COGNIZANT DN - 4.0 DEEP SKILLING

HANDS ON WEEK-1

Exercise 1- Implementing the Singleton Pattern.

Code:

using System;

public sealed class Singleton

{

private static readonly Singleton instance = new Singleton();

private Singleton()

{

Console.WriteLine("Singleton Instance Created");

}

public static Singleton Instance

{

get

{

return instance;

}

}

public void ShowMessage()

{

Console.WriteLine("Hello i am Ayushmaan");

}

}

class Program

{

static void Main(string[] args)

{

Singleton s1 = Singleton.Instance;

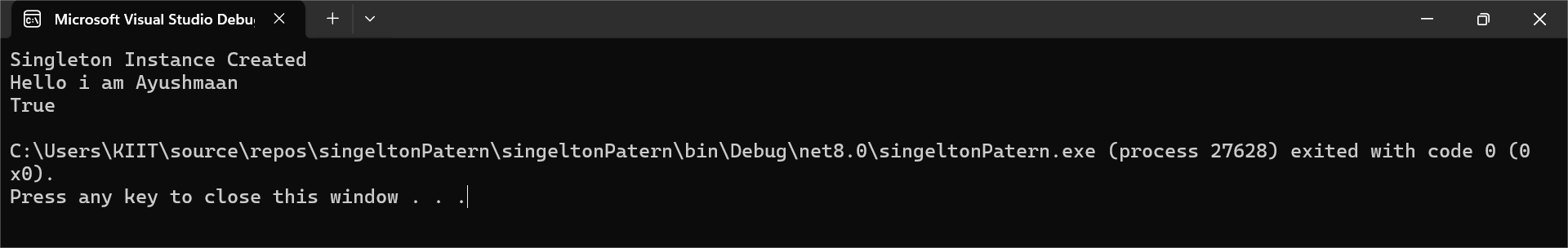
s1.ShowMessage();

Singleton s2 = Singleton.Instance;

Console.WriteLine(Object.ReferenceEquals(s1, s2));

}

}



Exercise 2-Implementing the Factory Method Pattern

Code:

using System;

public abstract class Document

{

public abstract void Print();

}

public class Resume : Document

{

public override void Print()

{

Console.WriteLine("Printing Resume Document...");

}

}

public class Report : Document

{

public override void Print()

{

Console.WriteLine("Printing Report Document...");

}

}

public abstract class DocumentFactory

{

public abstract Document CreateDocument();

}

public class ResumeFactory : DocumentFactory

{

public override Document CreateDocument()

{

return new Resume();

}

}

public class ReportFactory : DocumentFactory

{

public override Document CreateDocument()

{

return new Report();

}

}

class Program

{

static void Main(string[] args)

{

DocumentFactory resumeFactory = new ResumeFactory();

Document resume = resumeFactory.CreateDocument();

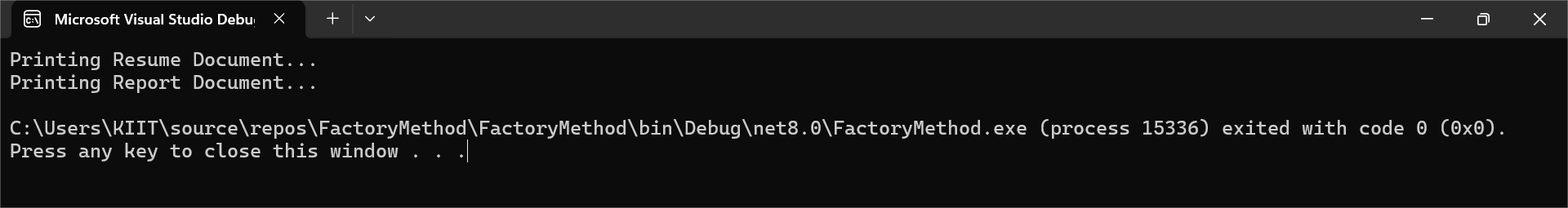
resume.Print();

DocumentFactory reportFactory = new ReportFactory();

Document report = reportFactory.CreateDocument();

report.Print();

}

}

Exercise 2 -E-commerce Platform Search Function

Code:

using System;

using System.Collections.Generic;

using System.Linq;

public class Product

{

public string Name { get; set; }

public string Category { get; set; }

public decimal Price { get; set; }

public Product(string name, string category, decimal price)

{

Name = name;

Category = category;

Price = price;

}

public void Display()

{

Console.WriteLine($"Product: {Name}, Category: {Category}, Price: ₹{Price}");

}

}

public class ProductSearch

{

private List<Product> products;

public ProductSearch(List<Product> products)

{

this.products = products;

}

public List<Product> SearchByName(string name)

{

return products.Where(p => p.Name.Contains(name, StringComparison.OrdinalIgnoreCase)).ToList();

