

Clase 1 - No Presencial. Por Guillermo Guastavino

Introducción al FRAMEWORK de Visual Studio .NET

Bienvenidos a todos, al curso de Framework .NET. La situación tan particular que estamos viviendo, nos obliga a encontrarnos por texto en vez de frente a frente.

Soy Guillermo Guastavino (simplemente Guillermo), y llevo más de 20 años dando los cursos de Certificación de Microsoft, y este es el 3er año que estoy al frente del curso de .NET en UNLZ, aunque precisamente ahora esté frente al monitor, en vez de mis alumnos. Soy de La Plata, y al menos me ahorré los viajes, hay que tratar de encontrar el lado positivo a todo.

Voy a tratar de habilitar un foro, para que puedan presentarse, estimo que entre ustedes se conocen, pero no sabemos cuantas clases van a ser por éste medio, así que habrá que acostumbrarse a los foros, las consultas y las respuestas en debate.

Trato de que lo que digo sea simple, sin palabras raras que no sean estrictamente requeridas, trataré de mantener el mismo principio en los textos.

Una clase

El Framework es un conjunto de clases, algo que ya vieron antes, pero como es fundamental tener claro al menos el concepto, les voy a dar unos ejemplos muy básicos para puntualizar la idea y que recuerden lo que ya aprendieron.

Como saben, una clase es un objeto, que tiene **propiedades**, **métodos** y **eventos**. Supongamos una máquina que fabrica sillas en serie. Los que programaron a la máquina, "la Clase", le dieron instrucciones para que las sillas que produzcan "las instancias del objeto silla", sean todas iguales. Uno no se sienta en la máquina (clase), se sienta en la silla (instancia), así que primero se debe crear una instancia del objeto silla, por ejemplo silla1, silla2, silla3, etc.

Todas las sillas comparten las mismas **propiedades**, el programador de la clase, quiso que la propiedad **color** sea igual a Rojo, la propiedad **tiene_respaldo** sea verdadero, la propiedad **altura** sea 1 metro, la propiedad **tiene_ruedas** sea verdadero, la propiedad **apoya_brazos** sea falso, etc. También programó **métodos**, que son las cosas que puede hacer la silla, como moverse, cambiar de altura, regular el ángulo del respaldo. Finalmente pudieron haber agregado eventos, que son acciones que se disparan ante una situación específica. Por ejemplo, si el usuario de la silla aprieta la palanca, la silla se baja, invocando al método "**cambiar_altura**".

El programador de la clase, **expuso propiedades** específicas, es decir, permitió que el usuario pudiera cambiar algunas propiedades, como la altura o cambiar el acolchado, porque tiene abrojos. Son **propiedades expuestas de la clase**. Si quisiéramos hacer algo que no está expuesto, como quitar el respaldo, sin duda romperíamos la silla. Lo mismo pasa con los **métodos y propiedades expuestas**.

La herencia

Al tener todo el código necesario para fabricar al objeto silla, podríamos crear el nuevo objeto "sillón ejecutivo", que **hereda** del objeto silla; ya que hereda todas las propiedades, métodos y eventos del objeto silla, sin necesidades de escribir ningún código nuevo, y sólo preocuparse de qué cosas agregamos o cambiamos.

Entonces, el objeto **sillón_ejecutivo** hereda de **silla** todas las características, y se le agrega apoyabrazos (propiedad apoyabrazos=true), y ya que estamos $\text{precio} = \text{precio} * 30 / 100 + \text{precio}$ (un 30% más caro para que se justifique el esfuerzo de agregar una instrucción a la clase...)

También podemos crear al objeto sillón_gerencial que hereda de sillón_ejecutivo, en dónde agregamos cromados y un respaldo con apoya cabezas.

Supongamos, que en la situación actual de cuarentena obligatoria, no nos entregan las ruedas que usamos para las sillas, y por ende para los sillones que heredan de ella, y tenemos que usar unas de plástico que teníamos guardadas. En vez de cambiar las instrucciones en las 3 clases para que usen las ruedas de plástico, sólo las cambiamos en el objeto silla, y como los otros objetos heredan de silla, heredarán automáticamente las instrucciones para usar las ruedas de plástico sin tener que hacer ninguna modificación de significativa.

Se entiende entonces la importancia de la herencia de clases. Podríamos haber empezado con un banquito, y haber heredado decenas de veces para obtener decenas de otros bancos, sillas y sillones, con sólo hacer pequeños cambios en cada nivel de la herencia.

Si quiero acceder al objeto sillón_gerencial, debo recorrer el **namespace** (el espacio de nombres) del objeto, desde el objeto más primitivo, hasta el que necesito. El namespace se construye separando a los objetos con puntos.:

silla.sillón_ejecutivo.sillón_gerencial

Esto permite saber de quienes hereda cada objeto.

Un ejemplo más "realista"

Creo primero una clase básica "**una clase primitiva**" llamada **Porcen**, que tiene las siguientes propiedades:

Número: un número dado sobre el que quiero obtener un porcentaje

Porciento: un número que representa al porcentaje que voy a utilizar sobre **Número**

Resultado: un número que es el resultado de extraer el **Porciento** % de **Número**

Para poder obtener el resultado, voy a crear el método **Calcular**. Al ejecutar el método calcular, habiendo antes cargado con valores a las propiedades **Número** y **Porciento**, podré obtener en **Resultado**, precisamente el resultado de la operación.

Si uso un lenguaje con la capacidad de crear una interfaz gráfica, podría crear un botón: **Button1**, que en el **evento click** (lo mínimo que espera un botón es que le hagan click con el mouse...), ejecute el evento **Calcular**. Para poder ingresar los valores pondría 2 cuadros de texto **Textbox1** y **Textbox2**, y para mostrar resultado, (que no debe ser alterado) pongo una etiqueta, una label; **Label1**.

Como en el ejemplo de las sillas, yo no puedo usar la clase Button, debo crear primero una instancia de la clase, a la que llamaré (en realidad es el nombre por defecto): Button1. Lo mismo para con los textboxes, creo 2 instancias de los textbox, y una instancia de la label. Luego puedo aprovechar las propiedades expuestas de los **objetos instanciados**, para cambiarles el texto, el color de la letra, la fuente, el color del fondo etc.

Ahora, creo una clase nueva **IVA_genérico**, que hereda de **Porcen**. Cambio el nombre de **Número** por **Monto**, y **Porciento** por **Porcentaje**.

Desde **IVA_genérico**, heredo para crear **IVA21**, **IVA27** e **IVA105**. Desde una misma clase, heredo 3 clases nuevas, en dónde ya no expongo la propiedad **Porcen**, ya que la dejo **Local**, y la reemplazo por 21, 27 y 10.5 respectivamente.

Por otro lado, fui creando una cadena de objetos heredados que me permitió definir finalmente al objeto Cliente, con sus propiedades Condición_Frente_al_IVA, CUIT etc. También otra larga cadena de objetos heredados que terminaron definiendo a Producto, con sus propiedades Categoría, Precio, Stock etc.

Puedo entonces crear ahora una clase que herede de IVA21 y de Producto, para crear Precio_Con_IVA, y luego heredando también de Cliente, Precio_al_Cliente_según_IVA.

Desde Cliente, Producto, Precio_Con_IVA, Precio_al_cliente_según_IVA y muchas clases más, crearía la clase Ventas, que unida a la clase Facturas, definiría la clase Facturación. Cobranzas tiene ya un larguísimo namespace de herencias, puedo unirla a Facturación, para crear Facturación_y_Cobranzas. Uniendo entonces clases de clases de clases y desde muchos namespaces diferentes, terminaría armando el Sistema de la Empresa.

Si el Gobierno considera cambiar el IVA por ejemplo a 25, no habría que asustarse demasiado por los cambios. Como miles de clases heredan de unas pocas, basta cambiar el valor 21 por 25 en esas clases primitivas, para que todo el sistema funcione con IVA 25. Es la enorme ventaja de trabajar con clases que heredan.

Lenguajes de Bajo, Medio y Alto Nivel

Seguramente saben, que un lenguaje de Bajo Nivel, es el que opera al nivel de las instrucciones del microprocesador. En vez de trabajar con instrucciones que se llaman por secuencias de ceros y unos, se reemplazan por instrucciones del lenguaje Ensamblador. Es más fácil escribir INC que su equivalente en ceros y unos. Cada instrucción "hace muy poquito", y es tedioso, pero al mismo tiempo es el más poderoso en términos de que no tengo restricciones de lenguaje. Yo empecé a los 10 años a programar con ceros y unos...

El lenguaje de Nivel Medio, lo relacionan con el Lenguaje C. Un lenguaje estructurado, en dónde una sola instrucción puede equivaler a cientos de instrucciones de código de máquina de Bajo Nivel.

Finalmente, un lenguaje de Alto Nivel, posiblemente lo relacionen con BASIC, y piensen que BASIC (que NO es Básico en inglés sino las siglas de **B**eginner's **A**ll-purpose **S**ymbolic **I**nstruction **C**ode) , está creado en C, y por eso es mucho más restrictivo, y especialmente mucho más lento que C...

El Framework .NET

Estos 3 conceptos, llevan más de 18 años de **desactualización**... En el 2002 estuve en la presentación oficial del Visual Studio .NET creado con el Framework .NET, que según Microsoft está compuesto de más de 180.000 clases. Estas clases, se heredan unas a otras en largos Namespaces, desde clases Primitivas que evolucionan, hasta que finalmente se dividen en ramas para formar los lenguajes C# (C Sharp), VB .NET (Visual Basic .NET), J# (Java Sharp) etc.

O sea... C# y Visual Basic, están al **MISMO nivel**... allá alto, contra el techo, y en vez de 3 niveles, habría 180.000. Es el Framework. Lo más importante del Framework, es que no importa que lenguaje uso "contra el techo", por debajo, son las mismas clases las que los producen, y no me tengo que limitar a rozar el techo con las instrucciones de cada lenguaje, sino que puedo sumergirme en los namespaces, para usar cualquiera de las clases intermedias o primitivas!. Programar en C#, VB, J# es exactamente lo mismo en términos de performance, porque en realidad, todos son un sólo "lenguaje", el FRAMEWORK.NET. Ya desde las primeras clases, van a notar que cada vez van a usar menos instrucciones de C# o de VB, para usar directamente clases instanciadas del Framework. El lenguaje que usen es indistinto... es sólo la cáscara de la manzana, la pulpa jugosa es el Framework.

Desde el Windows 7, todos están hechos con Framework .NET, así que de acuerdo al framework que usen y la versión de Windows, no van a necesitar en muchos casos, instalar un exe para que funcione, con solo ponerlo en una carpeta ya anda... porque el Windows tiene el framework que necesita tu exe.

Como Windows está construido sobre el Framework, vos también tenés acceso a las mismas clases que usa Windows para todas las cosas... solo hay que conocer el namespace de la clase que usa Windows para algo, para poder hacer lo mismo...

Framework te dá todo el poder, podrías pensar que C o VB te brindan... cuantas?, 300, 1000 instrucciones como mucho?... el Framework te brinda potencialmente 180.000.

Instalando el Visual Studio .NET 2019.

En UNLZ, está instalado el 2017, aunque éste es más difícil de encontrar en Microsoft, porque Microsoft quiere que uses el último, el 2019. Son muy parecidos, y al nivel del curso es igual que uses uno u otro, aunque no puede ejecutarse en 2017 un programa hecho en 2019.

El Visual Studio, lo bajás **LEGAL** y **GRATIS** de la página de Microsoft. La versión Community es gratuita para estudiantes y desarrolladores privados. Te dejo un tutorial que fui haciendo hoy paso a paso mientras instalaba el 2019:

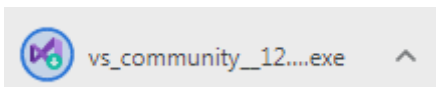
<https://visualstudio.microsoft.com/es/>



Este tipo de archivo puede dañar tu ordenador. ¿Quieres descargar vs_community_12....exe de todos modos?

Descargar

Rechazar



vs_community_128409594.1585307182
(3).exe
es seguro.

[Ver detalles](#)

Es oficial, y es seguro el exe

Tildar:

Instalando — Visual Studio Community 2019 — 16.5.1

Cargas de trabajo Componentes individuales Paquetes de idioma Ubicaciones de instalación

Web y nube (4)

- ☒ **Desarrollo de ASP.NET y web**
Compila aplicaciones web con ASP.NET Core, ASP.NET, HTML/JavaScript y contenedores, e incluye compatibilidad...
- ☐ **Desarrollo de Azure**
Proyectos, herramientas y SDK de Azure para desarrollar aplicaciones en la nube y crear recursos mediante .NET...
- ☐ **Desarrollo de Python**
Edición, depuración, desarrollo interactivo y control de código fuente de Python.
- ☐ **Desarrollo de Node.js**
Compile aplicaciones de red escalables con Node.js, un entorno de ejecución JavaScript controlado por eventos...

Móviles y de escritorio (5)

- ☒ **Desarrollo de escritorio de .NET**
Compila WPF, Windows Forms y aplicaciones de consola mediante C#, Visual Basic y F# con .NET Core y .NET...
- ☐ **Desarrollo para el escritorio con C++**
Cree aplicaciones modernas de C++ para Windows con las herramientas que prefiera, como MSVC, Clang, CMake o...
- ☒ **Desarrollo de la plataforma universal de Windows**
Cree aplicaciones para la Plataforma universal de Windows con C#, VB y, opcionalmente, C++.
- ☐ **Desarrollo para dispositivos móviles con .NET**
Compile aplicaciones multiplataforma para iOS, Android o Windows con Xamarin.
- ☐ **Desarrollo móvil con C++**
Compile aplicaciones multiplataforma para iOS, Android o Windows con C++.

Juegos (2)

- ☐ **Desarrollo de juego con Unity**
Crear juegos 2D y 3D con Unity, un entorno de desarrollo eficaz de varias plataformas.
- ☐ **Desarrollo de juegos con C++**
Utilizar toda la potencia de C++ para crear juegos profesionales con tecnología DirectX, Unreal o Cocos2d.

Otros conjuntos de herramientas (6)

- ☒ **Almacenamiento y procesamiento de datos**
Conecte, desarrolle y pruebe soluciones de datos con SQL Server, Azure Data Lake o Hadoop.
- ☐ **Aplicaciones de ciencia de datos y de análisis**
Lenguajes y herramientas para crear aplicaciones de ciencia de datos, como Python y F#.
- ☐ **Desarrollo de extensiones de Visual Studio**
Cree complementos y extensiones para Visual Studio, incluidos nuevos comandos, analizadores de código y...

Instalar durante la descarga

Visual Studio Installer

Instalados

Disponibles



Visual Studio Community 2019

Descarga y comprobación: 165 MB de 4,48 GB (13 MB /s)
3 %

Instalando: paquete 0 de 0

0 %

Creando un punto de restauración de Windows...

☒ Iniciar después de instalación

[Notas de la versión](#)

al terminar:

Visual Studio Installer

Instalados

Disponibles



Visual Studio Community 2019

16.5.1

IDE con un gran potencial, gratis para estudiantes, colaboradores de código abierto y personas

[Notas de la versión](#)

Modificar

Iniciar

Más ▾

Visual Studio

¡Hola!

Conéctese desde aquí a todos sus servicios de desarrollo.

Inicie sesión para empezar a usar los créditos de Azure, publicar código en un repositorio Git privado, sincronizar la configuración y desbloquear el IDE.

[Más información](#)

Iniciar sesión

¿No hay ninguna cuenta? ¡Créela!

De momento, no; quizás más tarde.

x

Visual Studio

Microsoft

Iniciar sesión

Correo electrónico, teléfono o Skype

¿No tiene una cuenta? [Cree una.](#)

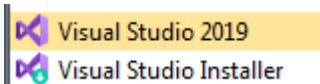
¿No puede acceder a su cuenta?

Opciones de inicio de sesión

Atrás

Siguiente

crear la sesión



luego iniciar sesión

Visual Studio

Hola, Guillermo



GdataWS@hotmail.com

[Ver su perfil de Visual Studio](#)

Estamos preparando todo para su primer uso

Esto puede tardar varios minutos.

Visual Studio 2019

Abrir recientes

Cuando abra proyectos, carpetas o archivos en Visual Studio, se mostrarán aquí para obtener un acceso rápido.

Puede anclar cualquier elemento que abra con frecuencia para que aparezca siempre en la parte superior de la lista.

Tareas iniciales



Clonar o extraer código del repositorio

Obtiene código desde un repositorio en línea, como GitHub o Azure DevOps.



Abrir un proyecto o una solución

Abre un archivo .sln o proyecto de Visual Studio local.



Abrir una carpeta local

Navegar y editar el código en cualquier carpeta

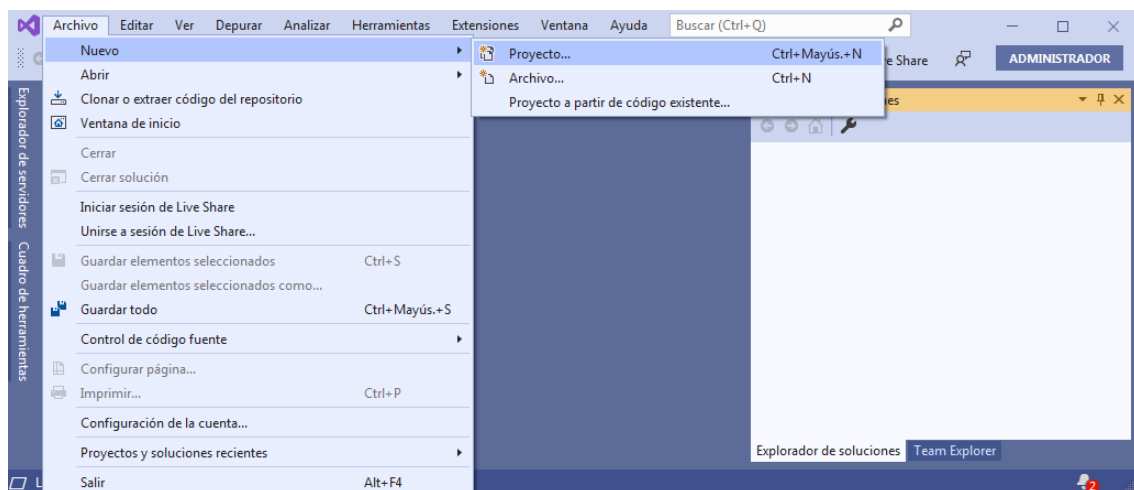
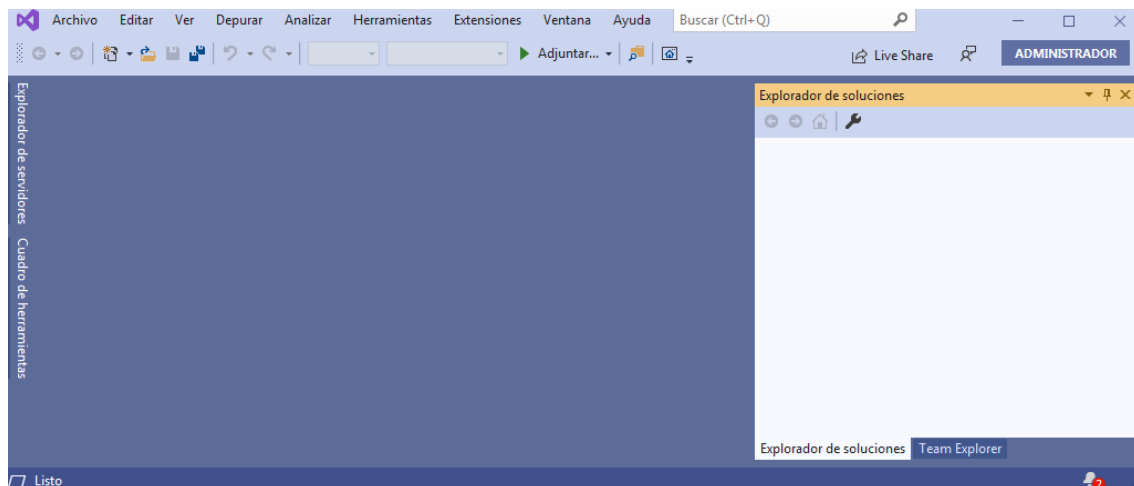


Crear un proyecto

Elija una plantilla de proyecto con la técnica scaffolding de código para comenzar.

[Continuar sin código →](#)

[Continuar sin código →](#)



Crear un proyecto

Plantillas de proyecto recientes

Aquí se mostrará una lista de las plantillas usadas recientemente.

Todos los lenguajes

Todos los lenguajes

C#

C++

F#

Java

JavaScript

Lenguaje de consulta

Python

TypeScript

Visual Basic

Todas las plataformas

Todos los tipos de proy...


Visual Basic

Windows

Linux

macOS

Consola



Aplicación web ASP.NET Core

Plantillas de proyecto para crear aplicaciones web de ASP.NET Core y API web para Windows, Linux y macOS con .NET Core o .NET Framework. Cree aplicaciones web con Razor Pages, MVC o aplicaciones de una sola página (SPA) con Angular, React o React + Redux.

C#

Linux


macOS

Windows

Nube

Servicio

Web



Aplicación Blazor

Plantillas de proyecto para crear aplicaciones Blazor que se ejecutan en el servidor en una aplicación ASP.NET Core o en el explorador en WebAssembly. Estas plantillas se pueden usar para crear aplicaciones web con interfaces de usuario (IU) dinámicas completas.

C#

Linux

macOS

Windows

Nube

Web

Siguiente

Mantenemos el suspenso hasta la Clase 2....

En la clase 2, les adelanto... vamos a ver como funciona la interface de .NET, y vamos a hacer un navegador de internet con la potencia de Google.... si no me creen... esperemos hasta el próximo sábado. Aprovechen la semana para bajar el 2019 o el 2017. Los dos usan muchos recursos de la PC. Si tienen una pc viejita, pueden probar con el 2008, yo hago muchos sistemas con 2008 o 2010, porque precisamente... muchas empresas tienen máquinas viejitas, las super están sólo en Atención al Público para impresionar...

Muy buen fin de semana para todos!