LogiTrack S.A

Iris Pérez Aparicio

2º DAM

Campus FP Emprende Humanes

Sistema de Gestión de Envíos (LogiTrack)

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 **LogiTrack S.A.** es una compañía de logística que necesita un sistema flexible para calcular los costos de envío y los tiempos de entrega de diferentes tipos de paquetes. El sistema debe ser capaz de gestionar una jerarquía común de envíos:

- **Envío Base:** Todos los envíos tienen una **Descripción** y un **Peso** en kilogramos. El costo base del manejo de cualquier paquete es de 2.0€ por kilogramo.
- **Paquete Estándar:** Además del costo base, tiene una **Tarifa Plana** fija por el tipo de servicio (10.0€). Su **Costo Total** es la suma del costo base más la tarifa plana.
- **Paquete Express:** Además del costo base, tiene un **Recargo por Urgencia** (un porcentaje aplicado al peso). Su **Costo Total** es la suma del costo base más el producto del **Recargo por Urgencia** por el **Peso**.

Un requisito fundamental es la **buena validación de entrada de datos**, para asegurar que ningún valor numérico crítico (**peso, tarifa, recargo**) pueda ser negativo. Si se intenta asignar un valor negativo, el sistema debe asignarle automáticamente **0.0** como medida preventiva.

Ejercicio 1: Abstracción, Encapsulación y Validación (Clase Base `Envio`)

Crea la clase base `Envio`:

- Propiedades automáticas: **Descripción**.
- Propiedades no automáticas: **Peso** (con validación de no negativo, asignando 0.0 si es el caso).
- Propiedad de Solo Lectura: **CostoBase** (fijada en 2.0€ por kilogramo).
- Constructor que inicialice las propiedades.
- Métodos polimórficos:
    - `CalcularCostoTotal()` (método).
    - `ToString()` (método virtual sobrescrito).

El objetivo es establecer la clase base y definir la encapsulación con validación para los atributos comunes.

Código (Archivo **Envio.cs**)

```
namespace IrisPerezAparicio_ASP_A2_10_LogiTrack_SA
{
    class Envio
    {
        // Propiedad automática
        public string Descripcion { get; set; }

        // Propiedad no automática con validación
        private double _peso;
        public double Peso
        {
            get => _peso;
            set => _peso = value < 0 ? 0.0 : value;
        }

        // Constante de coste base
        protected const double COSTO_POR_KG = 2.0;

        // Solo lectura
        public double CostoBase => Peso * COSTO_POR_KG;

        // Constructor base
        public Envio(string descripcion, double peso)
        {
            Descripcion = descripcion;
            Peso = peso;
        }

        // Método polimórfico base reutilizable
        public virtual double CalcularCostoTotal()
        {
            return CostoBase;
        }

        // ToString polimórfico
        public override string ToString()
        {
            return $"Descripción: {Descripcion}, Peso: {Peso:F2} kg,
Costo Base: {CostoBase:F2} euros";
        }
    }
}
```

Ejercicio 2: Herencia y Primera Implementación (`PaqueteEstandar`)

Crea la clase heredada `PaqueteEstandar`:

- Hereda de `Envio`.
- Propiedades no automáticas: **TarifaPlana** (con validación de no negativo).
- Constructor que inicialice las propiedades.
- Métodos polimórficos:
    - **`CalcularCostoTotal()`** (implementando la fórmula específica: CostoBase + TarifaPlana).
    - **`ToString()`** (extendiéndolo para incluir la tarifa).

Código (Archivo **PaqueteEstandar.cs**)

```
namespace IrisPerezAparicio_ASP_A2_10_LogiTrack_SA
{
    class PaqueteEstandar : Envio
    {
        // Tarifa plana fija recomendada: 10.0€
        private double _tarifaPlana;
        public double TarifaPlana
        {
            get => _tarifaPlana;
            set => _tarifaPlana = value < 0 ? 0.0 : value;
        }

        public PaqueteEstandar(string descripcion, double peso, double
tarifaPlana = 10.0)
            : base(descripcion, peso)
        {
            TarifaPlana = tarifaPlana;
        }

        // CostoTotal = base + tarifa plana
        public override double CalcularCostoTotal()
        {
            return base.CalcularCostoTotal() + TarifaPlana;
        }
```

```
        public override string ToString()
        {
            return $"[Estándar] {base.ToString()}, Tarifa Plana:
{TarifaPlana:F2} euros, Costo Total: {CalcularCostoTotal():F2} euros";
        }
    }
}
```

Ejercicio 3: Atributos Específicos y Polimorfismo Final (`PaqueteExpress`)

Crea la clase heredada `PaqueteExpress`:

- Hereda de `Envio`.
- Propiedades no automáticas: **RecargoUrgencia** (con validación de no negativo).
- Constructor que inicialice las propiedades.
- Métodos polimórficos:
    - **`CalcularCostoTotal()`** (implementando la fórmula específica: CostoBase + RecargoUrgencia x Peso).
    - **`ToString()`** (extendiéndolo para incluir el recargo).

