**Exercise 1: Implementing the Singleton Pattern**

// Logger.java

public class Logger {

  private static Logger instance;

  private Logger() {

        System.out.println("Logger Instance Created");

    }

public static Logger getInstance() {

        if (instance == null) {

            instance = new Logger(); // Create only once

        }

        return instance;

    }

public void log(String message) {

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

    }

}

// Main.java

public class Main {

    public static void main(String[] args) {

  Logger logger1 = Logger.getInstance();

        Logger logger2 = Logger.getInstance();

logger1.log("First log message");

        logger2.log("Second log message");

if (logger1 == logger2) {

            System.out.println("Both loggers are the same instance (Singleton works!)");

        } else {

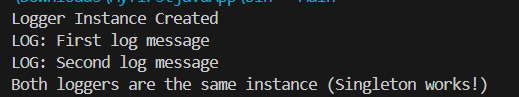
            System.out.println("Different instances (Singleton failed)");

        }

    }

}

**OUTPUT:**



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

// Document interface

interface Document {

void open();

}

// Concrete Document classes

class WordDocument implements Document {

public void open() {

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

}

}

class PdfDocument implements Document {

public void open() {

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

}

}

class ExcelDocument implements Document {

public void open() {

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

}

}

// Abstract Factory

abstract class DocumentFactory {

public abstract Document createDocument();

}

// Concrete Factories

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();

}

}

// Test class

public class Main {

public static void main(String[] args) {

DocumentFactory wordFactory = new WordFactory();

Document wordDoc = wordFactory.createDocument();

wordDoc.open();

DocumentFactory pdfFactory = new PdfFactory();

Document pdfDoc = pdfFactory.createDocument();

pdfDoc.open();

DocumentFactory excelFactory = new ExcelFactory();

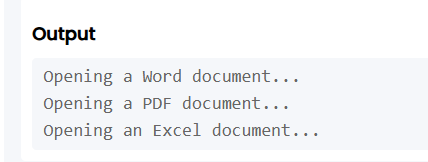
Document excelDoc = excelFactory.createDocument();

excelDoc.open();

}

}

**OUTPUT:**



**Exercise 3: E-commerce Platform Search Function**

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;

}

}

// Search utility class

class SearchUtils {

// Linear search by product name

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

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

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

return i;

}

}

return -1;

}

// Binary search by product name (only on sorted array)

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

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

while (left <= right) {

int mid = (left + right) / 2;

int compare = products[mid].productName.compareToIgnoreCase(targetName);

if (compare == 0) return mid;

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

else right = mid - 1;

}

return -1;

}

}

// Main class to run the search

class EcommerceSearch {

public static void main(String[] args) {

Product[] products = {

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

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

new Product(103, "Book", "Education"),

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

new Product(105, "Shoes", "Footwear")

};

String target = "Book";

// Linear Search

int result1 = SearchUtils.linearSearch(products, target);

System.out.println("Linear Search: " + (result1 != -1 ? "Found at index " + result1 : "Not Found"));

// Sort the array for Binary Search

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

// Binary Search

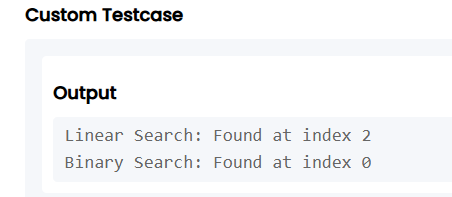
int result2 = SearchUtils.binarySearch(products, target);

System.out.println("Binary Search: " + (result2 != -1 ? "Found at index " + result2 : "Not Found"));

}

}

**OUTPUT:**



**Exercise 4: Financial Forecasting**

class FinancialForecast {

public static double calculateFutureValue(double principal, double rate, int years) {

if (years == 0) {

return principal;

}

return calculateFutureValue(principal, rate, years - 1) \* (1 + rate);

}

public static void main(String[] args) {

double principal = 10000; // Initial amount

double rate = 0.05; // 5% annual growth rate

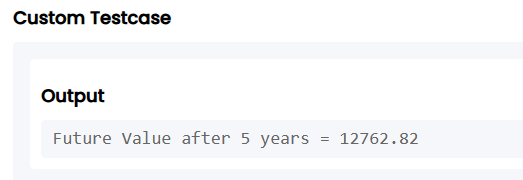
int years = 5; // Forecast for 5 years

double futureValue = calculateFutureValue(principal, rate, years);

System.out.printf("Future Value after %d years = %.2f%n", years, futureValue);

}

}

**OUTPUT:**