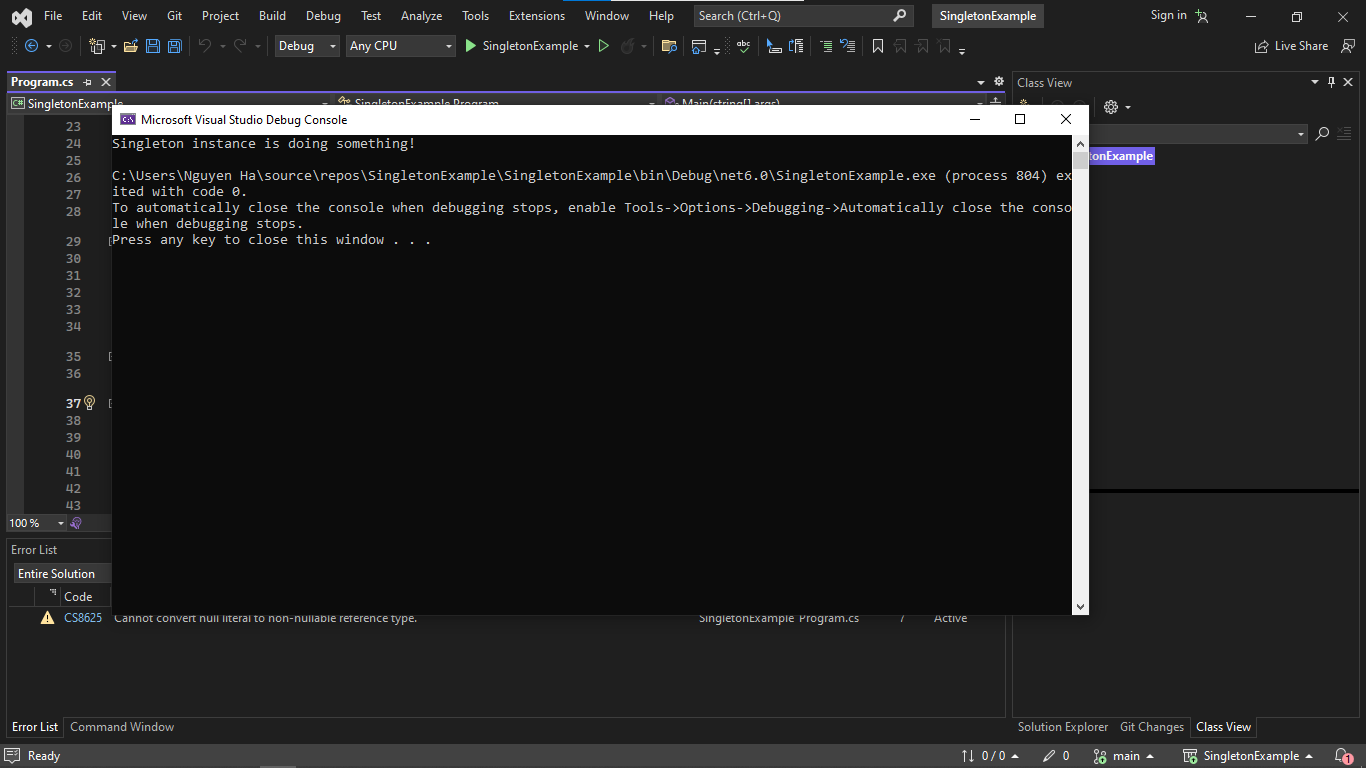
Họ và tên: Hồ Nguyên Hà . Mã SV: 20it043

Bài Tập Tuần 4

**SingletonEx:**



using System;

namespace SingletonExample

{

public sealed class Singleton

{

// Biến tĩnh để giữ thể hiện duy nhất của lớp Singleton

private static Singleton instance = null;

// Đối tượng để đồng bộ hóa việc tạo thể hiện

private static readonly object padlock = new object();

// Hàm tạo riêng tư để ngăn việc tạo thể hiện từ bên ngoài lớp

Singleton()

{

}

// Phương thức getter cho thể hiện duy nhất của lớp Singleton

public static Singleton Instance

{

get

{

// Đồng bộ hóa để ngăn việc tạo nhiều thể hiện trong các luồng khác nhau

lock (padlock)

