**NAME**: Nithishkumar R

**SUPER SET ID**: 6376501

**EMAIL ID**: [727822tuec133@skct.edu.in](mailto:727822tuec133@skct.edu.in)

# **WEEK 1 HANDS-ON EXERCISE**

DESIGN PRINCIPLES AND PATTERNS:

Exercise 1: Implementing the Singleton Pattern

CODE:

public class Main {

public static void main(String[] args) {

Logger logger1 = Logger.getInstance();

logger1.log("Starting application");

Logger logger2 = Logger.getInstance();

logger2.log("Application running");

if (logger1 == logger2) {

System.out.println("Both logger1 and logger2 are the same instance.");

} else {

System.out.println("Different instances of Logger detected.");

}

}

}

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;

}

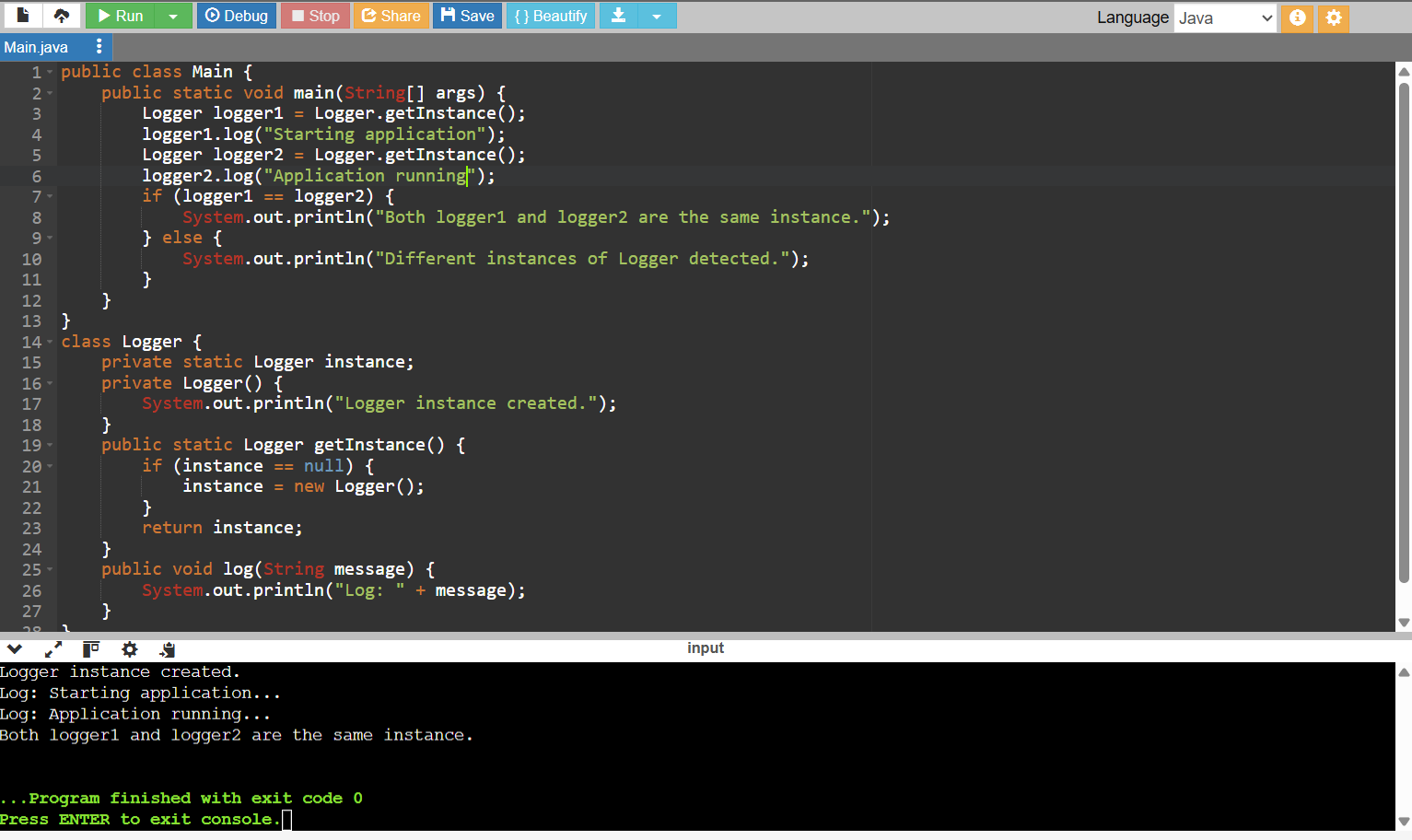
public void log(String message) {

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

}

}

OUTPUT:



Exercise 2: Implementing the Factory Method Pattern

CODE:

public class FactoryMethodPatternTest {

public static void main(String[] args) {

DocumentFactory wordFactory = new WordDocumentFactory();

Document word = wordFactory.createDocument();

word.open();

DocumentFactory pdfFactory = new PdfDocumentFactory();

Document pdf = pdfFactory.createDocument();

pdf.open();

