



U.A. PROGRAMACION MODULAR

Número de practica: 1

Academia de Programación

Título: Reconocimiento del IDE (Integrated Development Enviroment) Entorno de desarrollo integrado

Objetivo: El alumno reconoce el entorno de desarrollo integrado de C# para desarrollar programas que utilicen módulos.

Reconocimiento del IDE

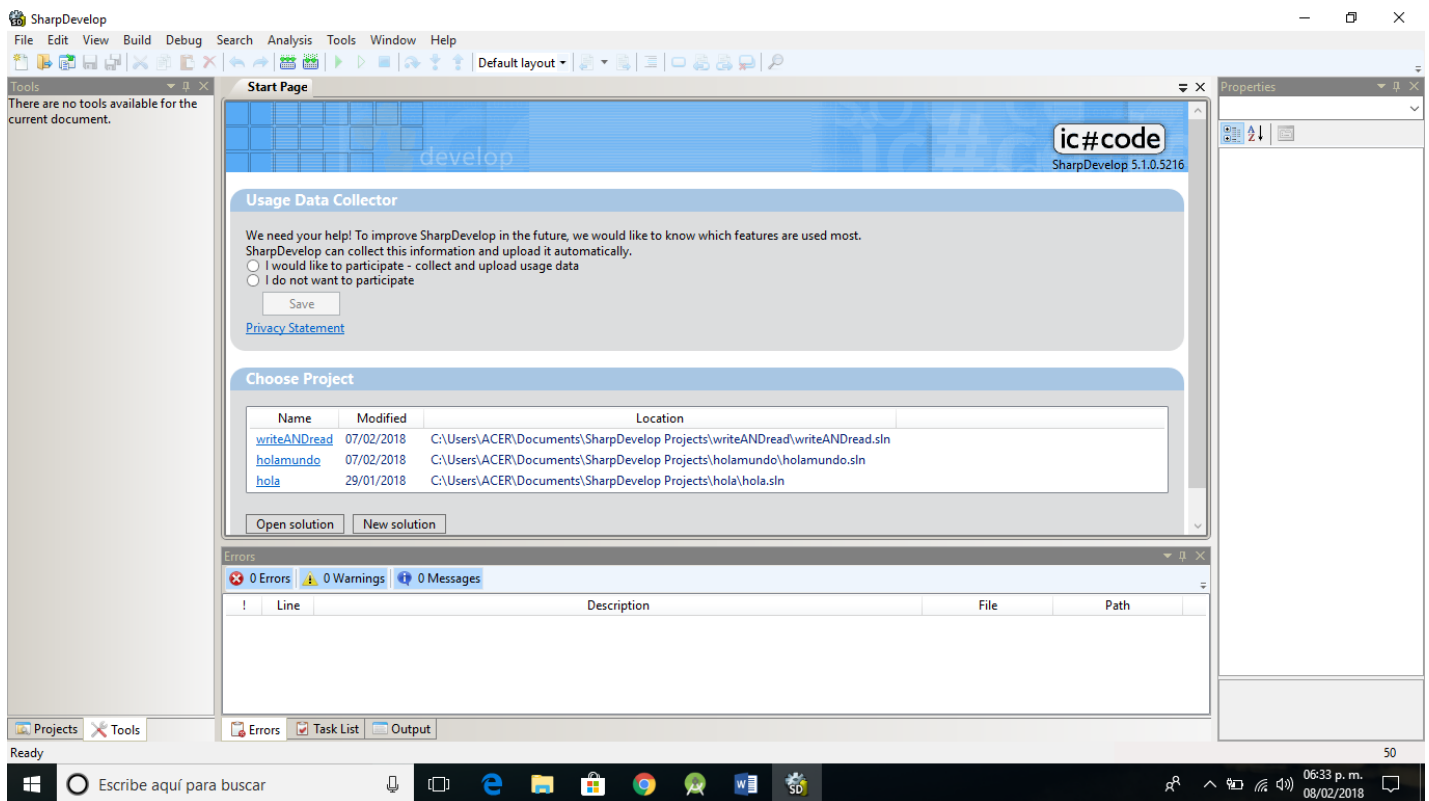
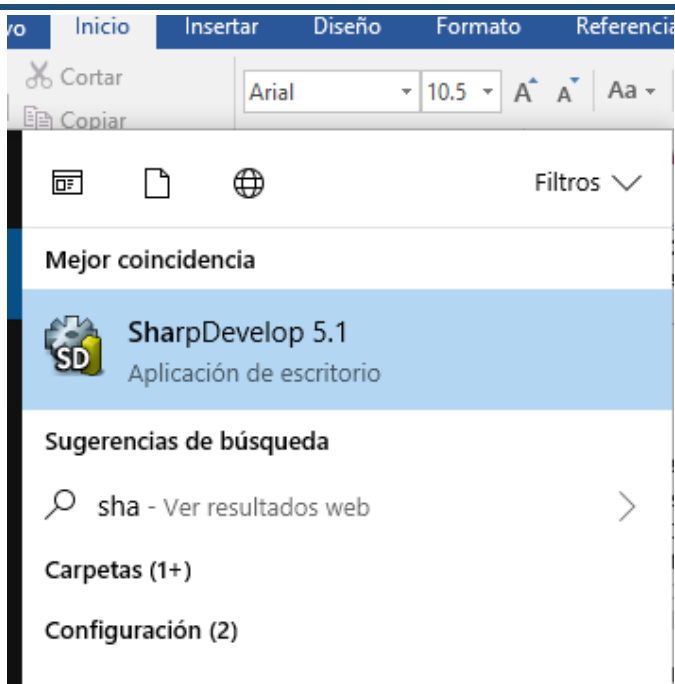
Microsoft C# es un nuevo lenguaje de programación diseñado para crear un amplio número de aplicaciones empresariales que se ejecutan en .NET Framework. Supone una evolución de Microsoft C y Microsoft C++; es sencillo, moderno, proporciona seguridad de tipos y está orientado a objetos. El código creado mediante C# se compila como código administrado, lo cual significa que se beneficia de los servicios de Common Language Runtime. Estos servicios incluyen interoperabilidad entre lenguajes, recolección de elementos no utilizados, mejora de la seguridad y mayor compatibilidad entre versiones.

C# se presenta como Visual C# en el conjunto de programas Visual Studio .NET. Visual C# utiliza plantillas de proyecto, diseñadores, páginas de propiedades, asistentes de código, un modelo de objetos y otras características del entorno de desarrollo. La biblioteca para programar en Visual C# es .NET Framework. (Referencia MSDN).

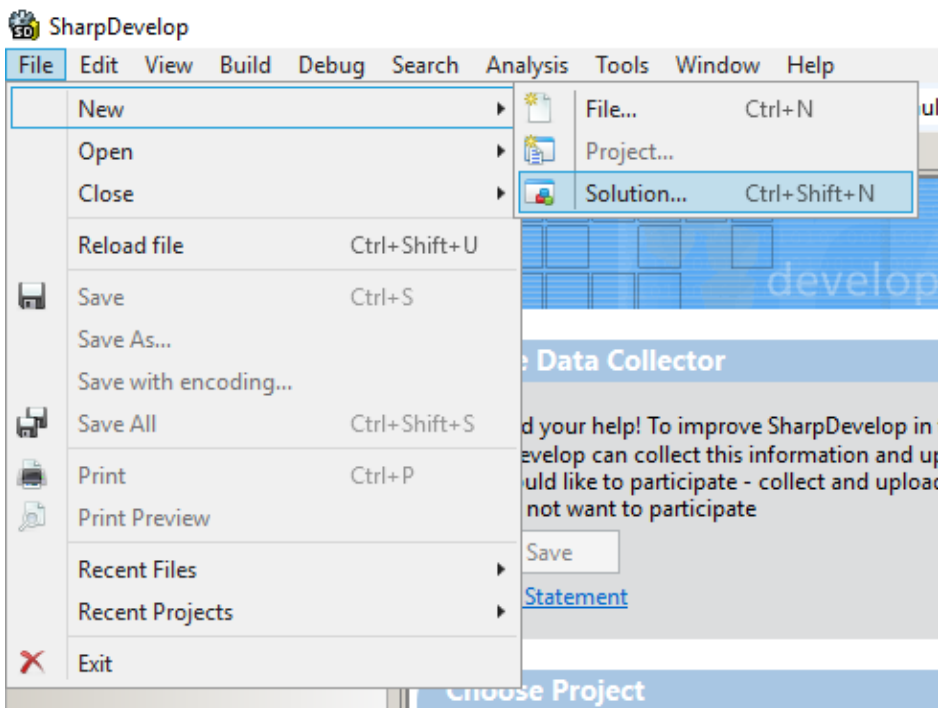
INTERFAZ

La interfaz para iniciar un primer programa es realmente amigable, los pasos siguientes explican cómo crear un primer programa:

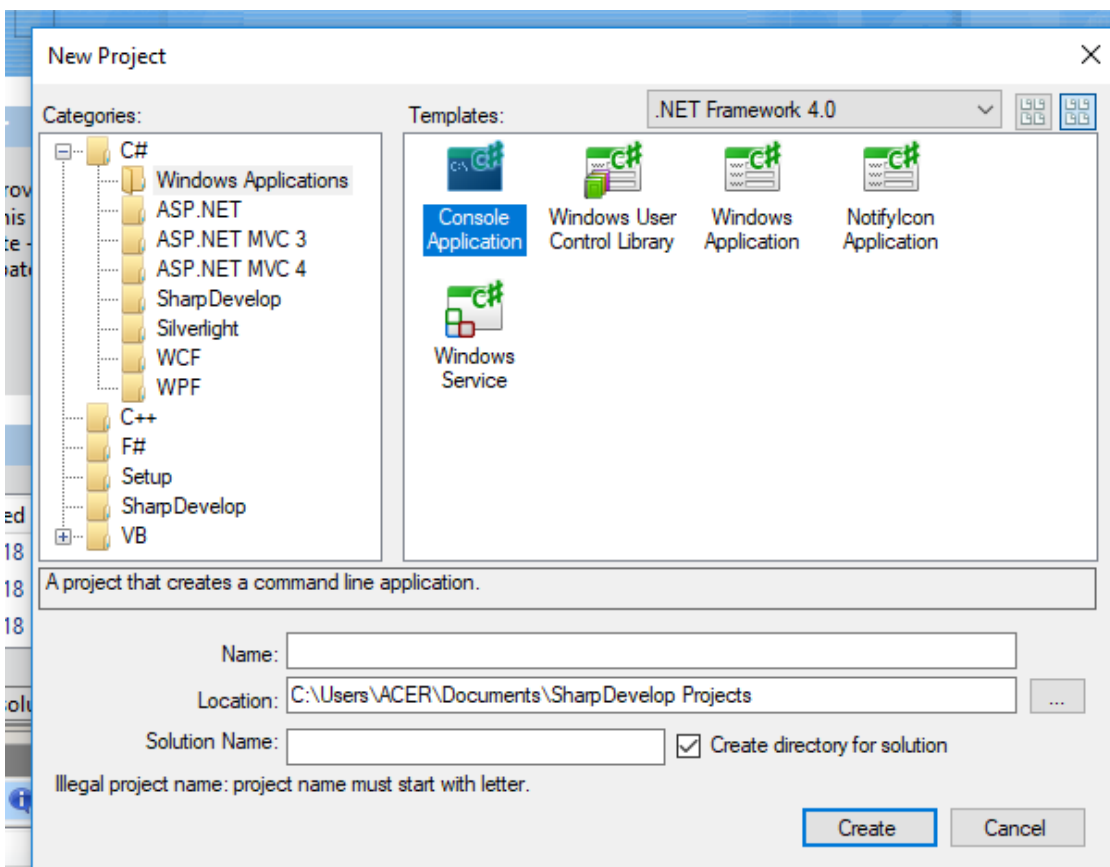
Para crear un primer proyecto, primero debe abrir el archivo ejecutable de SharpDevelop



Posteriormente indicamos que vamos a crear un nuevo proyecto y elegimos después el lenguaje a utilizar.

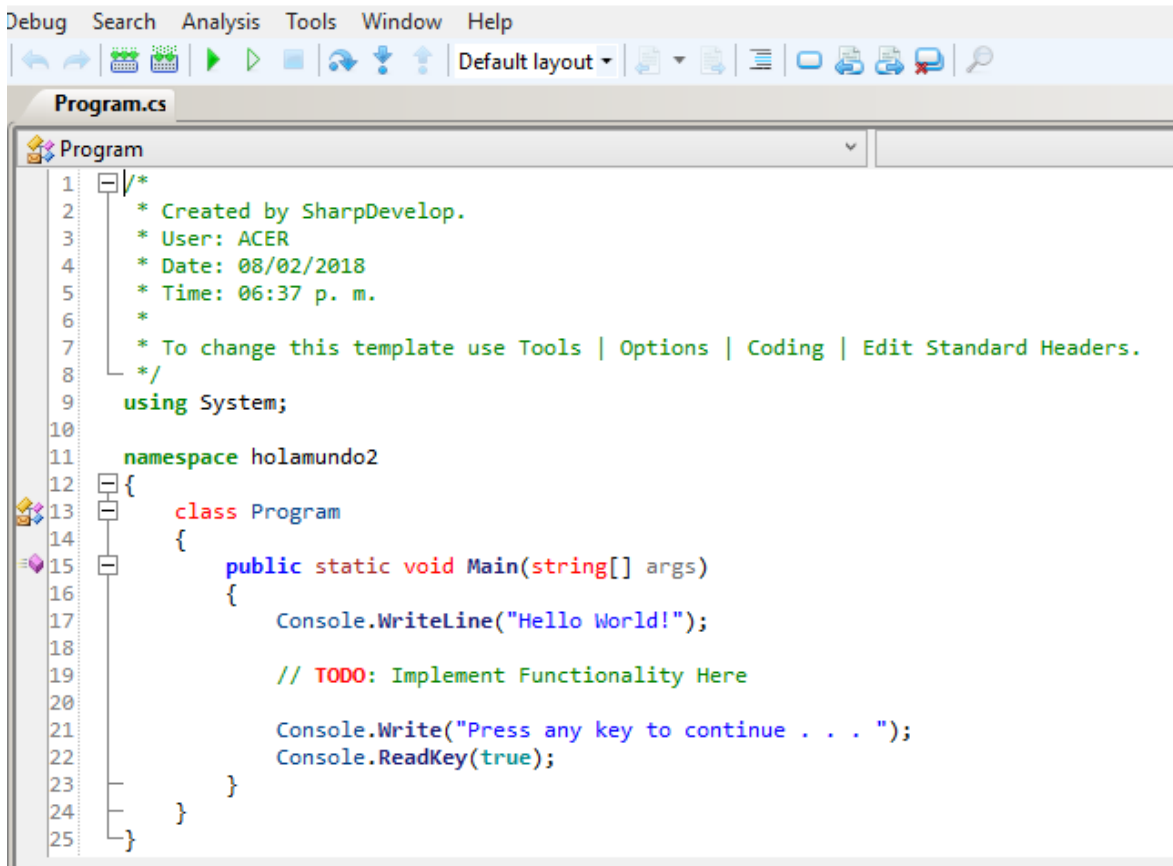


Como puedes observar en la imagen vamos a crear una aplicación de consola.



Como nombre le colocaremos holamundo.

Una vez que damos clic en aceptar nos aparece la siguiente ventana:



Por el momento solo nos concentraremos en la ventana de en medio que es la ventana de código. Veamos a continuación el código que nos genera visual c#

```
using System;
```

Los using son las librerías necesarias para que un programa de visual c# pueda funcionar .

```
namespace holamundo2
{
    class Program
    {
        static void Main(string[] args)
        {
        }
    }
}
```

El Namespace sirve para agrupar varias clases o tipos relacionados que tienen una funcionalidad común. Generalmente es el primer bloque que ves en el código de tu programa (después de los using), el namespace ayuda a prevenir la colisión de nombres, mantener un código agrupado, ordenado y facilitar la lectura para los programadores.

```
class Program
{
    static void Main(string[] args)
    {
    }
}
```

Como ya se menciona Visual C# es un lenguaje orientado a objetos por lo que hace uso de clases, en este curso no veremos a detalle cómo se crean la clase por lo que usaremos la clase que se genera automáticamente.

```
static void Main(string[] args)
{
}
```


En la mayoría de los lenguajes de programación modulares existe una función o método principal en el caso de Visual C# es static void Main, como toda función o método puede recibir parámetros en este caso string[] arg. Más adelante veremos que son y cómo se usan los parámetros.

Bien, ahora veremos el código para que nos aparezca el hola mundo en una pantalla de consola.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace holamundo
{
    class Program
    {
        static void Main(string[] args)
        {
            System.Console.WriteLine("Hola mundo. . .\n");
            System.Console.ReadKey();
        }
    }
}
```

La instrucción System.Console.WriteLine(" . . . "); le indica al sistema que se escribirá una cadena y después dará un salto de línea. La instrucción System.Console.ReadKey(); se usa para que se detenga un momento la consola y podamos visualizar el salida.

Para ejecutar nuestra aplicación, podemos utilizar la tecla de función F5 o dar clic en el botón de Iniciar  Y se desplegara la siguiente ventana

```
file:///C:/Users/edu.vazper/AppData/Local/Temporary Projects/h
Hola mundo. . .
```

Ejercicio

1. Crea una aplicación de consola donde se despliegue el nombre de la U.A., tu nombre completo, tu número de boleta y tu grupo.

Write y Read

El método Write()

Write escribe lo que sea sin añadir el carácter de fin de línea a la cadena, de modo que lo siguiente que se escriba se pondrá a continuación de lo escrito con Write.

El método WriteLine()

Este método es el que se usa para mostrar texto en la consola, el método escribe en la pantalla el valor que le pasemos como parámetro.

El parámetro que recibe el método puede ser de varios tipos, ya sea una cadena de caracteres, un número entero, una línea en blanco, etc...

El método ReadLine()

Este método se usa para recoger la información que el usuario introduce cuando la aplicación así lo requiera. Cuando invocamos al método Console.ReadLine() el sistema queda en espera hasta que el usuario pulsa la tecla Intro.

Si se asigna la llamada a Console.ReadLine() a una variable se consigue capturar el dato introducido por el usuario, para después poder operar con él.

El método Read()

Este es otro método que permite capturar información que proviene del usuario. La diferencia con el anterior es que Read() no espera a que el usuario pulse intro para capturar el dato introducido, sino que lo hace tras la pulsación de cualquier tecla, capturando el valor de la tecla pulsada en código ASCII.

Para verificar el funcionamiento de Write, WriteLine realiza la siguiente aplicación de consola en visual C#

```
Program.cs
writeandwriteline.Program
Main(string[] args)

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace writeandwriteline
{
    class Program
    {
        static void Main(string[] args)
        {
            System.Console.WriteLine("Mi nombre es Eduardo y mis apellidos");
            System.Console.Write("Vazquez ");
            System.Console.Write("Peralta ");
            System.Console.WriteLine("Y tengo:");
            System.Console.Write("25 años");
            System.Console.ReadKey();
        }
    }
}
```

file:///c:/users/edu.vazper/documents/visual studio 2013/Projects/w

```
Mi nombre es Eduardo y mis apellidos
Vazquez Peralta Y tengo:
25 años
```

Para verificar el funcionamiento de Read y ReadLine realiza la siguiente aplicación de consola en visual C#

```

Program.cs  X
ReadandReadLine.Program  Main(string[] args)
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ReadandReadLine
{
    class Program
    {
        static void Main(string[] args)
        {
            string nombre;
            int edad;
            Console.Write("Cual es tu nombre: ");
            nombre=Console.ReadLine();
            Console.Write("Cual es tu edad: ");
            edad=Convert.ToInt32(Console.Read());
            Console.WriteLine("Hasta luego. . . .");
            Console.ReadKey();
        }
    }
}

```

Como puedes observar podemos omitir en las instrucciones la palabra System.

Los métodos Read y ReadLine capturan cadenas desde el teclado por lo que se deben convertir a un valor entero con la instrucción Convert.ToInt32(). La ejecución seria como la siguiente:

file:///c:/users/edu.vazper/documents/visual studio 2013/Projects/ReadandReadl

```

Cual es tu nombre: Eduardo Vazquez
Cual es tu edad: 25
Hasta luego. . . .

```

Tipos de datos

C# es compatible con los siguientes tipos de datos integrados:

Tipo de datos	Intervalo
byte	0 .. 255
sbyte	-128 .. 127
short	-32,768 .. 32,767
ushort	0 .. 65,535
int	-2,147,483,648 .. 2,147,483,647
uint	0 .. 4,294,967,295
long	-9,223,372,036,854,775,808 .. 9,223,372,036,854,775,807
ulong	0 .. 18,446,744,073,709,551,615

float	-3.402823e38 .. 3.402823e38
double	-1.79769313486232e308 .. 1.79769313486232e308
decimal	-79228162514264337593543950335 .. 79228162514264337593543950335
char	Un carácter Unicode.
string	Una cadena de caracteres Unicode.
bool	True o False.

Operadores de asignación e igualdad

En C#, el operador signo igual (=) tiene la misma funcionalidad que en C y C++:

Operador	Finalidad
=	Asigna un valor.
==	Comprueba la igualdad.

Operadores matemáticos y lógicos

La siguiente lista muestra los operadores matemáticos básicos en orden de prioridad. Utiliza paréntesis para imponer otro orden.

Operador	Finalidad
*, /, %	Multiplicación, división, módulos
+, -	Suma, resta
&	AND lógico
^	XOR lógico
	OR lógico

Operadores de incremento y decremento

Operador	Finalidad
v++	Incrementar variable v por 1.
v+=n	Incrementar variable v por n.
v*=n	Multiplicar variable v por n.
v-=n	Restar n de variable v.

Operadores relacionales

Los siguientes operadores comparan dos valores y devuelven un resultado booleano:

Operador	Finalidad
==	Comprobar igualdad.
!=	Comprobar desigualdad.
>	Mayor que
<	Menor que
>=	Mayor o igual que
<=	Menor o igual que

Operadores lógicos de condición

Los operadores lógicos se utilizan para crear instrucciones de condición más flexibles combinando varias cláusulas:

Operador	Finalidad
&&	AND condicional.
	OR condicional.
!	NOT condicional.

Convertir a un valor numérico.

Muchos tipos, incluyendo todos los tipos numéricos, proporcionan el método Parse() que permite la conversión desde un dato string p.e.:

```
double d = double.Parse("20.5");  
float f = float.Parse("20.5");  
int i = int.Parse("20");  
long l = long.Parse("200000000");
```

Lista de cotejo

Reactivo		CUMPLE		OBSERVACIONES
		SI	NO	
(5%)	Usa correctamente la sintaxis para desplegar datos en pantalla usando la consola.			
(95%)	Genera el programa utilizando la consola para desplegar información en pantalla, obteniendo correctamente el resultado.			
100%	CALIFICACIÓN:			

Nota: En la columna “OBSERVACIONES” ocúpela cuando tenga que hacer comentarios referentes a lo observado.

