

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:** Indexadores (opcional).
- **Ejercicio 2:** Destructores.
- **Ejercicio 3:** Conversiones de tipo (opcional).
- **Ejercicio 4:** Herencia simple.
- **Ejercicio 5:** Redefinición de métodos.
- **Ejercicio 6:** Métodos virtuales.



Ejercicio 1: Indexadores (Opcional)

Se tomará como base para este ejercicio el proyecto **EjemploClaseAuto** de las descargas de Módulo 1. Crea una carpeta donde dejarás el proyecto base, llamada **IndexadorClaseAuto**.

1. Copia el proyecto llamado **EjemploClaseAuto** a otra carpeta. Lo modificaremos para poder ejercitar indexadores.
2. Modifica el código de la clase llamada **Auto**, por el que veremos en la siguiente pantalla. La simplificaremos para el ejemplo. Analiza el **constructor** y la sobreescritura del método **ToString** de **System.Object**.



```
using System;
using System.Collections.Generic; using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace EjemploClaseAuto
{
    public class Auto
    {
        private int VelocMax;
        private string Marca;
        private string Modelo;

        public Auto(string marca, string modelo, int velocMax)
        {
            this.VelocMax = velocMax;
            this.Marca = marca;
            this.Modelo = modelo;
        }

        public override string ToString()
        {
            return (this.Marca + " " + this.Modelo +
                " (" + this.VelocMax + " Km/h)");
        }
    }
}
```

3. En el proyecto de los indexadores, agrega una nueva clase llamada **DataColeccionCoches**. El siguiente código crea su estructura. La clase está implementada con un arreglo de elementos tipo **Auto** e indica en todo momento la cantidad máxima de autos a almacenar y la cantidad actual de autos estacionados.

Observa también el código del **constructor** de la estructura donde se inicializa la cantidad de **autos**, el tope y la instanciación del arreglo.



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace EjemploClaseAuto
{
    public struct DataColeccionCoches
    {
        public int numCoches;
        public int maxCoches;
        public Auto[] Coches;

        public DataColeccionCoches(int max)
        {
            numCoches = 0;
            maxCoches = max;
            Coches = new Auto[max];
        }
    } // struct DataColeccionCoches
}
```

4. En el proyecto de los indexadores, agrega una nueva clase llamada **Autos**, la cual manejará el indexador dependiente de la estructura **DataColeccionCoches**. Copia el código y analiza los constructores, las propiedades **MaximoDeAutos** y **CantidadDeAutos**, el método **MuestraDatos** y por último, el indexador que retorna un objeto tipo **Auto**.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace EjemploClaseAuto
```

```
...  
  
{  
    public class Autos  
    {  
        DataColeccionCoches data;  
  
        // Constructor sin parámetros. Inicializa el arreglo con 10 elementos.  
        public Autos()  
        {  
            data =  
            newDataColeccionCoches(10);  
        }  
  
        // Constructor con parámetro.  
        public Autos(int max)  
        {  
            data =  
            newDataColeccionCoches(max);  
        }  
  
        public int MaximoDeAutos  
        {  
            set { data.maxCoches = value; }  
            get { return (data.maxCoches); }  
        }  
    }  
}
```

...


```
public int CantidadDeCoche
{
    set { data.numCoche = value; }
    get { return (data.numCoche); }
}

public void MuestraDatos()
{
    Console.WriteLine(" --> Maximo= {0}", MaximoDeAutos);
    Console.WriteLine(" --> Real = {0}",
        CantidadDeCoche);
}

public override string ToString()
{
    string str1 = " --> Maximo= " + this.MaximoDeAutos;
    string str2 = " --> Real = " + this.CantidadDeCoche;
    return (str1 + "\n" + str2);
}
```

...

```
// El indexador devuelve un objeto Auto de acuerdo a un índice numérico.
public Auto this[int pos]
{
    // Devuelve un objeto del vector de coches.
    get
    {
        if (pos < 0 || pos >= CantidadDeCoches)
            throw new
                IndexOutOfRangeException("Fuera de rango");
        else
            return (data.Coches[pos]);
    }
    // Escribe en el vector
    set { this.data.Coches[pos] = value; }
}

} // class Autos
}
```



5. Analiza y copia el código del **Main**, usando todas las clases. Si lo necesitas, usa el debugger para hacer un seguimiento paso a paso del proyecto. El programa va agregando **autos** y mostrando el contenido del arreglo.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace EjemploClaseAuto
{
    class Program
    {
        static void Main(string[] args)
        {
            // Crear una colección de autos
            Autos MisAutos = new Autos(); // Por omisión (10)
            Console.WriteLine("***** Mis Autos *****");
            Console.WriteLine("Inicialmente: ");
            MisAutos.MuestraDatos();
        }
    }
}
```

...

