Name- Sayandeep Dey (SupersetID: 6363427)

WEEK – 1 (Handson- Exercises)

1. Design principles & Patterns:

Exercise 1: Implementing the Singleton Pattern:

Code:

In Logger.cs:

```
using System;

public class Logger
{
    private static Logger? instance = null;
    private static readonly object padlock = new object();

    private Logger()
    {
        Console.WriteLine("Logger Initialized");
    }

    public static Logger Instance
    {
        get
        {
            lock (padlock)
            {
                if (instance == null)
                {
                    instance = new Logger();
                }
                return instance;
            }
        }
    }

    public void Log(string message)
    {
        Console.WriteLine("LOG: " + message);
    }
}
```

In Program.cs:

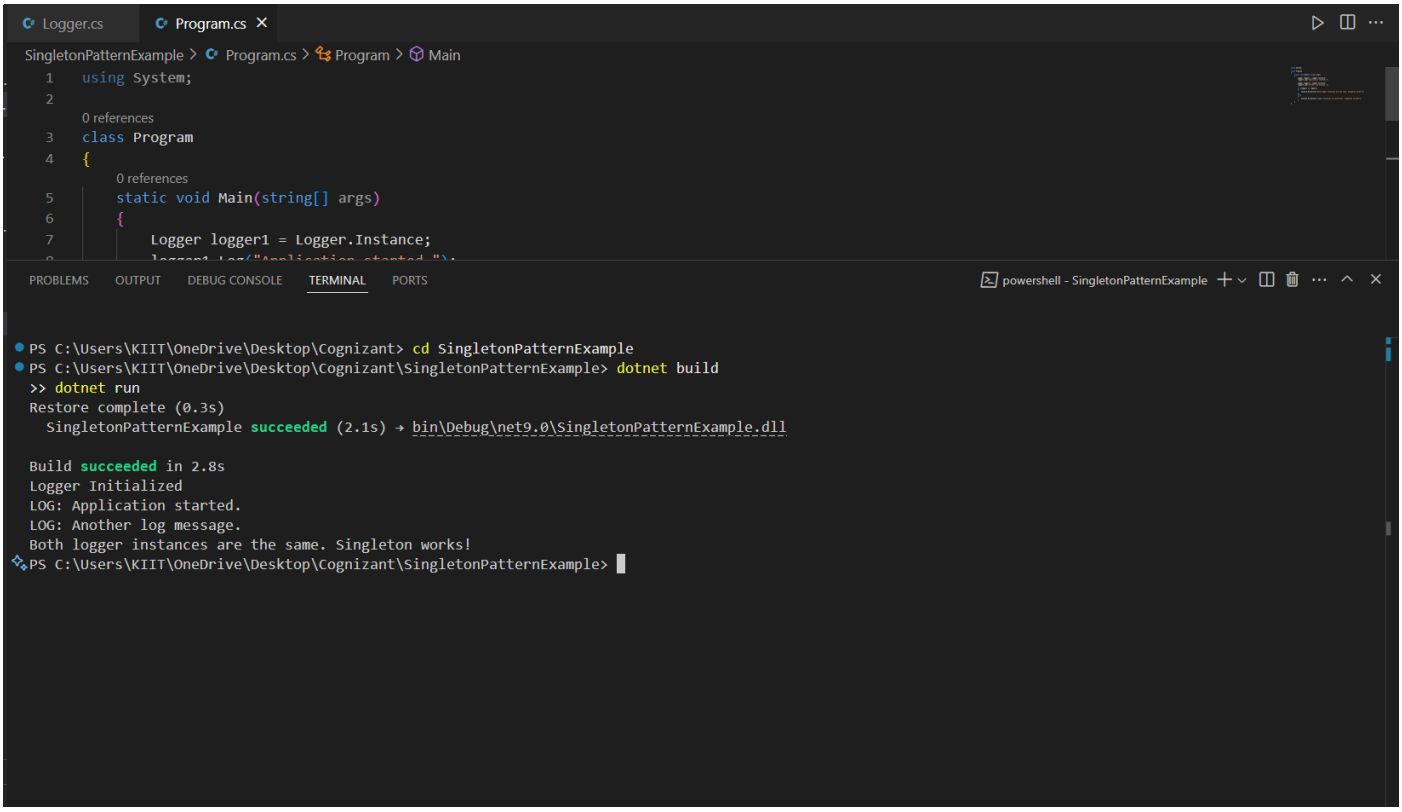
```
using System;
```

```
class Program
{
    static void Main(string[] args)
    {
        Logger logger1 = Logger.Instance;
        logger1.Log("Application started.");

        Logger logger2 = Logger.Instance;
        logger2.Log("Another log message.");

        if (logger1 == logger2)
        {
            Console.WriteLine("Both logger instances are the same. Singleton works!");
        }
        else
        {
            Console.WriteLine("Logger instances are different. Singleton failed!");
        }
    }
}
```

