**WEEK 1**

**Design Patterns and Principles**

**Exercise 1: Implementing the Singleton Pattern**

**File: Logger.java**

public class Logger {

// Step 2: Private static instance of the same class

private static Logger instance;

// Step 3: Private constructor to prevent instantiation

private Logger() {

System.out.println("Logger instance created");

}

// Step 4: Public static method to provide global access point

public static Logger getInstance() {

if (instance == null) {

instance = new Logger(); // Lazy initialization

}

return instance;

}

// Sample log method

public void log(String message) {

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

}

}

**File: LoggerTest.java**

public class LoggerTest {

public static void main(String[] args) {

Logger logger1 = Logger.getInstance();

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

Logger logger2 = Logger.getInstance();

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

// Check if both loggers point to the same instance

if (logger1 == logger2) {

System.out.println("Both logger instances are the same (Singleton works).");

} else {

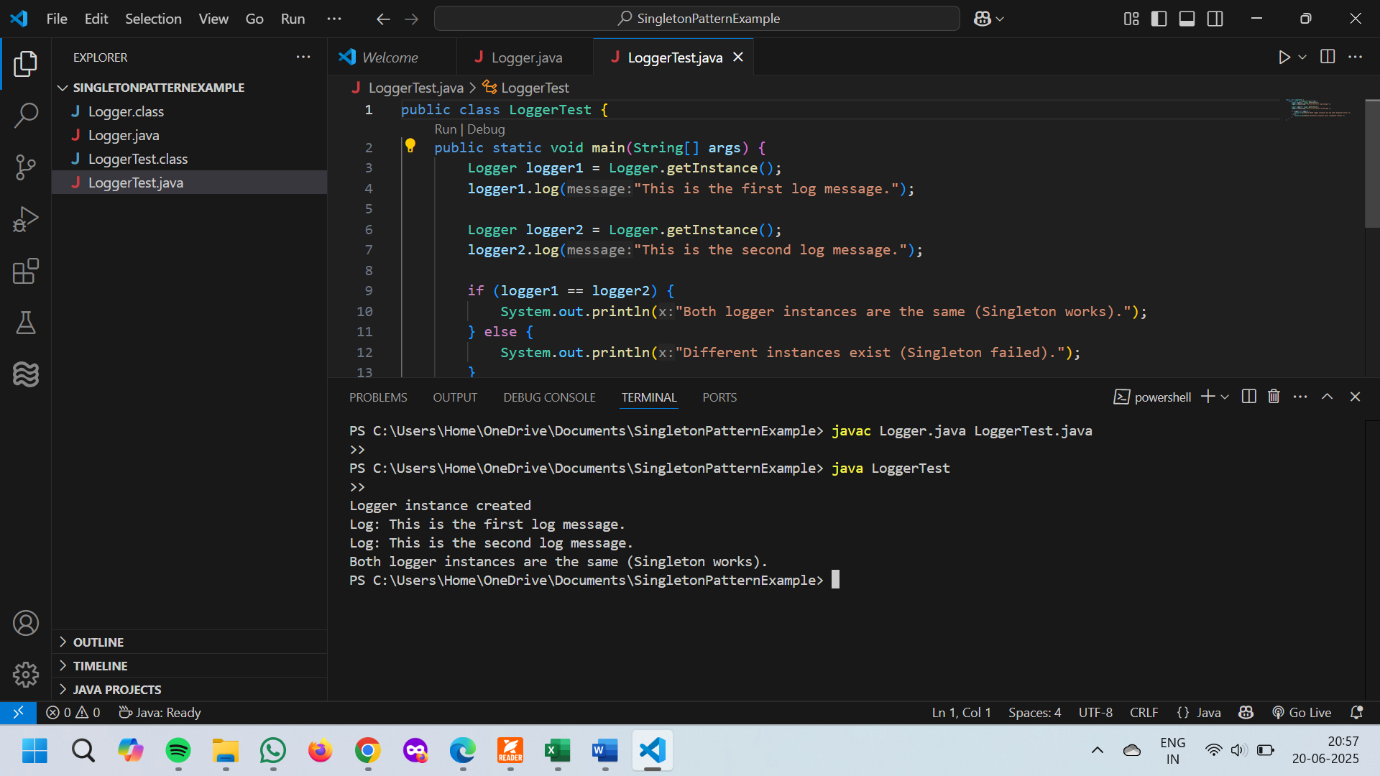
System.out.println("Different instances exist (Singleton failed).");

