**WEEK1-6362596(Superset id)**

**Design principles & Patterns**

1. **Exercise 1: Implementing the Singleton Pattern**

**CODE:-**

using System;

public class Singleton

{

private static Singleton \_instance;

private static readonly object \_lock = new object();

private Singleton()

{

Console.WriteLine("Singleton instance created.");

}

public static Singleton GetInstance()

{

if (\_instance == null)

{

lock (\_lock)

{

if (\_instance == null)

{

\_instance = new Singleton();

}

}

}

return \_instance;

}

public void ShowMessage()

{

Console.WriteLine("Hello from Singleton instance!");

}

}

class Program

{

static void Main(string[] args)

{

Singleton s1 = Singleton.GetInstance();

s1.ShowMessage();

Singleton s2 = Singleton.GetInstance();

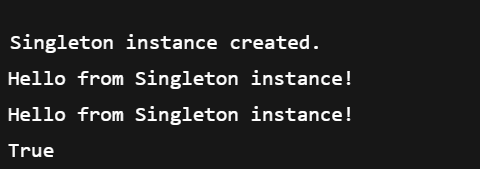
s2.ShowMessage();

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

}

}

**OUTPUT:-**



**Exercise 2: Implementing the Factory Method Pattern**

**CODE:-**

using System;

namespace FactoryPatternExample

{

public interface IShape

{

void Draw();

}

public class Circle : IShape

{

public void Draw()

{

Console.WriteLine("Drawing a Circle.");

}

}

public class Rectangle : IShape

{

public void Draw()

{

Console.WriteLine("Drawing a Rectangle.");

}

}

public class ShapeFactory

{

public IShape GetShape(string shapeType)

{

switch (shapeType.ToLower())

{

case "circle":

return new Circle();

case "rectangle":

return new Rectangle();

default:

throw new ArgumentException("Invalid shape type");

}

}

}

class Program

{

static void Main(string[] args)

{

ShapeFactory factory = new ShapeFactory();

IShape shape1 = factory.GetShape("circle");

shape1.Draw();

IShape shape2 = factory.GetShape("rectangle");

shape2.Draw();

try

{

IShape shape3 = factory.GetShape("triangle");

shape3.Draw();

}

catch (Exception ex)

{

Console.WriteLine($"Error: {ex.Message}");

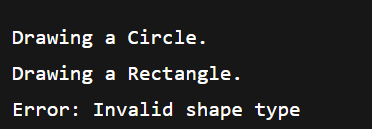
}

}

}

}

**OUTPUT:-**



**Exercise 2: E-commerce Platform Search Function**

**CODE:-**

using System;

using System.Collections.Generic;

using System.Linq;

namespace ECommerceSearch

{

public class Product

{

public int Id { get; set; }

public string Name { get; set; }

public double Price { get; set; }

public Product(int id, string name, double price)

{

Id = id;

Name = name;

Price = price;

}

public void Display()

{

Console.WriteLine($"ID: {Id}, Name: {Name}, Price: ₹{Price}");

}

}

class Program

{

static List<Product> SearchProducts(List<Product> products, string keyword)

{

return products

.Where(p => p.Name.ToLower().Contains(keyword.ToLower()))

.ToList();

}

static void Main(string[] args)

{

List<Product> productList = new List<Product>

{

new Product(1, "Red T-Shirt", 499.00),

new Product(2, "Blue Jeans", 999.00),

new Product(3, "Running Shoes", 1499.00),

new Product(4, "Smart Watch", 1999.00),

new Product(5, "Wireless Earbuds", 1299.00),

new Product(6, "Black T-Shirt", 450.00)

};

Console.WriteLine("Welcome to the E-Commerce Platform!");

Console.Write("Enter product name to search: ");

string input = Console.ReadLine();

List<Product> result = SearchProducts(productList, input);

if (result.Count > 0)

{

Console.WriteLine("\nSearch Results:");

foreach (var product in result)

{

product.Display();

}

}

else

{

Console.WriteLine("No products found matching your search.");

}

}

}

}

**OUTPUT:-**

Welcome to the E-Commerce Platform!

Enter product name to search: t-shirt

Search Results:

ID: 1, Name: Red T-Shirt, Price: ₹499

ID: 6, Name: Black T-Shirt, Price: ₹450

**Exercise 7: Financial Forecasting**

**CODE:-**

using System;

using System.Collections.Generic;

using System.Linq;

namespace FinancialForecasting

{

class Program

{

static List<double> ForecastRevenue(List<double> historicalData, int windowSize, int forecastMonths)

{

List<double> forecast = new List<double>();

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

{

var window = historicalData.Skip(historicalData.Count - windowSize).Take(windowSize).ToList();

double avg = window.Average();

forecast.Add(avg);

historicalData.Add(avg); // Add to historical for chained forecasting

}

return forecast;

}

static void Main(string[] args)

{

List<double> monthlyRevenue = new List<double> { 12.5, 13.0, 14.2, 15.1, 14.8, 16.0, 15.5, 16.3, 17.0, 17.5 };

Console.WriteLine("Financial Forecasting System");

Console.WriteLine("----------------------------");

Console.WriteLine("Historical Monthly Revenue (₹ in Lakhs):");

for (int i = 0; i < monthlyRevenue.Count; i++)

{

Console.WriteLine($"Month {i + 1}: ₹{monthlyRevenue[i]}");

}

int windowSize = 3;

int forecastMonths = 3;

List<double> forecasted = ForecastRevenue(new List<double>(monthlyRevenue), windowSize, forecastMonths);

Console.WriteLine("\nForecasted Revenue for Next 3 Months (Simple Moving Average):");

for (int i = 0; i < forecasted.Count; i++)

{

Console.WriteLine($"Month {monthlyRevenue.Count + i + 1}: ₹{forecasted[i]:0.00}");

}

}

}

}

**OUTPUT:-**

Financial Forecasting System

----------------------------

Historical Monthly Revenue (₹ in Lakhs):

Month 1: ₹12.5

Month 2: ₹13

Month 3: ₹14.2

Month 4: ₹15.1

Month 5: ₹14.8

Month 6: ₹16

Month 7: ₹15.5

Month 8: ₹16.3

Month 9: ₹17

Month 10: ₹17.5

Forecasted Revenue for Next 3 Months (Simple Moving Average):

Month 11: ₹16.93

Month 12: ₹17.27

Month 13: ₹17.57