

# Programación Web .NET Core

Módulo 2 - Laboratorio adicional

## Para poder realizar este laboratorio, se recomienda...

- Revisar contenidos previos.
- Realizar el laboratorio anterior.
- Descargar los elementos necesarios.



**Se recomienda realizar todos los ejercicios.** Si tienes restricciones de tiempo, puedes **exceptuar los indicados como opcionales.**

- **Ejercicio 1:** Variables de instancia vs. Miembros estáticos.
- **Ejercicio 2:** Sobrecarga de operadores (opcional).
- **Ejercicio 3:** Operadores especiales.
- **Ejercicio 4:** Combinando Virtual, Static y Herencia.

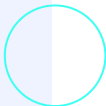


## Ejercicio 1: Variables de instancia vs. Miembros estáticos

1. Crea un nuevo proyecto de consola en una nueva solución. Identifica al proyecto y a la solución con el nombre ***MiembrosEstáticos***.
2. Agrega una clase llamada **Mensaje** con el código que verás en la siguiente pantalla.

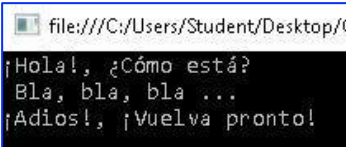


```
classMensaje
{
    publicstaticstring Bienvenida = "¡Hola!, ¿Cómo está?";
    publicstaticstring Despedida = "¡Adios!, ¡Vuelva pronto!";
    publicstaticvoid Hablar()
    {
        Console.WriteLine(" Bla, bla, bla ... ");
    }
}
```



3. Este es el código del **main**, Pruébalo.

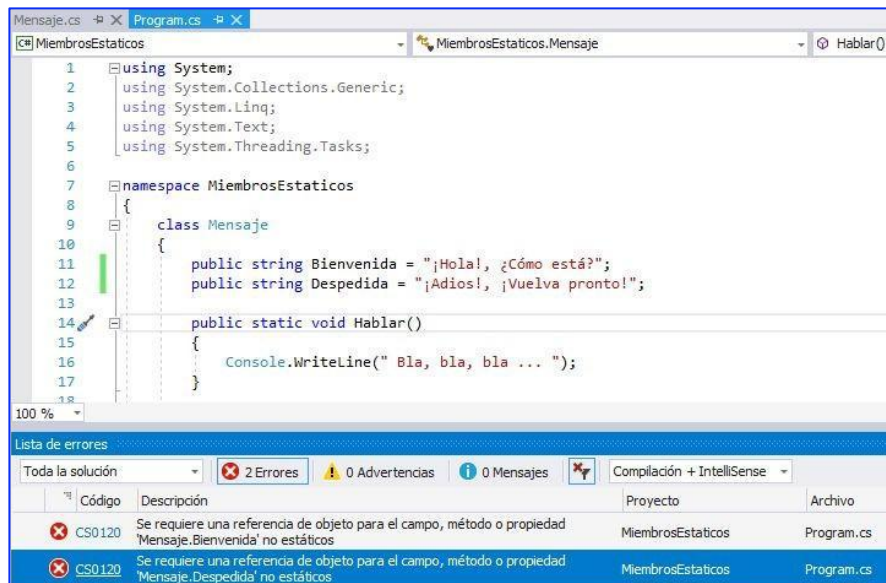
```
static void Main(string[] args)
{
    Console.WriteLine(Mensaje.Bienvenida);
    Mensaje.Hablar();
    Console.WriteLine(Mensaje.Despedida);
    Console.ReadKey();
}
```



file:///C:/Users/Student/Desktop/C

```
¡Hola!, ¿Cómo está?
Bla, bla, bla ...
¡Adios!, ¡Vuelva pronto!
```

4. Elimina de la clase **Mensaje**, todas las palabras reservadas **Static**. Intenta volver a ejecutar el programa y observa el resultado.



