CAPÍTULO 1

FASES EN EL DESARROLLO DE UN PROGRAMA

En este capítulo aprenderá lo que es un programa, cómo escribirlo utilizando el lenguaje *Visual Basic .NET* y qué hacer para que el ordenador lo ejecute y muestre los resultados perseguidos.

QUÉ ES UN PROGRAMA

Probablemente alguna vez haya utilizado un ordenador para escribir un documento o para divertirse con algún juego. Recuerde que en el caso de escribir un documento, primero tuvo que poner en marcha un procesador de textos, y que si quiso divertirse con un juego, lo primero que tuvo que hacer fue poner en marcha el juego. Tanto el procesador de textos como el juego son *programas* de ordenador.

Poner un programa en marcha es sinónimo de ejecutarlo. Cuando ejecutamos un programa, nosotros sólo vemos los resultados que produce (el procesador de textos muestra sobre la pantalla el texto que escribimos; el juego visualiza sobre la pantalla las imágenes que se van sucediendo) pero no vemos el guión seguido por el ordenador para conseguir esos resultados. Ese guión es el programa.

Ahora, si nosotros escribimos un programa, entonces sí que sabemos cómo trabaja y por qué trabaja de esa forma. Esto es una forma muy diferente y curiosa de ver un programa de ordenador, lo cual no tiene nada que ver con la experiencia adquirida en la ejecución de distintos programas.

Ahora, piense en un juego cualquiera. La pregunta es ¿qué hacemos si queremos enseñar a otra persona a jugar? Lógicamente le explicamos lo que debe hacer; esto es, los pasos que tiene que seguir. Dicho de otra forma, le damos ins-

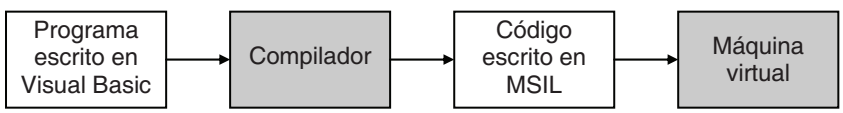
trucciones de cómo debe actuar. Esto es lo que hace un programa de ordenador. Un *programa* no es nada más que una serie de instrucciones dadas al ordenador en un lenguaje entendido por él, para decirle exactamente lo que queremos que haga. Si el ordenador no entiende alguna instrucción, lo comunicará generalmente mediante mensajes visualizados en la pantalla.

QUÉ ES Visual Basic .NET

Para entender lo que es Visual Basic .NET es imprescindible decir antes lo que es *Microsoft .NET Framework* o abreviadamente *.NET*. Se trata de un entorno de desarrollo multilenguaje diseñado por Microsoft para simplificar la construcción, distribución y ejecución de aplicaciones para Internet. Tiene fundamentalmente tres componentes: una máquina virtual (CLR: *Common Language Runtime*) que procesa código escrito en un lenguaje intermedio (MSIL: *Microsoft Intermediate Language*), una biblioteca de clases (biblioteca .NET) y ASP.NET que proporciona los servicios necesarios para crear aplicaciones *Web*.

Precisamente Visual Basic es uno de los lenguajes de programación de alto nivel que pertenecen al paquete .NET (otros lenguajes son C#, C/C++, etc.). Con Visual Basic .NET se pueden escribir tanto programas convencionales como para Internet. Las aplicaciones podrán mostrar una interfaz gráfica al usuario, o bien una interfaz de texto, como hacen las denominadas aplicaciones de consola.

El paquete .NET incluye un compilador (programa traductor) de Visual Basic que produce un código escrito en un lenguaje intermedio, común para todos los lenguajes de dicha plataforma, que será el que la máquina virtual ejecutará (esto es, cada lenguaje de la plataforma tiene su compilador que produce código correspondiente a un único lenguaje: MSIL).



Por lo tanto, MSIL es un lenguaje máquina que no es específico de ningún procesador, sino de la máquina virtual de .NET. En realidad se trata de un lenguaje de más alto nivel que otros lenguajes máquina: trata directamente con objetos y tiene instrucciones para cargarlos, guardarlos, iniciarlos, invocar a sus métodos, así como para realizar operaciones aritméticas y lógicas, para controlar el flujo de ejecución, etc. A su vez, la máquina virtual posee un recolector de basura (para eliminar los objetos cuando no estén referenciados) y proporciona traductores del lenguaje intermedio a código nativo para cada arquitectura soportada; se trata de compiladores JIT (*Just in Time*: al instante).

