TechSolutions S.L

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.

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.

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.

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`**.
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.

Recomendaciones para maximizar la reutilización

- Los constructores de una clase se apoyan en el constructor de la clase base.
- Cuando sea posible, un método polimorfico 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
    {
        static List<Figura> figuras = new List<Figura>();


        static int LeerOpcion()
        {
            while (true)
            {
                Console.Write("Introduce una opcion entre 1 y 5: ");
                int opcion = 0;
                if (int.TryParse(Console.ReadLine(), out opcion))
                {
                    if (opcion >= 1 && opcion <= 5)
                    {
                        return opcion;
                    }
```

```
                }
            }
        }



        // Rquisito funcional
        static void CrearFigura()
        {
            Console.WriteLine("Elija un circulo, rectangulo o rombo: ");
            string figura = Console.ReadLine().ToLower();

            if (figura.Equals("circulo"))
            {
                Console.Write("Radio: ");
                double radio;
                if (double.TryParse(Console.ReadLine(), out radio))
                {
                    Circulo circulo = new Circulo();
                    circulo.Radio = radio;
                    figuras.Add(circulo);

                }
                else
                {
                    Console.WriteLine("No se ha creado");
                }
            } else if (figura.Equals("rectangulo"))
            {
                Console.Write("Altura: ");
                double altura;
                Console.Write("Base: ");
                double basec;
                if (!double.TryParse(Console.ReadLine(), out altura))
                {
                    Console.WriteLine("Altura inválida. No se ha
creado");
                    return;
                }
                Console.Write("Base: ");
                double baseRect;
                if (!double.TryParse(Console.ReadLine(), out baseRect))
                {
                    Console.WriteLine("Base inválida. No se ha creado");
                    return;
                }
```

```
                    Rectangulo rectangulo = new Rectangulo();
                    rectangulo.Altura = altura;
                    rectangulo.Base = baseRect;
                    figuras.Add(rectangulo);

                }
            }


            static void VerColeccion()
            {
                foreach (Figura figura in figuras)
                {
                    Console.WriteLine(figura.ToString());
                }
            }
            static void CalcularAreaTotal()
            {
                Console.WriteLine($"Area total = {figuras.Sum(f =>
f.CalcularArea())}");
            }

            static void CalcularPerimetroTotal()
            {
                Console.WriteLine($"Perimetro  total = {figuras.Sum(f =>
f.CalcularPerimetro())}");
            }

            static void Menu()
            {
                Console.WriteLine("1- Crear Figura");
                Console.WriteLine("2- Ver coleccion");
                Console.WriteLine("3- Calcular Area Total");
                Console.WriteLine("4- Calcular Perimetro Total");
                Console.WriteLine("5- CTerminar");
            }

            public static void Main()
            {
                while (true)
                {
                    Menu();
                    int opcion = LeerOpcion();
                    switch (opcion)
                    {
                        case 1:
```

```
                    CrearFigura();
                    break;
                case 2:
                    VerColeccion();
                    break;

                case 3:
                    CalcularAreaTotal();
                    break;

                case 4:
                    CalcularPerimetroTotal();
                    break;

                case 5:
                    break;

            }
        }


    }

    // Requisito Tecnico de Abstraccion
    public abstract class Figura
    {


        // Rquisito técnico de polimorfismo
        public abstract double CalcularArea();

        public abstract double CalcularPerimetro();


    }

    // Requisito Tecnico de Herencia
    public class Circulo : Figura
    {
        // Requisito funcional
        // Propiedades no automaticas
        private double _radio;

        // Requisito de calidad si valores negativos
        public double Radio { get => _radio; set => _radio = value
<= 0 ? 1 : value; }
```

```
            // Requisitos tecnicos de propiedad de solo lectura
            public double Area { get => Math.PI * Math.Pow(Radio, 2); }
            public double Perimetro { get => 2 * Math.PI * Radio; }

            public override double CalcularArea()
            {
                return Area;
            }

            public override double CalcularPerimetro()
            {
                return Perimetro;
            }

            // Requisito funcional ver coleccion
            public override string ToString()
            {
                return $"Circulo de radio {Radio} con área {Area} y
perímetro {Perimetro}";
            }
        }

        public class Rectangulo : Figura
        {
            // Requisito funcional
            // Propiedades no automaticas
            private double _base;
            private double _altura;

            // Requisito de calidad si valores negativos
            public double Base { get => _base; set => _base = value <= 0
? 1 : value; }
            public double Altura { get => _altura; set => _altura =
value <= 0 ? 1 : value; }

            public double Area { get => Base * Altura; }
            public double Perimetro { get => 2 * (Base + Altura); }


            public override double CalcularArea()
            {
                return Area;
            }

            public override double CalcularPerimetro()
```

```
            {
                return Perimetro;
            }

            // Requisito funcional ver coleccion
            public override string ToString()
            {
                return $"Rectangulo de base {Base} y altura {Altura} con
área {Area} y perímetro {Perimetro}";
            }
        }
        public class Rombo : Figura
        {
            // Requisito funcional
            // Propiedades no automaticas
            private double _diagonalMayor;
            private double _diagonalMenor;

            // Requisito de calidad si valores negativos
            public double DiagonalMayor { get => _diagonalMayor; set =>
_diagonalMayor = value <= 0 ? 1 : value; }
            public double DiagonalMenor { get => _diagonalMenor; set =>
_diagonalMenor = value <= 0 ? 1 : value; }

            public double Area { get => (DiagonalMayor * DiagonalMenor)
/ 2; }
            public double Perimetro { get => 2 *
Math.Sqrt(Math.Pow(DiagonalMayor, 2) + Math.Pow(DiagonalMenor, 2)); }


            public override double CalcularArea()
            {
                return Area;
            }

            public override double CalcularPerimetro()
            {
                return Perimetro;
            }

            // Requisito funcional ver coleccion
            public override string ToString()
            {
                return $"Rombo de DiagonalMayor {DiagonalMayor} y
DiagonalMenor {DiagonalMenor} con área {Area} y perímetro {Perimetro}";
            }
```

```
        }


    }
}
```

Consola

=== TechSolutions S.L. - HRSystem ===
1- Contratar empleado
2- Ver nóminas individuales
3- Calcular coste total de nóminas
4- Salir
Introduce una opción entre 1 y 4: 1
Tipos de empleado disponibles: base / fijo / hora
Introduce el tipo de empleado: base
Nombre: iker
Salario base mensual: 1600
Empleado base contratado correctamente.

=== TechSolutions S.L. - HRSystem ===
1- Contratar empleado
2- Ver nóminas individuales
3- Calcular coste total de nóminas
4- Salir
Introduce una opción entre 1 y 4: 1
Tipos de empleado disponibles: base / fijo / hora
Introduce el tipo de empleado: fijo
Nombre: ikerazo
Salario base mensual: 2100
Bono anual: 1200
Empleado fijo contratado correctamente.

=== TechSolutions S.L. - HRSystem ===
1- Contratar empleado
2- Ver nóminas individuales
3- Calcular coste total de nóminas
4- Salir
Introduce una opción entre 1 y 4: 1
Tipos de empleado disponibles: base / fijo / hora

Introduce el tipo de empleado: hora
Nombre: ikerin
Salario base mensual: 1100
Tarifa por hora: 21
Horas trabajadas en el mes: 160
Empleado por hora contratado correctamente.

=== TechSolutions S.L. - HRSystem ===
1- Contratar empleado
2- Ver nóminas individuales
3- Calcular coste total de nóminas
4- Salir
Introduce una opción entre 1 y 4: 2
Empleado Base -> Nombre: iker, Salario base: 1600,00
Nómina mensual: 1600,00

Empleado Fijo -> Nombre: ikerazo, Salario base: 2100,00, Bono anual: 1200,00
Nómina mensual: 2200,00

Empleado Por Hora -> Nombre: ikerin, Salario base: 1100,00, Tarifa/hora: 21,00, Horas mes: 160,00
Nómina mensual: 4460,00

=== TechSolutions S.L. - HRSystem ===
1- Contratar empleado
2- Ver nóminas individuales
3- Calcular coste total de nóminas
4- Salir
Introduce una opción entre 1 y 4: 3
Coste total de nóminas del mes: 8260,00

=== TechSolutions S.L. - HRSystem ===
1- Contratar empleado
2- Ver nóminas individuales
3- Calcular coste total de nóminas
4- Salir
Introduce una opción entre 1 y 4: 4