{

// Nếu chưa có thể hiện nào, tạo một thể hiện mới

if (instance == null)

{

instance = new Singleton();

}

return instance;

}

}

}

// Một phương thức mẫu để minh họa việc sử dụng thể hiện Singleton

public void DoSomething()

{

Console.WriteLine("Singleton instance is doing something!");

}

}

class Program

{

static void Main(string[] args)

{

// Sử dụng thể hiện duy nhất của lớp Singleton để gọi phương thức DoSomething

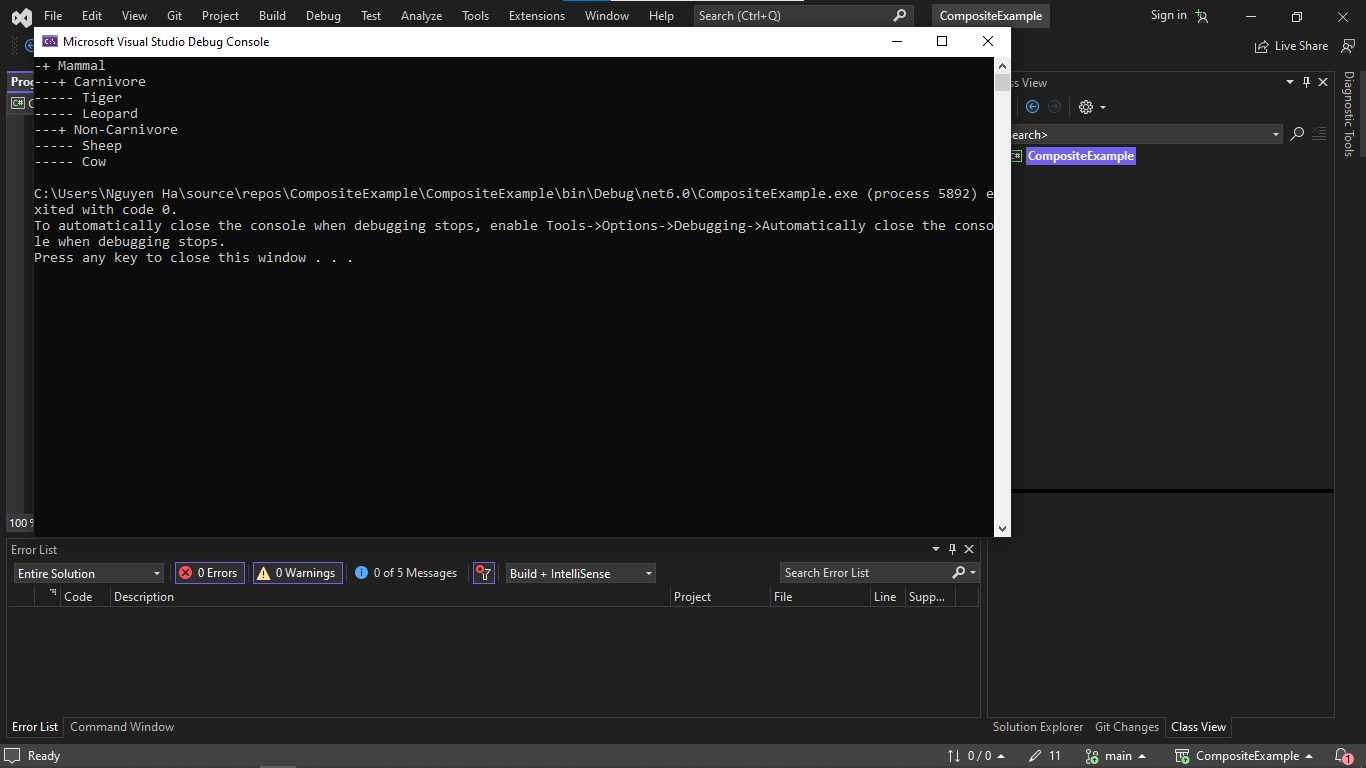
Singleton.Instance.DoSomething();

}

}

}

**CompositeEx:**



// Lớp trừu tượng Animal đại diện cho một đối tượng động vật

public abstract class Animal

{

// Thuộc tính Name lưu trữ tên của đối tượng động vật

protected string Name;

// Hàm khởi tạo của lớp Animal, nhận vào một tham số là tên của đối tượng động vật

protected Animal(string name)

{

Name = name;

}

// Phương thức ảo Add cho phép thêm một đối tượng Animal vào danh sách con của một đối tượng phức hợp

public virtual void Add(Animal animal)

{

throw new NotImplementedException();

}

// Phương thức ảo PrintTree cho phép in ra cấu trúc cây của đối tượng

public virtual void PrintTree(int indent)

{

throw new NotImplementedException();

}

}

// Lớp Species kế thừa từ lớp Animal, đại diện cho một loài động vật đơn giản

public class Species : Animal

{

// Hàm khởi tạo của lớp Species, gọi đến hàm khởi tạo của lớp cơ sở và truyền vào tham số name

public Species(string name) : base(name) { }

// Ghi đè phương thức PrintTree để in ra tên của loài động vật

public override void PrintTree(int indent)

{

Console.WriteLine(new String('-', indent) + " " + Name);

}

}

// Lớp SpeciesComposite kế thừa từ lớp Animal, đại diện cho một nhóm loài động vật phức hợp

public class SpeciesComposite : Animal

{

// Danh sách lưu trữ các đối tượng con của nhóm loài động vật này

private readonly List<Animal> \_animals = new List<Animal>();

// Hàm khởi tạo của lớp SpeciesComposite, gọi đến hàm khởi tạo của lớp cơ sở và truyền vào tham số name

public SpeciesComposite(string name) : base(name) { }

// Ghi đè phương thức Add để thêm một đối tượng Animal vào danh sách con của nhóm loài động vật này

public override void Add(Animal animal)

{

\_animals.Add(animal);

}

// Ghi đè phương thức PrintTree để in ra cấu trúc cây của nhóm loài động vật này và các loài động vật con bên trong nó

public override void PrintTree(int indent)

{

Console.WriteLine(new String('-', indent) + "+ " + Name);

foreach (var animal in \_animals)

{

animal.PrintTree(indent + 2);

}

}

}

// Lớp CompositeExample chứa hàm Main để chạy chương trình

public static class CompositeExample

{

public static void Main()

{

// Tạo ra các đối tượng loài động vật và nhóm loài động vật

var mammal = new SpeciesComposite("Mammal");

var carnivore = new SpeciesComposite("Carnivore");

var nonCarnivore = new SpeciesComposite("Non-Carnivore");

var tiger = new Species("Tiger");

var leopard = new Species("Leopard");

var sheep = new Species("Sheep");

var cow = new Species("Cow");

// Thêm các loài động vật vào nhóm loài động vật phù hợp

carnivore.Add(tiger);

carnivore.Add(leopard);

nonCarnivore.Add(sheep);

nonCarnivore.Add(cow);

// Thêm các nhóm loài động vật vào nhóm loài động vật lớn hơn

mammal.Add(carnivore);

mammal.Add(nonCarnivore);