Output:



Exercise 2: Implementing the Factory Method Pattern

Code:

Document.cs – Interface:

```
public interface Document
{
```

```
    void Open();
}
```

WordDocument.cs:

```
using System;

public class WordDocument : Document
{
    public void Open()
    {
        Console.WriteLine("Opening Word Document");
    }
}
```

PdfDocument.cs:

```
using System;

public class PdfDocument : Document
{
    public void Open()
    {
        Console.WriteLine("Opening PDF Document");
    }
}
```

ExcelDocument.cs:

```
using System;

public class ExcelDocument : Document
{
    public void Open()
    {
        Console.WriteLine("Opening Excel Document");
    }
}
```

DocumentFactory.cs:

```
public abstract class DocumentFactory
{
    public abstract Document CreateDocument();
}
```

WordDocumentFactory.cs:

```
public class WordDocumentFactory : DocumentFactory
```

```
{
    public override Document CreateDocument()
    {
        return new WordDocument();
    }
}
```

PdfDocumentFactory.cs:

```
public class PdfDocumentFactory : DocumentFactory
{
    public override Document CreateDocument()
    {
        return new PdfDocument();
    }
}
```

ExcelDocumentFactory.cs:

```
public class ExcelDocumentFactory : DocumentFactory
{
    public override Document CreateDocument()
    {
        return new ExcelDocument();
    }
}
```

Program.cs:

```
using System;

class Program
{
    static void Main(string[] args)
    {
        DocumentFactory wordFactory = new WordDocumentFactory();
        Document word = wordFactory.CreateDocument();
        word.Open();

        DocumentFactory pdfFactory = new PdfDocumentFactory();
        Document pdf = pdfFactory.CreateDocument();
        pdf.Open();

        DocumentFactory excelFactory = new ExcelDocumentFactory();
        Document excel = excelFactory.CreateDocument();
        excel.Open();
    }
}
```

Output:

Program.cs

FactoryMethodPatternExample > Program.cs > Program > Main

```
using System;

0 references
class Program
{
    0 references
    static void Main(string[] args)
    {
        DocumentFactory wordFactory = new WordDocumentFactory();
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS powershell - FactoryMethodPatternExample

```
PS C:\Users\KIIT\OneDrive\Desktop\Cognizant> cd FactoryMethodPatternExample
PS C:\Users\KIIT\OneDrive\Desktop\Cognizant\FactoryMethodPatternExample> dotnet build
>> dotnet run
Restore complete (0.3s)
  FactoryMethodPatternExample succeeded (1.9s) → bin\Debug\net9.0\FactoryMethodPatternExample.dll

Build succeeded in 2.7s
Opening Word Document
Opening PDF Document
Opening Excel Document
PS C:\Users\KIIT\OneDrive\Desktop\Cognizant\FactoryMethodPatternExample>
PS C:\Users\KIIT\OneDrive\Desktop\Cognizant\FactoryMethodPatternExample>
```

2. Data structures and Algorithms:

Exercise 2: E-commerce Platform Search Function:

Code:

In 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;
    }

    public override string ToString()
    {
        return $"ID: {ProductId}, Name: {ProductName}, Category: {Category}";
    }
}
```

In SearchEngine.cs:

```
using System;
```

```
public class SearchEngine
{
    // Linear Search
    public static Product? LinearSearch(Product[] products, string target)
    {
        foreach (var product in products)
        {
            if (product.ProductName.Equals(target, StringComparison.OrdinalIgnoreCase))
                return product;
        }
        return null;
    }

    // Binary Search (sorted by ProductName)
    public static Product? BinarySearch(Product[] sortedProducts, string target)
    {
        int left = 0;
        int right = sortedProducts.Length - 1;

        while (left <= right)
        {
            int mid = (left + right) / 2;
            int cmp = string.Compare(sortedProducts[mid].ProductName, target,
StringComparison.OrdinalIgnoreCase);

            if (cmp == 0)
                return sortedProducts[mid];
            else if (cmp < 0)
                left = mid + 1;
            else
                right = mid - 1;
        }

        return null;
    }
}
```

