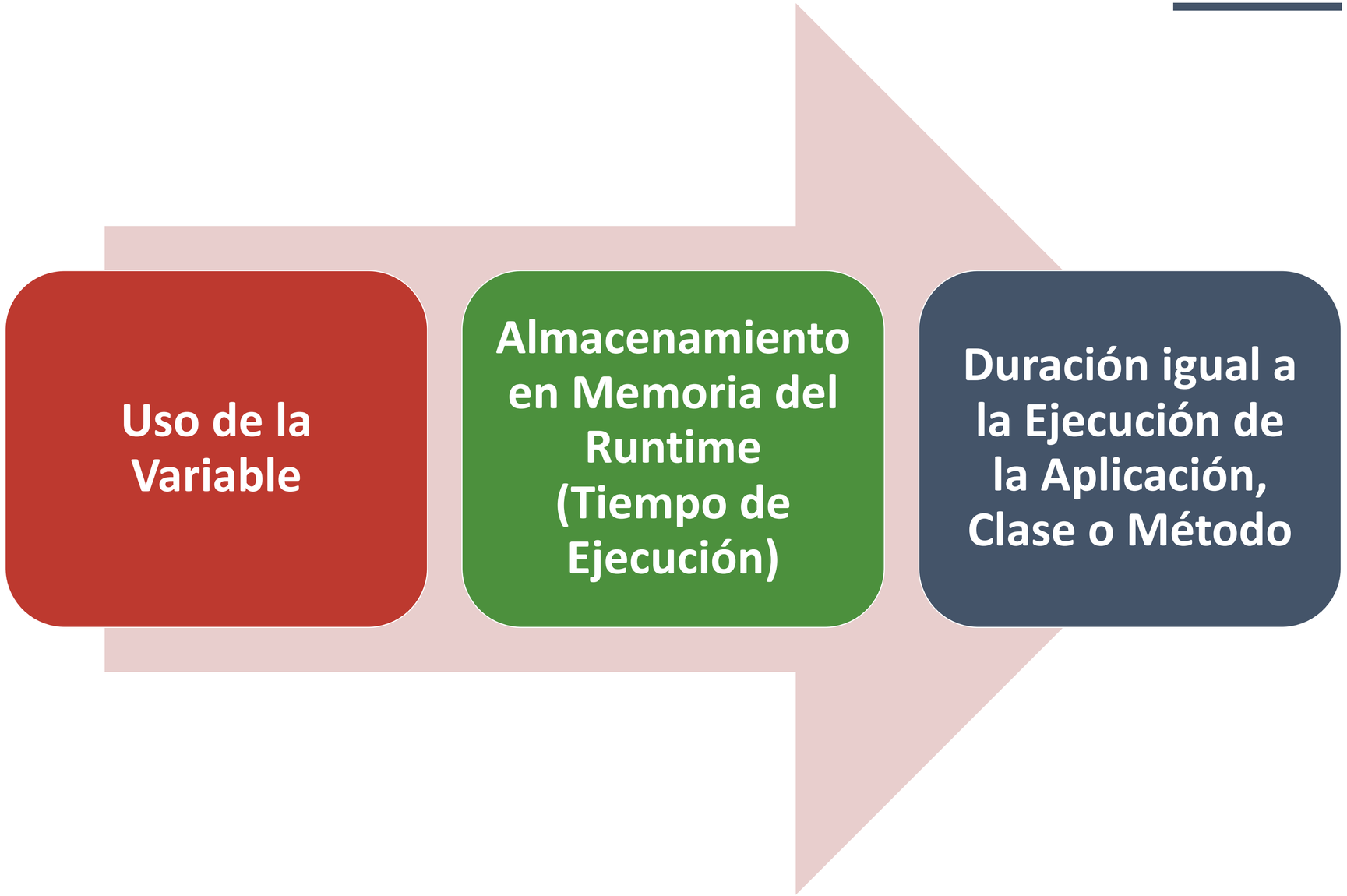
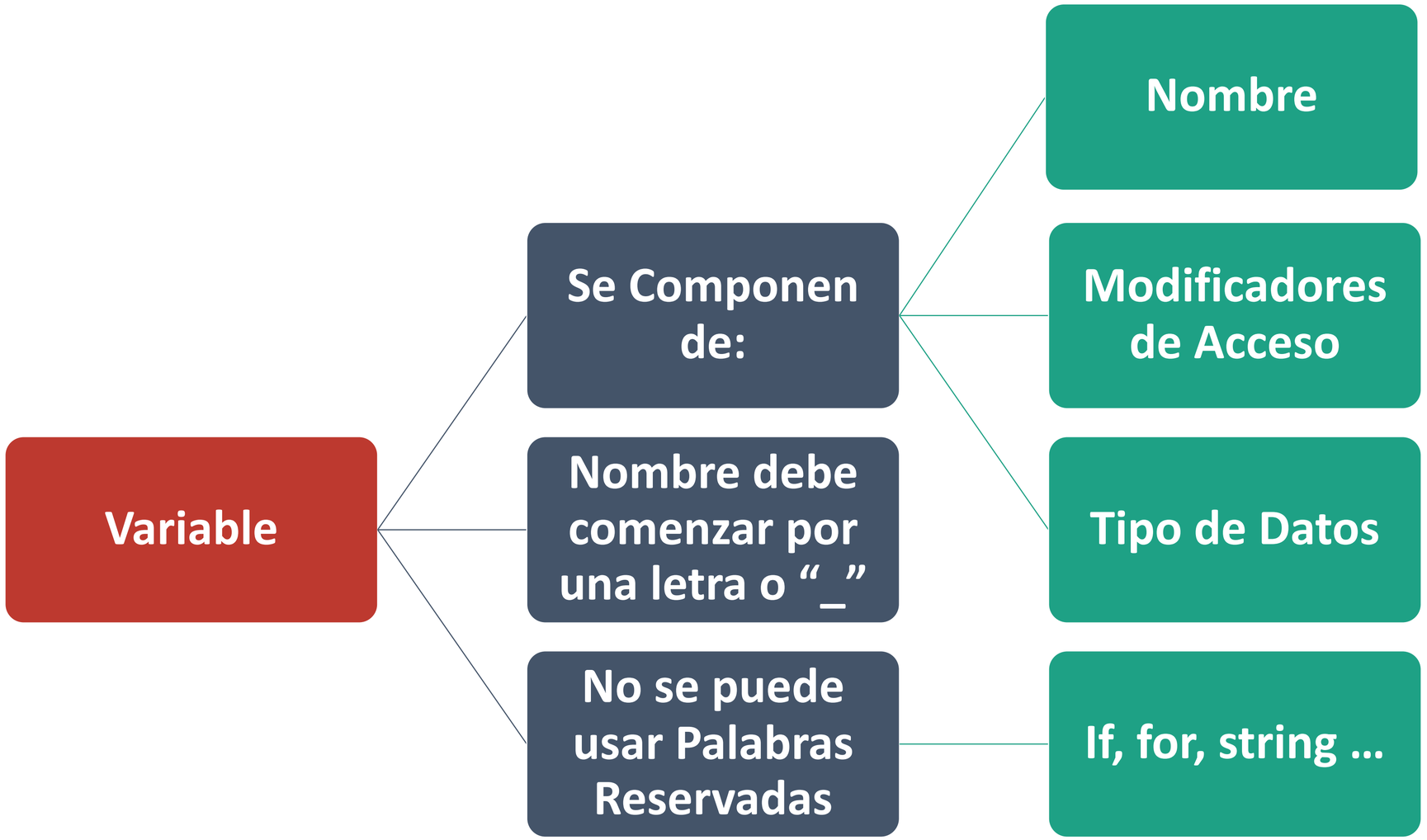
Variables