Código (Archivo **PaqueteExpress.cs**)

```
namespace IrisPerezAparicio_ASP_A2_10_LogiTrack_SA
{
    class PaqueteExpress : Envio
    {
        // Recargo de urgencia con validación (lo tratamos como importe
€/kg o factor según enunciado)
        private double _recargoUrgencia;
        public double RecargoUrgencia
        {
            get => _recargoUrgencia;
            set => _recargoUrgencia = value < 0 ? 0.0 : value;
        }
```

```
        public PaqueteExpress(string descripcion, double peso, double
recargoUrgencia)
            : base(descripcion, peso)
        {
            RecargoUrgencia = recargoUrgencia;
        }

        // CostoTotal = base + RecargoUrgencia x Peso
        public override double CalcularCostoTotal()
        {
            return base.CalcularCostoTotal() + (RecargoUrgencia * Peso);
        }

        public override string ToString()
        {
            return $"[Express] {base.ToString()}, Recargo Urgencia:
{RecargoUrgencia:F2} euros/kg, Costo Total: {CalcularCostoTotal():F2}
euros";
        }
    }
}
```

Ejercicio 4: Integración del Sistema de Consola (Polimorfismo con Colecciones)

La aplicación de consola debe incluir un menú con la siguiente funcionalidad:

1. **Crear Envío:** Permite al usuario seleccionar el tipo de paquete (Estándar o Express), introducir sus datos y añadir la instancia a una **colección interna de la clase base** `Envio`.
2. **Ver Costos Individuales:** Recorre la colección, mostrando los atributos del envío (usando `ToString()`) y su costo total (llamando a `CalcularCostoTotal()`).
3. **Calcular Ingreso Total:** Suma todos los costos totales de los envíos de la colección.
4. **Salir:** Termina la ejecución de la aplicación.

Código (Archivo **Program.cs**)

```
namespace IrisPerezAparicio_ASP_A2_10_LogiTrack_SA
{
    class Program
    {
        static List<Envio> envios = new List<Envio>();

        static int LeerEntero()
        {
            int num;
            while (!int.TryParse(Console.ReadLine(), out num))
                Console.Write("Introduce un número entero válido: ");
            return num;
        }

        static double LeerDouble()
        {
            double num;
            while (!double.TryParse(Console.ReadLine(), out num))
                Console.Write("Introduce un número válido: ");
            return num;
        }

        static void CrearEnvio()
        {
            Console.WriteLine("\nTIPO DE ENVÍO:");
            Console.WriteLine("1. Paquete Estándar");
            Console.WriteLine("2. Paquete Express");
            Console.WriteLine("3. Volver al menú");
            Console.Write("Seleccione una opción: ");
            int opcion = LeerEntero();

            if (opcion < 1 || opcion > 3)
            {
                Console.WriteLine("Opción no válida. Volviendo al
menú...");
                return;
            }

            if (opcion == 3)
            {
                Console.WriteLine("Volviendo al menú...");
                return;
            }
```

```
            Console.Write("\nDescripción: ");
            string descripcion = Console.ReadLine();
            Console.Write("Peso (kg): ");
            double peso = LeerDouble();

            switch (opcion)
            {
                case 1:
                    // Tarifa plana fija 10€, pero se mantiene
validación por coherencia general
                    envios.Add(new PaqueteEstandar(descripcion, peso,
10.0));
                    Console.WriteLine("Paquete Estándar creado
correctamente.\n");
                    break;

                case 2:
                    Console.Write("Recargo Urgencia (euros/kg): ");
                    double recargo = LeerDouble();
                    envios.Add(new PaqueteExpress(descripcion, peso,
recargo));
                    Console.WriteLine("Paquete Express creado
correctamente.\n");
                    break;
            }
        }

        static void VerCostosIndividuales()
        {
            Console.WriteLine("\nCOSTOS INDIVIDUALES:");
            if (envios.Count == 0)
            {
                Console.WriteLine("No hay envíos registrados.");
                return;
            }

            foreach (var envio in envios)
                Console.WriteLine(envio.ToString());
        }

        static void CalcularIngresoTotal()
        {
            double total = envios.Sum(e => e.CalcularCostoTotal());
            Console.WriteLine($"\nIngreso Total por Envíos: {total:F2}
euros");
```

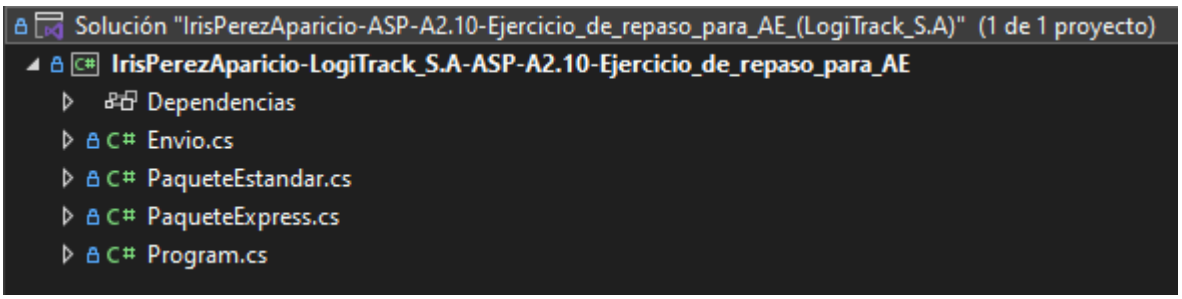
```
        }

        static void Main()
        {
            while (true)
            {
                Console.WriteLine("\n··· MENÚ ···");
                Console.WriteLine("1. Crear Envío");
                Console.WriteLine("2. Ver Costos Individuales");
                Console.WriteLine("3. Calcular Ingreso Total");
                Console.WriteLine("4. Salir");
                Console.Write("Seleccione una opción: ");

                int opcion = LeerEntero();

                switch (opcion)
                {
                    case 1: CrearEnvio(); break;
                    case 2: VerCostosIndividuales(); break;
                    case 3: CalcularIngresoTotal(); break;
                    case 4: return;
                    default: Console.WriteLine("Opción no válida
(1-4)."); break;
                }
            }
        }
    }
}
```

ESTRUCTURA FINAL DE ARCHIVOS:



Enlace al código en **GitHub**:

https://github.com/IrisCampusFP/Actividades_ASP_NET/tree/main/IrisPerezAparicio-ASP-A2.10-Ejercicio_de_repaso_para_AE_(LogiTrack_S.A)