Nombre: Jorge Daniel Lopez lopez

Inicio de capacitación: 06/06/2022

Fin de capacitación:

Puesto: Programador Jr

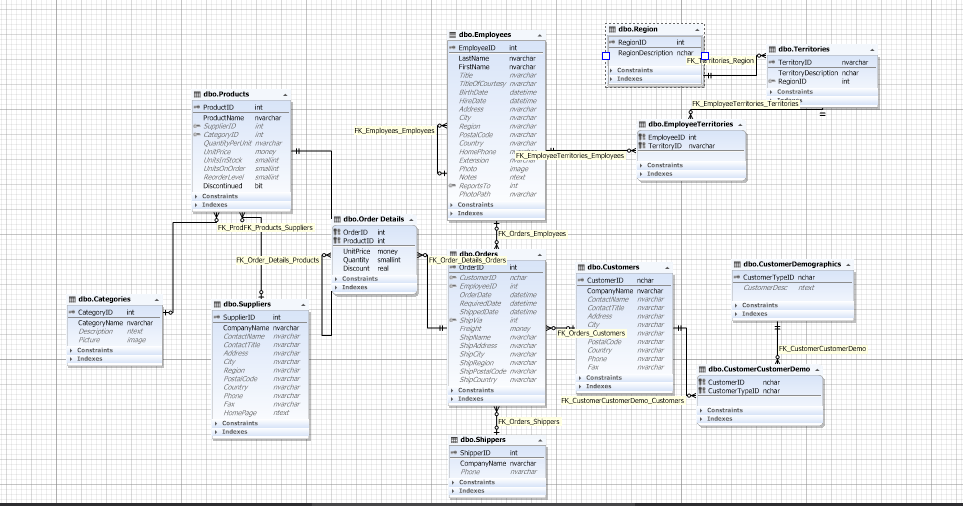
SQL

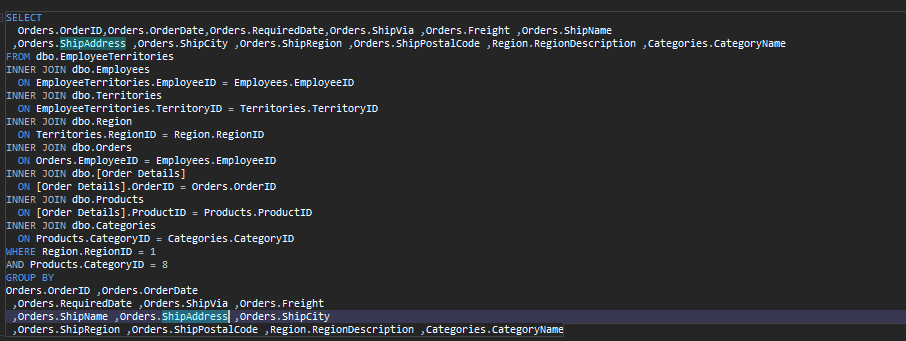
T-SQL

Transact-SQL es una extensión al SQL de Microsoft y Sybase. SQL Son las transacciones a la base de datos, para generar búsquedas especiales, alterar datos, modificarlos. Declarando sentencias específicas.

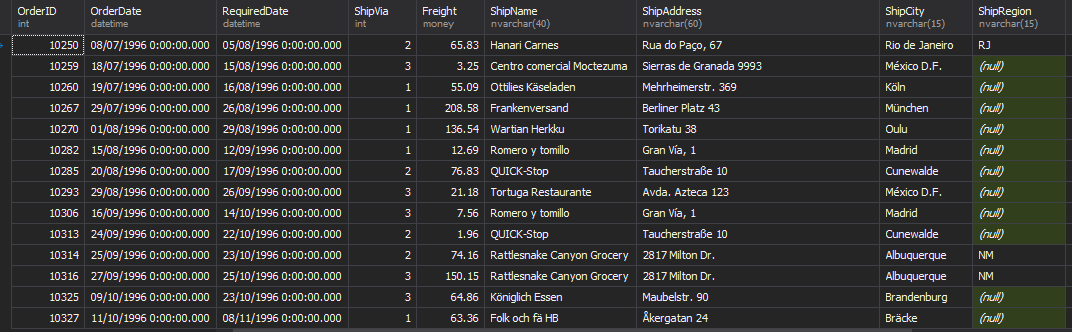
Para realizar algunos ejemplos, descargamos una BD, que es con la que estaremos trabajando, para demostrar como se usan los t-sql.

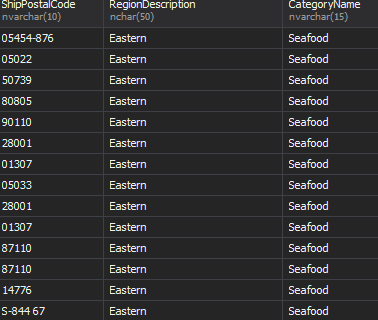
Agregamos nuestra base de datos, y nos quedará un diagrama que se muestra de la siguiente forma:

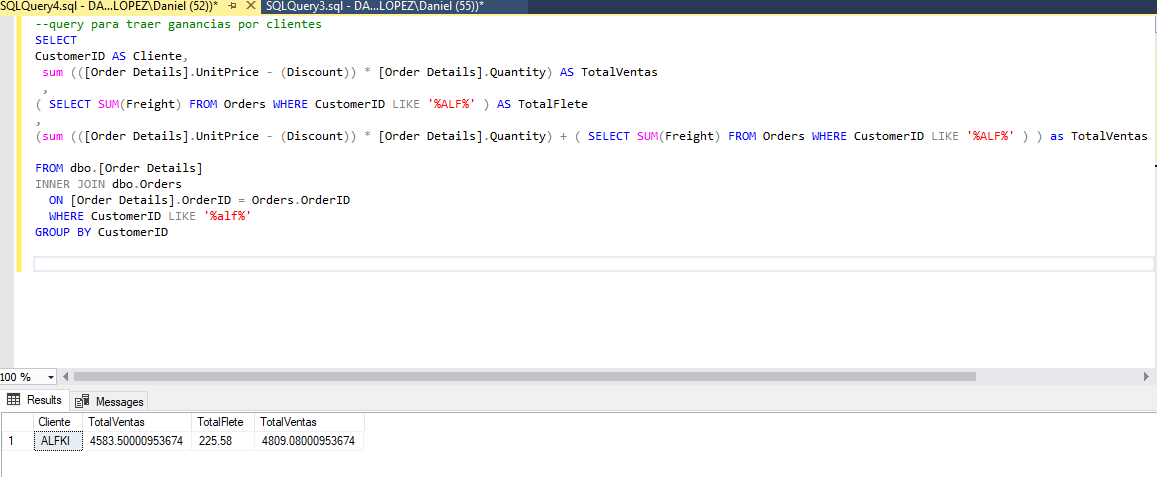


Para ejecutar una búsqueda de ordenes que hayan sido de empleados de Eastern y la categoría “Seafood” quedaría de la siguiente manera:

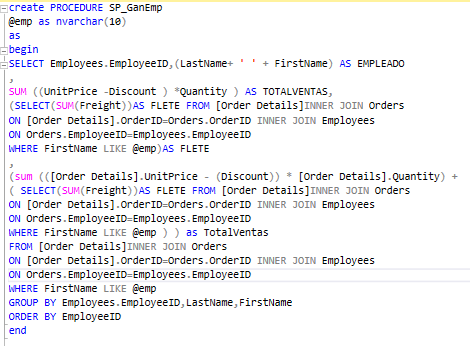
Lo cual nos mostrará un resultado de esta manera



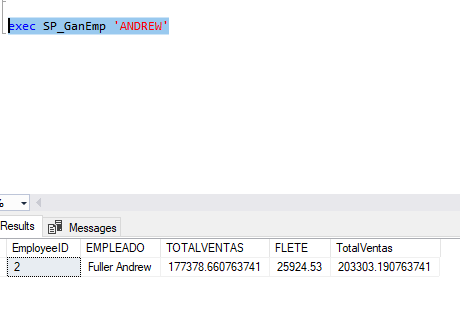


Para hacer una búsqueda especifica, tal como buscar ganancias por clientes, realizamos la siguiente consulta, la cual da el siguiente valor como resultado

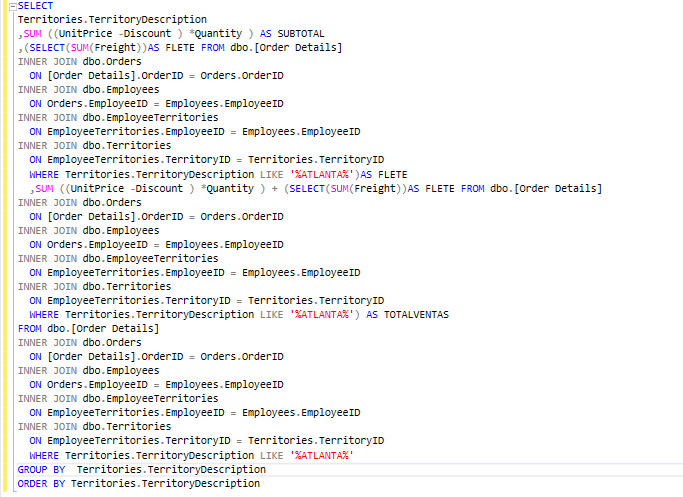
Y por empleados quedaría de la siguiente manera:



Creando un procedimiento almacenado que hará que haga la consulta con el siguiente código: esta vez buscamos con el empleado andew

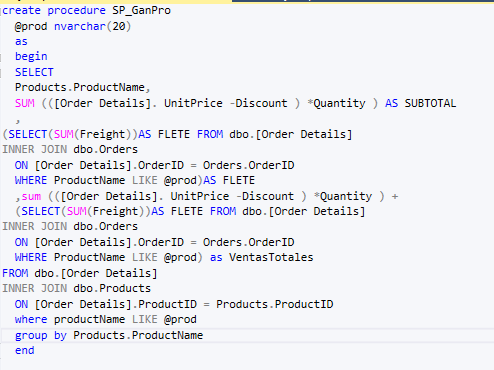


Por territorio y buscamos que nos dé una búsqueda relacionada como Atlanta

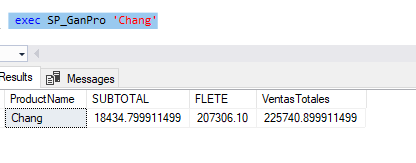
resultado



Por producto



El cual hacemos nuevamente un procedimiento almacenado para encontrar algo como deseemos, ejecutamos y nos dará el siguiente resultado:

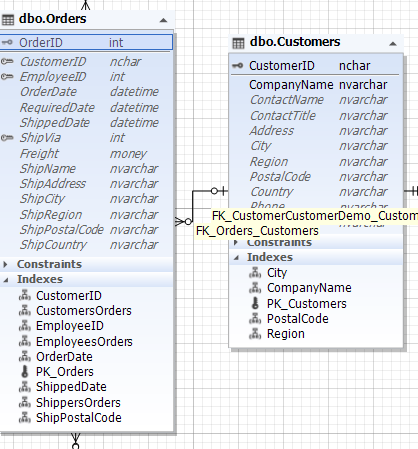


De esta manera podemos ejemplificar las diferentes formas de poder usar los t-sql

Indices sql

Se emplea un índice SQL para poder recuperar datos de una base de datos de una manera más rápida. El indexar una tabla o la vista es sin lugar a dudas, una de las mejores opciones de poder mejorar el rendimiento de las consultas y aplicaciones.

Un índice SQL es una tabla de búsqueda rápida para poder encontrar los registros que los usuarios necesitan buscar con mayor frecuencia. Ya que un índice es pequeño, rápido y optimizado para búsquedas rápidas. Además, que son muy útiles para conectar las tablas relacionales y la búsqueda de tablas grandes.

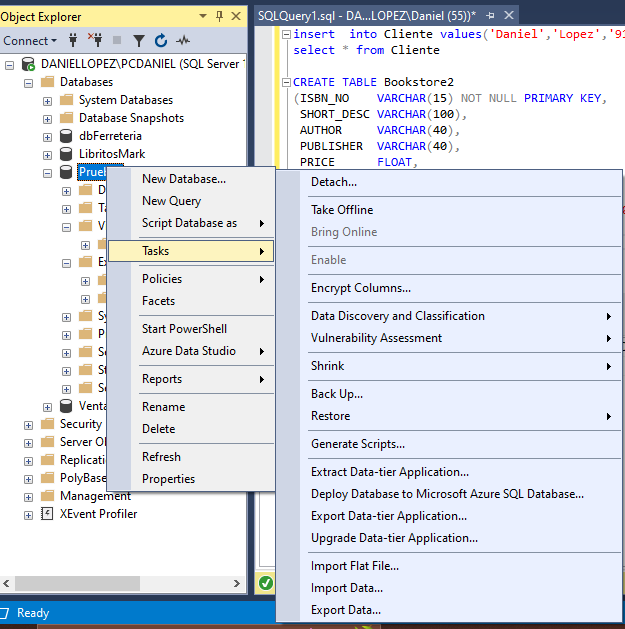
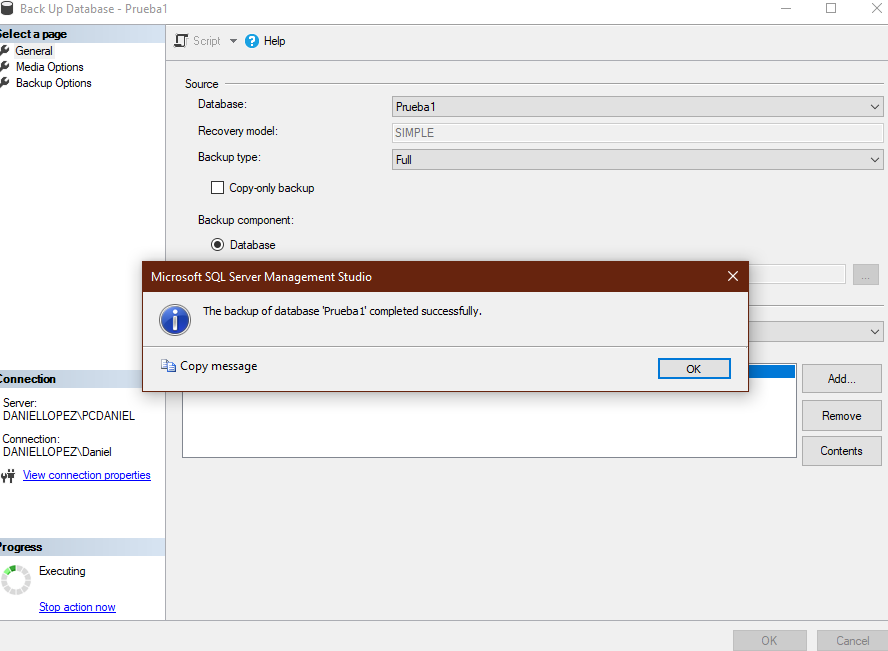
Dentro de los cuales existen las llaves PK, y FK, que son los conectores para enlazar nuestras tablas, según haya sido la relación del diagrama. Mostraré algunas

En este caso notamos que tipo de índices tiene cada tabla.