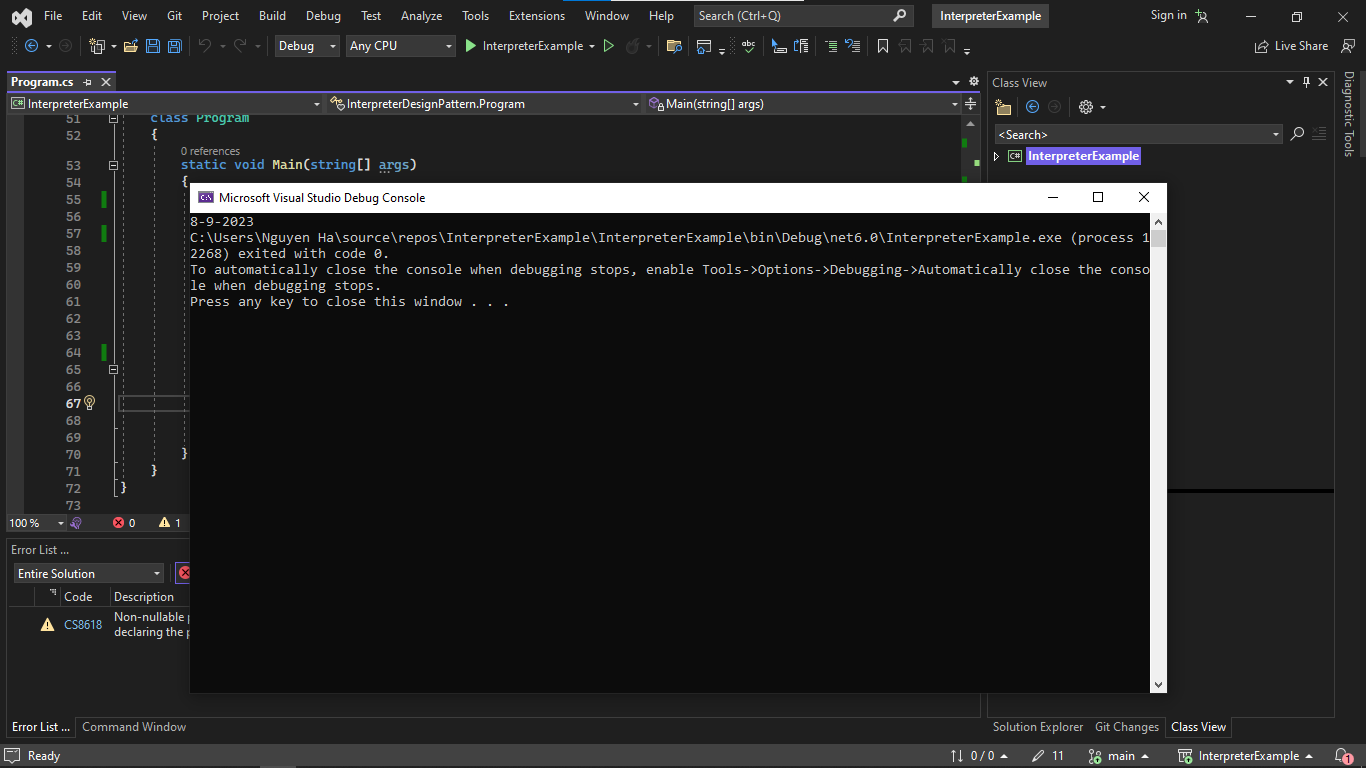
// In ra cấu trúc cây của toàn bộ hệ thống

mammal.PrintTree(1);

}

}

**InterpreterEx:**



using System;

namespace InterpreterDesignPattern

{

// Lớp Context chứa thông tin về biểu thức và ngày tháng

public class Context

{

public string Expression { get; set; }

public DateTime Date { get; set; }

public Context(DateTime date)

{

this.Date = date;

}

}

// Interface IExpression định nghĩa phương thức Evaluate để đánh giá biểu thức trong một Context cụ thể

public interface IExpression

{

string Evaluate(Context context);

}

// Lớp DayExpression đánh giá ngày trong một Context cụ thể

public class DayExpression : IExpression

{

public string Evaluate(Context context)

{

return context.Date.Day.ToString();

}

}

// Lớp MonthExpression đánh giá tháng trong một Context cụ thể

public class MonthExpression : IExpression

{

public string Evaluate(Context context)

{

return context.Date.Month.ToString();

}

}

// Lớp YearExpression đánh giá năm trong một Context cụ thể

public class YearExpression : IExpression

{

public string Evaluate(Context context)

{

return context.Date.Year.ToString();

}

}

// Lớp SeparatorExpression trả về dấu gạch ngang để phân tách các phần tử trong biểu thức

public class SeparatorExpression : IExpression

{

public string Evaluate(Context context)

{

return "-";

}

}

class Program

{

static void Main(string[] args)

{

// Khởi tạo một Context với ngày giờ hiện tại

var context = new Context(DateTime.Now);

// Khởi tạo danh sách các biểu thức để đánh giá

var expressionList = new System.Collections.Generic.List<IExpression>();

expressionList.Add(new DayExpression());

expressionList.Add(new SeparatorExpression());

expressionList.Add(new MonthExpression());

expressionList.Add(new SeparatorExpression());

expressionList.Add(new YearExpression());

// Đánh giá từng biểu thức và in kết quả ra màn hình

foreach (var expression in expressionList)

{

Console.Write(expression.Evaluate(context));

}

Console.ReadKey();

}

}

}