**1.Implement a Java program to swap two variables without using a third variable.**

import java.util.\*;

public class Swapping {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int var1 = sc.nextInt();

int var2 = sc.nextInt();

System.out.println("Before swapping: var1 = " + var1 + ", var2 = " + var2);

var1 = var1 + var2;

var2 = var1 – var2;

var1 = var1 – var2;

System.out.println("After swapping: var1 = " + var1 + ", var2 = " + var2);

}

}

**2. write a java program that demonstrates type promotion in expressions involving mixed datatypes**

class TypePromotionDemo

{

public static void main(String[] args)

{

int intVal = 10;

float floatVal = 5.5f;

double result1 = intVal + floatVal;

System.out.println("Result of int + float: " + result1);

char charVal = 'A';

int result2 = charVal + intVal;

System.out.println("Result of char + int: " + result2);

byte byteVal = 42;

short shortVal = 100;

int result3 = byteVal + shortVal;

System.out.println("Result of byte + short: " + result3);

long longVal = 1000L;

double doubleVal = 2.5;

double result4 = intVal + floatVal + longVal + doubleVal;

System.out.println("Result of int + float + long + double: " + result4);

}

}

**3. Write a java program to reverse the bits of an integer.**

Import java.util.\*;

class ReverseBits {

public static int reverseBits(int num) {

int result = 0;

for (int i = 0; i < 32; i++) { //since int has 32 bits

result <<= 1; // Shift result left

result |= (num & 1); //get last bit of num and add it to result

num >>= 1; // Shift num right

}

return result;

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int num = sc.nextInt();

int reversedNum = reverseBits(num);

System.out.println("Original Number: " + num);

System.out.println("Reversed Bits: " + reversedNum);

}

}

**4. Write a program to check whether a number is power of two using bitwise operators.**

Import java.util.\*;

class PowerOfTwo {

public static boolean isPowerOfTwo(int num) {

// A number is a power of two if it has only one bit set in its binary representation

return (num > 0) && ((num & (num - 1)) == 0);

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int num = sc.nextInt();

if (isPowerOfTwo(num)) {

System.out.println(num + " is a power of two.");

} else {

System.out.println(num + " is not a power of two.");

}

}

}

**5. Write a Java program to evaluate a mathematical expression given as a string and return the result.**

import java.util.Stack;

public class EvaluateExpression {

public static int evaluate(String expression) {

Stack<Integer> values = new Stack<>();

Stack<Character> operators = new Stack<>();

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

char ch = expression.charAt(i);

if (Character.isDigit(ch)) {

int num = 0;

while (i < expression.length() && Character.isDigit(expression.charAt(i))) {

num = num \* 10 + (expression.charAt(i) - '0');

i++;

}

values.push(num);

i--; // To counteract the last increment

} else if (ch == '(') {

operators.push(ch);

} else if (ch == ')') {

while (!operators.isEmpty() && operators.peek() != '(') {

values.push(applyOperation(operators.pop(), values.pop(), values.pop()));

}

operators.pop(); // Remove '('

} else if (ch == '+' || ch == '-' || ch == '\*' || ch == '/') {

while (!operators.isEmpty() && precedence(ch) <= precedence(operators.peek())) {

values.push(applyOperation(operators.pop(), values.pop(), values.pop()));

}

operators.push(ch);

}

}

while (!operators.isEmpty()) {

values.push(applyOperation(operators.pop(), values.pop(), values.pop()));

}

return values.pop();

}

private static int precedence(char op) {

if (op == '+' || op == '-') return 1;

if (op == '\*' || op == '/') return 2;

return 0;

}

private static int applyOperation(char op, int b, int a) {

switch (op) {

case '+': return a + b;

case '-': return a - b;

case '\*': return a \* b;

case '/': return a / b;

default: return 0;

}

}

public static void main(String[] args) {

String expression = "10 + 2 \* 6";

System.out.println("Result: " + evaluate(expression));

}

}

**6. Implement a Java program to find the sum of digits of a given number using a while loop.**

import java.util.Scanner;

public class SumOfDigits {

public static int sumOfDigits(int num) {

int sum = 0;

while (num != 0) {

sum += num % 10; // Extract last digit and add to sum

num /= 10; // Remove last digit

}

return sum;

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter a number: ");

int num = sc.nextInt();

System.out.println("Sum of digits: " + sumOfDigits(num));

}

}

**7. Implement a Java program that checks whether a given number is a prime number using a for loop.**

import java.util.Scanner;

public class PrimeCheck {

public static boolean isPrime(int num) {

if (num <= 1)

return false;

for (int i = 2; i <= Math.sqrt(num); i++) {

if (num % i == 0) {

return false; // Found a divisor, not a prime

}

}

return true;

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter a number: ");

int num = sc.nextInt();

if (isPrime(num)) {

System.out.println(num + " is a prime number.");

} else {

System.out.println(num + " is not a prime number.");

}

}

}

**Assessment-2:**

1. **Product Price Comparison**

import java.util.Scanner;

class Product {

String name;

double price;

Product(String name, double price) {

this.name = name;

this.price = price;

}

}

public class ProductPriceComparison {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

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

int n = sc.nextInt();

sc.nextLine();

Product[] products = new Product[n];

// Input product details

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

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

String name = sc.nextLine();

System.out.print("Enter price of " + name + ": ");

double price = sc.nextDouble();

sc.nextLine();

products[i] = new Product(name, price);

}

// Find the cheapest and most expensive product

Product cheapest = products[0];

Product expensive = products[0];

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

if (products[i].price < cheapest.price) {

cheapest = products[i];

}

if (products[i].price > expensive.price) {

expensive = products[i];

}

}

// Output results

System.out.println("\nProduct Price Comparison:");

System.out.println("Cheapest Product: " + cheapest.name + " (₹" + cheapest.price + ")");

System.out.println("Most Expensive Product: " + expensive.name + " (₹" + expensive.price + ")");

}

}

1. **Vehicle Inheritance System**
2. **Payment System using Method Overloading**

import java.util.Scanner;

class PaymentSystem {

// Method to process payment using credit card

void processPayment(String cardNumber, String expiryDate, double amount) {

System.out.println("Payment of ₹" + amount + " made using Credit Card: " + cardNumber);

}

// Method to process payment using UPI

void processPayment(String upiId, double amount) {

System.out.println("Payment of ₹" + amount + " made using UPI ID: " + upiId);

}

// Method to process payment using cash

void processPayment(double amount) {

System.out.println("Payment of ₹" + amount + " made using Cash");

}

}

public class PaymentSystemDemo {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

PaymentSystem payment = new PaymentSystem();

System.out.println("Select Payment Method: 1. Credit Card 2. UPI 3. Cash");

int choice = sc.nextInt();

scanner.nextLine(); // Consume newline

switch (choice) {

case 1:

System.out.print("Enter Credit Card Number: ");

String cardNumber = sc.nextLine();

System.out.print("Enter Expiry Date (MM/YY): ");

String expiryDate = sc.nextLine();

System.out.print("Enter Amount: ");

double cardAmount = sc.nextDouble();

payment.processPayment(cardNumber, expiryDate, cardAmount);

break;

case 2:

System.out.print("Enter UPI ID: ");

String upiId = sc.nextLine();

System.out.print("Enter Amount: ");

double upiAmount = sc.nextDouble();

payment.processPayment(upiId, upiAmount);

break;

case 3:

System.out.print("Enter Amount: ");

double cashAmount = sc.nextDouble();

payment.processPayment(cashAmount);

break;

default:

System.out.println("Invalid choice! Please select a valid payment method.");

}

scanner.close();

}

}

1. **Employee Salary Calculation using Method Overriding**

import java.util.Scanner;

class Employee {

String name;

double baseSalary;

Employee(String name, double baseSalary) {

this.name = name;

this.baseSalary = baseSalary;

}

double calculateSalary() {

return baseSalary;

}

void displaySalary() {

System.out.println("Employee: " + name);

System.out.println("Salary: ₹" + calculateSalary());

}

}

class Manager extends Employee {

double bonus;

Manager(String name, double baseSalary, double bonus) {

super(name, baseSalary);

this.bonus = bonus;

}

@Override

double calculateSalary() {

return baseSalary + bonus;

}

}

class Developer extends Employee {

double incentive;

Developer(String name, double baseSalary, double incentive) {

super(name, baseSalary);

this.incentive = incentive;

}

@Override

double calculateSalary() {

return baseSalary + incentive;

}

}

public class EmployeeSalaryCalculation {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter Employee Name: ");

String name = scanner.nextLine();

System.out.print("Enter Base Salary: ");

double baseSalary = scanner.nextDouble();

System.out.print("Enter Employee Type (1 for Manager, 2 for Developer): ");

int type = scanner.nextInt();

Employee employee;

if (type == 1) {

System.out.print("Enter Bonus: ");

double bonus = scanner.nextDouble();

employee = new Manager(name, baseSalary, bonus);

} else if (type == 2) {

System.out.print("Enter Incentive: ");

double incentive = scanner.nextDouble();

employee = new Developer(name, baseSalary, incentive);

} else {

System.out.println("Invalid choice! Defaulting to base salary employee.");

employee = new Employee(name, baseSalary);

}

System.out.println();

employee.displaySalary();

scanner.close();

}

}

1. **Constructor Chaining in a Banking System**

import java.util.Scanner;

class BankAccount {

String accountHolder;

String accountNumber;

double balance;

// Default constructor

BankAccount() {

this("Unknown", "000000", 0.0);

}

// Constructor with account holder and number

BankAccount(String accountHolder, String accountNumber) {

this(accountHolder, accountNumber, 0.0);

}

// Constructor with all parameters (Chaining)

BankAccount(String accountHolder, String accountNumber, double balance) {

this.accountHolder = accountHolder;

this.accountNumber = accountNumber;

this.balance = balance;

}

void displayAccountInfo() {

System.out.println("Account Holder: " + accountHolder);

System.out.println("Account Number: " + accountNumber);

System.out.println("Balance: ₹" + balance);

}

}

public class BankingSystem {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter Account Holder Name: ");

String name = scanner.nextLine();

System.out.print("Enter Account Number: ");

String accountNumber = scanner.nextLine();

System.out.print("Enter Initial Balance: ");

double balance = scanner.nextDouble();

BankAccount userAccount = new BankAccount(name, accountNumber, balance);

System.out.println("\nBank Account Details:");

userAccount.displayAccountInfo();

scanner.close();

}

}

**Assessment-3:**

1. **Advanced Banking System (Checked & Unchecked Exceptions)**

import java.util.Scanner;

// Custom checked exception for insufficient balance

class InsufficientBalanceException extends Exception {

public InsufficientBalanceException(String message) {

super(message);

}

}

// Custom unchecked exception for invalid account number format

class InvalidAccountException extends RuntimeException {

public InvalidAccountException(String message) {

super(message);

}

}

class BankAccount {

private String accountNumber;

private String accountHolder;

private double balance;

public BankAccount(String accountNumber, String accountHolder, double balance) {

if (!accountNumber.matches("\\d{6,10}")) {

throw new InvalidAccountException("Invalid Account Number! It should be 6 to 10 digits.");

}

this.accountNumber = accountNumber;

this.accountHolder = accountHolder;

this.balance = balance;

}

public void deposit(double amount) {

if (amount <= 0) {

throw new IllegalArgumentException("Deposit amount must be positive.");

}

balance += amount;

System.out.println("₹" + amount + " deposited successfully. New balance: ₹" + balance);

}

public void withdraw(double amount) throws InsufficientBalanceException {

if (amount <= 0) {

throw new IllegalArgumentException("Withdrawal amount must be positive.");

}

if (amount > balance) {

throw new InsufficientBalanceException("Insufficient Balance! Available: ₹" + balance);

}

balance -= amount;

System.out.println("₹" + amount + " withdrawn successfully. Remaining balance: ₹" + balance);

}

public void displayAccountInfo() {

System.out.println("Account Holder: " + accountHolder);

System.out.println("Account Number: " + accountNumber);

System.out.println("Balance: ₹" + balance);

}

}

public class AdvancedBankingSystem {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

try {

System.out.print("Enter Account Holder Name: ");

String name = scanner.nextLine();

System.out.print("Enter Account Number: ");

String accountNumber = scanner.nextLine();

System.out.print("Enter Initial Balance: ");

double balance = scanner.nextDouble();

BankAccount account = new BankAccount(accountNumber, name, balance);

account.displayAccountInfo();

System.out.print("Enter amount to deposit: ");

double depositAmount = scanner.nextDouble();

account.deposit(depositAmount);

System.out.print("Enter amount to withdraw: ");

double withdrawAmount = scanner.nextDouble();

account.withdraw(withdrawAmount);

} catch (InvalidAccountException | IllegalArgumentException e) {

System.out.println("Error: " + e.getMessage());

} catch (InsufficientBalanceException e) {

System.out.println("Error: " + e.getMessage());

} finally {

scanner.close();

}

}

}

1. **File Processing with Try-with-Resources & Custom Exception**

import java.io.\*;

import java.util.Scanner;

// Custom checked exception for file errors

class FileProcessingException extends Exception {

public FileProcessingException(String message) {

super(message);

}

}

public class FileProcessor {

// Method to read a file using try-with-resources

public static void readFile(String filePath) throws FileProcessingException {

try (BufferedReader reader = new BufferedReader(new FileReader(filePath))) {

String line;

System.out.println("File Contents:");

while ((line = reader.readLine()) != null) {

System.out.println(line);

}

} catch (FileNotFoundException e) {

throw new FileProcessingException("Error: File not found - " + filePath);

} catch (IOException e) {

throw new FileProcessingException("Error reading file: " + filePath);

}

}

// Method to write to a file using try-with-resources

public static void writeFile(String filePath, String content) throws FileProcessingException {

try (BufferedWriter writer = new BufferedWriter(new FileWriter(filePath))) {

writer.write(content);

System.out.println("Content written successfully to " + filePath);

} catch (IOException e) {

throw new FileProcessingException("Error writing to file: " + filePath);

}

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

try {

System.out.print("Enter file path: ");

String filePath = scanner.nextLine();

System.out.print("Enter content to write to file: ");

String content = scanner.nextLine();

writeFile(filePath, content);

readFile(filePath);

} catch (FileProcessingException e) {

System.out.println(e.getMessage());

} finally {

scanner.close();

}

}

}

1. **Multi-Catch Block: E-Commerce System**

import java.util.Scanner;

// Custom checked exception for invalid product quantity

class InvalidQuantityException extends Exception {

public InvalidQuantityException(String message) {

super(message);

}

}

public class ECommerceSystem {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

try {

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

String productName = scanner.nextLine();

System.out.print("Enter price of the product: ");

double price = scanner.nextDouble();

System.out.print("Enter quantity to purchase: ");

int quantity = scanner.nextInt();

if (quantity <= 0) {

throw new InvalidQuantityException("Quantity must be greater than zero.");

}

double totalCost = price \* quantity;

System.out.println("Order Summary:");

System.out.println("Product: " + productName);

System.out.println("Price per unit: ₹" + price);

System.out.println("Quantity: " + quantity);