DocumentFactory excelFactory = new ExcelDocumentFactory();

Document excel = excelFactory.createDocument();

excel.open();

}

}

interface Document {

void open();

}

class WordDocument implements Document {

@Override

public void open() {

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

}

}

class PdfDocument implements Document {

@Override

public void open() {

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

}

}

class ExcelDocument implements Document {

@Override

public void open() {

System.out.println("Opening an 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

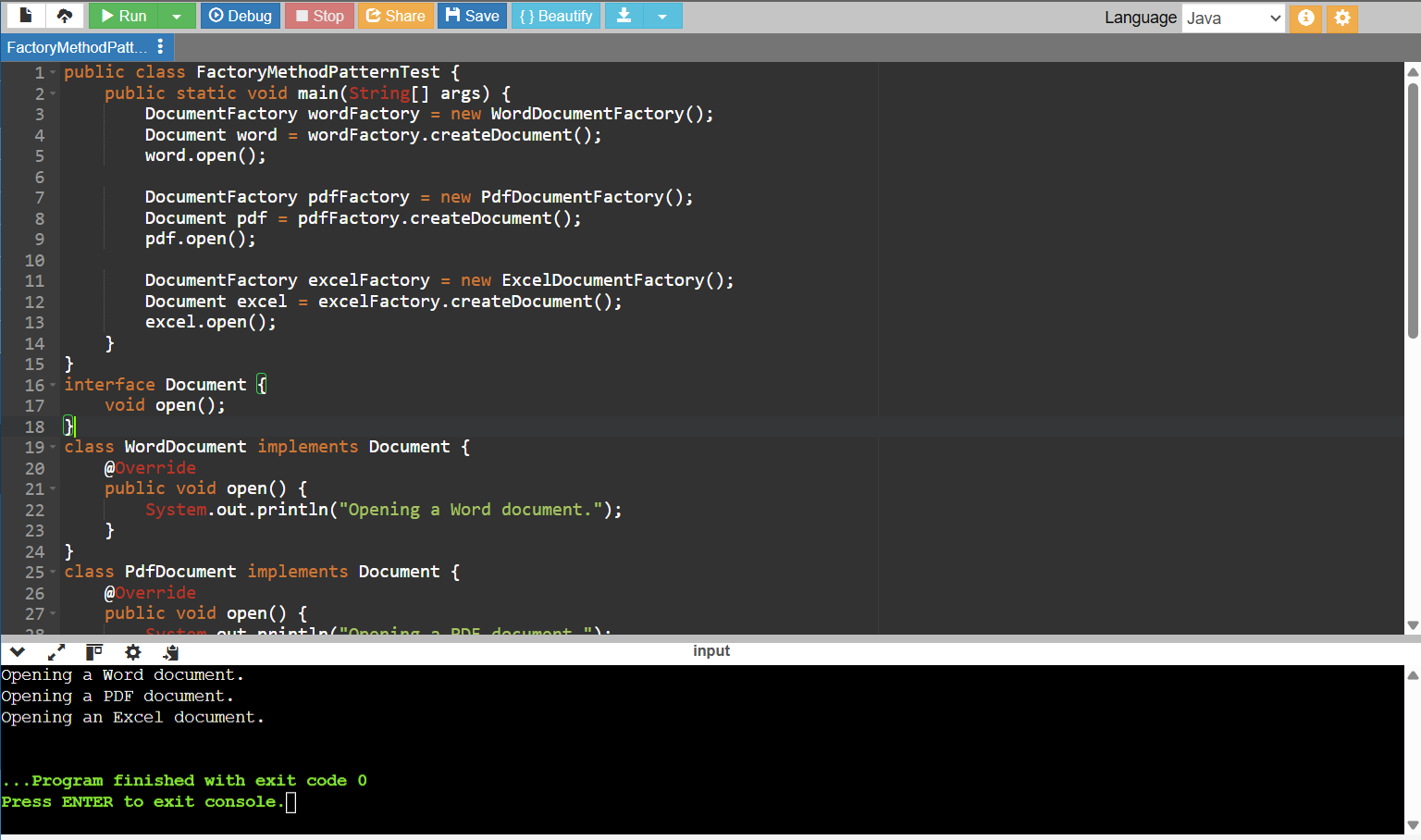
public Document createDocument() {

return new ExcelDocument();

}

}

OUTPUT:



DATA STRUCTURES AND ALGORITHMS:

Exercise 2: E-commerce Platform Search Function

CODE:

import java.util.Arrays;

import java.util.Comparator;

public class SearchTest {

public static void main(String[] args) {

Product[] products = {

new Product(101, "Shoes", "Footwear"),

new Product(102, "Laptop", "Electronics"),

new Product(103, "Watch", "Accessories"),

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

};

Product result1 = LinearSearch.linearSearch(products, "Watch");

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

Product result2 = BinarySearch.binarySearch(products, "Watch");

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

}

}

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;

}

@Override

public String toString() {

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

}

}

class LinearSearch {

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

for (Product p : products) {

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

return p;

