DESIGN PATTERNS AND PRINCIPLES

Exercise 1: Implementing the Singleton Pattern

public class SingletonLoggerDemo {

static class Logger {

private static Logger instance;

private Logger() {

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

}

public static Logger getInstance() {

if (instance == null) {

instance = new Logger();

}

return instance;

}

public void log(String message) {

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

}

}

public static void main(String[] args) {

System.out.println("Testing Singleton Logger...");

// Get first instance

Logger logger1 = Logger.getInstance();

logger1.log("First message");

// Get second instance

Logger logger2 = Logger.getInstance();

logger2.log("Second message");

// Verification

if (logger1 == logger2) {

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

} else {

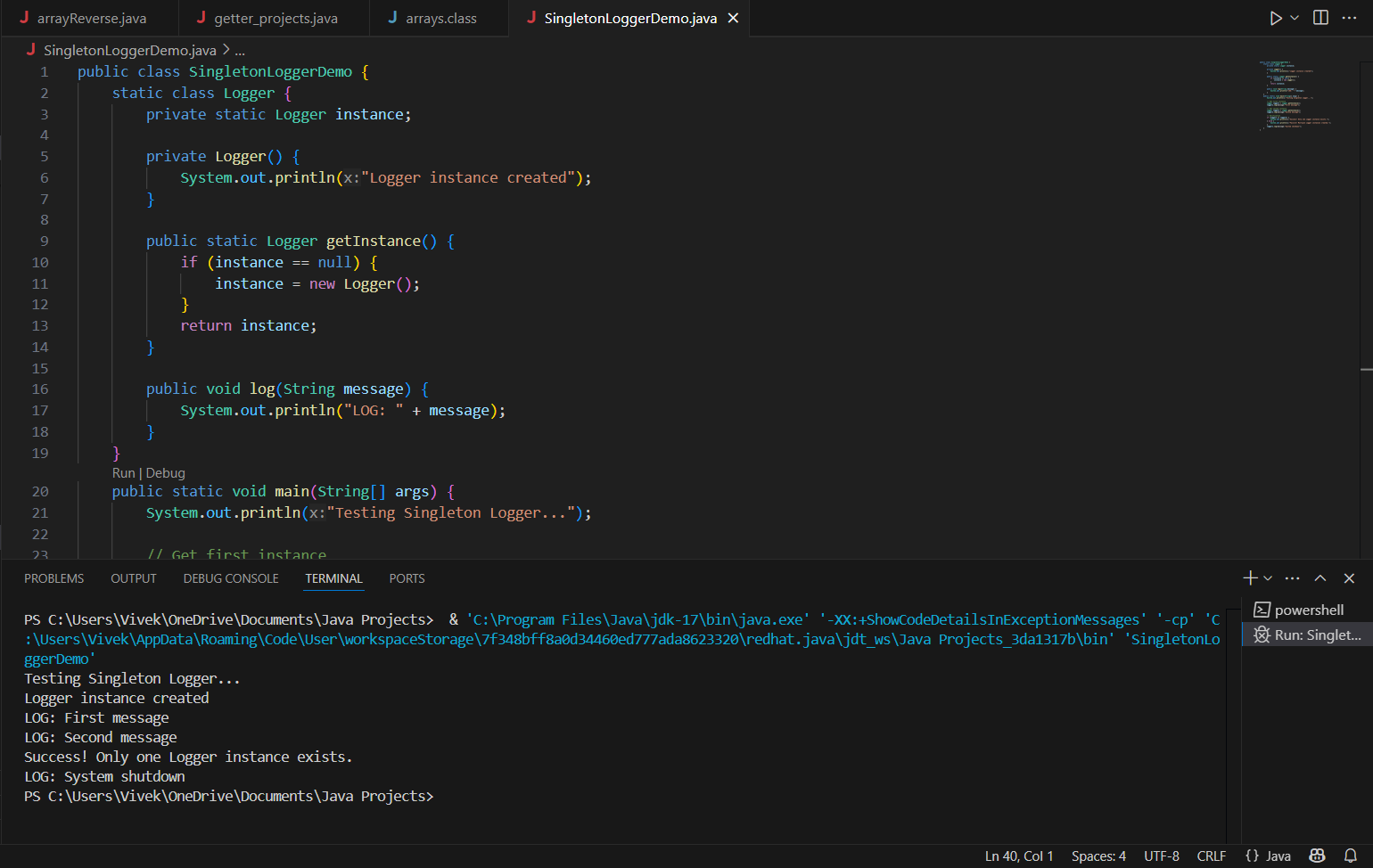
System.out.println("Failure! Multiple Logger instances created.");

}

logger1.log("System shutdown");

}

}



Exercise 2: Implementing the Factory Method Pattern

public class DocumentManagementSystem {

// Document Interface

interface Document {

void open();

void save();

void close();

}

// Concrete Document Implementations

static class WordDocument implements Document {

@Override

public void open() {

System.out.println("[Word] Opening document...");

}

@Override

public void save() {

System.out.println("[Word] Saving as .docx format");

}

@Override

public void close() {

System.out.println("[Word] Closing document");

}

}

static class PdfDocument implements Document {

@Override

public void open() {

System.out.println("[PDF] Opening document...");

}

@Override

public void save() {

System.out.println("[PDF] Saving as .pdf format");

}

@Override

public void close() {

System.out.println("[PDF] Closing document");

}

}

static class ExcelDocument implements Document {

@Override

public void open() {

System.out.println("[Excel] Opening spreadsheet...");

}

@Override

public void save() {

System.out.println("[Excel] Saving as .xlsx format");

}

@Override

public void close() {

System.out.println("[Excel] Closing spreadsheet");

}

}

// Factory Classes

abstract static class DocumentFactory {

public abstract Document createDocument();

public void newDocument() {

Document doc = createDocument();

doc.open();

}

}

static class WordDocumentFactory extends DocumentFactory {

@Override

public Document createDocument() {

return new WordDocument();

}

}

static class PdfDocumentFactory extends DocumentFactory {

@Override

public Document createDocument() {

return new PdfDocument();

}

}

static class ExcelDocumentFactory extends DocumentFactory {

@Override

public Document createDocument() {

return new ExcelDocument();

}

}

// Main Program to Test

public static void main(String[] args) {

System.out.println("=== Document Management System ===");

// Create factories

DocumentFactory wordFactory = new WordDocumentFactory();

DocumentFactory pdfFactory = new PdfDocumentFactory();

DocumentFactory excelFactory = new ExcelDocumentFactory();

// Create documents using factories

System.out.println("\nCreating documents:");

Document word = wordFactory.createDocument();

Document pdf = pdfFactory.createDocument();

Document excel = excelFactory.createDocument();

// Demonstrate document operations

System.out.println("\nWorking with documents:");

word.open();

word.save();

pdf.open();

pdf.close();

excel.save();

excel.close();

// Demonstrate factory method with newDocument()

System.out.println("\nUsing factory methods:");

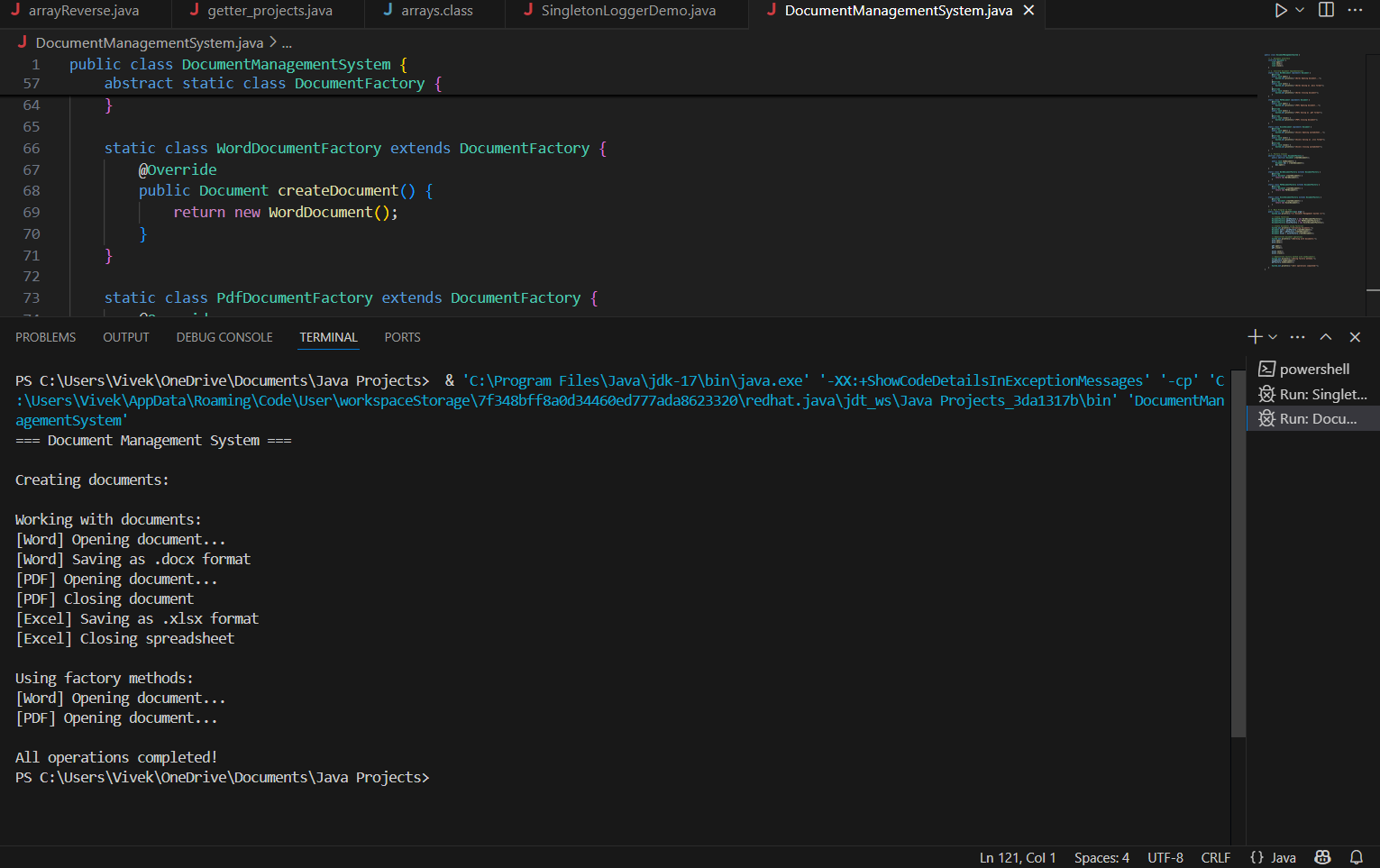
wordFactory.newDocument();

pdfFactory.newDocument();

System.out.println("\nAll operations completed!");

}

}



ALGORITHMS\_DATA STRUCTURES

Exercise 2: E-commerce Platform Search Function

import java.util.Arrays;

import java.util.Comparator;

public class ProductSearch {

// Product class representing items in our catalog

static class Product {

int productId;

String productName;

String category;

public Product(int id, String name, String category) {

this.productId = id;

this.productName = name;

this.category = category;

}

@Override

public String toString() {

return productId + ": " + productName + " (" + category + ")";

}

}

// Linear search implementation - O(n)

public static Product linearSearch(Product[] products, int targetId) {

for (Product product : products) {

if (product.productId == targetId) {

return product;

}

}

return null;

}

// Binary search implementation - O(log n)

public static Product binarySearch(Product[] sortedProducts, int targetId) {

int left = 0;

int right = sortedProducts.length - 1;

while (left <= right) {

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

if (sortedProducts[mid].productId == targetId) {

return sortedProducts[mid];

}

if (sortedProducts[mid].productId < targetId) {

left = mid + 1;

} else {

right = mid - 1;

}

}

return null;

}

public static void main(String[] args) {

// Create sample product catalog

Product[] products = {

new Product(101, "Wireless Headphones", "Electronics"),

new Product(203, "Organic Cotton T-Shirt", "Clothing"),

new Product(145, "Stainless Steel Water Bottle", "Home"),

new Product(307, "Smart Fitness Tracker", "Electronics"),

new Product(422, "Hardcover Notebook", "Stationery")

};

// Create sorted copy for binary search

Product[] sortedProducts = Arrays.copyOf(products, products.length);

Arrays.sort(sortedProducts, Comparator.comparingInt(p -> p.productId));

System.out.println("=== E-Commerce Product Search ===");

System.out.println("Product Catalog:");