```
// Agregar autos. Observar el acceso con [] ("set").
    MisAutos[0] = newAuto("Renault", "Duster", 50);
    MisAutos[1] = newAuto("Citroën", "Berlingo", 80);
    MisAutos[2] = newAuto("Ford", "Focus", 120);
    MisAutos.CantidadDeCoches = 3;

Console.WriteLine("Despues de insertar 3 coches: ");
MisAutos.MuestraDatos();

// Mostrar la colección Console.WriteLine ();
for (int i = 0; i < MisAutos.CantidadDeCoches; i++)
{
    Console.Write("Coche Num.{0}: ", i + 1);
    Console.WriteLine(MisAutos[i].ToString()); //
    Acceso ("get")
}
```

...



```
Console.ReadLine();

//-----
// Crear otra colección de autos.
Autos TusAutos = newAutos(4);
Console.WriteLine("***** Tus Autos *****");
Console.WriteLine("Inicialmente: ");
        TusAutos.MuestraDatos();

// Agregar autos. Observar el acceso con []
        TusAutos[TusAutos.CantidadDeCoches++] = newAuto("Opel", "Corsa",
110);
        TusAutos[TusAutos.CantidadDeCoches++] = new
Auto("Citröen", "C3", 90);

Console.WriteLine("Despues de insertar 2 coches: ");
TusAutos.MuestraDatos();
```



...

```
// Mostrar la colección Console.WriteLine ();  
for (int i = 0; i < TusAutos.CantidadDeCoches; i++)  
{  
    Console.Write("Coche Num.{0}: ", i + 1);  
    Console.WriteLine(TusAutos[i].ToString()); //  
    Acceso ("get")  
}  
  
Console.ReadKey();  
    } // Main  
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace EjemploClaseAuto
{
    class Program
    {
        static void Main(string[] args)
        {
            // Crear una colección de autos
            Autos MisAutos = new Autos(); // Por
            Console.WriteLine("***** Mis Autos **");
            Console.WriteLine("Inicialmente: ");
            MisAutos.MuestraDatos();
        }
    }
}
```

```
file:///C:/Users/Student/Desktop/Curso P00/EjemploClaseAuto/
***** Mis Autos *****
Inicialmente:
--> Maximo= 10
--> Real      = 0
Despues de insertar 3 coches:
--> Maximo= 10
--> Real      = 3
Coche Num.1: Renault Duster (50 Km/h)
Coche Num.2: Citroën Berlingo (80 Km/h)
Coche Num.3: Ford Focus (120 Km/h)
```

Ejercicio 2: Destruectores

Probaremos los **destrutores** (*finalizers*) con el ejemplo a continuación:

1. Crea un nuevo proyecto cuyo nombre será ***EjemploDestruectores*** en otra carpeta.
2. Crea tres clases, utilizando el código que se muestra en la siguiente pantalla. Todas las clases tienen incluidas destructores.




```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace EjemploDestructores
{
    class Clase1
    {
        ~Clase1()
        {
            System.Diagnostics.Trace.WriteLine("El destructor de la
Clase1 es llamado.");
        }
    }
}
```



3. La **Clase2** hereda de **Clase1**:

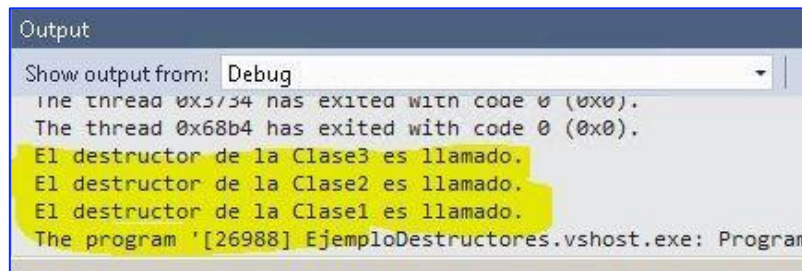
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace EjemploDestructores
{
    class Clase2: Clase1
    {
        ~Clase2()
        {
            System.Diagnostics.Trace.WriteLine("El destructor de la
Clase2 es llamado.");
        }
    }
}
```