The screenshot shows the Visual Studio IDE with two tabs: 'Mensaje.cs' and 'Program.cs'. The 'Mensaje.cs' file is open, showing the following code:

```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6
7 namespace MiembrosEstaticos
8 {
9     class Mensaje
10     {
11         public string Bienvenida = "¡Hola!, ¿Cómo estás?";
12         public string Despedida = "¡Adios!, ¡Vuelva pronto!";
13
14         public static void Hablar()
15         {
16             Console.WriteLine(" Bla, bla, bla ... ");
17         }
18     }
19 }
```

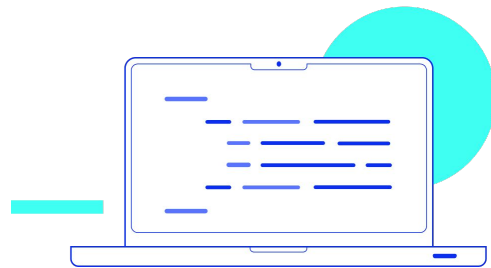
The bottom of the screen shows the 'Lista de errores' (Error List) pane. It indicates '2 Errores' (2 Errors) and '0 Advertencias' (0 Warnings). The errors are listed in a table:

Código	Descripción	Proyecto	Archivo
CS0120	Se requiere una referencia de objeto para el campo, método o propiedad 'Mensaje.Bienvenida' no estáticos	MiembrosEstaticos	Program.cs
CS0120	Se requiere una referencia de objeto para el campo, método o propiedad 'Mensaje.Despedida' no estáticos	MiembrosEstaticos	Program.cs

## Ejercicio 2: Sobrecarga de operadores (Opcional)

Probaremos ahora, la sobrecarga de operadores. Vamos a sobrecargar los operadores aritméticos y de igualdad, para operar con puntos X, Y.

1. Crea un nuevo proyecto, cuyo nombre será ***SobrecargaOperadores***, en otra carpeta.
2. Crea la clase **Point** con el código que se ve en la siguiente pantalla. Analiza las propiedades, constructores y métodos sobrecargados.





```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace SobrecargaOperadores
{
    public class Point
    {
        int x, y; // Campos

        public Point() // Constructor sin parámetros
        {
            this.x = 0;
            this.y = 0;
        }
        public Point(int x, int y) // Constructor común
        {
            this.x = x;
            this.y = y;
        }
    }
}
```

```
public int X // Propiedad
{
    get { return x; }
    set { x = value; }
}

public int Y // Propiedad
{
    get { return y; }
    set { y = value; }
}

// Operadores de igualdad
public static bool operator ==(Point p1, Point p2)
{
    return ((p1.x == p2.x) && (p1.y == p2.y));
}

public static bool operator !=(Point p1, Point p2)
{
    return !(p1 == p2);
}
```

...

```
// Operadores aritméticos
publicstaticPointoperator +(Point p1, Point p2)      {
    returnnewPoint(p1.x + p2.x, p1.y + p2.y);
}
publicstaticPointoperator -(Point p1, Point p2)
{
    returnnewPoint(p1.x - p2.x, p1.y - p2.y);
}
}
```



3. Analiza el siguiente código del **Main**, se crean 5 objetos tipo **Point** y se usan los operadores sobrecargados:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace SobrecargaOperadores
{
    class Program
    {
        static void Main(string[] args)
        {
            Point p1 = new Point(10, 20);
            Point p2 = new Point();
            p2.X = p1.X;
            p2.Y = p1.Y;
            Point p3 = new Point(22, 33);
            Console.WriteLine("p1 es: ({0},{1})", p1.X, p1.Y);
            Console.WriteLine("p2 es: ({0},{1})", p2.X, p2.Y);
            Console.WriteLine("p3 es: ({0},{1})", p3.X, p3.Y);
        }
    }
}
```

...

