**Fundamentos de Interfaces en C#**

1. **¿Qué es una interfaz?**

Una interface en C# define un contrato, especifica qué métodos, propiedades o eventos debe implementar una clase, pero no provee implementación (salvo métodos *default* desde C# 8.0).

1. **Características**

* Solo define **qué se debe hacer,** no **cómo se hace.**
* Una clase puede implementar **múltiples interfaces** (polimorfismo por contrato).
* No se pueden crear instancias de interfaces.
* Generalmente se nombran con el prefijo I (ILogger, IRepository).

1. **Ejemplos:**

En este ejemplo creamos una interfaz IShape, que representa el contrato que debe cumplir la clase que implemente esta interfaz. El método a implementar es el cálculo del área de una figura geométrica.

* 1. **Crea la siguiente interfaz y las clases que la implementan:**

namespace \_01\_Interface01

{

public interface IShape

{

double CalculateArea();

}

}

namespace \_01\_Interface01

{

public class Triangle : IShape

{

private double \_base;

private double \_height;

public Triangle(double triangleBase, double height)

=> (\_base, \_height) = (triangleBase, height);

public double CalculateArea()

{

return (\_base \* \_height) / 2;

}

}

}

namespace \_01\_Interface01

{

public class Rectangle : IShape

{

private double \_width;

private double \_height;

public Rectangle(double width, double height)

=> (\_width, \_height) = (width, height);

public double CalculateArea()

{

return \_width \* \_height;

}

}

}

namespace \_01\_Interface01

{

internal class Circle : IShape

{

private double \_radius;

public Circle(double radious) => \_radius = radious;

public double CalculateArea()

{

return Math.PI \* Math.Pow(\_radius, 2);

}

}

}

using \_01\_Interface01;

namespace GeometryExample

{

class Program

{

static void Main(string[] args)

{

List<IShape> shapes = new List<IShape>

{

new Triangle(10, 5),

new Circle(7),

new Rectangle(8, 6)

};

foreach (var shape in shapes)

{

Console.WriteLine($"Área de la figura: {shape.GetType().Name} {shape.CalculateArea():F2}");

}

}

}

}

* 1. **Explicación rápida:**
* IShape es la **interfaz**, que obliga a todas las figuras a tener el método CalculateArea().
* Triangle, Circle y Rectangle **implementan la interfaz**.
* En el Main, se usan **polimórficamente:** todas las figuras se manejan como IShape.

1. **Interfaces con métodos genéricos**

A continuación mostraré un ejemplo de interfaces con métodos genéricos, para ello realizaré un sencillo programa que representa un repositorio.

* 1. **Creamos la interfaz y las clases que la implementan.**

namespace \_02\_Interfaces02

{

public interface IRepository<T>

{

void Add(T entity);

void Remove(T entity);

IEnumerable<T> GetAll();

}

}

namespace \_02\_Interfaces02

{

public class DBRepository<T> : IRepository<T>

{

private List<T> \_list = new List<T>();

public void Add(T entity)

{

\_list.Add(entity);

Console.WriteLine("Elemento agregado");

}

public IEnumerable<T> GetAll()

{

return \_list;

}

public void Remove(T entity)

{

\_list.Remove(entity);

Console.WriteLine("Elemento removido");

}

}

}

namespace \_02\_Interfaces02

{

public class Employee

{

public int Id { get; set; }

public string Name { get; set; }

public override string ToString()

{

return $"Empleado: {Id} - {Name}";

}

}

}

using \_02\_Interfaces02;

public class Program

{

public static void Main(string[] args)

{

IRepository<Employee> db = new DBRepository<Employee>();

Employee employee1 = new Employee { Id = 1, Name = "Juancho Lopez"};

Employee employee2 = new Employee { Id = 2, Name = "Jacob Martinez"};

Employee employee3 = new Employee { Id = 3, Name = "Juli Martinez"};

db.Add(employee1);

db.Add(employee2);

db.Add(employee3);

foreach (var employee in db.GetAll())

{

Console.WriteLine(employee);

}

}

}

* 1. **Explicación**

1. **IRepository<T>**
   * Una interfaz genérica que define operaciones básicas: Add, Remove y GetAll.
   * El tipo T puede ser cualquier clase (Empleado, Producto, Cliente, etc.).
2. **DBRepository<T>**
   * Implementa la interfaz utilizando una List<T> en memoria.
3. **Employee**
   * Una clase de ejemplo que usamos como tipo concreto para el repositorio.
4. **Main**
   * Creamos un repositorio de Employee y lo usamos para agregar, listar y eliminar empleados.