DESIGN PRINCIPLE & PATTERN

**Exercise 1: Implementing the Singleton Pattern**

CODE:

class Logger {

private static Logger singleInstance;

private Logger() {

System.out.println("Logger Initialized!");

}

public static Logger getInstance() {

if (singleInstance == null) {

singleInstance = new Logger();

}

return singleInstance;

}

public void log(String message) {

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

}

}

public class Main {

public static void main(String[] args) {

Logger logger1 = Logger.getInstance();

logger1.log("First log message");

Logger logger2 = Logger.getInstance();

logger2.log("Second log message");

if (logger1 == logger2) {

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

} else {

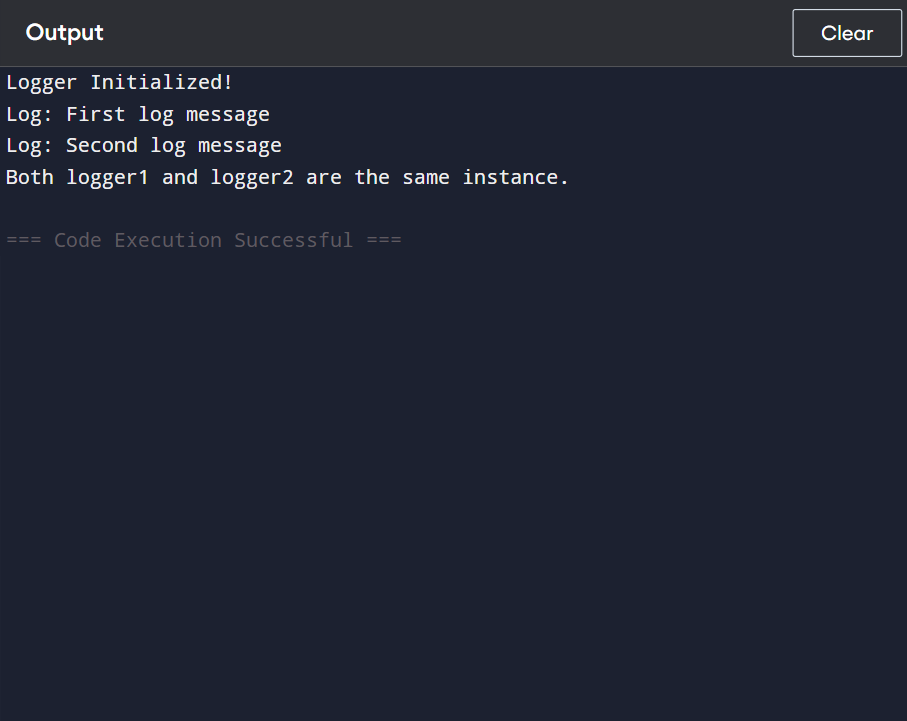
System.out.println("Different instances found! Singleton failed.");

}

}

}

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 WordDocumentFactory extends DocumentFactory {

public Document createDocument() {

return new WordDocument();

}

}

class PdfDocumentFactory extends DocumentFactory {

public Document createDocument() {

return new PdfDocument();

}

}

class ExcelDocumentFactory extends DocumentFactory {

public Document createDocument() {

return new ExcelDocument();

}

}

public class Main {

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();

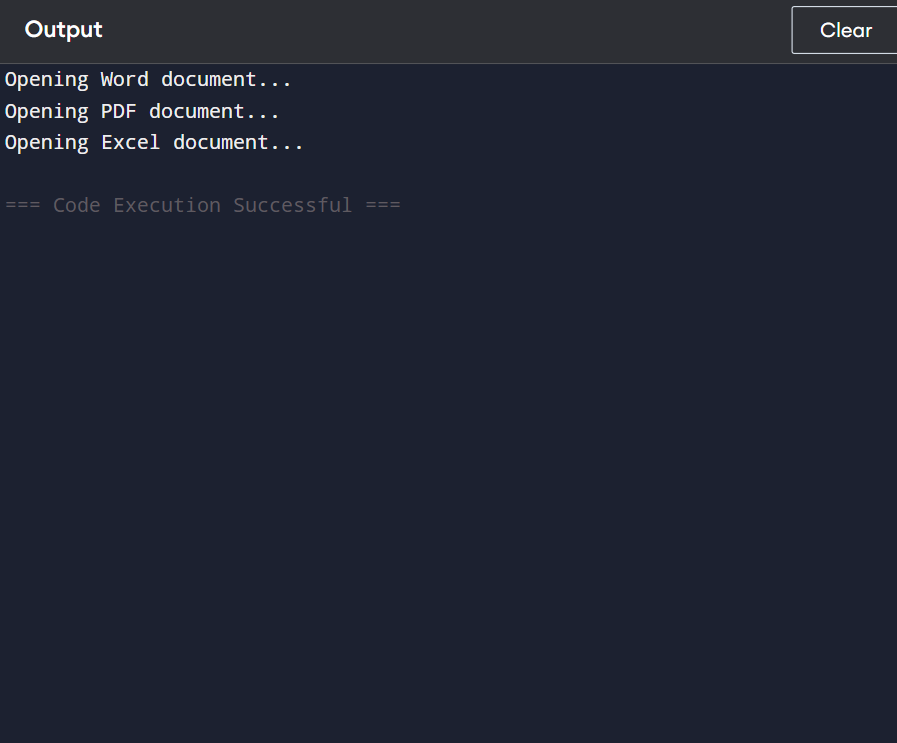
Document excel = excelFactory.createDocument();