```
if (p1 == p2) Console.WriteLine("p1 y p2 son iguales");
else Console.WriteLine("p1 y p2 son diferentes");

if (p1 == p3) Console.WriteLine("p1 y p3 son iguales");
else Console.WriteLine("p1 y p3 son diferentes");

Console.WriteLine();
Point p4 = p1 + p3;
Console.WriteLine("p4 (p1+p3) es: ({0},{1})", p4.X, p4.Y);
Point p5 = p1 - p1;
Console.WriteLine("p5 (p1-p1) es: ({0},{1})", p5.X, p5.Y);
Console.WriteLine();
Console.WriteLine("p1 es: ({0},{1})", p1.X, p1.Y);
    Console.WriteLine("p2 es: ({0},{1})", p2.X, p2.Y);
Console.WriteLine("p3 es: ({0},{1})", p3.X, p3.Y);
    Console.WriteLine("p4 es: ({0},{1})", p4.X, p4.Y);
    Console.WriteLine("p5 es: ({0},{1})", p5.X, p5.Y);

Console.ReadLine();
    }
}
```

```
file:///C:/Users/Student/Desktop  
p1 es: (10,20)  
p2 es: (10,20)  
p3 es: (22,33)  
p1 y p2 son iguales  
p1 y p3 son diferentes  
  
p4 (p1+p3) es: (32,53)  
p5 (p1-p1) es: (0,0)  
  
p1 es: (10,20)  
p2 es: (10,20)  
p3 es: (22,33)  
p4 es: (32,53)  
p5 es: (0,0)
```

Como extensión conceptual al ejercicio anterior, y para asegurar la compatibilidad con otros lenguajes de .NET, el operador sobrecargado podría llamar a un método que realmente realice la operación, como en el código a continuación:



```
// Operadores aritméticos
publicstatic Point operator +(Point p1, Point p2)
{
    return SumaPoints(p1, p2);
}

public static Point operator -(Point p1, Point p2)
{
    return RestaPoints(p1, p2);
}

publicstatic Point RestaPoints(Point p1, Point p2)
{
    returnnew Point(p1.x - p2.x, p1.y - p2.y);
}

publicstatic Point SumaPoints(Point p1, Point p2)
{
    return new    Point(p1.x + p2.x, p1.y + p2.y);
}
```

Y respecto a los operadores de comparación:

```
// Operadores de igualdad

public static bool operator ==(Point p1, Point p2)
{
    return (p1.Equals(p2));
}

public static bool operator !=(Point p1, Point p2)
{
    return (!p1.Equals(p2));
}
```





...

```
public override    bool  Equals(    object    o)
    {
    // Por valor
    if (((Point)o).x == this.x) && (((Point)o).y == this.y))
    return true ;
    elsereturnfalse    ;
    }

publicoverrideint GetHashCode()
    {
    return (this.ToString().GetHashCode());
    }
```

Para más información sobre **GetHashCode** consulte este [link](#).

Para finalizar, mencionamos que la sobrecarga de los operadores relacionales obliga a implementar la interface **IComparable**, concretamente el método **CompareTo**:

```
public class Point : IComparable
{
    .....

    // Operadores relacionales
    public int CompareTo(object o)
    {
        Point tmp = (Point)o;
        if (this.x > tmp.x) return 1;
        else
            if (this.x < tmp.x) return -1;
        else return 0;
    }

    public static bool operator <(Point p1, Point p2)
    {
        IComparable ic1 = (IComparable)p1;
        return (ic1.CompareTo(p2) < 0);
    }
}
```

...

```
public static bool operator > (Point p1, Point p2)
{
    IComparable ic1 = (IComparable)p1;
    return (ic1.CompareTo(p2) > 0);
}

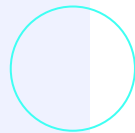
public static bool operator <= (Point p1, Point p2)
{
    IComparable ic1 = (IComparable)p1;
    return (ic1.CompareTo(p2) <= 0);
}

public static bool operator >= (Point p1, Point p2)
{
    IComparable ic1 = (IComparable)p1;
    return (ic1.CompareTo(p2) >= 0);
}
```

...

...