4. La **Clase3** hereda de **Clase2**:

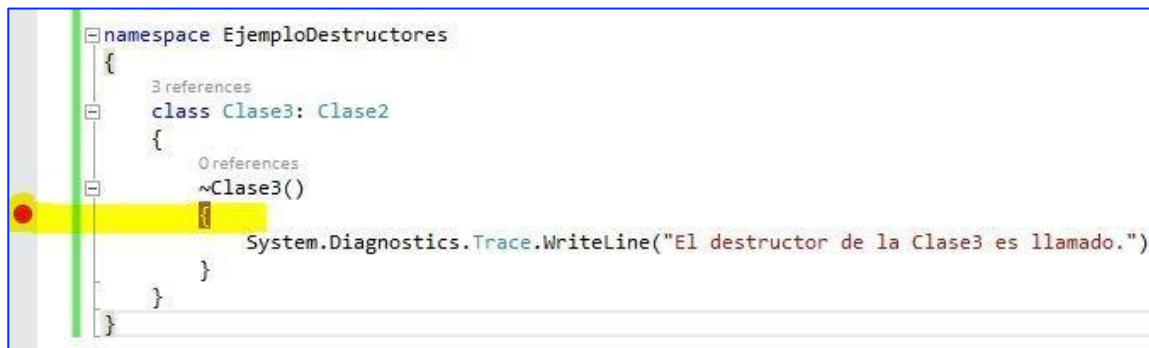
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace EjemploDestructores
{
    class Clase3: Clase2
    {
        ~Clase3()
        {
            System.Diagnostics.Trace.WriteLine("El destructor de la
Clase3 es llamado.");
        }
    }
}
```

5. En el **main** veremos cómo se ejecutan los destructores en cascada, el resultado de la ejecución saldrá en la ventana **Output**:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace EjemploDestructores
{
    class Program
    {
        static void Main(string[] args)
        {
            Clase3 t = new Clase3();
        }
    }
}
```



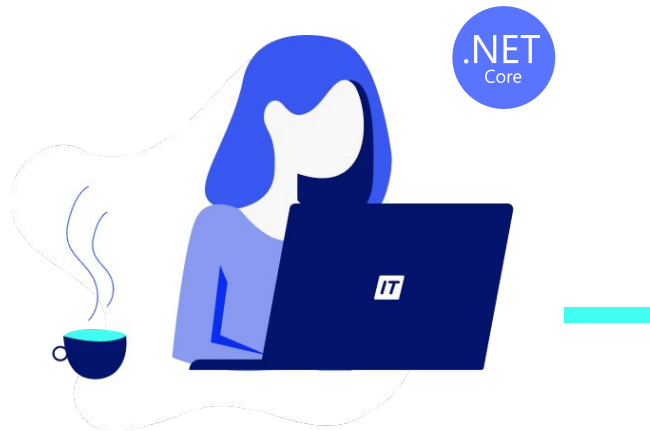
6. Prueba nuevamente la ejecución, pero antes ubica un **breakpoint** al comienzo de cada clase. Observa cuando la variable **t** queda fuera de alcance, se ejecutan los destructores en cascada. Veamos la siguiente pantalla.



Ejercicio 3: Conversiones de tipo (Opcional)

Programaremos un ejemplo de **conversiones de tipo** (tanto **explícitas** como **implícitas**):

1. Crea un nuevo proyecto cuyo nombre será **ConversionesTipo** en otra carpeta.
2. Crea una clase llamada **Euro** con el código que se muestra en la siguiente pantalla. En ella, se detallan las conversiones de tipo de dato de **Euro** a tipo **double** y viceversa.



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace ConversionesTipo
{
    public class Euro
    {
        private int cantidad; // Campo

        public Euro(int v) // Constructor común
        { cantidad = v; }

        public int valor // Propiedad
        { get { return cantidad; } }

        // Conversión implícita "double <-- Euro"
        public static implicit operator double(Euro x)
        {
            return ((double)x.cantidad);
        }
    }
}
```



```
// Conversión explícita "Euro <-- double"
public static explicit operator Euro(double x)
{
    double arriba = Math.Ceiling(x);
    double abajo = Math.Floor(x);
    int valor = ((x + 0.5 >= arriba) ? (int)arriba : (int)abajo);
    return new Euro(valor);
}

