Uso de variables de tipo valor

Contenido

Notas para el instructor

Este módulo ofrece a los estudiantes una introducción al sistema de tipos comunes (Common Type System, CTS). Los estudiantes conocerán las variables de tipo valor y también crearán y utilizarán tipos de datos definidos por el usuario.

Al final de este módulo, los estudiantes serán capaces de:

Describir los tipos de variables que se pueden emplear en aplicaciones C#.

Nombrar variables según las convenciones de nomenclatura estándar de C#.

Declarar variables utilizando tipos de datos predefinidos.

Asignar valores a variables.

Convertir variables existentes de un tipo de dato a otro.

Crear y utilizar sus propios tipos de datos.

◆ Descripción general

Objetivo del tema
Ofrecer una introducción a los contenidos y objetivos del módulo.

Explicación previa
En este módulo aprenderá a usar variables de tipo valor en C#.

- Sistema de tipos comunes (CTS)
- Nombres de variables
- Uso de tipos de datos predefinidos
- Creación de tipos de datos definidos por el usuario
- Conversión de tipos de datos

Recomendación al profesor
Si desea más información sobre los temas tratados en este módulo, consulte la documentación de la biblioteca de clases de .NET Framework y la especificación del lenguaje C# en los documentos de ayuda de Visual Studio .NET.

Todas las aplicaciones manipulan datos de alguna manera. Como desarrollador de C#, necesitará aprender a almacenar y procesar datos en sus aplicaciones. Cada vez que una aplicación tiene que almacenar datos temporalmente para usarlos durante la ejecución, esos datos se almacenan en una variable. Antes de utilizar una variable hay que definirla, y cuando se define una variable lo que se hace en realidad es reservar espacio de almacenamiento para ella, identificando su tipo de datos y dándole un nombre. Una vez definida, ya es posible asignar valores a esa variable.

En este módulo aprenderá a usar variables de tipo valor en C#. Estudiará cómo especificar el tipo de datos de las variables, cómo nombrar variables según las convenciones de nomenclatura estándar y cómo asignar valores a variables. Aprenderá también a convertir variables existentes de un tipo de dato a otro y a crear sus propias variables.

Al final de este módulo, los estudiantes serán capaces de:

Describir los tipos de variables que puede emplear en aplicaciones C#.

Nombrar variables según las convenciones de nomenclatura estándar de C#.

Declarar una variable utilizando tipos de datos predefinidos.

Asignar valores a variables.

Convertir variables existentes de un tipo de dato a otro.

Crear y utilizar sus propios tipos de datos.

Sistema de tipos comunes (CTS)

Objetivo del tema
Ofrecer una introducción a los temas tratados en esta sección.

Explicación previa
En esta sección conocerá algunos de los tipos de datos de C#, así como el sistema de tipos comunes.

- Aspectos generales del CTS
- Comparación de tipos de valor y de referencia
- Comparación de tipos de valor predefinidos y definidos por el usuario
- Tipos simples

Cada variable tiene un tipo de datos que determina los valores que se pueden almacenar en ella. C# es un lenguaje de especificaciones seguras (type-safe), lo que significa que el compilador de C# garantiza que los valores almacenados en variables son siempre del tipo adecuado.

El runtime de lenguaje común incluye un sistema de tipos comunes (Common Type System, CTS) que define un conjunto de tipos de datos predefinidos que se pueden utilizar para definir variables.

Al final de esta lección, usted será capaz de:

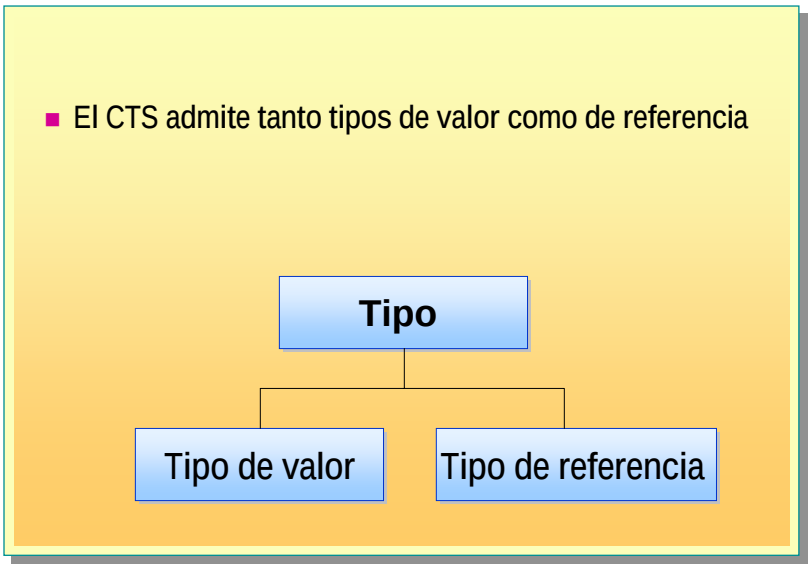
Explicar cómo funciona el CTS.

Elegir los tipos de datos más adecuados para sus variables.

Aspectos generales del CTS

Objetivo del tema
Describir el CTS.

Explicación previa
El CTS es compatible con estilos de programación procedurales y orientados a objetos. Además, el CTS admite tipos de valor y de referencia.



Al definir una variable es necesario elegir el tipo de datos correcto para ella. El tipo de datos determina los valores permitidos para esa variable, los cuales a su vez determinan las operaciones que se pueden efectuar sobre ella.

CTS

El CTS es una parte integral del runtime de lenguaje común y es compartido por los compiladores, las herramientas y el propio runtime. Es el modelo que define las reglas que sigue el runtime a la hora de declarar, usar y gestionar tipos. El CTS establece un marco que permite la integración entre lenguajes, la seguridad de tipos y la ejecución de código con altas prestaciones.

En este módulo estudiará dos tipos de variables:

Variables de tipos de valor.

Variables de tipos de referencia.

Comparación de tipos de valor y de referencia

Objetivo del tema
Describir las diferencias entre tipos de valor y de referencia.

Explicación previa
Este módulo se concentra en los tipos de valor. Entre los tipos de valor y los tipos de referencia existen considerables diferencias que los desarrolladores deben comprender.

- Tipos de valor:
  - Contienen sus datos directamente
  - Cada una tiene su propia copia de datos
  - Las operaciones sobre una no afectan a otra
- Tipos de referencia:
  - Almacenan referencias a sus datos (conocidos como objetos)
  - Dos variables de referencia pueden apuntar al mismo objeto
  - Las operaciones sobre una pueden afectar a otra

Tipos de valor

Las variables de tipos de valor contienen sus datos directamente. Cada una de ellas tiene su propia copia de los datos, por lo que las operaciones efectuadas sobre una variable no pueden afectar a otra.

Tipos de referencia

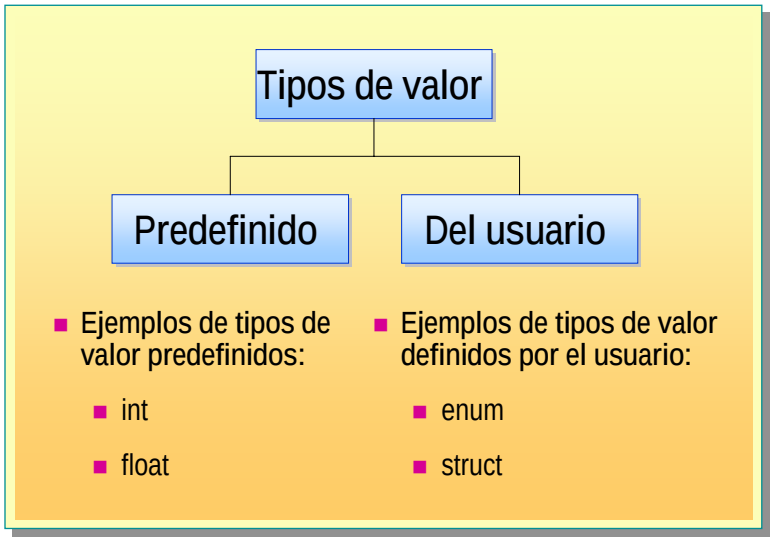
Las variables de tipos de referencia contienen referencias a sus datos. Los datos para estas variables se almacenan en un objeto. Es posible que dos variables de tipos de referencia apunten al mismo objeto, por lo que las operaciones efectuadas sobre una variable pueden afectar al objeto apuntado por otra variable de referencia.

Nota Todos los tipos de datos básicos están definidos en el espacio de nombres **System** y, en último término, se derivan de **System.Object**. Los tipos de valor se derivan de **System.ValueType**.

Comparación de tipos de valor predefinidos y definidos por el usuario

Objetivo del tema
Comparar tipos de valor predefinidos y definidos por el usuario.

Explicación previa
Los tipos de valor incluyen tipos predefinidos y definidos por el usuario. En este tema veremos algunos ejemplos de cada tipo.



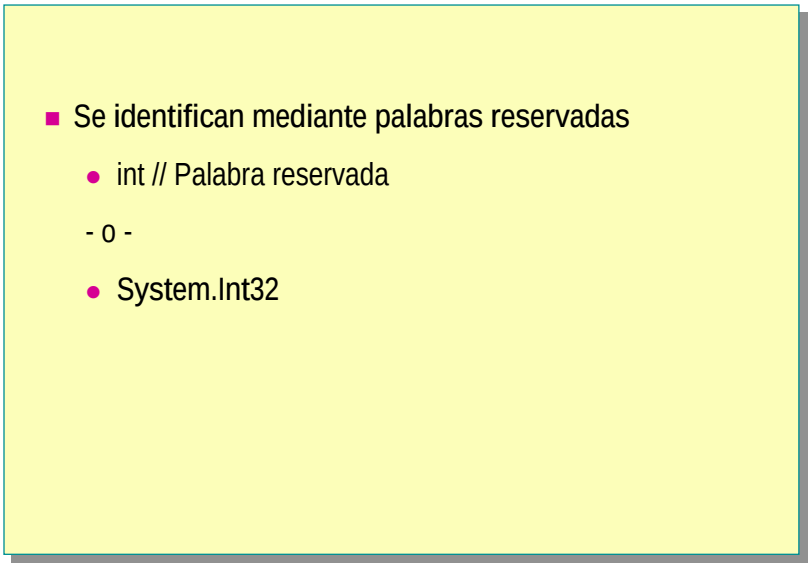
Los tipos de valor incluyen tipos de datos predefinidos y definidos por el usuario. La diferencia entre los dos tipos en C# es mínima, puesto que los tipos definidos por el usuario se pueden usar de la misma forma que los predefinidos. La única diferencia real entre tipos de datos predefinidos y tipos de datos definidos por el usuario es que para los primeros se pueden escribir valores literales. Todos los tipos de valor contienen datos directamente y no pueden ser **null**.

En este módulo aprenderá a crear tipos de datos definidos por el usuario, como tipos de enumeraciones y tipos de estructuras.

Tipos simples

Objetivo del tema
Describir los tipos simples.

Explicación previa
C# ofrece un conjunto de tipos de estructura predefinidos llamados tipos simples.



Los tipos de valor predefinidos también reciben el nombre de tipos de datos básicos o tipos simples. Los tipos simples se identifican por medio de palabras reservadas que son alias para tipos de estructura predefinidos.

Un tipo simple y el tipo de estructura correspondiente son *completamente indistinguibles*. En el código se puede utilizar la palabra reservada o el tipo de estructura. Los ejemplos que siguen ilustran ambas posibilidades:

```
byte // Palabra reservada
```

–O–

```
System.Byte // tipo de estructura
```

```
int // Palabra reservada
```

–O–

```
System.Int32 // tipo de estructura
```

Para más información sobre los tamaños e intervalos de tipos de valor predefinidos, busque “tipos de valor” en los documentos de ayuda de Microsoft® Visual Studio® .NET.

La siguiente tabla muestra las palabras reservadas y el tipo de estructura equivalente.

Palabras reservadas	Alias para tipo de estructura
sbyte	System.SByte
byte	System.Byte
short	System.Int16
ushort	System.UInt16
int	System.Int32
uint	System.UInt32
long	System.Int64
ulong	System.UInt64
char	System.Char
float	System.Single
double	System.Double
bool	System.Boolean
decimal	System.Decimal

◆ Nombres de variables

Objetivo del tema
Ofrecer una introducción a los temas tratados en esta sección.

Explicación previa
En esta sección aprenderá a asignar y utilizar nombres de variables significativos en una aplicación C#.

- Reglas y recomendaciones para nombrar variables
- Palabras clave de C#
- Problema: ¿Puede encontrar nombres de variables no permitidos?

Para usar una variable hay que elegir primero un nombre apropiado y significativo para ella. Cada variable tiene un nombre al que se conoce también por *identificador de variable*.

En el momento de nombrar variables se deben seguir las recomendaciones de nomenclatura estándar recomendadas para C#. También hay que conocer las palabras reservadas de C# que no se pueden utilizar para nombres de variables.

Al final de esta lección, usted será capaz de:

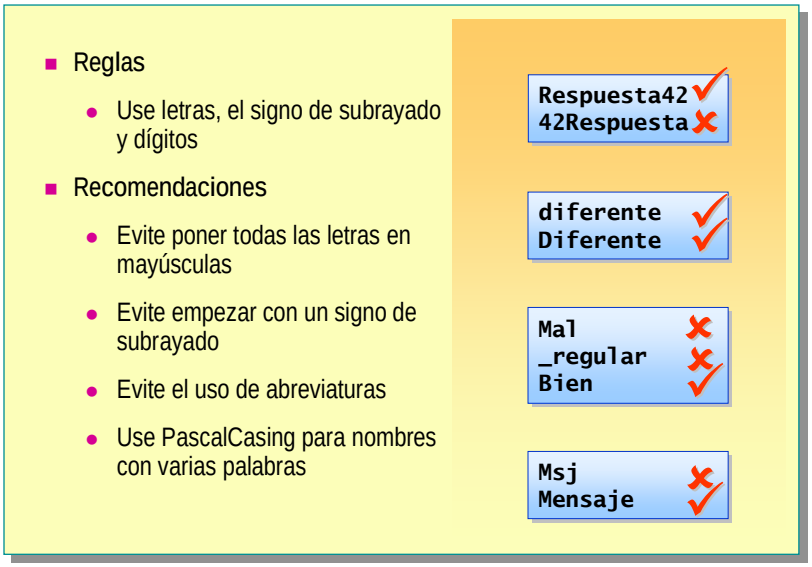
Identificar las palabras reservadas de C#.

Nombrar sus variables según las convenciones de nomenclatura estándar de C#.

Reglas y recomendaciones para nombrar variables

Objetivo del tema
Aprender las reglas para nombrar variables.

Explicación previa
Hay reglas y recomendaciones que se deben seguir al elegir el nombre de una variable.



Recomendación al profesor
Insista en que C# distingue mayúsculas y minúsculas, como C y C++.

Al nombrar variables hay que respetar las siguientes reglas y recomendaciones.

Reglas

Las siguientes son reglas de nomenclatura para variables de C#:

Comience todos los nombres de variables con una letra o un signo de subrayado.

Después del primer carácter, use letras, dígitos o el signo de subrayado.

No utilice palabras reservadas.

Si usa un nombre de variable no permitido, se producirá un error en tiempo de compilación.

Recomendación al profesor
Anime a los estudiantes a seguir estas recomendaciones. Explíqueles que el uso de convenciones de nomenclatura estándar les ayudará a desarrollar código coherente y de sencillo mantenimiento.

Recomendaciones

Es aconsejable seguir estas recomendaciones a la hora de nombrar variables:

Evite poner todas las letras en mayúsculas.

Evite empezar con un signo de subrayado.

Evite el uso de abreviaturas.

Use PascalCasing para nombres con varias palabras.

Convención de nomenclatura PascalCasing

La convención de nomenclatura PascalCasing consiste en poner en mayúscula el primer carácter de todas las palabras. Utilice PascalCasing para clases, métodos, propiedades, enumeraciones, interfaces, campos constantes y de sólo lectura, espacios de nombres y propiedades, como se muestra en el siguiente ejemplo:

```
void InicializarDatos( );
```

Convención de nomenclatura camelCasing

La convención de nomenclatura camelCasing consiste en poner en mayúscula el primer carácter de todas las palabras excepto la primera. Utilice camelCasing para variables que definan campos y parámetros, como se muestra en el siguiente ejemplo:

```
int bucleNumMax;
```

Para más información sobre convenciones de nomenclatura, busque “instrucciones de nomenclatura” en los documentos de ayuda del kit de desarrollo de software (SDK) de .NET.

Palabras clave de C#

Objetivo del tema
Dar una lista de algunas de las palabras reservadas más comunes.

Explicación previa
Las palabras clave son identificadores reservados para el lenguaje y no se pueden utilizar como nombres de variables.

- Las palabras clave son identificadores reservados

```
abstract, base, bool, default, if, finally
```

- No utilice palabras clave como nombres de variables
  - Produce errores en tiempo de compilación
- Procure no usar palabras clave cambiando mayúsculas y minúsculas

```
int INT;  // Mal estilo
```

Las palabras clave están reservadas, lo que significa que no se puede utilizar ninguna de ellas como nombre de variable en C#. El uso de una palabra clave como nombre de variable produce un error en tiempo de compilación.

Palabras clave en C#

A continuación se ofrece una lista de palabras clave en C#. Recuerde que ninguna de estas palabras se puede emplear como nombre de variable.

```
abstract    as          base        bool        break
byte        case        catch       char        checked
class       const       continue    decimal     default
delegate    do          double      else        enum
event       explicit    extern      false       finally
fixed       float       for         foreach     goto
if          implicit    in          int         interface
internal    is          lock        long        namespace
new         null        object      operator    out
override    params      private     protected   public
readonly    ref         return      sbyte       sealed
short       sizeof      stackalloc  static      string
struct      switch      this        throw       true
try         typeof      uint        ulong       unchecked
unsafe      ushort      using       virtual     void
volatile    while
```

Uso de tipos de datos predefinidos

Objetivo del tema
Ofrecer una introducción a los temas tratados en esta sección.

Explicación previa
Es importante comprender qué tipos de valor contienen datos directamente y no pueden contener valores null.

- Declaración de variables locales
- Asignación de valores a variables
- Asignación compuesta
- Operadores comunes
- Incremento y decremento
- Precedencia de operadores

Para crear una variable hay que elegir un nombre, declarar la variable y asignarle un valor, salvo si C# ya le ha asignado un valor automáticamente.

Al final de esta lección, usted será capaz de:

Crear una variable local utilizando tipos de datos predefinidos.

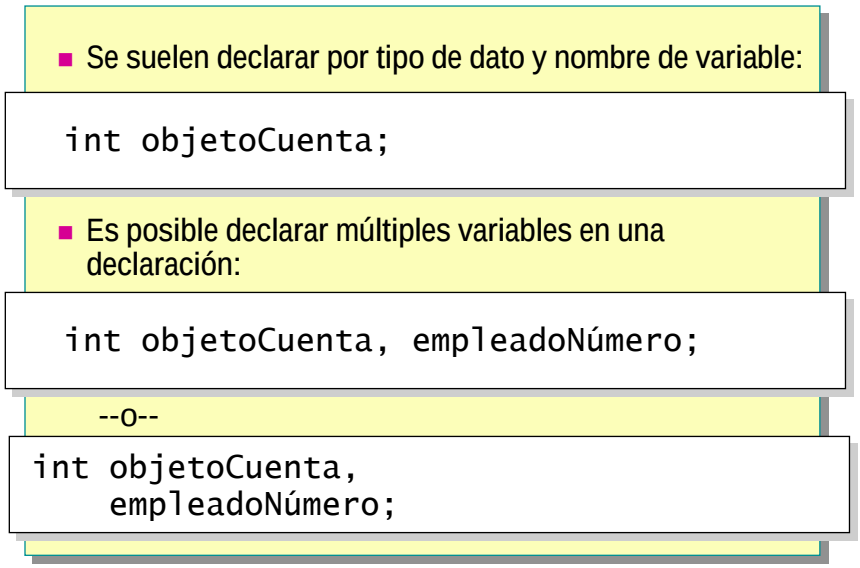
Usar operadores para asignar valores a variables.

Definir constantes y variables de sólo lectura.

Declaración de variables locales

Objetivo del tema
Explicar cómo declarar variables locales.

Explicación previa
En C# es muy importante saber usar variables locales.



Recomendación al profesor
Es posible que los estudiantes no estén familiarizados con métodos, propiedades e indizadores. Ésta es la primera vez en el curso en que aparecen estos términos. Explique que un método es como una función. Mencione que estos temas se tratarán en otros módulos del curso.

Las variables declaradas en métodos, propiedades o indizadores se llaman variables locales. Una variable local se declara normalmente especificando el tipo de dato seguido por el nombre de la variable, como se ve en el siguiente ejemplo:

```
int objetoCuenta;
```

Es posible declarar múltiples variables en una sola declaración separándolas con una coma, como en este ejemplo:

```
int objetoCuenta, empleadoNumero;
```

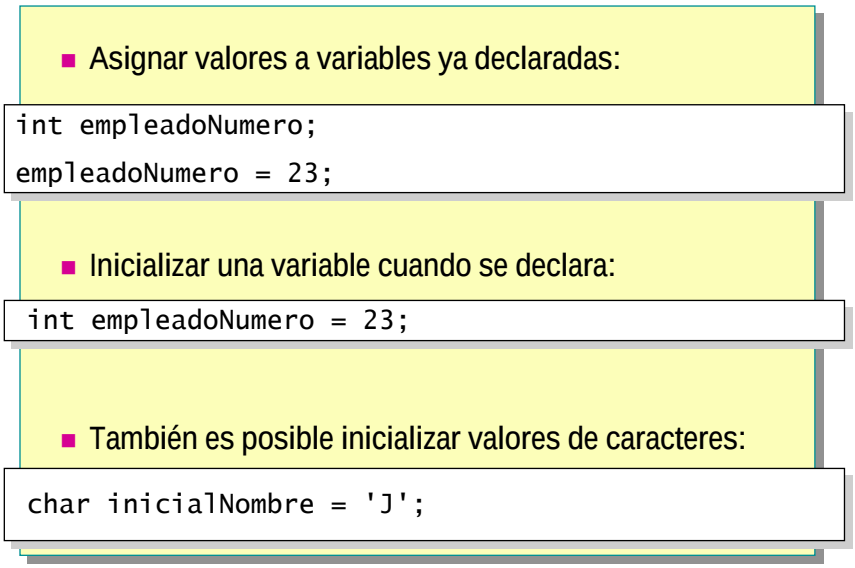
En C# no se pueden usar variables sin inicializar. Este código produciría un error en tiempo de compilación porque no se ha asignado un valor inicial a la variable *bucleCuenta*:

```
int bucleCuenta;
Console.WriteLine ("{0}", bucleCuenta);
```

Asignación de valores a variables

Objetivo del tema
Aprender a asignar valores a variables.

Explicación previa
Antes de poder usar variables sin inicializar hay que asignarles valores. En este tema veremos algunas instrucciones sencillas de asignación.



Para asignar un nuevo valor a una variable se utilizan operadores de asignación. En el caso de una variable ya declarada se usa el operador de asignación (=), como se ve en el siguiente ejemplo:

```
int empleadoNumero;
empleadoNumero = 23;
```

También es posible inicializar una variable al declararla, como en este ejemplo:

```
int empleadoNumero = 23;
```

Se puede usar el operador de asignación para asignar valores a variables de tipo carácter, como en este caso:

```
char inicialNombre = 'J';
```

Asignación compuesta

Objetivo del tema
Describir la unión de operadores para asignación compuesta.

