**Week 1: Design Patterns and Principles**

**Exercise 1 : Implementing the Singleton Pattern**

**Logger.java**

public class Logger {

    private static final Logger instance = new Logger();

private Logger() {

        System.out.println("Logger initialized.");

    }

public static Logger getInstance() {

        return instance;

    }

public void log(String message) {

        System.out.println("Log: " + message);

    }

}

**Main.java**

public class main {

    public static void main(String[] args) {

        Logger logger1 = Logger.getInstance();

        Logger logger2 = Logger.getInstance();

logger1.log("First log message.");

        logger2.log("Second log message.");

if (logger1 == logger2) {

            System.out.println("Only one Logger instance exists.");

        } else {

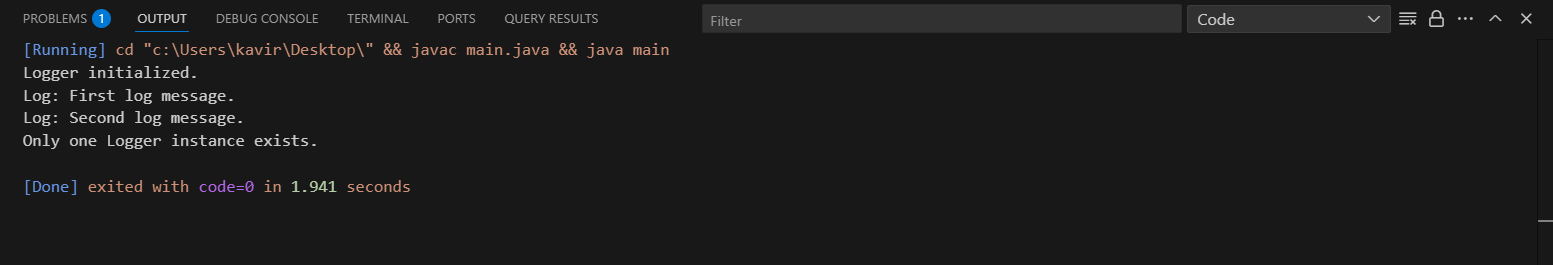
            System.out.println("Different Logger instances exist.");

        }

    }

}

**Output:**

****

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

**Main.java**

interface Document {

void open();

}

class Word implements Document {

public void open() {

System.out.println("Opening a Word Document.");

} }

class Pdf implements Document {

public void open() {

System.out.println("Opening a PDF Document.");

}}

class Excel implements Document {

public void open() {

System.out.println("Opening an Excel Document.");

}}

abstract class Document {

abstract Document createDocument();

}

class WordDocumentFactory extends DocumentFactory {

public Document createDocument() {

return new WordDocument();

} }

class PdfDocumentFactory extends DocumentFactory {

public Document createDocument() {

return new PdfDocument();

} }

class ExcelDocumentFactory extends DocumentFactory {

public Document createDocument() {

return new ExcelDocument();

}

}

public class FactoryMethodPatternExample {

public static void main(String[] args) {

DocumentFactory wordFactory = new WordDocumentFactory();

DocumentFactory pdfFactory = new PdfDocumentFactory();

DocumentFactory excelFactory = new ExcelDocumentFactory();

Document wordDoc = wordFactory.createDocument();

Document pdfDoc = pdfFactory.createDocument();

Document excelDoc = excelFactory.createDocument();

wordDoc.open();

pdfDoc.open();

excelDoc.open();

}

}

**Output:**

****

**Algorithms and Data Structures**

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

using System;

class Product

{

public int Id;

public string Name;

public string Category;

public Product(int id, string name, string category)

{

Id = id;

Name = name;

Category = category;

}

}

class Program

{

static Product LinearSearch(Product[] products, int searchId)

{

foreach (var p in products)

if (p.Id == searchId)

return p;

return null;

}

static Product BinarySearch(Product[] products, int searchId)

{

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

while (left <= right)

{

int mid = (left + right) / 2;

if (products[mid].Id == searchId)

return products[mid];

if (products[mid].Id < searchId)

left = mid + 1;

else

right = mid - 1;

}

return null;

}

static void Main()

{

Product[] products = {

new Product(3, "Mouse", "Electronics"),

new Product(1, "Laptop", "Electronics"),

new Product(2, "Shoes", "Fashion")

};

Console.WriteLine("Linear Search for ID = 2:");

var result1 = LinearSearch(products, 2);

if (result1 != null)

Console.WriteLine($"{result1.Id} - {result1.Name} ({result1.Category})");

else

Console.WriteLine("Product not found.");

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

Console.WriteLine("\nBinary Search for ID = 1:");

var result2 = BinarySearch(products, 1);

if (result2 != null)

Console.WriteLine($"{result2.Id} - {result2.Name} ({result2.Category})");

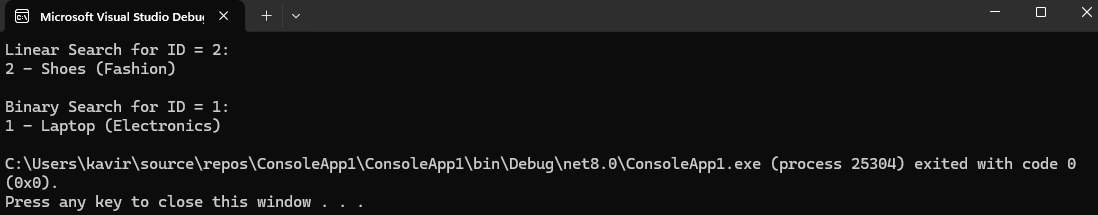
else

Console.WriteLine("Product not found.");

}

}

**Output:**



**Comparison between Linear Search and Binary Search:**

Linear Search Binary Search

|  |  |
| --- | --- |
| Time Complexity is O(n) | Time Complexity is O(log n) |
| It checks each item one by one | It divides the itemsets in half for searching |
| Small datasets can easily be sorted | It can handle large datasets but requires sorted data |
| It is slow because it checks one by one | It is much faster if the datasets are in sorted format |

Therefore, for E-Commerce Platform Search Function **Binary Search algorithm** is best suitable because its search speed is much faster for e-commerce search operation.

**Exercise 7: Financial Forecasting**

**Understanding Recursive Algorithms:**

Recursion means a method calling itself to solve the problems.

**Method to create recursive approach:**

static double Predict(int year, double initial, double growth)

{

if (year == 0) return initial;

return Predict(year - 1, initial, growth) \* (1 + growth);

}

**Implementation of Recursive Approach:**

using System;

class Program

{

static double Predict(int year, double initial, double growth)

{

if (year == 0) return initial;

return Predict(year - 1, initial, growth) \* (1 + growth);

}

static void Main()

{

double initial = 1000;

double rate = 0.10;

int years = 5;

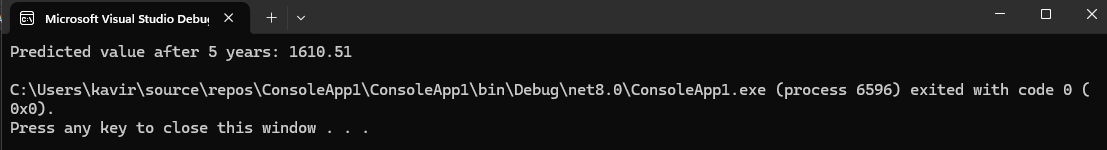
double futureValue = Predict(years, initial, rate);

Console.WriteLine($"Predicted value after {years} years: {futureValue:F2}");

}

}

**Output:**

****

**Analysis:**

* Time Complexity for predicting value each year will be O(n).

To optimize the recursive approach, we can use **Iterative Approach** to avoid excessive computation.

Iterative Approach:

static double Predict(int year, double initial, double growth)

{

double value = initial;

for (int i = 1; i <= year; i++)

value \*= (1 + growth);

return value;

}