**WEEK-1**

**Design pattern and principles:**

**Exercise 1: Implementing the Singleton Pattern**

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();

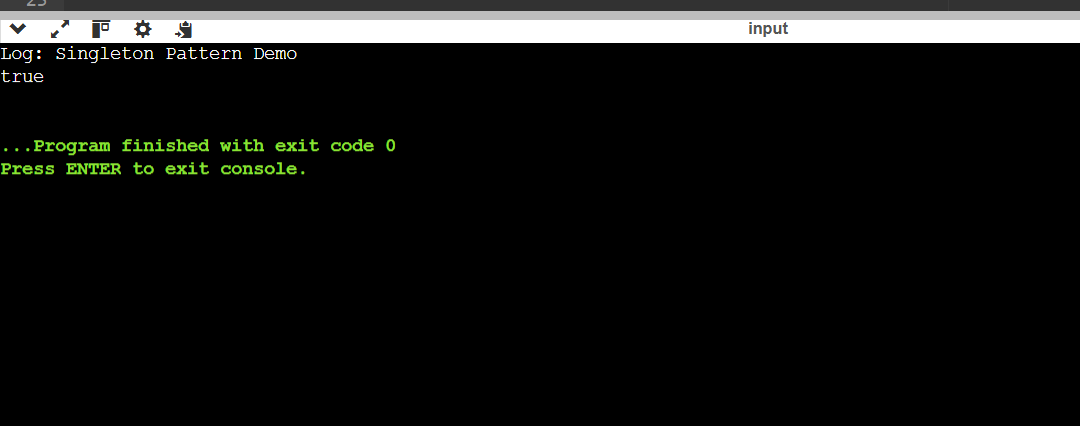
logger1.log("Singleton Pattern Demo");

System.out.println(logger1 == logger2);

}

}

**OUTPUT:**



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

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 {

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) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter document type (word/pdf/excel): ");

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

DocumentFactory factory;

switch (type) {

case "word":

factory = new WordDocumentFactory();

break;

case "pdf":

factory = new PdfDocumentFactory();

break;

case "excel":

factory = new ExcelDocumentFactory();

break;

default:

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

return;

}

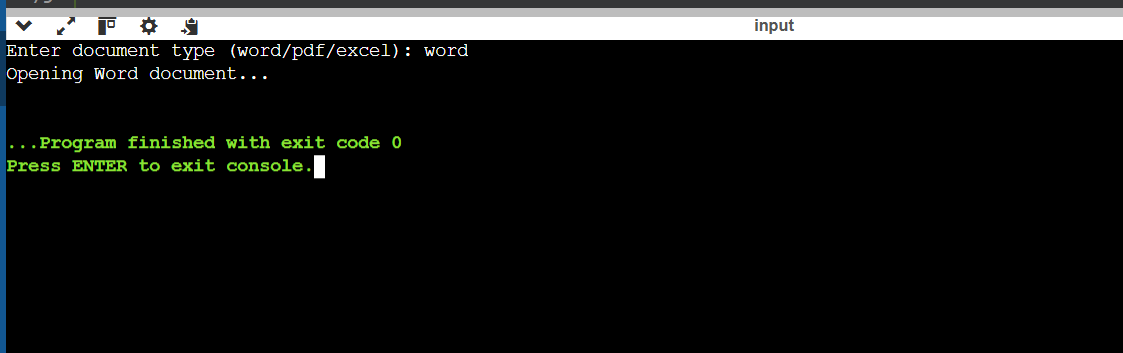
Document doc = factory.createDocument();

doc.open();

}

}

**OUTPUT:**

****

**Data structures and algorithms:**

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

import java.util.Arrays;

import java.util.Comparator;

class Product {

int productId;

String productName;

String category;

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

this.productId = productId;

this.productName = productName;

this.category = category;

}

}

public class Main {

public static void main(String[] args) {

Product[] products = {

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

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

new Product(150, "Mobile", "Electronics"),

new Product(310, "Watch", "Accessories")

};

Product result = linearSearch(products, 150);

if (result != null)

System.out.println("Found (Linear): " + result.productName);

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

result = binarySearch(products, 150);

if (result != null)

System.out.println("Found (Binary): " + result.productName);

}

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;

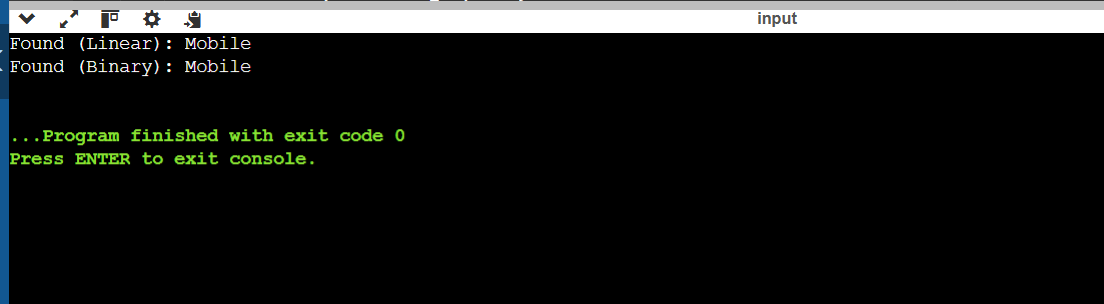
}

return null;

}

}

**OUTPUT:**



**Exercise 7: Financial Forecasting**

public class Main {

public static void main(String[] args) {

double initialAmount = 1000;

double growthRate = 0.10;

int years = 5;

double futureValue = calculateFutureValue(initialAmount, growthRate, years);

System.out.printf("Future value after %d years: ₹%.2f%n", years, futureValue);

}

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

if (years == 0) {

return amount;

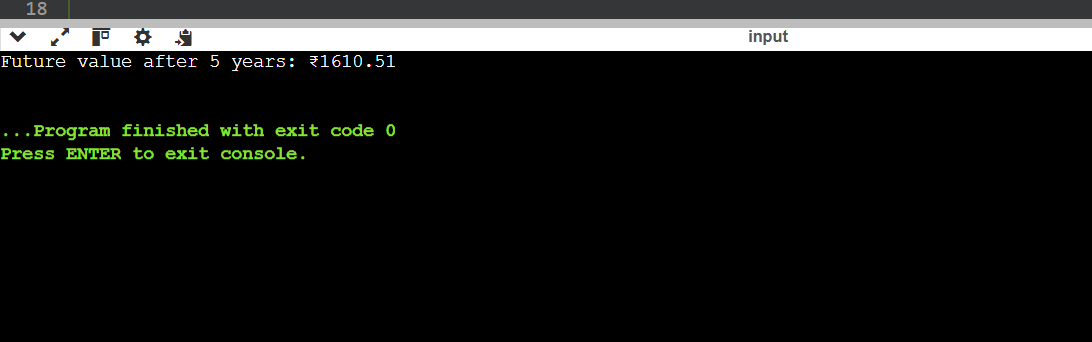
}

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

}

}

**OUTPUT:**



**Additional:**

**Exercise 1: Inventory Management System**

import java.util.HashMap;

class Product {

int productId;

String productName;

int quantity;

double price;

public Product(int productId, String productName, int quantity, double price) {

this.productId = productId;

this.productName = productName;

this.quantity = quantity;

this.price = price;

}

public String toString() {

return productId + " - " + productName + " (" + quantity + " pcs) ₹" + price;

}

}

public class Main {

static HashMap<Integer, Product> inventory = new HashMap<>();

public static void main(String[] args) {

addProduct(101, "Laptop", 10, 50000);

addProduct(102, "Mouse", 50, 499);

updateProduct(102, 45, 450);

deleteProduct(101);

for (Product p : inventory.values()) {

System.out.println(p);

}

}

public static void addProduct(int id, String name, int qty, double price) {

inventory.put(id, new Product(id, name, qty, price));

}

public static void updateProduct(int id, int newQty, double newPrice) {

Product p = inventory.get(id);

if (p != null) {

p.quantity = newQty;

p.price = newPrice;

}

}

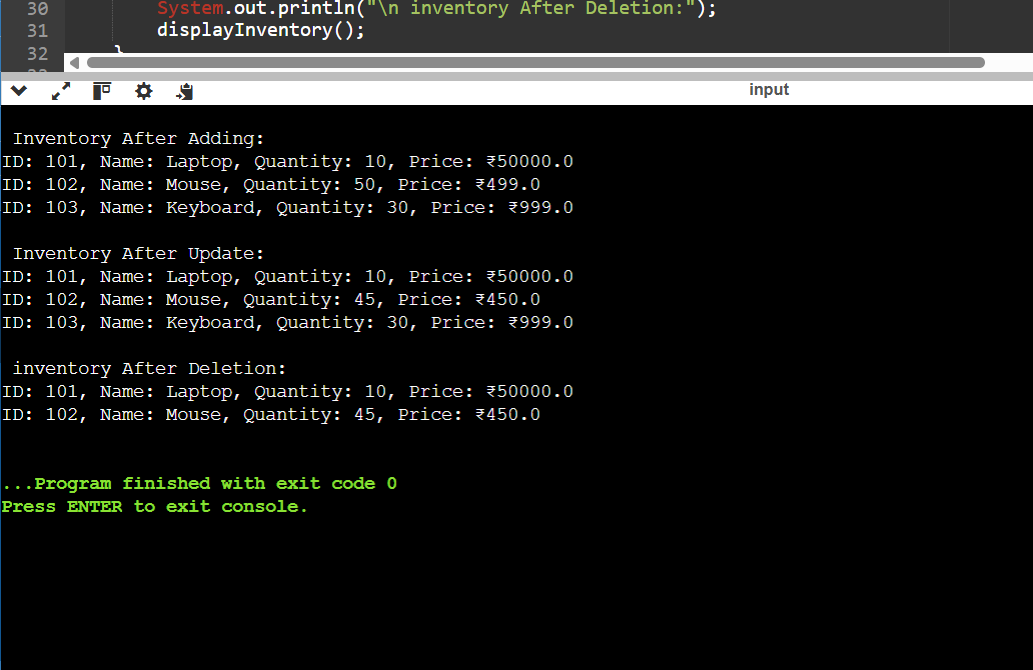
public static void deleteProduct(int id) {

inventory.remove(id);

