

# LABORATORIOS CLASE 2

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

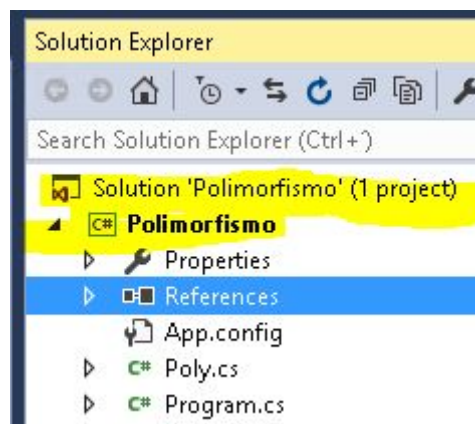
- Ejercicio 1: Polimorfismo
- Ejercicio 2: Enumeraciones
- Ejercicio 3: Estructuras (opcional)
- Ejercicio 4: Clases
- Ejercicio 5: Clases con lógica de negocio real, realizado en .Net Framework y en .Net Core
- Ejercicio Integrador del curso

## 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 polimorfismo es la capacidad de realizar una operación sobre un objeto sin conocer su tipo concreto.

Probaremos esta habilidad con un ejemplo:

1. Cree un nuevo proyecto de consola en una nueva solución. Identifique al proyecto y a la solución con el nombre **Polimorfismo**:



2. Agregue al proyecto un nuevo archivo de de clase, y en él cree 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. Recuerde que todo hereda de System.Object por lo que cualquier tipo de dato que se le envíe al método, podrá convertirse en este caso a un string para mostrarlo. **Esta es una forma de polimorfismo.**
4. Escriba el código siguiente en el Main. Tendrá que crear una referencia (lógica y física) a la librería **System.Drawing** (consulte al instructor cómo hacerlo).

```

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. Ejecute la aplicación y observe la salida. Independientemente del tipo de dato enviado al método Poly, obtenemos 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}
```

## 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 enumerados derivan de **System.Enum**.

Probaremos esta habilidad con un ejemplo:

1. Cree un nuevo proyecto de consola en una nueva solución. Identifique al proyecto y a la solución con el nombre **Enumeraciones**.
2. Escriba el código siguiente en la clase y en el Main. Hay diversos ejemplos sobre 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
}

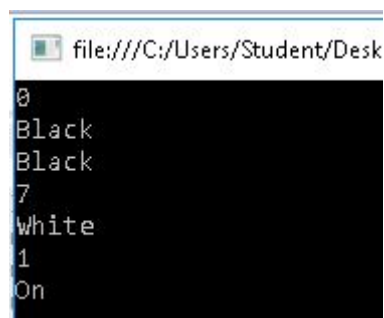
static void Main(string[] args)
{
    Color c = Color.Black;
    Console.WriteLine((int)c); // 0
    Console.WriteLine(c.ToString()); // Black
    Console.WriteLine(c); // Black
    Console.ReadKey();

    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. Ejecute la aplicación y observe la salida.



```

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

```

## Ejercicio 3: Estructuras

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 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. Cree un nuevo proyecto de consola en una nueva solución. Identifique al proyecto y a la solución con el nombre **Estructuras**.
2. Escriba 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)
        {
```

```

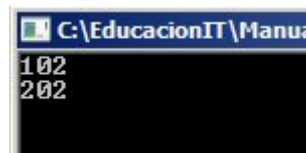
    Point p = new Point(2, 5);
    Point p2;

    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();

}
}
}

```



## 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 que permiten manipular dichos datos, y un programa en C# no es más que un conjunto de objetos que interaccionan unos con otros 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 es funcionamiento de un auto:

1. Cree un nuevo proyecto de consola en una nueva solución. Identifique al proyecto y a la solución con el nombre **EjemploClaseAuto**.
2. En el proyecto, agregue una clase que la llamará **Auto**.

3. En la clase Auto, defina las variables privadas para almacenar los valores de los atributos.

```
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;
    }
}
```

<p><b>Nota:</b> podríamos haber definido los atributos como public, pero no estaríamos aplicando el concepto de encapsulamiento de la POO.</p>
--

4. En la clase **Auto** y a continuación de la definición de variables, agregue 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)
public double Velocidad
```

```

{
    get { return _velocidad; }
}

```

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

```

public string Marca {
    get { return _marca; }
    set { _marca= value; }
}
public string Modelo
{
    get { return _modelo; }
    set { _modelo = value; }
}
public string Color
{
    get { return _color; }
    set { _color = value; }
}

