**Fundamentos de Interfaces en C#**

1. **¿Qué es la herencia?**

Es un mecanismo de la Programación Orientada a Objetos mediante el cual una clase (clase derivada o subclase) puede reutilizar, extender o modificar el comportamiento y los atributos definidos en otra clase (clase base o superclase). Este concepto permite establecer relaciones jerárquicas entre clases, fomentando la reutilización de código, la extensibilidad del sistema y el principio de abstracción.

1. **Características de la Herencia**

* Jerarquía de clases
  + Define una relación entre clases del tipo **“es-un” (is-a)**.
  + Una subclase hereda las propiedades y comportamientos de una superclase.
* Reutilización de código
  + Permite aprovechar atributos y métodos ya implementados en la clase base sin necesidad de duplicarlos.
* Extensibilidad
  + Las subclases pueden **agregar nuevas propiedades y métodos** o **sobrescribir** (override) los heredados para adaptar el comportamiento.
* Polimorfismo
  + Una referencia de tipo base puede apuntar a objetos de clases derivadas, lo que habilita el uso de métodos sobrescritos dinámicamente.
* Herencia simple y múltiple
  + En lenguajes como **C# y Java**, una clase solo puede heredar de **una clase base** (herencia simple), pero puede implementar múltiples interfaces.
  + En C++ sí es posible la herencia múltiple directa entre clases.
* Visibilidad de miembros
  + Solo los miembros **públicos** y **protegidos** de la clase base son accesibles en la clase derivada.
  + Los miembros **privados** no se heredan directamente, pero se pueden acceder mediante propiedades o métodos públicos/protegidos.
* Constructores y destructores
  + Los constructores de la clase base no se heredan, pero pueden ser invocados explícitamente desde la subclase usando palabras clave como base (C#) o super (Java).
* Especialización
  + La herencia permite que las subclases **especialicen** el comportamiento de la clase base adaptándolo a necesidades concretas.

1. **Ejemplos:**
   1. **Enunciado del Ejercicio**

Se desea implementar un sistema de facturación de servicios públicos domiciliarios utilizando el concepto de herencia en programación orientada a objetos (POO).

El sistema debe permitir calcular los cargos de consumo, cargos fijos y subsidios de diferentes servicios públicos como agua, electricidad y gas, partiendo de una clase base llamada Invoice (Factura).

Cada tipo de servicio tendrá reglas particulares para el cálculo de sus cobros:

1. Agua (WaterBill)
   * El cargo fijo depende del rango de consumo.
   * Si el consumo supera los 100 m³, se incluye un cálculo adicional de alcantarillado (CalcSewer).
   * Aplica un subsidio fijo de $2000.
2. Electricidad (ElectricityBill)
   * El cargo fijo se calcula como: el 20% del (consumo X 200).
   * No aplica subsidio ni cargos adicionales.
3. Gas (Gas)
   * Si el consumo es 0, no genera cargos.
   * Para consumos entre 1 y 50 unidades, se cobra $1.25 por unidad.
   * Para consumos mayores a 50 unidades, se cobra $2.50 por unidad.

El objetivo del ejercicio es:

* Modelar la clase base Invoice con atributos y métodos generales aplicables a cualquier servicio.
* Implementar clases derivadas (WaterBill, ElectricityBill, Gas) que sobrescriban (override) el método CalcFixedCharge para aplicar sus reglas específicas.
* Demostrar el uso del polimorfismo, permitiendo que cada clase hija defina su propio comportamiento de cálculo sobre una misma interfaz (Invoice).

namespace Herencia01

{

public abstract class Invoice

{

private int \_initialReading;

private int \_finalReading;

protected Invoice(int initialReading, int finalReading)

=> (\_initialReading, \_finalReading) = (initialReading, finalReading);

public int CalcConsumption()

{

return \_finalReading - \_initialReading;

}

public virtual decimal Subsidy()

