**WEEK – 1**

**Hands-On**

**Design Principles and Patterns**

**Exercise 1: Implementing the Singleton Pattern**

**Code:**

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 class SingletonExample {

public static void main(String[] args) {

Logger logger1 = Logger.getInstance();

logger1.log("Application started");

Logger logger2 = Logger.getInstance();

logger2.log("Processing data...");

if (logger1 == logger2) {

System.out.println("Same logger instance used.");

} else {

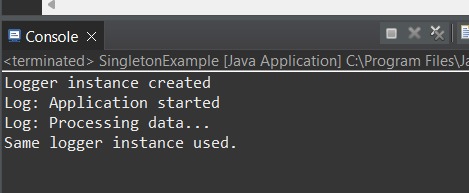
System.out.println("Different logger instances used.");

}

}

}

**Sample output:**



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

**Code:**

interface Document {

void open();

}

class WordDocument implements Document {

public void open() {

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

}

}

class PdfDocument implements Document {

public void open() {

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

}

}

class ExcelDocument implements Document {

public void open() {

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

}

}

abstract class DocumentFactory {

public abstract Document createDocument();

}

class WordFactory extends DocumentFactory {

public Document createDocument() {

return new WordDocument();

}

}

class PdfFactory extends DocumentFactory {

public Document createDocument() {

return new PdfDocument();

}

}

class ExcelFactory extends DocumentFactory {

public Document createDocument() {

return new ExcelDocument();

}

}

public class FactoryMethodExample {

public static void main(String[] args) {

DocumentFactory wordFactory = new WordFactory();

Document word = wordFactory.createDocument();

word.open();

DocumentFactory pdfFactory = new PdfFactory();

Document pdf = pdfFactory.createDocument();

pdf.open();

DocumentFactory excelFactory = new ExcelFactory();

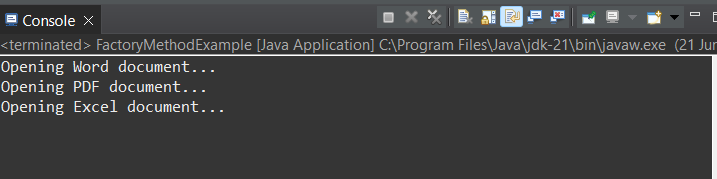
Document excel = excelFactory.createDocument();

excel.open();

}

}

**Sample output:**

****

**Algorithms and Data Structures:**

**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 id, String name, String category) {

this.productId = id;

this.productName = name;

this.category = category;

}

public String toString() {

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

}

}

public class ECommerceSearchExample {

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

for (Product p : products) {

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

return p;

}

}

return null;

}

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

int left = 0, 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(1, "Shoes", "Footwear"),

new Product(2, "Laptop", "Electronics"),

new Product(3, "Pen", "Stationery"),

new Product(4, "Notebook", "Stationery")

};

System.out.println("Linear Search:");

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

System.out.println(result1 != null ? "Found: " + result1 : "Not Found");

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

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

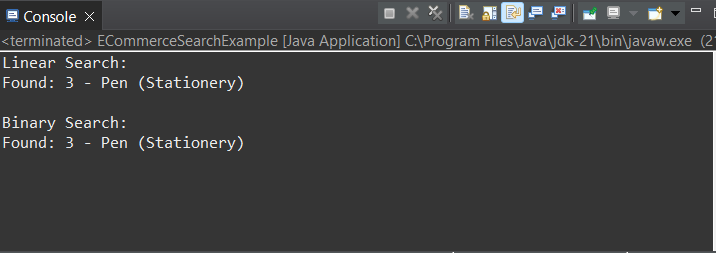
Product result2 = binarySearch(products, "Pen");

System.out.println(result2 != null ? "Found: " + result2 : "Not Found");

}

}

**Sample output:**

****

**Exercise 7: Financial Forecasting**

**Code:**

public class FinancialForecastExample {

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

if (years == 0) return currentValue;

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

}

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

double result = currentValue;

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

result \*= (1 + growthRate);

}

return result;

}

public static void main(String[] args) {

double currentValue = 1000;

double growthRate = 0.1; // 10%

int years = 5;

System.out.println("Recursive Forecast: " +

String.format("%.2f", forecastRecursive(currentValue, growthRate, years)));

System.out.println("Iterative Forecast: " +

String.format("%.2f", forecastIterative(currentValue, growthRate, years)));

}

}

**Sample Output:**

