**SUPERSET ID:** **6419902**

**WEEK – 1**

**Hands-On**

**Design Principles and Patterns**

**Exercise 1: Implementing the Singleton Pattern**

**Code:**

class Logger {

private static Logger instance;

private Logger() {

}

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("Starting the application");

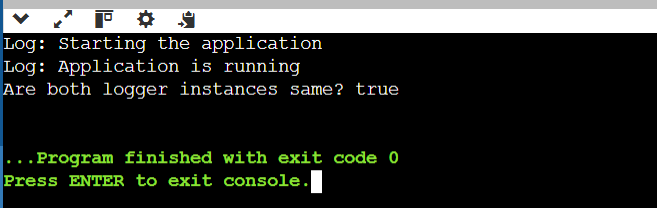
logger2.log("Application is running");

System.out.println("Are both logger instances same? " + (logger1 == logger2));

}

}

**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 {

abstract Document createDocument();

}

class WordDocumentFactory extends DocumentFactory {

Document createDocument() {

return new WordDocument();

}

}

class PdfDocumentFactory extends DocumentFactory {

Document createDocument() {

return new PdfDocument();

}

}

class ExcelDocumentFactory extends DocumentFactory {

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();

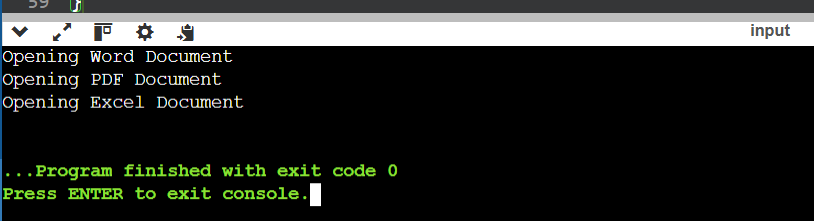
Document excelDoc = excelFactory.createDocument();

excelDoc.open();

}

}

**Sample output:**

****

**Algorithms and Data Structures:**

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

**Code:**

import java.util.Arrays;

class Product implements Comparable<Product> {

int productId;

String productName;

String category;

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

this.productId = productId;

this.productName = productName;

this.category = category;

}

public int compareTo(Product other) {

return this.productName.compareTo(other.productName);

}

public String toString()

{

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

}

}

public class Main {

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

for (int i = 0; i < products.length; i++) {

if (products[i].productName.equalsIgnoreCase(name)) {

return i;

}

}

return -1;

}

public static int 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 mid;

else if (cmp < 0) left = mid + 1;

else right = mid - 1;

}

return -1;

}

public static void main(String[] args) {

Product[] products = {

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

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

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

new Product(104, "Book", "Stationery"),

new Product(105, "Watch", "Accessories")

};

int index1 = linearSearch(products, "Phone");

System.out.println("Linear Search found at index: " + index1);

Arrays.sort(products);

int index2 = binarySearch(products, "Phone");

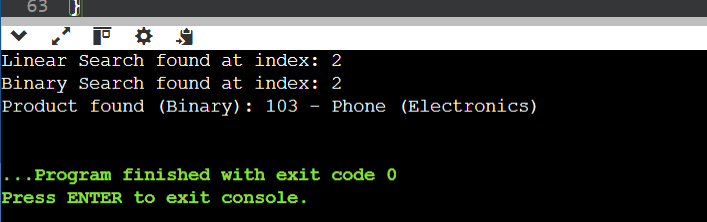
System.out.println("Binary Search found at index: " + index2);

System.out.println("Product found (Binary): " + (index2 >= 0 ? products[index2] : "Not found"));

}

}

**Sample output:**

****

**Exercise 7: Financial Forecasting**

**Code:**

public class Main {

public static double futureValue(double presentValue, double growthRate, int years) {

double result = presentValue;

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

result \*= (1 + growthRate);

}

return result;

}

public static void main(String[] args) {

double presentValue = 1000.0;

double growthRate = 0.05;

int years = 5;

double forecast = futureValue(presentValue, growthRate, years);

System.out.println("Future value after " + years + " years: " + forecast);

}

}

**Sample Output:**