```

7. En la clase **Auto** defina los siguientes métodos: **Acelerar**, **Girar**, **Frenar** y **Estacionar**. Observe como 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. Agregue 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)
        public double Velocidad
        {
            get { return _velocidad; }
        }

        public string Marca {
            get { return _marca; }
            set { _marca= value; }
        }
        public string Modelo
        {
            get { return _modelo; }
            set { _modelo = value; }
        }
        public string 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();
    }
}

```

The screenshot shows a console window titled "C:\EducacionIT\Manual Curso CSharp\Curso comp". The output text is as follows:

```

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

```

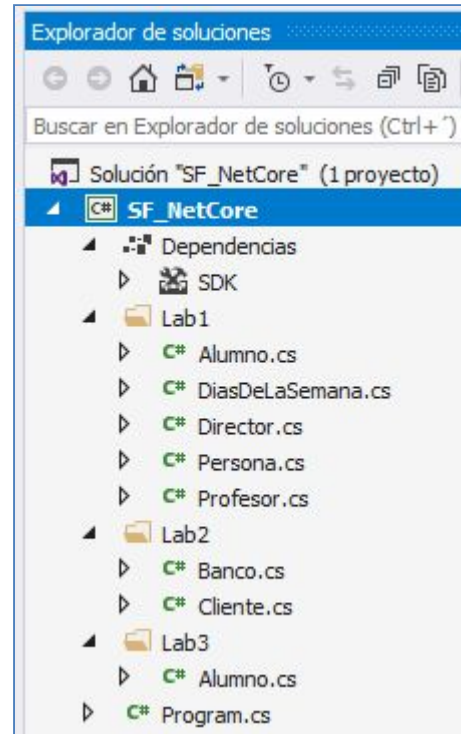
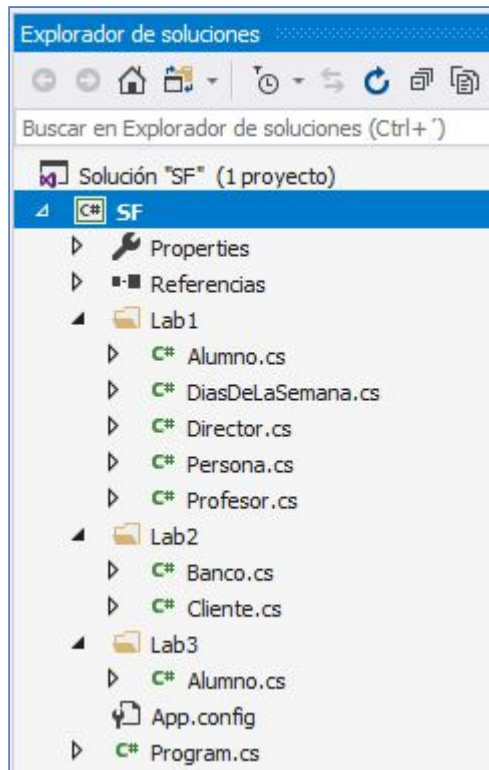
## Ejercicio 5: Clases con lógica de negocio real

Para un nuevo proyecto de la Software Factory (SF) solicitan que creemos la estructura correspondiente según se detalla en los siguientes dos 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 de **.Net Framework**
- **SF\_NetCore.zip** proyecto de consola de **.Net Core** (multiplataforma)

**Recomendamos trate de hacer primero los ejercicios antes de ver las posibles soluciones propuesta!**



## 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, crear las siguientes clases teniendo en cuenta lo antes nombrado. Las pruebas de funcionamiento de dichas clases, las realizará 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) del ejercicio anterior, también tiene un proyecto para un banco. En banco tiene 3 clientes que pueden hacer depósitos y extracciones. También el banco requiere que al final del día calcule la cantidad de dinero que hay depositada.

La Solución tendrá el siguiente esquema: Debemos definir las propiedades y los métodos de cada clase:

### Cliente

- Nombre
- Monto
- Métodos:
  - constructor
  - Depositar
  - Extraer
  - RetornarMonto

### Banco

- Cliente
- Métodos:
  - constructor
  - Operar
  - DepositosTotales

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

## Escenario 3:

La Software Factory (SF) decidió aparte, plantear una clase llamada **Alumno** y definir como atributos su nombre y su edad. En el constructor realizar el ingreso de datos. Definir otros dos métodos para imprimir los datos ingresados y un mensaje si es mayor o no de edad (edad  $\geq 18$ ).

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

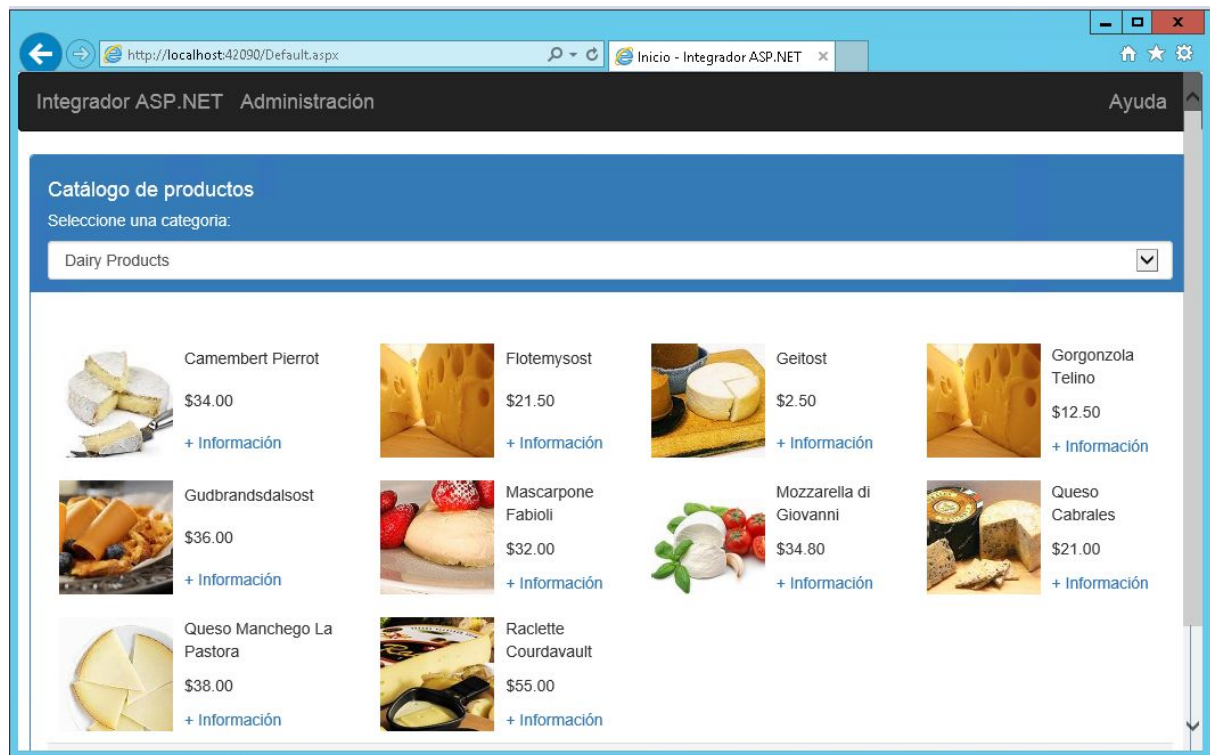
## Ejercicio Integrador del curso

Presentamos brevemente el ejercicio integrador del curso, el mismo se encuentra en la sección de descargas en el archivo **IntegradorTerminado.zip**.

**Deberá solicitarle al docente instrucciones para poder dejarla en línea y ejecutarla, dado que usa una base de datos y es necesario establecer la conexión a la misma, dado que Ud. todavía no tiene los conocimientos necesarios que obtendrá a lo largo del curso.**

En esta etapa del curso sólo observada el funcionamiento de la aplicación y los datos que muestra.

Se trata de una aplicación web que muestra información sobre productos clasificados en categorías.



Los datos de los productos se encuentran en una base de datos.

También puede cargar nuevos productos, modificar y eliminar productos existentes.

Integrador ASP.NET Administración Ayuda

Administración

		Nombre	Categoría	Proveedor	Precio
Modificar	Eliminar	Alice Mutton	Meat/Poultry	Pavlova, Ltd.	\$39.00
Modificar	Eliminar	Aniseed Syrup	Condiments	Exotic Liquids	\$10.00
Modificar	Eliminar	Boston Crab Meat	Seafood	New England Seafood Cannery	\$18.40
Modificar	Eliminar	Camembert Pierrot	Dairy Products	Gai pâturage	\$34.00
Modificar	Eliminar	Carnarvon Tigers	Seafood	Pavlova, Ltd.	\$62.50
Modificar	Eliminar	Chai	Beverages	Exotic Liquids	\$18.00
Modificar	Eliminar	Chang	Beverages	Exotic Liquids	\$19.00
Modificar	Eliminar	Chartreuse verte	Beverages	Aux joyeux ecclésiastiques	\$18.00
Modificar	Eliminar	Chef Anton's Cajun Seasoning	Condiments	New Orleans Cajun Delights	\$22.00
Modificar	Eliminar	Chef Anton's Gumbo Mix	Condiments	New Orleans Cajun Delights	\$21.35

12345678

Integrador ASP.NET Administración Ayuda

Camembert Pierrot

Nombre:

Categoría:

Proveedor:

Presentación:

Unidades en stock:     Unidades pedidas:     Nivel de reposición:

Precio:     Descontinuado: ☐