{

int consumption = CalcConsumption();

if(consumption <= 100)

{

return consumption \* 10.0m;

}

else

{

return 0.0m;

}

}

public abstract decimal CalcFixedCharge(int consumption);

}

}

namespace Herencia01

{

public class WaterBill : Invoice

{

public WaterBill(int initialReading, int finalReading)

: base(initialReading, finalReading) {}

public override decimal CalcFixedCharge(int consumption)

{

if(consumption == 0)

return 0m;

else if(consumption > 0 && consumption <= 100)

return 50m;

else if(consumption > 100 && consumption <= 200)

return 80m;

else

return 95m;

}

public decimal CalcSewer(int consumption, decimal price)

{

if (consumption > 100)

return (consumption \* price) \* 0.5m;

else

return price;

}

public override decimal Subsidy()

{

return 2000.00m;

}

}

}

namespace Herencia01

{

public class ElectricityBill : Invoice

{

public ElectricityBill(int initialReading, int finalReading)

: base(initialReading, finalReading) { }

public override decimal CalcFixedCharge(int consumption)

{

return (consumption \* 200) \* 0.2m;

}

}

}

namespace Herencia01

{

public class Gas : Invoice

{

public Gas(int initialReading, int finalReading)

: base(initialReading, finalReading) { }

public override decimal CalcFixedCharge(int consumption)

{

if(consumption == 0)

return 0m;

else if(consumption >= 1 && consumption <= 50)

return consumption \* 1.25m;

else return consumption \* 2.5m;

}

}

}

using Herencia01;

using System.Globalization;

using System.Threading;

class Program

{

static void Main(string[] args)

{

// Forzar a cultura Colombiana COP

Thread.CurrentThread.CurrentCulture = new CultureInfo("es-CO");

Thread.CurrentThread.CurrentUICulture = new CultureInfo("es-CO");

// Agua

Console.WriteLine("-= FACTURA DE AGUA =-");

int initialReadingWater = ReadInteger("Lectura anterior de agua: ");

int finalReadingWater = ReadInteger("Lectura actual de agua: ");

WaterBill waterBill = new WaterBill(initialReadingWater, finalReadingWater);

ShowInvoice(waterBill, 100);

// Electricidad

Console.WriteLine("\n-= FACTURA DE ELECTRICIDAD =-");

int initialReadingElectricity = ReadInteger("Lectura anterior de electricidad: ");

int finalReadingElectricity = ReadInteger("Lectura actual de electricidad: ");

ElectricityBill electricityBill = new ElectricityBill(initialReadingElectricity, finalReadingElectricity);

ShowInvoice(electricityBill, 200);

// Gas

Console.WriteLine("\n-= FACTURA DE GAS =-");

int initialReadingGas = ReadInteger("Lectura anterior de gas: ");

int finalReadingGas = ReadInteger("Lectura actual de gas: ");

Gas gasBill = new Gas(initialReadingGas, finalReadingGas);

ShowInvoice(gasBill, 150);

}

// Método auxiliar genérico para leer y validar ingreso de lecturas

static int ReadInteger(string message)

{

int value;

while (true)

{

Console.Write(message);

if (int.TryParse(Console.ReadLine(), out value))

return value;

Console.WriteLine("Error: digite un valor numérico válido.");

}

}

// Método auxiliar para mostrar la factura genérica

static void ShowInvoice(Invoice invoice, decimal unitPrice)

{

int consumption = invoice.CalcConsumption();

decimal fixedCharge = invoice.CalcFixedCharge(consumption);

decimal subsidy = invoice.Subsidy();

decimal total = (consumption \* unitPrice) - (fixedCharge + subsidy);

Console.WriteLine($"Consumo: {consumption}");

Console.WriteLine($"Cargo fijo: {fixedCharge:C}");

Console.WriteLine($"Subsidio: {subsidy:C}");

Console.WriteLine($"Total a pagar: {total:C}");

}

}