```
using System;
```

- Luego en el código puede utilizarse simplemente.

```
Console.WriteLine("Hola Mundo!");
```

Espacios de nombres

- Pueden declararse espacios de nombres propios para ayudar a controlar el ámbito de clase y nombres de método en proyectos de programación grandes.
- Para ello se utiliza la palabra clave `namespace`.

Espacios de nombres

- Ejemplo:


```
namespace NamespaceDeEjemplo
{
    using System
    class ClaseDeEjemplo
    {
        public void MetodoDeEjemplo()
        {
            Console.WriteLine("Ejemplo");
        }
    }
}
```


Estructuras de control

Bloque

Un bloque es una lista de cero o más sentencias encerradas entre llaves “{” “}”

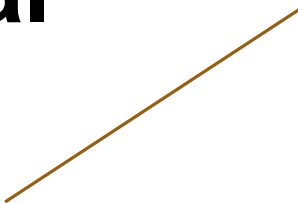
```
{  
    int i=1;  
    Console.WriteLine(i);  
}
```



El alcance de una variable declarada en el bloque es el propio bloque

Estructuras de control

Condicional



La condición siempre
va entre paréntesis

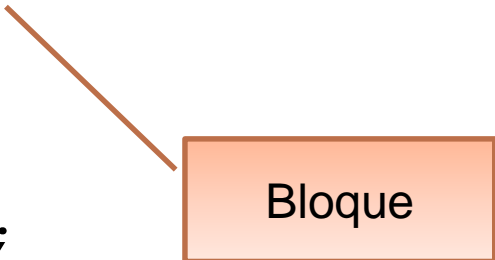
```
if (<test>)  
    <Código si test es verdadero>;  
else  
    <Código si test es falso>;
```

Estructuras de control

Condicional

Ambas partes del if pueden contener múltiples instrucciones encerradas entre llaves

```
if (<test>)  
{  
    <Código si <test> es verdadero>;  
}  
else  
{  
    <Código si <test> es falso>;  
}
```



Bloque

Estructuras de control

```
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

```
do  
{  
    <Código del bucle>  
} while (<Test>);
```

Ejemplo:

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

Estructuras de control

```
while (<Test>)  
{  
    <Código del bucle>  
}
```

Ejemplo:

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

Estructuras de control

```
for (<inicializ.>; <condición>; <operación>)  
{  
    <código del bucle>  
}
```

Esto trabaja igual que:

```
<inicializ.>  
while (<condición>)  
{  
    <código del bucle>  
    <operación>  
}
```

Estructuras de control

Ejemplo:

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

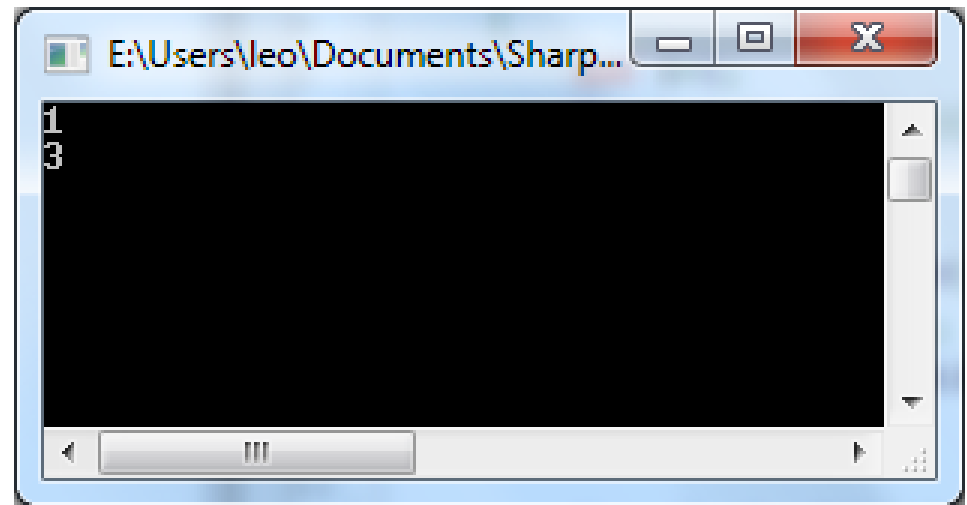

Estructuras de control

- Interrupción de bucles:
 - **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 recomendada)
 - **return** – Salta fuera del loop y de la función que lo contiene

Estructuras de control

- Ejemplo

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



Ámbito de las variables

- No se puede ocultar un nombre de un ámbito local redefiniendo el identificador en otro más interno.

```
int i = 0;  
if (true) {  
    int i = 1;  
}
```

Error de
compilación

```
if (true) {  
    int j = 0;  
}  
if (true) {  
    int j = 1;  
}
```

Válido

Ejercicios para practicar

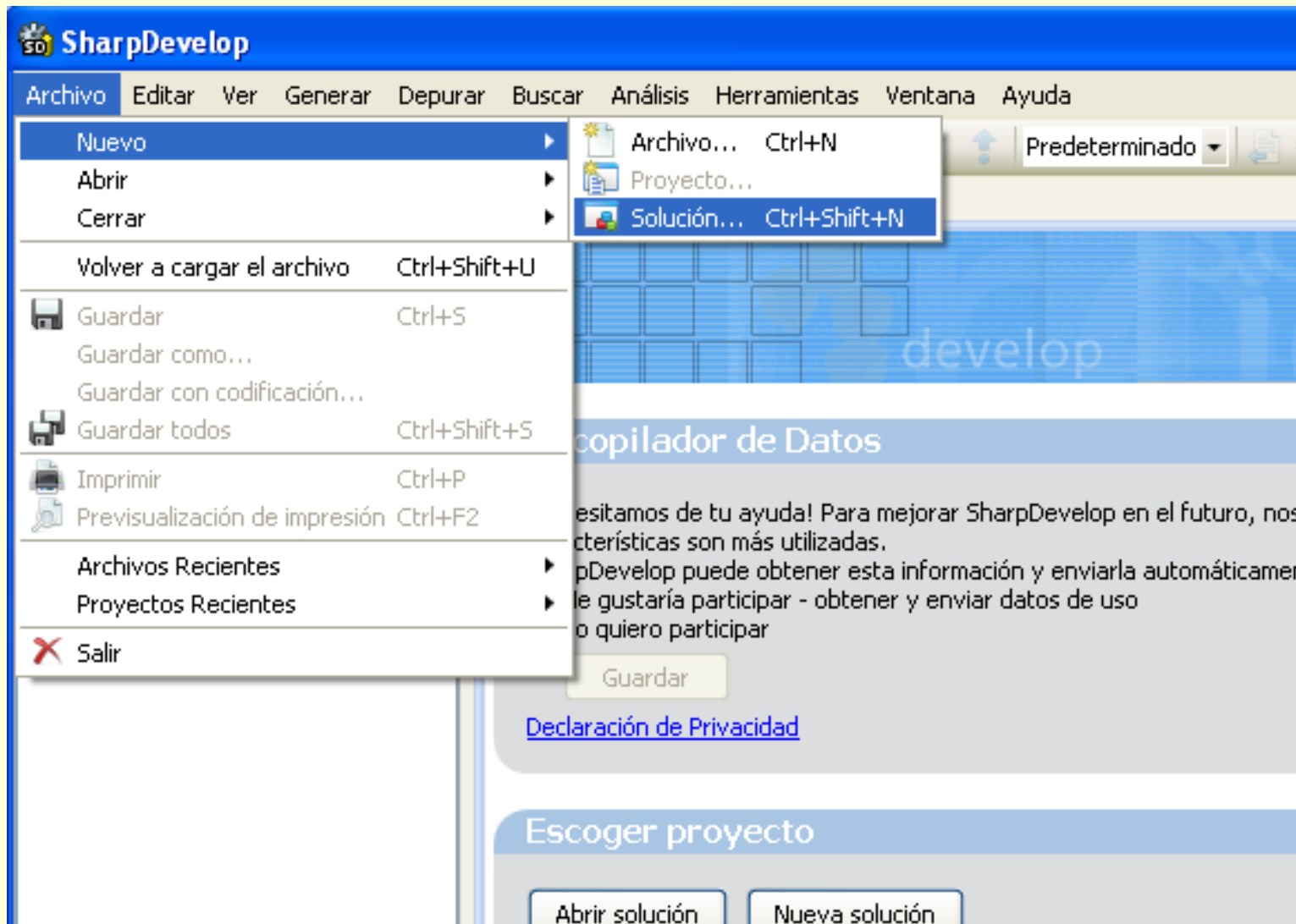
Ejercicio

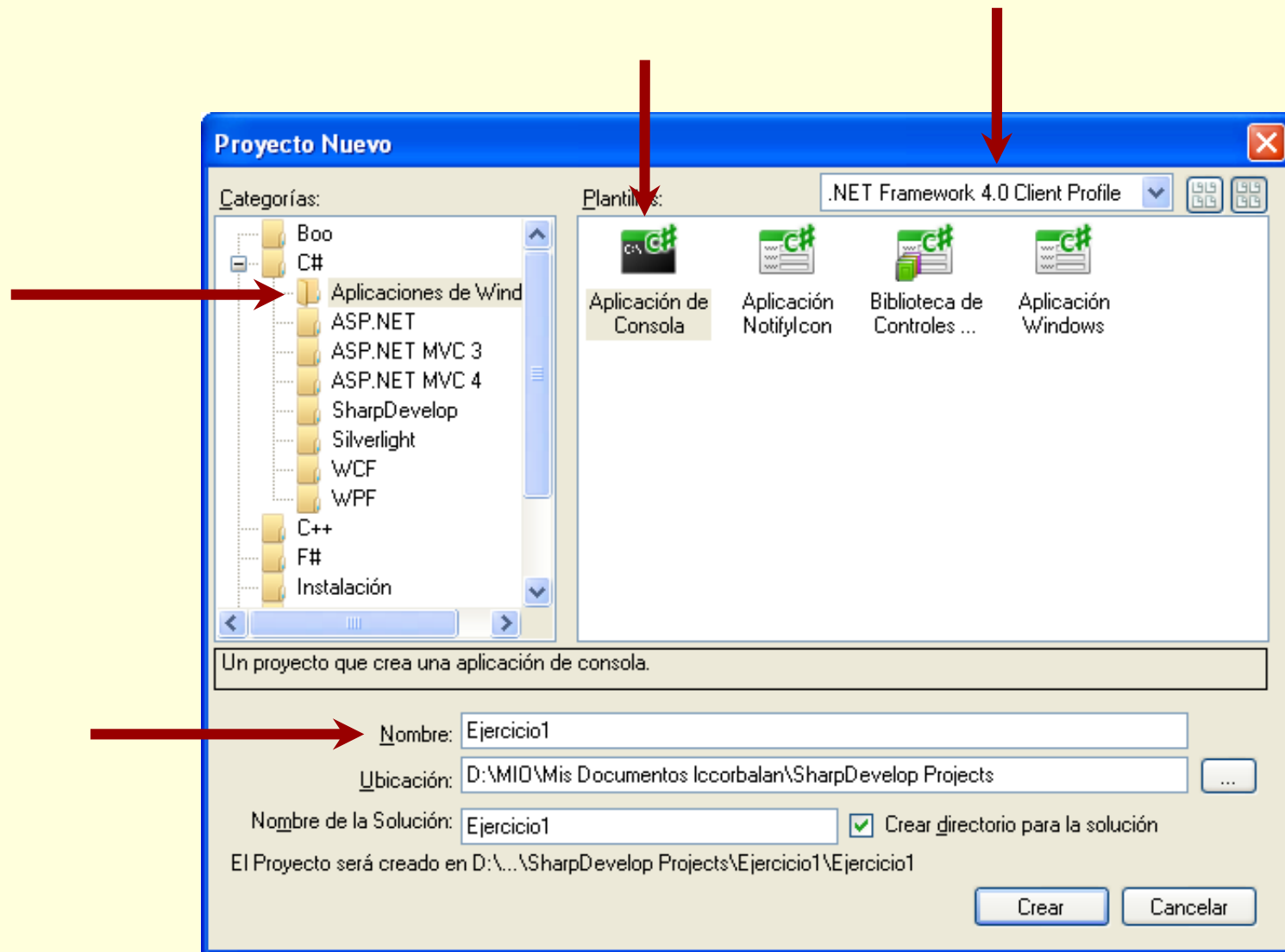
- Solicitar al usuario que ingrese por teclado su nombre y saludarlo de manera personalizada.

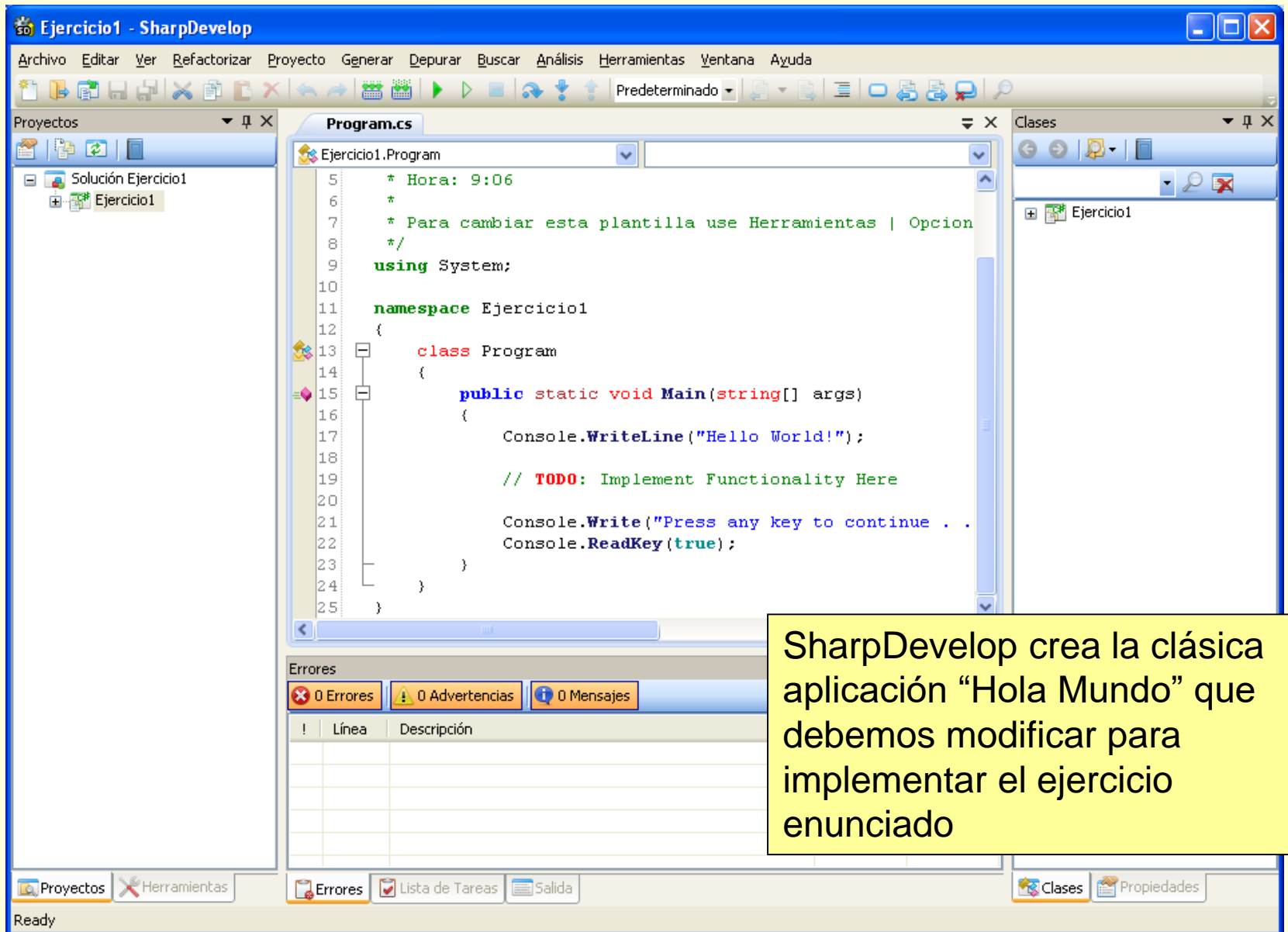
Tip: La instrucción

```
nombre = Console.ReadLine();
```

Lee un string desde la consola asignándolo a la variable nombre








```
7      * Para cambiar esta plantilla us
8      */
9      using System;
10
11      namespace Ejercicio1
12      {
13          class Program
14          {
15              public static void Main(string[] args)
16              {
17                  string nombre;
18                  Console.WriteLine("Ingrese su nombre ");
19                  nombre=Console.ReadLine();
20                  Console.WriteLine("Hola " + nombre);
21
22                  // TODO: Implement Functionality Here
23
24                  Console.Write("Press any key to continue . .
25                  Console.ReadKey(true);
26              }
27      }
```

Utilizar el método
Console.ReadLine()
que devuelve el string
ingresado por el usuario

Ejercicio 1

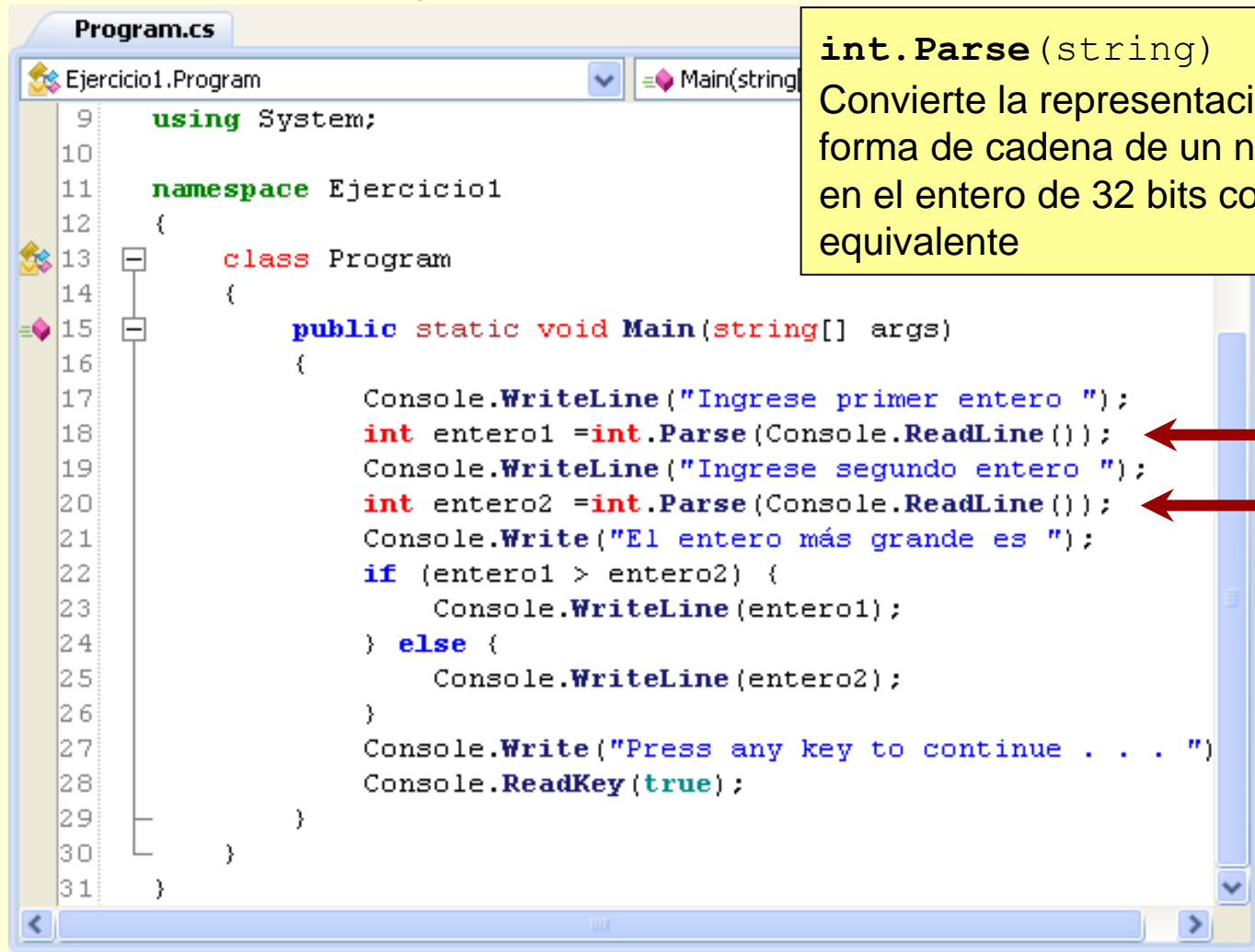
- Dados dos enteros ingresados por el usuario determinar cuál es el mayor de ellos.

Tip: La instrucción :

```
int entero1 = int.Parse(Console.ReadLine());
```

Lee un string desde la consola, lo convierte a entero y lo asigna a la variable entero1

Ejercicio 1



```
Program.cs
Ejercicio1.Program
Main(string[])
9  using System;
10
11  namespace Ejercicio1
12  {
13      class Program
14      {
15          public static void Main(string[] args)
16          {
17              Console.WriteLine("Ingrese primer entero ");
18              int entero1 = int.Parse(Console.ReadLine());
19              Console.WriteLine("Ingrese segundo entero ");
20              int entero2 = int.Parse(Console.ReadLine());
21              Console.Write("El entero más grande es ");
22              if (entero1 > entero2) {
23                  Console.WriteLine(entero1);
24              } else {
25                  Console.WriteLine(entero2);
26              }
27              Console.Write("Press any key to continue . . . ");
28              Console.ReadKey(true);
29          }
30      }
31  }
```

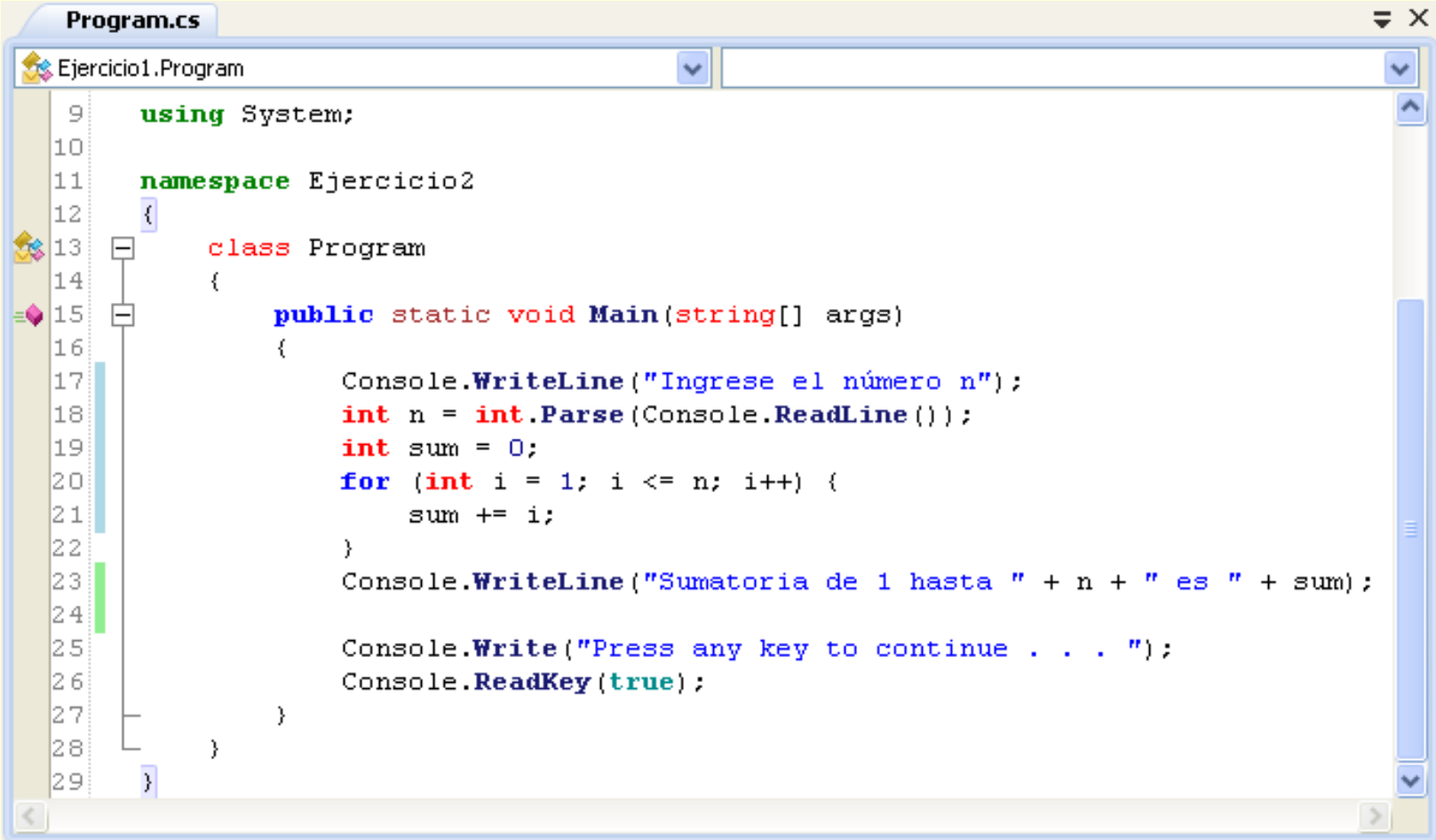
int.Parse(string)

Convierte la representación en forma de cadena de un número en el entero de 32 bits con signo equivalente

Ejercicio 2

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

Ejercicio 2

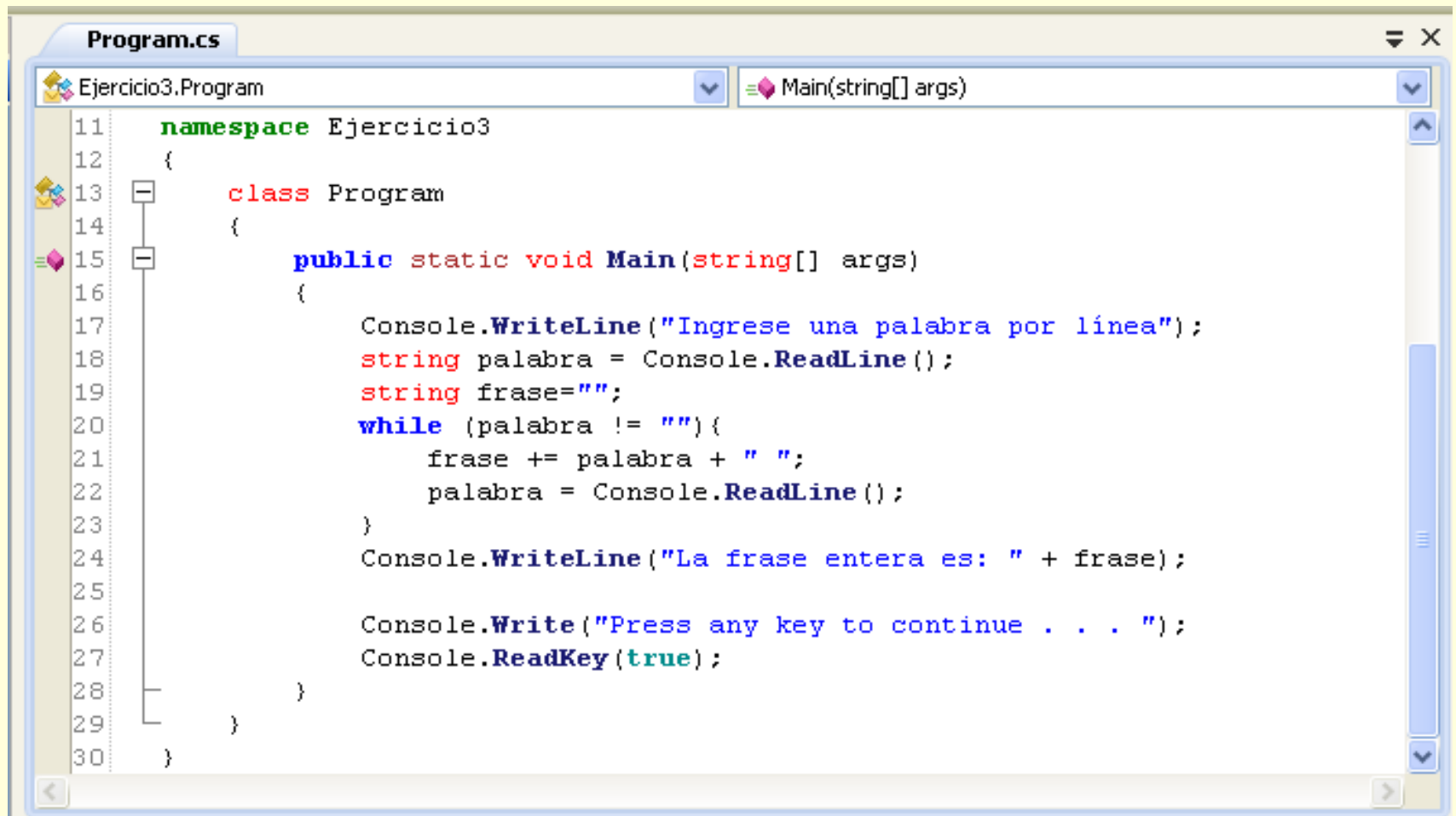


```
Program.cs
Ejercicio1.Program
9  using System;
10
11  namespace Ejercicio2
12  {
13      class Program
14      {
15          public static void Main(string[] args)
16          {
17              Console.WriteLine("Ingrese el número n");
18              int n = int.Parse(Console.ReadLine());
19              int sum = 0;
20              for (int i = 1; i <= n; i++) {
21                  sum += i;
22              }
23              Console.WriteLine("Sumatoria de 1 hasta " + n + " es " + sum);
24
25              Console.Write("Press any key to continue . . . ");
26              Console.ReadKey(true);
27          }
28      }
29  }
```

Ejercicio 3

- Solicitar al usuario que ingrese una frase palabra por palabra en distintas líneas (termina al ingresar una palabra en blanco) Luego mostrar la frase completa en una sola línea.

Ejercicio 3



```
11 namespace Ejercicio3
12 {
13     class Program
14     {
15         public static void Main(string[] args)
16         {
17             Console.WriteLine("Ingrese una palabra por línea");
18             string palabra = Console.ReadLine();
19             string frase="";
20             while (palabra != ""){
21                 frase += palabra + " ";
22                 palabra = Console.ReadLine();
23             }
24             Console.WriteLine("La frase entera es: " + frase);
25
26             Console.Write("Press any key to continue . . . ");
27             Console.ReadKey(true);
28         }
29     }
30 }
```

Bibliografía

Los contenidos publicados por la cátedra en la plataforma webUNLP cubren las necesidades surgidas de las actividades del curso.

Sin embargo se aconseja adquirir el hábito de consultar el material de referencia publicado en el sitio web MSDN library en relación a la plataforma .NET y al lenguaje C#.

<http://msdn.microsoft.com/es-es/library/ms123401.aspx>

Bibliografía complementaria

- Illustrated C# 2010, Daniel M. Solis. Apress 2010
- .NET Framework Essentials, Thuan L. Thai, Hoang Q. Lam, O'Reilly, 2003.
- Como Programar en C#, H. Deitel, Pearson. Prentice Hall 2007.
- Dissecting a C# Application Inside SharpDevelop, C. Holm, M. Krüger, B. Spuida, APress, 2004.
- C# al Descubierto, Joseph Mayo, ed. Prentice Hall, ISBN 84-205-3477-3
- Inside C#, Tom Archer, ed. Microsoft Press, ISBN 0735616485
- Learning XML, Second Edition, E. Ray, O'Reilly, 2003