}

}

return null;

}

}

class BinarySearch {

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

Arrays.sort(products, Comparator.comparing(p -> p.productName)); // sort by name

int low = 0, high = products.length - 1;

while (low <= high) {

int mid = (low + high) / 2;

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

if (cmp == 0)

return products[mid];

else if (cmp < 0)

high = mid - 1;

else

low = mid + 1;

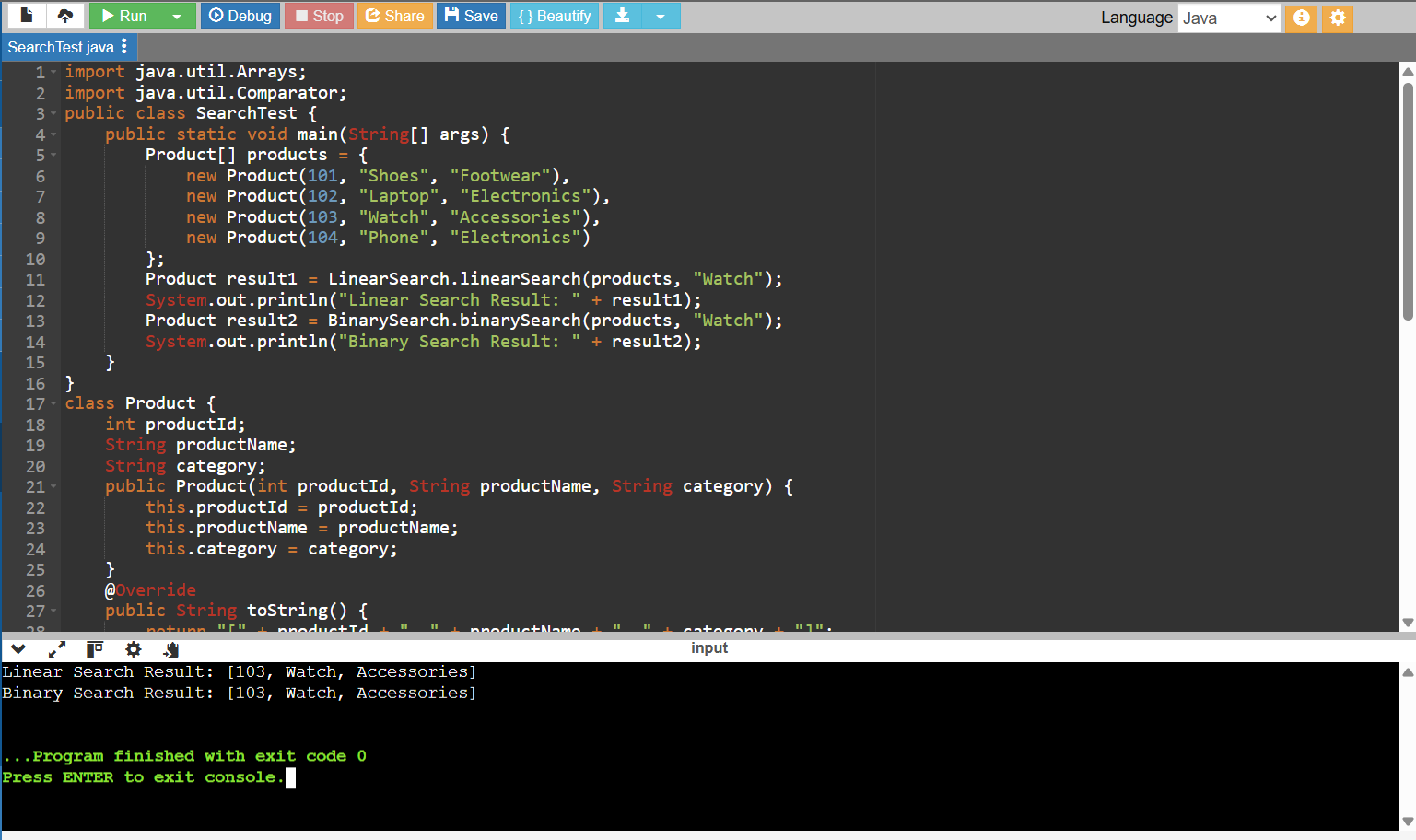
}

return null;

}

}

OUTPUT:



Exercise 7: Financial forecasting

CODE:

public class FinancialForecast {

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

if (years == 0) {

return presentValue;

}

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

}

public static double futureValueIterative(double presentValue, double rate, int years) {

double result = presentValue;

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

result \*= (1 + rate);

}

return result;

}

public static void main(String[] args) {

double presentValue = 10000;

double rate = 0.08;

int years = 5;

double futureRecursive = futureValueRecursive(presentValue, rate, years);

System.out.printf("Future value (Recursive) after %d years: ₹%.2f%n", years, futureRecursive);

double futureIterative = futureValueIterative(presentValue, rate, years);

System.out.printf("Future value (Iterative) after %d years: ₹%.2f%n", years, futureIterative);

}

}

OUTPUT:

