FleetManager S.A

Sistema de Gestión de Vehículos

Resultados de aprendizaje y criterios de evaluación **1**
Enunciado 2
Consideraciones de diseño **2**

Resultados de aprendizaje y criterios de evaluación

RA1. Comprender y aplicar los fundamentos del lenguaje C# en el desarrollo de aplicaciones	~~1a~~	~~Se han identificado los fundamentos de la programación estructurada y el lenguaje C# (tipos de datos, variables, operadores).~~
	~~1b~~	~~Se han identificado los fundamentos de la programación orientada a objetos.~~
	1c	Se han implementado clases con atributos, métodos y constructores aplicando encapsulación.
	1d	Se han aplicado los principios de herencia y polimorfismo en ejercicios prácticos.

Enunciado

La empresa **FleetManager S.A.** gestiona una flota de vehículos de transporte de pasajeros y de carga. Necesita un sistema flexible para calcular el **Costo Operacional por Kilómetro** de cada unidad.

Todo vehículo tiene una **Matrícula** y un **Consumo de Combustible** (litros por cada 100 km). Su **Costo Operacional Base** es fijo (0.15€ por litro).

Los vehículos de **transporte de pasajeros** tienen además una **Capacidad Máxima**.
En el caso de los **Autobuses** se aplica un **Factor de Desgaste** del 1.2€ al costo operacional base: *Coste Por KM = Consumo x Costo Operacional Base x Factor de Desgaste*.

Los vehículos de **transporte de carga** tienen además un **Peaje Anual**.
En el caso de los **Camiones** su costo operacional por kilómetro es el costo base más un **Costo Fijo de Peaje por Kilómetro**, calculado como **Peaje Anual / 100,000 km**: *Coste Por KM = Consumo x Costo Operacional Base x Peaje Anual / 100,000.0.*

El sistema debe **calcular** el **Coste Por Km** de cada vehículo de su flota.

Todos los atributos numéricos críticos (**consumo, capacidad, carga, peaje, …**) deben validarse de forma que si se les quiere asignar un **valor negativo**, el sistema debe asignarle automáticamente **0.0** como medida preventiva.

Diseña una aplicación de consola que incluya un menú con la siguiente funcionalidad:

1. **Registrar Vehículo:** Permite al usuario seleccionar el tipo de vehículo (Autobús o Camión), introducir sus datos y añadir la instancia a una **colección interna de vehículos**.
2. **Ver Costos Operacionales:** Recorre la colección, mostrando los atributos del vehículo (usando `ToString()`) y su Costo Operacional Total (llamando a `CalcularCostoPorKm()`).
3. **Calcular Costo Total de Flota (Consumo Fijo):** Suma todos los Costos Operacionales Totales de la colección, asumiendo una distancia fija de **100,000.0 km** por vehículo.
4. **Salir:** Termina la ejecución de la aplicación.

Consideraciones de diseño

- Los constructores de una clase se deben apoyar en el constructor de la clase base.
- Cuando sea posible, un método polimórfico de una clase se debe apoyar en el mismo método polimórfico de la clase base.
- Puede haber propiedades de sólo lectura.
- Puede haber casos en que una clase no necesite implementar los métodos polimórficos de su clase base.

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

internal class Program
{
    class Programa
    {
        // ======================================
        // COLECCIÓN INTERNA (POLIMORFISMO)
        // ======================================
        static List<Vehiculo> flota = new List<Vehiculo>();

        // ======================================
        // MÉTODOS DE UTILIDAD
        // ======================================

        static int LeerOpcion()
        {
            while (true)
            {
                Console.Write("Introduce una opción entre 1 y 4: ");
                int opcion;
                if (int.TryParse(Console.ReadLine(), out opcion))
                {
                    if (opcion >= 1 && opcion <= 4)
                        return opcion;
                }
                Console.WriteLine("Opción no válida.");
            }
        }

        static double LeerDoubleNoNegativo(string mensaje)
        {
            while (true)
            {
                Console.Write(mensaje);
                double valor;
                if (double.TryParse(Console.ReadLine(), out valor))
                {
                    if (valor < 0)
                    {
                        Console.WriteLine("Valor negativo detectado. Se
asignará 0.0 automáticamente.");
```

```
                    return 0.0;
                }
                return valor;
            }
            Console.WriteLine("Valor inválido. Introduce un número
válido.");
        }
    }

    static string LeerTexto(string mensaje)
    {
        Console.Write(mensaje);
        string texto = Console.ReadLine();
        if (string.IsNullOrWhiteSpace(texto))
        {
            return "Sin datos";
        }
        return texto;
    }

    // ======================================
    // OPCIÓN 1: REGISTRAR VEHÍCULO
    // ======================================
    static void RegistrarVehiculo()
    {
        Console.WriteLine("Tipos de vehículo: autobus / camion");
        Console.Write("Introduce el tipo de vehículo: ");
        string tipo = Console.ReadLine().ToLower();

        string matricula = LeerTexto("Matrícula: ");
        double consumo = LeerDoubleNoNegativo("Consumo (litros/100
km): ");

        if (tipo.Equals("autobus"))
        {
            double capacidad = LeerDoubleNoNegativo("Capacidad
máxima (pasajeros): ");
            Autobus bus = new Autobus(matricula, consumo,
capacidad);
            flota.Add(bus);
            Console.WriteLine("Autobús registrado correctamente.");
        }
        else if (tipo.Equals("camion"))
        {
            double peaje = LeerDoubleNoNegativo("Peaje anual (€):
");
```

```
                Camion cam = new Camion(matricula, consumo, peaje);
                flota.Add(cam);
                Console.WriteLine("Camión registrado correctamente.");
            }
            else
            {
                Console.WriteLine("Tipo de vehículo no reconocido.");
            }
        }

        // ======================================
        // OPCIÓN 2: VER COSTOS OPERACIONALES
        // ======================================
        static void VerCostosOperacionales()
        {
            if (flota.Count == 0)
            {
                Console.WriteLine("No hay vehículos registrados.");
                return;
            }

            foreach (Vehiculo v in flota)
            {
                Console.WriteLine(v.ToString());
                Console.WriteLine($"Costo operacional por km:
{v.CalcularCostoPorKm():0.0000} €/km");

Console.WriteLine("-----------------------------------------");
            }
        }

        // ======================================
        // OPCIÓN 3: COSTO TOTAL DE FLOTA
        // ======================================
        static void CalcularCostoTotalFlota()
        {
            const double DISTANCIA = 100000.0;
            double total = flota.Sum(v => v.CalcularCostoPorKm() *
DISTANCIA);
            Console.WriteLine($"Costo total estimado de la flota (para
{DISTANCIA:0} km por vehículo): {total:0.00} €");
        }

        // ======================================
        // MENÚ
        // ======================================
```

```
        static void Menu()
        {
            Console.WriteLine();
            Console.WriteLine("===== FleetManager S.A - Sistema de
Gestión de Vehículos =====");
            Console.WriteLine("1- Registrar vehículo");
            Console.WriteLine("2- Ver costos operacionales");
            Console.WriteLine("3- Calcular costo total de flota (100.000
km)");
            Console.WriteLine("4- Salir");
        }

        // ======================================
        // MAIN
        // ======================================
        public static void Main()
        {
            Console.OutputEncoding = Encoding.UTF8; // para que se vea
el símbolo €
            while (true)
            {
                Menu();
                int opcion = LeerOpcion();

                switch (opcion)
                {
                    case 1:
                        RegistrarVehiculo();
                        break;
                    case 2:
                        VerCostosOperacionales();
                        break;
                    case 3:
                        CalcularCostoTotalFlota();
                        break;
                    case 4:
                        return;
                }
            }
        }

        // ======================================
        // CLASES DE DOMINIO
        // ======================================

        // Ejercicio 1: Clase base Vehiculo
```

```
        // Requisito técnico de abstracción
        public abstract class Vehiculo
        {
            public string Matricula { get; set; }

            private double _consumo;
            public double Consumo
            {
                get => _consumo;
                set => _consumo = value < 0 ? 0.0 : value;
            }

            // Propiedad de solo lectura
            public double CostoOperacionalBase => 0.15;

            protected Vehiculo(string matricula, double consumo)
            {
                Matricula = string.IsNullOrWhiteSpace(matricula) ? "Sin
matrícula" : matricula;
                Consumo = consumo;
            }

            // Polimorfismo
            public virtual double CalcularCostoPorKm()
            {
                return Consumo * CostoOperacionalBase;
            }

            public override string ToString()
            {
                return $"Vehículo -> Matrícula: {Matricula}, Consumo:
{Consumo:0.00} L/100km, Costo base: {CostoOperacionalBase:0.00} €/L";
            }
        }

        // Ejercicio 2: Clase Autobus
        // Requisito técnico de herencia
        public class Autobus : Vehiculo
        {
            private double _capacidadMaxima;
            public double CapacidadMaxima
            {
                get => _capacidadMaxima;
                set => _capacidadMaxima = value < 0 ? 0.0 : value;
            }
```

```
            // Factor de desgaste de 1.2
            public double FactorDesgaste => 1.2;

            public Autobus(string matricula, double consumo, double
capacidadMaxima)
                : base(matricula, consumo)
            {
                CapacidadMaxima = capacidadMaxima;
            }

            // Coste por km = consumo × costo base × factor de desgaste
            public override double CalcularCostoPorKm()
            {
                return base.CalcularCostoPorKm() * FactorDesgaste;
            }

            public override string ToString()
            {
                return $"Autobús -> Matrícula: {Matricula}, Consumo:
{Consumo:0.00} L/100km, Capacidad: {CapacidadMaxima:0.00} pasajeros,
Factor desgaste: {FactorDesgaste}";
            }
        }

        // Ejercicio 3: Clase Camion
        public class Camion : Vehiculo
        {
            private double _peajeAnual;
            public double PeajeAnual
            {
                get => _peajeAnual;
                set => _peajeAnual = value < 0 ? 0.0 : value;
            }

            public Camion(string matricula, double consumo, double
peajeAnual)
                : base(matricula, consumo)
            {
                PeajeAnual = peajeAnual;
            }

            // Coste por km = consumo × costo base + (peaje anual /
100000)
            public override double CalcularCostoPorKm()
            {
                return base.CalcularCostoPorKm() + (PeajeAnual /
```

```
100000.0);
            }

            public override string ToString()
            {
                return $"Camión -> Matrícula: {Matricula}, Consumo:
{Consumo:0.00} L/100km, Peaje anual: {PeajeAnual:0.00} €";
            }
        }
    }
}
```

===== FleetManager S.A - Sistema de Gestión de Vehículos =====
1- Registrar vehículo
2- Ver costos operacionales
3- Calcular costo total de flota (100.000 km)
4- Salir
Introduce una opción entre 1 y 4: 1
Tipos de vehículo: autobus / camion
Introduce el tipo de vehículo: autobus
Matrícula: 1111ggg
Consumo (litros/100 km): 8
Capacidad máxima (pasajeros): 12
Autobús registrado correctamente.

===== FleetManager S.A - Sistema de Gestión de Vehículos =====
1- Registrar vehículo
2- Ver costos operacionales
3- Calcular costo total de flota (100.000 km)
4- Salir
Introduce una opción entre 1 y 4: 2
Autobús -> Matrícula: 1111ggg, Consumo: 8,00 L/100km, Capacidad: 12,00 pasajeros, Factor desgaste: 1,2
Costo operacional por km: 1,4400 €/km

===== FleetManager S.A - Sistema de Gestión de Vehículos =====
1- Registrar vehículo
2- Ver costos operacionales
3- Calcular costo total de flota (100.000 km)
4- Salir
Introduce una opción entre 1 y 4: 3
Costo total estimado de la flota (para 100000 km por vehículo): 144000,00 €

===== FleetManager S.A - Sistema de Gestión de Vehículos =====
1- Registrar vehículo
2- Ver costos operacionales
3- Calcular costo total de flota (100.000 km)
4- Salir
Introduce una opción entre 1 y 4: 4

C:\repositorio\ASPNet\ASP-A2.11 Ejercicio de repaso para FleetManager S.A\ASP-A2.11-FleetManager S.A\ASP-A2.11-FleetManager S.A\bin\Debug\net8.0\ASP-A2.11-FleetManager S.A.exe (proceso 32416) se cerró con el código 0 (0x0).
Para cerrar automáticamente la consola cuando se detiene la depuración, habilite Herramientas ->Opciones ->Depuración ->Cerrar la consola automáticamente al detenerse la depuración.
Presione cualquier tecla para cerrar esta ventana. . .