**Design principles and patterns**

**Exercise 1: Implementing the Singleton Pattern**

CODE:

class Logger {

private static Logger instance;

private Logger() {

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

}

public static Logger getInstance() {

if (instance == null) {

instance = new Logger();

}

return instance;

}

public void log(String message) {

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

}

}

public class Main {

public static void main(String[] args) {

Logger logger1 = Logger.getInstance();

Logger logger2 = Logger.getInstance();

logger1.log("First message");

logger2.log("Second message");

if (logger1 == logger2) {

System.out.println("Both logger instances are the same (singleton verified).");

} else {

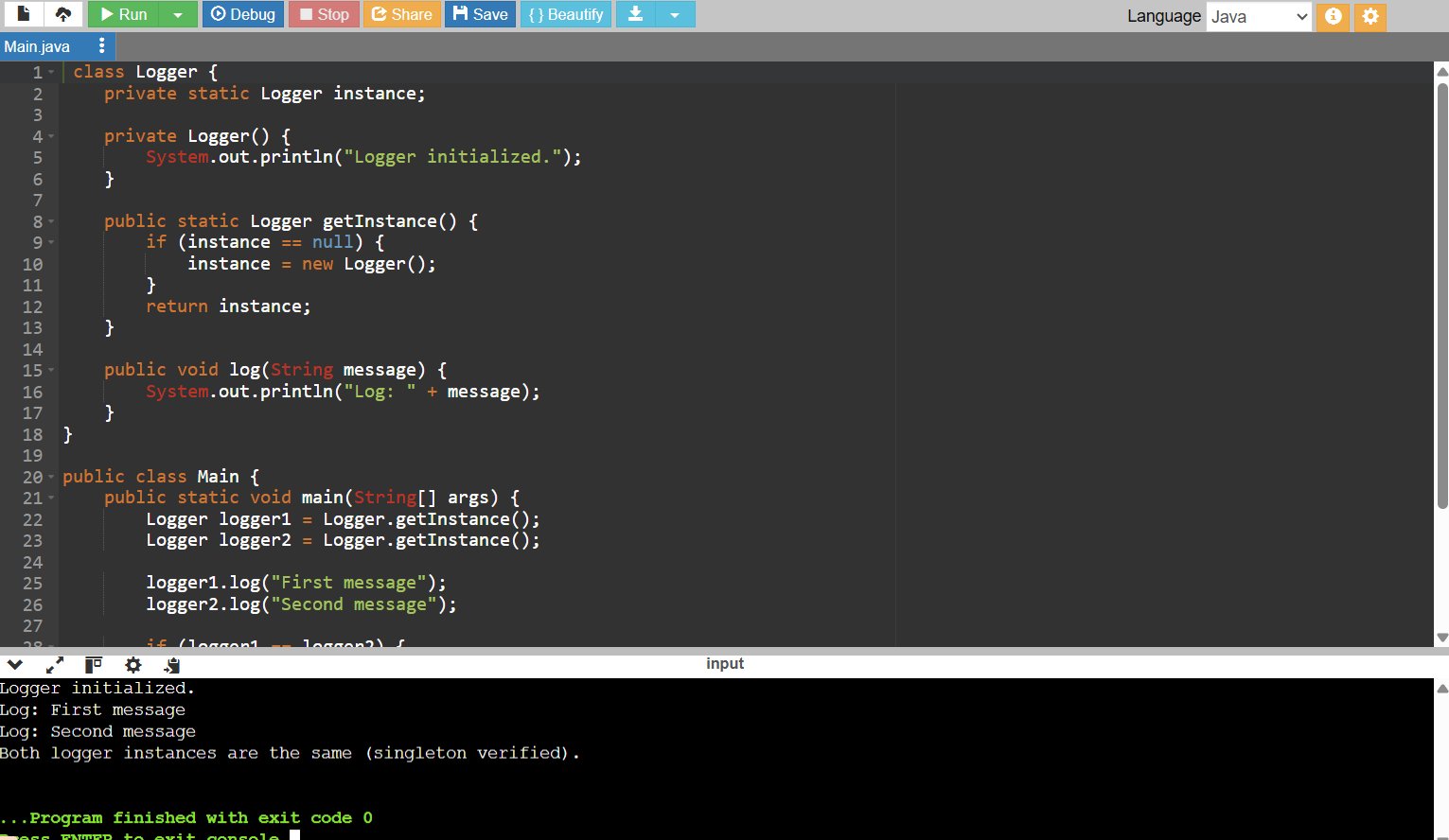
System.out.println("Different instances (singleton violated).");

}

}

}

OUTPUT:



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

CODE:

interface Document {

void open();

}

class WordDocument implements Document {

@Override

public void open() {

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

}}

class PdfDocument implements Document {

@Override

public void open() {

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

}

}

class ExcelDocument implements Document {

@Override

public void open() {

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

}

}

abstract class DocumentFactory {

public abstract Document createDocument();

}

class WordDocumentFactory extends DocumentFactory {

@Override

public Document createDocument() {

return new WordDocument();

}

}

class PdfDocumentFactory extends DocumentFactory {

@Override

public Document createDocument() {

return new PdfDocument();

}

}

class ExcelDocumentFactory extends DocumentFactory {

@Override

public Document createDocument() {

return new ExcelDocument();

}

}

public class Main {

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();

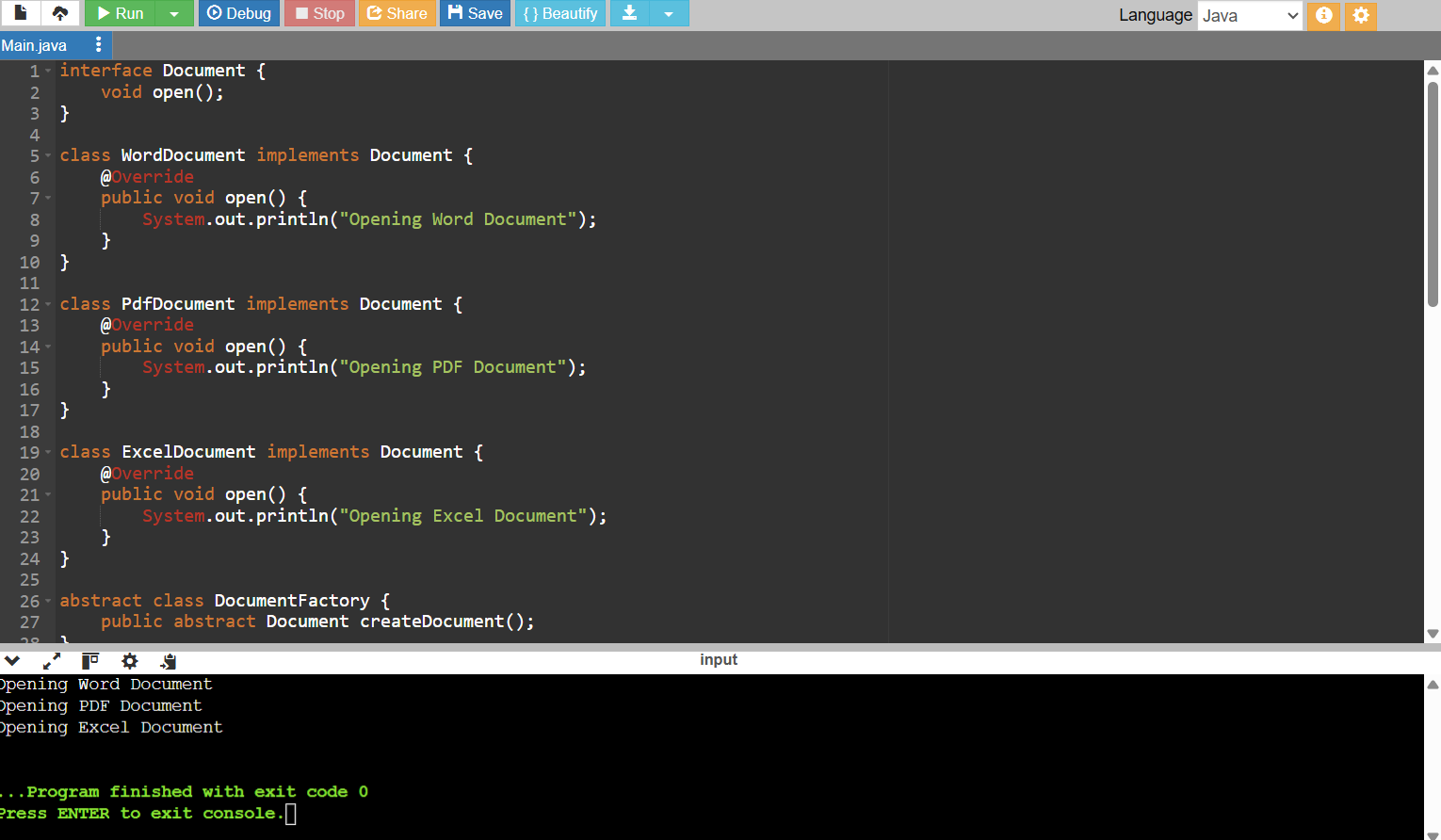
Document excelDoc = excelFactory.createDocument();

