



Clase presencial



Sesion3: Tipos de datos en Visual Studio

Instructor: David Paúl Porras Córdova

@iscodem



Objetivo General

- ❑ Identificar los diferentes tipos de datos existentes en Visual Studio .NET.
- ❑ Diferenciar en el uso de diferentes formas de declarar tipos de datos.
- ❑ Determinar la importancias de elegir correctamente un tipo de dato.
- ❑ Mejores prácticas para declarar variables.



Tipos de datos

- En la tabla siguiente se muestran los tipos de datos de Visual Basic .NET, los tipos compatibles con **Common Language Runtime**, su asignación de almacenamiento nominal y sus intervalos de valores.

Short Name	.NET Class	Type	Width	Range (bits)
byte	Byte	Unsigned integer	8	0 to 255
sbyte	SByte	Signed integer	8	-128 to 127
int	Int32	Signed integer	32	-2,147,483,648 to 2,147,483,647
uint	UInt32	Unsigned integer	32	0 to 4294967295
short	Int16	Signed integer	16	-32,768 to 32,767
ushort	UInt16	Unsigned integer	16	0 to 65535
long	Int64	Signed integer	64	-9223372036854775808 to 9223372036854775807
ulong	UInt64	Unsigned integer	64	0 to 18446744073709551615
float	Single	Single-precision floating point type	32	-3.402823e38 to 3.402823e38
double	Double	Double-precision floating point type	64	-1.79769313486232e308 to 1.79769313486232e308
char	Char	A single Unicode character	16	Unicode symbols used in text
bool	Boolean	Logical Boolean type	8	True or false
object	Object	Base type of all other types		
string	String	A sequence of characters		
decimal	Decimal	Precise fractional or integral type that can represent decimal numbers with 29 significant digits	128	$\pm 1.0 \times 10e-28$ to $\pm 7.9 \times 10e28$



Contenido de Agenda

- Variables.
 - ▣ Tipos de Variables
 - ▣ Ámbito de una variable
 - Variables globales
 - Variables locales
- Identificadores
- Buenas prácticas de programación
 - ▣ Identación.
 - ▣ Comentarios.
 - ▣ Inicializar.



Variables y Constantes

- Variables
- Constantes
- Identificadores
- Comentarios
- Indentación

1. Identificadores (IDs)

Son símbolos léxicos que nombran entidades. El concepto es análogo al de "nombres de procesamiento de la información". Nombrar las entidades hace posible referirse a las mismas, lo cual es esencial para cualquier tipo de procesamiento simbólico.

2. Variables

Una variable es un nombre simbólico que se refiere a un dato determinado almacenado en memoria. Las nombraremos de acuerdo a las siguientes pautas:

- **Siempre comenzarán por letra.**
- **Serán menores de 255 caracteres.**
- **No se permiten espacios (blancos), puntos (.), ni caracteres especiales.**
- **No pueden utilizarse palabras reservadas del lenguaje**

2.1. Ámbito de una variable

- El ámbito de un elemento declarado es el conjunto de todo el código que puede hacer referencia sin calificar su nombre o ponerlo a disposición a través de un **Imports** (instrucción Namespace de .NET y tipo). Un elemento puede tener ámbito en uno de los siguientes niveles:..

2.1. Ámbito de una variable

Nivel	Descripción
Ámbito de bloque	Disponible solo en el código del bloque en el que se declara
Ámbito de procedimiento	Disponible para todo el código dentro del procedimiento en el que se declara
Ámbito de módulo	Disponible para todo el código dentro del módulo, clase o estructura en la que se ha declarado
Ámbito de Namespace	Disponible para todo el código en el espacio de nombres en el que se declara

2.2. Especificar el ámbito de una variable

Especifique el ámbito de un elemento cuando se declara. El ámbito puede depender de los siguientes factores:

- ❑ La región (bloque, procedimiento, módulo, clase o estructura) en el que se declara el elemento.
- ❑ El espacio de nombres que contiene la declaración del elemento.
- ❑ El nivel de acceso que se declara para el elemento.

Tenga cuidado al definir las variables con el mismo nombre pero con un ámbito diferente, porque si lo hace, puede provocar resultados inesperados.