}

}

}

**OUTPUT:**

****

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

**File: Document.java**

public interface Document {

void open();

}

**File: WordDocument.java**

public class WordDocument implements Document {

public void open() {

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

}

}

**File: PdfDocument.java**

public class PdfDocument implements Document {

public void open() {

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

}

}

**File: ExcelDocument.java**

public class ExcelDocument implements Document {

public void open() {

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

}

}

**File: DocumentFactory.java**

public abstract class DocumentFactory {

public abstract Document createDocument();

}

**File: WordDocumentFactory.java**

public class WordDocumentFactory extends DocumentFactory {

public Document createDocument() {

return new WordDocument();

}

}

**File: PdfDocumentFactory.java**

public class PdfDocumentFactory extends DocumentFactory {

public Document createDocument() {

return new PdfDocument();

}

}

**File: ExcelDocumentFactory.java**

public class ExcelDocumentFactory extends DocumentFactory {

public Document createDocument() {

return new ExcelDocument();

}

}

**File: DocumentFactoryTest.java**

public class DocumentFactoryTest {

public static void main(String[] args) {

DocumentFactory wordFactory = new WordDocumentFactory();

Document wordDoc = wordFactory.createDocument();

wordDoc.open();

DocumentFactory pdfFactory = new PdfDocumentFactory();

Document pdfDoc = pdfFactory.createDocument();

pdfDoc.open();

DocumentFactory excelFactory = new ExcelDocumentFactory();

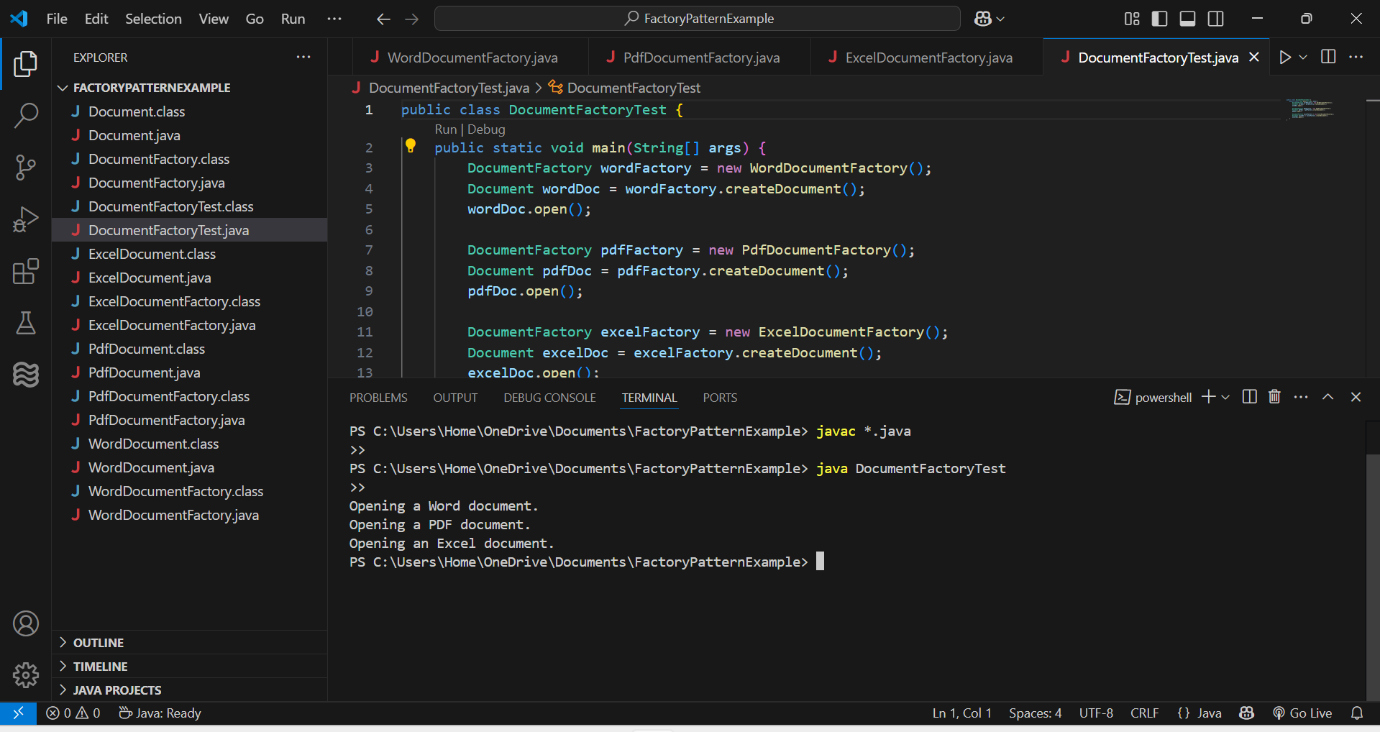
Document excelDoc = excelFactory.createDocument();

excelDoc.open();

}

}

**OUTPUT:**

****

**Algorithms\_Data Structures**

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

**File: Product.java**

public class Product {

int productId;

String productName;

String category;

public Product(int productId, String productName, String category) {

this.productId = productId;

this.productName = productName;

this.category = category;

}

public String toString() {

return "[" + productId + ", " + productName + ", " + category + "]";

}

}

**File: SearchEngine.java**

import java.util.Arrays;

import java.util.Comparator;

public class SearchEngine {

// Linear Search by product name

public static Product linearSearch(Product[] products, String name) {

for (Product p : products) {

if (p.productName.equalsIgnoreCase(name)) {

return p;

}

}

return null;

}

// Binary Search by product name (array must be sorted by name)

public static Product binarySearch(Product[] products, String name) {

int low = 0;

int high = products.length - 1;

while (low <= high) {

int mid = (low + high) / 2;

int compare = products[mid].productName.compareToIgnoreCase(name);

if (compare == 0) {

return products[mid];

} else if (compare < 0) {

low = mid + 1;

} else {

high = mid - 1;

}

}

return null;

}

// Sample products

public static Product[] getProducts() {

return new Product[]{

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

new Product(102, "Shirt", "Fashion"),

new Product(103, "Camera", "Electronics"),

new Product(104, "Shoes", "Fashion"),

new Product(105, "Mobile", "Electronics")

};

}

public static void main(String[] args) {

Product[] products = getProducts();

// Linear search

System.out.println("Linear Search: Looking for 'Camera'");

Product result1 = linearSearch(products, "Camera");

System.out.println("Result: " + result1);

// Binary search (requires sorted array)

Arrays.sort(products, Comparator.comparing(p -> p.productName.toLowerCase()));

System.out.println("Binary Search: Looking for 'Camera'");

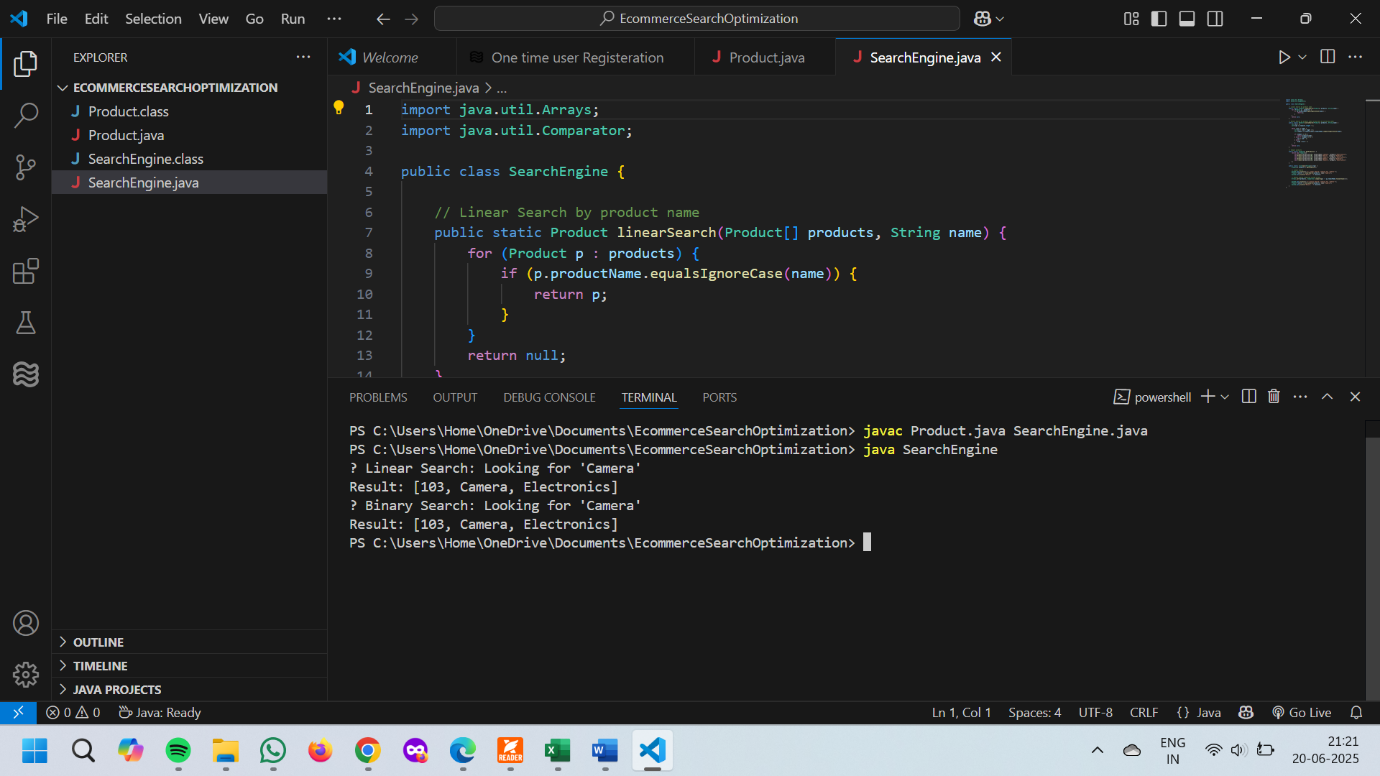
Product result2 = binarySearch(products, "Camera");

System.out.println("Result: " + result2);

}

}

**OUTPUT:**



**Exercise 7: Financial Forecasting**

**File: FinancialForecast.java**

public class FinancialForecast {

// Recursive method to calculate future value

public static double futureValueRecursive(double currentValue, double rate, int years) {

if (years == 0) {

return currentValue;

}

return futureValueRecursive(currentValue, rate, years - 1) \* (1 + rate);

}

public static void main(String[] args) {

double currentValue = 10000; // Current capital

double growthRate = 0.08; // 8% annual growth

int years = 5; // Forecast for 5 years

double result = futureValueRecursive(currentValue, growthRate, years);

System.out.printf("Future value after %d years: %.2f\n", years, result);

}

}

**OUTPUT:**

