TechSolutions S.L

Iris Pérez Aparicio

2º DAM

Campus FP Emprende Humanes

Sistema de Gestión de Personal (HRSystem)

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 **TechSolutions S.L.** es una consultora tecnológica que necesita modernizar su **Sistema de Recursos Humanos (HRSystem)** para calcular las nóminas de su personal de manera automatizada y flexible.

La empresa maneja tres tipos de personal que deben ser gestionados bajo una jerarquía común:

1. **Empleado Base:** Todos los empleados tienen un **Nombre** y un **Salario Base** mensual.
2. **Empleado Fijo:** Además del salario base, recibe un **Bono Anual**. Su nómina mensual se calcula prorrateando el bono: *Nómina mensual = Salario Base + Bono Anual / 12*
3. **Empleado Por Hora:** Además del salario base (por ejemplo, fijo para el seguro social), cobra una **Tarifa por Hora** multiplicada por las **Horas Trabajadas en el Mes**: *Nómina mensual = Salario Base + Tarifa por Hora x Horas Trabajadas en el mes*.

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

Ejercicios

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

Crea la clase base Empleado:

- Propiedades automáticas: Nombre.
- Propiedades no automáticas: SalarioBase.
- Constructor que inicialice las propiedades.
- Métodos polimórficos:
    - CalcularNomina.
    - ToString.

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

Código (Archivo **Empleado.cs**)

```
namespace IrisPerezAparicio_ASP_A_2_9_TechSolutions_SL
{
    abstract class Empleado
    {
        public string Nombre { get; set; }

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

        protected Empleado(string nombre, double salarioBase)
        {
            Nombre = nombre;
            SalarioBase = salarioBase;
        }

        public abstract double CalcularNomina();

        public override string ToString()
        {
            return $"Empleado: {Nombre}, Salario Base:
{SalarioBase:F2}";
        }
    }
}
```

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

Crea la clase heredada EmpleadoFijo:

- Hereda de Empleado
- Propiedades no automáticas: BonoAnual.
- Constructor que inicialice las propiedades.
- Métodos polimórficos:
    - CalcularNomina.
    - ToString.

Código (Archivo **EmpleadoFijo.cs**)

```
namespace IrisPerezAparicio_ASP_A_2_9_TechSolutions_SL
{
    // Requisito técnico de herencia
    class EmpleadoFijo : Empleado
    {
        // Propiedad no automática con encapsulación y validación
        private double _bonoAnual;
        public double BonoAnual
        {
            get => _bonoAnual;
            set => _bonoAnual = value < 0 ? 0.0 : value; // Validación:
si negativo, asigna 0.0
        }

        // Constructor que inicializa propiedades
        public EmpleadoFijo(string nombre, double salarioBase, double
bonoAnual)
            : base(nombre, salarioBase)
        {
            BonoAnual = bonoAnual;
        }

        // Implementación polimórfica del cálculo de nómina
        public override double CalcularNomina()
        {
            return SalarioBase + (BonoAnual / 12);
        }

        // Polimorfismo de representación
        public override string ToString()
        {
            return $"Empleado Fijo: {Nombre}, Salario Base:
{SalarioBase:F2}, Bono Anual: {BonoAnual:F2}, Nómina Mensual:
{CalcularNomina():F2}";
        }
    }
}
```

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

Crea la clase heredada EmpleadoPorHora:

- Hereda de Empleado
- Propiedades no automáticas: TarifaHora, HorasTrabajadasMes.
- Constructor que inicialice las propiedades.
- Métodos polimórficos:
    - CalcularNomina.
    - ToString.

Código (Archivo **EmpleadoPorHora.cs**)

```
namespace IrisPerezAparicio_ASP_A_2_9_TechSolutions_SL
{
    // Requisito técnico de herencia
    class EmpleadoPorHora : Empleado
    {
        // Propiedades no automáticas con encapsulación y validación
        private double _tarifaHora;
        public double TarifaHora
        {
            get => _tarifaHora;
            set => _tarifaHora = value < 0 ? 0.0 : value; // Validación:
si negativo, asigna 0.0
        }

        private double _horasTrabajadasMes;
        public double HorasTrabajadasMes
        {
            get => _horasTrabajadasMes;
            set => _horasTrabajadasMes = value < 0 ? 0.0 : value; //
Validación: si negativo, asigna 0.0
        }

        // Constructor que inicializa propiedades
        public EmpleadoPorHora(string nombre, double salarioBase, double
tarifaHora, double horasTrabajadasMes)
            : base(nombre, salarioBase)
        {
            TarifaHora = tarifaHora;
            HorasTrabajadasMes = horasTrabajadasMes;
```

```
        }

        // Implementación polimórfica del cálculo de nómina
        public override double CalcularNomina()
        {
            return SalarioBase + (TarifaHora * HorasTrabajadasMes);
        }

        // Polimorfismo de representación
        public override string ToString()
        {
            return $"Empleado por Hora: {Nombre}, Salario Base:
{SalarioBase:F2}, Tarifa por Hora: {TarifaHora:F2}, Horas Trabajadas:
{HorasTrabajadasMes:F2}, Nómina Mensual: {CalcularNomina():F2}";
        }
    }
}
```

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

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

1. **Contratar Empleado:** Permite al usuario seleccionar el tipo de empleado (Base, Fijo o Por Hora), introducir sus datos y añadir la instancia a una **colección interna de la clase base `Empleado`**.

   (Archivo **EmpleadoBase.cs**)

```
using IrisPerezAparicio_ASP_A_2_9_TechSolutions_SL;
using static System.Runtime.InteropServices.JavaScript.JSType;

class EmpleadoBase : Empleado
{
    public EmpleadoBase(string nombre, double salarioBase)
        : base(nombre, salarioBase) { }

    public override double CalcularNomina()
    {
        return SalarioBase;
    }
```

```
    public override string ToString()
    {
        return $"Empleado Base: {Nombre}, Salario Base:
{SalarioBase:F2}, Nómina: {CalcularNomina():F2}";
    }
}
```

2. **Ver Nóminas Individuales:** Recorre la colección, mostrando los atributos del empleado (usando `ToString()`) y su nómina mensual (llamando a `CalcularNomina()`).
3. **Calcular Coste Total de Nóminas:** Suma todas las nóminas mensuales de la colección.
4. **Salir:** Termina la ejecución de la aplicación.

Código (Archivo **Program.cs**)

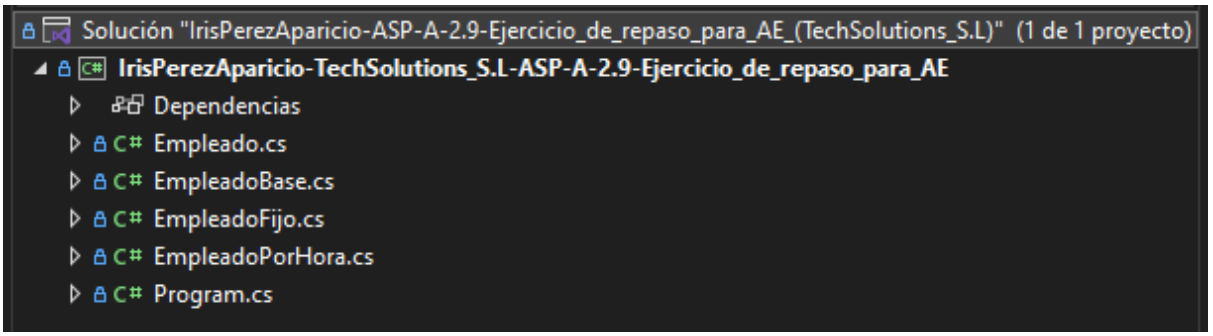
```
using System;
using System.Collections.Generic;
using System.Linq;

namespace IrisPerezAparicio_ASP_A_2_9_TechSolutions_SL
{
    class Program
    {
        // Colección polimórfica
        static List<Empleado> empleados = new List<Empleado>();

        // Métodos auxiliares para validación de entrada
        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 ContratarEmpleado()
        {
            Console.WriteLine("\nTIPO DE EMPLEADO:");
            Console.WriteLine("1. Empleado Base");
            Console.WriteLine("2. Empleado Fijo");
            Console.WriteLine("3. Empleado por Hora");
            Console.WriteLine("4. Volver al menú");
            Console.Write("Seleccione una opción: ");
            int opcion = LeerEntero();

            if (opcion < 1 || opcion > 4)
            {
                Console.WriteLine("Opción no válida. Volviendo al
menú...");
                return;
            }

            if (opcion == 4)
            {
                Console.WriteLine("Volviendo al menú...");
                return;
            }

            Console.Write("\nNombre: ");
            string nombre = Console.ReadLine();
            Console.Write("Salario Base: ");
            double salarioBase = LeerDouble();

            switch (opcion)
            {
                case 1:
                    empleados.Add(new EmpleadoBase(nombre,
salarioBase));
                    Console.WriteLine("Empleado Base contratado
correctamente.\n");
                    break;

                case 2:
                    Console.Write("Bono Anual: ");
                    double bono = LeerDouble();
                    empleados.Add(new EmpleadoFijo(nombre, salarioBase,
bono));
                    Console.WriteLine("Empleado Fijo contratado
correctamente.\n");
                    break;
```

```
                case 3:
                    Console.Write("Tarifa por Hora: ");
                    double tarifa = LeerDouble();
                    Console.Write("Horas Trabajadas: ");
                    double horas = LeerDouble();
                    empleados.Add(new EmpleadoPorHora(nombre,
salarioBase, tarifa, horas));
                    Console.WriteLine("Empleado por Hora contratado
correctamente.\n");
                    break;
            }
        }

        static void VerNominasIndividuales()
        {
            Console.WriteLine("\nNÓMINAS INDIVIDUALES:");
            if (empleados.Count == 0)
            {
                Console.WriteLine("No hay empleados registrados.");
                return;
            }

            foreach (var empleado in empleados)
                Console.WriteLine(empleado.ToString());
        }

        static void CalcularCosteTotalNominas()
        {
            double total = empleados.Sum(e => e.CalcularNomina());
            Console.WriteLine($"\nCoste Total de Nóminas: {total:F2}");
        }

        public static void Main()
        {
            while (true)
            {
                Console.WriteLine("\n··· MENÚ ···");
                Console.WriteLine("1. Contratar Empleado");
                Console.WriteLine("2. Ver Nóminas Individuales");
                Console.WriteLine("3. Calcular Coste Total de Nóminas");
                Console.WriteLine("4. Salir");
                Console.Write("Seleccione una opción: ");

                int opcion = LeerEntero();
```

```
                switch (opcion)
                {
                    case 1: ContratarEmpleado(); break;
                    case 2: VerNominasIndividuales(); break;
                    case 3: CalcularCosteTotalNominas(); break;
                    case 4: Console.WriteLine("Cerrando aplicación...");
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-A-2.9-Ejercicio_de_repaso_para_AE_(TechSolutions_S.L)