excelDoc.open();

}

}

OUTPUT:



**Data structures and Algorithms**

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

CODE:

import java.util.Arrays;

import java.util.Comparator;

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 + "]";

}

}

public class Main {

// Linear Search by productName

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

for (Product p : products) {

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

return p;

}

}

return null;

}

// Binary Search by productName (requires sorted array)

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

int left = 0;

int right = products.length - 1;

while (left <= right) {

int mid = (left + right) / 2;

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

if (cmp == 0) {

return products[mid];

} else if (cmp < 0) {

left = mid + 1;

} else {

right = mid - 1;

}

}

return null;

}

public static void main(String[] args) {

Product[] products = {

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

new Product(102, "Chair", "Furniture"),

new Product(103, "Pen", "Stationery"),

new Product(104, "Mobile", "Electronics"),

new Product(105, "Notebook", "Stationery")

};

// Linear Search

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

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

System.out.println(result1 != null ? result1 : "Product not found");

// Sort before Binary Search

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

// Binary Search

System.out.println("\nBinary Search for 'Pen':");

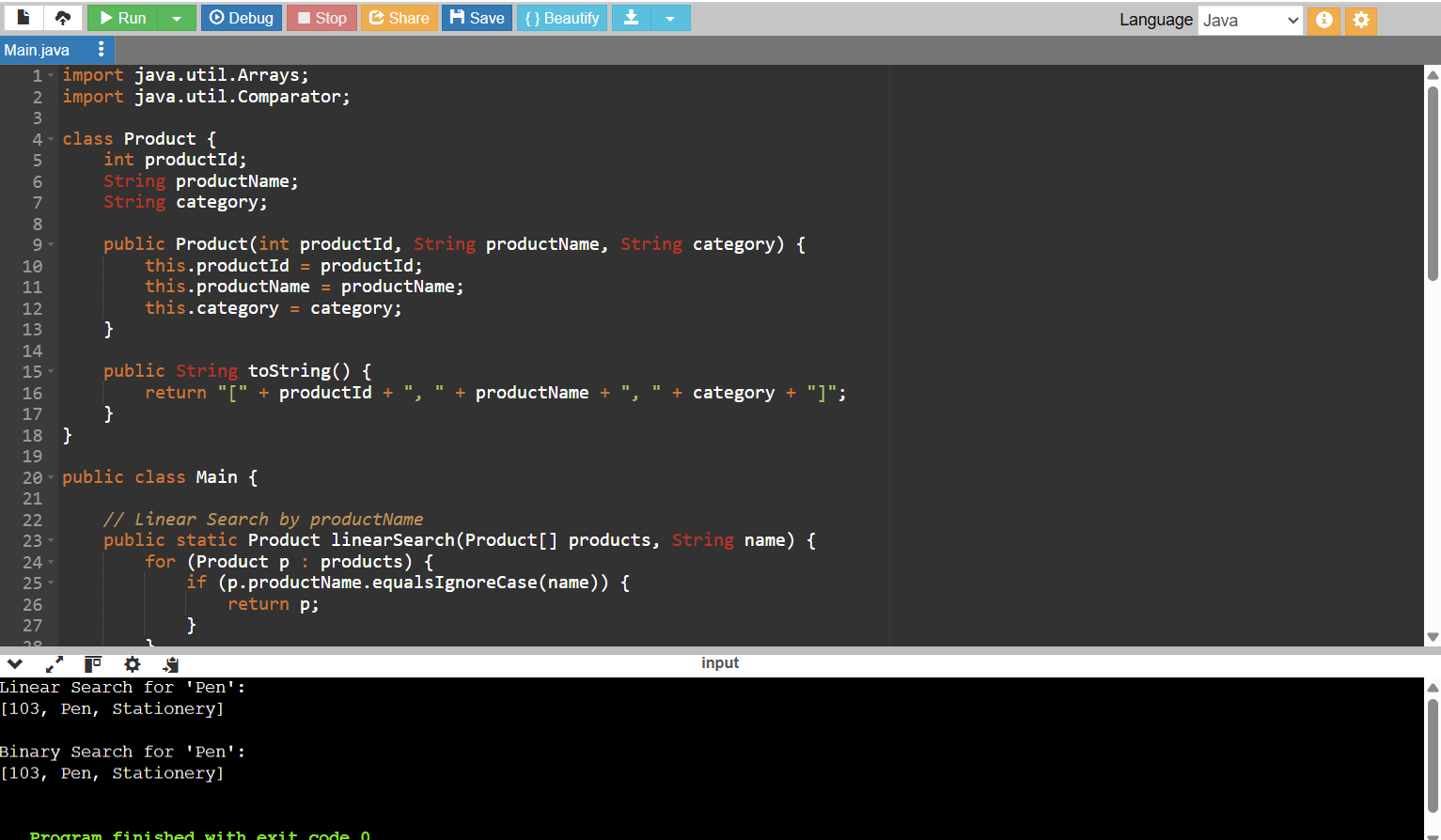
Product result2 = binarySearch(products, "Pen");

System.out.println(result2 != null ? result2 : "Product not found");

}

}

OUTPUT:



**Exercise 7: Financial Forecasting**

CODE:

import java.util.Scanner;

public class Main {

// Recursive forecast

public static double forecastRecursive(double initialValue, double growthRate, int years) {

if (years == 0) {

return initialValue;

}

return forecastRecursive(initialValue, growthRate, years - 1) \* (1 + growthRate);

}

// Iterative forecast

public static double forecastIterative(double initialValue, double growthRate, int years) {

double value = initialValue;

for (int i = 0; i < years; i++) {

value \*= (1 + growthRate);

}

return value;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// User input

System.out.print("Enter initial value: ");

double initialValue = scanner.nextDouble();

System.out.print("Enter annual growth rate (in %): ");

double ratePercent = scanner.nextDouble();

double growthRate = ratePercent / 100.0;

System.out.print("Enter number of years to forecast: ");

int years = scanner.nextInt();

// Choose method

System.out.print("Choose method (1 = Recursive, 2 = Iterative): ");

int choice = scanner.nextInt();

double result = 0;

if (choice == 1) {

result = forecastRecursive(initialValue, growthRate, years);

System.out.println("Calculated using RECURSION.");

} else {

result = forecastIterative(initialValue, growthRate, years);

System.out.println("Calculated using ITERATION.");

}

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

}

}

OUTPUT:

