**COMPULSORY QUESTIONS**

**DESIGN PATTERNS AND PRINCIPLES**

**1.SINGLETON METHOD**

**CODE:**

import java.util.Scanner;

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 logger = Logger.getInstance();

Scanner scanner = new Scanner(System.in);

System.out.println("Enter number of log messages:");

int count = scanner.nextInt();

scanner.nextLine(); // Consume the leftover newline

for (int i = 1; i <= count; i++) {

System.out.println("Enter log message " + i + ":");

String message = scanner.nextLine();

logger.log(message);

}

Logger anotherLogger = Logger.getInstance();

if (logger == anotherLogger) {

System.out.println("Only one Logger instance used.");

} else {

System.out.println("Multiple Logger instances detected.");

}

}

}

Output:

A screen shot of a computer

AI-generated content may be incorrect.

**2.FACTORY METHOD PATTERN**

**CODE:**

import java.util.Scanner;

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 {

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) {

Scanner scanner = new Scanner(System.in);

System.out.println("Enter document type (word, pdf, excel):");

String input = scanner.nextLine().toLowerCase();

DocumentFactory factory;

switch (input) {

case "word":

factory = new WordDocumentFactory();

break;

case "pdf":

factory = new PdfDocumentFactory();

break;

case "excel":

factory = new ExcelDocumentFactory();

break;

default:

System.out.println("Invalid document type.");

return;

}

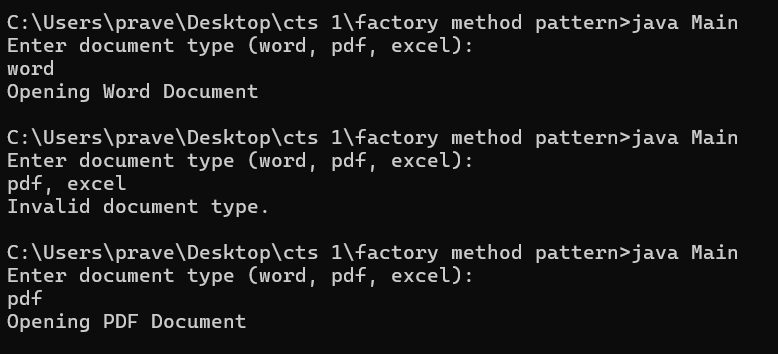
Document document = factory.createDocument();

document.open();

}

}

Output:



**PRACTICE QNS:**

**DESIGN PATTERNS AND PRINCIPLES**

**6.Proxypattern :**

**Code:**

import java.util.\*;

interface Image {

void display();

}

class RealImage implements Image {

private String filename;

public RealImage(String filename) {

this.filename = filename;

loadFromRemoteServer();

}

private void loadFromRemoteServer() {

System.out.println("Loading image from remote server: " + filename);

}

@Override

public void display() {

System.out.println("Displaying image: " + filename);

}

}a

class ProxyImage implements Image {

private String filename;

private RealImage realImage;

public ProxyImage(String filename) {

this.filename = filename;

}

@Override

public void display() {

if (realImage == null) {

realImage = new RealImage(filename); // Lazy initialization

} else {

System.out.println("Using cached image for: " + filename);

}

realImage.display();

}

}

public class ProxyPatternExample {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

Map<String, ProxyImage> imageCache = new HashMap<>();

System.out.println("Image Viewer Application (type 'exit' to quit)");

while (true) {

System.out.print("\nEnter image name to view: ");

String input = scanner.nextLine().trim();

if (input.equalsIgnoreCase("exit")) {

break;

}

ProxyImage image = imageCache.get(input);

if (image == null) {

image = new ProxyImage(input);

imageCache.put(input, image);

}

image.display();

}

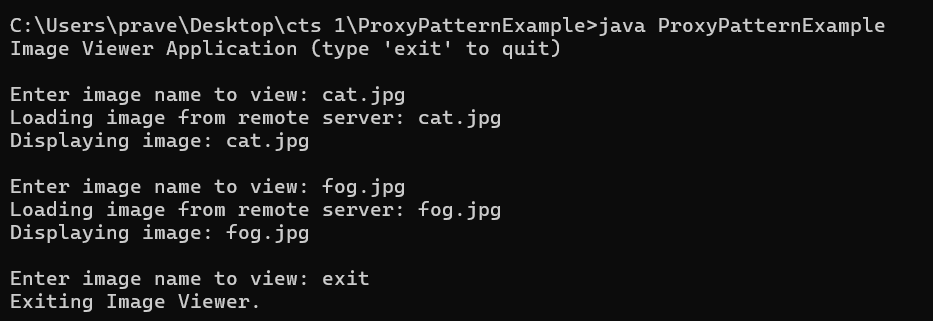
scanner.close();

System.out.println("Exiting Image Viewer.");

}

}

Output:



**COMPULSORY QUESTIONS**

**DATA STRUCTURES AND ALGORITHMS**

**1.E-COMMEREC PLATFORM SEARCH FUNCTION:**

**CODE:**

import java.util.\*;

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 + ")";

}

}

class Search {

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

if (check == 0) return products[mid];

if (check < 0) left = mid + 1;

else right = mid - 1;

}

return null;

}

}

public class Main {

public static void main(String[] args) {

Product[] products = {

new Product(1, "Laptop", "Electronics"),

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

new Product(3, "Shoes", "Footwear"),

new Product(4, "Watch", "Accessories"),

new Product(5, "Tablet", "Electronics")

};

Scanner scan = new Scanner(System.in);

System.out.print("Enter product name to search: ");

String name = scan.nextLine();

Product found1 = Search.linearSearch(products, name);

if (found1 != null) System.out.println("Linear Search: " + found1);

else System.out.println("Linear Search: Not Found");

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

Product found2 = Search.binarySearch(products, name);

if (found2 != null) System.out.println("Binary Search: " + found2);

else System.out.println("Binary Search: Not Found");

}

}

Output:

A computer screen with white text

AI-generated content may be incorrect.

**2.FINANCIAL FORECATING**

**CODE**:

import java.util.Scanner;

public class FinancialForecast {

public static double futureValue(double currentValue, double rate, int years) {

if (years == 0) return currentValue;

return futureValue(currentValue, rate, years - 1) \* (1 + rate);

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter current value (₹): ");

double currentValue = sc.nextDouble();

System.out.print("Enter annual interest rate (e.g., 0.05 for 5%): ");

double rate = sc.nextDouble();

System.out.print("Enter number of years: ");

int years = sc.nextInt();

double result = futureValue(currentValue, rate, years);

System.out.println("Future Value after " + years + " years: ₹" + result);

}

}

Output:

A black background with white text

AI-generated content may be incorrect.

**PRACTICE QNS:**

**3.SORTING CUSTOMERS**

**CODE:**

import java.util.\*;

class Order {

String orderId;

String customerName;

double totalPrice;

public Order(String orderId, String customerName, double totalPrice) {

this.orderId = orderId;

this.customerName = customerName;

this.totalPrice = totalPrice;

}

public String toString() {

return orderId + " - " + customerName + " - $" + totalPrice;

}

}

public class SortOrders {

public static void bubbleSort(Order[] orders) {

int n = orders.length;

for (int i = 0; i < n - 1; i++) {

for (int j = 0; j < n - i - 1; j++) {

if (orders[j].totalPrice > orders[j + 1].totalPrice) {

Order temp = orders[j];

orders[j] = orders[j + 1];

orders[j + 1] = temp;

}

}

}

}

public static void quickSort(Order[] orders, int low, int high) {

if (low < high) {

int pi = partition(orders, low, high);

quickSort(orders, low, pi - 1);

quickSort(orders, pi + 1, high);

}

}

public static int partition(Order[] orders, int low, int high) {

double pivot = orders[high].totalPrice;

int i = low - 1;

for (int j = low; j < high; j++) {

if (orders[j].totalPrice < pivot) {

i++;

Order temp = orders[i];

orders[i] = orders[j];

orders[j] = temp;

}

}

Order temp = orders[i + 1];

orders[i + 1] = orders[high];

orders[high] = temp;

return i + 1;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter number of orders: ");

int n = Integer.parseInt(scanner.nextLine());

Order[] orders1 = new Order[n];

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

System.out.println("Enter details for order " + (i + 1) + ":");

System.out.print("Order ID: ");

String id = scanner.nextLine();

System.out.print("Customer Name: ");

String name = scanner.nextLine();

System.out.print("Total Price: ");

double price = Double.parseDouble(scanner.nextLine());

orders1[i] = new Order(id, name, price);

}

Order[] orders2 = orders1.clone();

System.out.println("\nOriginal Orders:");

for (Order o : orders1) System.out.println(o);

bubbleSort(orders1);

System.out.println("\nSorted by Bubble Sort:");

for (Order o : orders1) System.out.println(o);

quickSort(orders2, 0, orders2.length - 1);

System.out.println("\nSorted by Quick Sort:");

for (Order o : orders2) System.out.println(o);

System.out.println("\nTime Complexity:");

System.out.println("Bubble Sort: O(n^2)");

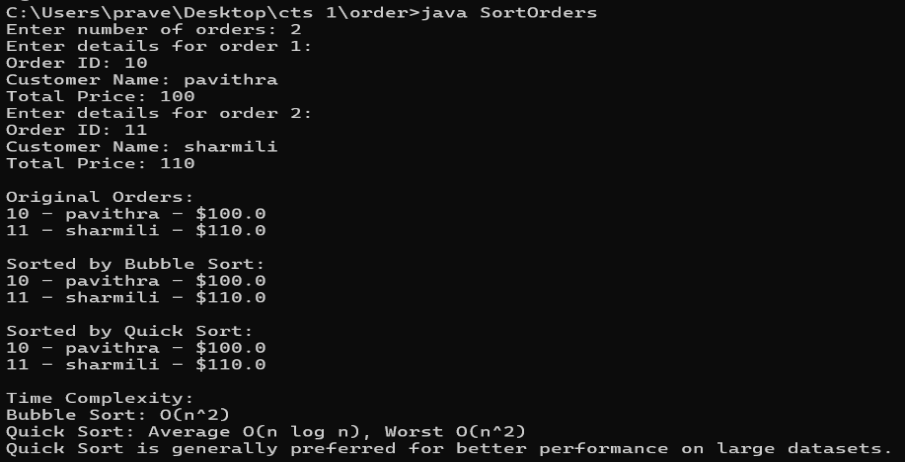
System.out.println("Quick Sort: Average O(n log n), Worst O(n^2)");

System.out.println("Quick Sort is generally preferred for better performance on large datasets.");

}

}

Output:



**4.LIBRARY MANAGEMENT SYSTEM**

**CODE:**

import java.util.\*;

class Book {

int bookId;

String title;

String author;

public Book(int bookId, String title, String author) {

this.bookId = bookId;

this.title = title.toLowerCase(); // normalize for search

this.author = author;

}

@Override

public String toString() {

return "Book ID: " + bookId + ", Title: " + title + ", Author: " + author;

}

}

public class LibraryManagementSystem {

public static Book linearSearchByTitle(List<Book> books, String targetTitle) {

for (Book book : books) {

if (book.title.equalsIgnoreCase(targetTitle)) {

return book;

}

}

return null;

}

public static Book binarySearchByTitle(List<Book> books, String targetTitle) {

int low = 0, high = books.size() - 1;

targetTitle = targetTitle.toLowerCase();

while (low <= high) {

int mid = (low + high) / 2;

int comparison = books.get(mid).title.compareTo(targetTitle);

if (comparison == 0) {

return books.get(mid);

} else if (comparison < 0) {

low = mid + 1;

} else {

high = mid - 1;

}

}

return null;

}

public static void sortBooksByTitle(List<Book> books) {

books.sort(Comparator.comparing(book -> book.title));

}

public static void main(String[] args) {

List<Book> books = new ArrayList<>();

books.add(new Book(101, "Data Structures", "Mark Weiss"));

books.add(new Book(102, "Introduction to Algorithms", "Cormen"));

books.add(new Book(103, "Clean Code", "Robert Martin"));

books.add(new Book(104, "Java Programming", "Herbert Schildt"));

books.add(new Book(105, "Artificial Intelligence", "Stuart Russell"));

Scanner scanner = new Scanner(System.in);

System.out.print("Enter book title to search (linear search): ");

String title1 = scanner.nextLine();

Book foundLinear = linearSearchByTitle(books, title1);

if (foundLinear != null) {

System.out.println("Book found (Linear Search): " + foundLinear);

} else {

System.out.println("Book not found (Linear Search).");

}

sortBooksByTitle(books); // sort before binary search

System.out.print("\nEnter book title to search (binary search): ");

String title2 = scanner.nextLine();

Book foundBinary = binarySearchByTitle(books, title2);

if (foundBinary != null) {

System.out.println("Book found (Binary Search): " + foundBinary);

} else {

System.out.println("Book not found (Binary Search).");

}

}

}

Output:

A black screen with white text

AI-generated content may be incorrect.