**LENGUAJE DE PROGRAMACION C#**

Barreda Zúñiga, Carlos

Lace Tello, Fidel

Febrero 2018

Universidad Científica del Sur

Facultad de Ingeniería de Sistemas y Gestión.

Lenguaje de Programación I

**Dedicatoria**

Dedicamos este trabajo a amigos y profesor, quien nos apertura al conocimiento de la programación Java y diversos lenguajes orientados a objetos, asi como las diversas tecnologías usadas hoy en día para un buen desarrollo de código eficiente en la programacion.

**Abstract**

El presente trabajo, tiene por objetivo, conocer brevemente el lenguaje de programación C# y su aplicación mediante ejemplos vistos en clase. Asimismo, plantea ofrecer una visión rápida de la amplia adopción como alternativa al lenguaje Java.

**Tabla de Contenidos**

[**Capítulo 1**Introducción](#_gjdgxs)5

[Breve](#_26in1rg) historia de la programacion en Windows 6

.NET 7

Caracteristicas del lenguaje 9

Clases 9

Tipos de dato 9

Funciones 11

Variables 11

JIT 11

[**Capitulo**](#_2lw5pt25u167) **2 11**

El ambiente de desarrollo 11

El metodo main 15

Arquitectura de diseño en 3 capas 22

[**Capítulo 3** Conclusiones.](#_ihv636)23

[Lista de referencias](#_32hioqz)24

# Capítulo 1 Introducción

La programación de computadoras es una actividad fascinante, el poder hacer una aplicación a la medida y de acuerdo a nuestras necesidades es una actividad interesante y llena de recompensas. Mucha gente llega a pensar que solamente con aprender la sintaxis de un lenguaje de programación es suficiente para poder hacer programas, pero se equivocan. Aún más importante que la sintaxis del lenguaje es la técnica de resolución del problema y el análisis adecuado. El tener buenos hábitos de análisis y programación desde el inicio es fundamental para convertirnos en buenos programadores. Una de los lenguajes que está adquiriendo gran popularidad es C#, un lenguaje sencillo, amigable y poderoso. Con C# se pueden crear todo tipo de aplicaciones, desde programas de consola para Windows y para páginas web en unión con ASP, hasta video juegos para Xbox 360 con XNA.

El convertirse en un buen programador requiere de mucha práctica y experimentación, pero también de una guía adecuada que vaya construyendo paso a paso las habilidades necesarias. Esperamos que este libro sea de utilidad a todos aquellos que, aun sin experiencia en programación, tengan el deseo de aprender cómo hacer sus propios programas.

C# o C Sharp es un lenguaje de programación que está incluido en la Plataforma .NET y corre en el Lenguaje Común en Tiempo de Ejecución (CLR, Common Language Runtime). El primer lenguaje en importancia para el CLR es C#, mucho de lo que soporta la Plataforma .NET está escrito en C#. C# intenta ser el lenguaje base para escribir aplicaciones .NET C# deriva de C y C++, es moderno, simple y enteramente orientado a objetos, simplifica y moderniza a C++ en las áreas de clases, namespaces, sobrecarga de métodos y manejo de excepciones. Se elimino la complejidad de C++ para hacerlo más fácil de utilizar y menos propenso a errores. C# es “case sensitive”, es decir, que distingue mayúsculas de minúsculas. HolaMundo es diferente a holamundo.

**Breve historia de la programación en Windows**

Hace algunos años la única forma como se podía programar para Windows era hacer uso de un compilador de C o C++ y de un **API** de Windows. El API es una gran colección de funciones que se relacionan, las que nos permiten comunicarnos con el sistema operativo. Por medio del API de **Win32** se programaban las ventanas, botones y demás elementos. El problema de este tipo de programación es que el API de Win32 es realmente complejo

y enorme. Con miles de funciones en su interior, por lo que pocos programadores podían conocerlo en su totalidad. Pero la complejidad no solamente estaba en la cantidad de funciones, también en la sintaxis y la forma como se programa.

Para facilitar la programación de aplicaciones para Windows surgen diferentes opciones; la finalidad de estos intentos era poder hacer las aplicaciones sin tener que pasar por la complejidad de Win32. Uno de estos intentos fue conocido como **OWL**; sin embargo, obtuvo más éxito **MFC**, creado por Microsoft. MFC es un conjunto de clases que envuelve a Win32 y facilita su programación. Con MFC los procesos más comunes se agrupan en funciones de tal forma que con una simple llamada a una función de MFC se puede hacer una determinada tarea, para la que antes necesitábamos por lo menos 10 llamadas en Win32 y muchos parámetros. Sin embargo, Win32 está debajo de MFC; la programación MFC simplifica mucho las cosas, pero muchos programadores que venían del paradigma de programación estructurada no se sentían a gusto con él.

Otra de las opciones que surgieron es **Visual Basic**, este lenguaje logró gran popularidad, especialmente en Latinoamérica. Visual Basic también trabaja por arriba de Win32, pero basa su sintaxis en el antiguo lenguaje **Basic**. Es muy sencillo de aprender y una de las características que le dio gran popularidad fue la facilidad con la que se podían crear interfaces de usuario y conectividad a bases de datos. Pero hasta antes de la versión .NET, este lenguaje tenía ciertas limitantes ya que no se podía llevar a cabo programación orientada a objetos con él.

Otro lenguaje que surge, pero con su propio **Framework**, es **JAVA**; su principal ventaja es ser multiplataforma. Una de sus características es el uso de un **runtime**, la aplicación en lugar de correr directamente en el microprocesador, se ejecuta en un programa llamado runtime y este se encarga de ejecutar el código en el microprocesador correspondiente. Si se tiene el runtime para Windows, sin problema se ejecuta el programa de JAVA.

Cuando nosotros deseábamos tener un programa que se pudiera ejecutar, era necesario **compilarlo**. Cada uno de los lenguajes tenía su propio **compilador**, por ello no era sencillo poder compartir código de C++ con código de Visual Basic ya que el traducir entre lenguajes era difícil. Para poder compartir código entre los lenguajes surge un modelo conocido como **COM**, éste nos permite crear componentes binarios, esto quiere decir que es posible programar un componente en Visual Basic y un programador de C++ puede tomarlo y hacer uso de él. Esto se debe a que el componente ya es código compilado y no código fuente en el lenguaje de origen; la programación de COM también

tenía sus complejidades y surge **ATL** para ayudar en su desarrollo.

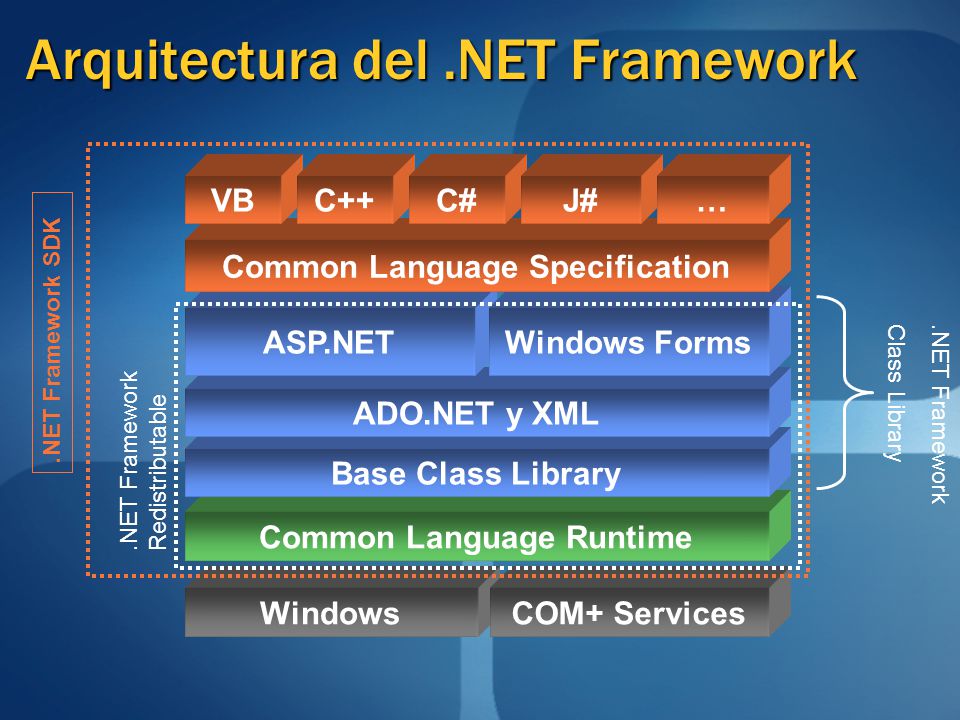
Con todo esto, llega el momento en el cual es necesario ordenar, facilitar y organizar

el desarrollo de las aplicaciones para Windows, con esta filosofía surge .NET.

**.NET**

El **Framework** de **.NET** es una solución a toda la problemática en torno al desarrollo de aplicaciones, brinda grandes beneficios no solamente al desarrollador, sino también al proceso de desarrollo. En primer lugar, .NET permite trabajar con código ya existente, podemos hacer uso de los componentes COM, e incluso, si lo necesitáramos usar el API de Windows. Cuando el programa .NET está listo es mucho más fácil de instalar en la computadora de los clientes, que las aplicaciones tradicionales ya que se tiene una integración fuerte entre los lenguajes.

Un programador de C# puede entender fácilmente el código de un programador de Visual Basic .NET y ambos pueden programar en el lenguaje con el que se sienten más cómodos. Esto se debe a que todos los lenguajes que hacen uso de .NET comparten las librerías de .NET, por lo que no importa en qué lenguaje programemos, las reconocemos en cualquiera.



El lenguaje de programación C# fue creado por el danés Anders Hejlsberg que diseño también los lenguajes Turbo Pascal y Delphi. El C# (pronunciado en inglés “C sharp” o en español “C sostenido”) es un lenguaje de programación orientado a objetos. Con este nuevo lenguaje se quiso mejorar con respecto de los dos lenguajes anteriores de los que deriva el C, y el C++.

Con el C# se pretendió que incorporase las ventajas o mejoras que tiene el lenguaje JAVA. Así se consiguió que tuviese las ventajas del C, del C++, pero además la productividad que posee el lenguaje JAVA y se le denomino C#.

**Características del lenguaje**

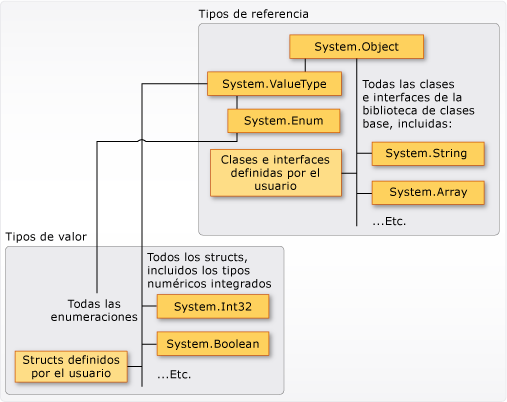
* Su código se puede tratar íntegramente como un objeto.
* Su sintaxis es muy similar a la de JAVA.
* Es un lenguaje orientado a objetos y a componentes.
* Armoniza la productividad del Visual Basic con el poder y la flexibilidad del C++.
* Ahorramos tiempo en la programación ya que tiene una librería de clases muy completa y bien diseñada.

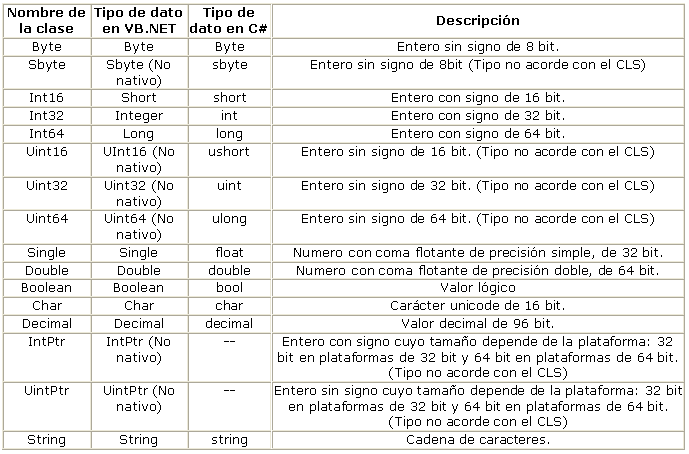
**Clases**

Todo el código y los datos en C# deben ser incluidos en una clase. Las clases pueden tener constructores, que se ejecutan cuando se crea un objeto de la clase y un destructor, que se ejecuta cuando un objeto de la clase es destruido. Las clases admiten herencias simples y todas las clases derivan al final de una clase base llamada objeto.

**Tipo de datos**

C# permite trabajar con dos tipos de datos: de valor y de referencia. Los de valor contiene calores reales. Los de referencia contienen referencias a valores almacenados en algún lugar de la memoria.





**Funciones**

Una función es un fragmento de código que puede ser invocado y que puede o no devolver un valor al código que lo invoco en un principio. Una función suele asociarse a fragmentos de código que devuelven información, mientras que un método no suele devolver información. Las funciones pueden tener 4 tipos de parámetros:

* Parámetros de entrada: tiene valores que son enviados a la función, pero la función no puede cambiar esos valores.
* Parámetros de salida: no tienen valor cuando son enviados a la función, pero la función puede darles un valor y enviar el valor de vuelta al invocador.
* Parámetros de referencia: introducen una referencia en otro valor. Tiene un valor de entrada para la función y ese valor puede ser cambiado dentro de la función.
* Parámetros Params: definen un numero variable de argumentos en una lista.

**Variables**

Pueden ser definidas como constantes. Las constantes tienen valores que no pueden cambiar durante la ejecución del código. Se declaran el tipo de dato que almacenara y su identificador. Por ejemplo: el valor de pie es una buena muestra de una constante porque el valor no cambia a medida que el código se ejecuta.

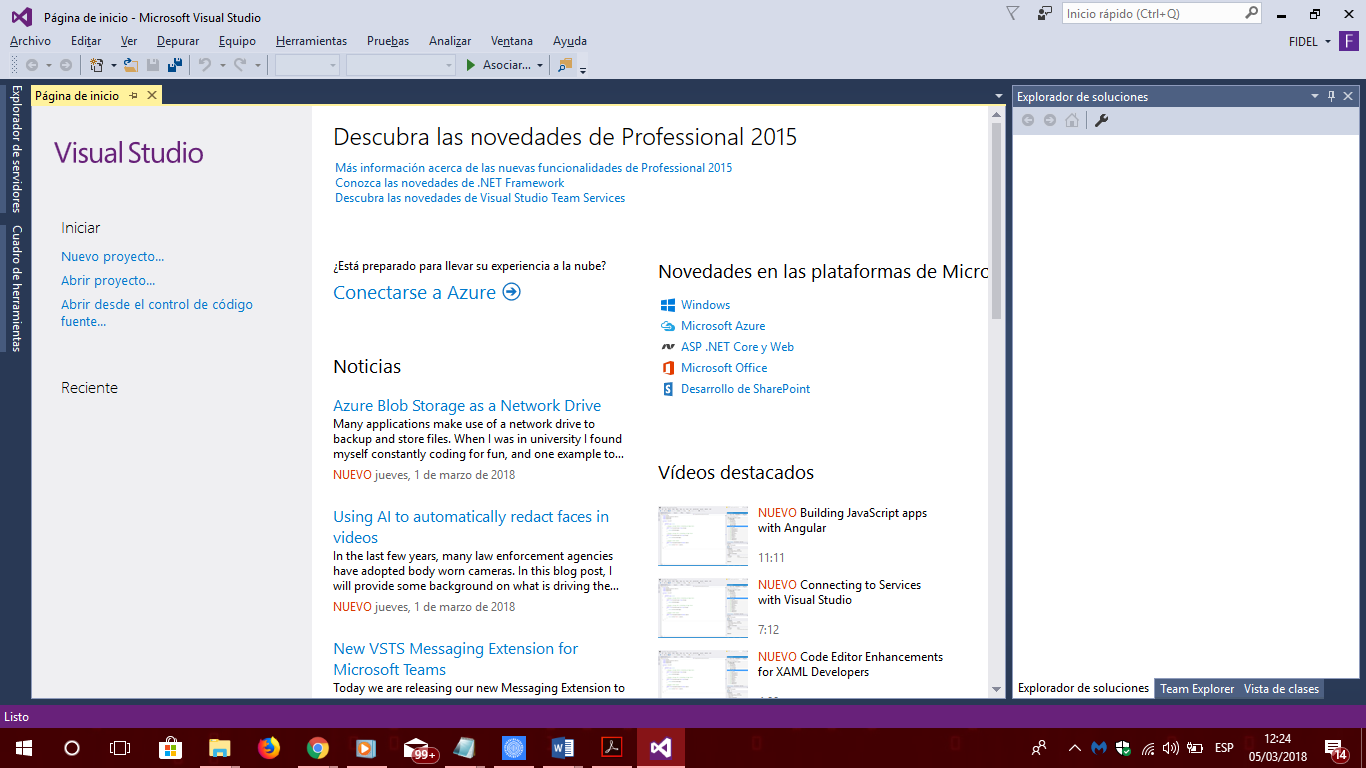
**El compilador JIT**

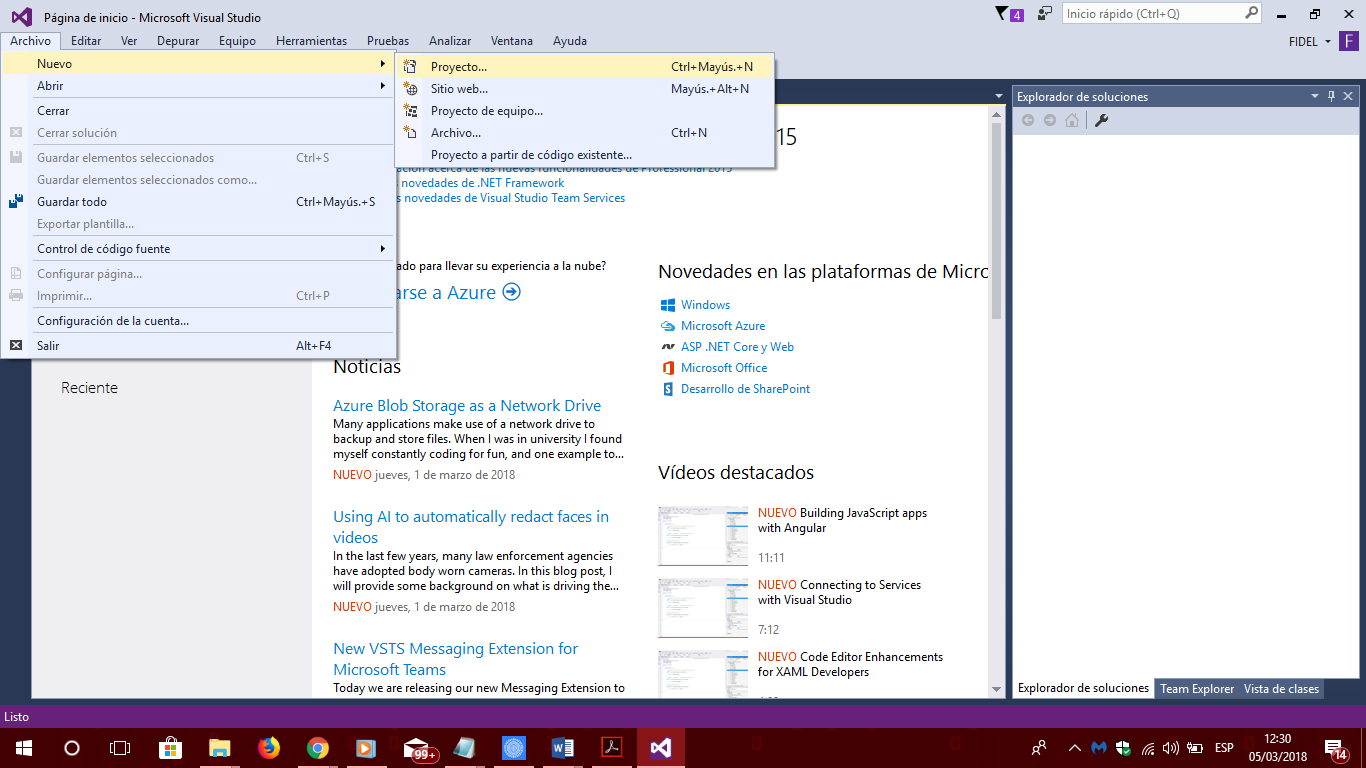
Al **compilador JIT** también se le conoce como **Jitter**. Forma parte del runtime y es muy eficiente, si el programa necesita volver a ejecutar un código que ya se ha compilado, el Jitter en lugar de volver a compilar, ejecuta lo ya compilado, mejorando de esta forma el desempeño y los tiempos de respuesta de cara al usuario.

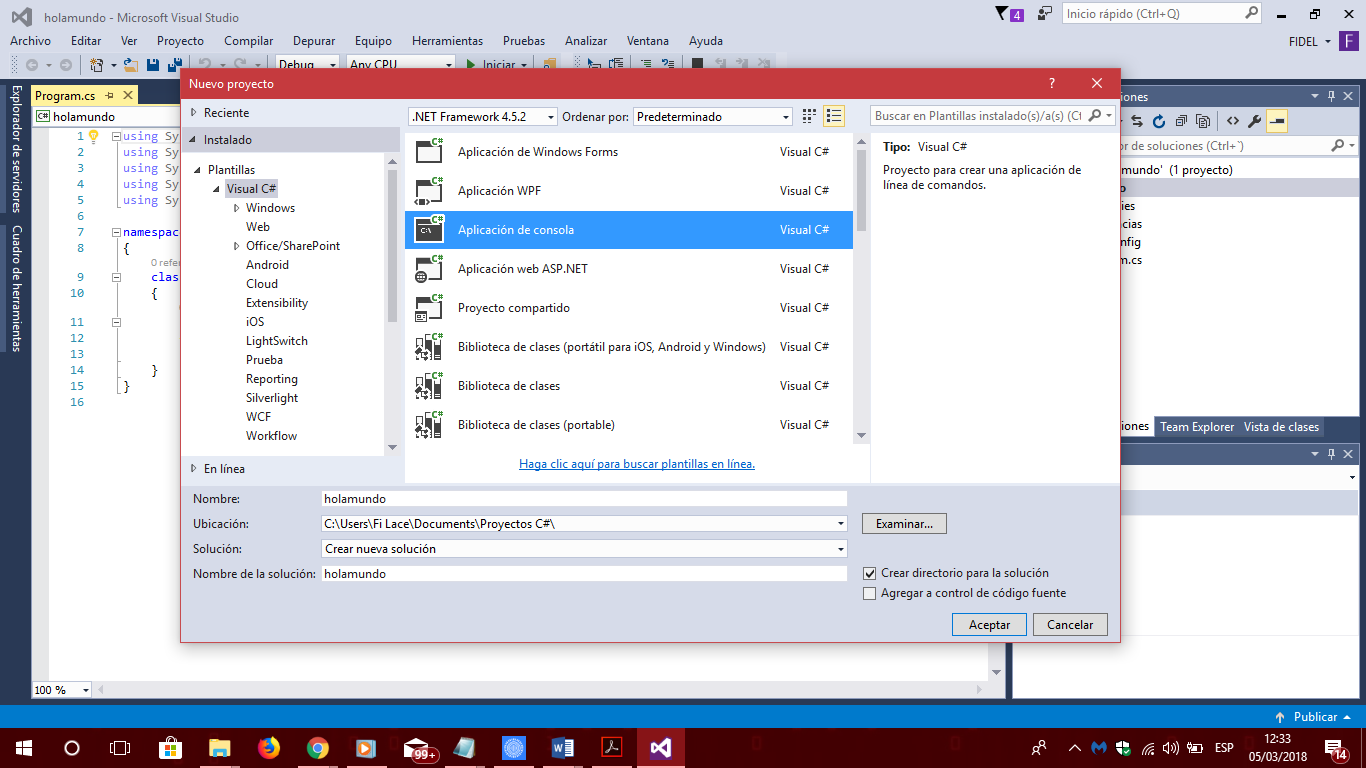
**Capítulo 2**

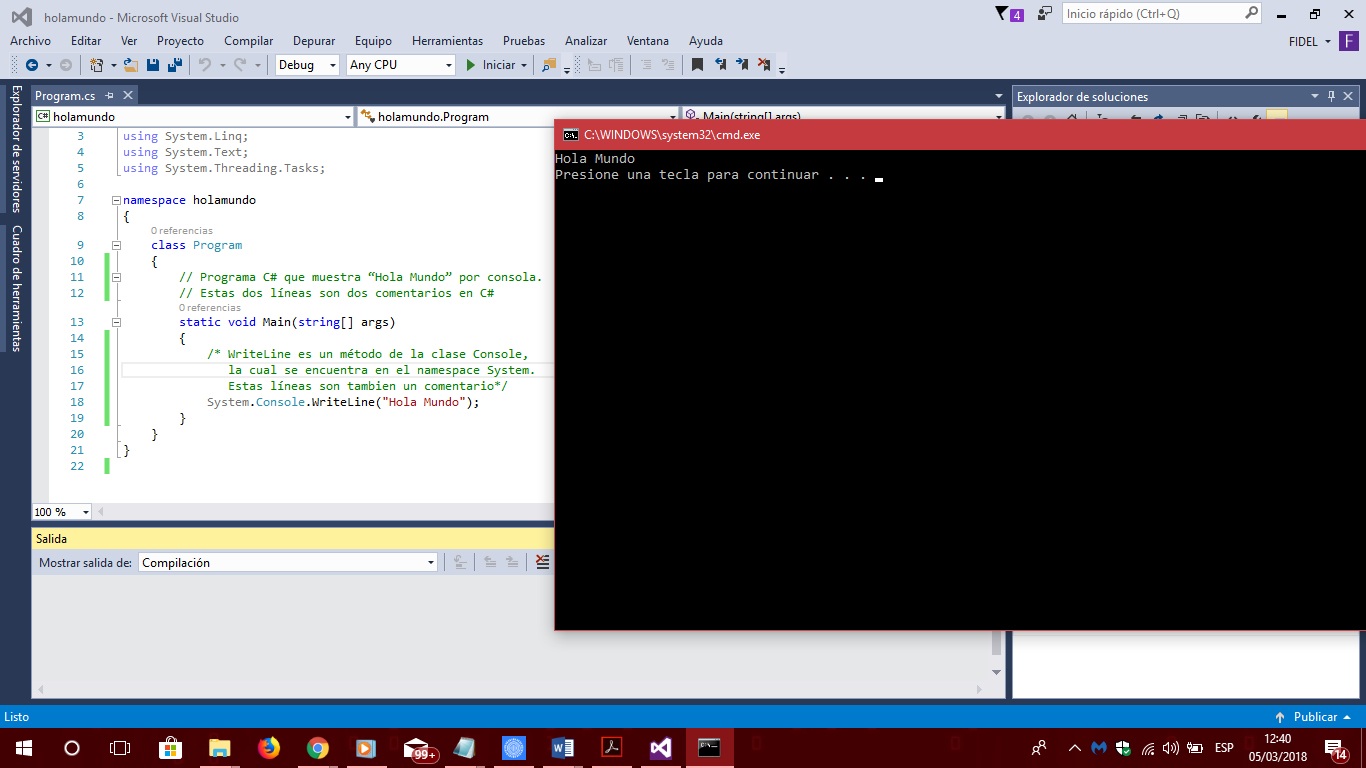
**El ambiente de desarrollo**

Una vez finalizada la instalación podemos iniciar el programa seleccionándolo desde el menú **Inicio** de Windows, veremos una ventana como la que aparece a continuación, que muestra la interfaz de uso de la aplicación.





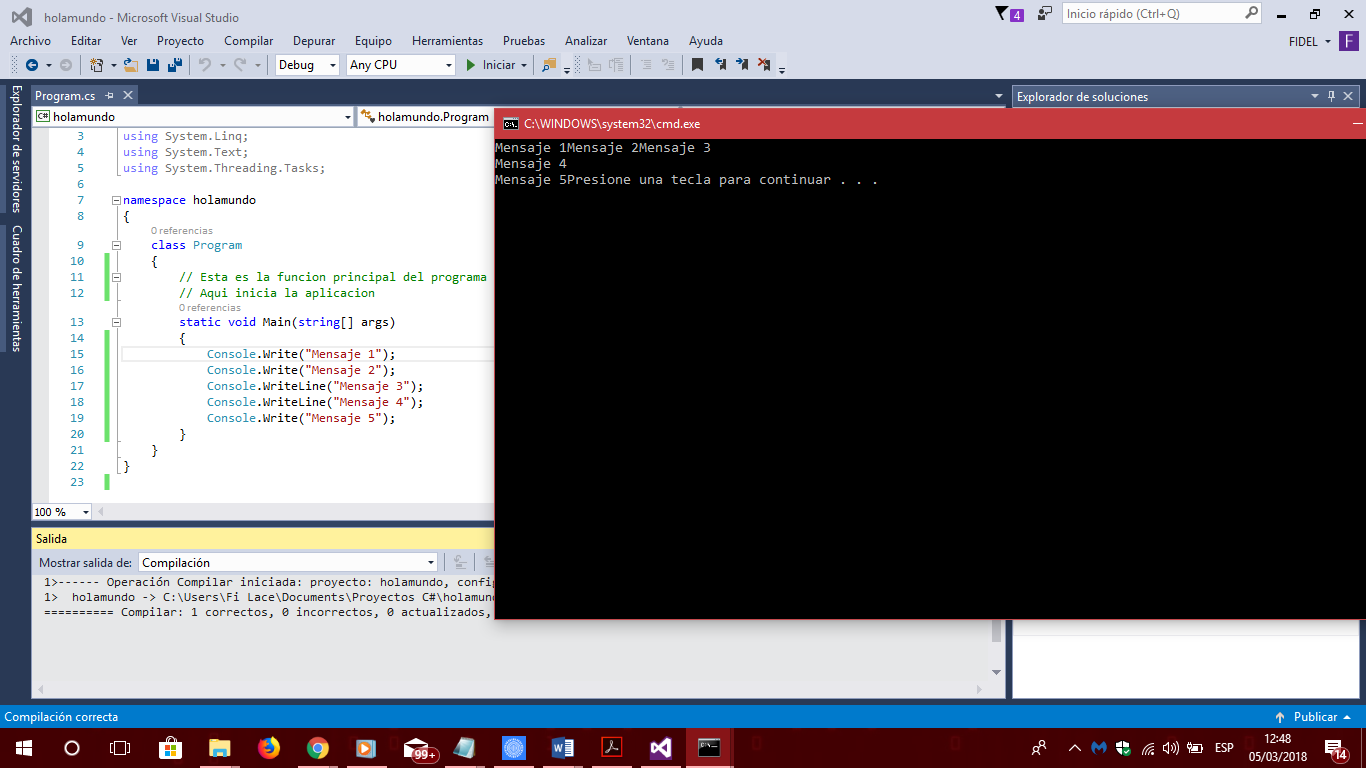




**Diferencia entre los método Write() y Writeline()**

Hemos visto que el método **WriteLine()** nos permite mostrar un mensaje en la consola, pero existe un método similar que se llama **Write()**. La diferencia es muy sencilla.

Después de escribir el mensaje **WriteLine()** inserta un salto de línea, por lo que lo próximo que se escriba aparecerá en el renglón siguiente. Por su parte, el método **Write()** no lleva a cabo ningún salto de línea y lo siguiente que se escriba será en la misma línea. Veamos todo esto en el siguiente ejemplo:



**El método Main()**

Todas las aplicaciones escritas en C# deben constar de un método llamado Main(). Un método es un conjunto de instrucciones que realizan una acción. Este método puede devolver información a la sección de código que lo llamo, pero en determinadas circunstancias no es necesario que lo haga.

**Estructuras Selectivas**

Las **estructuras selectivas** son aquellas que nos permiten hacer una selección entre dos o varias rutas de ejecución posibles. La selección se llevará a cabo según el valor de una expresión. Esta expresión puede ser una expresión relacional.

**El uso de if**

Nuestra primera estructura selectiva se conoce como **if**, que es un **si condicional**. Si tal cosa sucede, entonces haz tal cosa. El uso del **if** es sencillo:

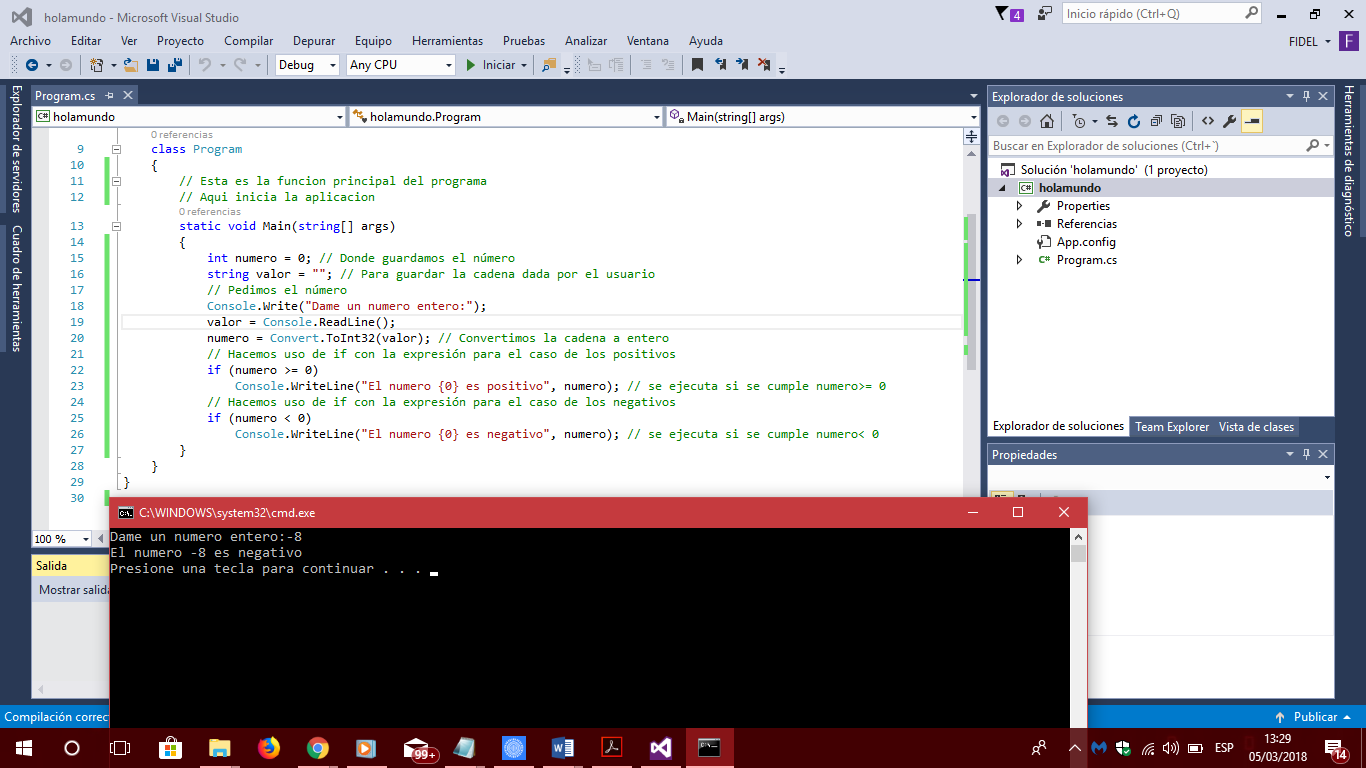
**if(expresión)**

**Sentencia a ejecutar**

El uso de **if** requiere que coloquemos una expresión a evaluar entre paréntesis. Si el resultado de esta expresión es **true**, entonces la sentencia a ejecutar se lleva a cabo. Si el resultado de la evaluación es **false**, entonces la sentencia a ejecutar nunca se lleva

a cabo, o dicho de otra forma, es ignorada.

Veamos un primer ejemplo donde nos puede servir **if** y las expresiones. Crearemos un programa que le pida al usuario un número, y la computadora debe decir si el número es positivo o negativo.



**El uso de else**

Con los programas anteriores con los que estuvimos trabajando, donde utilizamos la sentencia **if**, hemos visto que podemos indicar que determinada sentencia se ejecute cuando deseamos que una condición se cumpla. Sin embargo, puede ocurrir que a veces necesitamos que una sentencia se ejecute cuando se cumple la condición y que otra sentencia se ejecute cuando esa condición no se cumpla. Si nos encontramos en este caso, una forma de resolverlo sería colocar un **if** con una condición y luego otro **if** con una condición que sea complemento de la primera. Esto ya lo hemos hecho anteriormente en el programa de los números positivos y negativos. Sin embargo, existe otra forma más sencilla que la utilizada en este programa para hacerlo. Esta forma puede sernos de mucha utilidad para simplificar la lógica y que no nos veamos obligados de tener que colocar tantos **if** dentro de una sentencia. Para lograr esto haremos uso de **else**.

Siempre utilizaremos **else** a continuación de una sentencia **if**, ya que **else** no se puede usar sólo. Tenemos que colocar nuestro código de la siguiente forma:

**if(condición)**

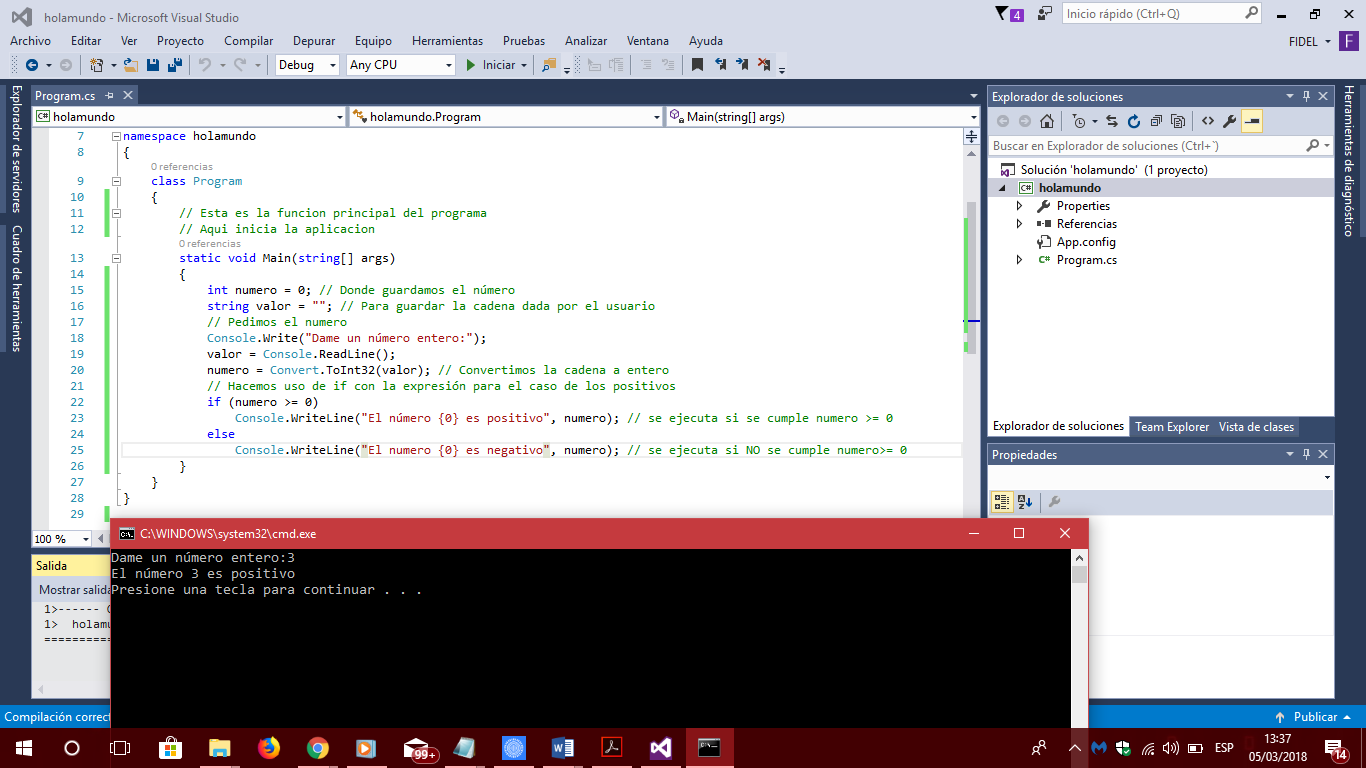
**Sentencia1;**

**else**

**Sentencia2;**

# 

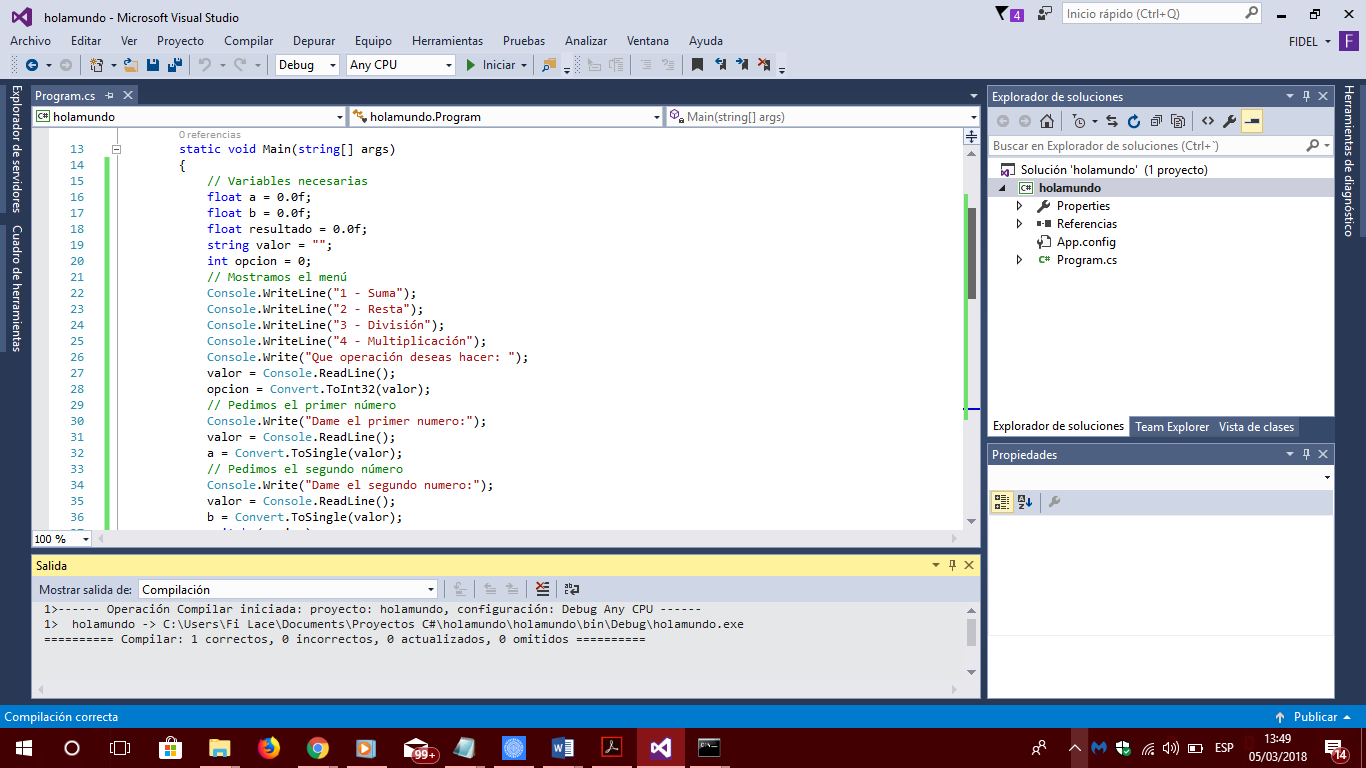
Si la condición es evaluada como verdadera, entonces se ejecuta la sentencia 1, en cambio, cuando la condición se evalúa como falsa, se ejecuta la sentencia 2. A continuación analizaremos cómo hacer uso de este caso para simplificar nuestro código. Para esto elegimos para representar este ejemplo el programa de los números positivos y negativos.

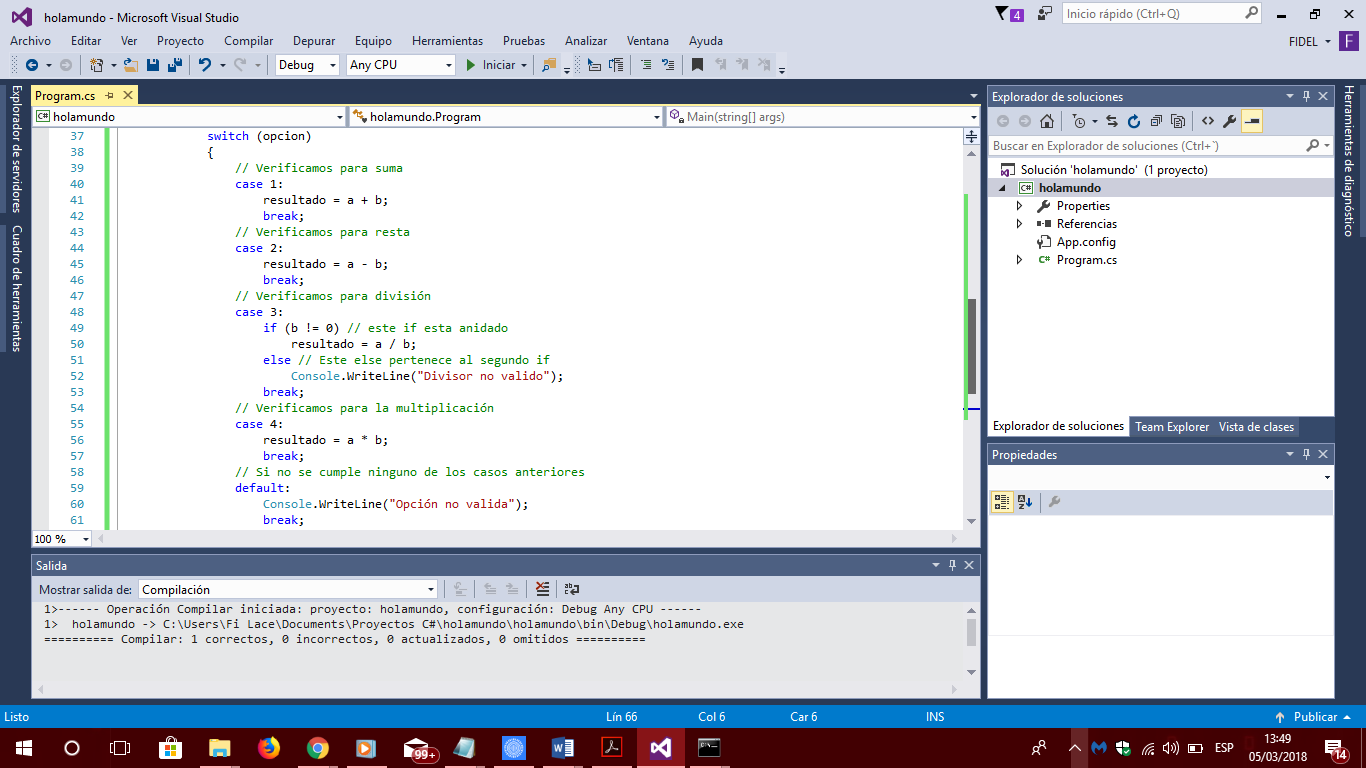


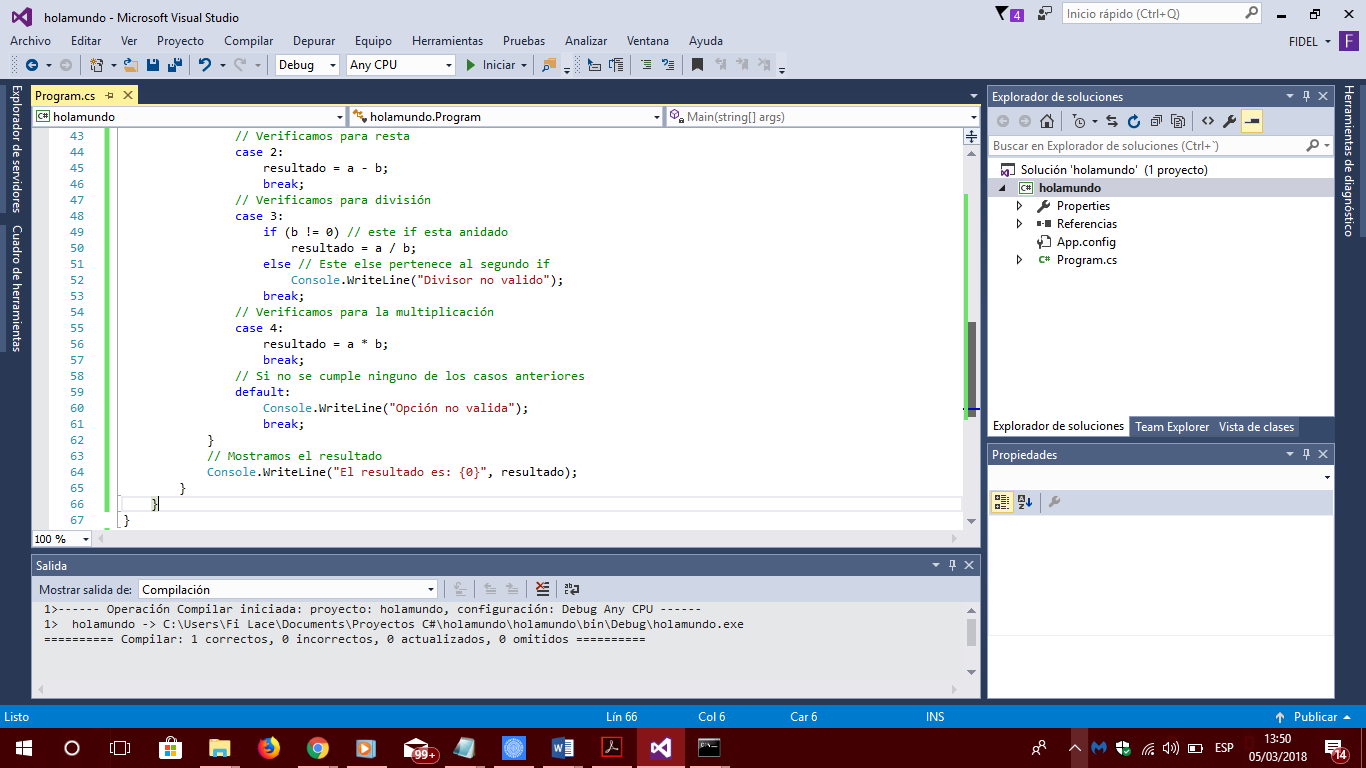
**El uso del switch**

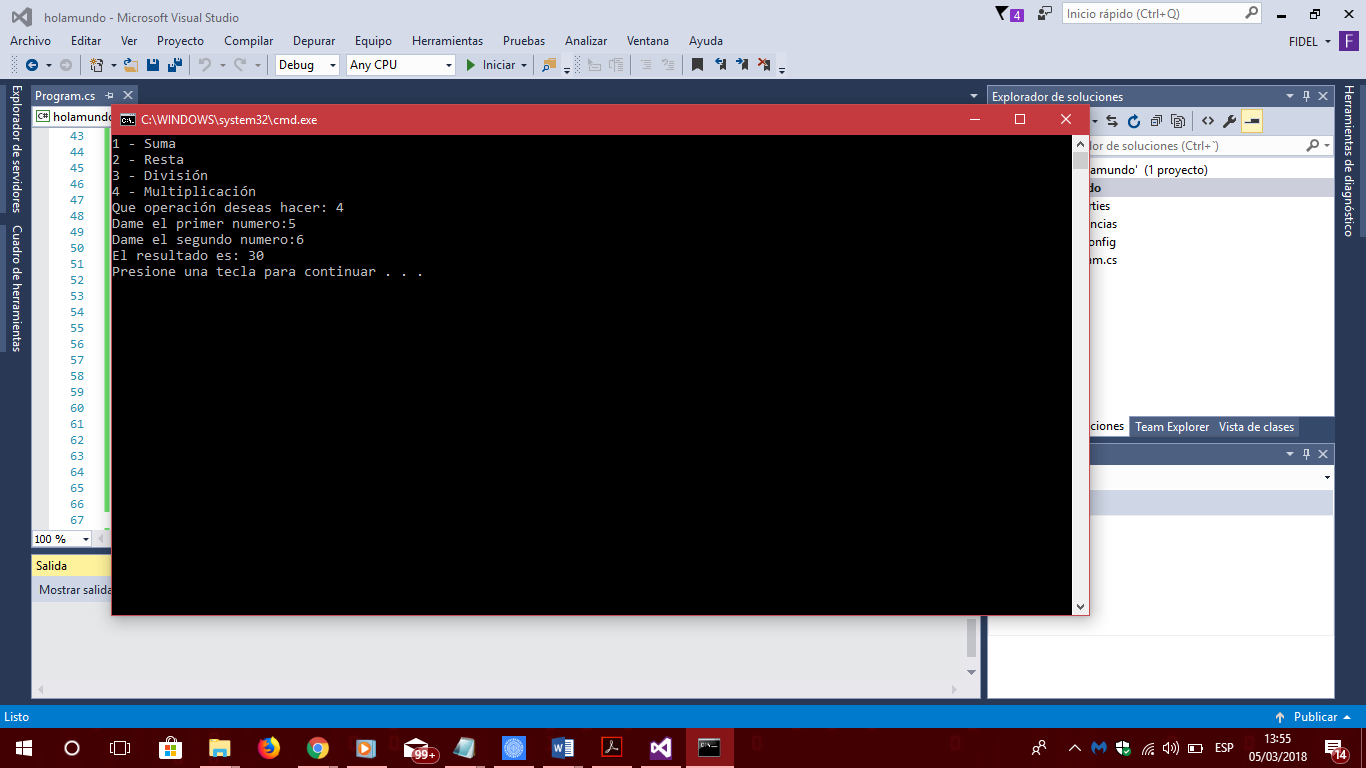
En varias ocasiones veremos que es necesario tomar diferentes opciones dependiendo del valor de una variable. Como esto puede ocurrir frecuentemente, tenemos una estructura selectiva que nos ayuda y nos permite colocar el código para estos casos con facilidad. Esta estructura se conoce como **switch**. Para ésta necesitamos una variable y varios casos.

Como ejemplo usaremos las operaciones aritméticas para que haga uso del **switch**. La variable de comparación será la variable **opción**, ya que el valor de ésta será comparado para cada operación.









En la vida real a veces repetimos la misma actividad muchas veces. También en los programas de computadora veremos que es necesario repetir algo un número de veces. Para esto haremos uso de los ciclos, que harán que nuestro código sea más sencillo al mismo tiempo que podemos repetir una actividad cuantas veces sea necesario.

**El ciclo for**

Cuando tenemos código que se debe repetir un número de veces podemos utilizar el ciclo **for**. Éste nos permite repetir la ejecución de un código un número **determinado** de veces.

**for( inicializacion; condicion; incremento)**

**código**

El ciclo **for** necesitará una o varias **variables de control**, aunque en la mayoría de los

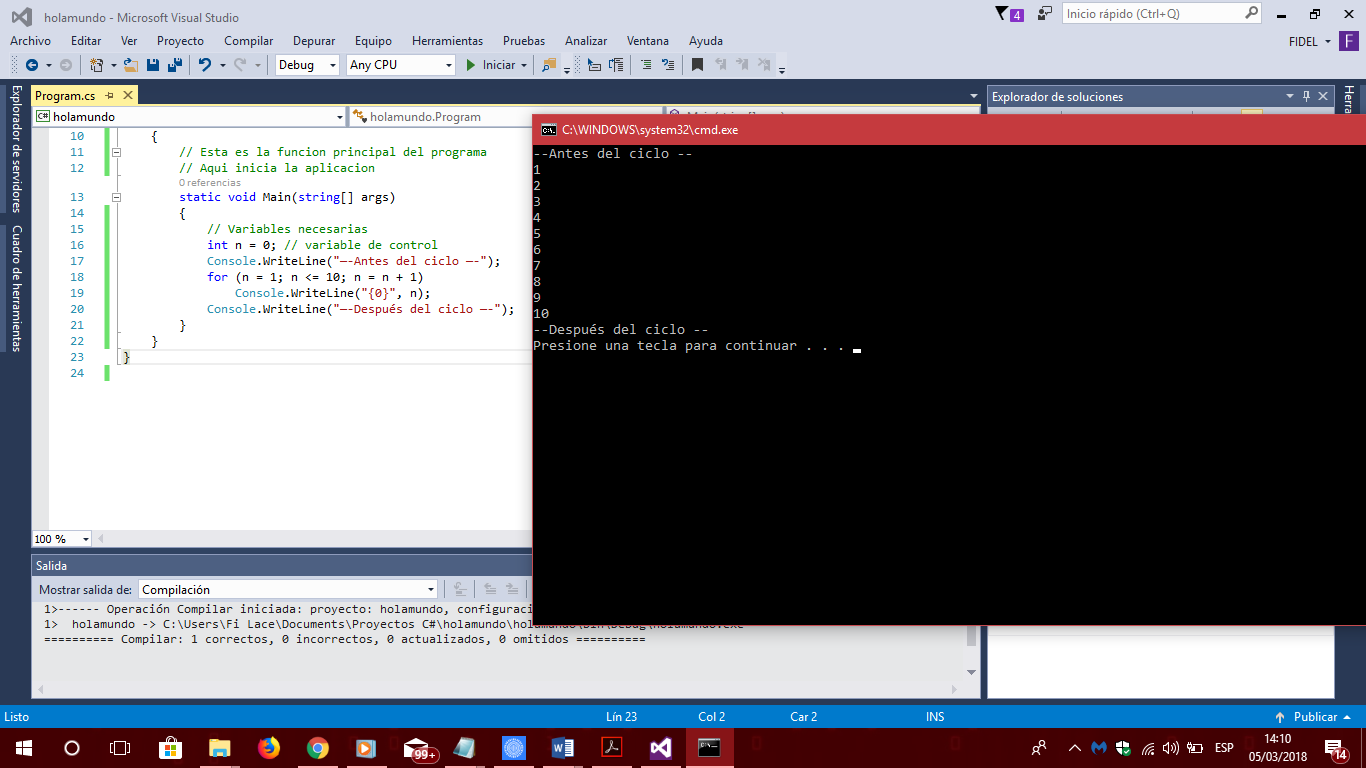
casos usaremos únicamente una variable.

En primer lugar, encontramos la **inicialización**. En esta sección le damos a la variable de control su valor inicial. El valor inicial se lleva a cabo por medio de una **asignación** normal, tal y como las que hemos trabajado. La segunda sección lleva una **condición** en forma de una expresión relacional. Ésta nos sirve para controlar cuándo termina el ciclo y generalmente aquí indicamos la cantidad de vueltas que da el ciclo.

Luego tenemos el **código**. En esta sección colocamos la parte del código que deseamos que se repita. Esta sección puede ser una sentencia o un bloque de código. Por último, se ejecuta la sección del **incremento**. Aquí indicamos cómo se modificará el valor de la variable de control para cada vuelta del ciclo.

Cuando se ejecuta el ciclo **for**, lo primero que se lleva a cabo es la inicialización y luego la condición. Si la condición es verdadera, entonces se pasa al código y después al incremento. Seguido del incremento vamos nuevamente a la condición y se repite el proceso. Si la condición no se cumple, entonces decimos que el ciclo finaliza, se salta al código y continúa la ejecución del programa con lo que sea que encontremos después del ciclo.

Para iniciar con facilidad, queremos que el ciclo se lleve a cabo diez veces. Primero veamos el programa y luego analicemos cada una de las partes del ciclo. Nuestro programa queda de la siguiente forma:



**El ciclo do while**

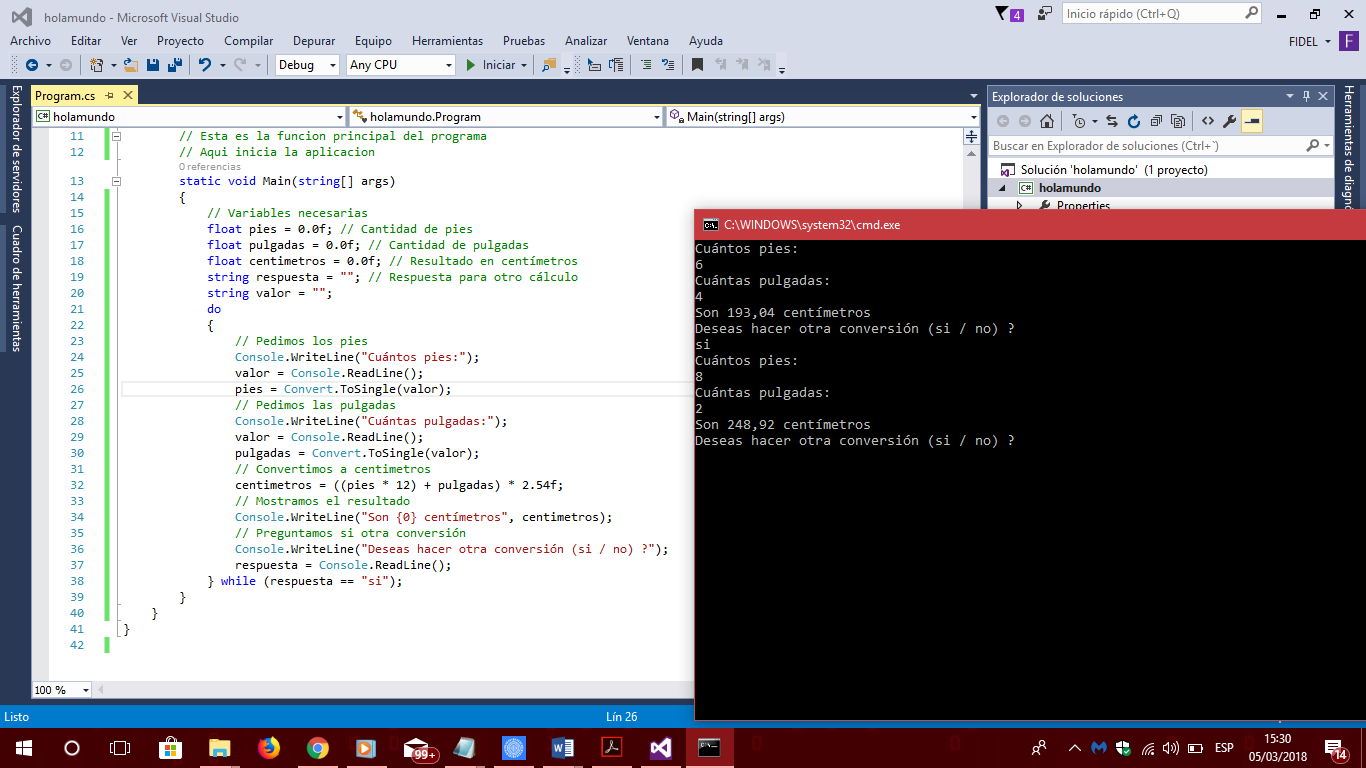
Permite que cierto código se repita mientras una condición se evalúe como verdadera. El valor de la evaluación dependerá del estatus del programa en un momento dado. El ciclo **do while** se codifica de la siguiente manera:

**do {**

**Código**

**}(condición);**

Crearemos el programa para la carpintería y usaremos el ciclo **do while** para que el programa se repita el número de veces necesarias, aun sin saber cuántas veces son.



**El ciclo while**

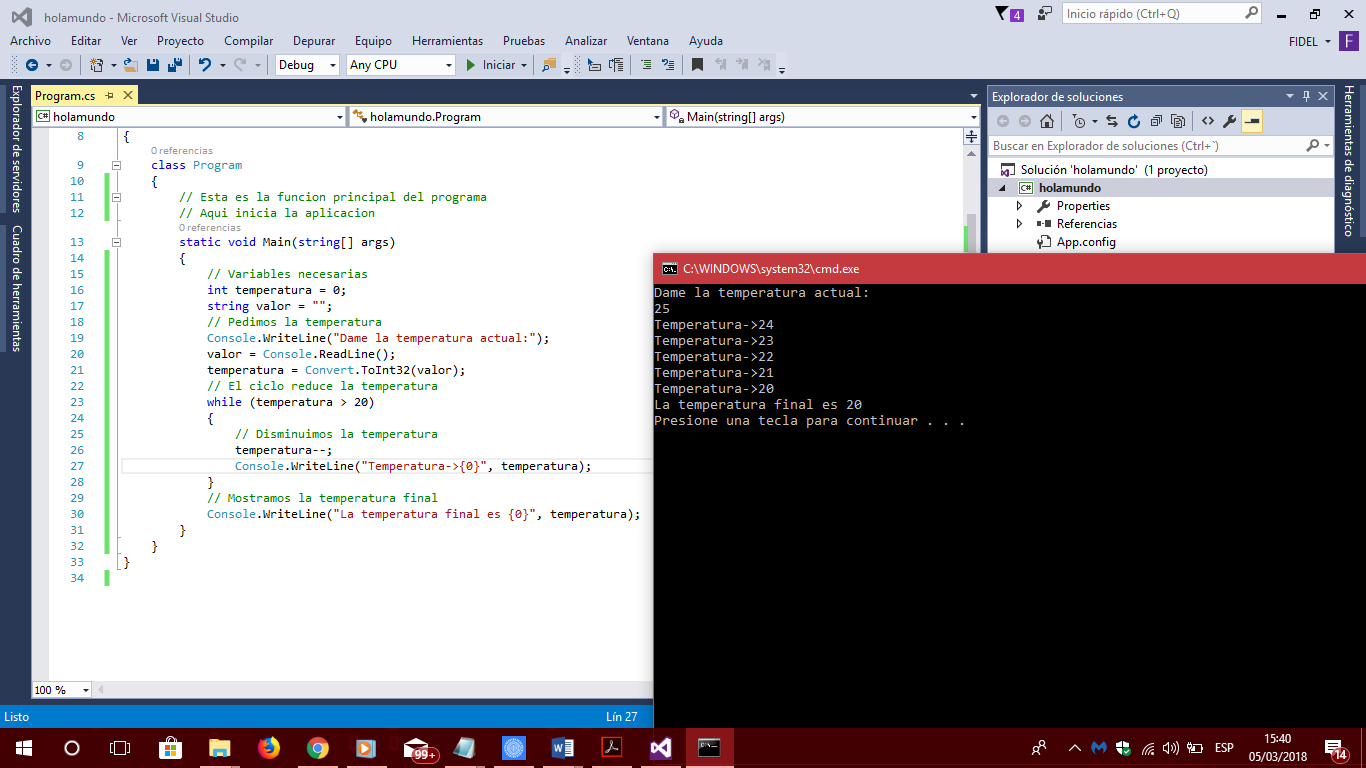
Este ciclo se conoce como **while** y en cierta forma se asemeja al anterior, pero tiene sus propias características que debemos conocer para utilizarlo correctamente.

El ciclo **while** también puede ser utilizado cuando tenemos algo que se debe repetir pero no conocemos el número de repeticiones previamente. La repetición del ciclo tiene que ver con el cumplimiento de una condición. A diferencia del ciclo **do while**, este ciclo puede no ejecutarse ni siquiera una vez. Su estructura es la siguiente:

**while(condicion) {**

**Código}**

En este ejemplo utilizaremos la característica del ciclo que puede repetirse o no repetirse ni siquiera una vez. Imaginemos que tenemos que hacer un programa de control para enfriar una caldera. La caldera debe ser enfriada a 20 grados centígrados.



**Arquitectura de diseño en 3 capas**

**La Interfaz o UI (User interface):**

Esta Capa es la encargada de interactuar con el usuario, es decir, son aquellas ventanas, mensajes, cuadros de diálogos o páginas web (en el caso del desarrollo web), que el usuario final utiliza para comunicarse con la aplicación, por medio de esta capa el usuario solicita que se ejecuten las tareas proporcionando parámetros de entrada y recibiendo datos como respuesta. Esta capa se comunica con la capa de Lógica de Negocio, enviando y solicitando información y con la capa de Entidades usando sus objetos para enviar y recibir esta información.

**La lógica de negocio o Business Logic:**

Se encarga de implementar, como su nombre lo dice, la lógica del negocio, es decir, todo lo que el Software debe de considerar antes de realizar una acción o el proceso que debe de seguir después de realizar una acción. Por ejemplo: Antes de solicitar a la capa de Datos la inserción de un grupo de registros en una tabla, valida que vayan todos los campos mandatorios dentro de esa solicitud si esta condición no se cumple entonces rechaza la inserción e informa del usuario del status de su solicitud; otro ejemplo podría ser, solicitar a la base de datos que valide la presencia de un registro antes de insertar el siguiente, validar los tipos de datos, etc. esos ejemplos por mencionar los más básicos y generales. Esta capa recibe de la Capa de Presentación las solicitudes, valida que las condiciones que establece el negocio se cumplan antes de realizar dicha acción o de hacer la respectiva solicitud a la Capa de Acceso a Datos.

**El acceso a Datos o Data Access:**

Esta capa es la encargada de la comunicación con la base de datos, en esta capa descansaran todas nuestras acciones CRUD (Create, Read, Update y Delete), será la única que “sabrá” que motor de base de datos se está utilizando pero le será completamente desconocido el “front”, es decir, jamás sabrá si nuestra aplicación es una aplicación web o desktop. Se encarga de recibir las peticiones de la Capa de Lógica de Negocio, ejecutar dichas acciones y devolver el resultado a la misma capa.

**Capa de Entidades o Entity Layer:**

Aunque aparentemente es una cuarta capa realmente no lo es, esta capa se encarga de contener todos aquellos objetos (clases) que representan al negocio, y esta es la única que puede ser instanciada en las 3 capas anteriores, es decir, solo ella puede tener comunicación con el resto, pero su función se limita a únicamente ser un puente de transporte de datos. Esta capa complementa a la Capa de Negocio

# Capítulo 3 Conclusiones

Ambos lenguajes son de nueva generación; ambos incluyen características avanzadas, como recolección de [basura](http://www.monografias.com/trabajos11/recibas/recibas.shtml), que facilitan las tareas de programación. Además, ambos son muy parecidos sintácticamente.

En estos lenguajes se realiza una compilación a un lenguaje intermedio: [Java](http://www.monografias.com/trabajos16/java/java.shtml) a bytecode, y C# a MSIL ([Microsoft](http://www.monografias.com/trabajos13/quienbill/quienbill.shtml) Intermediate Language). En ambos casos, ese lenguaje intermedio puede ser ejecutado en una máquina virtual apropiada.

Las similitudes de C# con Java son tan grandes que las diferencias se pueden detallar en un breve espacio de [tiempo](http://www.monografias.com/trabajos901/evolucion-historica-concepciones-tiempo/evolucion-historica-concepciones-tiempo.shtml).

A diferencia de Java, en C# incluso los tipos básicos (int y compañía) son objetos, lo que le hace un lenguaje aún más orientado a objetos. Además, permite la sobrecarga de operadores y tiene espacios de nombres, que terminan siendo muy parecidos a los paquetes en Java.

Se permiten las clases anidadas y existen los destructores, que son invocados de forma automática por el recolector de [basura](http://www.monografias.com/trabajos36/la-basura/la-basura.shtml). Las funciones puede recibir listas de parámetros de longitud no conocida y en C# existe el [concepto](http://www.monografias.com/trabajos10/teca/teca.shtml) de propiedades, concepto muy habitual en los objetos de GLib, y hay nuevas reglas de visibilidad "internal" y "protected internal" que hacen referencia a la visibilidad de [proyecto](http://www.monografias.com/trabajos12/pmbok/pmbok.shtml).

# Lista de referencias

La Biblia de C#

URL: https://www.educandose.com/wp-content/uploads/2017/08/la-biblia-de-c-sharp-gratis.pdf

C# Guía Total del Programador

Revista USERS