Uso de la Variable
Almacenamiento en Memoria del Runtime (Tiempo de Ejecución)
Duración igual a la Ejecución de la Aplicación, Clase o Método

Variables


Variable
Se Componen de:
Nombre debe comenzar por una letra o “_”
No se puede usar Palabras Reservadas
Nombre
Modificadores de Acceso
Tipo de Datos
If, for, string …

Variables - Composición

Modificador de Acceso

```
private   int numero;
```

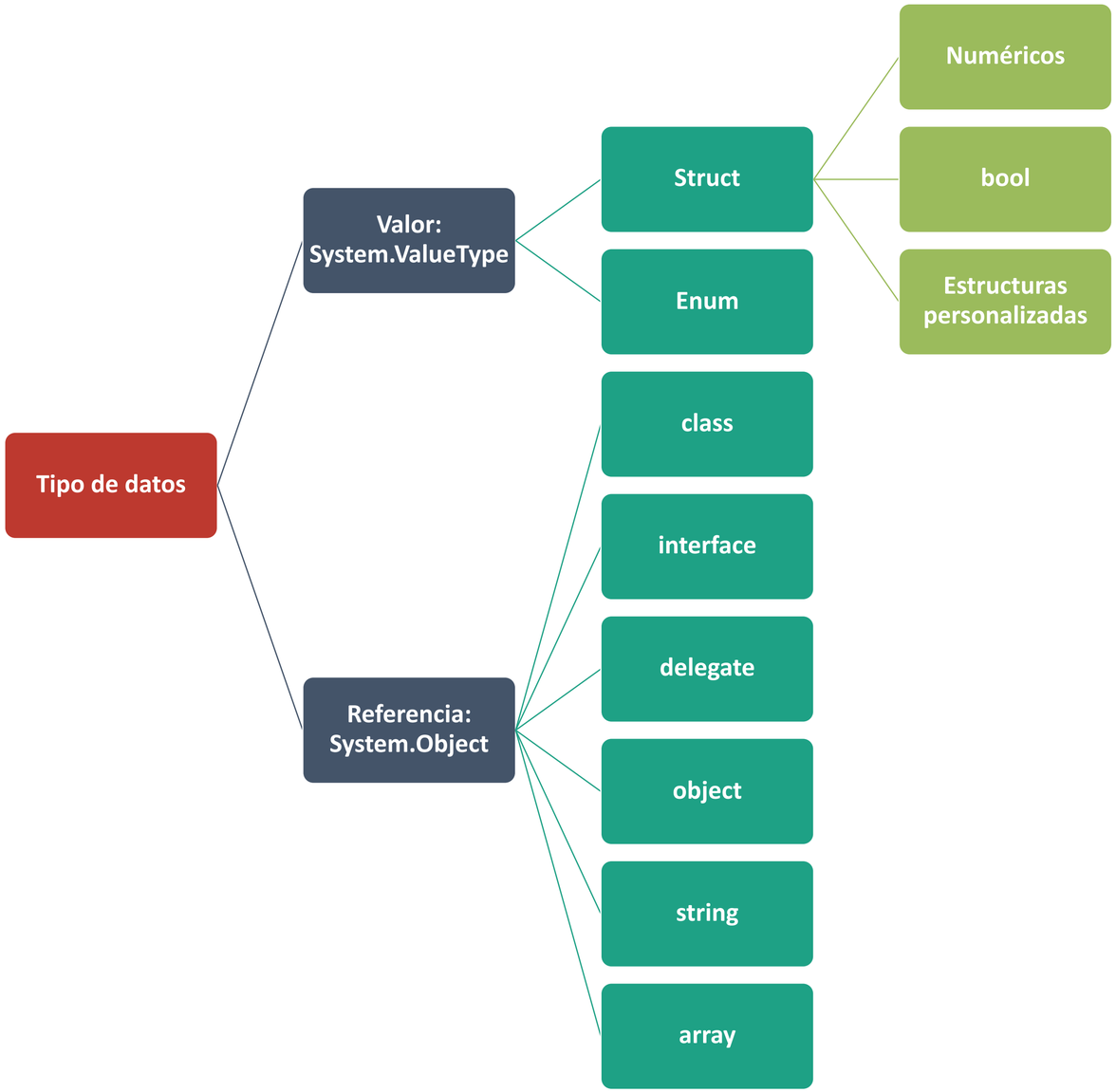
Tipo de Dato: entero

Nombre: numero

Variables – Modificadores de Acceso

Modificado	Función
public	Sin acceso restringido
protected	Acceso limitado a las clases o tipos derivados de la clase en la que está la variable
Internal	El acceso se limita al código de un mismo ensamblado pero no tendrá acceso en un código ensamblado diferente.
protected internal	El acceso está limitado al conjunto actual o a los tipos derivados de la clase contenedora.
private	El acceso está limitado a la clase en la que está la variable