Respaldos SQL

Las copias de seguridad de SQL Server proveen una importante solución para proteger datos críticos que están almacenados en bases de datos SQL. Y para minimizar el riego de pérdida de datos, usted necesita asegurarse de que respalda sus bases de datos regularmente tomando en consideración los cambios aplicados a sus datos.

Antes de programar el trabajo de copias de seguridad, es necesario tener una estimación de cuánto espacio de disco será usado por la copia de seguridad completa de la base de datos. También, es necesario tener una estimación inicial del incremento en el tamaño de la base de datos, porque cuando este se incrementa las copias de seguridad completas requerirán más espacio de almacenamiento.



Es importante ver cuál de todas las BD que tenemos, es la que le haremos el bakcup.

ORM

ORM, por sus siglas al inglés: Object Relational Mappper, no es más que una pieza de software que nos permite interactuar con nuestra base de datos sin la necesidad de conocer SQL (El lenguaje de consultas). Todo esto utilizando el paradigma de programación orientada a objetos.

Existen algunos frameworks que ayudan al uso de ORM, tales como:

* Active Record
* Eloquent
* Peewee
* SQLAlchemy
* Entity Framework
* Hibernate

Los ORMs se encarga de traducir nuestra instrucción en el lenguaje de programación que estemos utilizando a una sentencia SQL que el gestor de base de datos pueda entender.

Utilizando una base de datos relacional, MySQL por ejemplo. Nuestra consulta quedaría de la siguiente manera:

SELECT id, username, email FROM id = 101:

Una sentencia bastante sencilla, Ahora, Utilizando el active record de ruby on rails, la consulta pudiera quedar de la siguiente manera.

User.where(id:101).select(:id, :username, :email)

Es de esta forma como nos podemos dar cuenta de la diferencia entre usar SQL, y un ORM, uno lleva su propia clase, su propio método y atributos

EntityFrameWork

Entity Framework (EF) es la solución open-source que ofrece Microsoft para un ORM en las aplicaciones .NET. Este framework permite a los desarrolladores convertir sus estructuras de datos en clases para poder trabajar con esa información usando objetos de .NET, reduciendo considerablemente la cantidad de código que sería necesario para realizar las tareas de acceso y gestión de datos en nuestra aplicación.

Lo cual, este mismo presenta algunas características que son importantes mencionar:

* **Cross-platform:** EF Core es un framework que funciona libremente en Windows, Linux y Mac.
* **Modelado:** EF genera un EDM (Entity Data Model o Modelo de Datos de Entidad) con entidades de distintos tipos de datos, los cuales podemos asignar o recuperar (get y set). Estos modelos son lo utilizados para realizar las consultas y persistencia en base de datos
* **Transacciones:** EF usa transacciones por defecto al realizar consultas a la base de datos. Adicionalmente permite modificar la gestión de transacciones en caso de necesitar algo más específico.
* **Cache:** EF incluye una cache que previene sobrecargas de la base de datos en caso de recibir una misma consulta numerosas veces.
* **Configuración:** EF permite ajustar los modelos generados añadiendo Data Annotations para sobrescribir el comportamiento por defecto.
* Migraciones: EF incluye comandos para generar y ejecutar migraciones, para modificar el esquema de la base de datos.
* Consultas: EF permite usar consultas LINQ como opción alternativa para consulta de datos. El gestor de la base de datos se encargará de traducir las consultas LINQ al lenguaje con que trabaje nuestra base de datos (SQL por ejemplo). Por último, EF también permite ejecutar consultas nativas o “raw” directamente.

SQL-LOG

La transacción de log, es una parte vital de una base de datos, ya que en ella se guarda cada una de las transacciones que se hacen a la base de datos.

 log de transacciones es el que asegura que una transacción tenga la propiedad ACID (**A**tomicity, **C**onsistency, **I**solation and **D**urability), debido a que cuando se modifica un dato en la base de datos, lo primero que se hace para asegurar que la transacción ha sido confirmada es escribir en el log de transacciones los cambios realizados por la transacción, una vez realizado esto recién se puede decir que la transacción esta completada, esta es la única manera de asegurar que una transacción sea durable en el tiempo.

El log de transacciones se maneja dependiendo del modelo de recuperación que tenga configurado la base de datos.

Hay que tener en claro que los registros dentro de log de transacciones manejan estados (activo / inactivo), de esta manera se identifica los registros que ya no son útiles y se pueden sobrescribir, o de los cuales se pueden reclamar espacio en disco. Si no hay espacio libre o registros inactivos para que el log de transacciones pueda escribir sobre ellos, entonces el este crecerá un determinado número de bytes, y como ya hemos explicado este crecimiento es costoso en términos de performance. De esto podemos derivar la conclusión de que el log de transacciones tiene un comportamiento cíclico, es decir que cuando llega al final de archivo buscara espacio libre al inicio para poder seguir escribiendo las transacciones de la base de datos.

Al momento de tener en cuenta el crecimiento del log, es bueno asignar un tamaño fijo, considerando que tanto puede crecer según sean las transacciones a la BD.

El método de las escrituras es de forma secuencial, es decir, va en un orden secuencial no es como un archivo de datos de una base de datos que escribe la data en forma aleatoria según la ubicación de las páginas de datos. Por eso mismo es más rápido en cuanto a escritura de disco, concierne porque la aguja del disco duro no debe moverse aleatoriamente, sino que esta solo debe moverse en una dirección hasta que se llegue al final del archivo y se necesite aplicar el comportamiento cíclico del archivo y se regrese al inicio del archivo para seguir ingresando transacciones.

DBForge

dbForge Studio es un entorno integrado para el desarrollo, gestión, administración, creación de informes y análisis de datos de SQL Server, MySQL y MariaDB, Oracle, PostgreSQL. SQL Manager permite crear, editar, copiar, adjuntar, separar, respaldar y restaurar bases de datos fácilmente de un servidor a otro.

Es una herramienta que, de mi punto de vista objetivo, es muy ligero para el uso, de fácil manejo, útil para realizar cualquier tipo de sucesos a la BD.

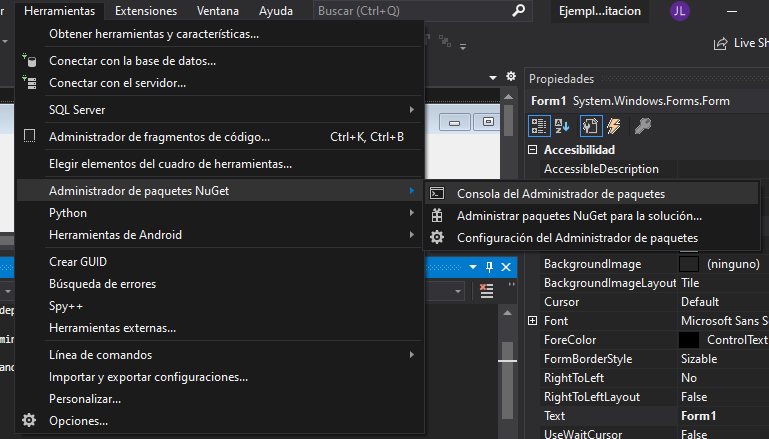
SDS

Es un IDE para hacer de forma más práctica el uso de nuestra base de datos, contiene diferentes tipos de funciones, visualizaciones que harán más fáciles los procedimientos que se vaya a realizar. Es un entorno totalmente amigable, intuitivo para poder realizar los trabajos correspondientes.

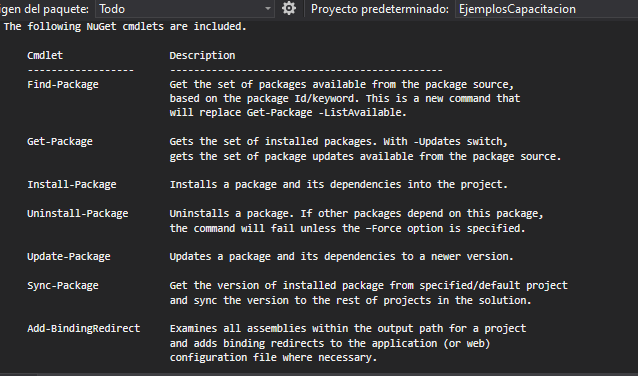
VISUAL ESTUDIO

Paquetes NUGET

Los paquetes de NuGet son unidades de código reutilizable que otros desarrolladores ponen a su disposición para que los use en sus proyectos. Los paquetes se instalan en un proyecto de Visual Studio mediante el Administrador de paquetes NuGet, la [Consola del Administrador de paquetes](https://github.com/NuGet/docs.microsoft.com-nuget.es-es/blob/live/docs/consume-packages/install-use-packages-powershell.md) o la [CLI de dotnet](https://github.com/NuGet/docs.microsoft.com-nuget.es-es/blob/live/docs/quickstart/install-and-use-a-package-using-the-dotnet-cli.md).



Para obtener algunos comandos del NuGet, es necesario escribir “get-help NuGet”, el cual nos dará una lista como la siguiente:



De acuerdo al paquete que vayamos a requerir será la instrucción que vamos a escribir.

Es una herramienta esencial para cualquier plataforma de desarrollo moderno, es un mecanismo a través del cual los desarrolladores pueden crear, compartir y consumir código útil. A menudo, este código se integra en "paquetes" que contienen código compilado (como archivos DLL) y otro contenido necesario en los proyectos que utilizan estos paquetes.

En .NET (incluido .NET Core), el mecanismo compatible con Microsoft para compartir código es NuGet, que define cómo se crean, hospedan y consumen paquetes en .NET, y ofrece las herramientas para cada uno de esos roles.

Desde un punto de vista sencillo, un paquete NuGet es un archivo ZIP con la extensión. nupkg que contiene código compilado (archivos DLL).

Los programadores con código para compartir crean paquetes y los publican en un host público o privado. Los consumidores de paquetes obtienen esos paquetes de los hosts adecuados, los agregan a sus proyectos y, después, llaman a la funcionalidad de un paquete en el código del proyecto. Después, el propio NuGet controla todos los detalles intermedios.

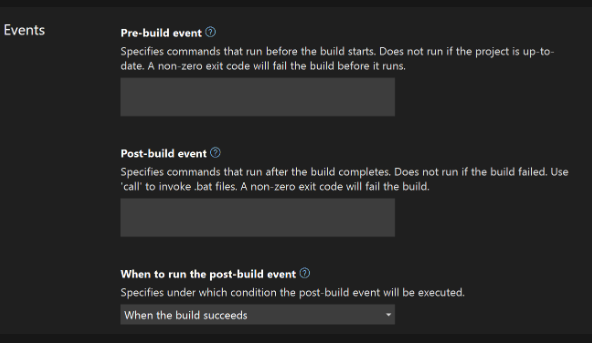
Evento PreBuild y PostBuild

Esto se refiere a que se puede ejecutar una sentencia antes de que este se compile, por ejemplo, puede ejecutar un archivo .bat antes de que comience una compilación o copiar nuevos archivos a una carpeta después de que se complete la compilación. Los eventos de compilación se ejecutan solo si la compilación alcanza con éxito esos puntos en el proceso de compilación.

Los eventos de compilación siguen la misma sintaxis que los comandos de DOS

Se tienen que usar eventos de compilación para especificar comandos que se ejecutan antes de que comience la compilación o después de que finalice.

Los eventos previos a la compilación no se ejecutan si el proyecto está actualizado y no se activa ninguna compilación.

En la sección Cuándo ejecutar el evento posterior a la compilación, especifique bajo qué condiciones ejecutar el evento posterior a la compilación.

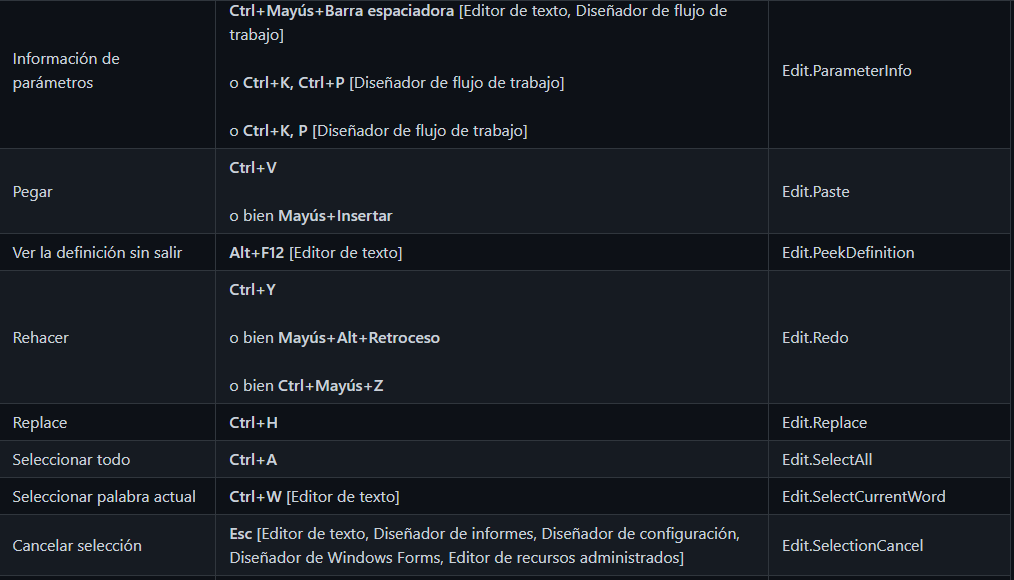
De esta manera podemos ver como se puede construir