2.2.1. Ejemplos de declaración de una variable

Ejemplos

```
// Declaración una variable entera x de tipo int
```

```
int x;
```

```
// Declaración de una variable real r de tipo double
```

```
double r;
```

```
// Declaración de una variable c de tipo char
```

```
char c;
```

```
// Múltiples declaraciones en una sola línea
```

```
int i, j, k;
```

2.3. Niveles de ámbito

Un elemento de programación está disponible en toda la región en la que se declara. Todo el código en la misma región puede hacer referencia al elemento sin calificar su nombre.

- Ámbito de bloque.
- Ámbito de procedimiento.
- Ámbito de módulo.
- Ámbito de Namespace

1.3. Niveles de ámbito

Do y Loop

For [Each] y Next

If y End If

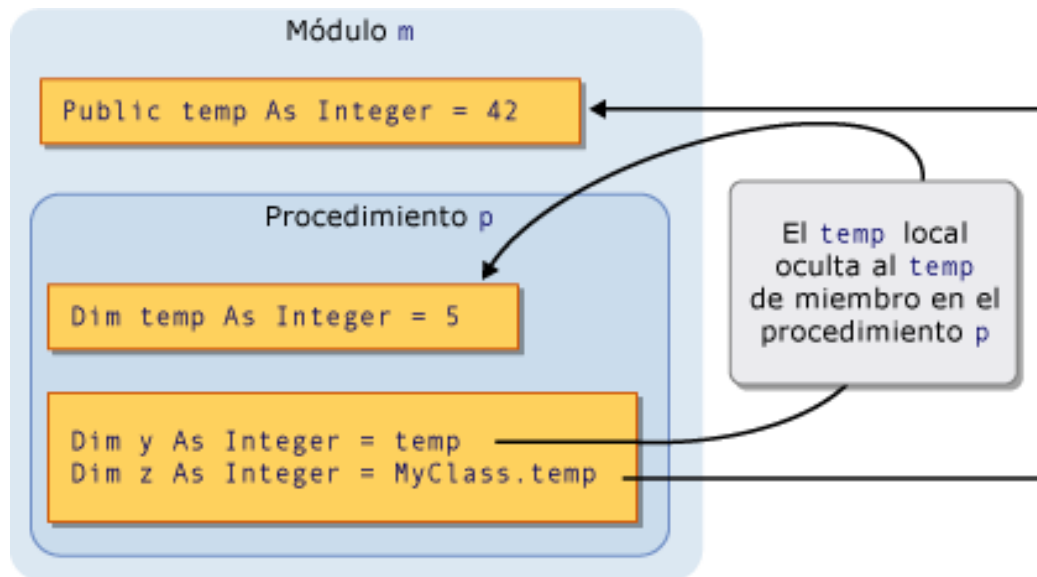
Select y End Select

SyncLock y End SyncLock

Try y End Try

While y End While

With y End With



2.4. Ventajas de las Variables locales

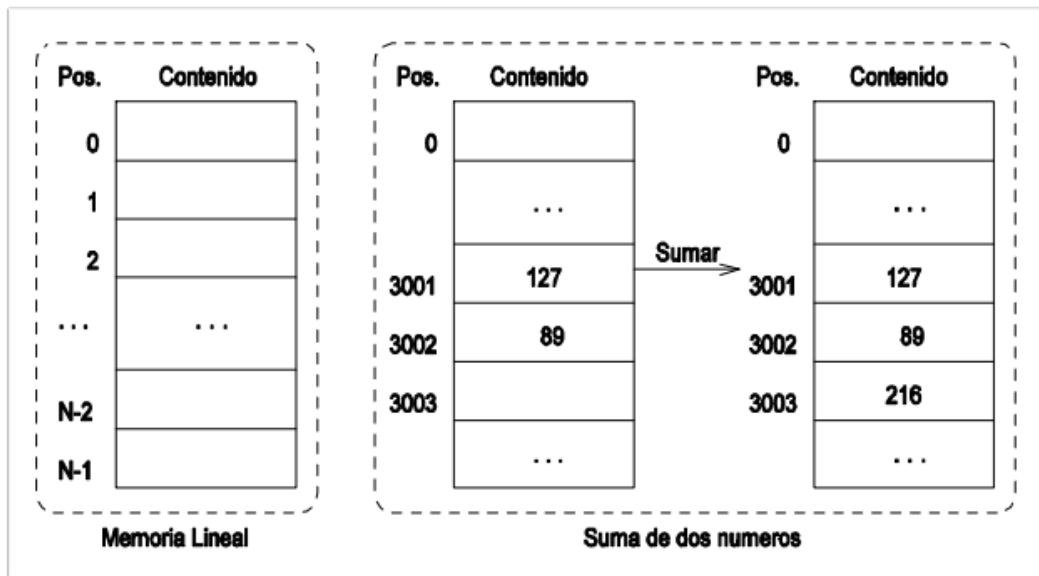
Las variables locales son una buena elección para cualquier tipo de cálculo temporal, por las razones siguientes:

- ❑ **Evitar conflictos de nombres.**
- ❑ **Consumo de memoria.**

Laboratorio: 3.1. Declaración de variables

- Ejercicio 1:
 - ▣ Tiempo: 10 minutos
 - ▣ Realizar un programa que utiliza el mismo nombre de variable en diferente ámbito e identificar las implicancias.

Cómo se comportan las variables



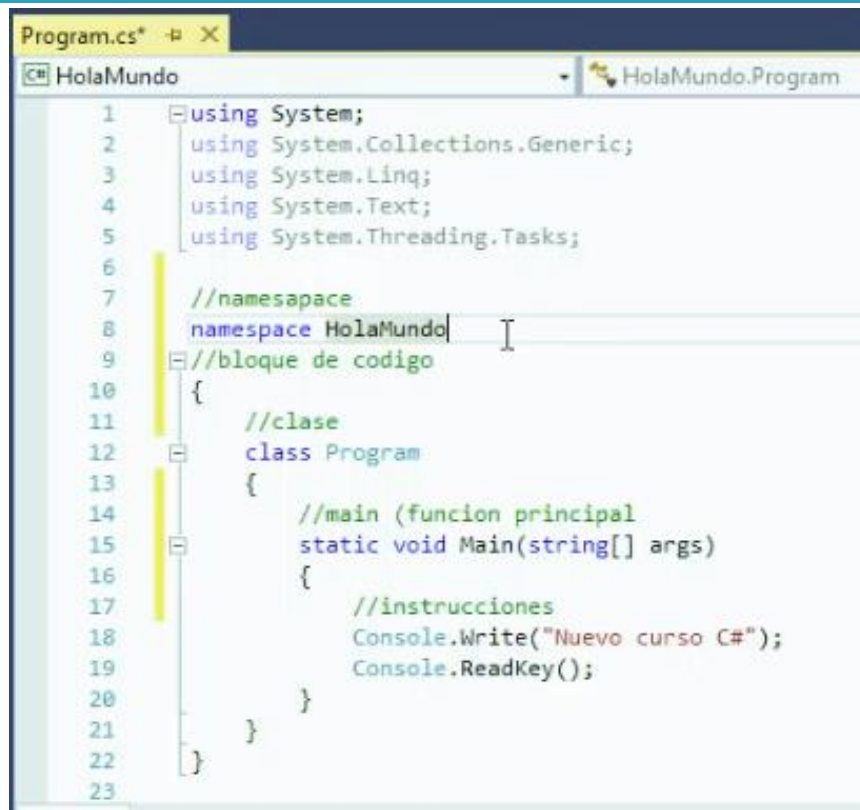
“Suma el contenido de la posición 3001 y la 3002 y lo almacenas en la posición 3003”

vs.

$\text{total} = \text{cantidad1} + \text{cantidad2}$

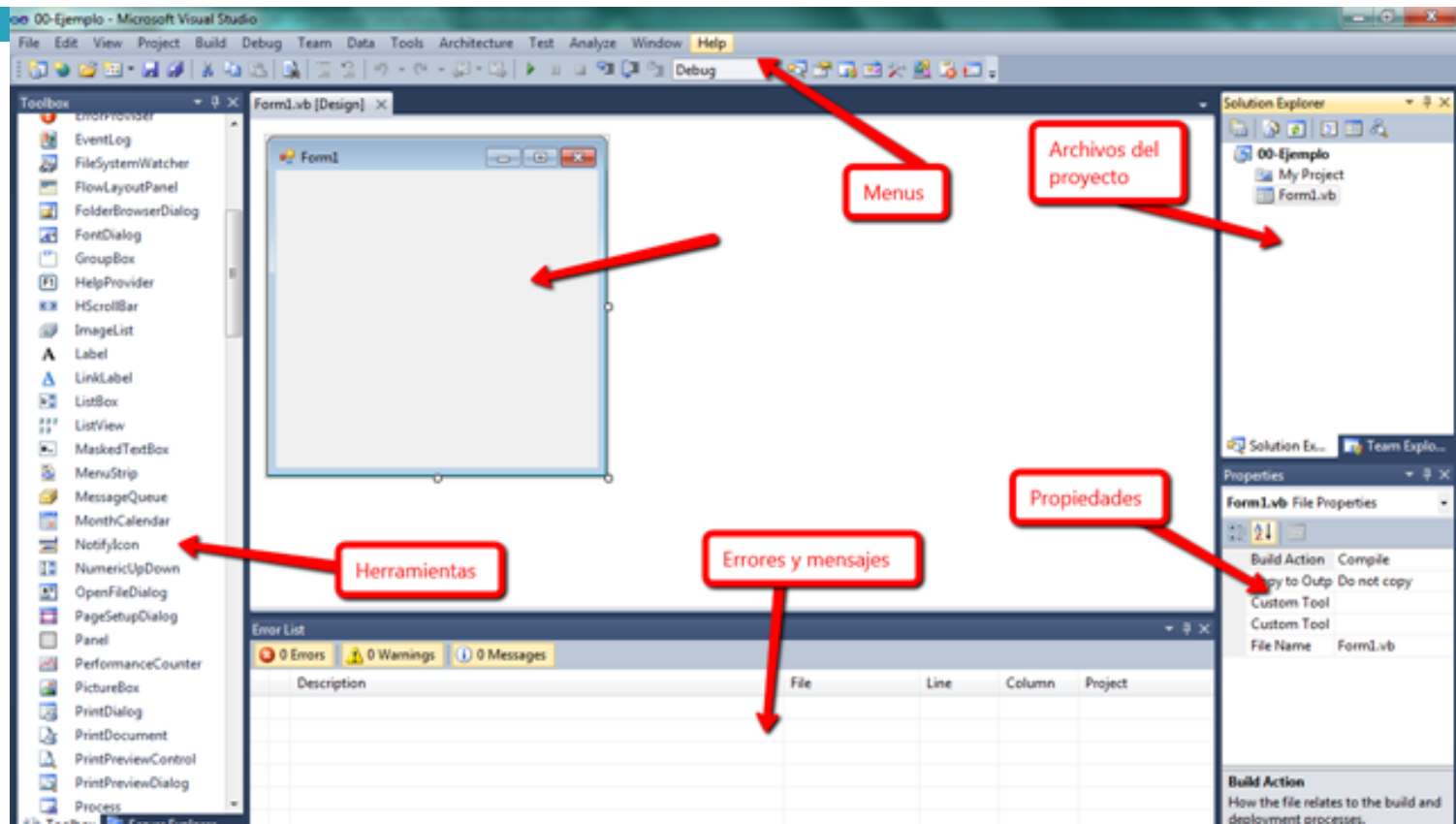
“Suma cantidad1 y cantidad2 y lo almacenas en total”

3. Partes de un programa



```
Program.cs* [X]
[+] HolaMundo [ - ] HolaMundo.Program
1  [ - ] using System;
2      using System.Collections.Generic;
3      using System.Linq;
4      using System.Text;
5      using System.Threading.Tasks;
6
7      //namespace
8      namespace HolaMundo
9      [ - ] //bloque de codigo
10     {
11         //clase
12         [ - ] class Program
13         {
14             //main (funcion principal
15             [ - ] static void Main(string[] args)
16             {
17                 //instrucciones
18                 Console.WriteLine("Nuevo curso C#");
19                 Console.ReadKey();
20             }
21         }
22     }
23
```

3. Partes de un programa



3. Buenas prácticas de programación

- ❑ Escribe tus programas lo más simple y directo posible.
- ❑ Todo programa debe ser previamente comentado, explicando el propósito, funcionamiento completo y el resultado esperado.
- ❑ Dentro de las funciones definidas, establece un espaciado o **indentación**, que resalte la estructura funcional de la aplicación y facilite la lectura al programador al que le corresponda analizar el código.
- ❑ Se recomienda declarar variables en líneas separadas, ya que se facilita la descripción de cada variable mediante comentarios.

3. Buenas prácticas de programación

- ❑ Poner un espacio después de cada coma(,) facilita la legibilidad del código.
- ❑ Se recomienda en algunas operaciones complejas, hacer uso de paréntesis redundantes o innecesarios que sirven para poder agrupar expresiones dentro de tales operaciones.
- ❑ Si el código soporta la separación de sentencias en varias líneas, procura realizar una separación coherente, en el que cada punto de ruptura tenga sentido.
- ❑ **Nunca** olvides inicializar los contadores y sumadores.