

Programación Web .NET Core

Módulo 1 - Laboratorio adicional

Para poder realizar este laboratorio, se recomienda...

- Revisar contenidos previos.
- Descargar los elementos necesarios.





Se recomienda realizar todos los ejercicios, excepto los indicados como opcionales si tuviera restricciones de tiempo. **El objetivo es asegurar la ejercitación adecuada para los contenidos de este módulo:**

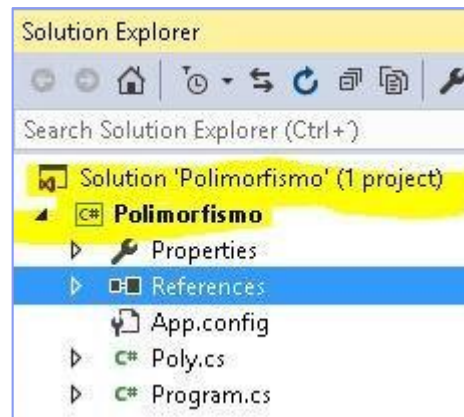
- **Ejercicio 1:** Polimorfismo.
- **Ejercicio 2:** Enumeraciones.
- **Ejercicio 3:** Estructuras (opcional).
- **Ejercicio 4:** Clases.
- **Ejercicio 5:** Clases con lógica de negocio real, realizado en .Net Framework y en .Net Core

Ejercicio 1: Polimorfismo

Este ejercicio tiene como objetivo lograr la comprensión del concepto de *polimorfismo*. Para ello se simplifica ejercitando con una aplicación de consola. Recordemos que el **polimorfismo es la capacidad de realizar una operación sobre un objeto sin conocer su tipo concreto**.

Probaremos esta habilidad con un ejemplo:

1. Crea un nuevo **proyecto de consola** en una nueva solución. Identifica al proyecto y a la solución con el nombre ***Polimorfismo***, como vemos en la imagen de la derecha.



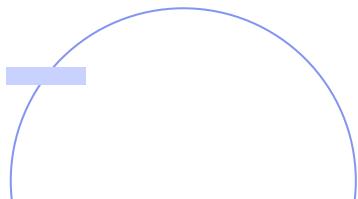
2. Agrega al proyecto un nuevo archivo de de clase, y en él crea la clase **ClasePoly**, con el método **Poly**, con el siguiente código:

```
namespace Polimorfismo
{
    class ClasePoly
    {
        public void Poly(object o)
        {
            Console.WriteLine(o.ToString())
        }
    }
}
```

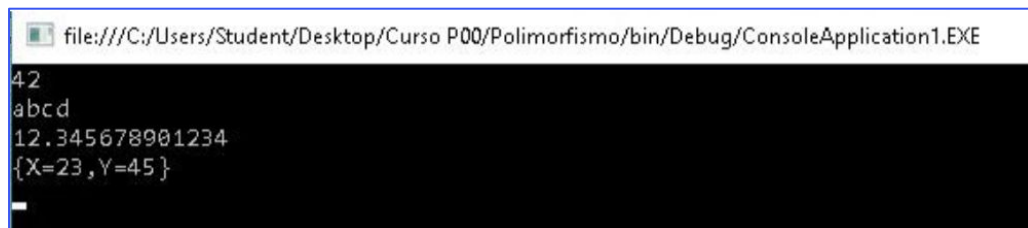
3. El método **Poly** espera como parámetro algo del tipo **Object**. Recuerda que **todo hereda de System.Object** por lo que cualquier tipo de dato que se envíe al método, podrá convertirse, en este caso, a un **string** para mostrarlo. **Esta es una forma de polimorfismo.**

4. Escribe el código del recuadro siguiente en el **Main**. Tendrás que crear una referencia (lógica y física) a la librería **System.Drawing**:

```
namespace Polimorfismo
{
    class Program
    {
        static void Main(string[] args)
        {
            ClasePoly x = new ClasePoly();
            x.Poly(42);
            x.Poly("abcd");
            x.Poly(12.345678901234m);
            x.Poly(new Point(23, 45));
            Console.ReadKey();
        }
    }
}
```