Por otra parte, antes de que el código MSIL pueda ser ejecutado por el procesador de nuestra máquina, debe ser convertido a código nativo. Ésta es la tarea del compilador JIT: producir código nativo para el microprocesador particular de nuestra máquina. Normalmente, el código MSIL es convertido a código nativo según se va ejecutando (el código que se va obteniendo se va guardando para que esté accesible para subsiguientes llamadas).

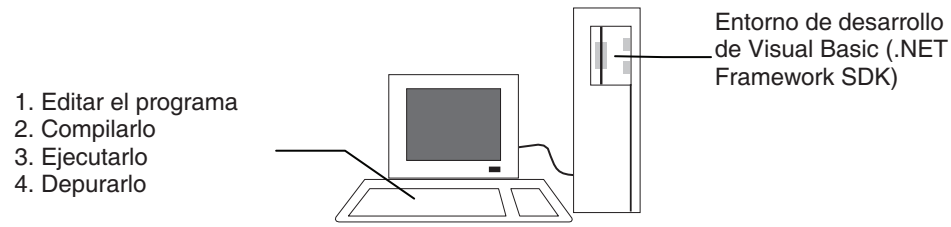
¿Dónde se consigue el paquete .NET? Puede obtenerlo a través de Internet en la dirección: *http://www.microsoft.com/downloads*.

Según lo expuesto, es fácil entender entonces que una de las ventajas significativas de Visual Basic sobre otros lenguajes de programación es que es independiente de la plataforma (lo mismo podemos decir respecto a los demás lenguajes incluidos en .NET). Esto quiere decir que el código producido por el compilador Visual Basic puede transportarse a cualquier plataforma (Intel, Sparc, Motorola, etc.) que tenga instalada una máquina virtual de .NET y ejecutarse. Pensando en Internet esta característica es crucial ya que esta red conecta ordenadores muy distintos.

REALIZACIÓN DE UN PROGRAMA EN Visual Basic .NET

En este apartado se van a exponer los pasos a seguir en la realización de un programa, por medio de un ejemplo.

La siguiente figura, muestra de forma esquemática lo que un usuario de Visual Basic .NET necesita y debe hacer para desarrollar un programa.



Evidentemente, para poder escribir programas se necesita un entorno de desarrollo Visual Basic .NET. *Microsoft*, propietario del lenguaje Visual Basic, proporciona uno de forma gratuita, *.NET Framework SDK*, que se puede obtener en la dirección de Internet:

```
http://www.microsoft.com/downloads
```

Asimismo, el CD que acompaña al libro incluye la versión de *.NET Framework SDK 2.0* para Windows, con la que podrá realizar todos los ejemplos incluidos en esta obra.

Para instalar la versión que incluye el CD mencionado en una plataforma Windows, hay que ejecutar el fichero *setup.exe*. De manera predeterminada el paquete será instalado en *\win...\Microsoft.NET*. Asimismo, se añadirá un menú *Inicio – Programas - Microsoft .NET Framework SDK* desde el que podrá visualizar la documentación relativa a .NET. Puede ver más detalles sobre la instalación al final del libro. Para otras plataformas, por ejemplo Linux, véase el apéndice D.

Sólo falta un editor de código fuente Visual Basic. Es suficiente con un editor de texto sin formato; por ejemplo el *bloc de notas* de Windows. No obstante, todo el trabajo de edición, compilación, ejecución y depuración, se hará mucho más fácil si se utiliza un entorno de desarrollo con interfaz gráfica de usuario que integre las herramientas mencionadas, en lugar de tener que utilizar la interfaz de línea de órdenes del SDK, como veremos a continuación.

Entornos de desarrollo integrados para Visual Basic .NET hay varios, pero por encima de todos destaca *Microsoft Visual Studio .NET*. No obstante, para el caso que nos ocupa, realizar los ejercicios del libro, cualquier otro entorno más simple puede servir (eche una ojeada a la carpeta *EDI* del CD que se proporciona con el libro). Por ejemplo, *Visual Basic 2005 Express* es un entorno de desarrollo proporcionado por Microsoft (gratuitamente en el momento de escribir esta obra) que le permitirá escribir aplicaciones de consola y aplicaciones con interfaz gráfica, y *Visual Web Developer 2005 Express* es otro que le permitirá construir aplicaciones para Internet con ASP.NET. Para aplicaciones empresariales es recomendable utilizar *Microsoft Visual Studio .NET* ya que incorpora utilidades que facilitan mucho los desarrollos más complejos.

Cómo crear un programa

Empecemos con la creación de un programa muy simple: el clásico ejemplo de mostrar un mensaje de saludo.

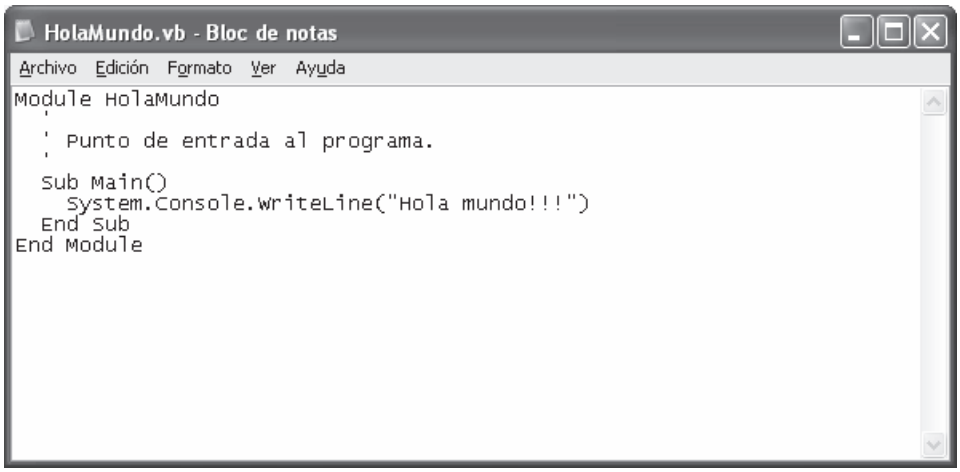
Este sencillo programa lo realizaremos desde los dos puntos de vista comentados anteriormente: utilizando la interfaz de línea de órdenes del SDK y utilizando un entorno de desarrollo integrado.

Interfaz de línea de órdenes

Empecemos por editar el fichero fuente Visual Basic .NET correspondiente al programa. Primeramente visualizaremos el editor de textos que vayamos a utili-

zar, el cual debe permitir guardar texto sin formato; por ejemplo, el *Bloc de notas* de Windows. El nombre del fichero elegido para guardar el programa en el disco, debe tener como extensión *vb*; por ejemplo *HolaMundo.vb*.

Una vez visualizado el editor, escribiremos el texto correspondiente al programa fuente. Escríbalo tal y como se muestra a continuación. Observe que una *sentencia* del lenguaje Visual Basic .NET se finaliza pulsando la tecla *Entrar (Enter o ↵)*.



¿Qué hace este programa?

Comentamos brevemente cada línea del programa anterior. No se apure si algunos de los términos no quedan muy claros ya que todos ellos se verán con detalle en capítulos posteriores.

La primera línea declara el módulo *HolaMundo*, porque el esqueleto de cualquier programa de consola Visual Basic .NET se basa en la definición de un módulo. A continuación se escribe el cuerpo del módulo encerrado entre las palabras clave **Module** y **End Module**. Ambas líneas definen el bloque de código en el que se escriben las acciones a llevar a cabo por el programa Visual Basic .NET. Con esto ya sabemos que el código de Visual Basic .NET se almacena en módulos. Aprenderemos más sobre ellos en los próximos capítulos.

Las siguientes líneas que empiezan por ' (comilla simple) son simplemente comentarios. Los comentarios no son tenidos en cuenta por el compilador, pero ayudan a entender un programa cuando se lee.

A continuación se escribe el procedimiento principal **Main**. Observe que un procedimiento se distingue por el modificador () que aparece después de su nom-

bre y que el bloque de código correspondiente al mismo, incluido entre **Sub** y **End Sub**, define las acciones que tiene que ejecutar dicho procedimiento. Cuando se compila un programa, Visual Basic .NET espera que haya un procedimiento **Main**. Este procedimiento define el punto de entrada y de salida del programa.

En el ejemplo se observa que el procedimiento **Main** llama para su ejecución al método **WriteLine** de la clase **Console** del espacio de nombres **System** de la biblioteca .NET (un espacio de nombres agrupa un conjunto de clases bajo un nombre), que escribe como resultado la expresión que aparece especificada entre comillas. Una secuencia de caracteres entre comillas se denomina *cadena de caracteres*.

Observe también que la sentencia que invoca a **WriteLine** finaliza con un retorno de carro (tecla *Entrar*), sucediendo lo mismo con la cabecera del módulo *HolaMundo*, con la cabecera del procedimiento **Main**, etc. Esto quiere decir que todas las declaraciones y sentencias en Visual Basic terminan con un retorno de carro. Resumiendo: un programa Visual Basic .NET se basa en la definición de un módulo, un módulo contiene procedimientos, además de otras definiciones, y un procedimiento, a su vez, contiene sentencias y otras definiciones, como veremos más adelante.

Guardar el programa escrito en el disco

El programa editado está ahora en la memoria. Para que este trabajo pueda tener continuidad, se debe grabar en el disco utilizando la orden correspondiente del editor. El nombre del programa fuente, programa escrito en Visual Basic .NET, puede ser diferente al del módulo que contiene. En nuestro caso, el nombre del módulo es *HolaMundo* y el nombre del fichero *HolaMundo.vb*, pero podría haber sido *saludo.vb*.

Un fichero con código Visual Basic no necesita de ningún otro fichero, como por ejemplo, los ficheros de cabecera de otros lenguajes.

Compilar y ejecutar el programa

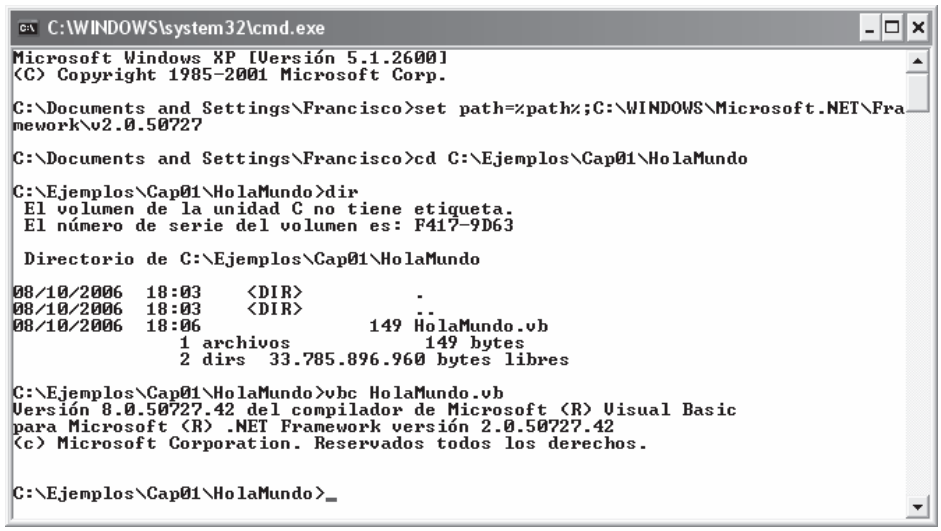
El siguiente paso es *compilar* el programa; esto es, traducir el programa fuente a código intermedio (MSIL) para posteriormente poder ejecutarlo. La figura siguiente muestra los pasos a seguir en este proceso que a continuación explicamos.

