**Week 1 Mandatory Hands-On**

**Design Patterns and Principles**

1. **Singleton Pattern :**

You need to ensure that a logging utility class in your application has only one instance throughout the application lifecycle to ensure consistent logging.

**CODE:**

**Logger.cs** :

using System;

public class Logger

{

    private static Logger? \_instance=null;

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

    private Logger()

    {

        Console.WriteLine("New instance created");

    }

    public static Logger GetInstance()

    {

        if (\_instance == null)

        {

            lock (\_lock)

            {

                if (\_instance == null)

                {

                    \_instance = new Logger();

                }

            }

        }

        return \_instance;

    }

    public void Log(string message)

    {

        Console.WriteLine(message);

    }

}

**Program.cs:**

using System;

class Program

{

    static void Main(string[] args)

    {

        Logger log1 = Logger.GetInstance();

        Logger log2 = Logger.GetInstance();

        log1.Log("First Message");

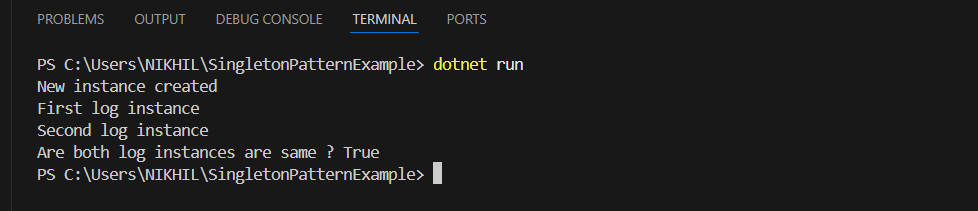
        log2.Log("Second Message");

        Console.WriteLine("Are both log instanceas are same ? "+ object.ReferenceEquals(log1, log2));

    }

}

Output:



1. **Factory Method Pattern:**

You are developing a document management system that needs to create different types of documents (e.g., Word, PDF, Excel). Use the Factory Method Pattern to achieve this.

**CODE:**

**IDocument:**

public interface IDocument

{

    void Open();

}

**Document Classes(Word,Pdf,Excel) :**

**WordDocument.cs :**

using System;

public class WordDocument : IDocument

{

    public void Open()

    {

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

    }

}

**PdfDocument.cs** :

using System;

public class PdfDocument : IDocument

{

    public void Open()

    {

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

    }

}

**ExcelDocument.cs :**

using System;

public class ExcelDocument : IDocument

{

    public void Open()

    {

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

    }

}

**Abstract Document Factory :**

**DocumentFactory.cs :**

public abstract class DocumentFactory

{

    public abstract IDocument CreateDocument();

}

**Concrete Factories for Documents :**

**WordFactor.cs :**

public class WordFactory : DocumentFactory

{

    public override IDocument CreateDocument()

    {

        return new WordDocument();

    }

}

**PdfFactory.cs :**

**public class PdfFactory : DocumentFactory**

**{**

**public override IDocument CreateDocument()**

**{**

**return new PdfDocument();**

**}**

**}**

**ExcelFactory.cs :**

public class ExcelFactory : DocumentFactory

{

    public override IDocument CreateDocument()

    {

        return new ExcelDocument();

    }

}

**Program.cs :**

**// See https://aka.ms/new-console-template for more information**

**using System;**

**class Program**

**{**

**static void Main(string[] args)**

**{**

**DocumentFactory wordFactory = new WordFactory();**

**IDocument wordDoc = wordFactory.CreateDocument();**

**wordDoc.Open();**

**DocumentFactory pdfFactory = new PdfFactory();**

**IDocument pdfDoc = pdfFactory.CreateDocument();**

**pdfDoc.Open();**

**DocumentFactory excelFactory = new ExcelFactory();**

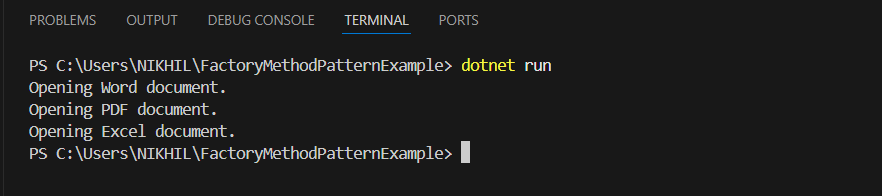
**IDocument excelDoc = excelFactory.CreateDocument();**

**excelDoc.Open();**

**}**

**}**

**Output :**

****

**Data Structures and Algorithms :**

1. **E-commerce Platform Search Function :**

You are working on the search functionality of an e-commerce platform. The search needs to be optimized for fast performance.