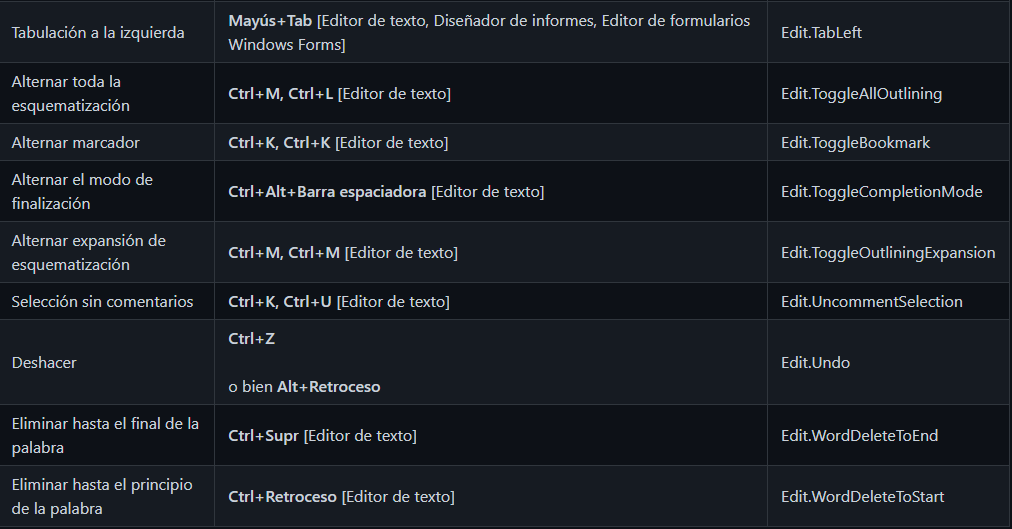
Shorcuts

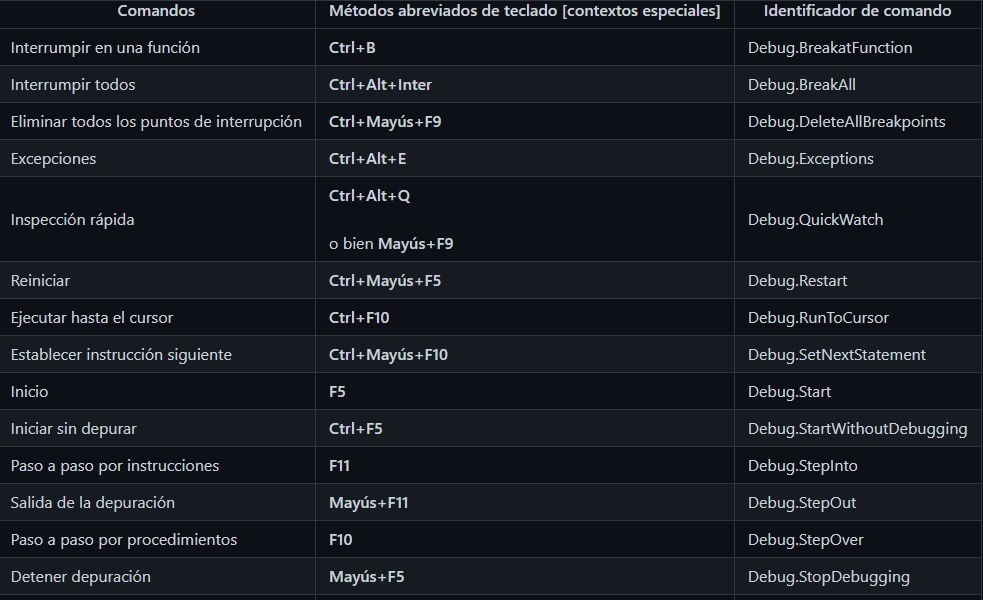
Es aquello que nos brindará los atajos para hacer algunos procesos en menos tiempo, lo cual dejaré una lista de lo que podemos usar:

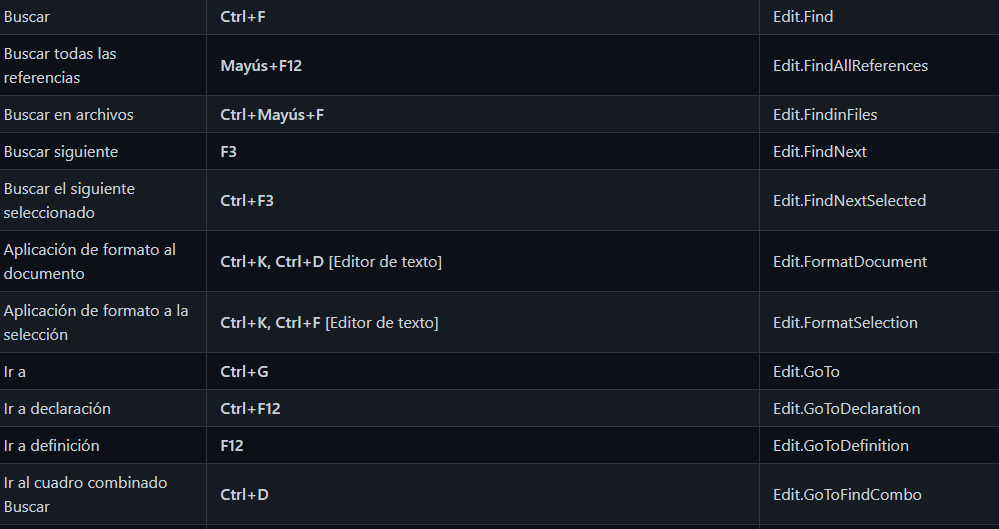










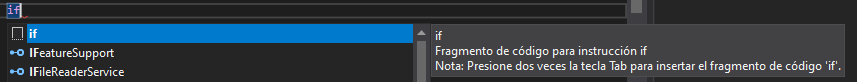




Fragmentos de código

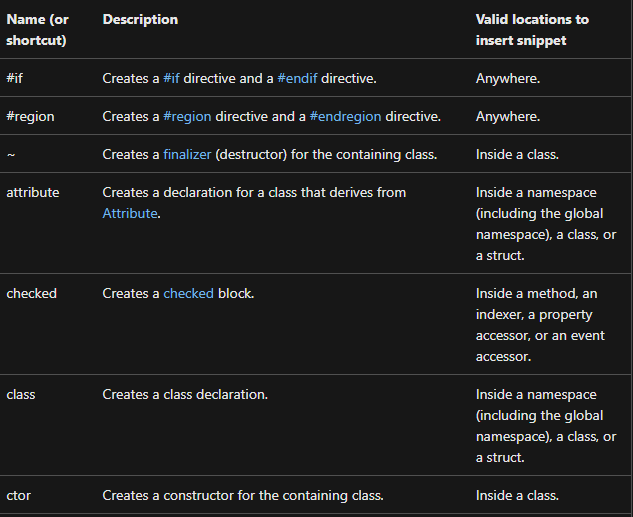
Estos fragmentos de código nos ayudarán a formar los códigos más rápido, solo basta dar dos veces “tab”, precedido de una parte de un código, y de forma automática nos dará lo que necesitamos, algunos ejemplos;

Al escribir la sentencia if

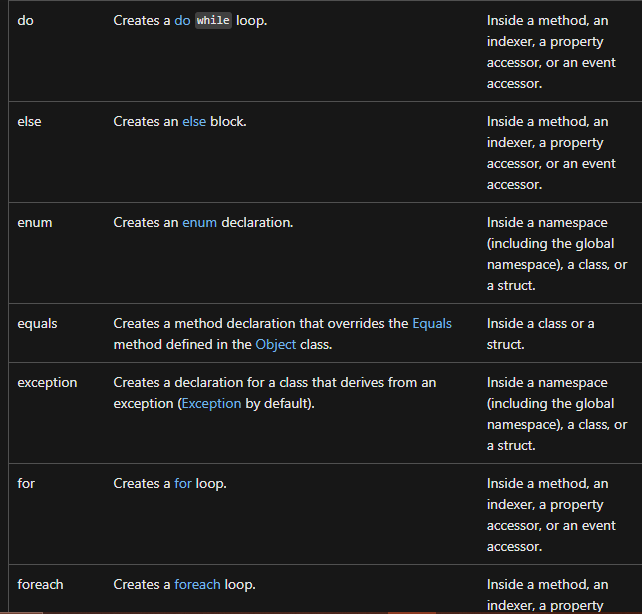


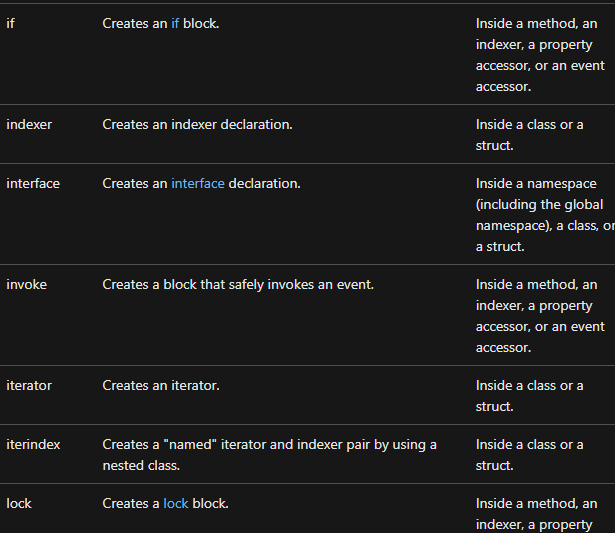
Y menciona que, si presionamos dos veces la tecla Tab, pondrá el fragmento completo

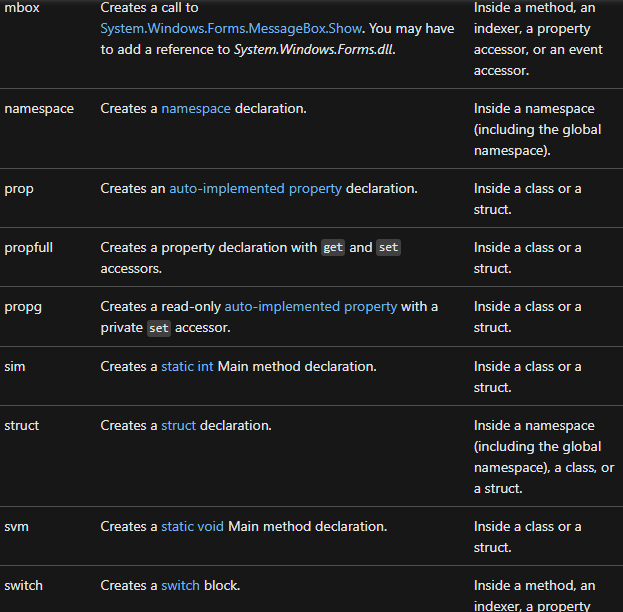
 y nos crea nuestra sentencia completa.

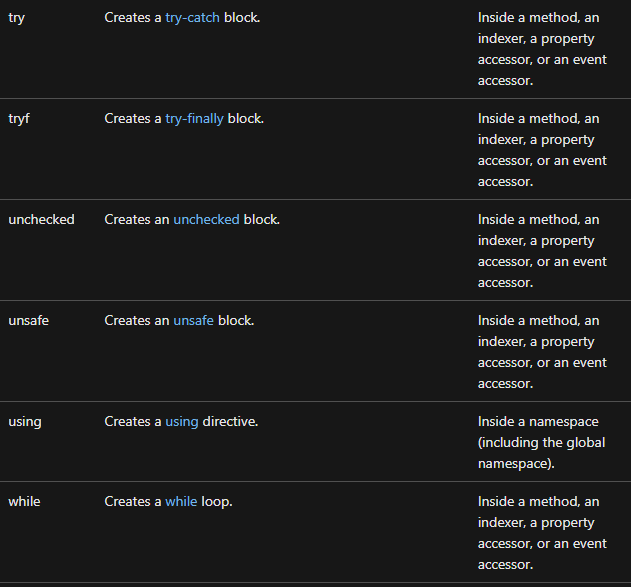


He acá una lista de como podremos usar estos fragmentos de código para agilizar nuestra programación, dejaré algunas más porque servirán en un futuro









Navegación en el código

Visual Studio ofrece numerosas formas de navegar por el código en el editor. En este tema se resumen las distintas maneras en las que se puede navegar por el código y se proporcionan vínculos a temas que incluyen más información

Puede usar los botones **Navegar hacia atrás** (**Ctrl**+**-**) y **Navegar hacia delante** (**Ctrl**+**Mayús**+**-**) de la barra de herramientas para mover el punto de inserción a ubicaciones anteriores o para volver a una ubicación más reciente desde una ubicación anterior. Estos botones retienen las últimas 20 ubicaciones del punto de inserción. Estos comandos también están disponibles en el menú **Vista**, en **Navegar hacia atrás** y **Navegar hacia delante**.

Usar barra de navegación

Puede usar la **barra de navegación** (los cuadros desplegables situados en la parte superior de la ventana de código) para navegar por el código de un código base. Puede elegir un tipo o un miembro para ir directamente a él. La barra de navegación aparece cuando se edita código en un código base de Visual Basic, C# o C++. En una clase parcial, los miembros definidos fuera del archivo de código actual pueden estar deshabilitados (se muestran en gris).



Comando para hacer navegación en el codigo

Ir a tiene los comandos siguientes, que están disponibles en el menú Editar en Ir a:

* Ir a la línea (Ctrl+G): moverse al número de línea especificado en el documento activo.
* Ir a todo (Ctrl+T o Ctrl+, ): Moverse a la línea, tipo, archivo, miembro o símbolo especificados.
* Ir al archivo (Ctrl+1, Ctrl+F): Moverse al archivo especificado en la solución.
* Ir al archivo reciente (Ctrl+1, Ctrl+R): Moverse al archivo especificado visitado recientemente en la solución.
* Ir al tipo (Ctrl+1, Ctrl+T): Moverse al tipo especificado en la solución.
* Ir al miembro (Ctrl+1, Ctrl+M): Moverse al miembro especificado en la solución.
* Ir al símbolo (Ctrl+1): Moverse al símbolo especificado en la solución.

Guía de productividad:

Es necesario dejar marcado como una de las páginas favoritas para utilizarlos en cualquier momento.

De los cuales deja en claro múltiples cosas para poder hacer un mejor uso de visual estudio, que atajos da, que formas de limpiar, como navegar fácilmente en el IDE, como trasladarse entre códigos. Si bien da varias listas de atajos para ser más productivos a la hora de estar programando, es demasiado útil cuando se tienen muchas líneas de código y múltiples formularios. Esto ayuda a que tengamos mejor movilidad y no cometamos muchos errores a la hora de escribir código.

Menciona también la forma de poder navegar en nuestro sistema, de cómo limpiar el código, qué significan los colores de subrayado, color rojo significa un error, verde es una advertencia, y los tres puntos en gris representa las sugerencias.

Nos da una serie de sugerencias de lo que podemos hacer, tales como depurar el código y que significa, que es procesar el código paso a paso, sirve para detectar de como está funcionando nuestro código, y como se comporta el mismo de acuerdo con las variables o de acuerdo a como lo estemos programando, y una lista de lo que nos ofrece es la siguiente:

* La capacidad de buscar una cadena dentro de las ventanas **Watch** (Inspeccionar), **Autos** (Autos) y **Locals** (Variables locales).
* *Ejecutar hasta hacer clic*, que permite mantener el puntero junto a una línea de código, hacer clic en el icono verde "reproducir" que aparece y ejecutar el programa hasta que llegue a esa línea.
* El **Asistente de excepciones**, que pone la información más importante en el nivel superior del cuadro de diálogo, por ejemplo, qué variable es null en NullReferenceException.
* La depuración [Retroceder](https://docs.microsoft.com/es-es/visualstudio/debugger/view-historical-application-state?view=vs-2022), que permite volver a puntos de interrupción anteriores y ver el estado de la aplicación tal y como estaba en un momento anterior.
* [Depuración de instantáneas](https://docs.microsoft.com/es-es/azure/application-insights/app-insights-snapshot-debugger), que permite investigar el estado de una aplicación web en directo en el momento en que se inició una excepción (debe estar en Azure).

Dejaré una tabla la cual muestra algunos accesos directos populares en visual estudio



Soluciones con múltiples proyectos

Proyecto

Debemos empezar a definir que serían los proyectos por que cuando cree una aplicación o un sitio web en Visual Studio, debe comenzar con un proyecto. En un sentido lógico, un proyecto contiene todos los archivos que se compilan en un archivo ejecutable, biblioteca o sitio web. Estos archivos pueden incluir código fuente, iconos, imágenes, archivos de datos, etc. Un proyecto también contiene la configuración del compilador y otros archivos de configuración que podrían ser necesarios en diversos servicios o componentes con los que el programa se comunica

Archivo de proyecto

Un archivo de proyecto Visual Studio usa MSBuild para compilar cada proyecto en una solución, y cada proyecto contiene un archivo de proyecto de MSBuild. La extensión de archivo refleja el tipo de proyecto, por ejemplo, un proyecto de C# (.csproj), un proyecto de Visual Basic (.vbproj) o un proyecto de base de datos (.dbproj). El archivo de proyecto es un documento XML que contiene toda la información y las instrucciones que MSBuild necesita para compilar el proyecto, incluidos el contenido, los requisitos de la plataforma, la información de versión, la configuración del servidor web o del servidor de bases de datos y las tareas que se deben llevar a cabo.

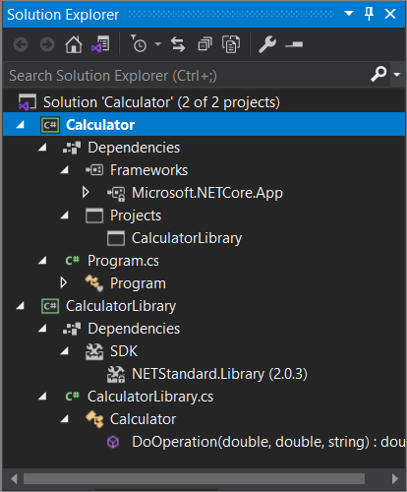
Soluciones

Un proyecto está incluido dentro de una solución. A pesar de su nombre, una solución no es una "respuesta", sino simplemente un contenedor con uno o más proyectos relacionados, junto con información de compilación, la configuración de ventanas de Visual Studio y archivos varios que no estén asociados a un proyecto determinado.

En Visual Studio se usan dos tipos de archivo ( *.sln* y *.suo*) para almacenar la configuración de las soluciones:



Carpeta de soluciones

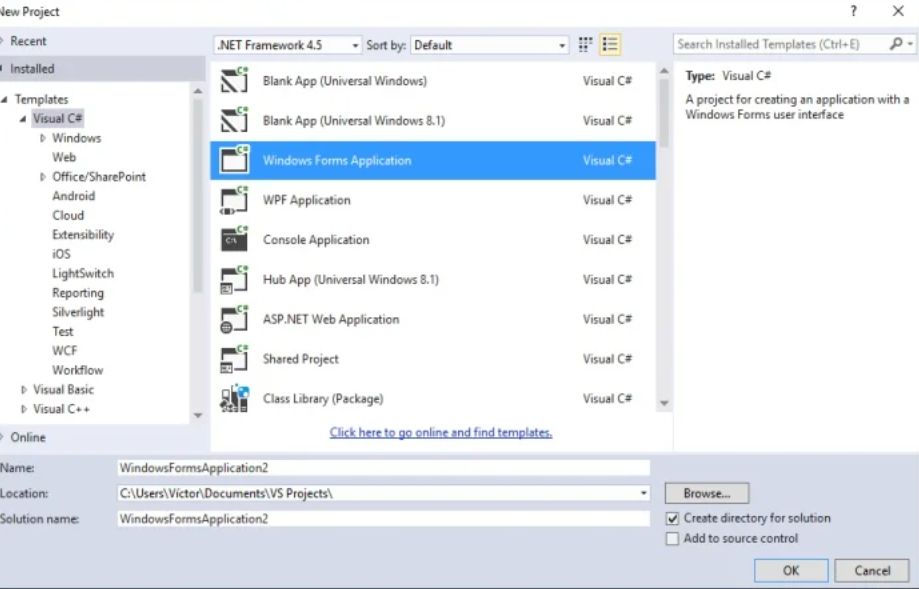
Una "carpeta de solución" es una carpeta virtual que solo se encuentra en el Explorador de soluciones, donde se puede usar para agrupar proyectos en una solución. Si desea buscar un archivo de una solución en un equipo, vaya a Herramientas>Opciones>Proyectos y soluciones>Ubicaciones. Para obtener más información, vea Cuadro de diálogo Opciones: Ubicaciones de proyectos y soluciones>.

Explorador de soluciones

Después de crear un proyecto nuevo, puede usar el Explorador de soluciones para ver y administrar el proyecto, la solución y sus elementos asociados. En la siguiente ilustración se muestra el Explorador de soluciones con una solución de C# que contiene dos proyectos:

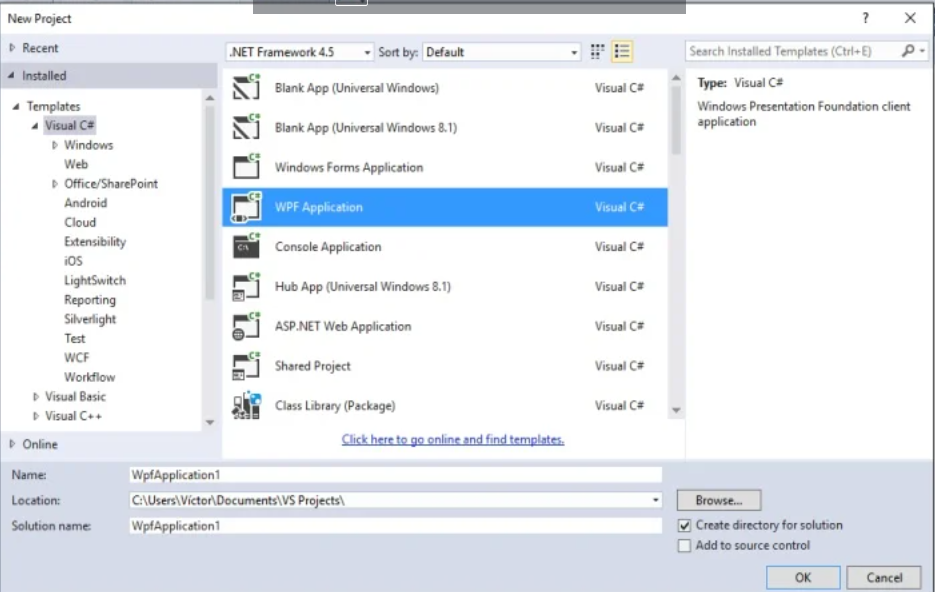
Elección de proyecto a ejecutar

Visual estudio tiene múltiples tipos de proyectos, de acuerdo a la necesidad que este presente, dejaré algunos de los proyectos más utilizados actualmente



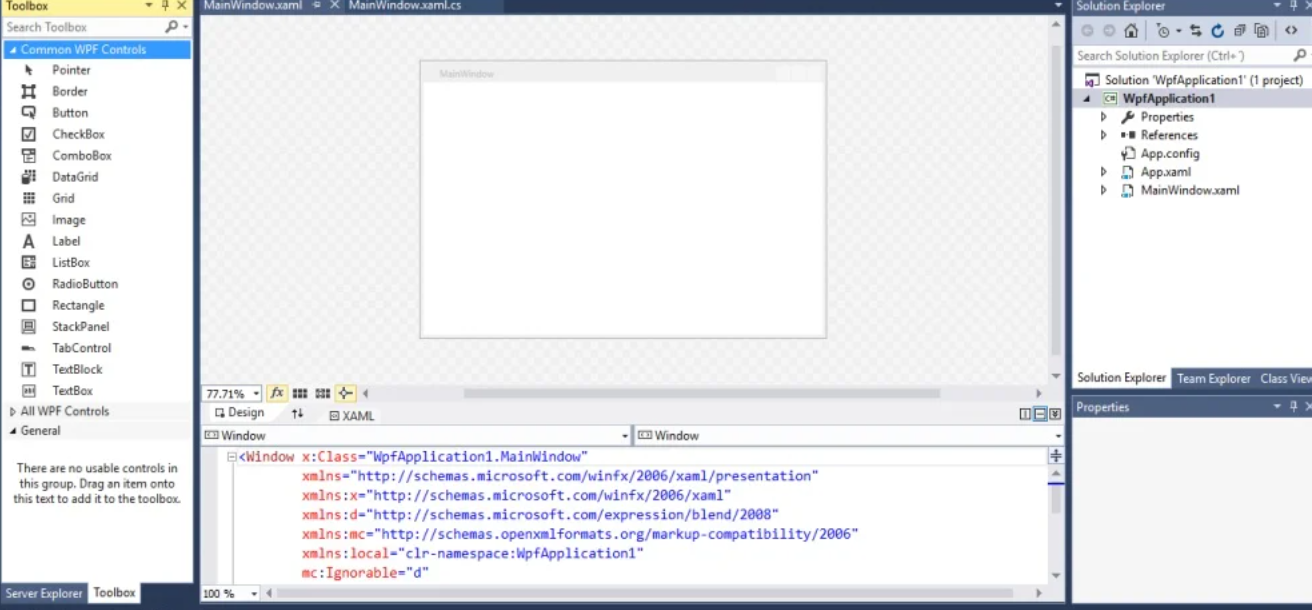
**Windows Forms**

Normalmente se escoge un proyecto de este tipo cuando se necesitan sistemas de uso diario en oficina, la utilización de teclado y mouse es indispensable. En una sola ventana (interfaz gráfica) pueden aparecer muchos controles de captura o de carga de información: TextBox, Labels, Grids, Menus, etc.

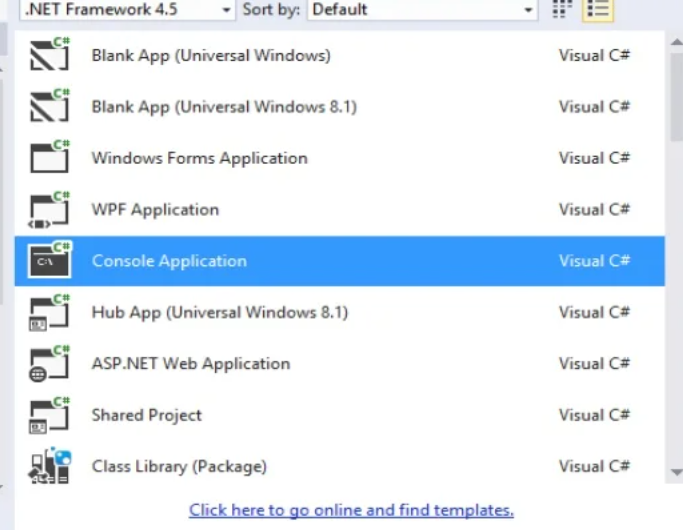


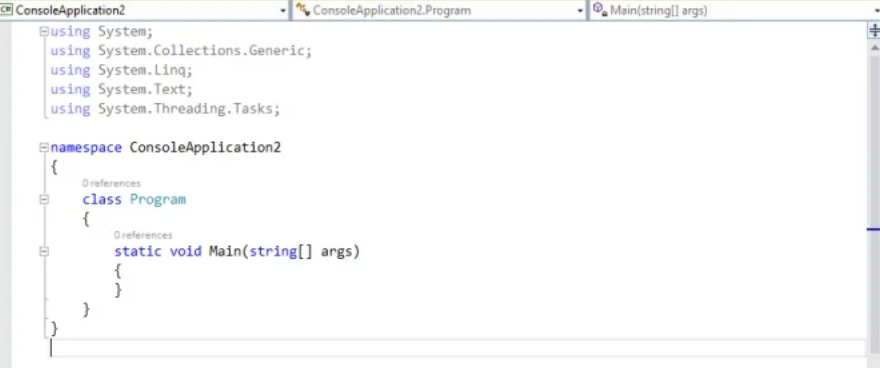
**Windows Presentation Foundation (WPF)**

Con el mismo objetivo que Windows Forms, la diferencia muy notable entre Windows Forms y WPF, es el uso de lenguaje XAML que separa el código de diseño del código funcional.

Windows Presentation Foundation (WPF) y XAML se combinan en un completo sistema de presentación para crear aplicaciones para el escritorio de Windows con experiencias de usuario visualmente impresionantes que incorporan interfaz de usuario, elementos multimedia y modelos empresariales complejos.

**Console Application**

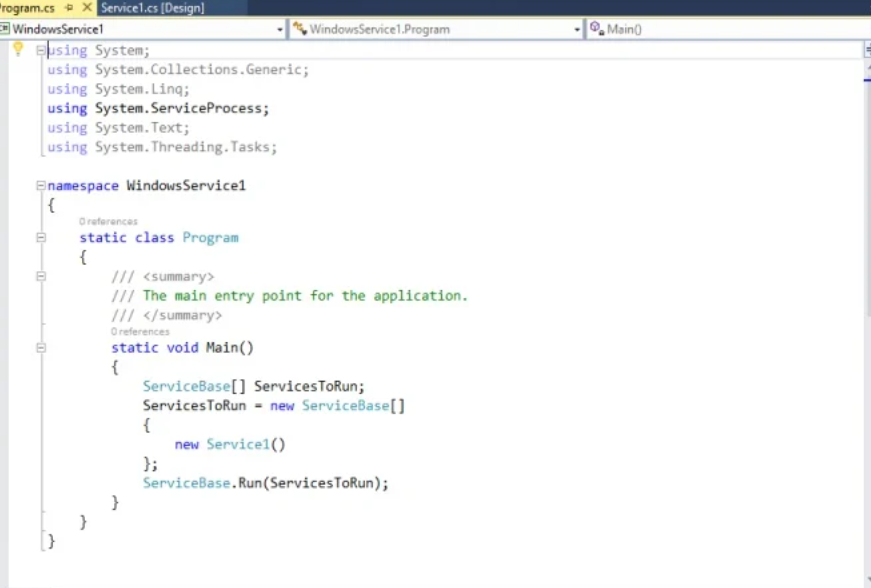
Este tipo de proyectos carecen de interfaz gráfica enriquecida, por ello son rápidas en su ejecución. Las aplicaciones de tipo consola se limitan a exponer un prompt como entrada de datos. Son recomendables cuando no se requiere demasiada entrada de información de los usuarios o los resultados de ejecución no necesitan un formato específico. Utilizan una interfaz como del tipo MSDOS



**Windows Services**

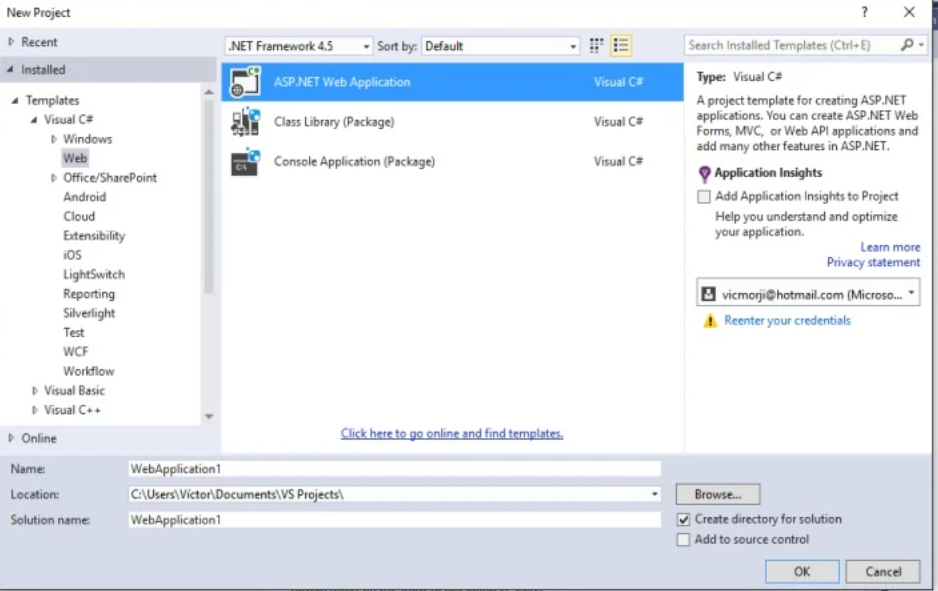
Un servicio de Windows es aquel que trabaja en paralelo, puede empezar a funcionar cuando arranca el sistema operativo, los servicios son aplicaciones en segundo plano que funcionan sin intervención del usuario.

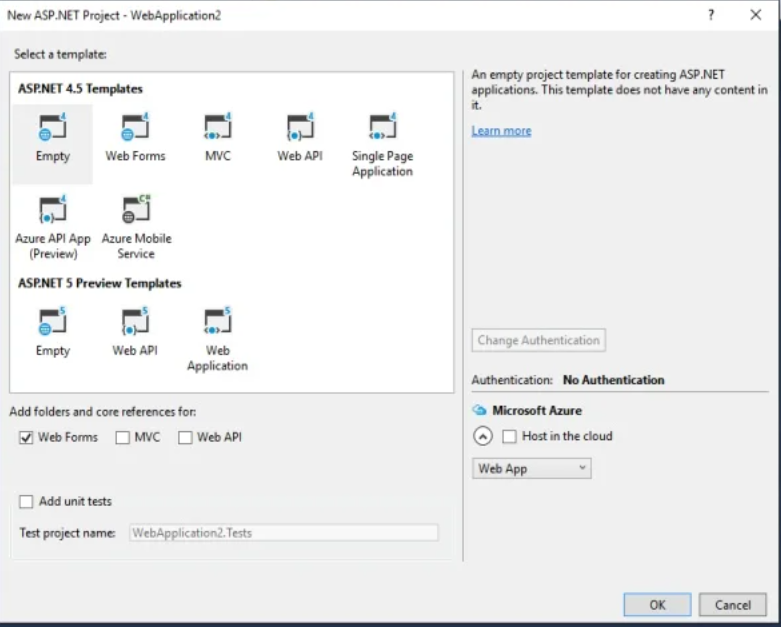




**ASP.Net**

Los proyectos ASP.Net trabajan con un comportamiento parecido a un proyecto tipo Windows Forms/WPF en el sentido de que existe la posibilidad de utilizar controles de usuario hacia nuestra página web (como si fuera una ventana de Windows). Aquí existen los llamados WebForms, que combinan HTML con lenguaje ASP.Net. Cada control que se coloca en nuestro WebForm tiene eventos programables y un montón de atributos a manipular, estos controles trabajan de lado del servidor, otorgando una respuesta por cada petición desde un browser.







**Silverlight**

Cuando las animaciones web estaban reinadas por Flash, llegó Microsoft Silverlight como otra opción. Los proyectos Silverlight permiten construir aplicaciones funcionales y enriquecidas al estilo de aplicaciones de escritorio WPF y XAML, la particularidad es que se utilizan desde el navegador web. De tal manera que, toda la experiencia de usuario y robustez de una aplicación de escritorio podía llevarse adentro de una página web. Al momento de escribir este artículo, Silverlight a perdido mucho terreno, puesto que los frameworks de JavaScript han evolucionado mucho y son preferibles para construir interfaces de usuario.

**Windows Phone**

Las verdaderas primeras aplicaciones móviles con tecnología Microsoft se llamaron Windows Phone (y digo esto porque antes existió un intento llamado Windows Mobile). Este tipo de proyectos utiliza XAML para el diseño la aplicación y C#/VB para programar la funcionalidad

**Xamarin**

Los proyectos de Xamarin permiten generar aplicaciones nativas para Windows, iOS y Android utilizando como lenguaje de programación C# y compartiendo el mismo código.

Archivos PDP

PDB es el acrónimo de Program Database. Es un formato propietario de Microsoft que almacena información de depuración de las aplicaciones ejecutables en Windows (.exe y .dll). Se crea a partir del código fuente para obtener una relación exhaustiva de los símbolos (variables, objetos, métodos...) del programa y su ubicación en el código y relacionarlos con el código compilado resultante. De esta manera, si necesitamos depurar la aplicación, las interfaces de depuración pueden "casar" las instrucciones de bajo nivel que se están ejecutando con las líneas de código concretas con las que se corresponden.

Con un PDB puedes depurar el código fuente a partir de volcados de memoria (y/o el ejecutable) con Visual Studio y herramientas como WinDBG

*Pero, si los archivos .pdb son para depuración, ¿por qué cuando genero una aplicación en modo release, para producción, también se me generan archivos .pdb? ¿Para qué me sirve eso?*

El motivo parece obvio, pero no lo es tanto. Y es que, si no tuviésemos este tipo de archivos, **no podríamos depurar la aplicación en producción**.

¿Cómo? ¿Depurar en producción? Parece un contrasentido. Al fin y al cabo, ya depuramos la aplicación mientras estamos desarrollándola ¿no?

En efecto. Durante las pruebas y el desarrollo ya depuramos la aplicación, por lo que la podemos desplegar a producción con ciertas garantías. Sin embargo, debemos tener en cuenta varias cosas:

* Hay muchas cosas que se nos van a escapar durante el desarrollo, que pueden producir errores una vez la aplicación está en funcionamiento.
* En ciertos tipos de aplicaciones, al estar en producción y tener muchos usuarios simultáneos, se dan situaciones muy difíciles de reproducir en desarrollo o pruebas, especialmente las que tienen que ver con código multisubproceso/asincronía.
* Si la aplicación se distribuye a muchos usuarios finales es muy probable que ciertos entornos personalizados (configuración, sistema operativo, nivel de parche, *drivers*...) den problemas que no hayamos podido tener en cuenta durante el desarrollo. En ese caso necesitaremos el .pdb para depurar el ejecutable concreto que presente los problemas.
* Cuando compilamos para *release* se le aplican multitud de optimizaciones al código final que no están al compilar en *debug* (no tanto por el *switch* /debug como el /optimize que se aplica en este caso). Generalmente esto no tiene impacto alguno sobre el funcionamiento de la aplicación más allá de mejorar su rendimiento, pero en ciertas ocasiones sí que pueden introducir cambios sutiles que afecten al resultado final e introduzcan nuevos bugs.
* Si hacemos *profiling* y optimización del ejecutable final, necesitaremos el .pdb para localizar las líneas que se pueden optimizar.

Diferencias entre Debug, Release, X86, X64

Un proyecto de Visual Studio tiene configuraciones independientes para las versiones de lanzamiento y depuración de un programa.Compile la versión de depuración (debug) para depurar y la versión de lanzamiento (release) para la versión de este tipo.

Se usan varias configuraciones para determinar si se compilará la versión de depuración o la comercial.Cuando se utiliza la configuración de compilación avanzada, el tipo de compilación viene determinado por los valores **Configuraciones de solución**, **Administrador de configuración** y **Propiedades del proyecto**, que incluyen la **Ruta de acceso de los resultados**.Cuando se utiliza la configuración de compilación simplificada, el tipo de compilación depende de si se hace clic en el comando **Iniciar depuración** o el comando **Compilar**.

Cuando compilas en debug, estás compilando para depurar, por lo cual la optimización es nula. Por el contrario, cuando compilas en Release, es para sacar una versión "comercial" del producto, por lo que se realizan muchas optimizaciones de código. Lo de x86 o x64 es dependiendo de si el producto se podrá ejecutar en pc de 32 o 64 bits

El "modo release" usualmente activa todas las optimizaciones importantes, para que el resultado sea veloz; mientras que el debug hace lo contrario. No es para que el resultado sea lento, sino para que al depurar el programa siga nuestras instrucciones

Bases de C#

Data table

Un objeto DataTable es prácticamente una tabla de datos, el cual puede estar relacionado con otro DataTable. Un ejemplo de como podemos usar un DataTable para hacer y crear tablas.

//Implementacion del datatable

DataTable tabla = new DataTable();

// Variables para las columnas y las filas

DataColumn column;

DataRow row;

// Se tiene que crear primero la columna asignandole Nombre y Tipo de datos

column = new DataColumn();

column.DataType = System.Type.GetType("System.Int32");

column.ColumnName = "Codigo";

tabla.Columns.Add(column);

// Se tienen que crear todas las columnas que queramos

column = new DataColumn();

column.DataType = Type.GetType("System.String");

column.ColumnName = "Descripcion";

tabla.Columns.Add(column);

column = new DataColumn();

column.DataType = Type.GetType("System.Decimal");

column.ColumnName = "ValorUnitario";

tabla.Columns.Add(column);

// Se crea una fila por cada registro que necesitemos agregar

for (int i = 1; i < 11; i++)

{

row = tabla.NewRow();

row["Codigo"] = i;

row["Descripcion"] = "Nombre del codigo " + i.ToString();

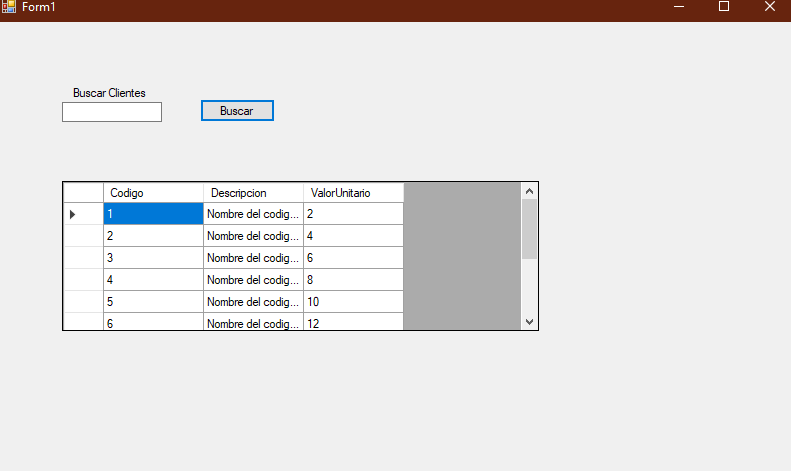
row["ValorUnitario"] = i \* 2;

tabla.Rows.Add(row);

}

dataGridView1.DataSource = tabla;

De lo cual hacemos un botón que ejecutará nuestro código, y veremos de la siguiente manera



DataSet

Un objeto DataSet es un contenedor, similar lógicamente a una base de datos. Es el que contendrá la totalidad de datos del conjunto sobre el que se desea trabajar. El tipo DataTable representa, como su propio nombre indica, una tabla.

Con una instancia de DataSet se puede usar cualquier número de objetos DataAdapter . Cada DataAdapter se puede usar para rellenar uno o varios objetos DataTable y resolver de nuevo las actualizaciones en el origen de datos correspondiente.

Para poder usar necesitamos usar una clase conexión que es la que hará contacto con nuestra base de datos la cual quedará de la siguiente manera:

public static class Conexion

{

public static int conf;

public static DataSet ds = new DataSet();

public static string s = "Server=DANIELLOPEZ\\PCDANIEL; DataBase= Northwind; Integrated Security=True";

public static SqlConnection conexion = new SqlConnection(s);

public static SqlCommand comando = new SqlCommand();

public static SqlDataAdapter sqlDA = new SqlDataAdapter();

static Conexion()

{

//CÓDIGO DEL CONSTRUCTOR

comando.Connection = conexion;

}

public static Tuple<int, DataSet> transacBD(string query, Boolean select)

{

DataSet dss = new DataSet();

if (select == true)

{

comando.CommandType = CommandType.Text;

comando.CommandText = query;

sqlDA.SelectCommand = comando;

conf = sqlDA.Fill(dss);

return Tuple.Create(conf, dss);

}

else

{

conexion.Open();

comando.CommandText = query;

conf = comando.ExecuteNonQuery();

conexion.Close();

return Tuple.Create(conf, dss);

}

}

}

Y vemos a continuación una variable de tipo dataSet, que es donde se están almacenando los datos que trae a partir de una base de datos sql. La cual retorna a la sentencia donde se esté llamando. Un ejemplo de como se usa la siguiente conexión sería de la siguiente manera:

//Ejemplo de como usar un data set

DataSet ejem = new DataSet();

var query = Conexion.transacBD("select CustomerID,CompanyName,ContactName,Address from customers where customerID like'%"+txtNombre.Text+"%' or 'CompanyName"+txtNombre.Text +"' ",true);

Conexion.conf = query.Item1;

Conexion.ds = query.Item2;

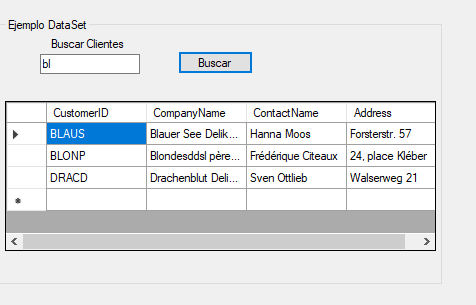
if (Conexion.ds.Tables[0].Rows.Count != 0)

{

dataGridView2.DataSource = Conexion.ds.Tables[0];

}

Lo cual nos hará una búsqueda de los clientes, según la base de datos que estuviéramos haciendo, la cual con un diseño ya establecido anteriormente quedará de la siguiente manera:



La búsqueda funciona con el CustomerID y CompanyName y se despliega en automático nuestra lista

Constructores

¿Cuál es la función de un constructor?