Según dijimos anteriormente, el *SDK* proporciona un programa ejecutable desde la línea de órdenes, *vbc*, para compilar cualquier programa fuente escrito en Visual Basic .NET. El resultado será un fichero de nombre igual al del fichero fuente y extensión *.exe* que almacenará el código intermedio obtenido en la tra-

ducción, código que puede ser directamente interpretado por la máquina virtual de .NET cuando requiramos ejecutar el programa. Según esto, lo primero es visualizar una ventana que muestre la línea de órdenes (para ello, los usuarios de Windows pueden ejecutar *cmd* o *command* desde la ventana *Inicio* - *Ejecutar*). Después, para informar al sistema operativo de la ubicación de la utilidad *vbc*, desde la línea de órdenes añadiremos a la variable de entorno *path* la ruta de la carpeta donde está almacenada esta utilidad y otras que utilizaremos a continuación. Por ejemplo:

```
set path=%path%;C:\WINDOWS\Microsoft.NET\Framework\vXXX
```

La expresión *%path%* representa el valor actual de la variable de entorno *path*. La ruta *C:\WINDOWS\Microsoft.NET\Framework\vXXX* es donde, en nuestro caso, están ubicadas las utilidades Visual Basic .NET. Observe que una ruta va separada de la anterior por un punto y coma. Este trabajo no será necesario hacerlo cuando al instalar el entorno .NET, esta ruta y otras sean añadidas automáticamente a la variable de entorno *path*. Puede comprobarlo ejecutando *path* desde la línea de órdenes.



A continuación, utilizando la orden *cd* nos cambiamos a la carpeta de trabajo, carpeta donde hemos guardado el fichero que deseamos compilar; en nuestro caso *C:\Ejemplos\Cap01\HolaMundo*. La orden *dir* nos permitirá ver el contenido de esta carpeta.

Finalmente, compilamos el fichero fuente que almacena el programa. La orden para compilar el programa *HolaMundo.vb* es la siguiente:

```
vbc HolaMundo.vb
```

Obsérvese que para compilar un programa hay que especificar la extensión *.vb*. El resultado de la compilación será un fichero *HolaMundo.exe* que contiene el código que ejecutará la máquina virtual de .NET.

Al compilar un programa, se pueden presentar *errores de compilación*, debidos a que el programa escrito no se adapta a la sintaxis y reglas del compilador. Estos errores se irán corrigiendo hasta obtener una compilación sin errores.

Por ejemplo, si al compilar el programa se muestra un mensaje de error como:

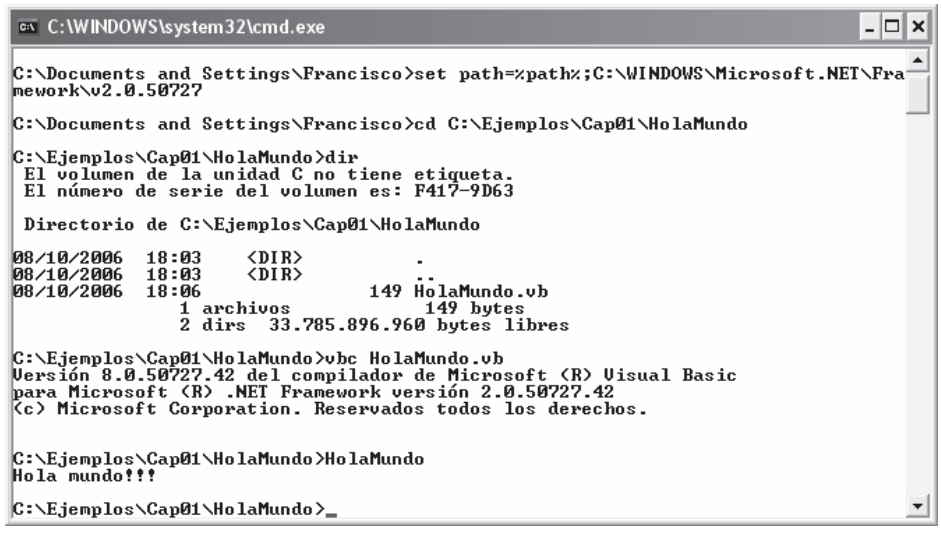
```
vbc : error BC30420: No se encontró 'Sub Main()' en 'HolaMundo'.
```

asegúrese de que el nombre del procedimiento **Main** lo ha escrito correctamente. Visual Basic es un lenguaje que no es sensible a las mayúsculas y minúsculas; por lo tanto, sería lo mismo escribir **main** que **Main**.

Para ejecutar el fichero resultante de la compilación y observar los resultados, basta con escribir en la línea de órdenes el nombre de dicho fichero, en nuestro caso *HolaMundo*, y después pulsar *Entrar*.

```
HolaMundo[Entrar]
```

En la figura siguiente se puede observar el proceso seguido para ejecutar *HolaMundo* desde la línea de órdenes. Observar que al escribir el nombre del programa se han hecho coincidir mayúsculas y las minúsculas, pero esto no es necesario. Asimismo, cabe resaltar que la extensión *.exe* no tiene que ser especificada.



Una vez ejecutado, se puede observar que el resultado es el mensaje: *Hola mundo!!!*

Biblioteca de clases

Visual Basic .NET carece de instrucciones de E/S, de instrucciones para manejo de cadenas de caracteres, etc. con lo que este trabajo queda para la biblioteca de clases provista con el compilador. Visual Basic no tiene una biblioteca de su propiedad, sino que utiliza la biblioteca .NET. Todos los lenguajes del paquete .NET utilizan esta misma biblioteca. Una biblioteca está formada por un conjunto de ficheros separados en el disco (con extensión *.dll*, *.lib*, *.tlb*, etc.) que contienen las clases que definen las tareas más comunes, para que nosotros no tengamos que escribirlas. Como ejemplo, hemos visto anteriormente el método **WriteLine** de la clase **Console** del espacio de nombres **System**. Si este método no existiera, sería labor nuestra escribir el código necesario para visualizar los resultados.

Por omisión, el compilador *vbc* sólo busca clases predefinidas en el fichero *mscorlib.dll* de la biblioteca .NET. Por lo tanto, cuando sea necesario utilizar clases guardadas en otros ficheros, habrá que utilizar la opción **/r** del compilador para especificarlos. Por ejemplo:

```
vbc /r:System.Windows.Forms.dll,System.Drawing.dll fichero.vb
```

En el código del ejemplo *HolaMundo* se puede observar que para utilizar un método de una clase de la biblioteca simplemente hay que invocarlo y pasarle los argumentos necesarios entre paréntesis. Por ejemplo:

```
System.Console.WriteLine("Hola mundo!!!")
```

Guardar el programa ejecutable en el disco

Como hemos visto, cada vez que se realiza el proceso de *compilación* del programa actual, Visual Basic genera automáticamente sobre el disco un fichero *.exe*. Este fichero puede ser ejecutado directamente desde el sistema operativo escribiendo en la línea de órdenes su nombre y pulsando la tecla *Entrar* (esta acción pondrá en marcha la máquina virtual de .NET).

Al ejecutar el programa, pueden producirse *errores durante la ejecución*. Por ejemplo, puede darse una división por cero. Estos errores solamente pueden ser detectados por Visual Basic cuando se ejecuta el programa y serán notificados con el correspondiente mensaje de error.

Hay *otro tipo de errores* que no dan lugar a mensaje alguno. Por ejemplo: un programa que no termine nunca de ejecutarse, debido a que presenta un lazo, donde no se llega a dar la condición de terminación. Para detener la ejecución en un caso como éste se tienen que pulsar las teclas *Ctrl+C* (en un entorno integrado se ejecutará una orden equivalente a *Detener ejecución*).

Depurar un programa

Una vez ejecutado el programa, la solución puede ser incorrecta. Este caso exige un análisis minucioso de cómo se comporta el programa a lo largo de su ejecución; esto es, hay que entrar en la fase de *depuración* del programa.

La forma más sencilla y eficaz para realizar este proceso es utilizar un programa *depurador*. El entorno de desarrollo de .NET proporciona para esto la utilidad *cordbg*. Éste es un depurador de línea de órdenes un tanto complicado de utilizar, por lo que, en principio, tiene escasa aceptación. Normalmente los entornos de desarrollo integrados potentes como *Microsoft Visual Studio .NET*, que anteriormente hemos mencionado, incorporan las órdenes necesarias para invocar y depurar un programa con facilidad (véase el apéndice B).

Para depurar un programa Visual Basic debe compilarlo con la opción **/debug**. Por ejemplo, desde la línea de órdenes esto se haría así:

```
vbc /debug HolaMundo.vb
```

La orden anterior genera un fichero con el mismo nombre y extensión *.pdb*. Ahora podemos ejecutar el programa Visual Basic paso a paso invocando al depurador desde la línea de órdenes así:

```
cordbg HolaMundo
```

Entornos de desarrollo integrado

Como hemos dicho anteriormente, cuando se utiliza un entorno de desarrollo integrado todo resulta más sencillo, porque las operaciones de crear un proyecto, editarlo, compilarlo, ejecutarlo y depurarlo están automatizadas. En el apéndice B se resume cómo utilizar los entornos de desarrollo anteriormente indicados.

EJERCICIOS RESUELTOS

Para practicar con un programa más, escriba el siguiente ejemplo y pruebe los resultados. Hágalo primero desde la línea de órdenes y después con el entorno de desarrollo integrado preferido por usted. El siguiente ejemplo visualiza como resultado la suma, la resta, la multiplicación y la división de dos cantidades enteras.

Abra el procesador de textos o el editor de su entorno integrado y edite el programa ejemplo que se muestra a continuación. Recuerde, el nombre del fichero

fuente, programa escrito en Visual Basic .NET, puede ser diferente al del módulo que contiene, *Aritmetica*, y debe tener extensión *.vb*.

```
Module Aritmetica
  '
  ' Operaciones aritméticas
  '
  Sub Main()
    Dim dato1, dato2, resultado As Integer

    dato1 = 20
    dato2 = 10

    ' Suma
    resultado = dato1 + dato2
    System.Console.WriteLine("{0} + {1} = {2}", dato1, dato2, resultado)
    ' Resta
    resultado = dato1 - dato2
    System.Console.WriteLine("{0} - {1} = {2}", dato1, dato2, resultado)
    ' Producto
    resultado = dato1 * dato2
    System.Console.WriteLine("{0} * {1} = {2}", dato1, dato2, resultado)
    ' Cociente
    resultado = dato1 / dato2
    System.Console.WriteLine("{0} / {1} = {2}", dato1, dato2, resultado)
  End Sub
End Module
```

Una vez editado el programa, guárdelo en el disco con el nombre *Aritmetica.vb*.

¿Qué hace este programa? Si nos fijamos en el procedimiento principal, **Main,** vemos que se han declarado tres variables enteras (de tipo **Integer**): *dato1, dato2* y *resultado*.

```
Dim dato1, dato2, resultado As Integer
```

El siguiente paso asigna el valor 20 a la variable *dato1* y el valor 10 a la variable *dato2*.

```
dato1 = 20
dato2 = 10
```

A continuación se realiza la suma de esos valores y se escriben los datos y el resultado.

```
resultado = dato1 + dato2
System.Console.WriteLine("{0} + {1} = {2}", dato1, dato2, resultado)
```

El método **WriteLine** escribe un resultado de la forma:

```
20 + 10 = 30
```

Observe que la expresión resultante está formada por cinco elementos: *dato1*, *" + "*, *dato2*, *" = "*, y *resultado*; unos elementos son numéricos y otros son constantes de caracteres. Esto se ha especificado mediante el formato *{0} + {1} = {2}*; una especificación de la forma *{número}* indica que se ha de mostrar el valor del argumento que está en la posición *número* (en el ejemplo, *dato1* es el argumento que está en la posición *0*, *dato2* está en la *1* y *resultado* en la *2*); cualquier otro carácter entre las comillas dobles, aparte de las especificaciones, se mostrará tal cual (en el ejemplo, los espacios en blanco, el + y el =).

Un proceso similar se sigue para calcular la diferencia, el producto y el cociente. Para finalizar, compile, ejecute el programa y observe los resultados.

EJERCICIOS PROPUESTOS

1. Practique la edición, la compilación y la ejecución con un programa similar al programa *Aritmetica.vb* realizado en el apartado anterior. Por ejemplo, modifíquelo para que ahora realice las operaciones de sumar, restar y multiplicar con tres datos: *dato1*, *dato2* y *dato3*. En un segundo intento, puede también combinar las operaciones aritméticas.

2. Realice el mismo ejercicio anterior, pero ahora utilizando un entorno de desarrollo integrado (EDI) análogamente a como se indica en el apéndice B.