In Program.cs:

```
using System;
using System.Linq;

class Program
{
    static void Main()
    {
        Product[] products = new Product[]
        {
            new Product(1, "iPhone", "Electronics"),
            new Product(2, "Shoes", "Footwear"),
            new Product(3, "Laptop", "Electronics"),
            new Product(4, "Book", "Education"),
            new Product(5, "T-Shirt", "Clothing")
```

```
        };

        Console.Write(" Enter the product name to search: ");
        string? input = Console.ReadLine();

        if (string.IsNullOrWhiteSpace(input))
        {
            Console.WriteLine("Invalid product name.");
            return;
        }

        Console.WriteLine("\n Linear Search Result:");
        var linearResult = SearchEngine.LinearSearch(products, input);
        Console.WriteLine(linearResult != null ? linearResult : "Product not found");

        Console.WriteLine("\n Binary Search Result:");
        var sorted = products.OrderBy(p => p.ProductName).ToArray();
        var binaryResult = SearchEngine.BinarySearch(sorted, input);
        Console.WriteLine(binaryResult != null ? binaryResult : "Product not found");
    }
}
```

Output:

```
Product.cs   SearchEngine.cs   Program.cs
EcommerceSearchExample > Program.cs > Program > Main
4   class Program
6       static void Main()
12          new Product(3, "Laptop", "Electronics"),
13          new Product(4, "Book", "Education"),
14          new Product(5, "T-Shirt", "Clothing")
15      };
16
17      Console.Write(" Enter the product name to search: ");
18      string? input = Console.ReadLine();

PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

PS C:\Users\KIIT\OneDrive\Desktop\Cognizant> cd C:\Users\KIIT\OneDrive\Desktop\Cognizant\EcommerceSearchExample
PS C:\Users\KIIT\OneDrive\Desktop\Cognizant\EcommerceSearchExample> dotnet build
 >> dotnet run
 Restore complete (0.4s)
   EcommerceSearchExample succeeded (0.4s) → bin\Debug\net9.0\EcommerceSearchExample.dll

 Build succeeded in 1.2s
  Enter the product name to search: Book

  Linear Search Result:
 ID: 4, Name: Book, Category: Education

  Binary Search Result:
 ID: 4, Name: Book, Category: Education
PS C:\Users\KIIT\OneDrive\Desktop\Cognizant\EcommerceSearchExample>

powershell
powershell...
powershell...
```

Analysis:

Algorithm	Time Complexity	Space Complexity	Notes
Linear Search	O(n)	O(1)	Works with unsorted data
Binary Search	O(log n)	O(1)	Requires sorted data

Exercise 7: Financial Forecasting:

Code:

In FinancialForecasting.cs:

```
using System;

public 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);
    }
}
```

In Program.cs:

```
using System;

class Program
{
    static void Main()
    {
        Console.WriteLine(" Financial Forecasting Tool");

        // User input for initial value
        Console.Write("Enter the initial amount (e.g., 1000): ");
        if (!double.TryParse(Console.ReadLine(), out double initialValue) || initialValue
< 0)
        {
            Console.WriteLine("Invalid initial value.");
            return;
        }

        // User input for growth rate
        Console.Write("Enter the annual growth rate in % (e.g., 10 for 10%): ");
        if (!double.TryParse(Console.ReadLine(), out double growthRate) || growthRate < 0)
        {
            Console.WriteLine("Invalid growth rate.");
            return;
        }

        // Convert percentage to decimal
```

```
        growthRate = growthRate / 100.0;

        // User input for number of years
        Console.Write("Enter the number of years to forecast (e.g., 5): ");
        if (!int.TryParse(Console.ReadLine(), out int years) || years < 0)
        {
            Console.WriteLine("Invalid number of years.");
            return;
        }

        // Call recursive method
        double futureValue = PredictFutureValue(initialValue, growthRate, years);

        // Display result
        Console.WriteLine($"\n Future value after {years} years: {futureValue:C2}");
    }

    // Recursive method to calculate future value
    static double PredictFutureValue(double initialValue, double growthRate, int years)
    {
        if (years == 0)
            return initialValue;

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

Output:

For the initial amount of 5000 and the annual growth rate of 15% and for 5 years the output will be:

FinancialForecast.cs | Program.cs

FinancialForecasting > Program.cs > ...

```
class Program
    }

    // Recursive method to calculate future value
    2 references
    static double PredictFutureValue(double initialValue, double growthRate, int years)
    {
        if (years == 0)
            return initialValue;
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

powershell - FinancialForecasting

```
PS C:\Users\KIIT\OneDrive\Desktop\Cognizant> cd FinancialForecasting
PS C:\Users\KIIT\OneDrive\Desktop\Cognizant\FinancialForecasting> dotnet build
>> dotnet run
Restore complete (0.4s)
  FinancialForecasting succeeded (2.2s) → bin\Debug\net9.0\FinancialForecasting.dll

Build succeeded in 3.1s
 Financial Forecasting Tool
Enter the initial amount (e.g., 1000): 5000
Enter the annual growth rate in % (e.g., 10 for 10%): 15
Enter the number of years to forecast (e.g., 5): 5

 Future value after 5 years: ₹ 10,056.79
PS C:\Users\KIIT\OneDrive\Desktop\Cognizant\FinancialForecasting>
```