

LogiTrack S.A

Sistema de Gestión de Envíos (LogiTrack)

Resultados de aprendizaje y criterios de evaluación	1
Enunciado	1
Ejercicio 1: Abstracción, Encapsulación y Validación (Clase Base Envio)	2
Ejercicio 2: Herencia y Primera Implementación (PaqueteEstandar)	2
Ejercicio 3: Atributos Especificos y Polimorfismo Final (PaqueteExpress)	2
Ejercicio 4: Integración del Sistema de Consola (Polimorfismo con Colecciones)	3
Recomendaciones para maximizar la reutilización	3

Resultados de aprendizaje y criterios de evaluación

RA1. Comprender y aplicar los fundamentos del lenguaje C# en el desarrollo de aplicaciones	4a	Se han identificado los fundamentos de la programación estructurada y el lenguaje C# (tipos de datos, variables, operadores).
	4b	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.

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).

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).

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 Envío**.
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.

Recomendaciones para maximizar la reutilización

- Los constructores de una clase se apoyan en el constructor de la clase base.
- Cuando sea posible, un método polimórfico de una clase se apoya en el mismo método polimórfico de la clase base.

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

internal class Program
{
    class Programa
    {
        // =====
        // COLECCIÓN (POLIMORFISMO)
        // =====
        // Requisito técnico: polimorfismo con colección de la clase
base
        static List<Envio> envios = new List<Envio>();

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

        // Requisito de calidad: Leer opción válida del menú
        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.");
    }
}

// Requisito de calidad: Lectura numérica con validación
// Si es negativo → se informa y se devuelve 0.0
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("Entrada inválida. Introduce un número válido.");
    }
}

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

```

```

// =====
// OPCIÓN 1: CREAR ENVÍO
// =====
// Ejercicio 4: creación polimórfica
static void CrearEnvio()
{
    Console.WriteLine("Tipos de envío disponibles: estandar /
express");
    Console.Write("Introduce el tipo de envío: ");
    string tipo = Console.ReadLine().ToLower();

    string descripcion = LeerTexto("Descripción del envío: ");
    double peso = LeerDoubleNoNegativo("Peso (kg): ");

    if (tipo.Equals("estandar"))
    {
        double tarifaPlana = LeerDoubleNoNegativo("Tarifa plana
(€): ");
        PaqueteEstandar pe = new PaqueteEstandar(descripcion,
peso, tarifaPlana);
        envios.Add(pe);
        Console.WriteLine("Paquete estándar creado
correctamente.");
    }
    else if (tipo.Equals("express"))
    {
        double recargo = LeerDoubleNoNegativo("Recargo urgencia
por kg (€): ");
        PaqueteExpress px = new PaqueteExpress(descripcion,
peso, recargo);
        envios.Add(px);
        Console.WriteLine("Paquete express creado
correctamente.");
    }
    else
    {
        Console.WriteLine("Tipo de envío no reconocido. No se ha
creado el envío.");
    }
}

// =====
// OPCIÓN 2: VER COSTOS INDIVIDUALES
// =====
static void VerCostosIndividuales()

```

```

    {
        if (envios.Count == 0)
        {
            Console.WriteLine("No hay envíos registrados.");
            return;
        }

        foreach (Envio e in envios)
        {
            // Polimorfismo: se ejecuta el ToString() y
            // CalculaCostoTotal() de la subclase real
            Console.WriteLine(e.ToString());
            Console.WriteLine($"Costo total:
{e.CalcularCostoTotal():0.00} €");

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

        // =====
        // OPCIÓN 3: CALCULAR INGRESO TOTAL
        // =====
        static void CalcularIngresoTotal()
        {
            double total = envios.Sum(e => e.CalcularCostoTotal());
            Console.WriteLine($"Ingreso total por envíos: {total:0.00}
€");
        }

        // =====
        // MENÚ
        // =====
        static void Menu()
        {
            Console.WriteLine();
            Console.WriteLine("===== LogiTrack S.A - Sistema de Gestión
de Envíos =====");
            Console.WriteLine("1- Crear envío");
            Console.WriteLine("2- Ver costos individuales");
            Console.WriteLine("3- Calcular ingreso total");
            Console.WriteLine("4- Salir");
        }

        // =====
        // MAIN
        // =====

```

```

public static void Main()
{
    while (true)
    {
        Menu();
        int opcion = LeerOpcion();

        switch (opcion)
        {
            case 1:
                CrearEnvio();
                break;
            case 2:
                VerCostosIndividuales();
                break;
            case 3:
                CalcularIngresoTotal();
                break;
            case 4:
                return; // salir del programa
        }
    }
}

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

// =====
// Ejercicio 1: Clase base Envio
// =====
// Requisito técnico: abstracción
public abstract class Envio
{
    // Propiedad automática (Requisito funcional)
    public string Descripcion { get; set; }

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

```

```

        // Propiedad solo Lectura: 2.0 € por kg (Requisito técnico)
        // Costo base = 2.0 * Peso
        public double CostoBase
        {
            get { return 2.0 * Peso; }
        }

        // Constructor con encapsulación + validación
        protected Envio(string descripcion, double peso)
        {
            Descripcion = string.IsNullOrEmpty(descripcion) ?
"Sin descripción" : descripcion;
            Peso = peso;
        }

        // Método polimórfico: se podrá sobrescribir
        public virtual double CalcularCostoTotal()
        {
            // Por defecto, el envío base solo cuesta el costo base
            return CostoBase;
        }

        // Método polimórfico ToString()
        public override string ToString()
        {
            return $"Envío -> Descripción: {Descripcion}, Peso:
{Peso:0.00} kg, Costo base: {CostoBase:0.00} €";
        }
    }

    // =====
    // Ejercicio 2: PaqueteEstandar
    // =====
    // Requisito técnico: herencia
    public class PaqueteEstandar : Envio
    {
        // Propiedad no automática con validación
        private double _tarifaPlana;
        public double TarifaPlana
        {
            get => _tarifaPlana;
            set => _tarifaPlana = value < 0 ? 0.0 : value;
        }

        // Constructor apoyado en el de la base
        public PaqueteEstandar(string descripcion, double peso,

```

```

double tarifaPlana)
    : base(descripcion, peso)
    {
        TarifaPlana = tarifaPlana;
    }

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

    public override string ToString()
    {
        return $"Paquete Estándar -> Descripción: {Descripcion},
Peso: {Peso:0.00} kg, " +
            $"Costo base: {CostoBase:0.00} €, Tarifa plana:
{TarifaPlana:0.00} €";
    }
}

// =====
// Ejercicio 3: PaqueteExpress
// =====
public class PaqueteExpress : Envio
{
    // Recargo por urgencia (por kg, según enunciado)
    private double _recargoUrgencia;
    public double RecargoUrgencia
    {
        get => _recargoUrgencia;
        set => _recargoUrgencia = value < 0 ? 0.0 : value;
    }

    // Constructor apoyado en la base
    public PaqueteExpress(string descripcion, double peso,
double recargoUrgencia)
        : base(descripcion, peso)
        {
            RecargoUrgencia = recargoUrgencia;
        }

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

```

```
Peso);  
    }  
  
    public override string ToString()  
    {  
        return $"Paquete Express -> Descripción: {Descripcion},  
Peso: {Peso:0.00} kg, " +  
            $"Costo base: {CostoBase:0.00} €, Recargo  
urgencia/kg: {RecargoUrgencia:0.00} €";  
    }  
}  
}
```

Consola

===== LogiTrack S.A - Sistema de Gestión de Envíos =====

- 1- Crear envío
- 2- Ver costos individuales
- 3- Calcular ingreso total
- 4- Salir

Introduce una opción entre 1 y 4: 1

Tipos de envío disponibles: estandar / express

Introduce el tipo de envío: estandar

Descripción del envío: correos

Peso (kg): 20

Tarifa plana (?): 30

Paquete estándar creado correctamente.

===== LogiTrack S.A - Sistema de Gestión de Envíos =====

- 1- Crear envío
- 2- Ver costos individuales
- 3- Calcular ingreso total
- 4- Salir

Introduce una opción entre 1 y 4: 1

Tipos de envío disponibles: estandar / express

Introduce el tipo de envío: express

Descripción del envío: amazon

Peso (kg): 10

Recargo urgencia por kg (?): 15

Paquete express creado correctamente.

===== LogiTrack S.A - Sistema de Gestión de Envíos =====

- ### 1- Crear envío

- 2- Ver costos individuales
- 3- Calcular ingreso total
- 4- Salir

Introduce una opción entre 1 y 4: 2

Paquete Estándar -> Descripción: correos, Peso: 20,00 kg, Costo base: 40,00 ?, Tarifa plana: 30,00 ?

Costo total: 70,00 ?

Paquete Express -> Descripción: amazon, Peso: 10,00 kg, Costo base: 20,00 ?, Recargo urgencia/kg: 15,00 ?

Costo total: 170,00 ?

===== LogiTrack S.A - Sistema de Gestión de Envíos =====

- 1- Crear envío
- 2- Ver costos individuales
- 3- Calcular ingreso total
- 4- Salir

Introduce una opción entre 1 y 4: 3

Ingreso total por envíos: 240,00 ?

===== LogiTrack S.A - Sistema de Gestión de Envíos =====

- 1- Crear envío
- 2- Ver costos individuales
- 3- Calcular ingreso total
- 4- Salir

Introduce una opción entre 1 y 4: 4

C:\repositorio\ASPNet\ASP-A2.10 Ejercicio de repaso para (LogiTrack S.A)\ASP-A2.10-LogiTrack S.A\ASP-A2.10-LogiTrack S.A\bin\Debug\net8.0\ASP-A2.10-LogiTrack S.A.exe (proceso 19004) 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. . .