Variables – Tipos de Datos



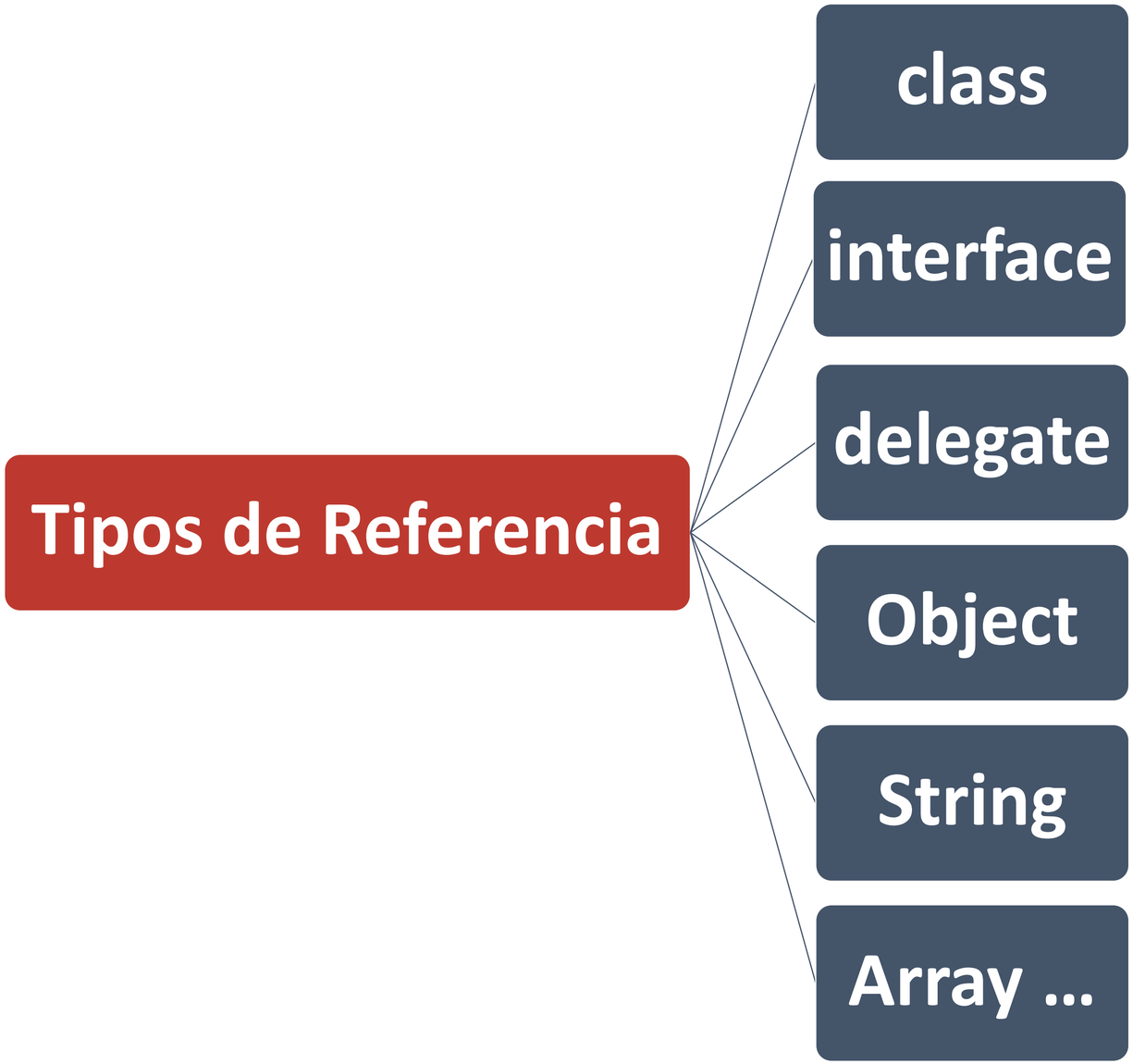
Variables – Tipos Numéricos

Tipo	Representa	Rango	Default Value
bool	Valor booleano	True o False	False
byte	Entero sin signo de 8-bit	0 to 255	0
char	Carácter Unicode 16-bit	U +0000 to U +ffff	'\0'
decimal	Valores decimales de precisión de 128 bits con 28-29 dígitos significativos	$(-7.9 \times 10^{28}$ to $7.9 \times 10^{28}) / 10^{0 \text{ to } 28}$	0.0M
double	Tipo de coma flotante de doble precisión de 64 bits	$(+/-)5.0 \times 10^{-324}$ to $(+/-)1.7 \times 10^{308}$	0.0D
float	Tipo de coma flotante de precisión simple de 32 bits	-3.4×10^{38} to $+ 3.4 \times 10^{38}$	0.0F
int	Tipo entero con signo de 32 bits	-2,147,483,648 to 2,147,483,647	0
long	Tipo entero con signo de 64 bits	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	0L
sbyte	Tipo entero con signo de 8 bits	-128 to 127	0
short	Tipo entero con signo de 16 bits	-32,768 to 32,767	0
uint	Tipo entero sin signo de 32 bits	0 to 4,294,967,295	0
ulong	Tipo entero sin signo de 64 bits	0 to 18,446,744,073,709,551,615	0
ushort	Tipo entero sin signo de 16 bits	0 to 65,535	0

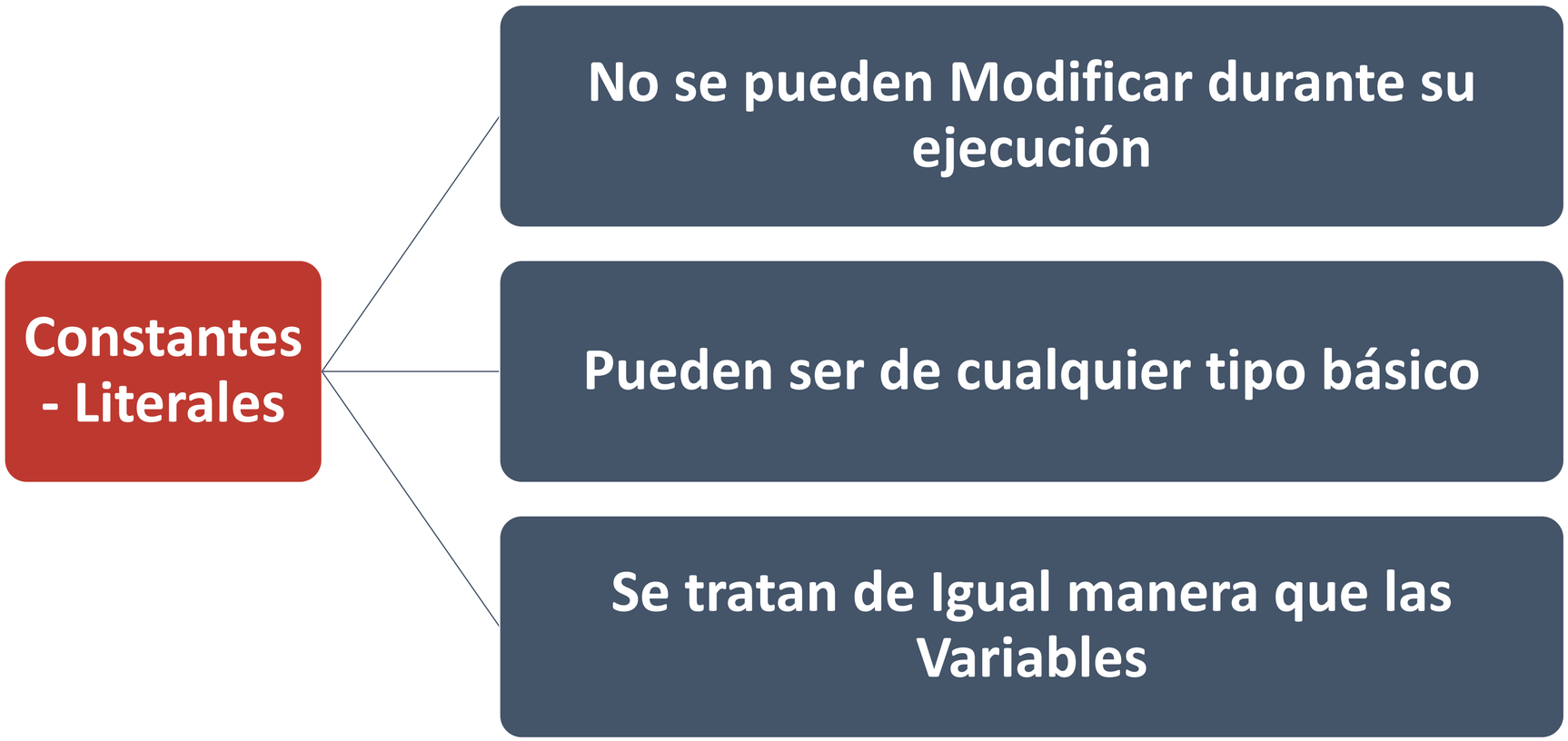
Variables – Enum

```
class pedido
        {
                public int numero;
                public DateTime FechaHora;
                public int situacion;
        }

Enum situacion

        {
                Abierto,
                Facturado,
                Cancelado
        }
```

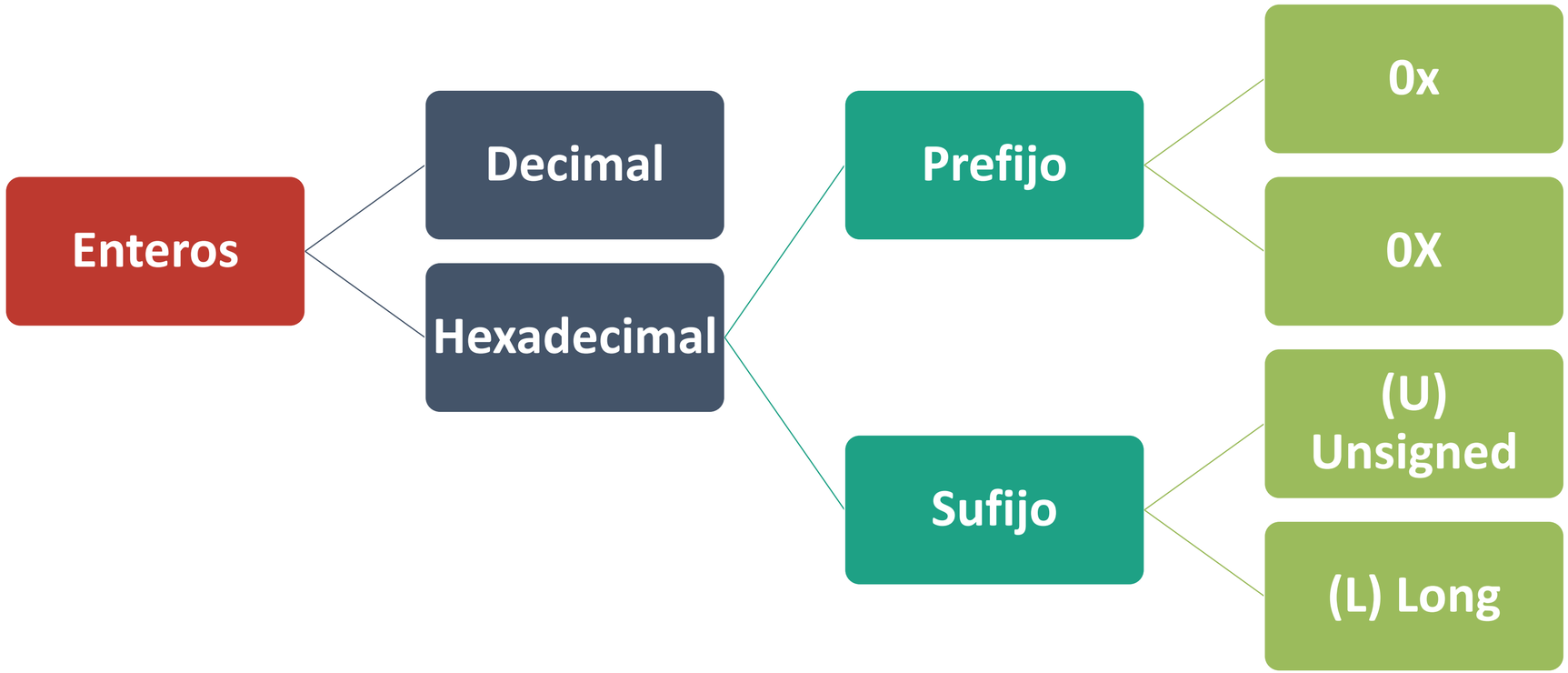
Variables – Tipos de Referencia



Constantes - Literales



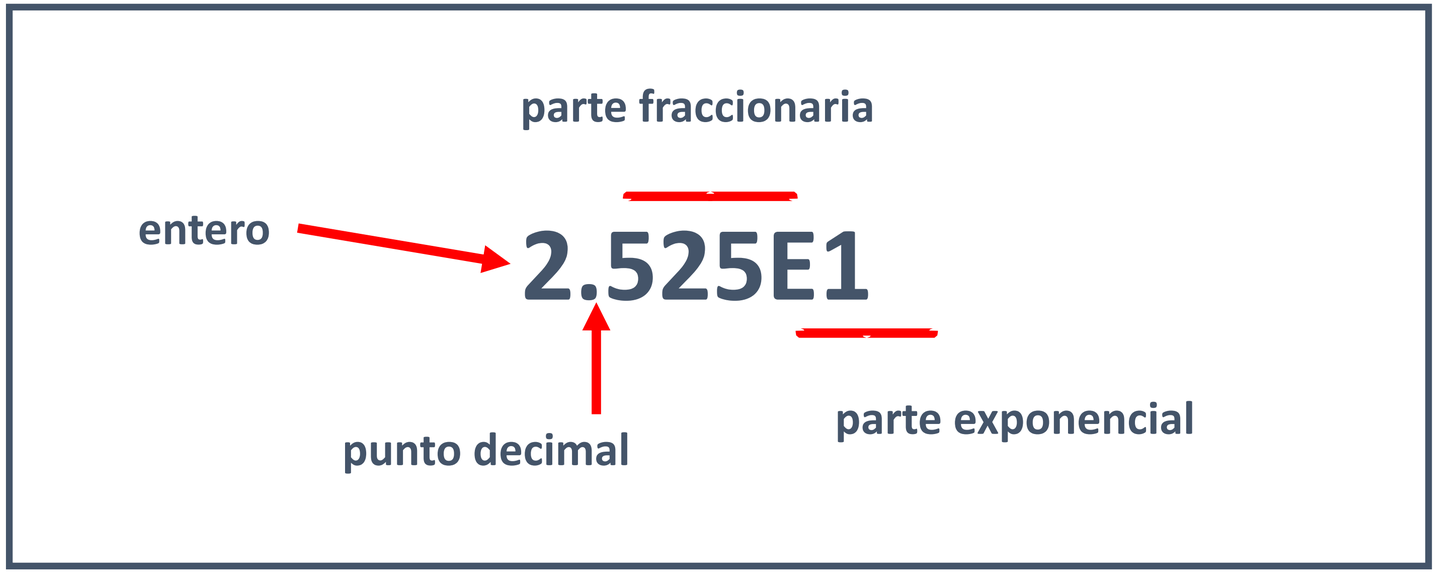
Constantes Enteros



Ejemplos

```
85      /* decimal */
0x4b    /* hexadecimal */
30      /* int */
30u     /* unsigned int */
30l     /* long */
30ul    /* unsigned long */
```

Constantes Punto Flotante



Ejemplos

```
25.25
2.525E1   /* = 25.25   */
2525e-2   /* = 25.25   */
-3.5e-3   /* = -0.0035 */
35E-4     /* = 0.0035 */
```

Ejemplos

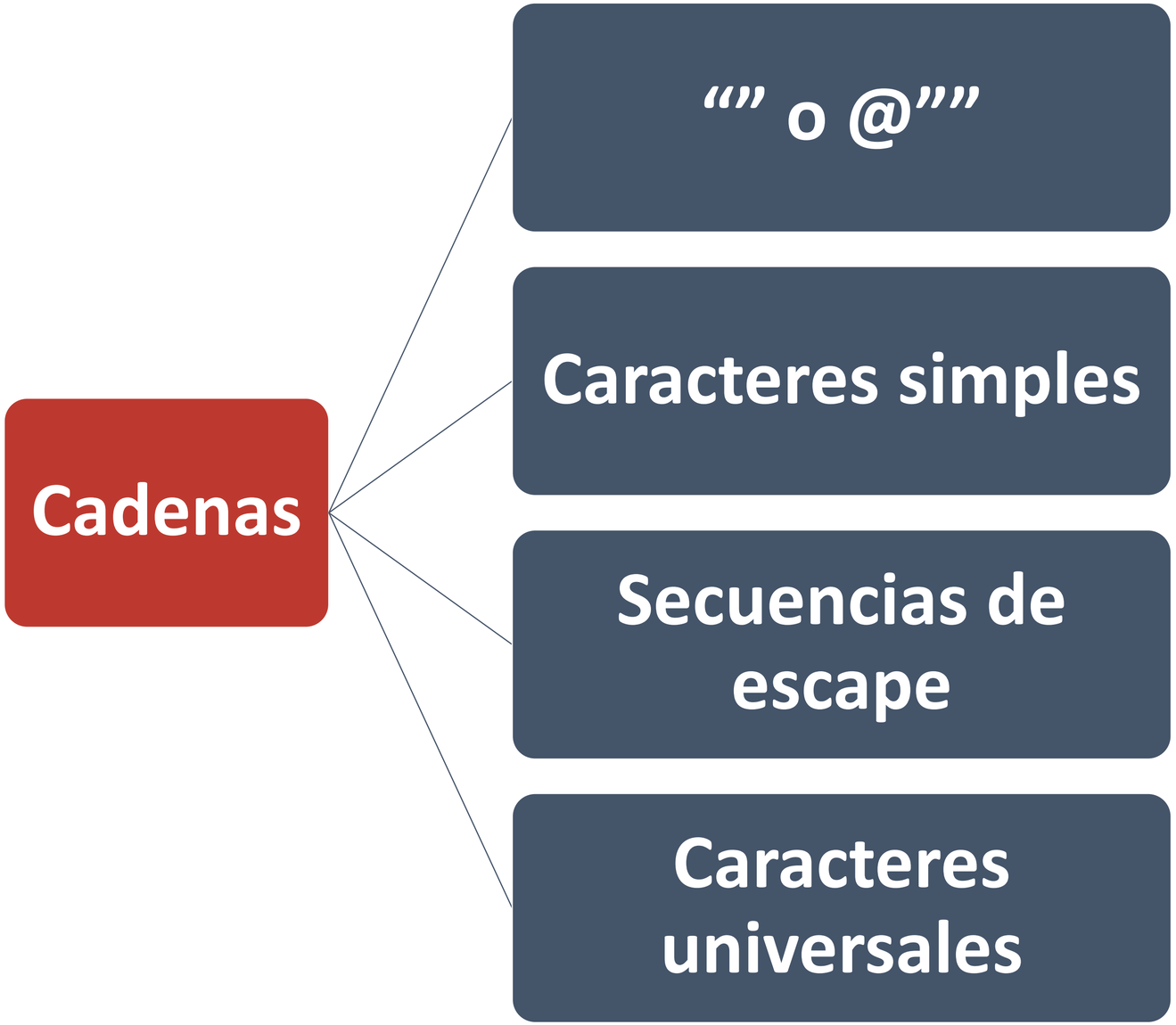
```
100    /* Tiene tipo double
100L   /* Tiene tipo long double  */
100F   /* Tiene tipo float        */
```

Constantes de Caracteres

Secuencia de Escape	Significa
\a	Campanilla (aviso)
\b	Retroceso
\f	Formfeed
\n	Nueva línea
\r	Retorno de carro
\t	Tabulación horizontal
\v	Tabulación vertical
\'	Comilla simple
\ "	Comilla doble
\\	Barra diagonal inversa
\?	signo de interrogación literal
ooo de \	Carácter ASCII en notación octal
hh de \ x	Carácter ASCII en notación hexadecimal
hhhh de \ x	Carácter Unicode en notación hexadecimal

‘x’	char
‘\t’	Tabulación horizontal
‘\u02C0’	Carácter universal

Constantes de Cadena



Ejemplo

“hola, que tal?“

“hola, \
que tal?“

“hola, " “que" “tal?“

@“hola, que tal?"