excel.open();

}

}

OUTPUT:



DATA STRUCTURE AND ALOGRITHM:

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

CODE:

import java.util.Arrays;

import java.util.Comparator;

class Product {

int productId;

String productName;

String category;

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 {

public static void main(String[] args) {

Product[] products = {

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

new Product(205, "Shoes", "Fashion"),

new Product(143, "Book", "Stationery"),

new Product(312, "Mobile", "Electronics"),

new Product(215, "Watch", "Accessories")

};

Product result1 = linearSearch(products, 312);

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

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

Product result2 = binarySearch(products, 312);

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

}

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

for (Product p : products) {

if (p.productId == targetId) {

return p;

}

}

return null;

}

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

int left = 0, right = products.length - 1;

while (left <= right) {

int mid = (left + right) / 2;

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

return products[mid];

} else if (products[mid].productId < targetId) {

left = mid + 1;

} else {

right = mid - 1;

}

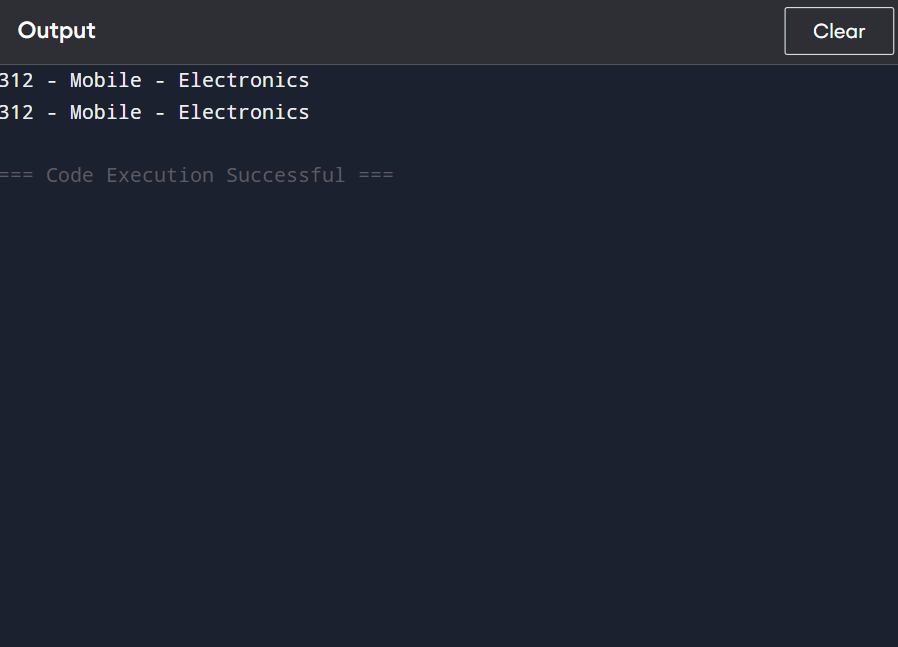
}

return null;

}

}

OUTPUT:



**Exercise 7: Financial Forecasting**

CODE:

import java.util.HashMap;

public class Main {

public static void main(String[] args) {

double initialValue = 1000.0;

double growthRate = 0.05; // 5% annual growth

int years = 5;

double future = forecast(initialValue, growthRate, years);

System.out.printf("Predicted value after %d years: %.2f\n", years, future);

}

public static double forecast(double value, double rate, int year) {

if (year == 0) {

return value;

}

return forecast(value, rate, year - 1) \* (1 + rate);

}

}

OUTPUT:

