**DN 4.0 Dotnet FSE (WEEK 1)**

**Name:Sparsh Guha**

**Superset ID:6361106**

Week 1-DeepSkilling

Design Pattern and Principles

**#1.Implementing the Singleton Pattern**

**Code:**

using System;

public sealed class Logger

{

private static Logger? \_instance;

private Logger() { }

public static Logger Instance

{

get

{

if (\_instance == null)

{

\_instance = new Logger();

}

return \_instance;

}}

public void Log(string message)

{

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

}}

public class Program1

{

public static void Main(string[] args)

{

Logger logger1 = Logger.Instance;

Logger logger2 = Logger.Instance;

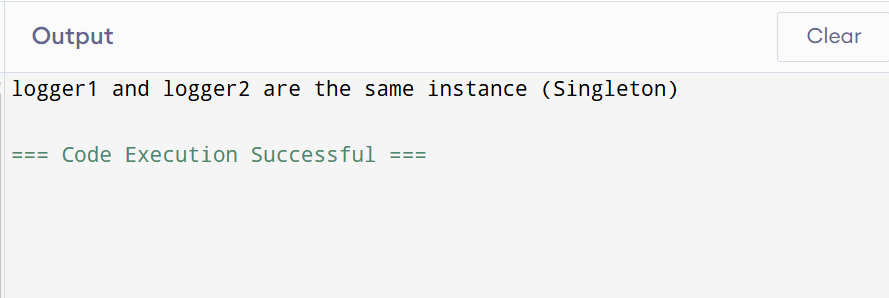
if (ReferenceEquals(logger1, logger2))

{

Console.WriteLine("logger1 and logger2 are the same instance (Singleton)");

}}}

**OUTPUT:**



**#2.Implementing the Factory Method Pattern**

**Code:**

using System;

public interface IDocument

{

void Open();

void Save();

}

public class WordDocument : IDocument

{

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

public void Save() => Console.WriteLine("Saving Word document...");

}

public class PdfDocument : IDocument

{

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

public void Save() => Console.WriteLine("Saving PDF document...");

}

public class ExcelDocument : IDocument

{

public void Open() => Console.WriteLine("Opening Excel document...");

public void Save() => Console.WriteLine("Saving Excel document...");

}

public abstract class DocumentFactory

{

public abstract IDocument CreateDocument();

public void ProcessDocument()

{

var doc = CreateDocument();

doc.Open();

doc.Save();

}}

public class WordDocumentFactory : DocumentFactory

{

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

}

public class PdfDocumentFactory : DocumentFactory

{

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

}

public class ExcelDocumentFactory : DocumentFactory

{

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

}

public class Program

{

public static void Main()

{

DocumentFactory wordFactory = new WordDocumentFactory();

DocumentFactory pdfFactory = new PdfDocumentFactory();

DocumentFactory excelFactory = new ExcelDocumentFactory();

Console.WriteLine("Creating Word Document:");

wordFactory.ProcessDocument();

Console.WriteLine("\nCreating PDF Document:");

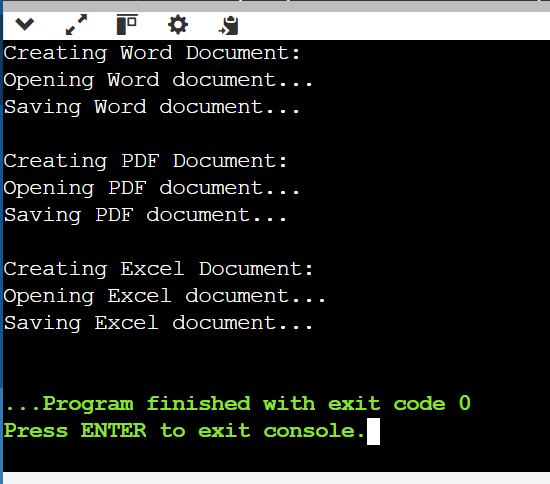
pdfFactory.ProcessDocument();

Console.WriteLine("\nCreating Excel Document:");

excelFactory.ProcessDocument();

}}

OUTPUT:



*Data Structures and Algorithms*

**#2.Ecommerce Platform Search Function**

**Code:**

using System;

using System.Diagnostics;

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 class SearchEngine

{

public static Product LinearSearch(Product[] products, int targetId)

{

foreach (var product in products)

{

if (product.ProductId == targetId)

return product;

}

return null;

}

public static Product BinarySearch(Product[] products, int targetId)

{

int left = 0;

int right = products.Length - 1;

while (left <= right)

{

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

if (products[mid].ProductId == targetId)

return products[mid];

if (products[mid].ProductId < targetId)

left = mid + 1;

else

right = mid - 1;

}

return null;

}}

public class Program

{

public static void Main()

{

Product[] products = {

new Product(101, "Laptop", "Electronics"),

new Product(203, "Desk Chair", "Furniture"),

new Product(305, "Coffee Maker", "Kitchen"),

new Product(407, "Smartphone", "Electronics"),

new Product(518, "Bookshelf", "Furniture")

};

Array.Sort(products, (x, y) => x.ProductId.CompareTo(y.ProductId));

Console.WriteLine("Enter Product ID:");

int searchId = Convert.ToInt32(Console.ReadLine());

var linearResult = SearchEngine.LinearSearch(products, searchId);

Console.WriteLine(linearResult != null

? $"Linear Search Found: {linearResult.ProductName}"

: "Product not found");

var binaryResult = SearchEngine.BinarySearch(products, searchId);

Console.WriteLine(binaryResult != null

? $"Binary Search Found: {binaryResult.ProductName}"

: "Product not found");

CompareSearchPerformance(products, searchId);

}

public static void CompareSearchPerformance(Product[] products, int targetId)

{

SearchEngine.LinearSearch(products, targetId);

SearchEngine.BinarySearch(products, targetId);

const int iterations = 1000000;

var timer = new Stopwatch();

timer.Start();

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

{

SearchEngine.LinearSearch(products, targetId);

}

timer.Stop();

Console.WriteLine($"\nLinear Search Time: {timer.ElapsedMilliseconds}ms");

timer.Restart();

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

{

SearchEngine.BinarySearch(products, targetId);

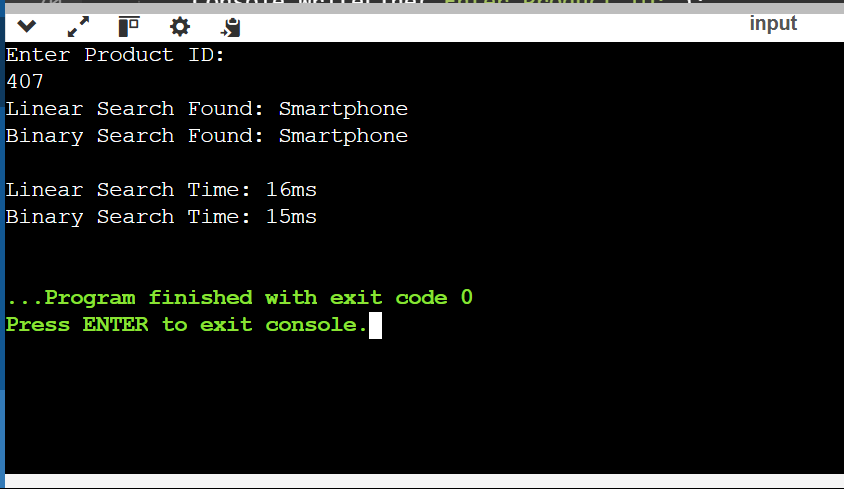
}

timer.Stop();

Console.WriteLine($"Binary Search Time: {timer.ElapsedMilliseconds}ms");

}}

**OUTPUT:**

****

**#7. Financial Forecasting**

**Code:**

using System;

public class FinancialForecasting

{

public static double CalculateFutureValue(double principal, double rate, int periods)

{

if (periods == 0)

return principal;

return CalculateFutureValue(principal, rate, periods - 1) \* (1 + rate);

}

public static double CalculateFutureValueOptimized(double principal, double rate, int periods)

{

double result = principal;

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

{

result \*= (1 + rate);

}

return result;

}}

public class Program

{

public static void Main()

{

double principal = 1000;

double rate = 0.05;

int periods = 10;

double futureValue = FinancialForecasting.CalculateFutureValue(principal, rate, periods);

Console.WriteLine($"Future value after {periods} periods (recursive): {futureValue:F2}");

double optimizedValue = FinancialForecasting.CalculateFutureValueOptimized(principal, rate, periods);

Console.WriteLine($"Future value after {periods} periods (optimized): {optimizedValue:F2}");

}}

**OUTPUT:**