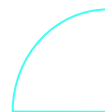


5. Ejecuta la aplicación y observa la salida. Independientemente del tipo de dato enviado al método **Poly**, se obtiene su valor en la consola de la aplicación, a través de **comportamiento polimórfico**.



```
file:///C:/Users/Student/Desktop/Curso P00/Polimorfismo/bin/Debug/ConsoleApplication1.EXE
42
abcd
12.345678901234
{X=23,Y=45}
```

The screenshot shows a console window with a black background and white text. The title bar indicates the file path: file:///C:/Users/Student/Desktop/Curso P00/Polimorfismo/bin/Debug/ConsoleApplication1.EXE. The output consists of four lines: the integer 42, the string 'abcd', the double 12.345678901234, and the object {X=23,Y=45}. A cursor is visible on the line following the last output.



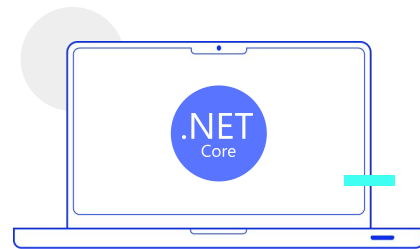
Ejercicio 2: Enumeraciones

Este ejercicio tiene como objetivo aprender el uso de **enumeraciones**. Para ello se simplifica ejercitando con una aplicación de consola. Una **enumeración** o **tipo enumerado** es un tipo especial de **estructura** en la que los literales de los valores que pueden tomar sus objetos, se indican explícitamente al definirla.

Pueden convertirse explícitamente en su valor entero (como en C). Admiten determinados operadores aritméticos (+, -, ++, --) y lógicos a nivel de bits (&, |, ^, ~). Todos los tipos que hemos enumerado, **derivan de System.Enum**.

Probaremos esta habilidad con un ejemplo:

1. Crea un nuevo proyecto de consola en una nueva solución. Identifica al proyecto y a la solución con el nombre **Enumeraciones**.



2. Escribe el código siguiente en la **clase** y en el **Main**. Hay varios ejemplos de enumeraciones.

```
using System;
using System.Collections.Generic; using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace Enumeraciones
{
    class Program
    {
        enum State { Off, On }

        enum Color
        {
            Red = 1,
            Green = 2,
            Blue = 4,
            Black = 0,
            White = Red | Green | Blue
        }
    }
}
```

...

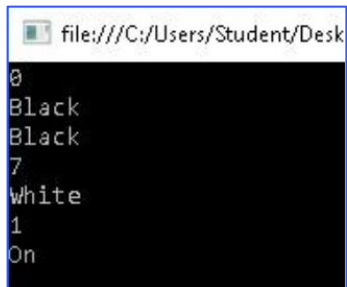
```
static void Main(string[] args)
{
    Color c = Color.Black;
    Console.WriteLine((int)c); // 0
    Console.WriteLine(c.ToString()); // Black
    Console.WriteLine(c); // Black
    Console.ReadKey();

    Color x = Color.White;
    Console.WriteLine((int)x); // 7 suma los valores de los colores
    Console.WriteLine(x.ToString()); // White
    Console.ReadKey();

    State y = State.On;
    Console.WriteLine((int)y); // 1
    Console.WriteLine(y.ToString()); // On
    Console.ReadKey();
}
}
```



3. Ejecuta la aplicación y observa la salida.



```
file:///C:/Users/Student/Desktop
0
Black
Black
7
white
1
On
```

Ejercicio 3: Estructuras (Opcional)

Este ejercicio tiene como objetivo aprender el uso de estructuras. Para ello se simplifica ejercitando con una aplicación de consola.

Una **estructura** es un tipo especial de **clase pensada para representar objetos ligeros** (que ocupen poca memoria y deban ser manipulados con velocidad) como objetos que representen, por ejemplo: puntos, fechas, etc.

Una estructura en C# es tipo valor, sus miembros pueden ser **public**, **internal** o **private**, y puede tener un constructor sin parámetros.

Probaremos esta habilidad con un ejemplo:

1. Crea un nuevo proyecto de consola en una nueva solución. Identifica al proyecto y a la solución con el nombre **Estructuras**.



2. Escribe el código siguiente en la **clase** y en el **Main**.

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

namespace Estructuras
{
    class Program
    {
        public struct Point
        {
            public int x, y;

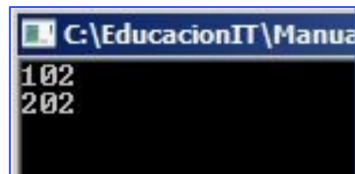
            public Point(int x, int y) //Constructor
            {
                this.x = x; this.y = y;
            }
        }
    }
}
```

```
static void Main(string[] args)
{
    Point p = new Point(2, 5);
    Point p2;

    p.x += 100;
    int px = p.x;    // px = 102
    p2 = p;           p2.x += 100;
    Console.WriteLine(px); // 102
    Console.WriteLine(p2.x); // 202

    Console.ReadKey();
}
}
```

Observa la salida.



Ejercicio 4: Clases

Este ejercicio tiene como objetivo aprender el uso de clases. Para ello se simplifica ejercitando con una aplicación de consola.

Un **objeto** es un agregado de datos y de métodos los cuales permiten manipular dichos datos; y un programa en C#, no es más que un conjunto de objetos que interaccionan unos con otros y lo hacen a través de sus métodos.

Una clase es la definición de las características concretas de un determinado tipo de objetos: cuáles son los **datos** y los **métodos** de los que van a disponer todos los objetos de ese tipo.

Probaremos esta habilidad simulando el funcionamiento de un auto:

1. Crea un nuevo proyecto de consola en una nueva solución. Identifica al proyecto y a la solución con el nombre **EjemploClaseAuto**.
2. En el proyecto agrega una clase, cuyo nombre será el siguiente: **Auto**.
3. En la clase **Auto**, define las variables privadas para almacenar los valores de los atributos. Veamos el detalle en la siguiente pantalla.

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

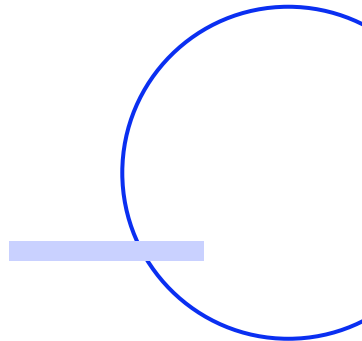
namespace EjemploClaseAuto
{
    class Auto
    {
        protected double _velocidad = 0; // Se accede dentro
        // de la clase Auto y en las clases derivadas
        private string _marca;
        private string _modelo;
        private string _color;
    }
}
```

Nota: se podrían haber definido los atributos como **public**, pero no se estaría aplicando el concepto de encapsulamiento de la POO.



4. En la clase **Auto** y a continuación de la definición de variables, agrega un constructor específico, además del constructor default que tiene toda clase.

```
// Constructor
public Auto(string marca, string modelo, string color)
{
    _marca = marca;
    _modelo = modelo;
    _color = color;
}
```



5. En la clase **Auto** y a continuación del constructor, agregue la propiedad **Velocidad**, que será de sólo lectura. La misma sólo será modificada a través de **métodos**.

```
// Propiedad Velocidad (solo lectura)
publicdouble Velocidad
{
    get{ return _velocidad; }
}
```



6. En la clase **Auto** agrega las propiedades de lectura/escritura: **Marca**, **Modelo** y **Color**:

```
publicstring Marca {
    get{ return _marca; }
    set{ _marca= value; }
}
publicstring Modelo
{
    get{ return _modelo; }
    set{ _modelo = value; }
}
publicstring Color
{
    get{ return _color; }
    set{ _color = value; }
}
```

7. En la clase **Auto** define los siguientes métodos, cuyos nombres son: **Acelerar**, **Girar**, **Frenar** y **Estacionar**. Observa cómo en Acelerar, Girar y Estacionar, se modifica el valor del atributo **velocidad**:

```
//      Método Acelerar
public void Acelerar(double cantidad)
{
    Console.WriteLine("--> Incrementando veloc. en {0} km/h", cantidad);
    _velocidad += cantidad;
}
// Método Girar
public void Girar(double cantidad)
{
    Console.WriteLine("--> Girando {0} grados", cantidad);
}
```



```
// Método Frenar
public void Frenar(double cantidad)
{
    Console.WriteLine("--> Reduciendo veloc. en {0} km/h", cantidad);
    _velocidad -= cantidad;
}

// Método Estacionar
public void Estacionar()
{
    Console.WriteLine("--> Estacionando auto");
    _velocidad = 0;
}
```



8. Agrega el código del programa principal en el **Main**:

```
static void Main(string[] args)
{
    Auto MiAuto = new Auto("Renault", "Duster", "Rojo");
    Console.WriteLine("Los datos de mi coche son:");
    Console.WriteLine("  Marca: {0}", MiAuto.Marca);
    Console.WriteLine("  Modelo: {0}", MiAuto.Modelo);
    Console.WriteLine("  Color: {0}", MiAuto.Color);
    Console.WriteLine();

    MiAuto.Acelerar(100);
    Console.WriteLine("La velocidad actual es de {0} km/h \n",
        MiAuto.Velocidad);

    MiAuto.Frenar(75);
    Console.WriteLine("La velocidad actual es de {0} km/h \n",
        MiAuto.Velocidad);
}
```



```
MiAuto.Girar(45);  
  
MiAuto.Estacionar();  
Console.WriteLine("La velocidad actual es de {0} km/h \n",  
MiAuto.Velocidad);  
  
Console.ReadLine();  
}
```



9. Se detalla el código completo de la clase y el programa:

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

namespace EjemploClaseAuto
{
    class Auto
    {
        // Almacenan el valor de los atributos públicos
        protected double _velocidad = 0; // Se accede dentro de la clase Auto y en las clases derivadas
        private string _marca;
        private string _modelo;
        private string _color;

        // Constructor
        public Auto(string marca, string modelo, string color)
        {
            _marca = marca;
            _modelo = modelo;
            _color = color;
        }
    }
}
```



```
//          Propiedad Velocidad (solo lectura)
publicdouble Velocidad
{
    get{ return _velocidad; }
}

publicstring Marca {
    get{ return _marca; }
    set{ _marca= value; }
}

publicstring Modelo
{
    get{ return _modelo; }
    set{ _modelo = value; }
}

publicstring Color
{
    get{ return _color; }
    set{ _color = value; }
}
```




```
// Método Acelerar
public void Acelerar(double cantidad)
{
    Console.WriteLine("--> Incrementando veloc. en {0} km/h", cantidad);
    _velocidad += cantidad;
}

// Método Girar
public void Girar(double cantidad)
{
    Console.WriteLine("--> Girando {0} grados", cantidad);
}

// Método Frenar

public void Frenar(double cantidad)
{
    Console.WriteLine("--> Reduciendo veloc. en {0} km/h", cantidad);
    _velocidad -= cantidad;
}
```

```
...  
  
// Método Estacionar  
public void Estacionar()  
{  
    Console.WriteLine("-->Estacionando auto");  
    _velocidad = 0;  
}  
}
```

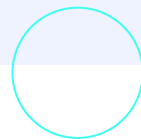
```
...
```

```
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)
        {
            Auto miAuto = new Auto("Renault", "Duster", "Rojo");
            Console.WriteLine("Los datos de mi coche son:");
            Console.WriteLine("Marca: {0}", miAuto.Marca);
            Console.WriteLine("Modelo: {0}", miAuto.Modelo);
            Console.WriteLine("Color: {0}", miAuto.Color);
            Console.WriteLine();

            miAuto.Acelerar(100);
            Console.WriteLine("La velocidad actual es de {0} km/h \n",
                miAuto.Velocidad);
        }
    }
}
```



```
MiAuto.Frenar(75);  
Console.WriteLine("La velocidad actual es de {0} km/h \n",  
MiAuto.Velocidad);  
  
MiAuto.Girar(45);  
  
MiAuto.Estacionar();  
Console.WriteLine("La velocidad actual es de {0} km/h \n",  
MiAuto.Velocidad);  
  
Console.ReadLine();  
    }  
}  
}
```



```
C:\EducacionIT\Manual Curso CSharp\Curso comp
Los datos de mi coche son:
    Marca: Renault
    Modelo: Duster
    Color: Rojo

--> Incrementando veloc. en 100 km/h
La velocidad actual es de 100 km/h

--> Reduciendo veloc. en 75 km/h
La velocidad actual es de 25 km/h

--> Girando 45 grados
--> Estacionando auto
La velocidad actual es de 0 km/h
```

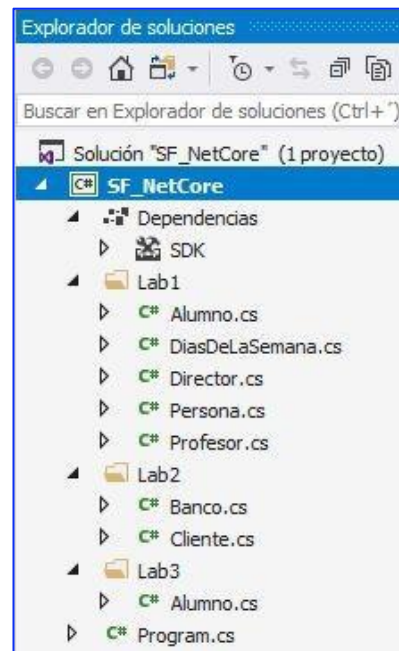
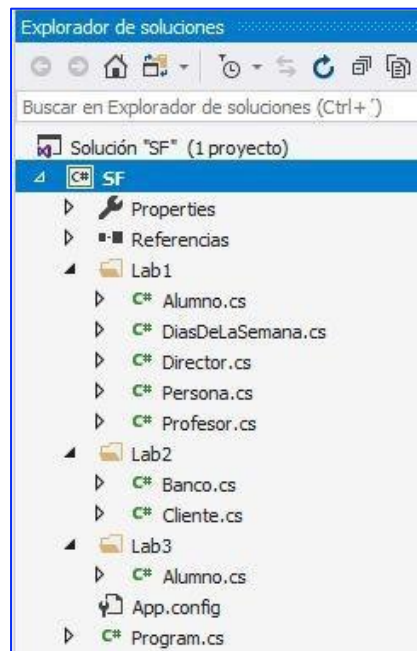
Ejercicio 5: Clases con lógica de negocio real

Para un nuevo proyecto de la *Software Factory* (SF) solicitan que creamos la estructura correspondiente, según se detalla en los siguientes tres escenarios. El proyecto de consola con archivos de clase y la solución completa, se encuentra en la sección **Descargas** de esta clase, en dos tipos de proyecto distintos:

- **SF.zip** proyecto de consola .Net Framework
- **SF_NetCore.zip** proyecto de consola de .Net Core (multiplataforma)

Recomendamos que, antes de ver las posibles soluciones propuestas, trates de hacer primero los ejercicios.





Escenario 1

Se va a desarrollar una aplicación para un instituto de enseñanza, donde las personas pueden cumplir una o más funciones. Utilizando **POO y herencia**, crea las siguientes **clases**, teniendo en cuenta lo antes nombrado. Las pruebas de funcionamiento de dichas clases, se realizarán con un **proyecto de Consola**.

- **Persona**

- **Apellido**
- **Nombre**
- **DNI**
- **Teléfono**

- **Alumno**: Mismos campos que **Persona** más:

- **Curso**
- **Horario**

- **Profesor**: Mismos campos que **Persona** más:

- **Dia**
- **Curso**

- **Director**: Mismos campos que **Profesor** más:

- **NroInstituto**

Escenario 2

La Software Factory (SF) decidió aparte, plantear una **clase** llamada **Alumno** y definir como atributos su **nombre** y su **edad**. En el **constructor**, realiza el **ingreso de datos**.

Define otros dos métodos para **imprimir los datos ingresados** y un mensaje si es **mayor o no de edad** (edad ≥ 18).

Las pruebas de funcionamiento de dichas clases, las realizarás con un **proyecto de Consola**.



Escenario 3

La Software Factory (SF) del ejercicio anterior, también tiene un proyecto para un banco. El banco tiene 3 clientes que pueden hacer depósitos y extracciones. Asimismo, el banco requiere que al final del día, se pueda calcular la cantidad de dinero que se encuentra depositada.

La solución tendrá el esquema que figura en los recuadros de la derecha. Se deberán **definir las propiedades y los métodos de cada clase**.

Las pruebas de funcionamiento de dichas clases, las realizarás con un **proyecto de Consola**.

Cliente

→ Nombre →

Monto

Métodos:

constructor

Depositar

Extraer

RetornarMonto

Banco

→ Cliente

Métodos:

constructor

Operar

DepositosTotales

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



**¡Sigamos
trabajando!**