Name : Aniket Khandelwal **Mail id : 2206156@kiit.ac.in**
Superset id : 6365314 **University Roll : 2206156**

WEEK - 01 : HandsOn Solutions

❖ Design Principles & Patterns :-

➢ Exercise - 1 : Implementing the Singleton Pattern.

Code :

```
using ...

namespace SingletonPatternExample
{
    0 references
    public class Program
    {
        0 references
        static void Main(string[] args)
        {
            Logger logger1 = Logger.GetInstance();
            Logger logger2 = Logger.GetInstance();

            logger1.Log("This is the first log message.");
            logger2.Log("This is the second log message.");

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

```
using ...
namespace SingletonPatternExample {
    8 references
    internal class Logger
    {
        private static Logger _instance;
        private static readonly object _lock = new object();
        1 reference
        private Logger()
        {
            Console.WriteLine("Logger instance created.");
        }
        2 references
        public static Logger GetInstance()
        {
            if (_instance == null)
            {
                lock (_lock)
                {
                    if (_instance == null)
                    {
                        _instance = new Logger();
                    }
                }
            }
            return _instance;
        }
        2 references
        public void Log(string message)
        {
            Console.WriteLine($"Log Entry: {message}");
        }
    }
}
```

Output:

```
C:\Users\KIIT\Desktop\Week
Logger instance created.
Log Entry: This is the first log message.
Log Entry: This is the second log message.
Both logger instances are the same, Hence Singleton works!
```

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

Code :

```
WordDocumentFactory.cs | DocumentFactory.cs | Program.cs | IDocument.cs | WordDocument.cs | PdfDocument.cs | ExcelDocument.cs
FactoryMethodPatternExample | FactoryMethodPatternExample.Program | Main(string[] args)

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Xml.Linq;

namespace FactoryMethodPatternExample
{
    0 references
    internal class Program
    {
        0 references
        static void Main(string[] args)
        {
            DocumentFactory wordFactory = new WordDocumentFactory();
            IDocument wordDoc = wordFactory.CreateDocument();
            wordDoc.Open();

            DocumentFactory pdfFactory = new PdfDocumentFactory();
            IDocument pdfDoc = pdfFactory.CreateDocument();
            pdfDoc.Open();

            DocumentFactory excelFactory = new ExcelDocumentFactory();
            IDocument excelDoc = excelFactory.CreateDocument();
            excelDoc.Open();

            Console.ReadLine();
        }
    }
}

100 % | No issues found | Ln: 9 | Ch: 2 | SPC | CRLF
```

```
WordDocumentFactory.cs | DocumentFactory.cs | Program.cs | IDocument.cs | WordDocument.cs | PdfDocument.cs | ExcelDocument.cs
FactoryMethodPatternExample | FactoryMethodPatternExample.IDocument | Open()

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace FactoryMethodPatternExample
{
    10 references
    public interface IDocument
    {
        6 references
        void Open();
    }
}
```

```
namespace FactoryMethodPatternExample
{
    6 references
    public abstract class DocumentFactory
    {
        6 references
        public abstract IDocument CreateDocument();
    }
}
```

```
namespace FactoryMethodPatternExample
{
    1 reference
    public class PdfDocumentFactory : DocumentFactory
    {
        4 references
        public override IDocument CreateDocument()
        {
            return new PdfDocument();
        }
    }

}
```

```
namespace FactoryMethodPatternExample
{
    1 reference
    public class WordDocumentFactory : DocumentFactory
    {
        4 references
        public override IDocument CreateDocument()
        {
            return new WordDocument();
        }
    }
}

```

```
namespace FactoryMethodPatternExample
{
    1 reference
    public class ExcelDocumentFactory : DocumentFactory
    {
        4 references
        public override IDocument CreateDocument()
        {
            return new ExcelDocument();
        }
    }

}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace FactoryMethodPatternExample
{
    1 reference
    public class PdfDocument : IDocument
    {
        4 references
        public void Open()
        {
            Console.WriteLine("Opening a PDF document.");
        }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace FactoryMethodPatternExample
{
    1 reference
    public class WordDocument : IDocument
    {
        4 references
        public void Open()
        {
            Console.WriteLine("Opening a Word document.");
        }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace FactoryMethodPatternExample
{
    1 reference
    public class ExcelDocument : IDocument
    {
        4 references
        public void Open()
        {
            Console.WriteLine("Opening an Excel document.");
        }
    }
}
```

Output :

```
Opening a Word document.
Opening a PDF document.
Opening an Excel document.
```

❖ Data Structures & Algorithms :-

➢ Exercise - 2 : E-commerce Platform Search Function

Code:

```
Program.cs*
E-commerce Platform Search | E_commerce_Platform_Search.Product | ProductId

using ...

namespace E_commerce_Platform_Search
{
    13 references
    public class Product
    {
        2 references
        public int ProductId { get; set; }
        6 references
        public string ProductName { get; set; }
        2 references
        public string Category { get; set; }

        5 references
        public Product(int productId, string productName, string category)
        {
            ProductId = productId;
            ProductName = productName;
            Category = category;
        }
    }
    0 references
    class Program
    {
        0 references
        static void Main(string[] args)
        {
            Product[] products = new Product[]
            {
                new Product(1, "Laptop", "Electronics"),
                new Product(2, "Phone", "Electronics"),
                new Product(3, "Shoes", "Fashion"),
                new Product(4, "Book", "Education"),
                new Product(5, "Tablet", "Electronics")
            };

100 %  No issues found  Ln: 6  Ch: 1  SPC  CRLF
```

```
Program.cs*
E-commerce Platform Search | E_commerce_Platform_Search.Product | ProductId

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

            Console.WriteLine("\n=== Linear Search result ===");
            var result1 = LinearSearch(products, searchQuery);
            PrintResult(result1);

            Array.Sort(products, (a, b) => string.Compare(a.ProductName, b.ProductName, StringComparison.OrdinalIgnoreCase));

            Console.WriteLine("\n=== Binary Search result ===");
            var result2 = BinarySearch(products, searchQuery);
            PrintResult(result2);

            Console.ReadKey();
        }
        1 reference
        static Product LinearSearch(Product[] products, string targetName)
        {
            foreach (var product in products)
            {
                if (product.ProductName.Equals(targetName, StringComparison.OrdinalIgnoreCase))
                {
                    return product;
                }
            }
            return null;
        }
        1 reference
        static Product BinarySearch(Product[] products, string targetName)
        {
            int left = 0;
            int right = products.Length - 1;

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

100 %  No issues found  Ln: 6  Ch: 1  SPC  CRLF
```

```
Program.cs*
E-commerce Platform Search | E_commerce_Platform_Search.Product | ProductId
        }
        1 reference
        static Product BinarySearch(Product[] products, string targetName)
        {
            int left = 0;
            int right = products.Length - 1;

            while (left <= right)
            {
                int mid = left + (right - left) / 2;
                int comparison = string.Compare(products[mid].ProductName, targetName, true);

                if (comparison == 0)
                    return products[mid];
                else if (comparison < 0)
                    left = mid + 1;
                else
                    right = mid - 1;
            }
            return null;
        }
        2 references
        static void PrintResult(Product product)
        {
            if (product != null)
            {
                Console.WriteLine($"Product Found: {product.ProductName} (ID: {product.ProductId}, Category: {product.Category})")
            }
            else
            {
                Console.WriteLine("Product not found.");
            }
        }
    }
}

100 % | No issues found | Ln: 6 | Ch: 1 | SPC | CRLF
```

Output :

```
C:\Users\KIIT\Desktop\Week

Enter product name to search: Tablet

=== Linear Search result ===
Product Found: Tablet (ID: 5, Category: Electronics)

=== Binary Search result ===
Product Found: Tablet (ID: 5, Category: Electronics)
```

Time Complexity analysis :

Algorithm	Best Case	Average Case	Worst Case
Linear Search	O(1)	O(n)	O(n)
Binary Search	O(1)	O(log n)	O(log n)

Which Algo is best suitable ?

The algorithm that is more suitable for our platform is Binary Search.

The main reason why binary search is more suitable for our e-commerce platform because , It provides logarithmic time performance(O(log n)), which is essential for scaling to thousands or millions of products.It also works well with sorted, indexed product catalogs, which is typical in backend databases or caching systems. It's more aligned with the needs of a high-performance, real-time search system.

➢ Exercise - 7 : Financial Forecasting

Code :

```
using ...

namespace Financial_Forecasting
{
    0 references
    class Program
    {
        0 references
        static void Main(string[] args)
        {
            Console.Write("Enter initial amount: ");
            double initialAmount = Convert.ToDouble(Console.ReadLine());

            Console.Write("Enter annual growth rate (e.g. 0.05 for 5%): ");
            double growthRate = Convert.ToDouble(Console.ReadLine());

            Console.Write("Enter number of years to forecast: ");
            int years = Convert.ToInt32(Console.ReadLine());

            double futureValue = CalculateFutureValueRecursive(initialAmount, growthRate, years);

            Console.WriteLine($"\nFuture Value after {years} years: {futureValue:F2}");

            Console.ReadKey();
        }

        2 references
        static double CalculateFutureValueRecursive(double currentValue, double growthRate, int yearsLeft)
        {
            if (yearsLeft == 0)
                return currentValue;

            return CalculateFutureValueRecursive(currentValue * (1 + growthRate), growthRate, yearsLeft - 1);
        }
    }
}
```

```
C:\Users\KIIT\Desktop\Week

Enter initial amount: 3500000
Enter annual growth rate (e.g. 0.05 for 5%): 0.12
Enter number of years to forecast: 3

Future Value after 3 years: 4917248.00
```

➢ Analysis :

Time Complexity = O(n)
Space Complexity = O(n)

Problem with above code is stack over flow for larger n value .

➢ Optimization :

Program.cs

Financial Forecasting | Financial_Forecasting.Program | Main(string[] args)

```
using ...

namespace Financial_Forecasting
{
    0 references
    class Program
    {
        0 references
        static void Main(string[] args)
        {
            Console.Write("Enter initial amount: ");
            double initialAmount = Convert.ToDouble(Console.ReadLine());

            Console.Write("Enter annual growth rate (e.g. 0.05 for 5%): ");
            double growthRate = Convert.ToDouble(Console.ReadLine());

            Console.Write("Enter number of years to forecast: ");
            int years = Convert.ToInt32(Console.ReadLine());

            double futureValue = CalculateFutureValueFormula(initialAmount, growthRate, years);

            Console.WriteLine($"\nFuture Value after {years} years: {futureValue:F2}");

            Console.ReadKey();
        }
        1 reference
        static double CalculateFutureValueFormula(double initialAmount, double growthRate, int years)
        {
            return initialAmount * Math.Pow(1 + growthRate, years);
        }
    }
}
```

C:\Users\KIIT\Desktop\Week

```
Enter initial amount: 200000
Enter annual growth rate (e.g. 0.05 for 5%): 0.12
Enter number of years to forecast: 10

Future Value after 10 years: 621169.64
```

Time Complexity: O(n)
Space Complexity: O(1)

No extra space is used and it is the Fastest and most efficient and best for production.