

Sistema básico de diseño geométrico



Iris Pérez Aparicio
2º DAM
Campus FP Emprende Humanes

ÍNDICE

Resultados de aprendizaje y criterios de evaluación	1
Contexto	1
Requisitos funcionales	1
Requisitos técnicos	2
Requisitos de calidad	2
Requisitos de entrega	2
Rúbrica	4
Anexos	6
Cálculo de áreas y perímetros de figuras geométricas	6
Funciones matemáticas de C# necesarias para los cálculos de áreas y perímetros.	6

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.

Contexto

La empresa GeoPlanner S.A. necesita un prototipo simple para que sus ingenieros puedan definir figuras geométricas y calcular automáticamente su área y perímetro. Se te solicita el desarrollo de una sencilla aplicación de consola, interactiva y en C#.

Requisitos funcionales

Se van a utilizar las figuras geométricas que se describen a continuación junto con sus atributos (valores decimales):

```
// Requisito técnico de abstracción
abstract class Figura
{
    // Requisito técnico de polimorfismo
    public abstract double CalcularArea();
    // Requisito técnico de polimorfismo
    public abstract double CalcularPerimetro();

    protected double _area;
    protected double _perimetro;
}
```

- Círculo, definido por su radio.

```
// Requisito técnico de herencia
class Circulo : Figura
{
    // Requisito funcional
    // Propiedad no automática
    private double _radio;
    // Requisito de calidad si valores negativos
    public double Radio { get => _radio; set => _radio = value <= 0 ? 1 : value; }

    // Requisito técnico de propiedades de sólo lectura.
    public double Area { get => Math.PI * Math.Pow(Radio, 2); }
    // Requisito técnico de propiedades de sólo lectura.
    public double Perimetro { get => 2 * Math.PI * Radio; }
```

```

// Requisito técnico de polimorfismo
public override double CalcularArea()
{
    _area = Area;
    return _area;
}

// Requisito técnico de polimorfismo
public override double CalcularPerimetro()
{
    _perimetro = Perimetro;
    return Perimetro;
}

// Requisito funcional ver colección
public override string ToString()
{
    return $"Circulo de Radio {Radio:F2} con área {Area:F2} y
perímetro {Perimetro:F2}";
    // (F2 hace que se muestren solo 2 decimales del double)
}
}

```

- Rectángulo, definido por su base y su altura.

```

// Requisito técnico de herencia
class Rectangulo : Figura {
    // Requisito funcional
    // Propiedad no automática
    private double _base;
    // Requisito de calidad si valores negativos
    public double Base { get => _base; set => _base = value <= 0 ? 1 :
value; }

    // Requisito funcional
    // Propiedad no automática
    private double _altura;
    // Requisito de calidad si valores negativos
    public double Altura { get => _altura; set => _altura = value <= 0 ?
1 : value; }

    // Requisito técnico de propiedades de sólo lectura.
    public double Area { get => Base * Altura; }
}

```

```

// Requisito técnico de propiedades de sólo lectura.
public double Perimetro { get => 2 * (Base + Altura); }
// Requisito técnico de polimorfismo
public override double CalcularArea()
{
    _area = Area;
    return Area;
}

// Requisito técnico de polimorfismo
public override double CalcularPerimetro()
{
    _perimetro = Perimetro;
    return Perimetro;
}
// Requisito funcional ver colección
public override string ToString()
{
    return $"Rectángulo de Base {Base:F2} y Altura {Altura:F2} con
    área {Area} y perímetro {Perimetro:F2}";
}
}

```

- Rombo, definido por su diagonal mayor y su diagonal menor.

```

// Requisito técnico de herencia
class Rombo : Figura
{
    // Requisito funcional
    // Propiedad no automática
    private double _diagonalMayor;
    // Requisito de calidad si valores negativos
    public double DiagonalMayor { get => _diagonalMayor; set =>
    _diagonalMayor = value <= 0 ? 1 : value; }
    // Requisito funcional
    // Propiedad no automática
    private double _diagonalMenor;
    // Requisito de calidad si valores negativos
    public double DiagonalMenor { get => _diagonalMenor; set =>
    _diagonalMenor = value <= 0 ? 1 : value; }
}

```

```

// Requisito técnico de propiedades de sólo lectura.
public double Area { get => DiagonalMayor * DiagonalMenor / 2; }
// Requisito técnico de propiedades de sólo lectura.
public double Perimetro { get => 2 *
Math.Sqrt(Math.Pow(DiagonalMayor, 2) * Math.Pow(DiagonalMenor, 2)); }

public override double CalcularArea()
{
    _area = Area;
    return Area;
}

// Requisito técnico de polimorfismo
public override double CalcularPerimetro()
{
    _perimetro = Perimetro;
    return Perimetro;
}

// Requisito funcional ver colección
public override string ToString()
{
    return $"Rombo de Diagonal Mayor {DiagonalMayor:F2} y Diagonal
Menor {DiagonalMenor:F2} con área {Area:F2} y perímetro {Perimetro:F2}";
}
}

```

Para cada figura geométrica se necesita calcular su área y su perímetro.

La aplicación de consola tendrá un menú con las siguientes opciones:

1. Crear una figura (círculo, rectángulo o rombo). La figura creada se añadirá a una colección interna de figuras.
2. Ver colección: mostrará todas las figuras de la colección interna de figuras. Para cada figura mostrará:
 - a. Tipo de figura.
 - b. Atributos de la figura.
 - c. Área.
 - d. Perímetro.

3. Calcular área total: muestra la suma de todas las áreas de las figuras de la colección interna.
4. Calcular perímetro total: muestra la suma de todos los perímetros de las figuras de la colección interna.
5. Terminar.

```
class Programa
{
    static List<Figura> figuras = new List<Figura>();
    static int LeerOpcion()
    {
        int opcion = 0;
        Console.Write("Seleccione una opción: ");
        bool valido = int.TryParse(Console.ReadLine(), out opcion);

        while (!valido)
        {
            Console.Write("Por favor, introduce un número entero: ");
            valido = int.TryParse(Console.ReadLine(), out opcion);
        }
        return opcion;
    }
    static double LeerDouble()
    {
        double num = 0;
        bool valido = double.TryParse(Console.ReadLine(), out num);

        while (!valido)
        {
            Console.WriteLine("Por favor, introduce un número: ");
            valido = double.TryParse(Console.ReadLine(), out num);
        }
        return num;
    }

    // Requisito funcional
    static void CrearFigura()
    {
        Console.WriteLine("\nElija una figura:\n1- Circulo\n2- Rectangulo\n3- Rombo\n4- Volver al menu");
        int opcion = LeerOpcion();
    }
}
```

```

switch (opcion)
{
    case 1:
        Console.WriteLine("\nCIRCULO");
        Console.Write("Radio: ");
        double radio = LeerDouble();

        Circulo circulo = new Circulo();
        circulo.Radio = radio;
        figuras.Add(circulo);

        Console.WriteLine("Círculo creado correctamente.\n(" +
circulo.ToString() + ")");
        break;
    case 2:
        Console.WriteLine("\nRECTANGULO");
        Console.Write("Base: ");
        double baseRect = LeerDouble();
        Console.Write("Altura: ");
        double alturaRect = LeerDouble();

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

        Console.WriteLine("Rectángulo creado correctamente.\n(" +
+ rectangulo.ToString() + ")");
        break;
    case 3:
        Console.WriteLine("\nROMBO");
        Console.Write("Diagonal Mayor: ");
        double diagonalMayor = LeerDouble();
        Console.Write("Diagonal Menor: ");
        double diagonalMenor = LeerDouble();

        Rombo rombo = new Rombo();
        rombo.DiagonalMayor = diagonalMayor;
        rombo.DiagonalMenor = diagonalMenor;
        figuras.Add(rombo);

        Console.WriteLine("Rombo creado correctamente.\n(" +
rombo.ToString() + ")");
        break;
    case 4:
        Console.WriteLine("Volviendo al menú...");
        break;
}

```

```

        default:
            Console.WriteLine("Opción no válida. Volviendo al
menú...");
            break;
        }
    }
    // Requisito funcional
    static void VerColeccion()
    {
        Console.WriteLine("\nCOLECCION DE FIGURAS:");
        figuras.ForEach(figura => Console.WriteLine(figura));
    }
    // Requisito funcional
    static void CalcularAreaTotal()
    {
        Console.WriteLine($"Area total = {figuras.Sum(figura =>
figura.CalcularArea()):F2}");
    }
    // Requisito funcional
    static void CalcularPerimetroTotal()
    {
        Console.WriteLine($"Perimetro total = {figuras.Sum(figura =>
figura.CalcularPerimetro()):F2}");
    }

    public static void Main()
    {
        while (true)
        {
            // Requisito funcional
            Console.WriteLine("\n... MENU ...");
            Console.WriteLine("1.- Crear Figura");
            Console.WriteLine("2.- Ver colección");
            Console.WriteLine("3.- Calcular Área Total");
            Console.WriteLine("4.- Calcular Perímetro Total");
            Console.WriteLine("5.- Terminar");

            int opcion = LeerOpcion();

            switch (opcion)
            {
                case 1: CrearFigura(); break;
                case 2: VerColeccion(); break;
                case 3: CalcularAreaTotal(); break;
                case 4: CalcularPerimetroTotal(); break;
                case 5: Console.WriteLine("Cerrando programa...");
            }
        }
    }
}
return;

```

```

                        default: Console.WriteLine("Opción no válida (introduce
un número del 1 al 5)"); break;
                    }
                }
            }
        }
    }
}

```

Requisitos técnicos

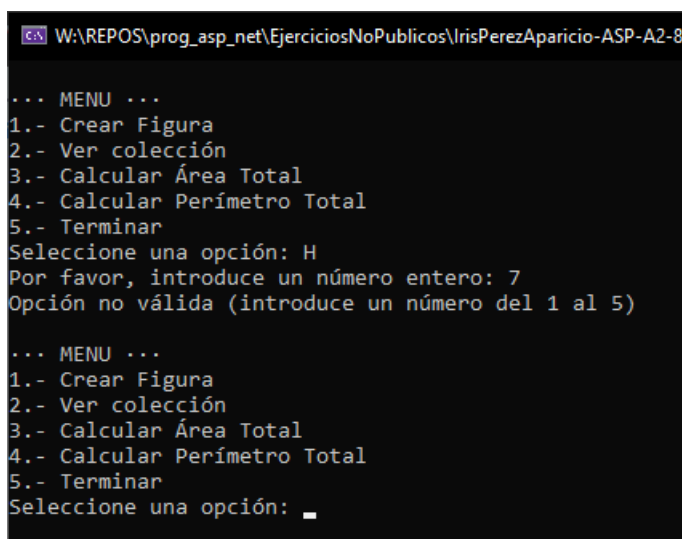
La implementación deberá hacer uso de:

- Abstracción.
- Herencia.
- Encapsulación.class
- Polimorfismo.
- Propiedades no automáticas.
- Propiedades de sólo lectura.

Requisitos de calidad

La aplicación deberá dar una respuestas controlada a al menos a estas situaciones:

- Se introduce una opción de menú no válida.



```

W:\REPOS\prog_asp_net\EjerciciosNoPublicos\IrisPerezAparicio-ASP-A2-8
... MENU ...
1.- Crear Figura
2.- Ver colección
3.- Calcular Área Total
4.- Calcular Perímetro Total
5.- Terminar
Seleccione una opción: H
Por favor, introduce un número entero: 7
Opción no válida (introduce un número del 1 al 5)
... MENU ...
1.- Crear Figura
2.- Ver colección
3.- Calcular Área Total
4.- Calcular Perímetro Total
5.- Terminar
Seleccione una opción: _

```

- Se asigna un valor negativo a un atributo de una figura geométrica. En este caso se asignará el valor 1.0 al atributo en cuestión

```
CIRCULO  
Radio: -5  
Círculo creado correctamente.  
(Círculo de Radio 1,00 con área 3,14 y perímetro 6,28)
```

Código C# con comentarios no triviales, abundantes y explicativos.

Encuentra en [GitHub](#) el código completo para descargarlo y probarlo en ejecución:

https://github.com/IrisCampusFP/Actividades_ASP_NET/tree/main/IrisPerezAparicio-ASP-A2-8-Simulacro%20de%20actividad%20evaluable