}

}

**OUTPUT:**



**Exercise 3: Sorting Customer Orders**

class Order {

int orderId;

String customerName;

double totalPrice;

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

this.orderId = orderId;

this.customerName = customerName;

this.totalPrice = totalPrice;

}

public String toString() {

return "Order ID: " + orderId + ", Customer: " + customerName + ", Total: ₹" + totalPrice;

}

}

public class Main {

public static void main(String[] args) {

Order[] orders = {

new Order(101, "Arjun", 1200),

new Order(102, "Priya", 7500),

new Order(103, "Ravi", 2500),

new Order(104, "Neha", 1500)

};

System.out.println("Original Orders:");

printOrders(orders);

Order[] bubbleSorted = orders.clone();

bubbleSort(bubbleSorted);

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

printOrders(bubbleSorted);

Order[] quickSorted = orders.clone();

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

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

printOrders(quickSorted);

}

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 pivotIndex = partition(orders, low, high);

quickSort(orders, low, pivotIndex - 1);

quickSort(orders, pivotIndex + 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 printOrders(Order[] orders) {

for (Order o : orders) {

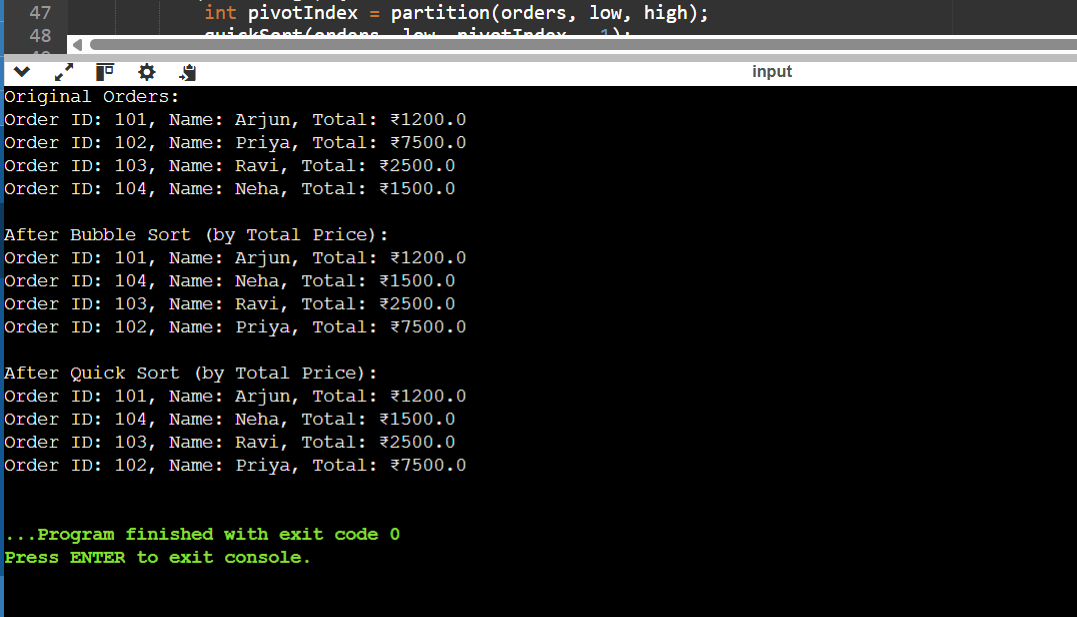
System.out.println(o);

}

}

}

**OUTPUT:**



**Exercise 5: Implementing the Decorator Pattern**

interface Notifier {

void send(String message);

}

class EmailNotifier implements Notifier {

public void send(String message) {

System.out.println("Sending Email: " + message);

}

}

abstract class NotifierDecorator implements Notifier {

protected Notifier notifier;

public NotifierDecorator(Notifier notifier) {

this.notifier = notifier;

}

public void send(String message) {

notifier.send(message);

}

}

class SMSNotifierDecorator extends NotifierDecorator {

public SMSNotifierDecorator(Notifier notifier) {

super(notifier);

}

public void send(String message) {

super.send(message);

System.out.println("Sending SMS: " + message);

}

}

class SlackNotifierDecorator extends NotifierDecorator {

public SlackNotifierDecorator(Notifier notifier) {

super(notifier);

}

public void send(String message) {

super.send(message);

System.out.println("Sending Slack Message: " + message);

}

}

public class Main {

public static void main(String[] args) {

Notifier notifier = new SlackNotifierDecorator(

new SMSNotifierDecorator(

new EmailNotifier()));

notifier.send("Order shipped successfully!");

}

}

**OUTPUT:**