**Code :**

**Product.cs :**

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;

    }

}

**Program.cs :**

**using System;**

**using System.Linq;**

**class Program**

**{**

**static void Main(string[] args)**

**{**

**Product[] products = new Product[]**

**{**

**new Product(101, "Mobile", "Electronics"),**

**new Product(102, "Soap", "Personal Care"),**

**new Product(103, "Notebook", "Stationery"),**

**new Product(104, "Washing Machine", "Electronics"),**

**new Product(105, "Pen", "Stationery"),**

**};**

**Console.WriteLine("Enter product name");**

**String s = Console.ReadLine();**

**var result1 = LinearSearch(products,s);**

**if (result1 == null)**

**{**

**Console.WriteLine($"Not found {result1} in linear search");**

**}**

**else {**

**Console.WriteLine($"Found {result1} in linear search");**

**}**

**var sortedProducts = products.OrderBy(p => p.ProductName).ToArray();**

**var result2 = BinarySearch(sortedProducts,s);**

**if (result2 == null)**

**{**

**Console.WriteLine($"Not found {result1} in Binary search");**

**}**

**else {**

**Console.WriteLine($"Found {result1} in Binary search");**

**}**

**}**

**static Product? LinearSearch(Product[] products, string name)**

**{**

**foreach (var p in products)**

**{**

**if (p.ProductName.Equals(name,StringComparison.OrdinalIgnoreCase))**

**return p;**

**}**

**return null;**

**}**

**static Product? BinarySearch(Product[] products, String name)**

**{**

**int left = 0;**

**int right = products.Length - 1;**

**while (left <= right)**

**{**

**int mid = (left + right) / 2;**

**int comparsion = String.Compare(products[mid].ProductName, name, true);**

**if(comparsion == 0) {**

**return products[mid];**

**}**

**else if (comparsion < 0)**

**left = mid + 1;**

**else**

**right = mid - 1;**

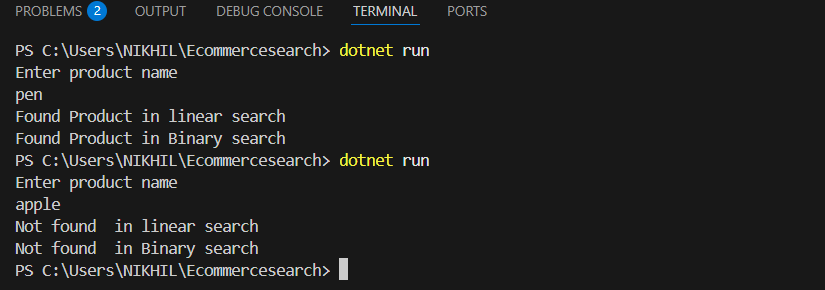
**}**

**return null;**

**}**

**}**

**Output :**

****

1. **Financial Forecasting :**

You are developing a financial forecasting tool that predicts future values based on past data.

**Code :**

**Program.cs :**

**using System;**

**class Program**

**{**

**static void Main(string[] args)**

**{**

**Console.WriteLine("Enter initial amount for investment: ");**

**double initial = Convert.ToDouble(Console.ReadLine());**

**Console.WriteLine("Enter annual growth rate (in percentage (%)): ");**

**double rate = Convert.ToDouble(Console.ReadLine());**

**Console.WriteLine("Enter number of years to Forecast : ");**

**int years = Convert.ToInt32(Console.ReadLine());**

**double forecastvalue= CalculateforecastValue(initial, rate / 100, years);**

**Console.WriteLine($"\nForecasted Value after {years} years: ${forecastvalue:F2}");**

**}**

**static double CalculateforecastValue(double currentValue, double growthRate, int years)**

**{**

**if (years == 0)**

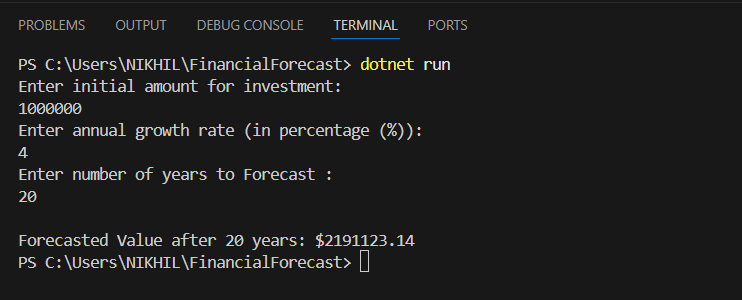
**return currentValue;**

**return CalculateforecastValue(currentValue \* (1 + growthRate), growthRate, years - 1);**

**}**

**}**

**Output :**

****