}

public List<Product> SearchByCategory(string category)

{

return products.Where(p => p.Category.Equals(category, StringComparison.OrdinalIgnoreCase)).ToList();

}

public List<Product> SearchByPriceRange(decimal minPrice, decimal maxPrice)

{

return products.Where(p => p.Price >= minPrice && p.Price <= maxPrice).ToList();

}

}

class Program

{

static void Main(string[] args)

{

List<Product> products = new List<Product>

{

new Product("Redmi Note 13", "Mobile", 13999),

new Product("Samsung Galaxy S21", "Mobile", 49999),

new Product("HP Pavilion", "Laptop", 65999),

new Product("Apple MacBook Air", "Laptop", 99999),

new Product("Boat Headphones", "Accessories", 1999)

};

ProductSearch search = new ProductSearch(products);

Console.WriteLine("Search by Category: 'Laptop'");

var laptops = search.SearchByCategory("Laptop");

laptops.ForEach(p => p.Display());

Console.WriteLine("\nSearch by Name: 'Galaxy'");

var galaxy = search.SearchByName("Galaxy");

galaxy.ForEach(p => p.Display());

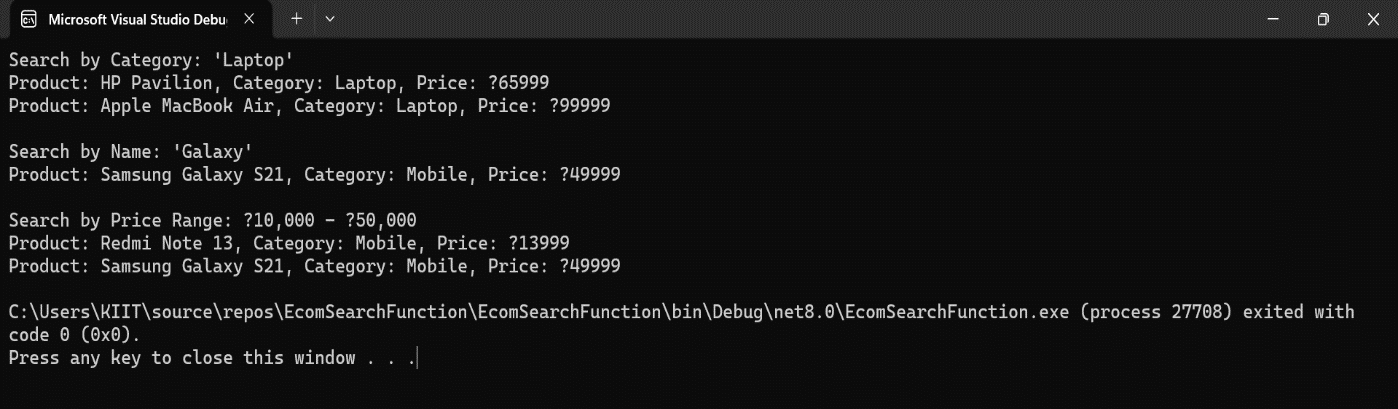
Console.WriteLine("\nSearch by Price Range: ₹10,000 - ₹50,000");

var midRange = search.SearchByPriceRange(10000, 50000);

midRange.ForEach(p => p.Display());

}

}



Exercise 7-Financial Forecasting

Code:

using System;

namespace FinancialForecasting

{

class Program

{

static void Main(string[] args)

{

Console.WriteLine(" Financial Forecasting Tool");

Console.Write("Enter current value (e.g., revenue or investment): ");

double currentValue = Convert.ToDouble(Console.ReadLine());

Console.Write("Enter annual/monthly growth rate (in %, e.g., 10 for 10%): ");

double growthRatePercent = Convert.ToDouble(Console.ReadLine());

Console.Write("Enter number of periods (e.g., years or months): ");

int periods = Convert.ToInt32(Console.ReadLine());

double growthRate = growthRatePercent / 100;

Console.WriteLine("\n--- Using Recursive Method ---");

double forecastRecursive = ForecastRecursive(currentValue, growthRate, periods);

Console.WriteLine($"Forecasted Value after {periods} periods: {forecastRecursive:F2}");

Console.WriteLine("\n--- Using Iterative Method (Optimized) ---");

double forecastIterative = ForecastIterative(currentValue, growthRate, periods);

Console.WriteLine($"Forecasted Value after {periods} periods: {forecastIterative:F2}");

Console.WriteLine("\n Forecasting complete.");

}

static double ForecastRecursive(double value, double rate, int periods)

{

if (periods == 0)

return value;

return ForecastRecursive(value, rate, periods - 1) \* (1 + rate);

}

static double ForecastIterative(double value, double rate, int periods)

{

for (int i = 0; i < periods; i++)

{

value \*= (1 + rate);

}

return value;

}

}

}

