Argentina: $5,90 (recargo al interior$0,20) México: $30

USERS

Microsoft

Curso teórico y práctico de programación

Desarrollador .net 2

Con toda la potencia de **Visual Basic .NET** y **C#**

Herramientas de desarrollo | Arquitectura de software | Componentes de framework .NET | Los lenguajes de programación | Primeros pasos en Visual Basic .NET

La mejor forma de aprender a programar desde cero

SALIDA LABORAL GARANTIZADA
La mejor preparación para rendir los exámenes

★★★★★

Basado en el programa Desarrollador Cinco Estrellas de Microsoft

ISBN 978-987-1347-43-8
00002
9 789871 347438

RedUSERS

COMUNIDAD DE TECNOLOGIA

¡SUMATE!

EL SITIO Nº1 DE TECNOLOGIA

Noticias al instante // Entrevistas y coberturas exclusivas // Análisis y opinión de los máximos referentes // Reviews de productos // Trucos para mejorar la productividad // Registrate, participa, y comparte tus opiniones

SUSCRIBITE

SIN CARGO A CUALQUIERA DE NUESTROS **NEWSLETTERS** Y RECIBÍ EN TU CORREO ELECTRÓNICO TODA LA INFORMACIÓN DEL UNIVERSO TECNOLÓGICO ACTUALIZADA **AL INSTANTE**

INGRESÁ A
redusers.com/suscribirse-al-newsletter
¡Y REGISTRATE YA!

Foros · Encuestas · Tutoriales · Agenda de eventos · Videos · ¡Y mucho más!

redusers.com

Seguinos en:
www.facebook.com/redusers
www.twitter.com/redusers
www.youtube.com/redusersvideos

Compiladores

En nuestra práctica anterior, Visual Studio realizó algunas tareas para mostrar nuestra aplicación. Analicemos cuáles son.

Luego de escribir el código fuente de nuestro programa, falta un paso más para poder ejecutarlo y ver nuestro trabajo funcionando: la compilación. Éste es el proceso por el cual el código fuente (C#, Visual Basic .NET, etc.) se transforma en código que pueda ser entendido por la máquina. En .NET el resultado de la compilación es un poco diferente. Cuando compilamos un programa escrito en este lenguaje, el resultado no es código de máquina sino código en un lenguaje intermedio creado para la plataforma .NET. Este lenguaje se denomina MSIL (*Microsoft Intermediate Language*, o lenguaje intermedio de Microsoft), que se asemeja mucho a un Assembler. El código MSIL generado se almacena en un archivo denominado ensamblado (o *assembly*, en inglés). En Windows los ensamblados ejecutables tienen extensión **exe**, y los que son bibliotecas de clases o de controles tienen extensión **dll**. Como MSIL es independiente de la plataforma, se logra una ventaja fundamental: no dejar atado el programa compilado a una plataforma dada. Luego, al momento de ejecutar el programa, un componente denominado CLR (que veremos luego) se ocupa de leer el código MSIL y de convertirlo en código propio de la máquina en la que se va a ejecutar. Para compilar, el CRL se vale del JIT-Compiler (JIT es el acrónimo de *Just In Time*, que puede traducirse al español como "en el momento"). El JIT, o "jitter", se encarga de hacer la compilación final.

Luego de escribir el código, deberemos compilar el programa para poder ejecutarlo.

Compilación estándar y .NET

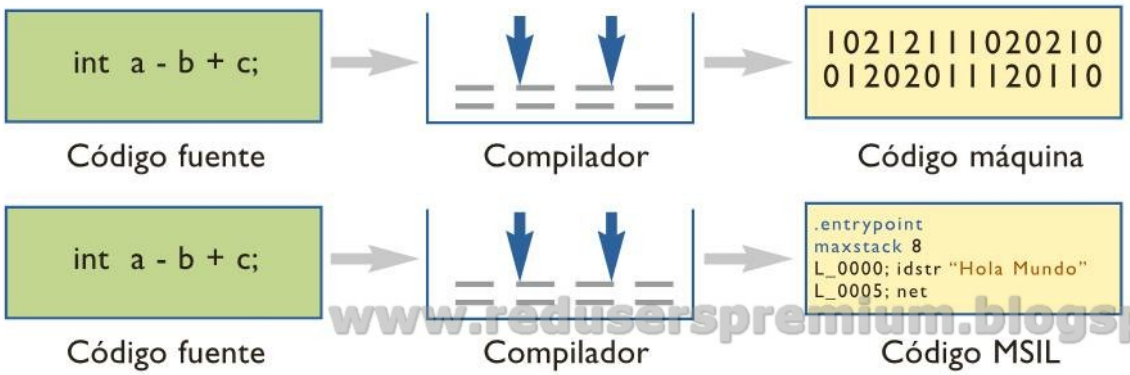


FIGURA 009 | En los lenguajes tradicionales, el compilador toma el código fuente y produce un archivo con código en lenguaje de máquina.

Herramientas de desarrollo

La plataforma .NET incluye, además de los compiladores, un conjunto de herramientas para facilitar la tarea del desarrollador.

Como pudimos ver, la compilación es una de las etapas fundamentales. Cualquier inconveniente que detecte el compilador puede convertirse en un dolor de cabeza si no contamos con herramientas que nos asistan para manejar los errores. La mejor ayuda que tendremos para esto es, sin dudas, el IDE, del cual ya conocimos algunas herramientas y funciones. Éste ofrece un potente editor con coloreo de sintaxis y tecnología IntelliSense (que muestra menús con opciones sensibles al contexto a medida que se escribe). Esto es lo que nos permitirá minimizar muchos de los errores que más frecuentemente se cometen al escribir el código.

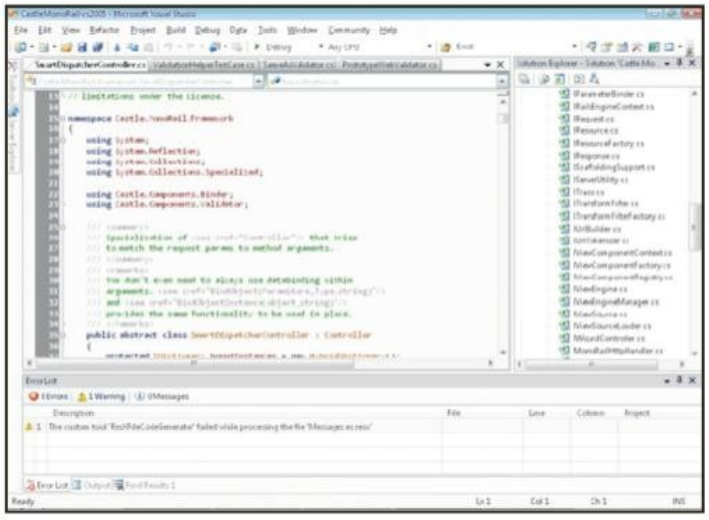


FIGURA 010 | Visual Studio proporciona herramientas para ayudar al programador en la tarea de desarrollar y depurar los programas.

Las herramientas de edición del IDE permiten evitar los errores más frecuentes que se cometen al escribir código.

Además, Visual Studio ofrece opciones para depurar el código y dejarlo libre de errores, que permiten seguir la ejecución paso a paso, monitorizar los valores de las variables e, incluso, alterar el orden de ejecución para probar determinada parte del código fuente. Otra opción interesante que provee Visual Studio es la depuración remota, consistente en seguir paso a paso la ejecución de un programa que no se encuentra en la misma computadora donde está el IDE, como un servicio en un servidor o una aplicación en un dispositivo móvil conectado a la PC. Todas estas funciones las iremos desarrollando a lo largo del curso.

Además de Visual Studio, la plataforma .NET cuenta con otras herramientas interesantes, algunas desarrolladas por Microsoft y otras, por terceros. Una muy útil es ILDasm.exe, que permite tomar un ensamblado ya compilado y obtener el código MSIL que contiene. Con ella, podemos estudiar código ya compilado, para revisar cómo queda. Ahora bien, como se puede ver el código MSIL tan fácilmente, surge de inmediato una cuestión importante: la de la propiedad del código. Es decir, alguien que tome un ensamblado creado originalmente por nosotros puede usar ILDasm para ver su contenido. Si bien el código MSIL no es comprensible para cualquier persona, es fácil ver de qué manera está implementado un algoritmo. Y no es necesario aclarar que esto puede ser muy perjudicial cuando en el ensamblado interviene información sensible involucrada, por ejemplo, con el negocio de una empresa. Más adelante, analizaremos algunas alternativas que permitirán brindar seguridad al código programado.

Versiones del framework

Para que el compilador funcione como corresponde, necesitamos un framework. Veamos de qué se trata.

A continuación, dedicaremos algunas páginas a comprender más en detalle el funcionamiento del framework .NET, y así saber cuál es, específicamente, la función que cumple. Primero, conozcamos un poco más acerca de su evolución en el tiempo.

La primera versión del framework .NET vio la luz a principios del año 2002, junto con la nueva versión de Visual Studio, que en ese momento se llamaba Visual Studio .NET. Como entorno de desarrollo, este Visual Studio fue el sucesor del popular Visual Studio 6, pero con una novedad muy atractiva: era el mismo IDE para todos los lenguajes del paquete, a diferencia de su predecesor, que tenía un IDE distinto (muy distinto) para cada uno. Esta primera versión ya tenía implementados los lenguajes C# y Visual Basic .NET, además de una versión para .NET de C++ (conocida como Manager C++). Esta primera versión ya contemplaba la creación de aplicaciones de escritorio, aplicaciones Web y servicios Web. Aproximadamente un año más tarde, en 2003, se liberó la siguiente versión, la 1.1. Su novedad más interesante fue el Compact Framework, una versión reducida para usar en dispositivos móviles que tuvieran Windows CE o Pocket PC. Con respecto a los lenguajes, se incorporó uno nuevo llamado J#, un derivado de Java para .NET.

> Sin dudas, la principal novedad de Visual Studio es utilizar el mismo IDE para todos los lenguajes.

El presente

A fines de 2005 apareció la versión 2.0 del framework .NET. Acompañando a varias novedades en los lenguajes, llegó la versión 2005 de Visual Studio, que trajo muchos cambios, tanto en los aspectos estéticos como en los funcionales. Además, incluyó un conjunto numeroso de controles y componentes para usar en aplicaciones tanto Web como Windows. El framework .NET va por la versión 3.0, liberada en 2007. Es una extensión de la versión

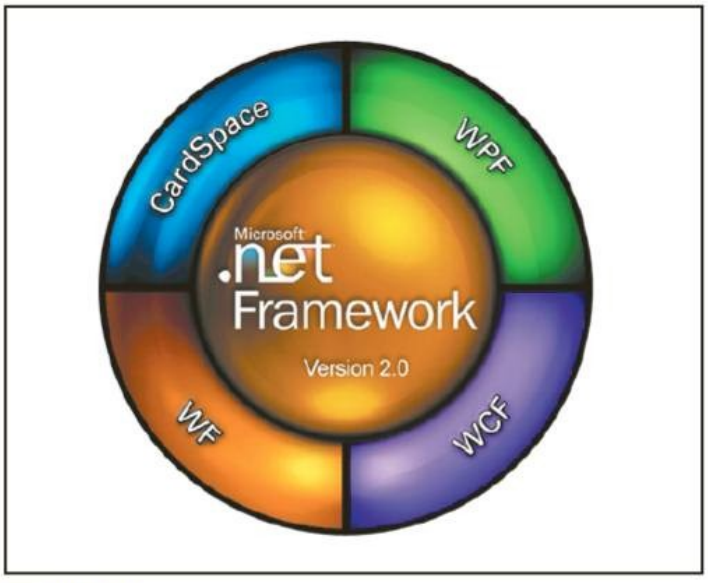


FIGURA 011 | El framework 3.0 se presenta como un conjunto de "componentes agregados" sobre el framework 2.

2.0, que incluye cuatro nuevas tecnologías: **Windows Presentation Foundation** (para la interfaz de usuario), **Windows Communication Foundation** (para la comunicación entre aplicaciones), **Windows Workflow Foundation** (para diseñar e implementar workflows) y **CardSpace** (para identidades electrónicas).

Arquitectura de software

La arquitectura nos permite desarrollar aplicaciones más complejas, sin tener que empezar desde cero.

Éste es un concepto fundamental que deberemos conocer antes de adentrarnos en el mundo de la programación. ¿A qué nos referimos precisamente cuando hablamos de arquitectura de software? La arquitectura define cómo se organizarán los distintos componentes del software, de manera que, juntos, puedan resolver el problema. Los objetivos de las distintas arquitecturas son favorecer la mayor cantidad de características del software, desde el alto rendimiento hasta la facilidad de mantenimiento, la extensibilidad (con cuánta facilidad pueden agregarse nuevos elementos) y la escalabilidad (cuánto puede crecer el software en cantidad de usuarios y datos). Veamos algunos ejemplos de arquitectura.

MONOLÍTICA

En esta arquitectura, el software se estructura como un único bloque. El resultado es un software difícil de mantener y con baja escalabilidad.

CLIENTE-SERVIDOR

Hace referencia a la arquitectura física más que a la lógica. El software se divide en dos componentes, pero no queda claro qué partes se colocan en cada uno.

EN CAPAS O NIVELES

En este caso, el software se divide en tres o más niveles, cada uno de los cuales se comunica sólo con el que tiene debajo.

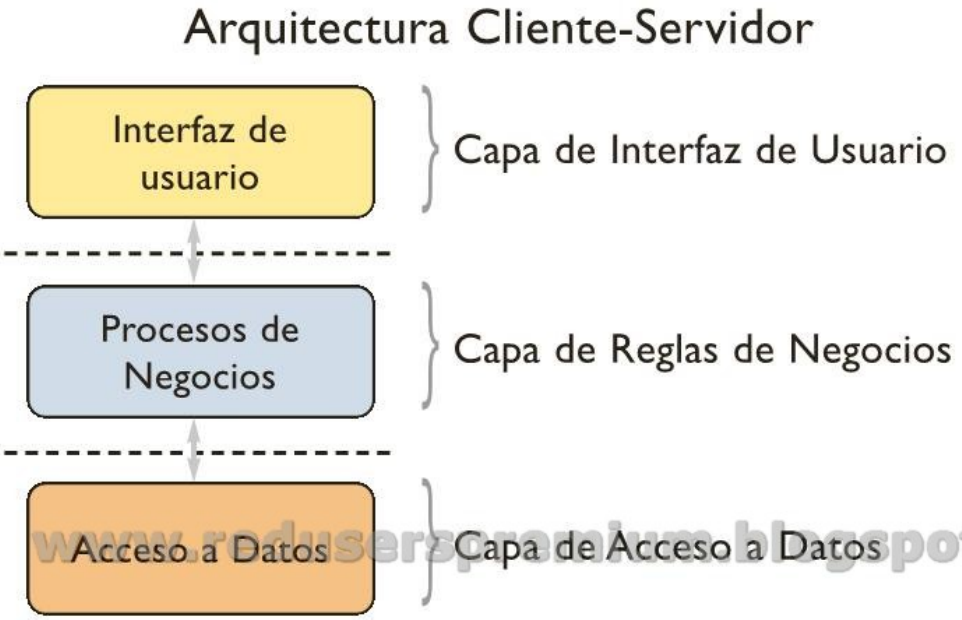


FIGURA 012 | En este diagrama podemos ver los niveles o etapas de una arquitectura Cliente-Servidor.

Componentes del framework

El framework .NET está compuesto por un gran número de elementos. Veremos cuáles son y qué función cumplen.

El framework .NET es una plataforma de desarrollo y ejecución de aplicaciones, que precisa apoyarse sobre un sistema operativo que le dé el soporte necesario para llevar adelante las tareas propias, como el manejo de archivos y de dispositivos.
Concentrándonos en el framework propiamente dicho, podemos hacer una primera separación en dos grandes partes. Por un lado, tenemos la parte distribuible, es decir, el conjunto de elementos que deben ser distribuidos junto con nuestra aplicación para que ésta pueda ser ejecutada en cualquier equipo.

> Las últimas versiones de Windows ya tienen el framework .NET preinstalado.

Las primeras versiones del framework debían instalarse en Windows para poder correr aplicaciones .NET, pero desde Windows Server 2003, los componentes necesarios ya vienen preinstalados junto con el sistema operativo, lo cual es una ayuda muy importante.

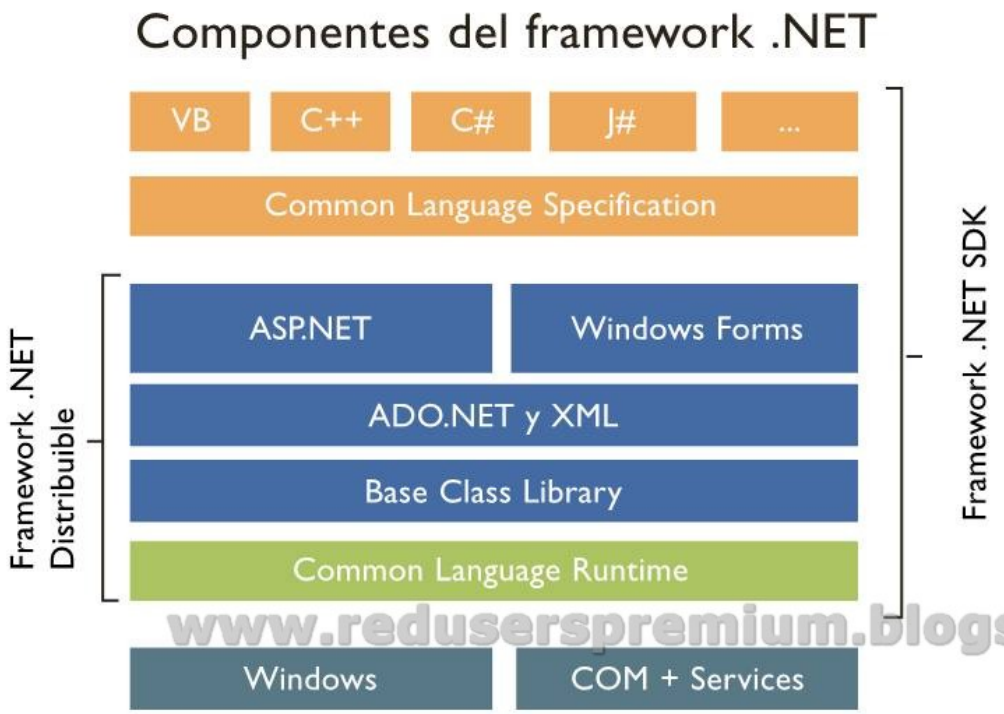


FIGURA 013 | El framework .NET está formado por varios componentes, cada uno de los cuales es responsable de un conjunto bien definido de tareas.

Common Language Runtime

La parte distribuible del framework está formada por varios componentes. El más importante es el CLR. Como vimos antes, el modelo de ejecución de .NET está dentro de los denominados "de máquina virtual", es decir que el código no se compila a instrucciones nativas de la máquina física, sino que se genera en un lenguaje intermedio. Veamos en detalle los componentes más importantes del CLR, para entender mejor sus incumbencias.

Tengamos en cuenta que para desarrollar aplicaciones, no es necesario saber al pie de la letra todos estos conceptos teóricos. Pero al igual que sucede en un automóvil, más allá de saber manejarlo, hay que saber por dónde cargarle combustible y cuáles son los componentes necesarios para su correcto funcionamiento (aceite, refrigerante, frenos, etc.).

CLASS LOADER

El class loader es el responsable de cargar en memoria el código necesario para ser ejecutado y de analizar la metadata de cada ensamblado con el objetivo de proveer la información requerida para la ejecución.

IL TO NATIVE COMPILERS

Como ya vimos, al momento de ejecutar la aplicación, el código MSIL se traduce en código nativo de la máquina. Esta tarea es llevada a cabo por el CLR a través de los compiladores de código intermedio a código nativo.

El debug engine permite depurar las aplicaciones y hacer el seguimiento durante la ejecución.

GARBAGE COLLECTOR

En .NET, el programador no necesita preocuparse por pedir memoria para los datos ni por liberarla cuando ya no la necesita. El CLR provee servicios de administración automática de memoria. El garbage collector (recolector de basura) es el responsable de liberar la memoria cuando ya no queda espacio, al desechar aquellos objetos que no se utilizan. Cada vez que la aplicación necesita más memoria y ya se llenó el espacio que le fue asignado, el CLR invoca al garbage collector para liberar espacio y, así, poder seguir satisfaciendo los requerimientos de la aplicación.

SECURITY ENGINE

Una de las premisas de diseño más importantes de .NET como tecnología es la seguridad. Cuando ejecutamos aplicaciones .NET, podemos especificar niveles de confianza dependiendo del origen de la aplicación o del lugar donde se encuentren los archivos al momento de la ejecución. También, al escribir la aplicación, podemos exigir que se cumplan ciertos requisitos en lo que respecta a la seguridad para ejecutar una porción de código. El security engine, entonces, es el responsable de asegurar que se cumplan las condiciones de seguridad necesarias.

DEBUG ENGINE

Este componente permite depurar las aplicaciones y hacer el seguimiento del código durante la ejecución. Si bien una aplicación puesta en producción puede no contener información de debug, este componente es uno de los más útiles durante su desarrollo.

TYPE CHECKER

El lenguaje MSIL fue diseñado para asegurar la seguridad de tipos. Esto significa que el MSIL, por diseño, ya nos asegura que no podremos asignar por error un valor de otro tipo a una variable (por ejemplo, asignar un texto a una variable de tipo numérico). Si bien muchos errores

por incompatibilidad de tipos pueden detectarse durante la compilación, otros deben dejarse para el momento de la ejecución, proveyendo servicios que hagan la conversión automática siempre que sea posible. El type checker es el componente del CLR que se asegura de que cada conversión automática que deba realizarse durante la ejecución sea válida y esté permitida.

EXCEPTION MANAGER

Una excepción es una situación anormal que ocurre durante la ejecución de una aplicación. Por ejemplo, si queremos escribir un archivo en disco pero éste está lleno, obtendremos una excepción. El exception manager (o administrador de excepciones) es el encargado de gestionar y provocar las excepciones. Al momento de provocar una excepción, este componente recolecta bastante información del contexto de la excepción (como el nombre de la máquina o la pila de ejecución) para facilitar su manejo o, incluso, la depuración de la aplicación.

THREAD SUPPORT

En .NET, es relativamente sencillo ejecutar código en hilos independientes; es decir, que se ejecutan al mismo tiempo entre ellos y al mismo tiempo que la aplicación principal. Este componente del CLR provee los servicios necesarios para iniciar los hilos independientes, como así también para coordinarlos, detenerlos, etc.

COM MARSHALER

Brinda soporte para interactuar con componentes COM (*Component Object Model*). Con él, es posible intercambiar datos y comunicarse con aplicaciones que tengan soporte para COM.

SOPORTE PARA LAS LIBRERÍAS DE CLASES BASE

Este componente permite integrar la aplicación con cualquier otra que soporte la librería de clases base (BCL, del inglés *Base Class Library*).

ASSEMBLIES

Los assemblies o ensamblados son la unidad básica de ejecución y despliegue en las aplicaciones .NET. Se diferencian de otras unidades de código (como los archivos dll ActiveX) en que son autodescriptivos; esto es que tienen en sí mismos toda la información que el CLR necesita para ejecutar su código.

El ensamblado se almacena en un archivo con extensión exe o dll, dependiendo del tipo. En su interior, un ensamblado contiene código MSIL junto con una sección denominada manifiesto. Éste puede verse como un encabezado que incluye información sobre el ensamblado, además de recursos adicionales al código (como mensajes de error o imágenes). La información referida al ensamblado se llama metadata (o metadatos) y, como dijimos, es utilizada por el CLR para dar soporte a los servicios en tiempo de ejecución. Entre la información que contiene la metadata, podemos mencionar la versión del assembly, el idioma por defecto, información de firma digital para proteger la identidad del código (para asegurar que no fue modificado por alguien no autorizado), los ensamblados que necesita para funcionar, etc.

En este punto, debemos tener en cuenta que es muy importante la información de la versión, ya que en una misma computadora podemos tener más de una versión de un ensamblado, y el CLR es capaz de utilizar la que sea apropiada para cada aplicación que lo necesite. Una característica muy interesante de los ensamblados es que no requieren ser registrados en ningún lugar para estar disponibles,

Es importante tener en cuenta que los espacios de nombres son una división lógica, no física.

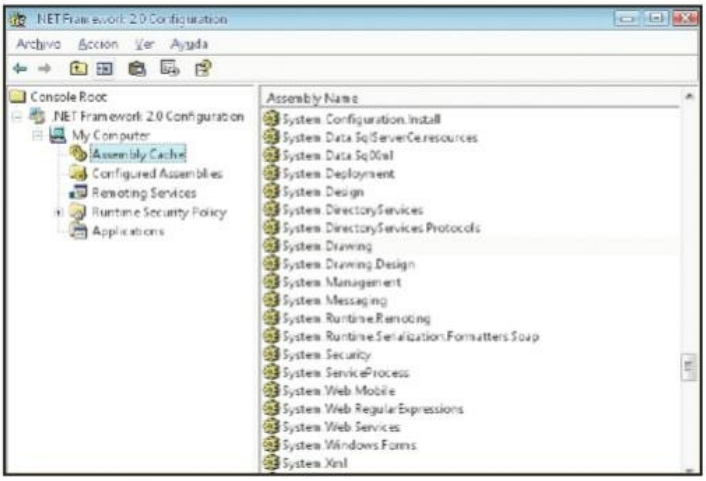


FIGURA 014 | La GAC contiene una lista de los assemblies públicos que están en la computadora. Los ensamblados se colocan en la GAC para no tener que copiarlos a las carpetas de cada aplicación.

tal como sucedía con los componentes COM. Esto, junto con la característica de autodescripción, facilita rotundamente la instalación y la actualización de aplicaciones, ya que para instalar una aplicación basta con copiar en una carpeta todos los ensamblados que la conforman.

Assemblies públicos y privados

Una aplicación .NET depende de varios ensamblados. Para encontrar aquellos que son necesarios, el CLR busca en la carpeta donde está el assembly principal y, si no lo encuentra, lo busca en un repositorio global denominado *Global Assembly Cache* (GAC), que es una especie de directorio de ensamblados compartidos. Un ensamblado que se encuentra en la GAC puede ser utilizado por más de una aplicación, sin necesidad de copiarlo a su carpeta. Los assemblies que están en la GAC se denominan assemblies públicos, en tanto que los que están en la misma carpeta se llaman privados.

La biblioteca de clases del framework

El framework .NET incluye un gran número de clases y de componentes que ayudan al programador en su tarea diaria. Estas clases integran la denominada **biblioteca de clases del framework**, o FCL por sus siglas en inglés (*.NET Framework Class Library*). La FCL contiene los bloques básicos para construir aplicaciones, y provee clases para realizar la mayoría de las tareas comunes en cualquier programa, como acceso a archivos, manejo de textos, lectura y navegación de archivos XML, conexión a bases de datos, etc. Dada la cantidad de clases que componen la FCL, los creadores del framework decidieron organizarlas en una estructura jerárquica que permita encontrar fácilmente la que se está buscando. Esta estructura de nombres se denomina **namespace** (espacio de nombres). Los namespaces, además, permiten evitar conflictos de nombres, ya que podemos tener dos clases que se llamen de la misma manera, pero en namespaces diferentes.

Common Language Specification

Una de las premisas de diseño de .NET fue ser independiente del lenguaje utilizado para desarrollar, y el hecho de que, desde un lenguaje, se pueda acceder fácilmente a librerías y clases escritas en otros. Para eso se creó la **especificación de lenguaje común**, o CLS por sus siglas en inglés (*Common Language Specification*). La CLS define ciertas características que debe tener un lenguaje para ser compatible con el CLR. Si bien Microsoft provee algunos lenguajes para .NET, la CLS hace posible que otras compañías creen nuevos lenguajes que permitan escribir aplicaciones que corran sobre el framework .NET.

Funcionamiento del CLR

En esta etapa veremos cómo .NET realiza la conversión del código fuente a un lenguaje intermedio para que funcione en el sistema.

Como vimos antes, las aplicaciones .NET son compiladas a un lenguaje intermedio, con el objetivo de independizarlas de la plataforma final donde serán ejecutadas. Esto se denomina CLI (*Common Language Infrastructure*, o infraestructura de lenguaje común). No debemos confundir el CLI con el CLR: el CLI es una **especificación**, no una **implementación**; y el CLR, si bien cumple con esta especificación, tiene aspectos que van más allá de ella. Hay que tener presente que el CLI es independiente de .NET; es una especificación que puede usarse para desarrollar otras plataformas y tecnologías. El .NET de Microsoft, de hecho, es un superconjunto del CLI, ya que está implementado de manera de cumplir con todas las especificaciones del estándar.

El modelo de ejecución

El modelo de desarrollo y ejecución de .NET es de los denominados "en dos etapas" o de "compilación diferida". El desarrollo comienza con la escritura del código fuente de la aplicación en alguno de los lenguajes con soporte para el framework .NET. Una vez escrito el código fuente, debe ser compilado utilizando el compilador apropiado para el lenguaje en cuestión.

Especificaciones y librerías en .NET

.NET Framework

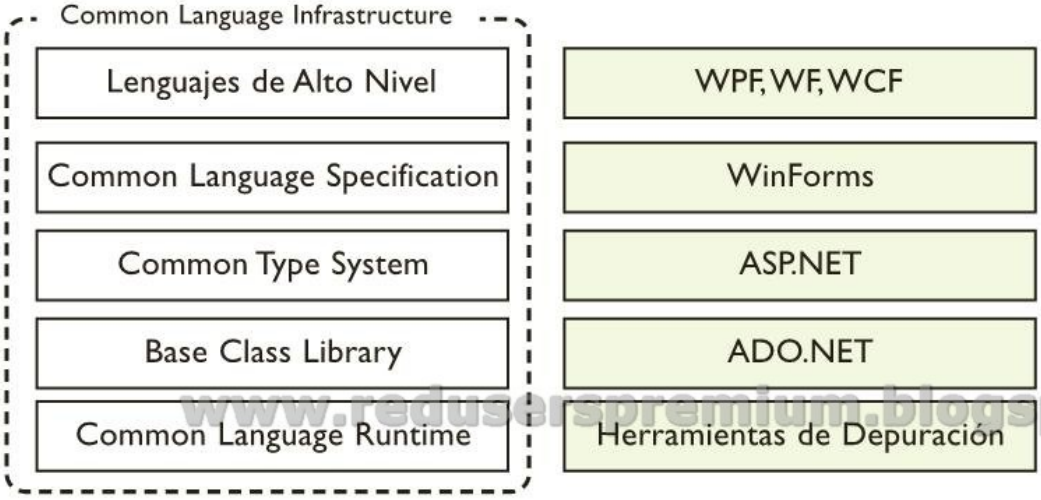


FIGURA 015 | Microsoft .NET es un superconjunto del CLI que implementa las especificaciones del estándar, y agrega un conjunto de librerías y herramientas muy útiles.

§ Características del CLI

Hubo muchas premisas que se tuvieron en cuenta a la hora de crear la especificación de CLI. Según ésta, la arquitectura de CLI, entre otras cosas, debe tener las siguientes características:

- Permitir la escritura de componentes reutilizables e interoperables independientes de la plataforma y del lenguaje de programación utilizado.
- Para facilitar la operación entre lenguajes, se debe proveer un sistema de tipos único y completo, común a todos ellos.
- Cada unidad de código o componente debe ser totalmente autodescriptiva, para facilitar su independencia y su portabilidad.
- Proveer un entorno de ejecución que permita supervisar el código para controlar y hacer cumplir políticas de seguridad.
- Diseñar toda la infraestructura basándose en metadatos, de manera tal que toda la arquitectura pueda acomodarse fácilmente a los cambios e incorporaciones y, así, se facilite la extensibilidad del modelo.
- Realizar tareas de bajo nivel, como carga de tipos en memoria o compilación a código nativo sólo cuando sea necesario (*just in time*).
- Proveer un conjunto de funcionalidades comunes que los programadores puedan utilizar para desarrollar las aplicaciones. Para estar acorde con el CLI, estas funcionalidades deben estar construidas de tal manera que su uso no deje una aplicación atada a una determinada plataforma (por ejemplo, .NET brinda métodos para conocer el carácter de fin de línea o el carácter usado para separar directorios en el disco, que varían según el sistema operativo).

El resultado de la compilación es uno o más ensamblados, de los cuales uno debe tener extensión "exe" (a menos que sea una aplicación Web) y será el punto de entrada para la ejecución de la aplicación. Como ya hemos explicado, cada assembly contiene código intermedio (MSIL) además de los recursos y metadatos. Al momento de ejecutar la aplicación, el sistema operativo detecta que se trata de un ensamblado .NET y deriva la ejecución al CLR. Entonces, éste toma el código MSIL y lo traduce a código nativo de la máquina para poder ejecutarlo, ya que las computadoras actuales no son capaces de reconocer instrucciones MSIL.

La compilación a código máquina no se efectúa completa al momento de ejecutar, sino que se realiza a medida que se la necesita. Si una porción de código nunca se ejecuta, nunca será compilada a código nativo. Para lograr la compilación por demanda, el CLR lee cada clase y, antes de comenzar la ejecución, agrega porciones de código a cada método, de manera tal que cuando un método sea ejecutado, se pueda hacer la invocación al compilador JIT (*Just In Time*). Éste traduce las instrucciones del método a código nativo, y reemplaza el código que agregó el CLR por la dirección en memoria del código nativo que acaba de generar. De este modo, en las subsecuentes llamadas al método, se ejecutará directamente código nativo, y el JIT no intervendrá.

El modelo de ejecución de .NET prevé controles de seguridad del código ejecutado, para evitar código malintencionado. Para lograrlo, antes de ordenar la compilación a código nativo, el CLR analiza la metadata del assembly y le aplica las políticas de seguridad que fueron configuradas en la computadora. Si el assembly viola alguna de ellas (por ejemplo, está firmado por una empresa en la que no se confía), se aborta la ejecución de la aplicación.

Además de los controles antes de la compilación a código nativo, el CLR verifica el código

también durante la ejecución e interviene en distintos aspectos, entre los que podemos mencionar:

- **Administración automática de memoria**
  Durante la ejecución, el CLR administra cada pedido de espacio de memoria y se encarga de liberar la memoria que ya no se está utilizando.

- **Verificación de tipos de datos**
  Si bien gran parte de la consistencia de tipos de datos puede realizarse durante la compilación, hay características de .NET que hacen necesario relegar otras verificaciones. Cada vez que se hace una conversión de tipos en tiempo de ejecución, el CLR controla si es válida.

Hay recursos que el CLR no administra porque son del sistema operativo.

- **Manejo y coordinación de hilos**
  Ciertas aplicaciones necesitan hacer procesamientos en segundo plano para poder continuar atendiendo al usuario o a otros sistemas de manera eficiente. El CLR proporciona mecanismos para facilitar la ejecución en hilos independientes, y se encarga tanto de la coordinación entre ellos como de la finalización y del control de cada hilo en particular.

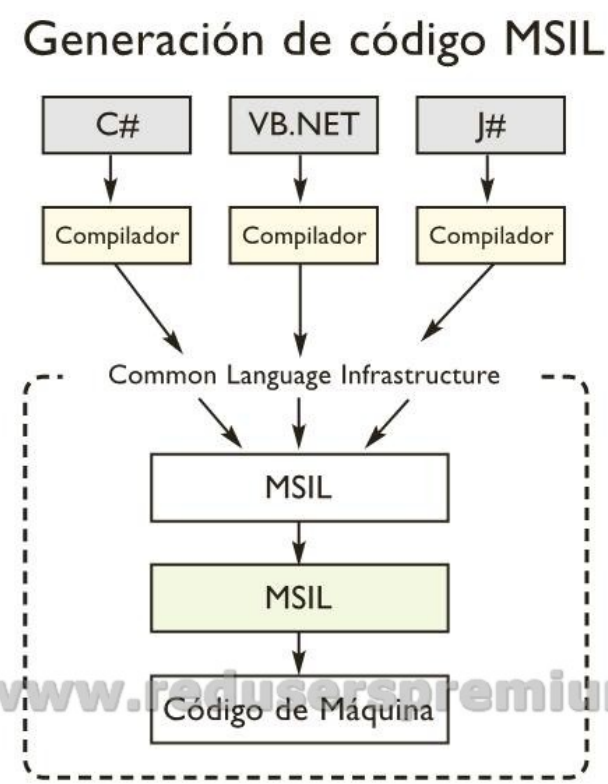


FIGURA 016 | En .NET, las aplicaciones se compilan a un lenguaje intermedio para, luego, compilarlas al código nativo de la plataforma donde se ejecutan.

Application domains

Tanto los sistemas operativos como los entornos de ejecución suelen proveer algún mecanismo para el aislamiento de las aplicaciones, necesario para asegurar que el código que corre en una aplicación no afecte al de otras. Los application domains, o fronteras de aplicación, brindan una unidad de procesamiento segura que el CLR puede usar para generar aislamiento entre aplicaciones.

Los application domains son creados por un proceso denominado CLR Host, que se ejecuta antes que el CLR y cuya función es cargar el CLR en un proceso del sistema operativo, crear los application domains necesarios dentro de ese proceso y ejecutar las aplicaciones dentro de los application domains. Esta forma de trabajo permite ejecutar varios application domains en un mismo proceso, pero con el nivel de aislamiento que existe entre procesos separados, sin incurrir en el costo que significa cambiar la ejecución de uno a otro o de hacer llamadas entre ellos. Además, la capacidad de correr varias aplicaciones dentro de un mismo proceso incrementa considerablemente la escalabilidad en los servidores. El aislamiento de aplicaciones provisto por los application domains tiene algunos beneficios interesantes. Por un lado, al estar aisladas, si una aplicación falla, no afecta a las demás. Por el otro, una aplicación puede ser detenida sin necesidad de frenar todo el proceso y las demás aplicaciones.

El Common Type System

El sistema de tipos común (CTS) define todos los tipos básicos del framework .NET. Su objetivo es especificar las reglas que rigen a cada tipo de datos en cualquier aplicación .NET.

Si bien cada lenguaje puede tener su propia sintaxis para definir los tipos de datos, el código MSIL resultante debe cumplir las reglas del CTS. Esto es fundamental para permitir la interoperabilidad de lenguajes exigida por la CLI.

El CTS define dos grandes familias de tipos de datos: los tipos por valor y los tipos por referencia. Los tipos por valor heredan de un tipo básico llamado ValueType y conforman los llamados tipos básicos o tipos primitivos. Las variables definidas con tipos por valor se almacenan directamente en la pila y, por lo tanto, son liberadas cuando se cierra el bloque de código que las definió. Por otro lado, los tipos por referencia heredan de una clase base llamada Object y se almacenan en el heap. La memoria ocupada por variables de tipos por referencia es liberada por el garbage collector.

Interpretación del CTS

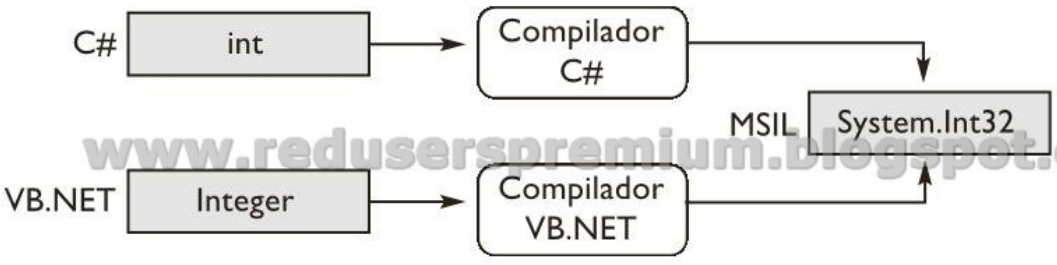


FIGURA 017 | Cada lenguaje le da distintos nombres a cada tipo de dato, pero al momento de compilar, se traducen en los mismos tipos de datos, según lo especifica el CTS.

Librerías de clases base

Como vimos anteriormente, el framework .NET brinda un conjunto de clases que podemos utilizar. Veamos cuáles son.

Las clases están ordenadas de manera jerárquica en espacios de nombres o namespaces, para que estén mejor organizadas y que resulte sencillo encontrar la que se busca. A continuación, veremos algunos de los namespaces más importantes, junto con una descripción de su contenido.

System

El espacio de nombres System es la raíz de todo el árbol de nombres. Toda clase provista por el framework está en el namespace System o en alguna de las ramas que parten de él. Además, en él se encuentran los tipos por valor básicos (como Int32, Char, String, Boolean, etc.).

System.Collections

Contiene clases que definen diversas colecciones de objetos, como listas, tablas Hash, pilas, colas y listas ordenadas. También provee interfaces que establecen las bases para la creación de nuevas clases de tipo colección y listas de objetos sobre las que se puede iterar elemento a elemento. Desde la versión 2.0 del framework, hay una rama muy interesante de este namespace llamada System.Collections.Generic, que contiene clases que representan colecciones fuertemente tipadas.

System.Configuration

Proporciona clases e interfaces para acceder y manipular por código los archivos de configuración de las aplicaciones (archivos .config).

System.Diagnostics

Proporciona clases que posibilitan interactuar con procesos del sistema, registros de eventos y contadores de rendimiento. Este espacio de nombres también provee clases que permiten depurar la aplicación y realizar un seguimiento de la ejecución del código. Por ejemplo, con clases de este namespace, podemos conocer la carga de trabajo de la CPU, o agregar código de depuración con las clases Debug y Trace.

System.Globalization

Contiene clases útiles para escribir aplicaciones con soporte para más de una cultura e idioma, lo que se conoce como internacionalización o globalización. Las clases que incluye permiten manipular formatos de fecha, de número y de monedas, como así también manejar los criterios para ordenar cadenas de textos con el fin de adecuar la lógica de la aplicación a la cultura del país donde se la está usando.

System.IO

Proporciona clases para trabajar con archivos y secuencias de datos. Mediante las clases de este namespace, no sólo podemos leer y escribir archivos, sino que también podemos manipular directorios (por ejemplo, obtener la lista de archivos). Contiene, además, un conjunto de clases que permite abstraer el medio físico con el que se trabaja, para lograr mayor flexibilidad (por ejemplo, podemos manipular una secuencia de datos sin importar si se trata de un archivo o de los datos recibidos por la red).

System.Net

Proporciona clases para acceder a recursos de redes, sobre todo, de Internet. Las clases están diseñadas de tal manera que logran independizar al

El espacio de nombres System es la raíz de todo el árbol de nombres.

programador de los protocolos subyacentes (por ejemplo, se pueden hacer invocaciones HTTP sin conocer los detalles del protocolo).

System.Relection

Este interesante espacio de nombres contiene clases que permiten recuperar información sobre los ensamblados, módulos, parámetros y cualquier otro tipo de dato de código administrado, leyendo los metadatos de los ensamblados. También provee clases para manipular y cargar tipos de datos en tiempo de ejecución y hacer invocaciones a métodos. La rama System.Reflection.Emit permite generar código en tiempo de ejecución.

System.Resources

Contiene clases e interfaces que permiten manipular recursos asociados a la referencia cultural de la aplicación, tales como cadenas de texto o imágenes almacenadas dentro de un ensamblado.

System.Security

Contiene clases relacionadas al subsistema de seguridad del CLR, que permiten conocer información del contexto de seguridad, como así también especificar requerimientos de seguridad para la ejecución del código.

System.ServiceProcess

Proporciona clases que permiten a los desarrolladores crear e instalar servicios. Los servicios son aplicaciones de larga duración que corren en segundo plano y sin interfaz de usuario.

System.Text

Incluye clases para la manipulación de cadenas de texto en distintas codificaciones

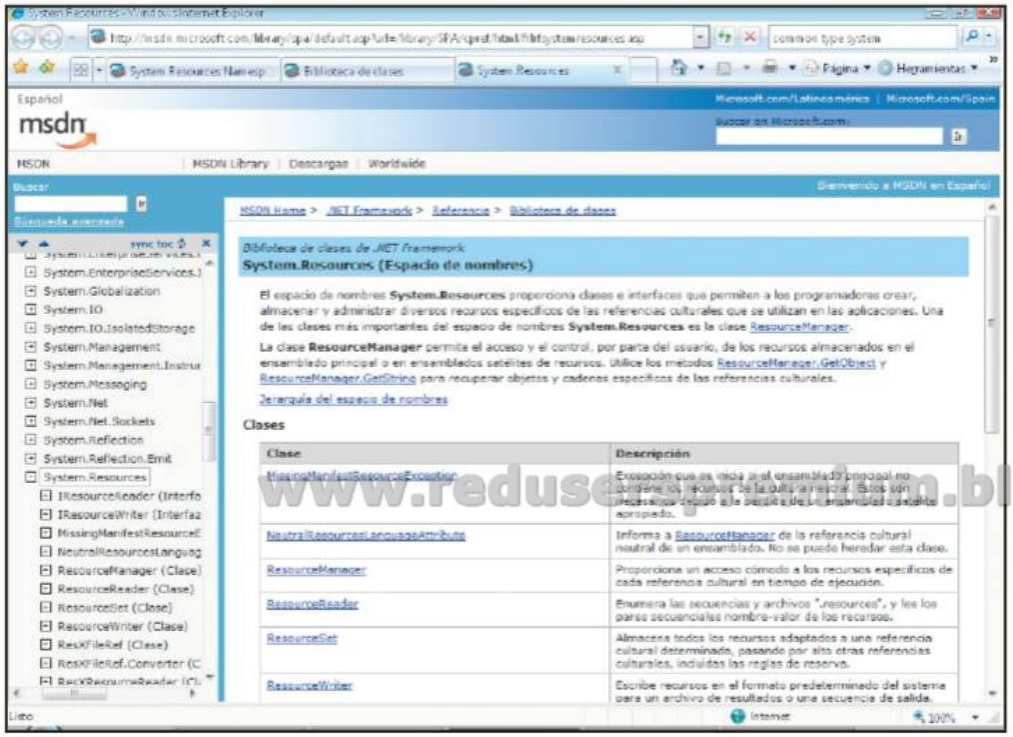


FIGURA 018 | En el sitio de MSDN hay una descripción detallada de cada uno de los espacios de nombres del framework .NET.

(ASCII, Unicode, UTF-7 y UTF-8). Provee también clases para convertir bloques de bytes en cadenas de texto, y viceversa.

System.Threading

Este espacio de nombres contiene las clases necesarias para crear aplicaciones con soporte multi hilo, como así también para su manipulación y coordinación. Incluye también una clase que permite ejecutar código en intervalos de tiempo determinados (la clase Timer).

System.Runtime.InteropServices

Proporciona clases para desarrollar aplicaciones que interactúen con componentes COM y código no administrado.

System.Runtime.Remoting

Contiene clases e interfaces que permiten la construcción de aplicaciones distribuidas. Con ellas se pueden crear objetos que se comuniquen con otros objetos que no están en el mismo application domain; incluso, que no estén en la misma computadora.

System.Runtime.Serialization

Contiene clases para serializar y deserializar objetos. La serialización es el proceso mediante el cual un objeto o un conjunto de objetos relacionados se convierten en una secuencia lineal de bytes para su almacenamiento o transmisión a otra ubicación. Por su parte, la deserialización implica recoger la información almacenada y volver a crear objetos a partir de ella. La serialización es muy utilizada en ambientes distribuidos, ya que para transmitir un objeto hacia otro application domain, es necesario serializarlo.

ADO.NET

ADO.NET es la evolución de la tecnología ADO (*ActiveX Data Objects*), tan popular en la época de Visual Basic 6. Las clases de ADO.NET brindan todo lo necesario para acceder a datos desde las aplicaciones .NET. El modelo de acceso a datos de ADO.NET permite manipular datos independientemente de la fuente original y de manera desconectada. Esto significa que podemos trabajar con los datos sin necesidad de estar conectados a la base de datos, lo cual es muy útil en ambientes distribuidos y desconectados (como el caso de servicios web). Además, ADO.NET fue diseñado de manera tal de independizar las formas de acceso del tipo de fuente de datos. Por ejemplo, una vez que aprendemos a trabajar con ADO.NET contra un SQL Server, podremos empezar otra base de datos, como Oracle, sin ningún problema. Cada conjunto de clases para acceder a un motor de base de datos en particular se denomina proveedor, y por eso se dice que ADO.NET es un modelo de acceso a datos basado en proveedores. El espacio de nombres System.Data conforma la raíz de todos los espacios de nombres de las clases de ADO.NET. Veamos qué contiene cada uno:

De memoria

Obviamente, no será necesario conocer cada una de estas librerías, sus características y funciones. Podemos tener este material como una guía de consulta permanente, hasta que nos vayamos familiarizando con cada uno de los componentes.

System.Data

En este espacio de nombres se encuentran las clases fundamentales de la arquitectura ADO.NET. Sin duda, la clase más importante es DataSet, que puede contener información de diferentes orígenes, mediante una colección de objetos DataTable. Cada DataTable contiene datos de un único origen, y está formado por una

colección de objetos DataColumn y una colección de objetos Constrain (UniqueConstrain y ForeignKeyConstrain), que permiten definir un esquema en memoria tal como si fuera una base de datos. El DataSet también puede tener una colección de objetos DataRelation para crear relaciones entre columnas de distintas tablas.

System.Data.Common

Este espacio de nombres contiene clases, en su mayoría, abstractas, compartidas y heredadas por todos los proveedores de acceso a datos. La clase que más se destaca es DataAdapter, que incluye un conjunto de comandos SQL para acceder a la base de datos, tanto para recuperar como para actualizar datos. Cada proveedor debe luego heredar de esta clase para implementar las particularidades del motor de base de datos.

System.Data.SqlClient

Este espacio de nombres contiene las clases que componen el proveedor de acceso a datos utilizado por ADO.NET (el cual veremos más adelante con mayor detalle) para SQL Server de Microsoft.

System.Data.OracleClient

Contiene las clases que implementan el proveedor de acceso a datos de ADO.NET para trabajar con bases de datos Oracle. Así como el proveedor para SQL Server implementa un SqlDataAdapter, éste implementa el OracleAdapter, también heredando de DataAdapter.

System.Data.OleDb

Brinda las clases necesarias para acceder a datos mediante el proveedor OLE DB, con el cual es posible conectarse a cualquier fuente de datos.

System.Data.Odbc

Este espacio de nombres otorga un proveedor de acceso a datos que permite trabajar prácticamente con fuentes de datos que implementen un driver ODBC (*Open DataBase Connection*).

System.Data.SqlTypes

Proporciona clases para los tipos de datos nativos de SQL Server. Estas clases ofrecen una alternativa más rápida y segura a otros tipos de datos. Las clases de este espacio de nombres sirven para evitar los errores de conversión

Modelo de acceso a datos ADO.NET

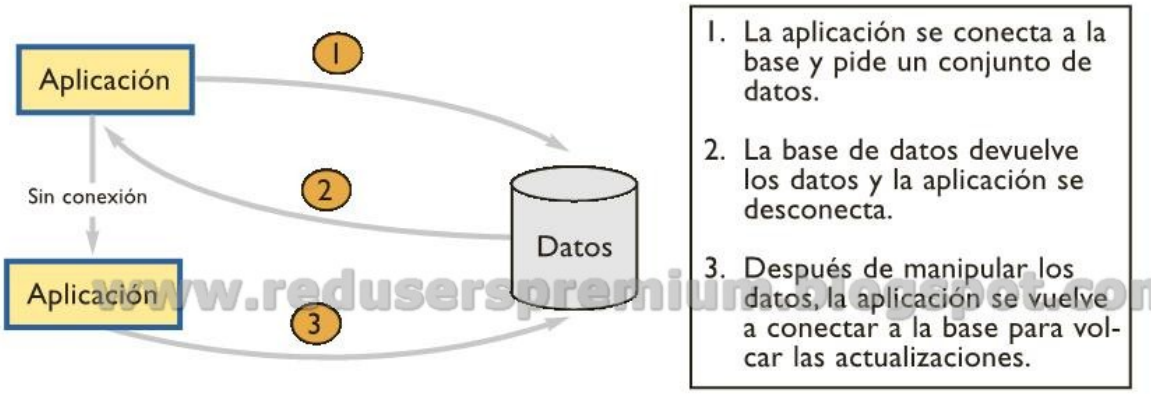


FIGURA 019 | El modelo de acceso a datos de ADO.NET proporciona una forma desconectada de trabajar, lo cual permite manipular la información sin estar conectado a la base de datos.

de tipos que pueden ocasionar una pérdida de precisión al recuperar información de la base de datos.

System.Xml

El framework .NET tiene un extenso soporte para trabajar con documentos XML como fuente de información. Mediante las clases del espacio de nombres System.Xml y sus derivados, es posible leer documentos XML, guardarlos, hacer transformaciones complejas, navegarlos de manera orientada a objetos e, incluso, acceder a partes específicas del documento a través de consultas XPath.

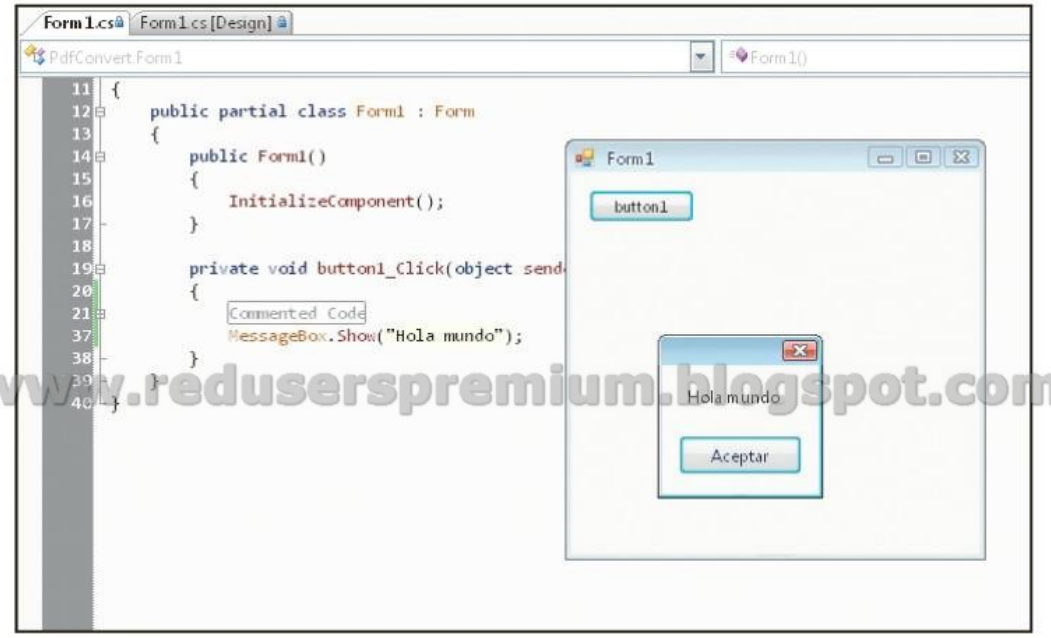
Windows Forms

Las aplicaciones .NET que corren en ventanas tradicionales de Windows se denominan aplicaciones WinForms, y el framework .NET brinda un gran soporte para su creación. El namespace más importante en este aspecto es System.Windows.Forms, que contiene clases para facilitar la creación de ventanas y de controles; en resumen, incluye todo lo necesario para la creación de interfaces de usuario basadas en ventanas.

System.Windows.Forms

Como decíamos, este espacio de nombres contiene las clases básicas para la creación de aplicaciones basadas en ventanas. Sin dudas, la clase más importante de este grupo es Form, que representa una ventana. Otra clase muy importante es Control, de la que heredan todos los controles visuales que podemos colocar en un Form (botones, listas de selección, casillas de verificación, etc.). Ésta, además, sienta las bases para crear controles propios. También hay clases que permiten imprimir textos y documentos, y manipular las propiedades de impresión (márgenes u orientación, por ejemplo).
Además, este namespace contiene clases útiles para la comunicación con el usuario a través de cuadros de diálogo. Por ejemplo, la clase OpenFileDialog permite mostrar una ventana para que el usuario seleccione el archivo que desea abrir.

FIGURA 020 | El namespace System.Windows.Forms contiene las clases necesarias para escribir aplicaciones basadas en ventanas.



Los servicios Web no son una tecnología propietaria de Microsoft, sino un estándar aprobado internacionalmente.

System.Drawing

Este namespace contiene clases que permiten acceder desde código manejado a las APIs del sistema gráfico de Windows (GDI+). Mediante las clases de System.Drawing podemos dibujar figuras y textos sobre las ventanas, pintar con distintas tramas y colores degradados, como así también manipular imágenes.

ASP.NET

Además de las aplicaciones basadas en ventanas, .NET permite construir aplicaciones basadas en exploradores de Internet, denominadas aplicaciones Web. El subconjunto de componentes del framework destinado a hacerlo se denomina ASP.NET y representa una evolución con respecto a su antecesor ASP (*Active Server Pages*). Mediante ADO.NET, es posible crear aplicaciones y sitios Web dinámicos (en el sentido de que el contenido puede cambiar en respuesta a las necesidades y acciones del usuario).

Un detalle interesante es que ASP.NET provee un modelo de desarrollo similar al modelo de creación de aplicaciones Windows, ya que brinda un conjunto de clases y controles tendientes a escribir las aplicaciones Web de una manera sencilla y muy parecida a como se escriben las aplicaciones para ventanas, aunque su funcionamiento en tiempo de ejecución sea radicalmente diferente. Cuando desarrollamos una aplicación Web, el objetivo es siempre producir código HTML para que sea interpretado por el navegador del usuario. Sin embargo, el modelo de ASP.NET permite utilizar controles de usuario que, en cierta medida, abstraen al programador del código HTML, que se genera automáticamente cuando la página se ejecuta en el servidor.

System.Web.UI

Este espacio de nombres proporciona clases e interfaces que permiten crear los controles y las páginas que aparecerán en las aplicaciones Web como elementos de interfaz de usuario. Incluye la clase Control, que proporciona todos los controles de servidor –ya sean de servidor HTML, de servidor Web y de usuario–, con un conjunto común de funciones. Además, se utiliza como clase base para la creación de nuevos controles de ASP.NET. Incluye también la clase Page, que representa la unidad básica de construcción de páginas Web dinámicas.

System.Web.Services

ASP.NET no sólo permite crear aplicaciones interactivas basadas en un navegador, sino que también da la posibilidad de crear servicios Web. Básicamente, un servicio Web es un conjunto de funcionalidades (sin interfaz de usuario) a las que se puede acceder mediante los protocolos normalmente utilizados para ingresar en la Web (HTTP y XML). Los servicios Web permiten conectar aplicaciones distribuidas de manera segura y sencilla, ya que al usar protocolos estándar, pueden colocarse detrás de un firewall que sólo permita acceso por el puerto 80 (el usado por los servidores Web).

Los servicios Web no son una tecnología propietaria de .NET ni de Microsoft, sino que son un estándar aprobado y regulado por organismos internacionales. Esto es fundamental, porque gracias a esta característica, representan una forma de integración de aplicaciones construidas con las más diversas herramientas. El espacio de nombres System.Web.Services contiene las clases base que se pueden utilizar para crear servicios Web. La más importante es WebService.

Los lenguajes

Comenzar a programar

2

Contenidos

Finalmente, estamos listos para comenzar a programar. Es por eso que en este capítulo, aprenderemos a utilizar los lenguajes más populares de la tecnología .NET para desarrollar nuestros proyectos: Visual Basic .NET y C#.

Temas tratados

- Introducción a Visual Basic .NET
- La sintaxis utilizada por VB.NET para desarrollar aplicaciones
- Las funciones propias del lenguaje
- Desarrollo de aplicaciones sencillas bajo este lenguaje
- Introducción a C#
- La sintaxis básica de C#
- Sintaxis de clases
- Las funciones propias del lenguaje
- Desarrollo de las primeras aplicaciones

Los lenguajes

Los lenguajes de programación son los que nos permitirán darle las instrucciones al sistema para que nuestra aplicación funcione.

Los lenguajes

Conoceremos los dos lenguajes: Visual Basic .NET y C#, los más utilizados en la plataforma .NET. Su aprendizaje nos permitirá desarrollar todo tipo de aplicaciones.

- Características que nos ofrece la plataforma para dar los primeros pasos
- Conocimiento de las herramientas
- Familiarizarnos para empezar a desarrollar

Sintaxis básica

A través de la sintaxis, podremos escribir las instrucciones necesarias para que el sistema interprete qué queremos que haga la aplicación que programamos.

- Operadores
- Variables
- Sentencias

Sintaxis de clase

A través de las clases, podremos organizar y estructurar mucho mejor nuestras aplicaciones. Ésta es la base fundamental para que nuestro trabajo sea lo más eficiente posible.

- Sentencia If
- Sentencia Select Case
- Sentencia For
- Sentencia While

Funciones propias

Aprenderemos a utilizar las principales funciones de los lenguajes, para que nos resulte mucho más fácil programar características en nuestros desarrollos.

- Funciones propias de cada lenguaje
- Ejemplos prácticos
- Consejos para llevar adelante una buena práctica

Ejercicios de consola en C# y Visual Basic

En esta etapa practicaremos todos los conceptos adquiridos y los utilizaremos en algunas aplicaciones. Emplearemos como control de salida de datos la consola del sistema.

- Primeros pasos en la consola
- Ejercicios prácticos
- Compilación y testeo

Visual Basic .NET

Este lenguaje continúa con la tradición de su predecesor en cuanto a su facilidad de uso. Veamos cómo utilizarlo.

A lo largo de esta sección, estudiaremos la sintaxis de Visual Basic .NET, desde los constructores fundamentales para cualquier programa, hasta la creación de clases para programas orientados a objetos.

Sintaxis básica

Siempre que escribimos código, nos encontramos ante la necesidad de dejar algún comentario explicando alguna porción o cualquier otra cosa que debamos documentar. El compilador no debe tener en cuenta estos textos, que se denominan comentarios. En Visual Basic .NET, los comentarios se escriben comenzando con una comilla simple ('), y se los puede colocar tanto en una línea independiente como al final de una línea de código.
Veamos algunos ejemplos:

```
' Este es un comentario de una línea
Console.WriteLine("Hola Mundo") 'Este es un
comentario al final
```

En la primera línea del ejemplo, tenemos un comentario que ocupa toda la línea, mientras que en la segunda, se ve una sentencia normal (que se ejecutará) seguida de un comentario. Hay que tener en cuenta que, luego de un comentario, en la misma línea no se puede escribir una sentencia ejecutable.

Operadores

Los operadores son elementos que nos permiten combinar variables, constantes o instrucciones para obtener un valor como resultado; es decir, nos permiten construir y evaluar expresiones. Una expresión está compuesta por operadores y operandos. Por ejemplo, en "2+3", 2 y 3 son los operandos, y "+" es el operador. Visual Basic .NET provee un gran número de operadores para construir expresiones de distintos tipos. En las Tabla 1 veremos los más importantes, y a lo largo del curso, iremos aprendiendo otros.

Variables

En Visual Basic .NET, las variables se declaran usando la palabra reservada Dim.
Veamos un ejemplo:

```
Dim variable1 As String
```

En este caso, estamos declarando una variable llamada **variable1** de tipo String.
Tanto la palabra clave **Dim** como la palabra clave **As** son obligatorias. Ésta es la mínima expresión para declarar una variable. Sin embargo, si queremos declarar más de una variable del mismo tipo en una sola línea, podemos hacerlo separando los nombres con coma, antes de la palabra clave As. Del mismo modo, también podemos declarar variables de otros tipos de datos usando la misma línea de sentencia Dim, colocando el nombre y el tipo separados por coma, luego de cada tipo de dato.

VB.NET no diferencia entre mayúsculas y minúsculas; es decir, no es case sensitive.

Veamos ambos casos en un ejemplo:

```
' Declaro dos variables del mismo tipo en
la misma línea
Dim variable1, Variable2 As String

' Declaro dos variables de distinto tipo en
la misma línea
Dim variable1 As String, Variable2 as Integer
```

Para asignar valor a una variable, se utiliza el operador de asignación, representado por el signo "=". La sintaxis es la siguiente:

```
Variable1 = 5
```

En este caso, la variable con nombre Variable1 queda asignada con el valor 5.

Sentencia If

La sentencia If representa la estructura de selección más simple en Visual Basic .NET, y permite ejecutar una porción de código sólo si se cumple una determinada condición. Se escribe comenzando la línea con la palabra clave **If**, seguida de la condición por evaluar y, luego, la palabra clave **Then**. Después debe ir la instrucción o grupo de instrucciones que se van a ejecutar si la condición es verdadera. Opcionalmente, podemos utilizar la palabra clave **Else** luego del bloque de código, para especificar una instrucción que se ejecutará si la condición es falsa.

```
If a > 0 Then
c = b / a
a = 1
Else
c = 0
End If
```

En este caso, si el valor de la variable **a** es mayor que **0**, a la variable **c** se le asignará el resultado de dividir el valor de la variable **b** por el de la variable **a**; en caso contrario, se le asignará el valor **0**. Si necesitamos ejecutar más de una instrucción en cualquiera de las dos partes del If, deberemos comenzar una nueva línea luego de la palabra Then y, también, luego de Else en caso de que exista. Además, en este caso, debemos cerrar la estructura con las palabras **End If** .

Tabla 1 | Operadores utilizados por Visual Basic .NET

OPERADOR	NOMBRE	DESCRIPCIÓN
+	Suma	Suma dos números.
-	Resta / Negación	Como operador binario, resta dos números. Como operador unario, representa la negación aritmética.
*	Multiplicación	Multiplica dos números.
/	División real	Divide dos números reales, y devuelve un número real (por ejemplo, 10/4 es 2,5).
\	División entera	Divide dos números enteros, y devuelve un entero (por ejemplo, 10\4 es 2).
Mod	Resto	Devuelve el resto de una división entera (por ejemplo, 10 Mod 4 es 2).
^	Potenciación	Devuelve un número elevado a otro como potencia (por ejemplo, 2^3 es 8).
&	Concatenación	Concatena dos cadenas de texto ("Hola" & "Mundo" devuelve "HolaMundo").
>	Mayor	Devuelve verdadero si el valor de la izquierda es mayor que el de la derecha.
<	Menor	Devuelve verdadero si el valor de la izquierda es menor que el de la derecha.
<>	Distinto	Devuelve verdadero si los dos operandos son distintos.
=	Igualdad	Devuelve verdadero si los dos operandos son iguales.

Observemos que en la parte de Else, si bien tenemos una sola instrucción, no podemos escribirla en la misma línea que la palabra Else, ya que con Then iniciamos un bloque If complejo. Si no tenemos un bloque Else, igualmente debemos cerrar con End If.

Sentencia Select Case

La sentencia **Select Case** puede verse como un caso generalizado de If, pero en vez de ejecutar una porción de código dependiendo de una condición lógica, divide la ejecución en un grupo de casos disjuntos (es decir que sólo se ejecutará uno de ellos).
Su sintaxis puede resumirse de esta manera:

```
Select Case Expresión
Case Caso1
  ' código si se cumple el Caso1
  ' .....
 Case Caso2, Caso3, Caso4
  ' código si se cumple el Caso2 el Caso3 o
el Caso4
  ' .....
Case Else
  ' código si no se cumple ningún caso
  ' .....
End Select
```

Cuando se alcanza una sentencia Select Case, se evalúa la expresión y se obtiene un valor. Luego, se compara el valor obtenido con el caso1; si son iguales, se ejecuta el código comprendido entre la línea siguiente a la que contiene el caso1, y la próxima línea que contenga una palabra **case** o **end select**. Si los valores no coinciden, se evalúa el caso2 y se procede de la misma manera que para el caso1. Si ninguno de los casos coincide con el valor de la expresión, y existe una línea con Case Else, se ejecuta el bloque de código correspondiente. Si no hay un Case Else, se continúa con la línea siguiente a la línea End Select. Veamos un pequeño ejemplo para entender mejor el funcionamiento de Select Case. Supongamos que tenemos una variable que contiene un número de mes, y queremos mostrar en pantalla la cantidad de días de ese mes (para simplificar, no vamos a considerar años bisiestos). El código necesario es el siguiente:

```
Select Case Numerotes
  Case 1,3,5,7,8,10,12
   Console.WriteLine(31)
  Case 2
   Console.WriteLine(28)
 Case Else
   Console.WriteLine(30)
End Select
```

Sentencia For

La sentencia **For** permite ejecutar un bloque de código una cantidad determinada y fija de veces. La sintaxis básica del constructor For es la siguiente:

```
For i=0 To 10
  'Sentencias a ejecutar
Next
```

Analicemos cada elemento de esta estructura. El constructor siempre comienza con la palabra clave **For** seguida de la variable contadora y su valor inicial (i=0). Sigue la palabra clave **To** y, por último, el valor final de la variable contadora (en el ejemplo es el valor 10). Después, en una nueva línea escribimos el bloque de código que queremos repetir (una o más líneas de código) y cerramos la estructura con la palabra clave **Next**, que también es obligatoria y marca el fin del bloque de repetición. Si queremos conocer la cantidad de veces que se ejecutará el bloque de código, debemos restar el valor final de la variable al valor inicial y sumarle 1, ya que la iteración comienza con el valor inicial. Si realizamos este ejercicio con el código del ejemplo

anterior, debemos hacer 10 (el valor final) menos 0 (el valor inicial) más 1, que nos da el valor 11; es decir, el código que esté dentro del For se ejecutará 11 veces.
En condiciones normales como las del ejemplo anterior, la variable contadora se incrementa en 1 en cada iteración. Hay situaciones en las que necesitamos que los pasos sean mayores; es decir, que en cada iteración la variable contadora se incremente en una cantidad distinta de 1. Para lograrlo, podemos usar la palabra clave **Step**, que permite indicar el paso o valor en que se incrementa el contador en cada iteración. Por ejemplo, si queremos mostrar los números pares del 2 al 10, podemos escribir lo siguiente:

```
For i=2 To 10 Step 2
  Console.WriteLine(i)
Next
```

Aquí, la porción de código Step 2 indica que en cada iteración se sume 2 al valor actual de la variable contadora. Al poder modificar el valor que se le suma a la variable contadora en cada iteración, podemos hacer que el ciclo For cuente hacia atrás, usando un valor negativo como paso.

Sentencia While

Muchas veces necesitamos repetir un bloque de código, pero no sabemos de antemano la cantidad de iteraciones que debemos hacer, porque esto dependerá de alguna condición relacionada con el bloque mismo que se ejecuta como parte de la iteración. En estos casos, el constructor For no nos sirve, dado que dentro del bloque por iterar no se puede alterar el valor de la variable contadora. Para esta situación, contamos con un constructor llamado comúnmente **While**, que permite la ejecución de un bloque de código repetidas veces mientras se cumpla una condición lógica. La sintaxis básica de la sentencia While es la siguiente:

```
While Condición
  'Hacer algo
Loop
```

Como se aprecia en el ejemplo, el bloque se debe cerrar con la palabra clave **Loop**. En ejecución, cuando el código alcanza una sentencia While, se evalúa la condición, y si es verdadera, se ejecuta el código que sigue hasta alcanzar la palabra clave Loop. Ahí se vuelve a evaluar la expresión de While, y si es verdadera, se vuelve a ejecutar el bloque de código; en caso contrario, la ejecución continúa en la línea siguiente a la palabra clave Loop.Veamos un breve ejemplo en el que se le pide al usuario que ingrese un número y se imprima en pantalla el cuadrado de éste. La acción se repite hasta que el usuario ingresa el número 0. Para hacer esto, debemos abrir Visual Studio e ir a **Archivo/Nuevo Proyecto**. En la parte central seleccionamos Aplicación de Consola. Luego de aceptar, dentro del Sub Main que se creó, escribimos el siguiente código:

```
Dim Numero As Integer
Numero = -1
While Numero <> 0
  Console.Write ("Ingrese un número. Cero
para terminar:")
  Numero = Integer.Parse(Console.ReadLine())
  Console.WriteLine(Numero □ 2)
Loop
Console.WriteLine("Hemos terminado. Presione
una tecla para salir")
Console.ReadKey()
```

No nos preocupemos por ahora por las instrucciones que no conocemos (en el futuro trabajaremos con aplicaciones Windows y no de consola). Presionando <F5>, podremos ejecutar la aplicación. Veremos que mientras ingresemos valores distintos de 0, el programa imprimirá el cuadrado del número escrito y nos pedirá otro; así sucesivamente hasta que ingresemos un 0.

USERS

CURSOS INTENSIVOS

CURSOS.REDUSERS.COM

SALIDA LABORAL

Los temas más importantes del universo de la tecnología desarrollados con la mayor profundidad y con un despliegue visual de alto impacto: Explicaciones teóricas, procedimientos paso a paso, videotutoriales, infografías y muchos recursos mas.

Brinda las habilidades necesarias para planificar, instalar y administrar redes de computadoras de forma profesional. Basada principalmente en tecnologías Cisco, es una obra actual, que busca cubrir la necesidad creciente de formar profesionales.

- 25 Fascículos
- 600 Páginas
- 3 CDs / 1 Libro

- 25 Fascículos
- 600 Páginas
- 4 CDs

Curso para dominar las principales herramientas del paquete Adobe CS3 y conocer los mejores secretos para diseñar de manera profesional. Ideal para quienes se desempeñan en diseño, publicidad, productos gráficos o sitios web.

Obra teórica y práctica que brinda las habilidades necesarias para convertirse en un profesional en composición, animación y VFX (efectos especiales).

- 25 Fascículos
- 600 Páginas
- 2 CDs / 1 DVD / 1 Libro

- 26 Fascículos
- 600 Páginas
- 2 DVDs / 2 Libros

Obra ideal para ingresar en el apasionante universo del diseño web y utilizar Internet para una profesión rentable. Elaborada por los máximos referentes en el área, con infografías y explicaciones muy didácticas.

Llegamos a todo el mundo con OCA * y DHL **

usershop@redusers.com +54 (011) 4110-8700

usershop.redusers.com.ar

** Válido en todo el mundo excepto Argentina. * Sólo válido para la República Argentina

Argentina: $5,90 (recargo al interior$0,20) México: $30

USERS

Microsoft

Curso teórico y práctico de programación

Desarrollador .net 2

Con toda la potencia de **Visual Basic .NET** y **C#**

Herramientas de desarrollo | Arquitectura de software | Componentes de framework .NET | Los lenguajes de programación | Primeros pasos en Visual Basic .NET

La mejor forma de aprender a programar desde cero

SALIDA LABORAL GARANTIZADA
La mejor preparación para rendir los exámenes

★★★★★

Basado en el programa Desarrollador Cinco Estrellas de Microsoft

ISBN 978-987-1347-43-8
00002
9 789871 347438