**Superset\_ID**- 6362873

**Name** - Abhijeet Ranjan

**Week-01 HandsOn Solution**

* **Design Patterns and Principles**

**Exercise 1: Implementing the Singleton Pattern**

* **Code:**

using System;

public class Logger{

    private static Logger? instance;

private static readonly object lockObj = new object();

private Logger() { }

    public static Logger Instance

    {

        get

        {

            lock (lockObj)

            {

                if (instance == null)

                    instance = new Logger();

                return instance;

            }

        }

}

    public void Log(string message){

        Console.WriteLine("Log: " + message);

    }

}

public class Program{

    public static void Main()

    {

        Logger log1 = Logger.Instance;

        Logger log2 = Logger.Instance;

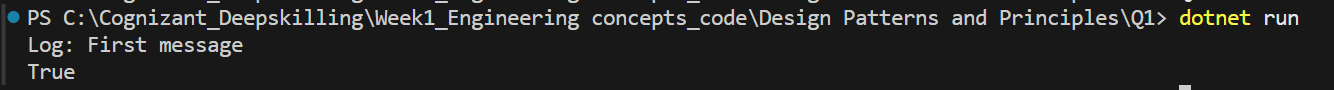
        log1.Log("First message");

        Console.WriteLine(log1 == log2);

    }

}

* **Output:**



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

* **Code:**

using System;

interface IDocument{

    void Open();

}

class WordDocument : IDocument{

    public void Open() => Console.WriteLine("Word document opened.");

}

class PdfDocument : IDocument{

    public void Open() => Console.WriteLine("PDF document opened.");

}

abstract class DocumentFactory{

    public abstract IDocument CreateDocument();

}

class WordFactory : DocumentFactory{

    public override IDocument CreateDocument() => new WordDocument();

}

class PdfFactory : DocumentFactory{

    public override IDocument CreateDocument() => new PdfDocument();

}

public class Program{

    public static void Main(){

        DocumentFactory factory = new PdfFactory();

        IDocument doc = factory.CreateDocument();

        doc.Open();

    }

}

* **Output:**



* **Algorithms and Data Structures**

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

* **Code:**

using System;

class Product {

    public int ProductId { get; set; }

    public string ProductName { get; set; }

public string Category { get; set; }

    public Product(int productId, string productName, string category) {

        ProductId = productId;

        ProductName = productName;

        Category = category;

}

    public override string ToString() {

        return $"ID: {ProductId}, Name: {ProductName}, Category: {Category}";

    }

}

class SearchFunction {

    public static int LinearSearch(Product[] products, string name) {

        for (int i = 0; i < products.Length; i++) {

            if (products[i].ProductName.Equals(name, StringComparison.OrdinalIgnoreCase)) {

                return i;

            }

        }

        return -1;

}

    public static int BinarySearch(Product[] products, string name) {

        int left = 0, right = products.Length - 1;

        while (left <= right) {

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

            int cmp = string.Compare(products[mid].ProductName, name, StringComparison.OrdinalIgnoreCase);

            if (cmp == 0)

                return mid;

            else if (cmp < 0)

                left = mid + 1;

            else

                right = mid - 1;

        }

        return -1;

    }

}

class Program {

    static void Main(string[] args) {

        Product[] products = {

            new Product(1, "Apple", "Fruits"),

            new Product(2, "Banana", "Fruits"),

            new Product(3, "Carrot", "Vegetables"),

            new Product(4, "Mango", "Fruits")

        };

        Console.WriteLine("=== Linear Search ===");

        int index = SearchFunction.LinearSearch(products, "Carrot");

        Console.WriteLine(index != -1 ? $"Found: {products[index]}" : "Product not found");

        Array.Sort(products, (p1, p2) => p1.ProductName.CompareTo(p2.ProductName));

        Console.WriteLine("\n=== Binary Search ===");

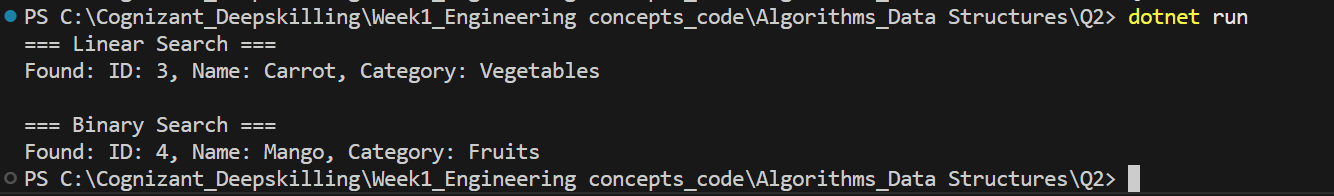
        index = SearchFunction.BinarySearch(products, "Mango");

        Console.WriteLine(index != -1 ? $"Found: {products[index]}" : "Product not found");

    }

}

* **Output:**



* **Analysis:**
* Linear Search: **O(n)**
* Binary Search: ****O(log n)**** (on **sorted** data)
* Use Binary Search for **performance-critical** scenarios with **sorted** data.

**Exercise 2: Financial Forecasting**

* **Code:**

using System;

class Program {

    static double PredictFutureValue(double currentValue, double growthRate, int years) {

        if (years == 0) return currentValue;

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

}

    static double PredictUsingLoop(double currentValue, double growthRate, int years) {

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

            currentValue \*= (1 + growthRate);

        }

        return currentValue;

}

    static void Main(string[] args) {

        double initial = 10000;

        double rate = 0.1;

        int years = 5;

        double futureRecursive = PredictFutureValue(initial, rate, years);

        double futureIterative = PredictUsingLoop(initial, rate, years);

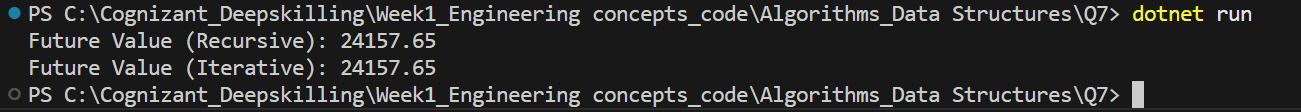
        Console.WriteLine($"Future Value (Recursive): {futureRecursive}");

        Console.WriteLine($"Future Value (Iterative): {futureIterative}");

    }

}

* **Output:**



* **Analysis:**

· Recursive: **O(n)** time, **O(n)** space (stack)

· Optimized (iterative): **O(n)** time, **O(1)** space

· Use **loops** or **memoization** to reduce stack overhead.