Explicación previa
Muchas veces se necesita sumar o restar un valor a una variable en lugar de asignarle un valor totalmente nuevo.

- Es muy habitual sumar un valor a una variable

```
itemCount = itemCount + 40;
```

- Se puede usar una expresión más práctica

```
itemCount += 40;
```

- Esta abreviatura es válida para todos los operadores aritméticos:

```
itemCount -= 24;
```

Recomendación al profesor
Los programadores de Microsoft Visual Basic® no estarán familiarizados con la asignación compuesta.

Es muy habitual sumar u valor a una variable

El siguiente código declara una variable **int** llamada *objetoCuenta*, le asigna el valor 2 y posteriormente la incrementa en 40:

```
int objetoCuenta;
objetoCuenta = 2;
objetoCuenta = objetoCuenta + 40;
```

Se puede usar una expresión más práctica

Este código funciona para incrementar una variable, pero es un poco farragoso porque hay que escribir dos veces el identificador que se desea incrementar. Esto no suele ser un problema para identificadores simples, salvo que haya muchos identificadores con nombres similares. No obstante, es posible utilizar expresiones tan complejas como se quiera para designar el valor que se incrementa, como en este ejemplo:

```
items[(index + 1) % 32] = items[(index + 1) % 32] + 40;
```

En estos casos es muy fácil cometer algún pequeño error al escribir dos veces la expresión. Afortunadamente, existe una forma abreviada que evita la duplicación:

```
objetoCuenta += 40;
items[(index + 1) % 32] += 40;
```

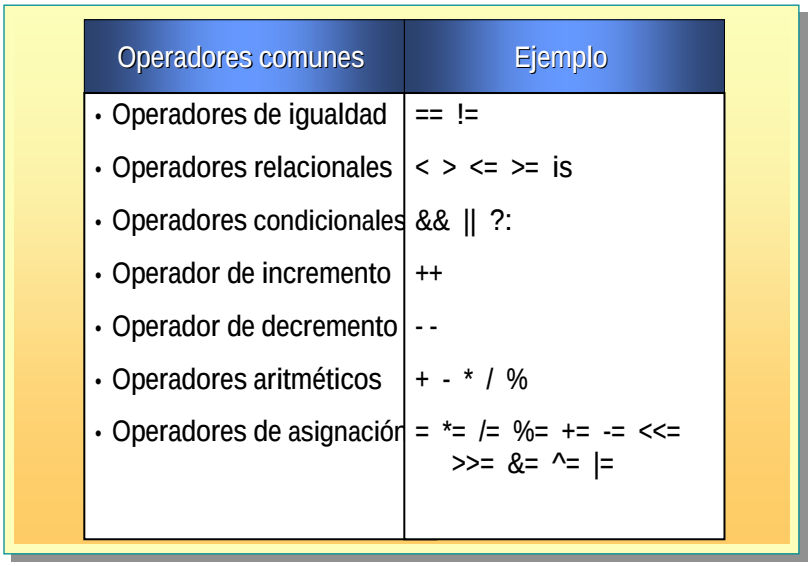
Esta abreviatura es válida para todos los operadores aritméticos

```
var += expresion; // var = var + expresión
var -= expresion; // var = var - expresión
var *= expresion; // var = var * expresión
var /= expresion; // var = var / expresión
var %= expresion; // var = var % expresión
```

Operadores comunes

Objetivo del tema
Describir operadores comunes que se pueden emplear en C#.

Explicación previa
Los operadores de asignación se usan para asignar valores a variables, descriptores de acceso a propiedades y descriptotes de acceso a índices. Aquí veremos algunos de los operadores comunes que se pueden usar para asignar un valor a una variable.



Las expresiones están formadas por *operandos* y *operadores*. Los operadores de una expresión indican las operaciones que se aplican a los operandos.

Algunos ejemplos de operadores son el operador de concatenación y suma (+), el operador de resta (-), el operador de multiplicación (*) y el operador de división (/). Literales, campos, variables locales y expresiones son ejemplos de operandos.

Operadores comunes

La siguiente tabla describe algunos de los operadores más comunes que se utilizan en C#.

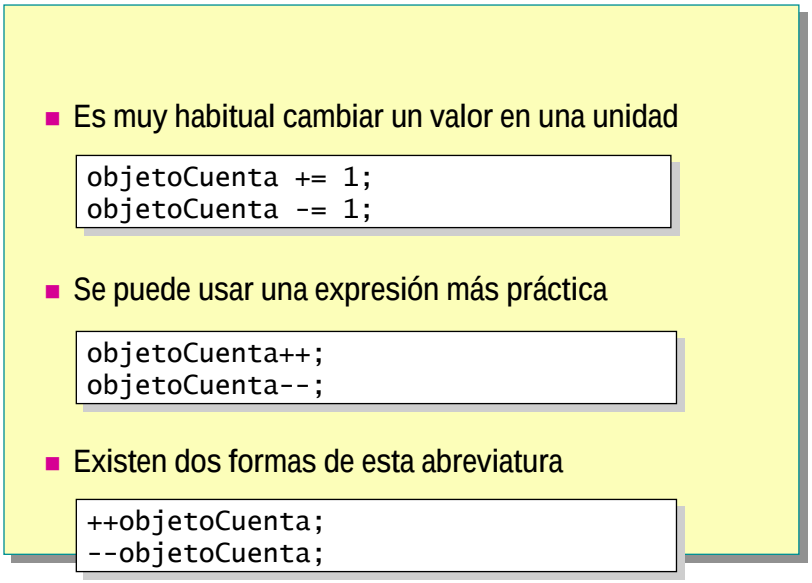
Tipo	Descripción
Operadores de asignación	Asignan valores a variables usando una asignación simple. Para ello es necesario que el valor en el lado derecho de la asignación sea de un tipo que se pueda convertir implícitamente al tipo de la variable en el lado izquierdo de la asignación.
Operadores lógicos relacionales	Comparan dos valores.
Operadores lógicos	Realizan operaciones bit a bit sobre valores.
Operador condicional	Selecciona entre dos expresiones, dependiendo de una expresión booleana.
Operador de incremento	Aumenta en uno el valor de una variable.
Operador de decremento	Reduce en uno el valor de una variable.
Operadores aritméticos	Realizan operaciones aritméticas estándar.

Para más información sobre los operadores de C#, busque “expresiones” en los documentos de ayuda de Visual Studio .NET sobre el lenguaje C#.

Incremento y decremento

Objetivo del tema
Describir los operadores de incremento y decremento y, en particular, explicar la diferencia entre las versiones de prefijo y sufijo.

Explicación previa
Muchas veces (cuando se cuenta algo, por ejemplo) se necesita sumar o restar uno a una variable.



Es muy habitual cambiar un valor en una unidad

A menudo hay que escribir instrucciones que incrementan o decrementan valores en una unidad. Se podría hacer así, por ejemplo:

```
objetoCuenta = objetoCuenta + 1;
objetoCuenta = objetoCuenta – 1;
```

Pero también se puede usar una expresión más práctica, como acabamos de ver:

```
objetoCuenta += 1;
objetoCuenta -= 1;
```

Expresión práctica

Los incrementos y decrementos unitarios de un valor son tan comunes que existe una expresión aún más abreviada:

```
objetoCuenta++; // objetoCuenta += 1;
objetoCuenta--; // objetoCuenta -= 1;
```

El operador ++ recibe el nombre de operador de incremento, mientras que -- es el operador de decremento. Se puede pensar en ++ como un operador que cambia un valor por su posterior y en -- como un operador que cambia un valor por su anterior.

La abreviatura es, también en este caso, la forma preferida por los programadores de C# para incrementar o decrementar un valor en una unidad.

Nota: ¡C++ se llamó C++ porque era el sucesor de C!

Existen dos formas de esta abreviatura

Los operadores ++ y -- se pueden usar de dos maneras.

1. Se puede colocar el símbolo del operador *antes* del identificador, como en los siguientes ejemplos. A esto se le llama notación de *prefijo*.

   ```
   ++objetoCuenta;
   --objetoCuenta;
   ```

2. Se puede colocar el símbolo del operador *después* del identificador, como en los siguientes ejemplos. A esto se le llama notación de *sufijo*.

   ```
   objetoCuenta++;
   objetoCuenta--;
   ```

En ambos casos se incrementa (para ++) o decrementa (para --) *objetoCuenta* en una unidad. Pero entonces, ¿por qué hay dos notaciones? Para responder a esta pregunta es necesario saber más sobre la asignación:

Una característica importante de C# es que la asignación es un operador. Esto quiere decir que, además de asignar un valor a una variable, una expresión de asignación tiene en sí misma un valor, o resultado, que es el valor de la variable después de que se realice la asignación. En la mayor parte de las instrucciones se descarta el valor de la expresión de asignación, pero es posible usarlo en una expresión mayor, como en el siguiente ejemplo:

```
int objetoCuenta = 0;
Console.WriteLine(objetoCuenta = 2); // Imprime 2
Console.WriteLine(objetoCuenta = objetoCuenta + 40); // Imprime 42
```

Una asignación compuesta también es una asignación. Esto significa que una expresión de asignación compuesta, además de asignar un valor a una variable, tiene también un valor o resultado. También en este caso se descarta el valor de la expresión de asignación compuesta en la mayor parte de las instrucciones, pero es posible usarlo en una expresión mayor, como en el siguiente ejemplo:

```
int objetoCuenta = 0;
Console.WriteLine(objetoCuenta += 2); // Imprime 2
Console.WriteLine(objetoCuenta -= 2); // Imprime 0
```

El incremento y el decremento también son asignaciones. Esto significa, por ejemplo, que una expresión de incremento, además de incrementar una variable en uno, tiene también un valor o resultado. También en este caso se descarta el valor de la expresión de incremento en la mayor parte de las instrucciones, pero es posible usarlo en una expresión mayor, como en el siguiente ejemplo:

```
int objetoCuenta = 42;
int prefijoValor = ++objetoCuenta;  // prefijoValor == 43
int sufijoValor = objetoCuenta++; // sufijoValor = 43
```

El valor de la expresión de incremento varía dependiendo de que se use la versión de prefijo o de sufijo. *objetoCuenta* se incrementa en los dos casos. La diferencia no está ahí, sino en el valor de la expresión de incremento. El valor de la expresión de incremento/decremento como prefijo es el valor de la variable *después* de realizar el incremento/decremento, mientras que el valor de un incremento/decremento como sufijo es el valor de la variable *antes* del incremento/decremento.

Precedencia de operadores

Objetivo del tema
Explicar la precedencia de operadores.

Explicación previa
Para entender cómo funcionan los programas en C#, es preciso conocer el orden en que C# evalúa los operadores empleados.

- Precedencia y asociatividad de operadores
  - Todos los operadores binarios, salvo los de asignación, son asociativos por la izquierda
  - Los operadores de asignación y el operador condicional son asociativos por la derecha

---

Recomendación al profesor
Puede ser conveniente repasar los distintos tipos de operadores.

Operadores monarios. Los operadores monarios se aplican a un operando y usan notación de prefijo (como --x) o de sufijo (como x++).

Operadores binarios. Los operadores monarios se aplican a dos operandos y usan notación de prefijo (como x + y).

El operador ternario. Existe sólo un operador ternario (?:). El operador ternario se aplica a tres operandos y usan notación de infijo (como c? x: y).

Precedencia de operadores

Si una expresión contiene varios operadores, la precedencia de estos controla el orden en que se evalúa cada uno de ellos. Por ejemplo, la expresión x + y * z se evalúa como x + (y * z) porque el operador de multiplicación tiene una precedencia mayor que el de suma. Una *expresión aditiva*, por ejemplo, consta de una secuencia de *expresiones multiplicativas* separadas por operadores + o -, por lo que los operadores + y – tienen una precedencia menor que los operadores *, /, y %.

Asociatividad

Si un operando está entre dos operadores con la misma precedencia, la *asociatividad* de los operadores controla el orden en que se efectúan las operaciones. Por ejemplo, x + y + z se evalúa como (x + y) + z. Esto tiene especial importancia para operadores de asignación. Por ejemplo, x = y = z se evalúa como x = (y = z).

Todos los operadores binarios, salvo los de asignación, son *asociativos por la izquierda*, lo que quiere decir que las operaciones se realizan de izquierda a derecha.

Los operadores de asignación y el operador condicional (?:)son *asociativos por la derecha*, lo que quiere decir que las operaciones se realizan de derecha a izquierda.

Es posible controlar la precedencia y la asociatividad utilizando paréntesis. Por ejemplo, x + y * z multiplica primero y por z y luego suma el resultado a x, pero (x + y) * z suma primero x más y, y luego multiplica el resultado por z.

Creación de tipos de datos definidos por el usuario

Objetivo del tema
Ofrecer una introducción a los temas tratados en esta sección.

Explicación previa
En esta sección aprenderá a crear tipos de datos enum y struct.

- Enumeraciones
- Estructuras

En las aplicaciones es necesario saber cómo crear tipos de datos de **enumeración** (**enum**) y **estructura** (**struct**) definidos por el usuario.

Al final de esta lección, usted será capaz de:

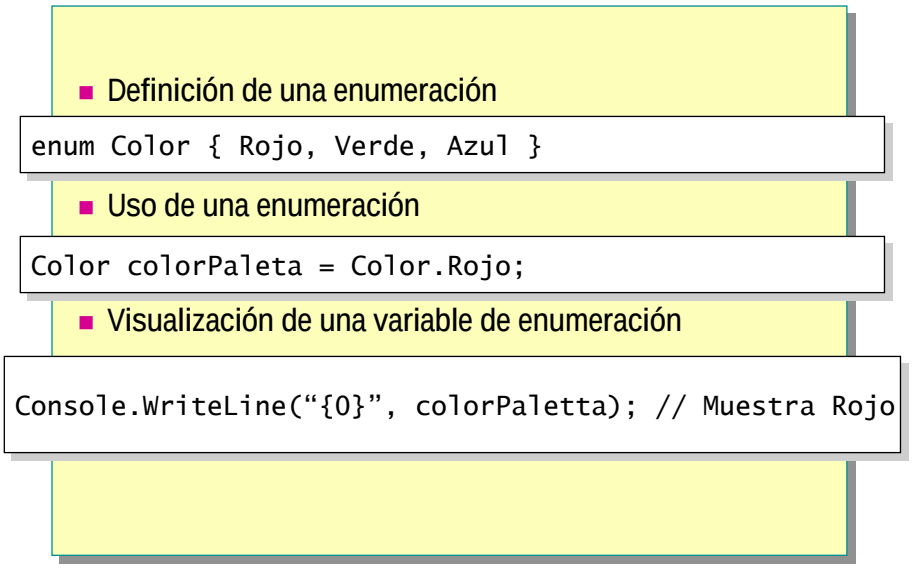
Crear tipos de datos **enum** definidos por el usuario.

Crear tipos de datos **struct** definidos por el usuario.

Enumeraciones

Objetivo del tema
Explicar cómo se crea una variable del tipo enum en C#.

Explicación previa
Una enumeración crea un conjunto único de valores constantes relacionados.



Los enumeradores son útiles cuando una variable sólo puede tener un conjunto de valores concreto.

Definición de enumeraciones

Para declarar una enumeración se usa la palabra clave **enum** seguida de un nombre de variable enum y los valores iniciales. Por ejemplo, esta enumeración define tres constantes enteras, llamadas valores del enumerador.

```
enum Color { Rojo, Verde, Azul }
```

Por defecto, los valores de enumeradores empiezan en 0. En el ejemplo anterior, *Rojo* tiene el valor 0, *Verde* tiene el valor 1 y *Azul* tiene el valor 2.

Es posible inicializar una enumeración especificando literales enteros.

Uso de una enumeración

Se puede declarar una variable *colorPaleta* de tipo **Color** empleando la siguiente sintaxis:

```
Color colorPaleta;        // Declara la variable
colorPaleta = Color.Rojo; // Asigna el valor
```

- O -

```
colorPaleta = (Color)0;  // Tipo int a Color
```

Visualización de un valor de enumeración

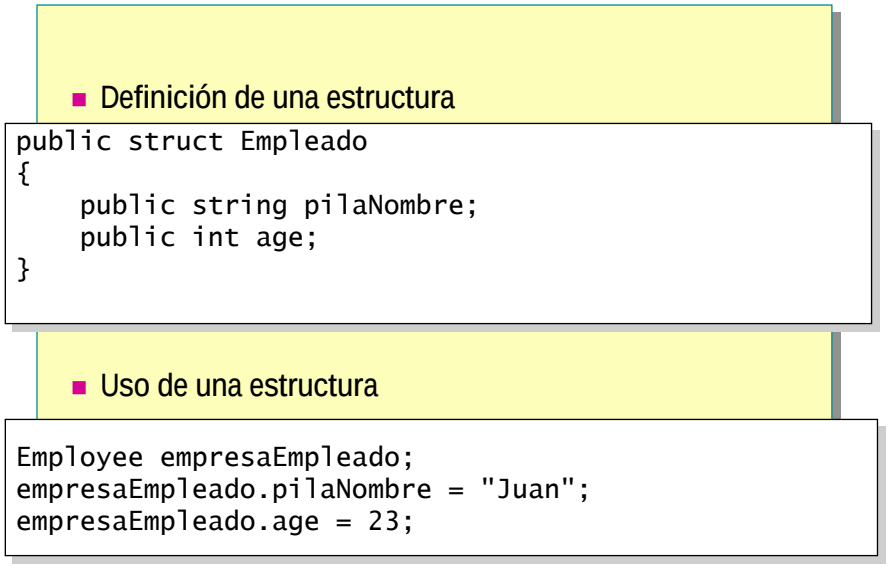
Para ver un valor de enumeración en un formato legible se puede usar la siguiente instrucción:

```
Console.WriteLine("{0}", colorPaleta);
```

Estructuras

Objetivo del tema
Explicar cómo se crean estructuras definidas por el usuario (structs) en C#.

Explicación previa
Una estructura es una agrupación de tipos arbitrarios.



Las estructuras se pueden emplear para crear objetos que se comportan como tipos de valor predefinidos. Las estructuras se almacenan en línea y no se asignan por montones, por lo que la necesidad de eliminar elementos no utilizados en el sistema es menor que en el caso de las clases.

Todos los tipos de datos simples, como **int**, **float** y **double**, son estructuras predefinidas en .NET Framework.

Definición de una estructura

Recomendación al profesor
Nota: Las estructuras no permiten la herencia.

Presente un ejemplo sencillo de estructura, como una estructura de Punto. Algunos ejemplos de estructura son punto, rectángulo y número complejo.

Se puede utilizar una estructura para agrupar varios tipos arbitrarios, como muestra el siguiente ejemplo:

```
public struct Empleado
{
    public string pilaNombre;
    public int   edad;
}
```

Este código define un nuevo tipo, llamado **Empleado**, que consta de dos elementos: el nombre de pila y la edad.

Uso de una estructura

Para acceder a los elementos de una estructura se puede emplear la siguiente sintaxis:

```
Empleado empresaEmpleado;              // Declara la variable
empresaEmpleado.pilaNombre =  "Juan"; // Asigna el valor
empresaEmpleado.edad = 23;
```

Conversión de tipos de datos

Objetivo del tema
Ofrecer una introducción a los temas tratados en esta sección.

Explicación previa
En C# hay dos tipos de conversión: implícita y explícita.

- Conversión implícita de tipos de datos
- Conversión explícita de tipos de datos

En C# hay dos tipos de conversión:

Conversión implícita de tipos de datos.

Conversión explícita de tipos de datos.

Al final de esta lección, usted será capaz de:

Realizar una conversión implícita de datos.

Realizar una conversión explícita de datos.

Conversión implícita de tipos de datos

Objetivo del tema
Explicar como hacer una conversión implícita de tipos de datos.

Explicación previa
La conversión implícita no puede fallar para tipos de datos predefinidos.

- Conversión de int a long

```
using System;
class Test
{
   static void Main( )
   {
       int intValor = 123;
       long longValor = intValor;
       Console.WriteLine("(long) {0} = {1}", intValor,
   ➥longValor);
   }
}
```

- Las conversiones implícitas no pueden fallar
  - Se puede perder precisión, pero no magnitud

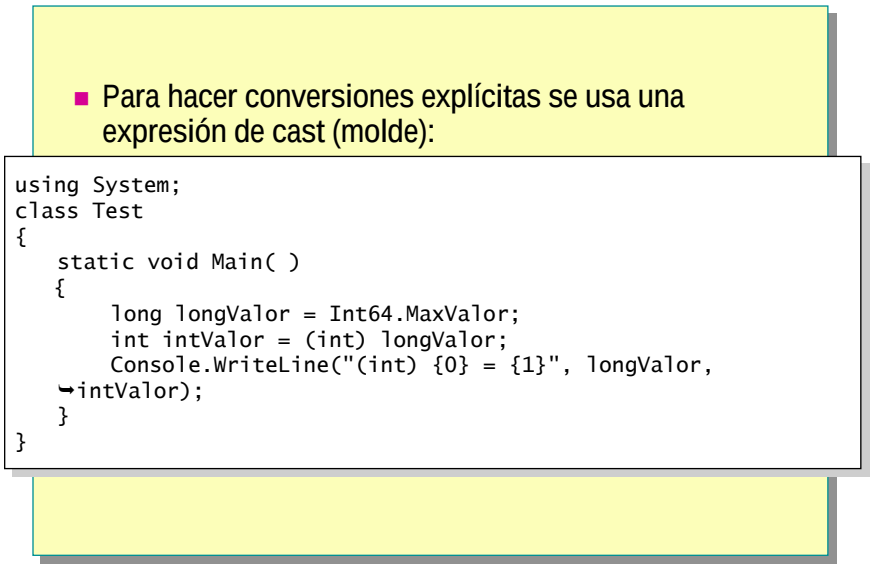
La conversión de un tipo de datos **int** a otro **long** es implícita. Esta conversión siempre tiene éxito y jamás produce una pérdida de información. El siguiente ejemplo muestra cómo convertir la variable *intValor* de un **int** a un **long**:

```
using System;
class Test
{
    static void Main( )
    {
        int intValor = 123;
        long longValor = intValor;
        Console.WriteLine("(long) {0} = {1}", intValor,
➥longValor);
    }
}
```

Conversión explícita de tipos de datos

Objetivo del tema
Explicar como hacer una conversión implícita de tipos de datos.

Explicación previa
Para hacer una conversión explícita de datos se emplea una expresión de cast (molde).



Para hacer una conversión explícita de tipos de variables se puede emplear una expresión de cast (molde). El siguiente ejemplo muestra cómo convertir la variable *longValor* de un tipo de datos **long** a otro **int** utilizando una expresión de cast:

```
using System;
class Test
{
    static void Main( )
    {
        long longValor = Int64.MaxValor;
        int intValor = (int) longValor;
        Console.WriteLine("(int) {0} = {1}", longValor,
➥intValor);
    }
}
```

En este ejemplo se produce un desbordamiento, por lo que el resultado es el siguiente:

```
(int) 9223372036854775807 = -1
```

Para que esto no ocurra se puede usar la instrucción **checked**, que lanza una excepción en caso de fallo en una conversión:

```
using System;
class Test
{
  static void Main( )
  {
     checked
     {
        long longValor = Int64.MaxValor;
        int intValor = (int) longValor;
        Console.WriteLine("(int) {0} = {1}", longValor,
➥intValor);
     }
  }
}
```