En **Programación** Orientada a Objetos (POO), un **constructor** es una subrutina cuya misión es inicializar un objeto de una clase. En el **constructor** se asignan los valores iniciales del nuevo objeto.

Cada vez que se crea una clase o estructura, se llama a su constructor. Una clase o estructura puede tener varios constructores que toman argumentos diferentes. Los constructores permiten al programador establecer valores predeterminados, limitar la creación de instancias y escribir código flexible y fácil de leer.

En C# podemos definir un método que se ejecute inicialmente y en forma automática. Este método se lo llama constructor.

El constructor tiene las siguientes características:

* Tiene el mismo nombre de la clase.
* Es el primer método que se ejecuta.
* Se ejecuta en forma automática.
* No puede retornar datos.
* Se ejecuta una única vez.
* Un constructor tiene por objetivo inicializar atributos.

Pondremos un ejemplo para aclarar lo siguiente:

public class Constructor1

{

int a;

int b;

//declaramos el constructor

public Constructor1(int x, int y)

{

a = x;

b = y;

}

public int Suma()

{

return a + b;

}

public int resta()

{

return a - b;

}

}

Este acaba de realizar una suma y una resta, nosotros podemos incluir lo que necesitemos. Se crea el constructor, declaramos el constructor y asignamos las variables, y para poderlos emplear debemos de hacer lo siguiente:

int n1,n2;

n1 = Convert.ToInt32( txtNum1.Text);

n2 = Convert.ToInt32(txtNum2.Text);

int rest, sum;

string cal;

Constructor1 ob= new Constructor1(n1,n2) ;

rest = ob.resta();

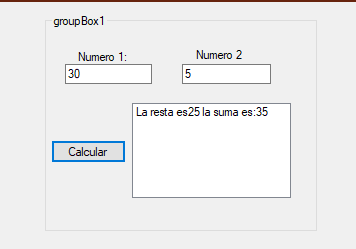
sum = ob.Suma();

cal=Convert.ToString("La resta es"+ rest + " "+ "la suma es:" + sum);

listBox1.Items.Add(cal);

declaramos las variables donde se asignaran los valores, y empezamos.

Tenemos que instanciar nuestro constructor a ob, y ya de ahí debemos retomar los datos para hacer suma o resta, la cual guardamos nuestro resultado a una variable, la cual luego asignamos un listbox que hará que se imprima lo que escribimos en los textbox, lo cual se muestra de la siguiente manera:



Anonymus type

Los tipos anónimos brindan una forma conveniente de encapsular un conjunto de propiedades de solo lectura en un solo objeto sin tener que definir explícitamente un tipo primero. El compilador genera el nombre del tipo y no está disponible en el nivel del código fuente. El compilador infiere el tipo de cada propiedad.

Los tipos anónimos se crean utilizando el newoperador junto con un inicializador de objeto. Para obtener más información acerca de los inicializadores de objetos, consulte Inicializadores de objetos y colecciones.

Un ejemplo práctico, junto con la demostración anterior del dataset, vemos como se usa una variable de tipo anónima, en este caso toma un valor que da soporte a un query, sin antes mencionar que es para llamar a una clase constructora que hace conexión a la base de datos.

var query = Conexion.transacBD("select CustomerID,CompanyName,ContactName,Address from customers where customerID like'%"+txtNombre.Text+ "%' or CompanyName like'%" + txtNombre.Text +"%' ",true);

Conexion.conf = query.Item1;

Conexion.ds = query.Item2;

if (Conexion.ds.Tables[0].Rows.Count != 0)

{

dataGridView2.DataSource = Conexion.ds.Tables[0];

}

Los tipos anónimos contienen una o más propiedades públicas de solo lectura. No son válidos otros tipos de miembros de clase, como métodos o eventos. La expresión que se utiliza para inicializar una propiedad no puede ser null una función anónima o un tipo de puntero.

Enumerable Type

En el lenguaje C#, enum(también denominado enumeración) es un tipo de valor definido por el usuario que se utiliza para representar una lista de constantes enteras con nombre. Se crea utilizando la enum palabra clave dentro de una clase, estructura o espacio de nombres. Mejora la legibilidad y el mantenimiento de un programa y reduce la complejidad.

La sintaxis básica para declarar en “enum” es:

public class ejemplenum

{

public enum meses

{

enero,

febrero,

abril,

junio,

agosto,

octubre,

noviembre,

diciembre

}

De esa manera indico que estoy asignando los valores al enum

}

// carga los datos en el load

private void EjemploConstructor\_Load(object sender, EventArgs e)

{

//Para unsar los enumerable tipe

string[] values = Enum.GetNames(typeof(ejemplenum.meses));

foreach (string s in values)

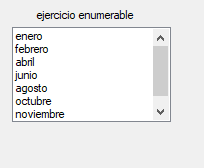
{

listBox2.Items.Add(s);

}

}

De esa manera puedo obtener los datos de lo que haya escrito, lo cual se muestra de la siguiente manera

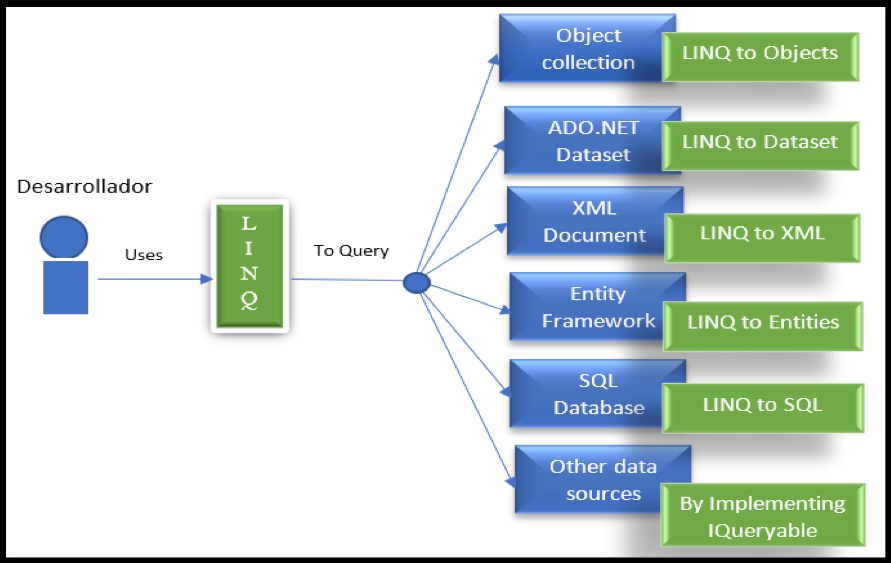


LINQ (lenguaje Integrated Query)

LINQ (Language Integrated Query) es una sintaxis de consulta uniforme en C # y VB.NET utilizada para guardar y recuperar datos de diferentes fuentes. Está integrado en C # o VB, lo que elimina la falta de correspondencia entre los lenguajes de programación y las bases de datos, y proporciona una interfaz de consulta única para diferentes tipos de fuentes de datos.

Por ejemplo, SQL es un lenguaje de consulta estructurado que se usa para guardar y recuperar datos de una base de datos. De la misma manera, LINQ es una sintaxis de consulta estructurada construida en C # y VB.NET utilizada para guardar y recuperar datos de diferentes tipos de fuentes de datos, como una colección de objetos, una base de datos SQL server, XML, MySQL o un servicio web entre otros.

LINQ siempre trabaja con objetos para que pueda usar los mismos patrones básicos de codificación para consultar y transformar los datos.



Realizamos un pequeño ejercicio para aplicar de lo que significa LINQ

class Carro

{

public string idCarro { get; set; }

public string Modelo { get; set; }

public string Year { get; set; }

public string Valor { get; set; }

public string idMarca { get; set; }

public string GetCArro()

{

string ds = ("idCarro: "+idCarro+"Modelo: "+ Modelo+"año :"+ Year+ "Valor: "+Valor+"La marca Corresponde a la empresa" +idMarca);

return ds;

}

}

class Marca

{

public int idMarca { get; set; }

public string nombre { get; set; }

}

Creamos nuestras clases, que será lo que estaremos ocupando, como si fueran tablas en bd

class controlCarro

{

/// <summary>

/// Constructor que ayudará a agregar datos a las tablas creadas anteriormente

/// </summary>

public controlCarro()

{

ListaCarro = new List<Carro>();

ListaMarca = new List<Marca>();

ListaMarca.Add(new Marca {idMarca=1, nombre= "Nissan" } );

ListaMarca.Add(new Marca { idMarca = 2, nombre = "Chevrolet" });

ListaMarca.Add(new Marca { idMarca = 3, nombre = "Ford" });

ListaCarro.Add(new Carro { idCarro = "1", Modelo="N25-Versa", Year = "2020",Valor = "2500000", idMarca="1" });

ListaCarro.Add(new Carro { idCarro = "1", Modelo = "C87-Camaro", Year = "2022", Valor = "12500000", idMarca = "2" });

ListaCarro.Add(new Carro { idCarro = "1", Modelo = "F69-Fiesta", Year = "2022", Valor = "250060", idMarca = "3" });

ListaCarro.Add(new Carro { idCarro = "1", Modelo = "N35-350z", Year = "2021", Valor = "980000", idMarca = "1" });

ListaCarro.Add(new Carro { idCarro = "1", Modelo = "C87-Aveo", Year = "2020", Valor = "250600", idMarca = "2" });

ListaCarro.Add(new Carro { idCarro = "1", Modelo = "F52-Mustang", Year = "2021", Valor = "293000", idMarca = "3" });

}

public List<Carro> ListaCarro;

public List<Marca> ListaMarca;

En este fragmento de código obtenemos los datos que estamos buscando y usamos un foreach para recorrer todos los datos que se encontraron.

public void getCarro( string año )

{

IEnumerable<Carro> Obmarca = from Carro in ListaCarro where Carro.Year == año select Carro;

foreach (Carro marca1 in Obmarca)

{

marca1.GetCArro();

}

}

}

Y al final asignamos una clase para llamar a ejecutar todo este código

private void btnMuestra\_Click(object sender, EventArgs e)

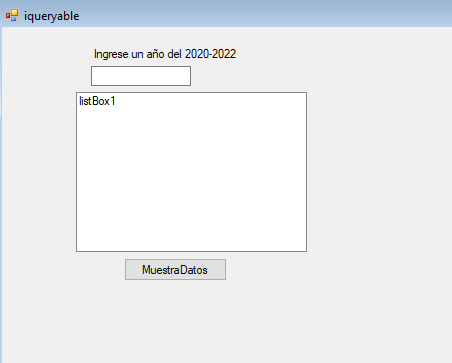
{

controlCarro crC = new controlCarro();

crC.getCarro(txtYear.Text);

}

Asigné dentro de un botón, que se ve de esta forma

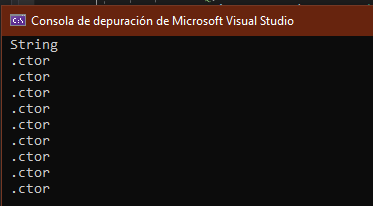
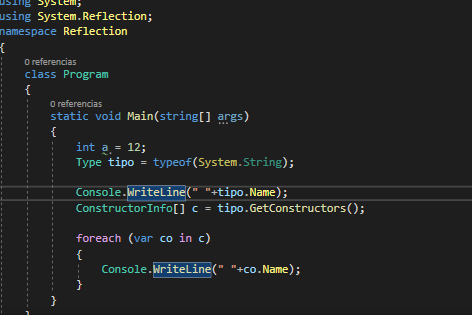


Lo que hace es hacer una búsqueda de los carros cuyo año nosotros pongamos

Reflexion

Reflection proporciona objetos (de tipo [Type](https://docs.microsoft.com/en-us/dotnet/api/system.type) ) que describen ensamblajes, módulos y tipos. Puede usar la reflexión para crear dinámicamente una instancia de un tipo, vincular el tipo a un objeto existente u obtener el tipo de un objeto existente e invocar sus métodos o acceder a sus campos y propiedades. Si está utilizando atributos en su código, la reflexión le permite acceder a ellos. Para obtener más información, consulte [Atributos](https://docs.microsoft.com/en-us/dotnet/standard/attributes/) .

Aquí hay un ejemplo simple de reflexión utilizando el método [GetType()](https://docs.microsoft.com/en-us/dotnet/api/system.object.gettype#system-object-gettype) , heredado por todos los tipos de la Objectclase base, para obtener el tipo de una variable:



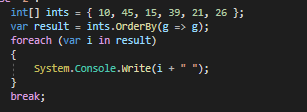
Por lo tanto esto nos muestra el contenido de cada variable, mostramos que información tiene el constructor, y por eso se ve de esa manera.

Métodos de Extensión

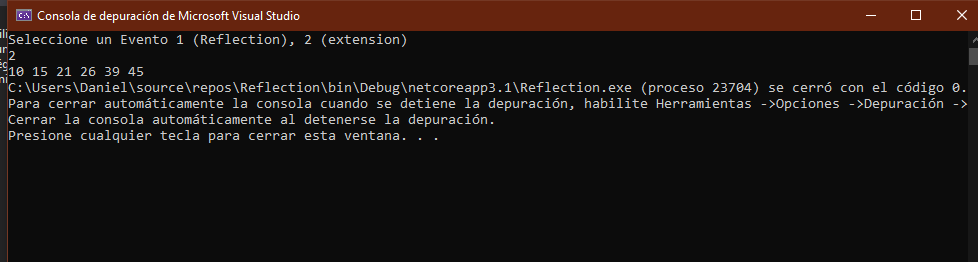
Los métodos de extensión permiten "agregar" métodos a los tipos existentes sin crear un nuevo tipo derivado, recompilar o modificar de otra manera el tipo original. Los métodos de extensión son métodos estáticos, pero se les llama como si fueran métodos de instancia en el tipo extendido. En el caso del código de cliente escrito en C#, F# y Visual Basic, no existe ninguna diferencia aparente entre llamar a un método de extensión y llamar a los métodos definidos en un tipo.

Los métodos de extensión más comunes son los operadores de consulta LINQ estándar, que agregan funciones de consulta a los tipos [System.Collections.IEnumerable](https://docs.microsoft.com/es-es/dotnet/api/system.collections.ienumerable) y [System.Collections.Generic.IEnumerable<T>](https://docs.microsoft.com/es-es/dotnet/api/system.collections.generic.ienumerable-1) existentes. Para usar los operadores de consulta estándar, inclúyalos primero en el ámbito con una directiva using System.Linq. A partir de ese momento, cualquier tipo que implemente [IEnumerable<T>](https://docs.microsoft.com/es-es/dotnet/api/system.collections.generic.ienumerable-1) parecerá tener métodos de instancia como [GroupBy](https://docs.microsoft.com/es-es/dotnet/api/system.linq.enumerable.groupby), [OrderBy](https://docs.microsoft.com/es-es/dotnet/api/system.linq.enumerable.orderby), [Average](https://docs.microsoft.com/es-es/dotnet/api/system.linq.enumerable.average), etc. Puede ver estos métodos adicionales en la finalización de instrucciones de IntelliSense al escribir "punto" después de una instancia de un tipo [IEnumerable<T>](https://docs.microsoft.com/es-es/dotnet/api/system.collections.generic.ienumerable-1), como [List<T>](https://docs.microsoft.com/es-es/dotnet/api/system.collections.generic.list-1) o [Array](https://docs.microsoft.com/es-es/dotnet/api/system.array).

Ejemplo de Order by:



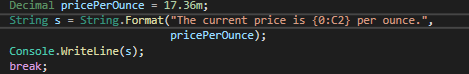
Que se vería de la siguiente manera: se ve claramente como se ordena los números internos

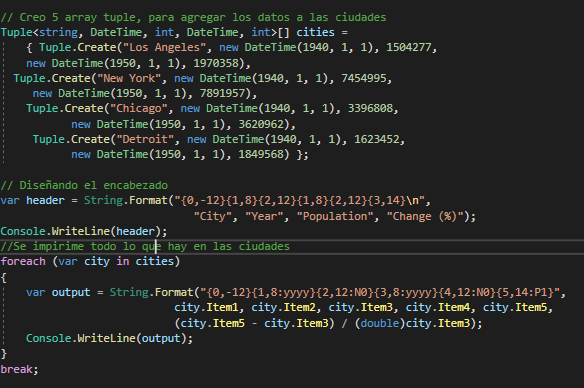


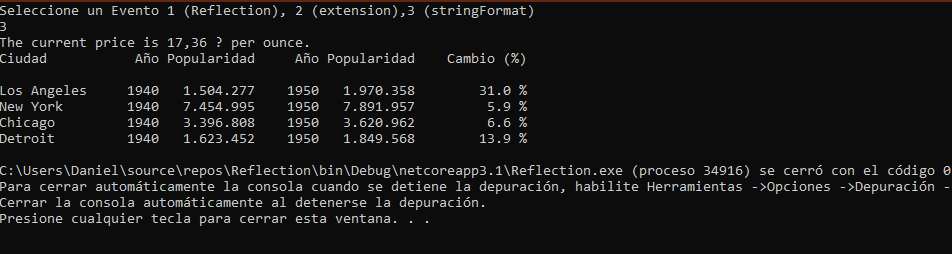
String Metodo

Convierte el valor de los objetos en cadenas según los formatos especificados y los inserta en otra cadena.

Utilice String.Format si necesita insertar el valor de un objeto, variable o expresión en otra cadena. Por ejemplo, puede insertar el valor de un valor decimal en una cadena para mostrárselo al usuario como una sola cadena:







Hay una diversidad de formas que podemos emplear el metodostring, todo sea de acuerdo a lo que se esté necesitando.

String Interpolation

La función de interpolación de cadenas se basa en la función de formato compuesto y proporciona una sintaxis más legible y conveniente para incluir resultados de expresiones formateadas en una cadena de resultados.

Para identificar un literal de cadena como una cadena interpolada, antepóngalo con el $símbolo. Puede incrustar cualquier expresión de C# válida que devuelva un valor en una cadena interpolada. En el siguiente ejemplo, tan pronto como se evalúa una expresión, su resultado se convierte en una cadena y se incluye en una cadena de resultado:

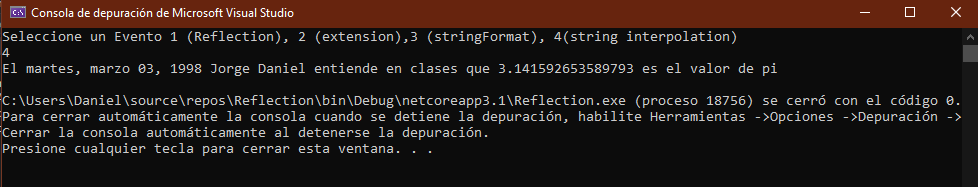
Especifique una cadena de formato compatible con el tipo de resultado de la expresión siguiendo la expresión de interpolación con dos puntos (":") y la cadena de formato:

{<interpolationExpression>:<formatString>}

El siguiente ejemplo muestra cómo especificar cadenas de formato estándar y personalizadas para expresiones que producen resultados numéricos o de fecha y hora:

var date = new DateTime(1998, 3, 3);

Console.WriteLine($"El {date:dddd, MMMM dd, yyyy} Jorge Daniel entiende en clases que {Math.PI} es el valor de pi");

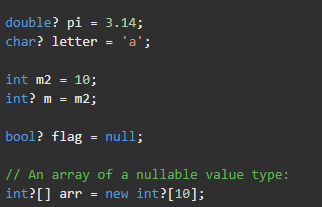


NUllable Types

Un *tipo de valor que acepta valores NULL representa todos los valores de su*[tipo de valor](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/value-types)T? subyacente y un valor [NULL](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/null) adicional . Por ejemplo, puede asignar cualquiera de los tres valores siguientes a una variable: , o . Un tipo de valor subyacente no puede ser un tipo de valor anulable en sí mismo. Tbool?truefalsenullT

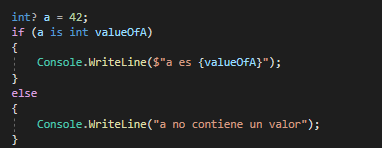
Cualquier tipo de valor que acepta valores NULL es una instancia de la estructura genérica System.Nullable<T> . Puede hacer referencia a un tipo de valor que acepta valores NULL con un tipo subyacente Ten cualquiera de las siguientes formas intercambiables: Nullable<T>o T?.

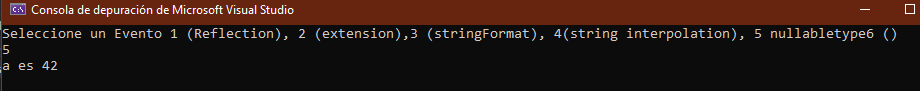
Dado que un tipo de valor se puede convertir implícitamente en el tipo de valor que acepta valores NULL correspondiente, puede asignar un valor a una variable de un tipo de valor que acepta valores NULL como lo haría con su tipo de valor subyacente. También puede asignar el nullvalor. Por ejemplo:



El valor predeterminado de un tipo de valor que admite valores NULL representa null, es decir, es una instancia cuya propiedad Nullable<T>.HasValue devuelve false.

Siempre puede usar las siguientes propiedades de solo lectura para examinar y obtener un valor de una variable de tipo de valor anulable:





Nullable<T>.HasValue indica si una instancia de un tipo de valor que acepta valores NULL tiene un valor de su tipo subyacente.

Nullable<T>.Value obtiene el valor de un tipo subyacente si HasValue es true. Si HasValue es false, la propiedad Value genera una InvalidOperationException .

NewtonJson

En la actualidad los datos JSON se utilizan sobre todo al recibir datos en formato JSON de un servicio web con AJAX y obtener datos de él. Para poder obtener datos en formato JSON o crear texto JSON a partir de un objeto personalizado, usaremos la serialización y la deserialización JSON en C#.

**¿Qué es JSON?**

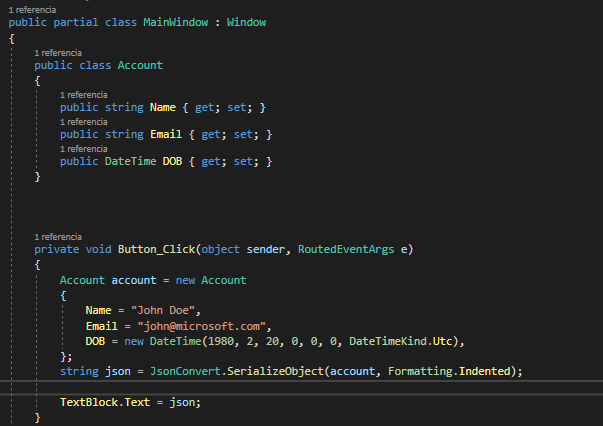
JSON (JavaScript Object Notation) es un formato liviano para el intercambio de datos. Es fácil para los humanos leer, escribir, analizar y generar un JSON y es un formato de texto que es completamente independiente del lenguaje.

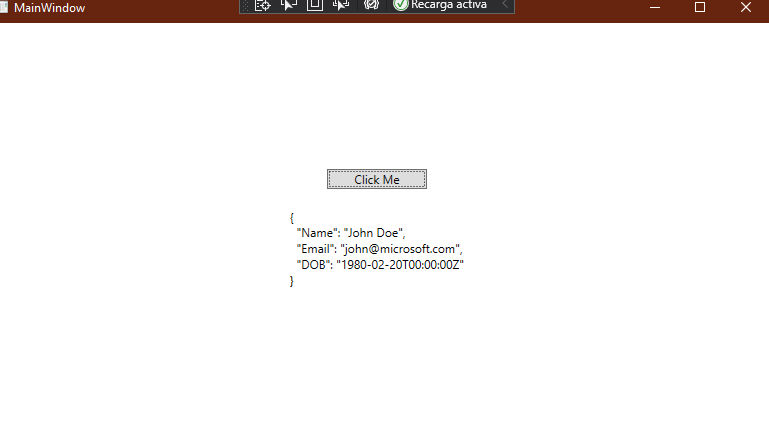
El formato JSON permite las estructuras de datos siguientes:

* **Colección de pares: nombre/valor, esta estructura de datos es compatible con diferentes lenguajes de programación.**
* **Lista ordenada de valores: puede ser una matriz, lista, vector o secuencia, etc.**

**¿Qué es Serialización?**

En ciencias de la computación, la serialización (o marshalling en inglés) consiste en un proceso de codificación de un objeto en un medio de almacenamiento (como puede ser un archivo, o un buffer de memoria) con el fin de transmitirlo a través de una conexión en red como una serie de bytes o en un formato humanamente más legible como XML o JSON, entre otros.





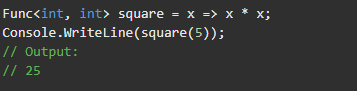
Con lo cual producimos un texto como ese, que es una estructura en JSON

Lambda

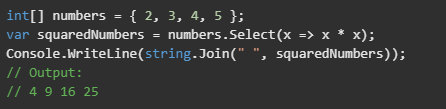
Una expresión lambda es una sintaxis para crear funciones anónimas en línea. Una expresión lambda es una función anónima que puede usar para crear delegados o tipos de árbol de expresiones. Al utilizar expresiones lambda, puede escribir funciones locales que pueden pasarse como argumentos o devolverse como el valor de las llamadas a funciones.

Una expresión lambda se crea utilizando el operador => . Ponga los parámetros en el lado izquierdo del operador. En el lado derecho, ponga una expresión que pueda usar esos parámetros; esta expresión se resolverá como el valor de retorno de la función. Más raramente, si es necesario, se puede usar un {code block} completo en el lado derecho. Si el tipo de retorno no es nulo, el bloque contendrá una declaración de retorno.

Cualquier expresión lambda se puede convertir en un tipo de delegado . El tipo de delegado al que se puede convertir una expresión lambda se define por los tipos de sus parámetros y el valor de retorno. Si una expresión lambda no devuelve un valor, se puede convertir a uno de los Actiontipos de delegado; de lo contrario, se puede convertir a uno de los Functipos de delegado. Por ejemplo, una expresión lambda que tiene dos parámetros y no devuelve ningún valor se puede convertir en un delegado Action<T1,T2> . Una expresión lambda que tiene un parámetro y devuelve un valor se puede convertir en un delegado Func<T,TResult> . En el siguiente ejemplo, la expresión lambda x => x \* x, que especifica un parámetro con nombre xy devuelve el valor del cuadrado, se asigna a una variable de tipo delegado:



Puede usar expresiones lambda en cualquier código que requiera instancias de tipos delegados o árboles de expresión, por ejemplo, como un argumento para el método Task.Run(Action) para pasar el código que debe ejecutarse en segundo plano. También puede usar expresiones lambda cuando escribe LINQ en C# , como muestra el siguiente ejemplo:



*El cuerpo de una expresión lambda puede consistir en una llamada de método. Sin embargo, si está creando árboles de expresión que se evalúan fuera del contexto de .NET Common Language Runtime (CLR), como en SQL Server, no debe usar llamadas a métodos en expresiones lambda. Los métodos no tendrán significado fuera del contexto de .NET Common Language Runtime (CLR).*

**Parámetros de entrada de una expresión lambda**

Encierra los parámetros de entrada de una expresión lambda entre paréntesis. Especifique cero parámetros de entrada con paréntesis vacíos:



Si una expresión lambda tiene solo un parámetro de entrada, los paréntesis son opcionales: 

Dos o más parámetros de entrada están separados por comas:



A veces, el compilador no puede inferir los tipos de parámetros de entrada. Puede especificar los tipos explícitamente como se muestra en el siguiente ejemplo:



**Expresiones lambda y tuplas**

A partir de C# 7.0, el lenguaje C# proporciona soporte integrado para tuplas . Puede proporcionar una tupla como argumento para una expresión lambda y su expresión lambda también puede devolver una tupla. En algunos casos, el compilador de C# utiliza la inferencia de tipos para determinar los tipos de componentes de tupla.

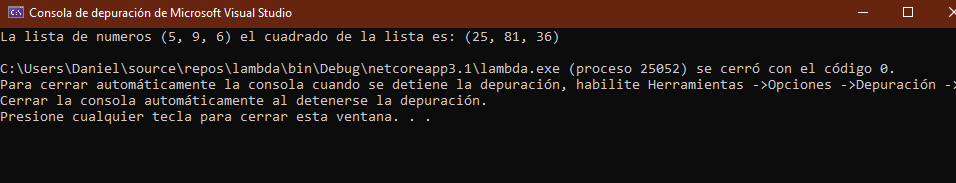
Una tupla se define encerrando una lista delimitada por comas de sus componentes entre paréntesis. El siguiente ejemplo utiliza una tupla con tres componentes para pasar una secuencia de números a una expresión lambda, que duplica cada valor y devuelve una tupla con tres componentes que contiene el resultado de las multiplicaciones de las mismas para obtener el cuadrado de los valores.

Func<(int, int, int), (int, int, int)> doubleThem = ns => (ns.Item1 \* ns.Item1, ns.Item2 \* ns.Item2, ns.Item3\* ns.Item3);

var numbers = (5,9, 6);

var doubledNumbers = doubleThem(numbers);

Console.WriteLine($"La lista de numeros {numbers} el cuadrado de la lista es: {doubledNumbers}")



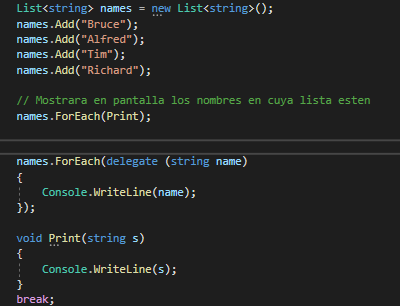
Action, function, delegates

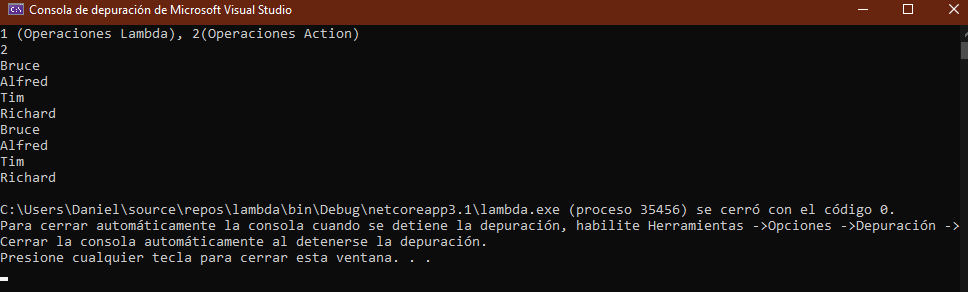
**Action**

Encapsula un método que tiene un único parámetro y no devuelve ningún valor.

Action<T>, es un delegado genérico que encapsula un método que posee un solo parámetro y no retorna un valor. Es decir, un Action<T> nos permite crear un delegado sin tener que declarar el tipo. Como un método void

El siguiente ejemplo demuestra el uso del delegado [Action<T>](https://docs.microsoft.com/en-us/dotnet/api/system.action-1?view=net-6.0) para imprimir el contenido de un objeto [List<T>](https://docs.microsoft.com/en-us/dotnet/api/system.collections.generic.list-1?view=net-6.0) . En este ejemplo, el Printmétodo se usa para mostrar el contenido de la lista en la consola. Además, el ejemplo de C# también demuestra el uso de métodos anónimos para mostrar el contenido en la consola. Tenga en cuenta que el ejemplo no declara explícitamente una variable [Action<T> .](https://docs.microsoft.com/en-us/dotnet/api/system.action-1?view=net-6.0)En su lugar, pasa una referencia a un método que toma un solo parámetro y que no devuelve un valor al método [List<T>.ForEach](https://docs.microsoft.com/en-us/dotnet/api/system.collections.generic.list-1.foreach?view=net-6.0) , cuyo único parámetro es un delegado [Action<T> .](https://docs.microsoft.com/en-us/dotnet/api/system.action-1?view=net-6.0)De manera similar, en el ejemplo de C#, un [Action<T>](https://docs.microsoft.com/en-us/dotnet/api/system.action-1?view=net-6.0)El delegado no se crea explícitamente porque la firma del método anónimo coincide con la firma del delegado [Action<T>](https://docs.microsoft.com/en-us/dotnet/api/system.action-1?view=net-6.0) que espera el método [List<T>.ForEach](https://docs.microsoft.com/en-us/dotnet/api/system.collections.generic.list-1.foreach?view=net-6.0) .





**Por lo tanto un delegado:**

Un delegado es un tipo que representa referencias a métodos con una lista de parámetros determinada y un tipo de valor devuelto. Cuando se crea una instancia de un delegado, puede asociar su instancia a cualquier método mediante una signatura compatible y un tipo de valor devuelto. Puede invocar (o llamar) al método a través de la instancia del delegado.

Los delegados se utilizan para pasar métodos como argumentos a otros métodos. Los controladores de eventos no son más que métodos que se invocan a través de delegados. Cree un método personalizado y una clase, como un control de Windows, podrá llamar al método cuando se produzca un determinado evento. En el siguiente ejemplo se muestra una declaración de delegado:

Para terminar de aclarar Action<T>, es un delegado genérico que encapsula un método que posee un solo parámetro y no retorna un valor. Es decir, un Action<T> nos permite crear un delegado sin tener que declarar el tipo.

Cualquier método de cualquier clase o struct accesible que coincida con el tipo de delegado se puede asignar al delegado. El método puede ser estático o de instancia. Esta flexibilidad significa que puede cambiar las llamadas de método mediante programación, o bien agregar código nuevo a las clases existentes.



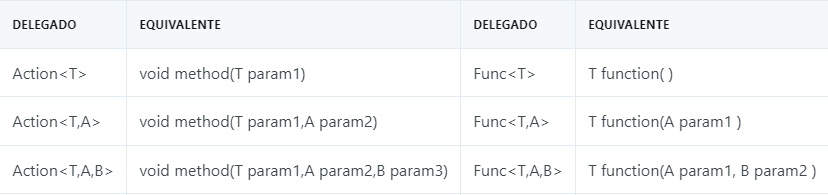
Los delegados tienen las propiedades siguientes:

* Los delegados son similares a los punteros de función de C++, pero los primeros están completamente orientados a objetos y, a diferencia de los punteros de C++ de funciones de miembro, los delegados encapsulan una instancia de objeto y un método.
* Los delegados permiten pasar los métodos como parámetros.
* Los delegados pueden usarse para definir métodos de devolución de llamada.
* Los delegados pueden encadenarse entre sí; por ejemplo, se puede llamar a varios métodos en un solo evento.
* No es necesario que los métodos coincidan exactamente con el tipo de delegado.
* Las expresiones lambda son una manera más concisa de escribir bloques de código alineado. En determinados contextos, las expresiones lambda se compilan en tipos de delegado.

**Function**

Una Func es muy parecida a una action<T>, de hecho ambos son delegados genéricos, la diferencia es que en el caso de la Func<T>, esta devuelve un valor.

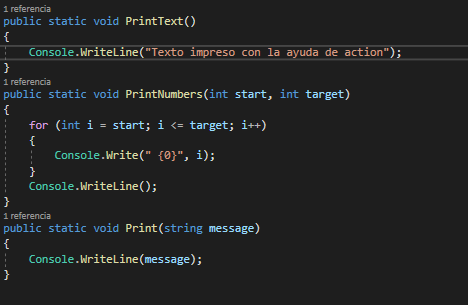
**¿Cuantos parámetros como máximo puede tener un Action<> o Func<>?**



Ejemplo de como se usan los Function y Action



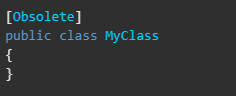
Se tuvo que crear una librería de clases para observar cómo funciona el siguiente programa



Atributes

Los atributos proporcionan una forma de asociar información con el código de forma declarativa. También pueden proporcionar un elemento reutilizable que se puede aplicar a una variedad de objetivos.

Considere el [Obsolete]atributo. Se puede aplicar a clases, estructuras, métodos, constructores y más. Declara que *el* elemento está obsoleto. Luego depende del compilador de C# buscar este atributo y realizar alguna acción en respuesta.



Los atributos proporcionan un método eficaz para asociar metadatos o información declarativa con código (ensamblajes, tipos, métodos, propiedades, etc.). Después de asociar un atributo con una entidad de programa, el atributo se puede consultar en tiempo de ejecución mediante una técnica llamada *reflexión*.

Los atributos tienen las siguientes propiedades:

* Los atributos agregan metadatos a su programa. *Los metadatos* son información sobre los tipos definidos en un programa. Todos los ensamblados de .NET contienen un conjunto específico de metadatos que describen los tipos y los miembros de tipo definidos en el ensamblado. Puede agregar atributos personalizados para especificar cualquier información adicional que se requiera. Para obtener más información, consulte [Creación de atributos personalizados (C#)](https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/attributes/creating-custom-attributes) .
* Puede aplicar uno o más atributos a ensamblajes completos, módulos o elementos de programa más pequeños, como clases y propiedades.
* Los atributos pueden aceptar argumentos de la misma forma que los métodos y las propiedades.
* Su programa puede examinar sus propios metadatos o los metadatos de otros programas utilizando la reflexión.

Crear un atributo es tan simple como heredar de la Attributeclase base.

Los atributos se pueden utilizar en varios "objetivos". Los ejemplos anteriores los muestran en las clases, pero también se pueden usar en:

* Asamblea
* Clase
* Constructor
* Delegar
* enumeración
* Evento
* Campo
* Parámetro genérico
* Interfaz
* Método
* Módulo
* Parámetro
* Propiedad
* valor de retorno
* estructura

Los atributos son utilizados por muchas herramientas y marcos. NUnit usa atributos como [Test]y [TestFixture]que usa el corredor de pruebas de NUnit. ASP.NET MVC usa atributos como [Authorize] y proporciona un marco de filtro de acción para realizar preocupaciones transversales en las acciones de MVC. PostSharp usa la sintaxis de atributos para permitir la programación orientada a aspectos en C#.

Aquí hay algunos atributos notables integrados en las bibliotecas de clases base de .NET Core:

* [Obsolete]. Este se usó en los ejemplos anteriores y vive en el espacio de Systemnombres. Es útil proporcionar documentación declarativa sobre una base de código cambiante. Se puede proporcionar un mensaje en forma de cadena y se puede usar otro parámetro booleano para pasar de una advertencia del compilador a un error del compilador.
* [Conditional]. Este atributo está en el espacio de System.Diagnosticsnombres. Este atributo se puede aplicar a métodos (o clases de atributos). Debe pasar una cadena al constructor. Si esa cadena no coincide con una #definedirectiva, el compilador de C# eliminará todas las llamadas a ese método (pero no al método en sí). Por lo general, esto se usa con fines de depuración (diagnóstico).
* [CallerMemberName]. Este atributo se puede usar en parámetros y reside en el espacio de System.Runtime.CompilerServicesnombres. Este es un atributo que se usa para inyectar el nombre del método que está llamando a otro método. Por lo general, esto se usa como una forma de eliminar las "cadenas mágicas" al implementar INotifyPropertyChanged en varios marcos de interfaz de usuario.

Los atributos aportan poder declarativo a C#, pero son una forma de código de metadatos y no actúan por sí mismos.

Servicios de Windows

¿Qué son los servicios de Windows y para qué sirven?

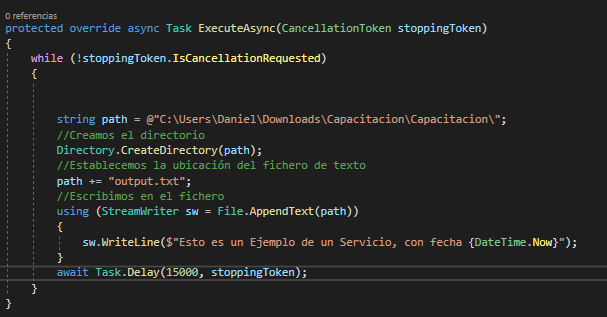
Los servicios de Microsoft Windows, anteriormente conocidos como servicios NT, le permiten crear aplicaciones de larga ejecución que se ejecutan en sesiones propias de Windows.

Los servicios de Microsoft Windows, anteriormente conocidos como servicios NT, le permiten crear aplicaciones de larga ejecución que se ejecutan en sesiones propias de Windows. Estos servicios se pueden iniciar automáticamente al arrancar el equipo, se pueden pausar y reiniciar, y no muestran ninguna interfaz de usuario. Estas características hacen que los servicios sean idóneos para su uso en un servidor o siempre que necesite una funcionalidad de larga duración que no interfiera con otros usuarios que estén trabajando en el mismo equipo. También puede ejecutar servicios en el contexto de seguridad de una cuenta de usuario específica que sea diferente del usuario conectado o de la cuenta de equipo predeterminada.

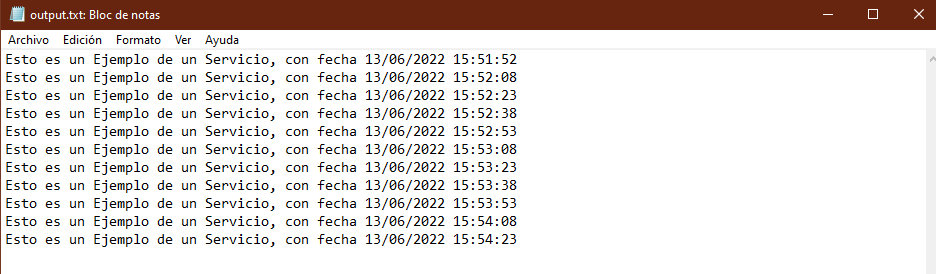
Puede crear servicios fácilmente mediante la creación de una aplicación que se instale como un servicio. Por ejemplo, suponga que desea supervisar datos del contador de rendimiento y reaccionar a valores de umbral. Puede escribir una aplicación de servicio de Windows que escuche los datos del contador de rendimiento, implemente la aplicación y comience a recopilar y analizar datos.

Cree el servicio como un proyecto de Microsoft Visual Studio y defina el código dentro de él que controle qué comandos se pueden enviar al servicio y qué acciones se deben realizar cuando se reciben esos comandos. Los comandos que se pueden enviar a un servicio incluyen iniciar, pausar, reanudar y detener el servicio; también puede ejecutar comandos personalizados.

Después de crear y compilar la aplicación, puede instalarla mediante la ejecución de la utilidad de línea de comandos InstallUtil.exe y el paso de la ruta de acceso al archivo ejecutable del servicio. A continuación, puede usar el Administrador de control de servicios para iniciar, detener, pausar, reanudar y configurar el servicio. También puede realizar muchas de estas mismas tareas en el nodo Servicios Explorador de servidores o mediante la clase .



Lo que hace este fragmento de código es que cada 15 segundos se actualiza el mensaje en un archivo txt, que guardé en la carpeta de capacitación, el cual dejó un mensaje de la siguiente manera



Esto solo es un ejemplo de como tener un servicio.