```
.....  
} // class Point  
static void Main(string[] args)  
{ .....  
if (p1 > p2) Console.WriteLine("p1 > p2");  
if (p1 >= p2) Console.WriteLine("p1 >= p2");  
if (p1 < p2) Console.WriteLine("p1 < p2") ;  
if (p1 <= p2) Console.WriteLine("p1 <= p2");  
if (p1 == p2) Console.WriteLine("p1 == p2");  
    Console.WriteLine();  
  
if (p1 > p3) Console.WriteLine("p1 > p3");  
if (p1 >= p3) Console.WriteLine("p1 >= p3");  
if (p1 < p3) Console.WriteLine("p1 < p3");  
if (p1 <= p3) Console.WriteLine("p1 <= p3");  
if (p1 == p3) Console.WriteLine("p1 == p3");  
    Console.WriteLine();  
}  
}
```



El operador de asignación (**=**), cuando se aplica a clases, copia la referencia, no el contenido. Si se desea copiar instancias de clases, lo habitual en C# es redefinir (**override**) el método llamado **MemberwiseCopy()** que heredan, por defecto, todas las clases en C# de **System.Object**.



## Ejercicio 3: Operadores especiales

Presentamos y ejercitamos aquí, los operadores especiales **is**, **as** y **typeof**, muy útiles al trabajar con distintos tipos de objetos, conversiones y herencia. **is** se utiliza para comprobar en forma dinámica si el tipo de un objeto es compatible con un tipo especificado. No conviene abusar de este operador (es preferible diseñar correctamente una jerarquía de tipos).

El siguiente ejemplo comprueba si el parámetro **o** es de tipo **Auto**, si es verdadero lo convierte al tipo **Auto** para ejecutar el método **Conducir**:

```
static void HacerAlgo(object o)
{
    if (o is Auto) ((Auto)o).Conducir();
}
```



## as

El operador **as** se utiliza para realizar conversiones entre tipos compatibles. El operador **as** se utiliza en expresiones de la forma siguiente:

```
<expresion> as type
```

**<expresion>** es una expresión de un tipo de referencia y **type** es un tipo de referencia. Si la conversión de tipo no es posible, el resultado es **null**.

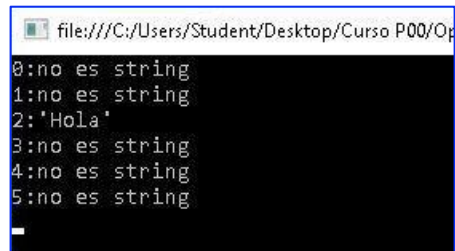
1. Crea un nuevo proyecto de consola en una nueva solución. Identifica al proyecto y a la solución con el nombre **EjemploOperadorAs**.
2. Analiza el código que se muestra en la próxima pantalla y después cópialo en el archivo **Program.cs**. Observa que se crean dos clases personalizadas, **Clase1** y **Clase2**; y en el **Main** se crea un arreglo de elementos tipo genérico **Object** y a cada elemento se le asigna un valor. Luego, en el ciclo se recorre el arreglo y se trata de convertir cada elemento a tipo **string** con el operador **as**:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace EjemploOperadorAs
{
    class Program
    {
        class Clase1 { }
        class Clase2 { }
        public class Demo
        {
            public static void Main()
            {
                object[] Objetos = new object[6];
                Objetos[0] = new Clase1();
                Objetos[1] = new Clase2();
                Objetos[2] = "Hola";
                Objetos[3] = 123;
                Objetos[4] = 123.4;
                Objetos[5] = null;
            }
        }
    }
}
```



...

```
for (int i = 0; i < Objetos.Length; ++i)
{
    string s = Objetos[i] as string;
    Console.Write("{0}:", i);
    if (s != null)
        Console.WriteLine("'" + s + "'");
    else
        Console.WriteLine("no es string");
}
Console.ReadLine();
}
```



```
file:///C:/Users/Student/Desktop/Curso P00/Op...
0:no es string
1:no es string
2:'Hola'
3:no es string
4:no es string
5:no es string
```

## typeof

El operador **typeof** devuelve el objeto derivado de **System.Type** correspondiente al tipo especificado. Una expresión **typeof** se presenta así: **typeof(tipo)** donde **tipo** es el tipo cuyo objeto **System.Type** se desea obtener.