} // class Euro
}
```

Para más información sobre las funciones **Ceiling** y **Floor** consulta estos links:

- **Ceiling:** [Devuelve el número entero más pequeño mayor o igual que el número especificado.](#)
- **Floor:** [Devuelve el número entero más grande menor o igual que el número especificado.](#)

3. Analiza y prueba el código del **Main**, observa las llamadas a conversiones implícitas y explícitas, puedes usar el **debugger** para seguir el código:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace ConversionesTipo
{
    class Program
    {
        static void Main(string[] args)
        {
            double d1 = 442.578;
            double d2 = 123.22;

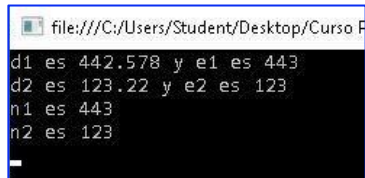
            Euro e1 = (Euro)d1; // Conversión explícita a "Euro"
            Euro e2 = (Euro)d2; // Conversión explícita a "Euro"
        }
    }
}
```

...

```
Console.WriteLine("d1 es {0} y e1 es {1}", d1, e1.valor);
Console.WriteLine("d2 es {0} y e2 es {1}", d2, e2.valor);
double n1 = e1; // Conversión implícita "double <--Euro"
double n2 = e2; // Conversión implícita "double
<--Euro"

Console.WriteLine("n1 es {0}", n1);
Console.WriteLine("n2 es {0}", n2);

Console.ReadLine();
    }
}
}
```

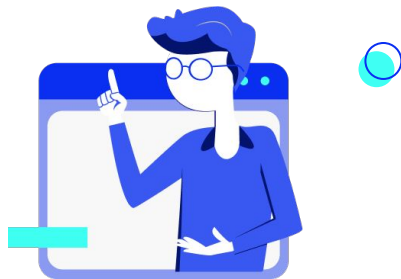


```
file:///C:/Users/Student/Desktop/Curso P...
d1 es 442.578 y e1 es 443
d2 es 123.22 y e2 es 123
n1 es 443
n2 es 123
```

Ejercicio 4: Herencia simple

1. Crea un nuevo proyecto cuyo nombre será **Herencia** en otra carpeta. Usa la clase **Auto** ya definida en otros ejemplos y extiende la clase **Taxi** heredando de **Auto**.
2. Agrega al proyecto una clase **Auto** con el código que se muestra en la pantalla siguiente; y analiza su contenido.

Atención: este ejercicio de **herencia simple** es la base para luego poder hacer los ejercicios **redefinición de métodos** y **métodos virtuales**. Te sugiero que lo realices en primer lugar.



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace Herencia
{
    class Auto
    {
        private int VelocMax;
        private string Marca;
        private string Modelo;

        public Auto()
        {
            VelocMax = 0;
            Marca = "?";
            Modelo = "?";
        }
    }
}
```

...

```
public Auto(string marca, string mod, int velMax)
{
    VelocMax = velMax;
    Marca = marca;           Modelo = mod;
}

public void MuestraAuto()
{
    Console.WriteLine(Marca + " " + Modelo + " (" + VelocMax + " Km/h)");
}
} // class Auto
}
```



3. Agrega al proyecto una clase **Taxi** con el siguiente código y analiza su contenido. Esta heredará de la clase **Auto**. Observa que el constructor de **Taxi** hereda del constructor de **Auto**.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace Herencia
{
    class Taxi : Auto
    {
        private string CodLicencia;
```



...

```
public Taxi() { }  
public Taxi(string marca, string mod, int vel, string lic)  
:  
base(marca, mod, vel)  
    {  
        CodLicencia = lic;  
    }  
public string Licencia  
    {  
get { return CodLicencia; }  
    }  
} // class Taxi }
```

4. Analice y pruebe el código del **Main**:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace Herencia
{
    class Program
    {
        static void Main(string[] args)
        {
            Auto MiAuto = new Auto("Citroën", "Xsara Picasso", 220);
            Auto TuAuto = new Auto("Opel", "Corsa", 190);
            Auto UnAuto = new Auto();
        }
    }
}
```



...

```
Console.Write("Mi auto: ");
    MiAuto.MuestraAuto();
Console.Write("El tuyo: ");
    TuAuto.MuestraAuto();
Console.Write("Un auto sin identificar: ");
    UnAuto.MuestraAuto();

Console.WriteLine();

Taxi ElTaxiDesconocido = newTaxi();
Console.Write("Un taxi sin identificar: ");
    ElTaxiDesconocido.MuestraAuto();

Taxi NuevoTaxi = newTaxi("Citroën", "C5", 250, "GR1234");
Console.Write("Un taxi nuevo: ");
    NuevoTaxi.MuestraAuto();
Console.Write("    Licencia: {0}", NuevoTaxi.Licencia);
Console.ReadLine();
    }
}
```

```
file:///C:/Users/Student/Desktop/Curso P00/Herencia/Herencia/bin/Deb
Mi auto: Citroën Xsara Picasso (220 Km/h)
El tuyo: Opel Corsa (190 Km/h)
Un auto sin identificar: ?? ?? (0 Km/h)

Un taxi sin identificar: ?? ?? (0 Km/h)
Un taxi nuevo: Citroën C5 (250 Km/h)
Licencia: GR1234_
```

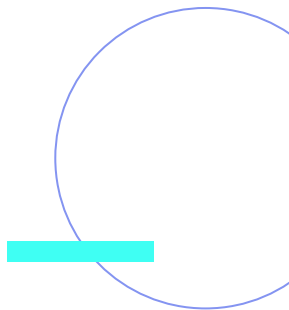


Ejercicio 5: Redefinición de métodos

Probaremos la redefinición de métodos en el proyecto que ejercitaste **Herencia**.

1. En la clase **Auto** redefine el método **ToString** con el código a continuación. Esto evitará usar el método **MuestraAuto()** para ver los datos del auto:

```
public override string ToString()
{
    return (Marca + " " + Modelo + " (" + VelocMax + " Km/h)");
}
```



2. Y en la clase **Taxi**, dado que hereda de la clase **Auto**, tomaremos el método **ToString** heredado de la clase base y lo reescribiremos para mostrar también los atributos del taxi:

```
public override string ToString()
{
    return (base.ToString() + "\n    Licencia: " +
        Licencia);
}
```

3. Para poder probar la redefinición, en el **Main** comenta la línea de código:

```
//Console.Write("Un taxi sin identificar: ");
//ElTaxiDesconocido.MuestraAuto();
```

4. y agrega debajo la siguiente:

```
Console.WriteLine("Un taxi sin identificar: " +
    ElTaxiDesconocido);
```

5. Haz lo mismo que en el paso anterior, pero con el objeto **NuevoTaxi**:

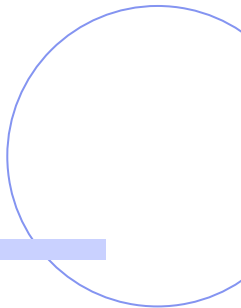
```
//Console.Write("Un taxi nuevo: ");
//NuevoTaxi.MuestraAuto();
//Console.Write("    Licencia: {0}",
    NuevoTaxi.Licencia);
Console.WriteLine("Un taxi nuevo: " + NuevoTaxi);
```

Ejercicio 6: Métodos virtuales

Indicaremos métodos virtuales en el proyecto que ejercitaste **Herencia**.

1. En el proyecto de Herencia, prueba redefinir el método **MuestraAuto** en la clase **Taxi**. Recuerda que este método está definido en la clase base **Auto**:

```
public override void MuestraAuto()
{
    Console.WriteLine(Marca + " " + Modelo + " " + Licencia +
        " (" + VelocMax + " Km/h)");
}
```



2. Observa que aparece un error que indica que el método heredado **MuestraAuto** no se puede sobrescribir si no es **virtual**, **abstract** u **override**.

```
void Taxi.MuestraAuto()
```

'Taxi.MuestraAuto()': cannot override inherited member 'Auto.MuestraAuto()' because it is not marked virtual, abstract, or override



3. La solución a esto es definir en la clase **Auto** al método **MuestraAuto** como virtual, y a los atributos **VelocMax**, **Marca** y **Modelo** como **protected**. Veamos cómo queda el código de la clase **Auto** con estas modificaciones:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace Herencia
{
    class Auto
    {
        protected int VelocMax;
        protected string Marca;
        protected string Modelo;
    }
}
```



```
...  
  
public Auto()  
{  
    VelocMax = 0;  
    Marca = "??";  
    Modelo = "??";  
}  
  
public Auto(string marca, string mod, int velMax)  
{  
    VelocMax = velMax;  
    Marca = marca;           Modelo = mod;  
}  
  
public virtual void MuestraAuto()  
{  
    Console.WriteLine(Marca + " " + Modelo + " (" + VelocMax + " Km/h)");  
}  
  
public override string ToString()  
{  
    return (Marca + " " + Modelo + " (" + VelocMax + " Km/h)");  
}  
  
} // class Auto  
}
```

Nota: Las propiedades, los indexadores y asimismo los eventos, también pueden ser virtuales.

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



**¡Sigamos
trabajando!**