Arrays.stream(products).forEach(System.out::println);

// Test searches

int[] searchIds = {145, 422, 999};

System.out.println("\nLinear Search Results:");

for (int id : searchIds) {

Product result = linearSearch(products, id);

System.out.println("Search for " + id + ": " +

(result != null ? "Found - " + result : "Not found"));

}

System.out.println("\nBinary Search Results:");

for (int id : searchIds) {

Product result = binarySearch(sortedProducts, id);

System.out.println("Search for " + id + ": " +

(result != null ? "Found - " + result : "Not found"));

}

// Performance analysis

System.out.println("\nPerformance Analysis:");

System.out.println("- Linear Search: O(n) time complexity");

System.out.println(" - Best case: O(1) (first element)");

System.out.println(" - Average case: O(n/2)");

System.out.println(" - Worst case: O(n) (last element or not found)");

System.out.println("\n- Binary Search: O(log n) time complexity");

System.out.println(" - Requires sorted data (O(n log n) preprocessing)");

System.out.println(" - Best case: O(1) (middle element)");

System.out.println(" - Average/Worst case: O(log n)");

System.out.println("\nRecommendation:");

System.out.println("For our e-commerce platform with thousands of products:");

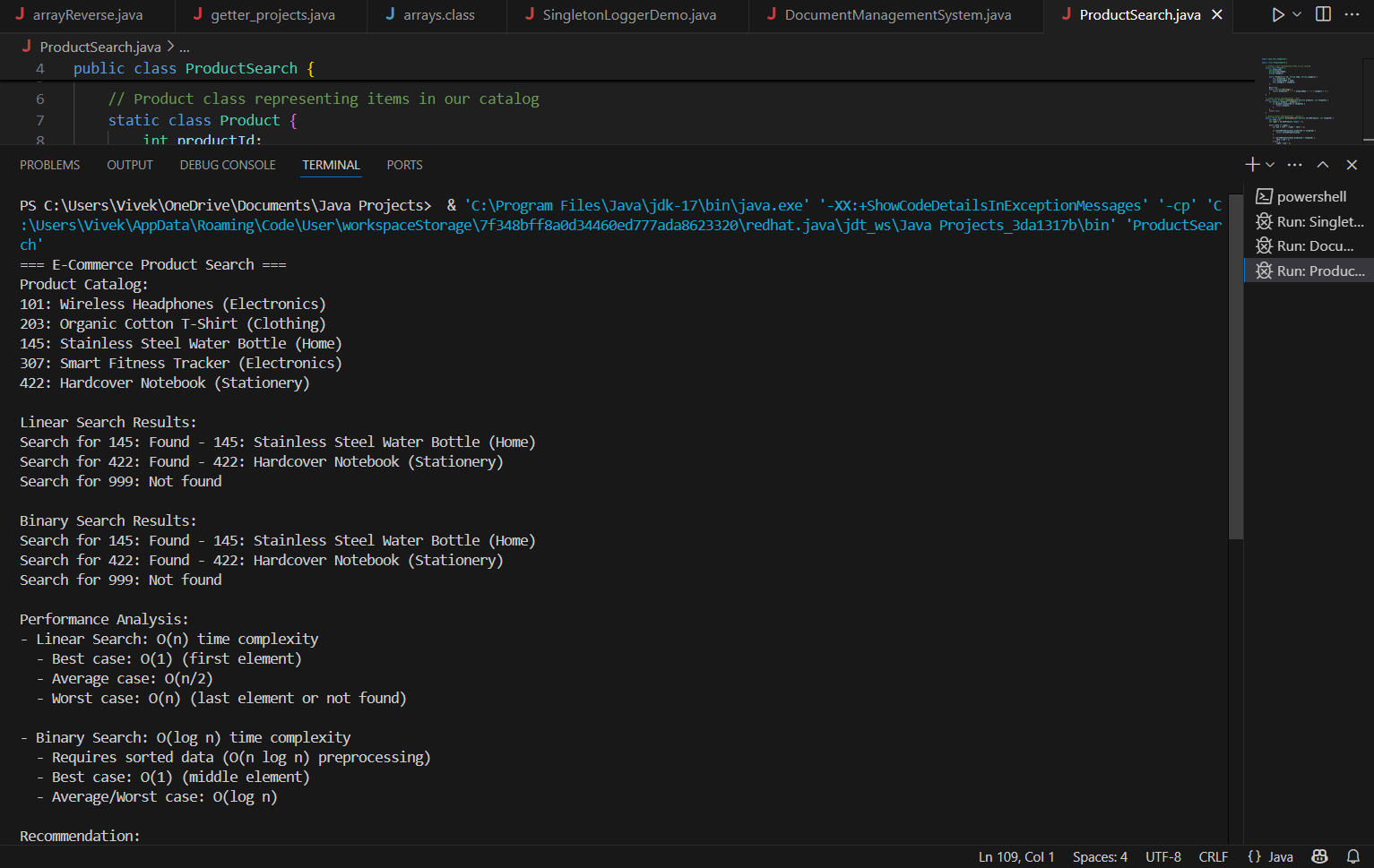
System.out.println("1. Use binary search for product ID lookup (fastest)");