De esta manera, se puede hacer reflexión para obtener en forma dinámica información sobre los tipos.



```
Console.WriteLine("typeof(int).FullName ES {0}", typeof(int) .FullName);
Console.WriteLine("typeof(System.Int32 ).Name ES {0}", typeof(System.Int32).Name);
Console.WriteLine("typeof(float).Module ES {0}", typeof(float) .Module);
Console.WriteLine("typeof(double).IsPublic ES {0}", typeof(double) .IsPublic);
Console.WriteLine("typeof(Point).MemberType ES {0}", typeof(Point).MemberType);
Console.WriteLine("typeof(int).BaseType ES {0}", typeof(int).BaseType); Console.ReadLine();
```

Obtendrás la siguiente salida:

```
typeof(int).FullName ES System.Int32
typeof(System.Int32 ).Name ES Int32
typeof(float).Module ES CommonLanguageRuntimeLibrary
typeof(double).IsPublic ES True
typeof(Point).MemberType ES TypeInfo
typeof(int).BaseType ES System.ValueType
-
```

Para obtener más información sobre reflexión de código, consulta por favor el siguiente [link](#).

Veremos un ejemplo en el que se emplean los operadores **is**, **as** y **typeof** para discriminar el tipo del argumento recibido en una función polimórfica y poder trabajar con él.

1. Crea un nuevo proyecto de consola en una nueva solución. Identifica al proyecto y a la solución con el nombre **EjemplolsAsTypeOf**.
2. Analiza el código de la pantalla siguiente y después cópialo en el archivo **Program.cs**. Observa que se crean dos clases personalizadas, **Clase1**, **Clase2** y **Demo**. Dentro de la clase **Demo**, un método llamado **Prueba** que evalúa los tipos de sus variables. Luego, en el **Main**, se crean diversos objetos y se llama al método **Prueba** para evaluarlos con **is**, **as** y **typeof**.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace EjemploIsAsTypeOf
{
    class Program
    {
        class Clase1 { }
        class Clase2 { }
        public class Demo
        {
            public static void Prueba(object o)
            {
                Clase1 a;
                Clase2 b;
                Type t;
```

```
...  
  
if (o is Clase1)  
{  
    Console.WriteLine("\n o es de Clase1");  
    a = (Clase1)o;  
    Console.WriteLine("--> {0}",typeof(Clase1).FullName);  
    t = typeof(Clase1);  
}  
  
elseif (o is Clase2)  
{  
    Console.WriteLine("\n o es de Clase2");  
    b = o as Clase2;  
    t = typeof(Clase2);  
}  
  
else  
{  
    Console.WriteLine("\n o no es ni de Clase1 ni de Clase2.");  
    t = o.GetType();  
    Console.WriteLine("Su tipo es " + t.FullName);  
}  
  
...
```

...

```
public static void Main()
{
    Clase1 c1 = new Clase1();
    Clase2 c2 = new Clase2();
    Prueba(c1);
    Prueba(c2);
    Prueba("Un string");
    Console.ReadLine();
}
}
```

```
o es de Clase1
--> EjemploIsAsTypeOf.Program+Clase1
Su tipo es EjemploIsAsTypeOf.Program+Clase1

o es de Clase2
Su tipo es EjemploIsAsTypeOf.Program+Clase2

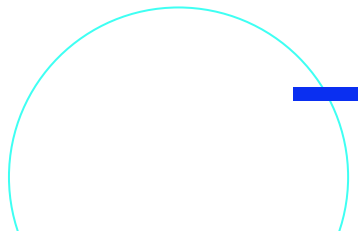
o no es ni de Clase1 ni de Clase2.
Su tipo es System.String
_
```

## Ejercicio 4: Combinación Virtual, Static y Herencia

1. Para ejercitar estos conceptos, crea una aplicación de consola: **LabCuentaBancaria**.
2. Agrega una clase **CuentaBancaria**, que representa una cuenta bancaria con propiedades:
  - **Titular** (**string**).
  - **Numero** (entero que contiene un valor único asignado en la instanciación; el primer número de cuenta es *1000*).
  - **Saldo** (**double**).
3. La clase también incluye los métodos:
  - **Credito** (permite ingresar dinero en la cuenta).
  - **Debito** (permite retirar dinero de la cuenta).
  - **ConsultaSaldo** (muestra toda la información de la cuenta).



4. Agrega una clase con el siguiente nombre: **CuentaBancariaRemunerada** que representa una cuenta bancaria remunerada que hereda de la clase creada antes (**CuentaBancaria**) y, por lo tanto, el constructor recibirá como argumentos el nombre del titular y el porcentaje de remuneración de la cuenta.
5. Redefine el método **Credito** de la clase llama-  
da **CuentaBancariaRemunerada** para que la cantidad ingresada aumente el porcentaje de remuneración definido en el constructor. ¡No hace falta soñar!, este funcionamiento bancario atípico, es solo para simplificar el ejercicio.
6. Codifica en el **Main** una secuencia que permita comprobar el funcionamiento de las clases (La solución completa se encuentra en la sección **Descargas** de este módulo). Veamos el recuadro de la siguiente pantalla.





```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace LabCuentaBancaria
{
    class Program
    {
        static void Main(string[] args)
        {
            CuentaBancaria cb1 = new CuentaBancaria("Victor");
            cb1.Credito(1000);
            System.Console.WriteLine(cb1.ConsultaSaldo());
            cb1.Debito(200);
            System.Console.WriteLine(cb1.ConsultaSaldo());

            CuentaBancariaRemunerada cb2
                = new
            CuentaBancariaRemunerada("González-Aller", 2);
            cb2.Credito(500);
            System.Console.WriteLine(cb2.ConsultaSaldo());
            System.Console.ReadKey();
        }
    }
}
```



```
classCuentaBancaria
{
    privatestaticint numCuenta = 100;

    publicstring Titular { get; set; }
    publicint Numero { get; set; }
    publicdouble Saldo { get; set; }

    publicvirtualvoid Credito(double credito)
    {
        Saldo += credito;
    }

    publicvoid Debito(double debito)
    {
        Saldo -= debito;
    }

    publicstring ConsultaSaldo()
    {
        returnstring.Format(
            "Cuenta nº{0} Titular {1} Saldo {2} pesos",
                Numero, Titular, Saldo);
    }
}
```





```
public CuentaBancaria(string titular)
{
    this.Titular = titular;
    this.Numero = CuentaBancaria.numCuenta++;
}

class CuentaBancariaRemunerada : CuentaBancaria
{
    public double PorcentajeRendimiento { get; set; }

    public CuentaBancariaRemunerada(
        string titular, double porcentajeRendimiento)
        : base(titular)
    {
        PorcentajeRendimiento = porcentajeRendimiento;
    }

    public override void Credito(double credito)
    {
        base.Credito(credito * (1 + PorcentajeRendimiento /
            100));
    }
}
```



```
C:\EducacionIT\Manual Curso CSharp\Curso completo\Descargas\Clase4\LabCui
Cuenta n°100 Titular Victor Saldo 1000 pesos
Cuenta n°100 Titular Victor Saldo 800 pesos
Cuenta n°101 Titular González-Aller Saldo 500 pesos
```



Puedes ver otro ejemplo completo sobre **herencia** en este [link](#).



En la sección de **Descargas** encontrarás los recursos necesarios para realizar los ejercicios y su resolución para que verifiques cómo te fue.



**¡Sigamos  
trabajando!**