Name - Ishita Chatterjee

Superset ID- 6362148

Design principles and patterns

Q1)Implementing the Singleton Pattern

Code:

using System;

public sealed class Logger

{

    private static readonly Lazy<Logger> \_instance =

        new Lazy<Logger>(() => new Logger());

    private static int \_instanceCount = 0;

    private Logger()

    {

        \_instanceCount++;

        Console.WriteLine("Logger instance created");

    }

    public static Logger Instance => \_instance.Value;

    public void Log(string message)

    {

        Console.WriteLine($"[LOG] {DateTime.Now}: {message}");

    }

    public static int GetInstanceCount()

    {

        return \_instanceCount;

    }

}

class Program

{

    static void Main(string[] args)

    {

        Logger logger1 = Logger.Instance;

        Logger logger2 = Logger.Instance;

        logger1.Log("First log message");

        logger2.Log("Second log message");

        Console.WriteLine($"Same instance? {ReferenceEquals(logger1, logger2)}");

        Console.WriteLine($"Instance count: {Logger.GetInstanceCount()}");

    }

}

Output:



Q2)Implementing the Factory Method Pattern

Code:

using System;

namespace FactoryMethodPatternExample

{   public interface IDocument

    {

                void Open();

    }

    public class WordDocument : IDocument

    {        public void Open()

        {

            Console.WriteLine("Opening a Word document.");

        }

    }

    public class PdfDocument : IDocument

    {

        public void Open()

        {

            Console.WriteLine("Opening a PDF document.");

        }

    }

    public class ExcelDocument : IDocument

    {

        public void Open()

        {

            Console.WriteLine("Opening an Excel document.");

        }

    }

    public abstract class DocumentFactory

    {

        public abstract IDocument CreateDocument();

    }

    public class WordDocumentFactory : DocumentFactory

    {

        public override IDocument CreateDocument()

        {

            return new WordDocument();

        }

    }

    public class PdfDocumentFactory : DocumentFactory

    {

        public override IDocument CreateDocument()

        {

            return new PdfDocument();

        }

    }

    public class ExcelDocumentFactory : DocumentFactory

    {

        public override IDocument CreateDocument()

        {

            return new ExcelDocument();

        }

    }

    class Program

    {

        static void Main(string[] args)

        {

            DocumentFactory wordFactory = new WordDocumentFactory();

            IDocument word = wordFactory.CreateDocument();

            word.Open();

            DocumentFactory pdfFactory = new PdfDocumentFactory();

            IDocument pdf = pdfFactory.CreateDocument();

            pdf.Open();

            DocumentFactory excelFactory = new ExcelDocumentFactory();

            IDocument excel = excelFactory.CreateDocument();

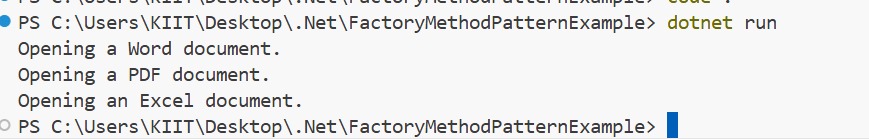
            excel.Open();

        }

    }

}

Output:



**Algorithms And Data Structures**

Q2)E-commerce Platform Search Function

Code:

using System;

using System.Linq;

public class Product

{

    public int ProductId { get; set; }

    public string ProductName { get; set; }

    public string Category { get; set; }

    public Product(int id, string name, string category)

    {

        ProductId = id;

        ProductName = name;

        Category = category;

    }

}

class Program

{

    public static Product LinearSearch(Product[] products, string targetName)

    {

        foreach (var product in products)

        {

            if (product.ProductName.Equals(targetName, StringComparison.OrdinalIgnoreCase))

                return product;

        }

        return null;

    }

    public static Product BinarySearch(Product[] sortedProducts, string targetName)

    {

        int left = 0;

        int right = sortedProducts.Length - 1;

        while (left <= right)

        {

            int mid = (left + right) / 2;

            int comparison = string.Compare(sortedProducts[mid].ProductName, targetName, true);

            if (comparison == 0)

                return sortedProducts[mid];

            else if (comparison < 0)

                left = mid + 1;

            else

                right = mid - 1;

        }

        return null;

    }

    static void Main(string[] args)

    {

        Product[] products = new Product[]

        {

            new Product(1, "Laptop", "Electronics"),

            new Product(2, "Headphones", "Electronics"),

            new Product(3, "Shoes", "Fashion"),

            new Product(4, "Coffee Maker", "Home"),

        };

        string searchName = "Shoes";

        Product linearResult = LinearSearch(products, searchName);

        Console.WriteLine("Linear Search Result: " +

            (linearResult != null ? $"{linearResult.ProductName} in {linearResult.Category}" : "Product not found"));

        Product[] sortedProducts = products.OrderBy(p => p.ProductName).ToArray();

        Product binaryResult = BinarySearch(sortedProducts, searchName);

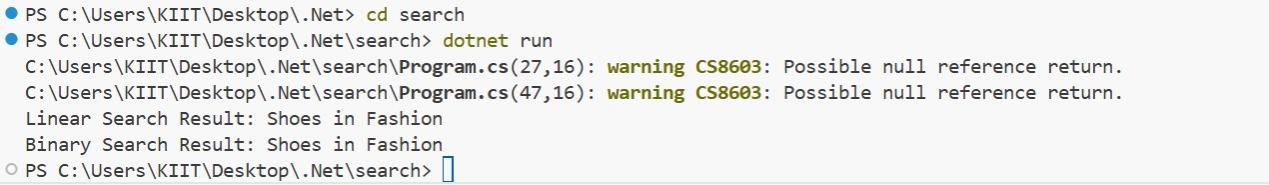
        Console.WriteLine("Binary Search Result: " +

            (binaryResult != null ? $"{binaryResult.ProductName} in {binaryResult.Category}" : "Product not found"));

    }

}

Output:



Q7)Financial Forecasting

Code:

using System;

class FinancialForecast

{

    public static double PredictFutureValue(double initialValue, double growthRate, int years)

    {

        if (years == 0)

            return initialValue;

        return PredictFutureValue(initialValue, growthRate, years - 1) \* (1 + growthRate);

    }

    public static double PredictFutureValueIterative(double initialValue, double growthRate, int years)

    {

        double value = initialValue;

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

        {

            value \*= (1 + growthRate);

        }

        return value;

    }

    static void Main(string[] args)

    {

        // Input values

        double initialValue = 10000;

        double growthRate = 0.05;

        int years = 10;

        double futureValueRecursive = PredictFutureValue(initialValue, growthRate, years);

        Console.WriteLine($"[Recursive] Future Value after {years} years: {futureValueRecursive:F2}");

        double futureValueIterative = PredictFutureValueIterative(initialValue, growthRate, years);

        Console.WriteLine($"[Iterative] Future Value after {years} years: {futureValueIterative:F2}");

    }

}

Output:

