

Clase 2

El Common Language Runtime (CLR)

Como mencioné en la Clase 1, los lenguajes .NET sacan provecho de todo un **framework** diseñado con el propósito de suplir la explotación más adecuada de todos los recursos con los que cuenta el sistema operativo donde se está programando.

Dentro de ese framework se identifican los “**Assemblies**” de .NET que podemos entender como “librerías”. Un **assembly**, como si fuera un paquete, una caja de herramientas, ofrece una serie de objetos pre-definidos y categorizados en

Namespaces que podemos consumir desde nuestro programa para por ejemplo: mostrar un texto en pantalla, leer datos del teclado, leer datos de un archivo o puerto de comunicación, trazar líneas y figuras por pantalla, manejar diferentes threads, usar controles (como botones, listas, cuadros de texto, etc si estamos desarrollando para Windows), usar sockets, consumir a través de SQL datos de una Base de Datos y muchísimos otros objetos que proveen toda la funcionalidad necesaria para cumplir con su cometido.

Uno de esos assembly es “**system.dll**” que cuenta con la mayoría de los namespaces que necesitaremos usar.

Namespace: En namespaces es la forma en que se agrupan objetos con funcionalidades comunes en un assembly. Por ejemplo en el namespace System.IO se supone que se encuentran todos los objetos necesarios para interactuar con archivos en disco, en System.Collections están todos los objetos que cumplen la función de colecciones como los Arrays y las Listas. Si queremos una lista con los nombres de los empleados de una empresa, entonces un objeto List del namespace System.Collections quizás sea el mejor recurso o quizás un objeto Array. Este tipo de decisiones se van tomando de acuerdo a los requerimientos de nuestra aplicación y las posibilidades brindadas por el objeto.

El namespace System del assembly system.dll es como el nodo principal de todos los namespaces contenidos en tal assembly y la instrucción **using** nos permite especificar una vez en nuestro programa que de allí en adelante cuando nos refiramos a determinados objetos serán estos de tal namespace.

```
using System;  
using System.IO;
```

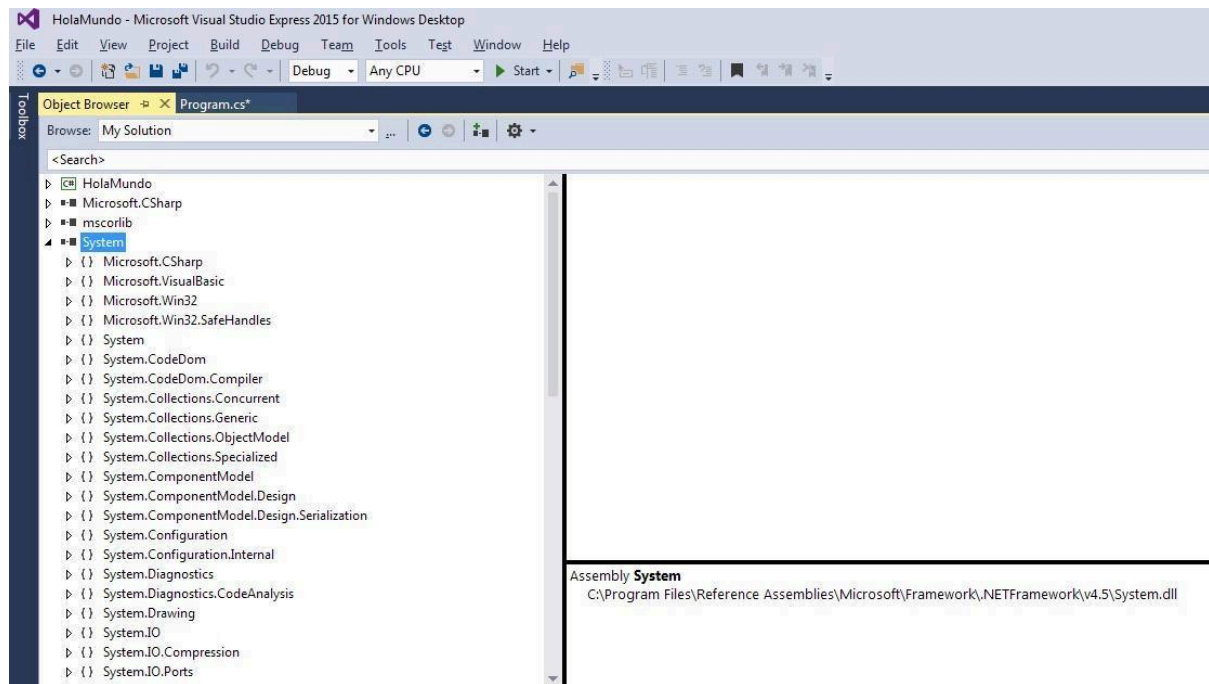
Con lo cual si queremos, por ejemplo, escribir un texto en pantalla y antes ya especificamos que usaremos el namespace System haríamos:

```
Console.WriteLine(“Hola mundo”);
```

Si no especificamos previamente el namespace que usaremos tendremos entonces que “arrastrar” en cada instrucción que haga uso de un objeto de dicho namespace, el nombre de tal namespace seguido de un . (punto)

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

En el ejemplo anterior Console es el objeto que maneja la consola dentro de un entorno Windows (consola es la ventana de comandos, el subyacente DOS). Como tal objeto (Console) se encuentra dentro de System, podemos hacer `System.Console.[Metodo]` para acceder al método apropiado (en nuestro caso es el método `WriteLine` el que nos permite desplegar un texto por pantalla). Si previamente especificamos un `using System;` luego no tendremos que referirnos a dicho namespace cada vez que queramos acceder a un objeto.



Aplicaciones de Consola.

La manera más fácil y recomendada de aprender el lenguaje C# es dominar su sintaxis básica y descubrir los diferentes namespaces escribiendo aplicaciones de consola, aplicaciones que corren dentro de una ventana de consola.

Iniciando en esta clase y extendiéndonos durante varias clases iremos creando diferentes aplicaciones de consola donde aprenderemos a dominar la sintaxis de C# y exploraremos diversos namespaces. Luego veremos que es aún más fácil escribir aplicaciones Windows.

Este paso genera el proyecto, el cual incluye un archivo con extensión **.cs** que es donde escribiremos el código fuente de C#.

Seguramente el entorno ya nos escribió algo de código en ese archivo (qué bueno no? ☺) pero por ser el primero borremos todos ese código y escribamos nosotros de nuevo lo siguiente (luego vamos a sacar provecho del código que nos genera el entorno para avanzar más rápido, pero por este primer ejemplo no).

```
using System;

namespace HolaMundo
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hola Mundo");
        }
    }
}
```

Analicemos línea por línea.

```
using System;
```

Lo primero que hacemos es indicar todos los namespaces que usaremos. Como ya dijimos el objeto Console que representa la consola pertenece al namespace System. Por eso indicamos que usaremos el namespace System.

```
namespace HolaMundo
{
    ...
    ...
    ...
}
```

También dijimos que en C# es común agrupar los objetos en namespaces y nuestra aplicación, que será un objeto, ¿por qué no puede estar en un namespace? Esto no es estrictamente obligatorio pero esta bueno hacerlo. Es por eso que encerramos la definición de la clase Program (nuestra clase) en un namespace, en este caso, llamado HolaMundo.

```
class Program
{
    ...
    ...
    ...
}
```

Nuestro programa es una clase y su nombre debe coincidir con el nombre del archivo **.cs**.

Veremos más sobre las clases en subsiguientes clases (valga la redundancia) pero por lo pronto aceptemos el hecho que nuestras aplicaciones son una clase también y de allí su definición.

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

```
...  
...  
}
```

Un programa tiene un método Main (sólo uno y siempre se llama Main). Este método es el punto de entrada de su ejecución. En este método escribiremos el código que queremos que se ejecute.

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

Nosotros queremos mostrar el texto “Hola Mundo” por pantalla, para eso el namespace System nos provee el objeto Console que tiene los métodos para interactuar con la consola.

WriteLine justamente recibe un parámetro string que identifica el texto a mostrar. Toda sentencia ejecutable de C# termina en ;

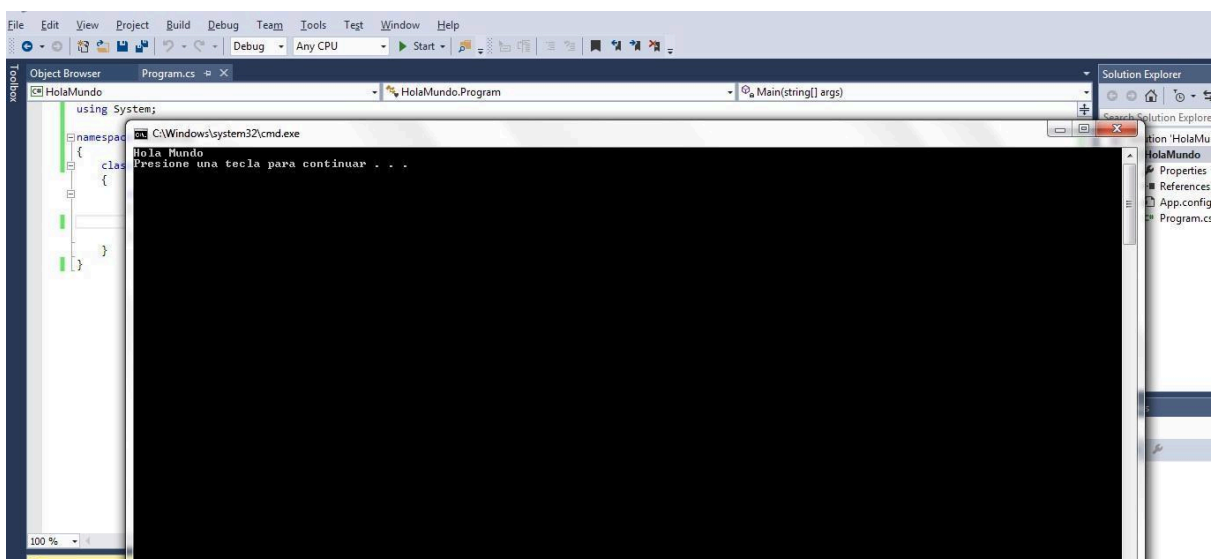
Construir y ejecutar una aplicación.

Una vez que escribimos el código debemos compilar, en realidad son dos pasos: compilar y linkar, en el menú Build ☐ Build Solution (o pulsando F7) se intentaran ejecutar estas dos acciones. Si esta todo bien una ventana al pie del entorno, la ventana Output, mostrará la siguiente leyenda:

```
===== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped =====
```

Esto es genial, significa que nuestro código se ha podido construir con éxito y un ejecutable dentro de la carpeta bin de nuestro proyecto se ha creado.

Hacemos entonces Ctrl+F5 y la aplicación se ejecutará:

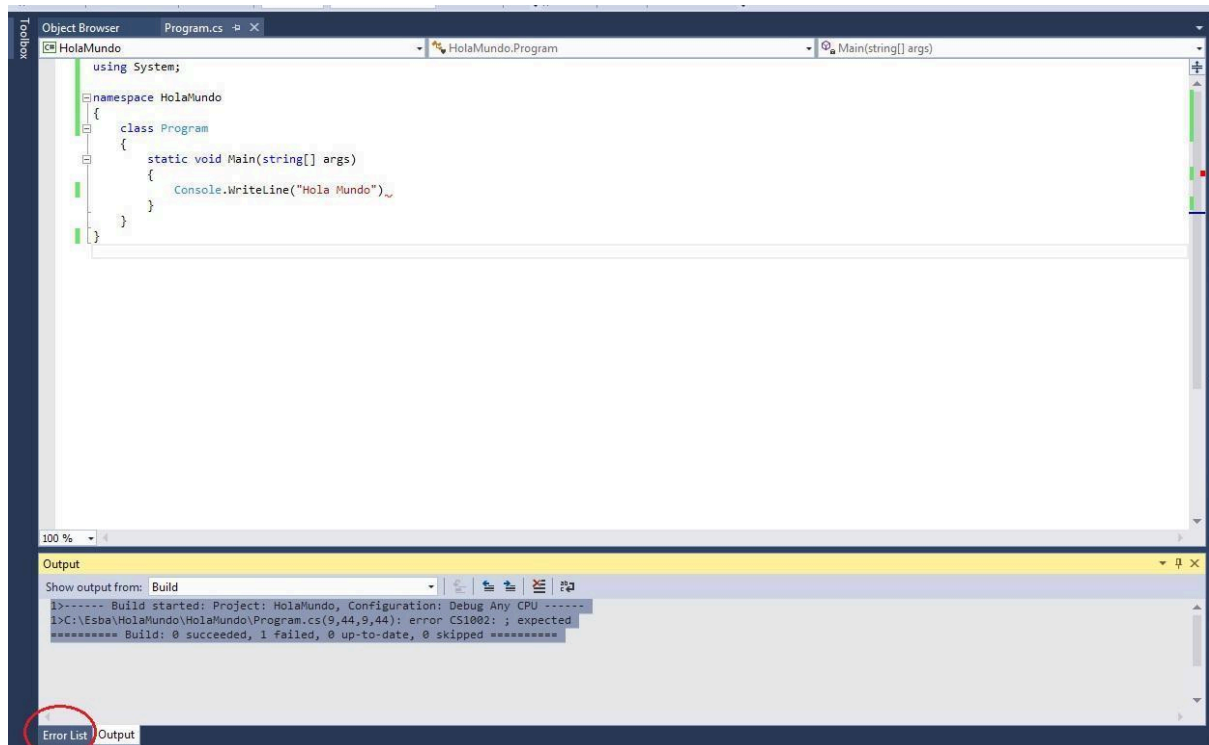


Resolviendo errores.

Como acabamos de ver la ventana Output nos informa si la aplicación pudo construirse con éxito o no.

Forcemos entonces un error en nuestra aplicación para ver cómo nos informa de los mismo.

Por ejemplo borren el ; final de la línea Console.WriteLine y pulsemos F7. La ventana Output nos informará del error:



```
1>----- Build started: Project: HolaMundo, Configuration: Debug Any CPU -----
1>C:\Esba\HolaMundo\HolaMundo\Program.cs(9,44,9,44): error CS1002: ; expected
===== Build: 0 succeeded, 1 failed, 0 up-to-date, 0 skipped =====
```

Claramente nos dice que falta un ;

Si accedemos a la solapa Error List (marcada con un círculo rojo en la figura) podremos ver esta información más detalladamente.

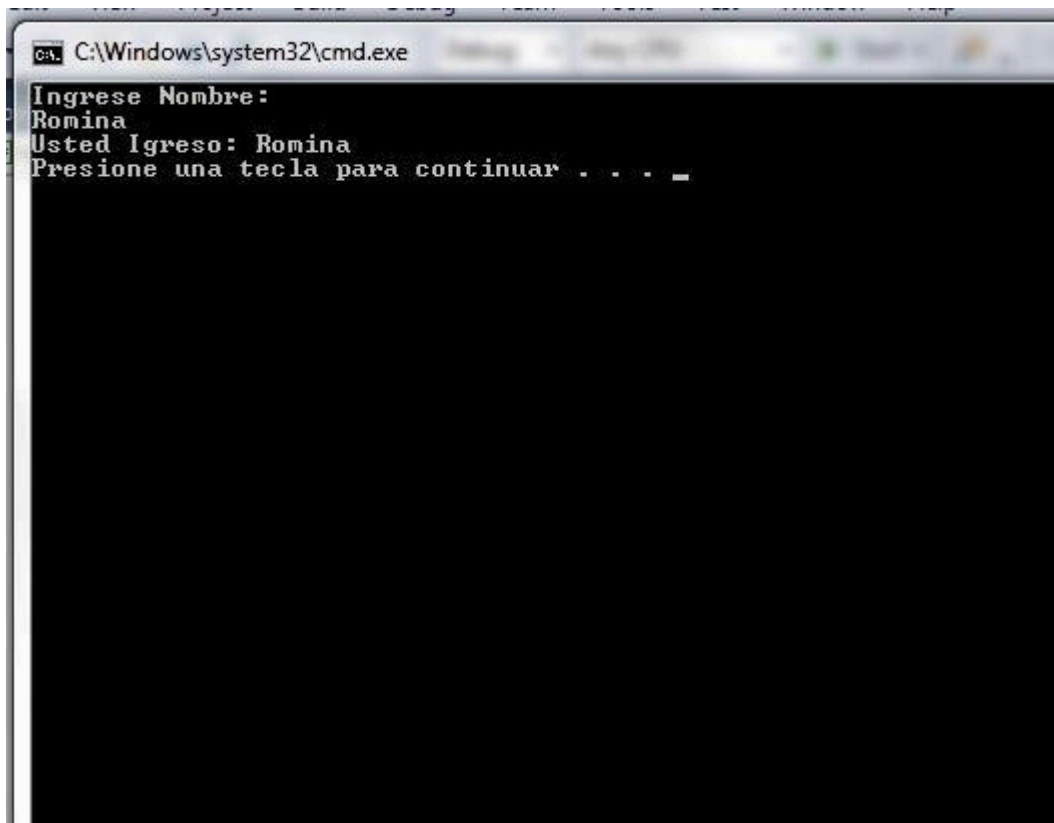
Sólo hay que agregar el ; faltante en el código e intentar nuevamente.

Ingreso por teclado, cadenas y números.

Ya dijimos varias veces que el objeto Console nos provee el acceso a la consola (la ventana del DOS subyacente en Windows). Si queremos aceptar input desde el teclado tenemos el método ReadLine que retorna un string con el texto ingresado. Por ejemplo, digamos que queremos aceptar el ingreso de un nombre por teclado para luego mostrarlo por pantalla, haríamos entonces:

```
Console.WriteLine("Ingreso Nombre: ");  
string strNombre;  
strNombre = Console.ReadLine();  
Console.WriteLine("Usted Ingreso: " + strNombre);
```

Esto daría lugar a una salida como:



```
C:\Windows\system32\cmd.exe  
Ingreso Nombre:  
Romina  
Usted Ingreso: Romina  
Presione una tecla para continuar . . . _
```

Nota: para seguir este ejemplo pueden borrar en nuestro ejemplo anterior la línea `Console.WriteLine("Hola Mundo")` y escribir estas líneas o mejor pueden crear un proyecto de consola nuevo. Eso espero que vayan manejándolo ustedes solos así toman confianza con el entorno en lugar de comandar yo desde aquí todo paso a paso.

string [nombre de variable];

Es la forma que tienen de declarar variables que acepten cadenas. Los tipos de datos que pueden manejar los vimos en la clase 1, pero revisemos algunos básicos que nos permitan hacer algunos ejercicios ahora.

int [nombre de variable]; □ cuando quieren una variable de tipo entera.

float [nombre de variable]; □ cuando quieren una variable flotante de simple precisión.

double [nombre de variable]; □ cuando quieren una variable flotante de doble precisión.

Estos tipos de datos son además objetos, con lo cual tienen métodos. Algunos de ellos que nos pueden resultar útiles son:

int.Parse([variable string]) □ convierte una variable string a entero, por ejemplo:

```
int x;  
string strCadena = "150";  
x = int.Parse(strCadena);
```

Esto es particularmente útil cuando se ingresan números por teclado usando Console.ReadLine() ya que el número ingresado esta en formato string

```
int x;  
string strCadena;  
strCadena = Console.ReadLine();  
x = int.Parse(strCadena);
```

[variable].ToString() □ Devuelve una representación de tipo cadena de la variable.

```
int x = 5212;  
Console.WriteLine(x.ToString());  
  
double x = 20.85;  
Console.WriteLine(x.ToString());
```

Actividades:

A continuación algunas pequeñas aplicaciones de consola para que vayan ejercitando la sintaxis y conociendo el entorno (recuerden que cada uno de estos ejercicios es un proyecto de consola nuevo).

- 1) Escribir un programa que pida el ingreso de cinco nombres de personas y sus edades. El programa debe mostrar el promedio de edad de las personas ingresadas.
- 2) Escribir un programa que calcule el área de un círculo. Se ingresa el radio del círculo por teclado. Puede usar la constante PI de la clase Math, haciendo Math.Pi.

La formula del área de un círculo es: $PI \times (Radio)^2$

Para calcular el cuadrado del radio pueden hacer Radio x Radio o bien usar el método Pow de Math, haciendo:

Math.Pow(Radio, 2) □ devuelve el resultado de Radio elevado al cuadrado.

- 3) Una estación de servicio se encuentra ubicada en la ruta 51. No existe otra estación expendedora de combustible en 200 km a la redonda. Escriba un programa que permita averiguar si un vehículo necesita combustible para llegar a la próxima estación. El programa pide la siguiente información:

- La capacidad de combustible del tanque.

- Cuánto indica la aguja del marcador de combustible en porcentaje.
- Los kilómetros por cada litro de consumo.

El programa debe imprimir en pantalla “Necesita combustible” o “Combustible suficiente” dependiendo si el auto puede o no transitar los 200 km con el combustible existente en el tanque.

- 4) “Cacho y Marta Delicatessen” (altas confituras) desean un programa para tomar pedidos. El programa pide por un producto y su precio. Además se cobra envío de acuerdo a la siguiente consideración: pedidos menores a \$20 es de \$2, superiores a \$20 el costo es de \$3. Si el pedido se realiza después de medianoche se le suman \$5 más.

Ejemplo de ejecución del programa:

```
Ingrese producto:
Pastafrola de dos pisos.
Precio:
12.00
Despues de medianoche (0==no, 1==yes)
1
```

Factura:	
Pastafrola dos pisos	12.00
Envío	7.00
Total	11.50