Definir una Constante

const <tipo_dato> <nombre_constante> = valor;

Flujos de Ejecución – Parte I

if … else

```
if (expresion booleana)
        {
                // código 1
        }
else
        {
                // código 2
        }
```

Flujos de Ejecución – Parte I

```
if (expresión booleana 1)
{
   // código 1
}
else if (expresión booleana 2)
{
   // código 2
}
else
{
   // código 3
}
```

if … else if … else

Flujos de Ejecución – Parte I

Operador Ternario

expresión booleana? código 1: código 2;

Flujos de Ejecución – Parte II

Switch … Case

```
switch (variable o valor)
{

case valor1:
    // código 1
break;

case valor2:
    // código 2
break;

}
```

Flujos de Ejecución – Parte II

```
switch (variable o valor)
{
        case valor1:
        case valor2:
        case valor3:
                // código 1
                break;

        case valor4:
        case valor5:
        case valor6:
                // código 2
                break;
}
```

Switch … Case

Flujos de Ejecución – Parte II

```
switch (variable o valor)
{

case valor1:
        // código 1
break;

case valor2:
        // código 2
break;

default:
        // código 3
        break;
}
```

Switch … Case

Operador *Default*

Estructuras de Repetición– Parte I

While

```
while (condición)
{
        //bloque de código
}
```

Estructuras de Repetición– Parte I

do … while

```
do
{

  //bloque de código

} while (condición);
```

Estructuras de Repetición– Parte I

Break

```
int i = 1;

while (i < 10)
{
if(i == 5)
break;

Console.WriteLine(i);
i++;
}
```

Estructuras de Repetición– Parte I

Continue

```
int i = 0;
while (i < 10)
    {
     i++;
     if (i % 2 == 0)
     continue;

Console.WriteLine(i);
    }
Console.ReadKey();
```

Estructuras de Repetición– Parte II

For

for (*inicializador***;** *condición***;** *iterador***)**

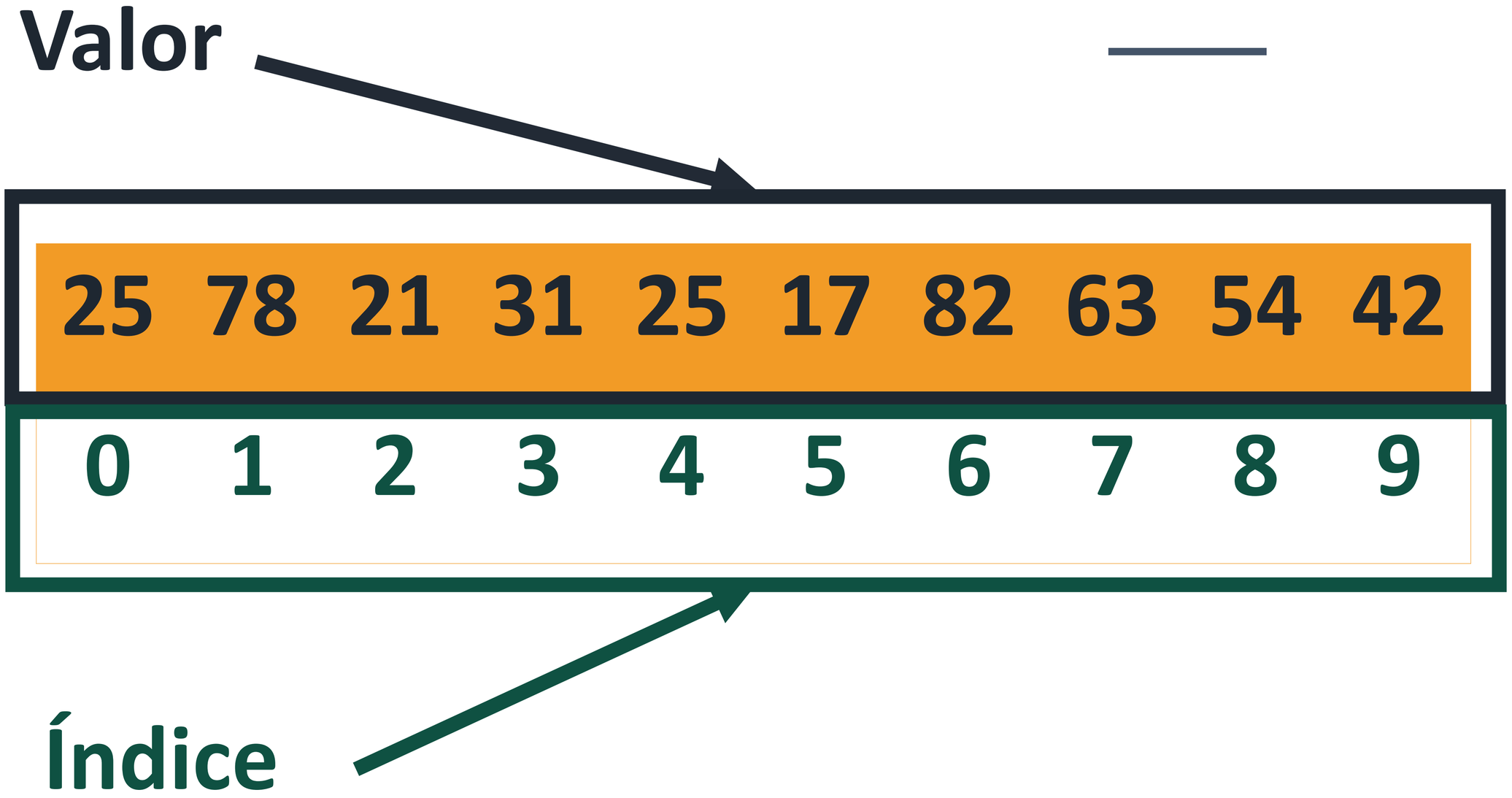
Bloque de Código

Estructuras de Repetición– Parte II

Foreach

```
foreach (tipo elemento in colección)
{
  //bloque de código
}
```

Arrays



Arrays – Declaración

```
tipo[] nombreArray = new tipo[tamanho_array];
```

Ejemplo

```
Int [] array = new int [10];
```

Arrays – Inserción de datos

Durante la declaración

```
//cantidad de elementos explícita
int[] array1 = new int[5] { 1, 3, 7, 12, 8 };

//cantidad de elementos implícita
int[] array2 = { 1, 3, 2, 7, 6 };
```

Arrays – Inserción de datos

Inserción mediante índice

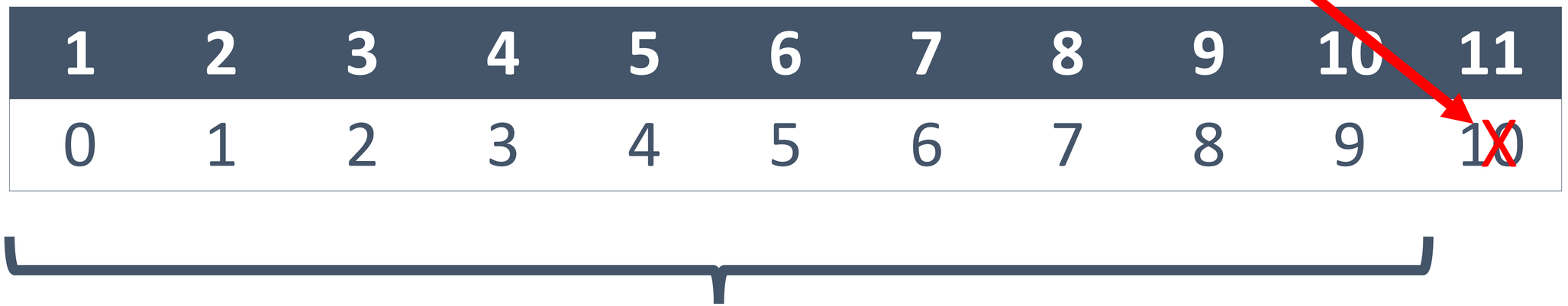
```
int[] array = new int[50];
  for (int i = 0; i < 50; i++)
  {
  array[i] = i + 1;
  }
```

array[0] == 1, array[1] == 2, array[2] == 3 …

Arrays – Acceso a los datos

Console.WriteLine (array [10]);

IndexOutOfRangeException



Int [] array = new int [10];

Arrays – Iteraciones sobre los arrays

For

```
for (int i = 0; i < 10; i++)
{
    Console.WriteLine(array[i]);
}
```

While

```
int i = 0;
while (i < 10)
{
    Console.WriteLine(array[i]);
    i++;
}
```

Foreach

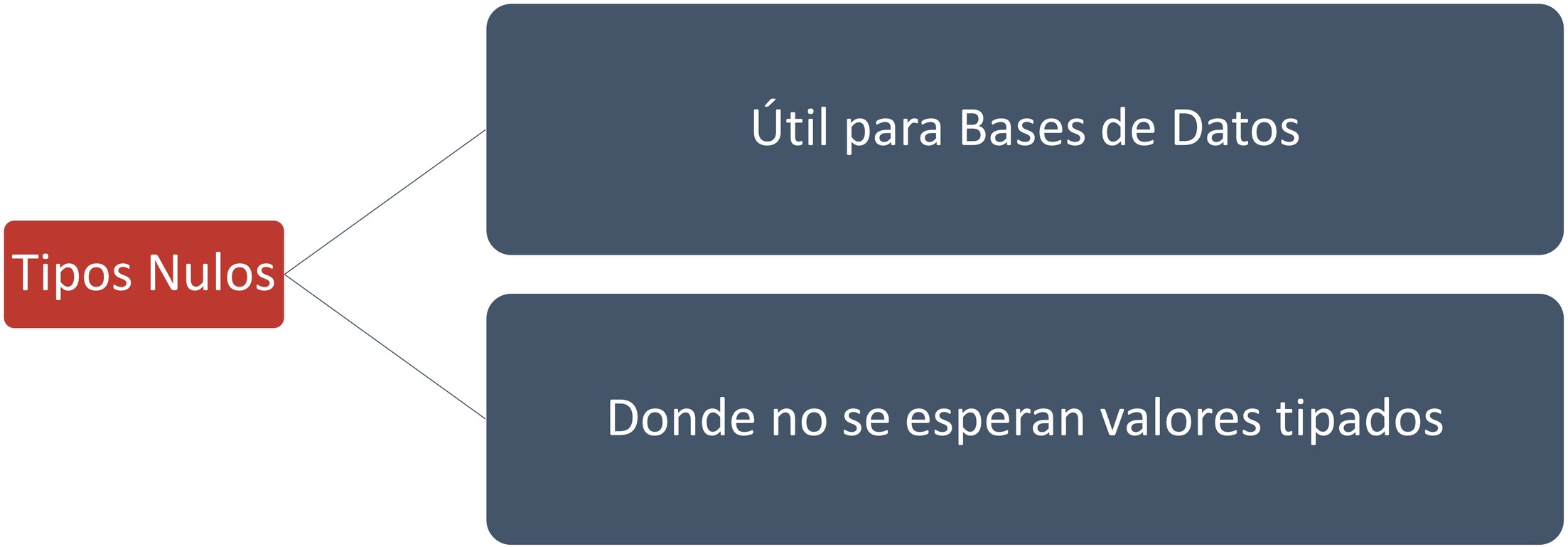
```
foreach (int x in array)
{
   Console.WriteLine(x);
}
```

Interface

IEnumerable

IEnumerable <T>

Tipos Nulos


Tipos Nulos
Útil para Bases de Datos
Donde no se esperan valores tipados

Tipos Nulos – Operadores ? y ??

```
// x puede ser un int32 o nulo

int? x = null;

// y puede ser True / False o Nulo

¿Bool? y = null;

// z puede ser un Double o Nulo

doble? z = null;
```

Tipos Nulos – Propiedad HasValue

```
// x puede ser un int32 o nulo

int? x = null;


(x.HasValue)

// x contiene un valor entero válido
```

Tipos Nulos – Propiedad Value

```
int? x = null;

// y es un valor Int32

int y = 0;

try

{

y = x.Value;

}}

catch (InvalidOperationException)

{

MessageBox.Show ("Operación no válida!");

}}
```

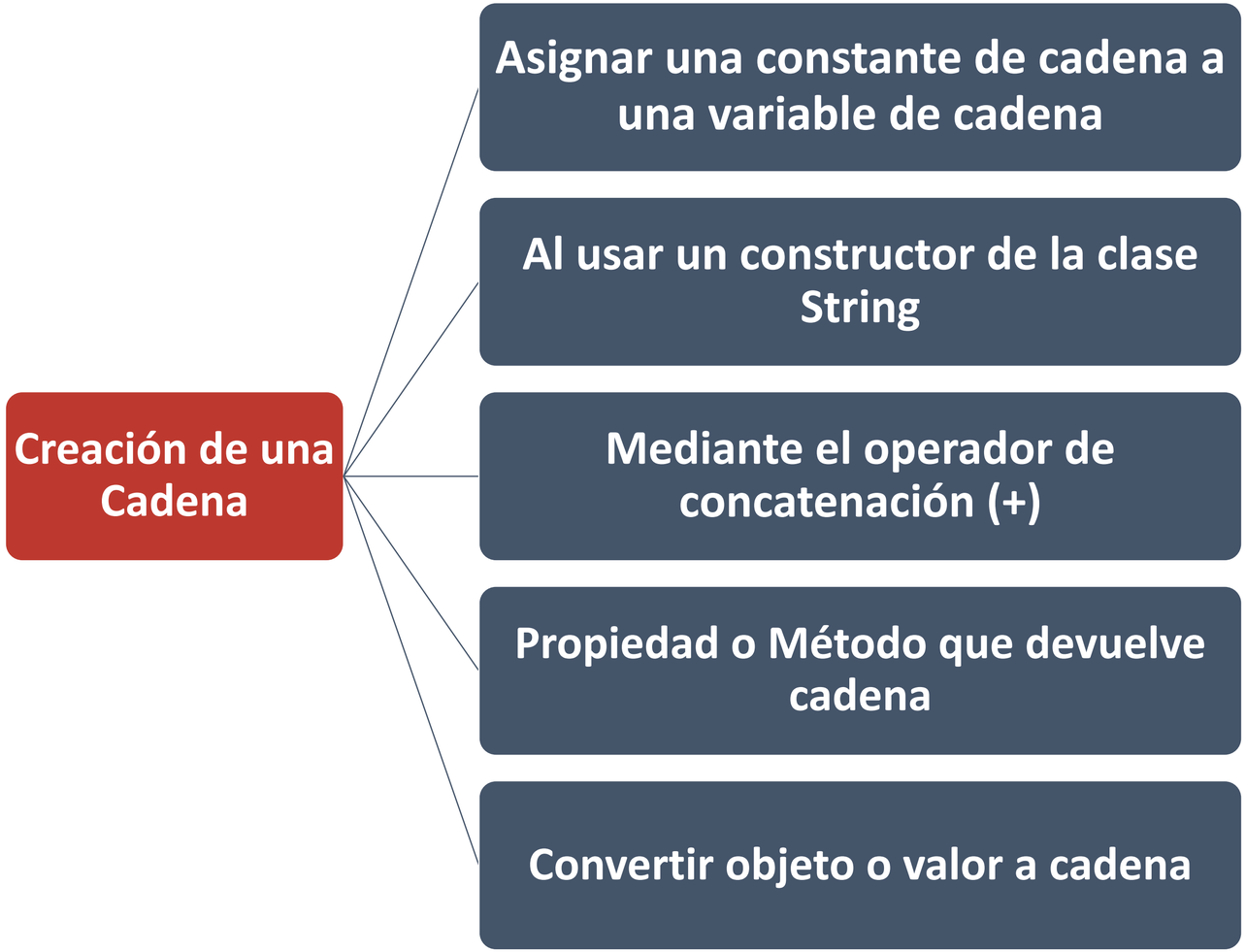
Tipos Nulos – Método GetValueOrDefault ()

```
// x puede ser un int32 o nulo

int? x = null;

// y recibirá valor 0 (cero) que es el valor predeterminado para el tipo Int32

int y = x.GetValueOrDefault ();
```

Tipos Nulos – Operador ??

```
// variable de tipo nulo

bool? b = null;

// variable de tipo Bool

bool c = false;

// el operador ?? asignará un valor True porque b es nulo

c = b ?? true;

// variable de tipo nulo

int? j = null;

// variable de tipo Int32

int k = 0;

// la variable k recibirá valor 42 pues j es nulo

k = j ?? 42;
```

Strings



string nombre = “Cadena de Texto";

System.String

Strings - Substring

```
string nombre = "Alexandre Gonzalez";
string apellidos = nombre.Substring (10, 8);
```

A	L	E	X	A	N	D	R	E		G	O	N	Z	A	L	E	Z
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

Strings - IndexOf

```
string nombre = "Alexandre Gonzalez - Programando en C #";
int numero = nombre.IndexOf ("Programando");

// Retornará 22

// A partir del índice 5
int numero = nombre.IndexOf ("Gonzalez", 5);
```

A	L	E	X	A	N	D	R	E		G	O	N	Z	A	L	E	Z		-		P	R	O	G	R	A	M	A	N	D	O		E	N		C		#
0	1	2	3	4	5	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	**22**	23	24	….	…	…	…	…	…	…	…	…	…	…	…	…	…	…

Strings - ToUpper y ToLower

string nombre = “Juan Rodriguez";
string minusculas = "";
cadena mayusculas = "";

mayusculas = nombre**.ToUpper ();**
minusculas = nombre**.ToLower ();**

Strings - StartsWith y EndsWith

```
string archivo = "comctl32.dll";
        if (archivo.StartsWith("com"))
            Console.WriteLine("Comienza por COM");
        if (archivo.EndsWith(".dll"))
            Console.WriteLine("es una dll");
        Console.ReadKey();
```

Strings - TrimStart, TrimEnd y Trim

```
string nombre = " RODRIGUEZ";
nombre = nombre.TrimEnd();
nombre = nombre.TrimStart();
nombre = nombre.Trim();
```

Strings - PadLeft y PadRight

```
string nombre = "Gonzalez";
        nombre = nombre.PadRight(10, 'XX’);
            // "GonzalezXX"
string codigo = "123";
        codigo = codigo.PadLeft(6, '0’);
            // "000123"
```

Strings - String.Join y String.Split

```
 string linea = "Prueba, 10, 20, 10/06/2018";
string[] campos = linea.Split(new char[] { ',' });
Console.WriteLine(campos[0]);
Console.WriteLine(campos[1]);
Console.WriteLine(campos[2]);
Console.WriteLine(campos[3]);

string lineaNueva = String.Join(";", campos);
Console.WriteLine(lineaNueva);
Console.ReadKey();
```

Strings - String.Format

```
string prueba = String.Format ("Voy a poner el {0} aquí.", "parámetro");
```

```
string prueba = String.Format("Formato de la cadena  con un {0} parámetro. Ahora son las {1}. Valor numérico: {2}", 1, DateTime.Now, 15.5);

Console.WriteLine(prueba);
Console.ReadKey();
```

Strings – Método Replace()

```
string texto = "Curso de CSharp de Ángel Arias, Curso de ASP.NET MVC de Ángel Arias";
string textoModificado = texto.Replace(" Curso ", " Video Curso ");

Console.WriteLine(textoModificado);
Console.ReadKey();
```

Strings – La Propiedad Length

```
string nombre = "Cursos Ángel Arias";

Console.WriteLine(nombre.Length);
Console.ReadKey();
```