**Exercise 1: Implementing the Singleton Pattern**

**Scenario:**

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:**

using System;

public class Logger

{

private static Logger \_instance;

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

private Logger()

{

Console.WriteLine("Logger initialized.");

}

public static Logger Instance

{

get

{

lock (\_lock)

{

return \_instance ??= new Logger();

}

}

}

public void Log(string message)

{

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

}

}

class Program

{

static void Main()

{

Logger logger1 = Logger.Instance;

logger1.Log("First message");

Logger logger2 = Logger.Instance;

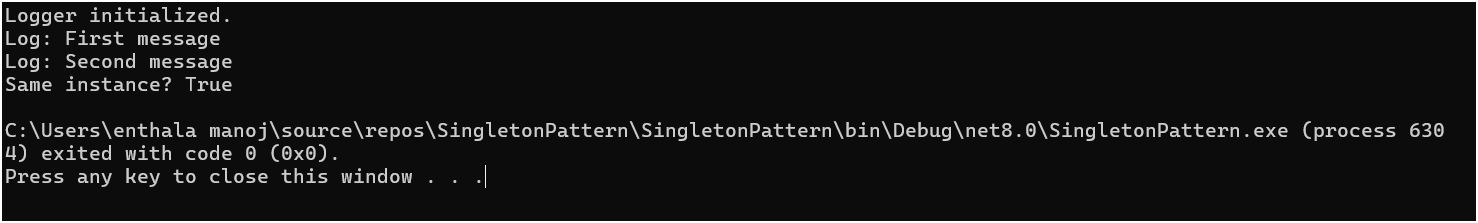
logger2.Log("Second message");

Console.WriteLine("Same instance? " + (logger1 == logger2));

}

}

**Output:**

****

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

**Scenario:**

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:**

**using System;**

**public interface IDocument**

**{**

**void Open();**

**}**

**public class WordDocument : IDocument**

**{**

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

**}**

**public class PdfDocument : IDocument**

**{**

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

**}**

**public class ExcelDocument : IDocument**

**{**

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

**}**

**public abstract class DocumentFactory**

**{**

**public abstract IDocument CreateDocument();**

**}**

**public class WordFactory : DocumentFactory**

**{**

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

**}**

**public class PdfFactory : DocumentFactory**

**{**

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

**}**

**public class ExcelFactory : DocumentFactory**

**{**

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

**}**

**class Program**

**{**

**static void Main()**

**{**

**DocumentFactory factory;**

**factory = new WordFactory();**

**IDocument word = factory.CreateDocument();**

**word.Open();**

**factory = new PdfFactory();**

**IDocument pdf = factory.CreateDocument();**

**pdf.Open();**

**factory = new ExcelFactory();**

**IDocument excel = factory.CreateDocument();**

**excel.Open();**

**}**

**}**

**Output:**

****

**Exercise 2: E-commerce Platform Search Function**

**Scenario:**

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

**Code:**

**using System;**

**public 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 Program**

**{**

**static void Main()**

**{**

**Product[] products = {**

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

**new Product(102, "Shoes", "Fashion"),**

**new Product(103, "Laptop", "Electronics"),**

**new Product(104, "Book", "Education"),**

**new Product(105, "Watch", "Accessories")**

**};**

**Console.WriteLine("🔍 Linear Search:");**

**Product foundLinear = LinearSearch(products, 103);**

**Console.WriteLine(foundLinear != null ? foundLinear.ToString() : "Product not found");**

**Console.WriteLine("\n🔍 Binary Search:");**

**Array.Sort(products, (a, b) => a.ProductId.CompareTo(b.ProductId));**

**Product foundBinary = BinarySearch(products, 103);**

**Console.WriteLine(foundBinary != null ? foundBinary.ToString() : "Product not found");**

**Console.ReadLine();**

**}**

**static Product LinearSearch(Product[] products, int productId)**

**{**

**foreach (Product product in products)**

**{**

**if (product.ProductId == productId)**

**return product;**

**}**

**return null;**

**}**

**static Product BinarySearch(Product[] products, int productId)**

**{**

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

**while (left <= right)**

**{**

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

**if (products[mid].ProductId == productId)**

**return products[mid];**

**else if (products[mid].ProductId < productId)**

**left = mid + 1;**

**else**

**right = mid - 1;**

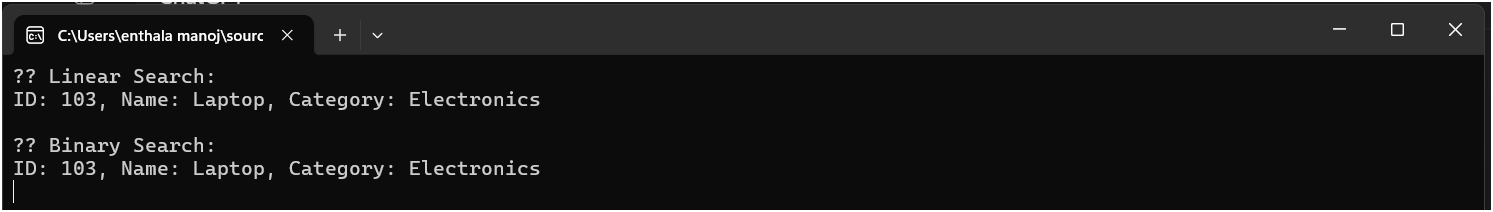
**}**

**return null;**

**}**

**}**

**Output:**



**Exercise 7: Financial Forecasting**

**Scenario:**

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

**Code:  
using System;**

**class Program**

**{**

**static void Main()**

**{**

**double initialAmount = 10000;**

**double annualGrowthRate = 0.08;**

**int years = 5;**

**Console.WriteLine($"Initial Amount: ₹{initialAmount}");**

**double futureValue = ForecastFutureValue(initialAmount, annualGrowthRate, years);**

**Console.WriteLine($"Forecast after {years} years: ₹{futureValue:F2}");**

**}**

**static double ForecastFutureValue(double amount, double growthRate, int years)**

**{**

**if (years == 0)**

**return amount;**

**return ForecastFutureValue(amount \* (1 + growthRate), growthRate, years - 1);**

**}**

**}**

**Output:**