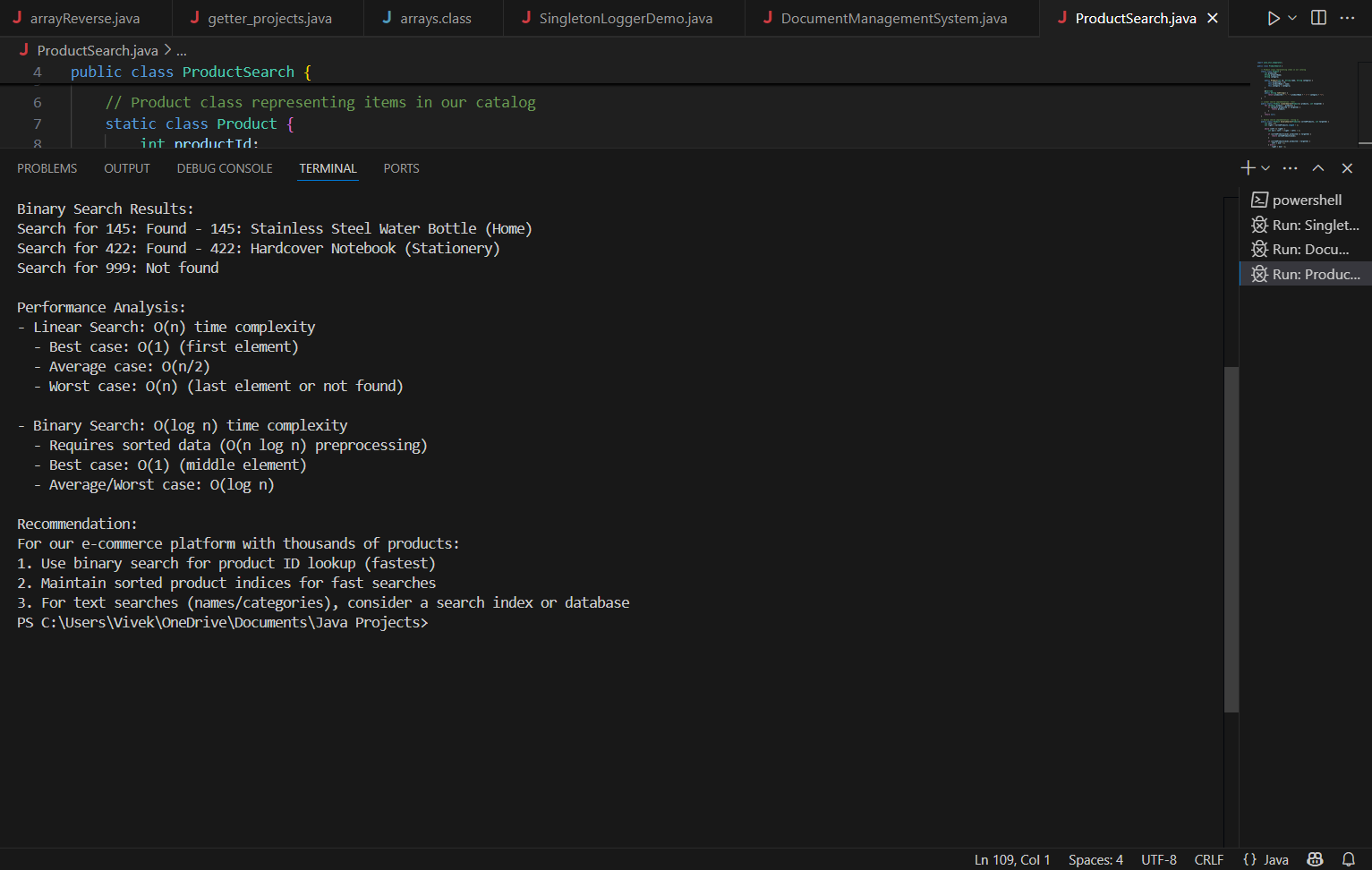
System.out.println("2. Maintain sorted product indices for fast searches");

System.out.println("3. For text searches (names/categories), consider a search index or database");

}

}





Exercise 7: Financial Forecasting

import java.util.Arrays;

import java.util.HashMap;

import java.util.Map;

public class FinancialForecaster {

// Standard recursive approach (O(n) time, O(n) space)

public static double recursiveFutureValue(double currentValue,

double[] growthRates,

int period) {

if (period >= growthRates.length) {

return currentValue;

}

double grownValue = currentValue \* (1 + growthRates[period]);

return recursiveFutureValue(grownValue, growthRates, period + 1);

}

// Optimized tail-recursive version (O(n) time, O(1) space if tail-call optimized)

public static double tailRecursiveFV(double currentValue,

double[] growthRates,

int period,

double accumulator) {

if (period >= growthRates.length) {

return accumulator;

}

return tailRecursiveFV(currentValue,

growthRates,

period + 1,

accumulator \* (1 + growthRates[period]));

}

// Memoized recursive version (O(n) time, O(n) space)

private static Map<String, Double> memo = new HashMap<>();

public static double memoizedFutureValue(double currentValue,

double[] growthRates,

int period) {

String key = currentValue + "-" + period;

if (memo.containsKey(key)) {

return memo.get(key);

}

if (period >= growthRates.length) {

return currentValue;

}

double result = memoizedFutureValue(

currentValue \* (1 + growthRates[period]),

growthRates,

period + 1

);

memo.put(key, result);

return result;

}

// Iterative version (O(n) time, O(1) space)

public static double iterativeFutureValue(double currentValue, double[] growthRates) {

double result = currentValue;

for (double rate : growthRates) {

result \*= (1 + rate);

}

return result;

}

public static void main(String[] args) {

double initialInvestment = 1000.0;

double[] growthRates = {0.05, 0.07, 0.03, 0.08, 0.06};

System.out.println("Financial Forecasting Results:");

System.out.printf("Initial Investment: $%.2f%n", initialInvestment);

System.out.println("Growth Rates: " + Arrays.toString(growthRates));

// Standard recursive

long start = System.nanoTime();

double recursiveResult = recursiveFutureValue(initialInvestment, growthRates, 0);

long recursiveTime = System.nanoTime() - start;

// Tail recursive

start = System.nanoTime();

double tailRecResult = tailRecursiveFV(initialInvestment, growthRates, 0, initialInvestment);

long tailRecTime = System.nanoTime() - start;

// Iterative

start = System.nanoTime();

double iterativeResult = iterativeFutureValue(initialInvestment, growthRates);

long iterativeTime = System.nanoTime() - start;

System.out.println("\nResults:");

System.out.printf("Standard Recursive: $%.2f (%,d ns)%n", recursiveResult, recursiveTime);

System.out.printf("Tail Recursive: $%.2f (%,d ns)%n", tailRecResult, tailRecTime);

System.out.printf("Iterative: $%.2f (%,d ns)%n", iterativeResult, iterativeTime);

System.out.println("\nPerformance Analysis:");

System.out.println("1. Time Complexity: All versions are O(n) for n periods");

System.out.println("2. Space Complexity:");

System.out.println(" - Standard Recursive: O(n) call stack");

System.out.println(" - Tail Recursive: O(1) if optimized, else O(n)");

System.out.println(" - Iterative: O(1) constant space");

System.out.println("\nOptimization Recommendations:");

System.out.println("- For small datasets: Use standard recursion for clarity");

System.out.println("- For large datasets: Use iterative approach");

System.out.println("- For repeated calculations: Use memoization");

System.out.println("- In production: Combine with confidence intervals");

}

}