System.out.println("Total Cost: ₹" + totalCost);

} catch (InvalidQuantityException e) {

System.out.println("Error: " + e.getMessage());

} catch (ArithmeticException | NumberFormatException e) {

System.out.println("Invalid input: " + e.getMessage());

} catch (Exception e) {

System.out.println("An unexpected error occurred: " + e.getMessage());

} finally {

scanner.close();

}

}

}

**Assessment-4:**

1. **Employee Management System (List, Set, Map, Comparator)**

import java.util.\*;

class Employee {

int id;

String name;

double salary;

public Employee(int id, String name, double salary) {

this.id = id;

this.name = name;

this.salary = salary;

}

@Override

public String toString() {

return "ID: " + id + ", Name: " + name + ", Salary: ₹" + salary;

}

}

class SalaryComparator implements Comparator<Employee> {

@Override

public int compare(Employee e1, Employee e2) {

return Double.compare(e1.salary, e2.salary);

}

}

public class EmployeeManagementSystem {

public static void main(String[] args) {

List<Employee> employeeList = new ArrayList<>();

Set<Employee> employeeSet = new HashSet<>();

Map<Integer, Employee> employeeMap = new HashMap<>();

// Adding employees

Employee e1 = new Employee(101, "Alice", 50000);

Employee e2 = new Employee(102, "Bob", 60000);

Employee e3 = new Employee(103, "Charlie", 55000);

employeeList.add(e1);

employeeList.add(e2);

employeeList.add(e3);

employeeSet.add(e1);

employeeSet.add(e2);

employeeSet.add(e3);

employeeMap.put(e1.id, e1);

employeeMap.put(e2.id, e2);

employeeMap.put(e3.id, e3);

// Sorting employees by salary

Collections.sort(employeeList, new SalaryComparator());

// Displaying employees

System.out.println("Employees (Sorted by Salary):");

for (Employee emp : employeeList) {

System.out.println(emp);

}

System.out.println("\nEmployees in Set:");

for (Employee emp : employeeSet) {

System.out.println(emp);

}

System.out.println("\nEmployees in Map:");

for (Map.Entry<Integer, Employee> entry : employeeMap.entrySet()) {

System.out.println("Key: " + entry.getKey() + ", Value: " + entry.getValue());

}

}

}

1. **Flight Ticket Booking System (Queue, PriorityQueue, Comparator)**

import java.util.\*;

class Ticket {

String passengerName;

String flightNumber;

int priority;

public Ticket(String passengerName, String flightNumber, int priority) {

this.passengerName = passengerName;

this.flightNumber = flightNumber;

this.priority = priority;

}

@Override

public String toString() {

return "Passenger: " + passengerName + ", Flight: " + flightNumber + ", Priority: " + priority;

}

}

class TicketComparator implements Comparator<Ticket> {

@Override

public int compare(Ticket t1, Ticket t2) {

return Integer.compare(t1.priority, t2.priority);

}

}

public class FlightTicketBookingSystem {

public static void main(String[] args) {

Queue<Ticket> ticketQueue = new LinkedList<>();

PriorityQueue<Ticket> priorityQueue = new PriorityQueue<>(new TicketComparator());

// Adding tickets

Ticket t1 = new Ticket("Alice", "AA101", 2);

Ticket t2 = new Ticket("Bob", "BB202", 1);

Ticket t3 = new Ticket("Charlie", "CC303", 3);

ticketQueue.add(t1);

ticketQueue.add(t2);

ticketQueue.add(t3);

priorityQueue.add(t1);

priorityQueue.add(t2);

priorityQueue.add(t3);

// Processing tickets in FIFO order

System.out.println("Processing tickets in Queue (FIFO Order):");

while (!ticketQueue.isEmpty()) {

System.out.println(ticketQueue.poll());

}

// Processing tickets in priority order

System.out.println("\nProcessing tickets in PriorityQueue (Priority Order):");

while (!priorityQueue.isEmpty()) {

System.out.println(priorityQueue.poll());

}

}

}

1. **E-Commerce Product Catalog (TreeMap, HashSet, Comparator)**

import java.util.\*;

class Product {

int id;

String name;

double price;

public Product(int id, String name, double price) {

this.id = id;

this.name = name;

this.price = price;

}

@Override

public String toString() {

return "ID: " + id + ", Name: " + name + ", Price: ₹" + price;

}

}

class PriceComparator implements Comparator<Product> {

@Override

public int compare(Product p1, Product p2) {

return Double.compare(p1.price, p2.price);

}

}

public class ECommerceProductCatalog {

public static void main(String[] args) {

TreeMap<Integer, Product> productCatalog = new TreeMap<>();

HashSet<Product> productSet = new HashSet<>();

List<Product> productList = new ArrayList<>();

// Adding products

Product p1 = new Product(101, "Laptop", 75000);

Product p2 = new Product(102, "Smartphone", 50000);

Product p3 = new Product(103, "Headphones", 3000);

productCatalog.put(p1.id, p1);

productCatalog.put(p2.id, p2);

productCatalog.put(p3.id, p3);

productSet.add(p1);

productSet.add(p2);

productSet.add(p3);

productList.add(p1);

productList.add(p2);

productList.add(p3);

// Sorting by price

Collections.sort(productList, new PriceComparator());

// Displaying products

System.out.println("Products sorted by ID (TreeMap):");

for (Map.Entry<Integer, Product> entry : productCatalog.entrySet()) {

System.out.println(entry.getValue());

}

System.out.println("\nUnique Products (HashSet):");

for (Product product : productSet) {

System.out.println(product);

}

System.out.println("\nProducts sorted by Price (List with Comparator):");

for (Product product : productList) {

System.out.println(product);

}

}

}

**4. University Course Enrollment System (LinkedList, Iterator, ListIterator)**

import java.util.\*;

class Course {

String courseCode;

String courseName;

public Course(String courseCode, String courseName) {

this.courseCode = courseCode;

this.courseName = courseName;

}

@Override

public String toString() {

return "Course Code: " + courseCode + ", Course Name: " + courseName;

}

}

public class UniversityCourseEnrollment {

public static void main(String[] args) {

LinkedList<Course> courseList = new LinkedList<>();

// Adding courses

courseList.add(new Course("CS101", "Data Structures"));

courseList.add(new Course("CS102", "Algorithms"));

courseList.add(new Course("CS103", "Artificial Intelligence"));

// Using Iterator to display courses

System.out.println("Available Courses:");

Iterator<Course> iterator = courseList.iterator();

while (iterator.hasNext()) {

System.out.println(iterator.next());

}

// Using ListIterator to traverse in both directions

System.out.println("\nCourses in Reverse Order:");

ListIterator<Course> listIterator = courseList.listIterator(courseList.size());

while (listIterator.hasPrevious()) {

System.out.println(listIterator.previous());

}

}

}

**Assessment-5:**

1. **Employee Attendance Tracker (Streams, Optional, Date/Time API)**

import java.time.LocalDate;

import java.util.\*;

import java.util.stream.Collectors;

class Employee {

int id;

String name;

LocalDate attendanceDate;

public Employee(int id, String name, LocalDate attendanceDate) {

this.id = id;

this.name = name;

this.attendanceDate = attendanceDate;

}

@Override

public String toString() {

return "ID: " + id + ", Name: " + name + ", Date: " + attendanceDate;

}

}

public class EmployeeAttendanceTracker {

public static void main(String[] args) {

List<Employee> attendanceList = new ArrayList<>();

// Adding attendance records

attendanceList.add(new Employee(101, "Alice", LocalDate.now()));

attendanceList.add(new Employee(102, "Bob", LocalDate.now()));

attendanceList.add(new Employee(103, "Charlie", LocalDate.now().minusDays(1)));

// Filtering attendance for today

List<Employee> todayAttendance = attendanceList.stream()

.filter(emp -> emp.attendanceDate.equals(LocalDate.now()))

.collect(Collectors.toList());

// Finding an employee by ID using Optional

Optional<Employee> searchEmployee = attendanceList.stream()

.filter(emp -> emp.id == 102)

.findFirst();

// Display attendance

System.out.println("Today's Attendance:");

todayAttendance.forEach(System.out::println);

// Display search result

System.out.println("\nSearch Employee with ID 102:");

searchEmployee.ifPresent(System.out::println);

}

}

1. **Movie Streaming Recommendation System (Lambda, Streams, Comparator)**

import java.util.\*;

import java.util.stream.Collectors;

class Movie {

String title;

String genre;

double rating;

public Movie(String title, String genre, double rating) {

this.title = title;

this.genre = genre;

this.rating = rating;

}

@Override

public String toString() {

return "Title: " + title + ", Genre: " + genre + ", Rating: " + rating;

}

}

public class MovieStreamingRecommendation {

public static void main(String[] args) {

List<Movie> movieList = new ArrayList<>();

// Adding movies

movieList.add(new Movie("Inception", "Sci-Fi", 8.8));

movieList.add(new Movie("Interstellar", "Sci-Fi", 8.6));

movieList.add(new Movie("The Dark Knight", "Action", 9.0));

movieList.add(new Movie("Forrest Gump", "Drama", 8.8));

// Filtering by genre using Streams and Lambda

String preferredGenre = "Sci-Fi";

List<Movie> filteredMovies = movieList.stream()

.filter(movie -> movie.genre.equalsIgnoreCase(preferredGenre))

.collect(Collectors.toList());

// Sorting by rating using Comparator and Lambda

List<Movie> sortedMovies = movieList.stream()

.sorted(Comparator.comparingDouble(Movie::getRating).reversed())

.collect(Collectors.toList());

// Display recommendations

System.out.println("Movies in Genre: " + preferredGenre);

filteredMovies.forEach(System.out::println);

System.out.println("\nTop Rated Movies:");

sortedMovies.forEach(System.out::println);

}

}

1. **Stock Market Data Analysis (Streams, Aggregation, Optional)**

import java.util.\*;

import java.util.stream.Collectors;

class Stock {

String symbol;

double price;

double change;

public Stock(String symbol, double price, double change) {

this.symbol = symbol;

this.price = price;

this.change = change;

}

@Override

public String toString() {

return "Symbol: " + symbol + ", Price: ₹" + price + ", Change: " + change + "%";

}

}

public class StockMarketAnalysis {

public static void main(String[] args) {

List<Stock> stockList = new ArrayList<>();

// Adding stock data

stockList.add(new Stock("AAPL", 175.50, 1.2));

stockList.add(new Stock("GOOGL", 2800.75, -0.5));

stockList.add(new Stock("TSLA", 950.30, 3.1));

stockList.add(new Stock("AMZN", 3300.40, -1.8));

// Filtering positive gainers using Streams

List<Stock> gainers = stockList.stream()

.filter(stock -> stock.change > 0)

.collect(Collectors.toList());

// Finding stock with the highest price using Optional

Optional<Stock> highestPricedStock = stockList.stream()

.max(Comparator.comparingDouble(stock -> stock.price));

// Calculating average stock price using Aggregation

double avgPrice = stockList.stream()

.mapToDouble(stock -> stock.price)

.average()

.orElse(0.0);

// Display results

System.out.println("Top Gainers:");

gainers.forEach(System.out::println);

System.out.println("\nStock with Highest Price:");

highestPricedStock.ifPresent(System.out::println);

System.out.println("\nAverage Stock Price: ₹" + avgPrice);

}

}

**Assessment-6:**

1. **Secure User Data Storage System (File Handling, Streams, Encryption)**

import java.io.\*;

import java.nio.charset.StandardCharsets;

import java.nio.file.\*;

import java.security.\*;

import java.util.Base64;

import javax.crypto.\*;

import javax.crypto.spec.SecretKeySpec;

class SecureStorage {

private static final String KEY = "MySecretKey12345"; // 16-char secret key

private static final String FILE\_PATH = "user\_data.txt";

// Encrypt data using AES

public static String encrypt(String data) throws Exception {

SecretKeySpec secretKey = new SecretKeySpec(KEY.getBytes(StandardCharsets.UTF\_8), "AES");

Cipher cipher = Cipher.getInstance("AES");

cipher.init(Cipher.ENCRYPT\_MODE, secretKey);

byte[] encryptedData = cipher.doFinal(data.getBytes(StandardCharsets.UTF\_8));

return Base64.getEncoder().encodeToString(encryptedData);

}

// Decrypt data using AES

public static String decrypt(String encryptedData) throws Exception {

SecretKeySpec secretKey = new SecretKeySpec(KEY.getBytes(StandardCharsets.UTF\_8), "AES");

Cipher cipher = Cipher.getInstance("AES");

cipher.init(Cipher.DECRYPT\_MODE, secretKey);

byte[] decodedData = Base64.getDecoder().decode(encryptedData);

return new String(cipher.doFinal(decodedData), StandardCharsets.UTF\_8);

}

// Save encrypted user data to file

public static void saveUserData(String username, String password) {

try (BufferedWriter writer = Files.newBufferedWriter(Paths.get(FILE\_PATH), StandardCharsets.UTF\_8, StandardOpenOption.CREATE, StandardOpenOption.APPEND)) {

String encryptedPassword = encrypt(password);

writer.write(username + "," + encryptedPassword);

writer.newLine();

System.out.println("User data saved securely.");

} catch (Exception e) {

e.printStackTrace();

}

}

// Retrieve and decrypt user data from file

public static void readUserData() {

try (BufferedReader reader = Files.newBufferedReader(Paths.get(FILE\_PATH), StandardCharsets.UTF\_8)) {

System.out.println("Stored User Data:");

reader.lines().forEach(line -> {

String[] parts = line.split(",");

if (parts.length == 2) {

try {

String decryptedPassword = decrypt(parts[1]);

System.out.println("Username: " + parts[0] + ", Password: " + decryptedPassword);

} catch (Exception e) {

e.printStackTrace();

}

}

});

} catch (IOException e) {

e.printStackTrace();

}

}

}

public class SecureUserDataStorage {

public static void main(String[] args) {

SecureStorage.saveUserData("Alice", "password123");

SecureStorage.saveUserData("Bob", "securePass456");

SecureStorage.readUserData();

}

}

1. **Large File Processor (Buffered Streams, Multi-threading)**

import java.io.\*;

import java.util.concurrent.ExecutorService;

import java.util.concurrent.Executors;

class FileProcessor implements Runnable {

private final String filePath;

public FileProcessor(String filePath) {

this.filePath = filePath;

}

@Override

public void run() {

try (BufferedReader reader = new BufferedReader(new FileReader(filePath))) {

String line;

while ((line = reader.readLine()) != null) {

processLine(line);

}

} catch (IOException e) {

e.printStackTrace();

}

}

private void processLine(String line) {

// Simulate processing (e.g., counting words, parsing data)

System.out.println(Thread.currentThread().getName() + " processed: " + line);

}

}

public class LargeFileProcessor {

public static void main(String[] args) {

String filePath = "large\_data.txt"; // Replace with actual large file

int numThreads = 4; // Number of threads for parallel processing

ExecutorService executor = Executors.newFixedThreadPool(numThreads);

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

executor.execute(new FileProcessor(filePath));

}

executor.shutdown();

}

}

1. **Employee Database Management (Serialization, Object Streams)**

import java.io.\*;

import java.util.\*;

class Employee implements Serializable {

private static final long serialVersionUID = 1L;

int id;

String name;

double salary;

public Employee(int id, String name, double salary) {

this.id = id;

this.name = name;

this.salary = salary;

}

@Override

public String toString() {

return "ID: " + id + ", Name: " + name + ", Salary: ₹" + salary;

}

}

public class EmployeeDatabaseManagement {

private static final String FILE\_PATH = "employees.dat";

public static void saveEmployees(List<Employee> employees) {

try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(FILE\_PATH))) {

oos.writeObject(employees);

System.out.println("Employee data saved successfully.");

} catch (IOException e) {

e.printStackTrace();

}

}

public static List<Employee> loadEmployees() {

try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(FILE\_PATH))) {

return (List<Employee>) ois.readObject();

} catch (IOException | ClassNotFoundException e) {

System.out.println("No existing data found.");

return new ArrayList<>();

}

}

public static void main(String[] args) {

List<Employee> employees = loadEmployees();

employees.add(new Employee(101, "Alice", 50000));

employees.add(new Employee(102, "Bob", 60000));

employees.add(new Employee(103, "Charlie", 55000));

saveEmployees(employees);

System.out.println("\nEmployee Database:");

employees.forEach(System.out::println);

}

}

**Assessment-7:**

1. **Memory Leak Detector and Fixer (Garbage Collection, JVM Memory Model)**

import java.lang.ref.\*;

import java.util.\*;

class MemoryLeakExample {

private List<byte[]> memoryLeakList = new ArrayList<>();

public void createLeak() {

for (int i = 0; i < 10; i++) {

memoryLeakList.add(new byte[1024 \* 1024]); // Allocating 1MB blocks

System.out.println("Allocated " + (i + 1) + "MB");

}

}

}

public class MemoryLeakDetector {

public static void main(String[] args) {

ReferenceQueue<Object> referenceQueue = new ReferenceQueue<>();

List<WeakReference<Object>> weakReferences = new ArrayList<>();

for (int i = 0; i < 5; i++) {

Object obj = new Object();

weakReferences.add(new WeakReference<>(obj, referenceQueue));

}

System.gc(); // Request Garbage Collection

// Detect memory leaks

System.out.println("Checking for memory leaks...");

WeakReference<?> ref;

while ((ref = (WeakReference<?>) referenceQueue.poll()) != null) {

weakReferences.remove(ref);

System.out.println("Memory leak fixed: Object collected by GC");

}

// Simulating a memory leak scenario

MemoryLeakExample leak = new MemoryLeakExample();

leak.createLeak();

// Fix: Nullify and run GC

leak = null;

System.gc();

System.out.println("Forced GC after nullifying memory leak object.");

}

}

**2. Multi-threaded Order Processing System (Threads, Synchronization)**

import java.util.concurrent.\*;

class OrderQueue {

private final BlockingQueue<String> orders = new LinkedBlockingQueue<>();

public void placeOrder(String order) throws InterruptedException {

synchronized (this) {

orders.put(order);

System.out.println("Order Placed: " + order);

}

}

public String processOrder() throws InterruptedException {

synchronized (this) {

String order = orders.take();

System.out.println("Processing Order: " + order);

return order;

}

}

}

class OrderProducer implements Runnable {

private final OrderQueue orderQueue;

public OrderProducer(OrderQueue orderQueue) {

this.orderQueue = orderQueue;

}

@Override

public void run() {

try {

for (int i = 1; i <= 5; i++) {

orderQueue.placeOrder("Order " + i);

Thread.sleep(500);

}

} catch (InterruptedException e) {

Thread.currentThread().interrupt();

}

}

}

class OrderConsumer implements Runnable {

private final OrderQueue orderQueue;

public OrderConsumer(OrderQueue orderQueue) {

this.orderQueue = orderQueue;

}

@Override

public void run() {

try {

for (int i = 1; i <= 5; i++) {

orderQueue.processOrder();

Thread.sleep(1000);

}

} catch (InterruptedException e) {

Thread.currentThread().interrupt();

}

}

}

public class MultiThreadedOrderProcessing {

public static void main(String[] args) {

OrderQueue orderQueue = new OrderQueue();

Thread producerThread = new Thread(new OrderProducer(orderQueue));

Thread consumerThread = new Thread(new OrderConsumer(orderQueue));

producerThread.start();

consumerThread.start();

try {

producerThread.join();

consumerThread.join();

} catch (InterruptedException e) {

Thread.currentThread().interrupt();

}

}

}

**3.Flight Ticket Booking System (Concurrency Utilities, ExecutorService, Future)**

import java.util.concurrent.\*;

class FlightBookingTask implements Callable<String> {

private final String passengerName;

private final String flightNumber;

public FlightBookingTask(String passengerName, String flightNumber) {

this.passengerName = passengerName;

this.flightNumber = flightNumber;

}

@Override

public String call() throws Exception {

Thread.sleep(2000); // Simulate booking delay

return "Booking confirmed for " + passengerName + " on Flight " + flightNumber;

}

}

public class FlightTicketBookingSystem {

public static void main(String[] args) {

ExecutorService executor = Executors.newFixedThreadPool(3);

Future<String> booking1 = executor.submit(new FlightBookingTask("Alice", "AI-101"));

Future<String> booking2 = executor.submit(new FlightBookingTask("Bob", "BA-202"));

Future<String> booking3 = executor.submit(new FlightBookingTask("Charlie", "LH-303"));

try {

System.out.println(booking1.get());

System.out.println(booking2.get());

System.out.println(booking3.get());

} catch (InterruptedException | ExecutionException e) {

e.printStackTrace();

}

executor.shutdown();

}

}

**Assessment-8**

1. **Debugging a Multi-Threaded Race Condition (Breakpoints, Step Execution)**

import java.util.concurrent.atomic.AtomicInteger;

class SharedResource {

private int counter = 0;

private final AtomicInteger atomicCounter = new AtomicInteger(0);

public void increment() {

counter++; // Non-thread-safe increment

atomicCounter.incrementAndGet(); // Thread-safe increment

}

public int getCounter() {

return counter;

}

public int getAtomicCounter() {

return atomicCounter.get();

}

}

class WorkerThread extends Thread {

private final SharedResource resource;

public WorkerThread(SharedResource resource) {

this.resource = resource;

}

@Override

public void run() {

for (int i = 0; i < 1000; i++) {

resource.increment();

}

}

}

public class RaceConditionDebugging {

public static void main(String[] args) throws InterruptedException {

SharedResource resource = new SharedResource();

Thread t1 = new WorkerThread(resource);

Thread t2 = new WorkerThread(resource);

t1.start();

t2.start();

t1.join();

t2.join();

System.out.println("Non-thread-safe Counter: " + resource.getCounter()); // May produce incorrect result

System.out.println("Thread-safe Atomic Counter: " + resource.getAtomicCounter()); // Correct result

}

}

1. **Testing a REST API Service (JUnit, Assertions, Mocking)**

import static org.junit.jupiter.api.Assertions.\*;

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.Test;

import org.mockito.InjectMocks;

import org.mockito.Mock;

import org.mockito.MockitoAnnotations;

import java.util.\*;

class UserService {

private final UserRepository userRepository;

public UserService(UserRepository userRepository) {

this.userRepository = userRepository;

}

public User getUserById(int id) {

return userRepository.findById(id).orElse(null);

}

}

interface UserRepository {

Optional<User> findById(int id);

}

class User {

int id;

String name;

public User(int id, String name) {

this.id = id;

this.name = name;

}

}

public class UserServiceTest {

@Mock

private UserRepository userRepository;

@InjectMocks

private UserService userService;

@BeforeEach

void setUp() {

MockitoAnnotations.openMocks(this);

}

@Test

void testGetUserById() {

User mockUser = new User(1, "Alice");

when(userRepository.findById(1)).thenReturn(Optional.of(mockUser));

User result = userService.getUserById(1);

assertNotNull(result);

assertEquals("Alice", result.name);

}

}

1. **Debugging Memory Leaks in a Java Application (Heap Analysis, Profiling)**

import java.util.\*;

public class MemoryLeakExample {

private static final List<byte[]> memoryLeakList = new ArrayList<>();

public static void createMemoryLeak() {

for (int i = 0; i < 10; i++) {

memoryLeakList.add(new byte[1024 \* 1024]); // Allocating 1MB per iteration

System.out.println("Allocated " + (i + 1) + "MB");

}

}

public static void main(String[] args) {

System.out.println("Starting Memory Leak Example");

createMemoryLeak();

// Simulating heap dump trigger

System.out.println("Triggering garbage collection");

System.gc();

System.out.println("Memory Leak Example completed");

}

}

**Assessment-10:**

**1.sorting:**

import java.util.Arrays;

public class SortingAlgorithms {

// Selection Sort

public static void selectionSort(int[] arr) {

int n = arr.length;

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

int minIndex = i;

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

if (arr[j] < arr[minIndex]) {

minIndex = j;

}

}

int temp = arr[minIndex];

arr[minIndex] = arr[i];

arr[i] = temp;

}

}

// Insertion Sort

public static void insertionSort(int[] arr) {

int n = arr.length;

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

int key = arr[i];

int j = i - 1;

while (j >= 0 && arr[j] > key) {

arr[j + 1] = arr[j];

j--;

}

arr[j + 1] = key;

}

}

// Sliding Window Approach - Maximum Sum of Subarray of Size K

public static int maxSubarraySum(int[] arr, int k) {

int maxSum = 0, windowSum = 0;

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

windowSum += arr[i];

}

maxSum = windowSum;

for (int i = k; i < arr.length; i++) {

windowSum += arr[i] - arr[i - k];

maxSum = Math.max(maxSum, windowSum);

}

return maxSum;

}

// Two-Pointer Approach - Finding Pair with Given Sum

public static boolean hasPairWithSum(int[] arr, int target) {

Arrays.sort(arr); // Sorting required for two-pointer approach

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

while (left < right) {

int sum = arr[left] + arr[right];

if (sum == target) return true;

else if (sum < target) left++;

else right--;

}

return false;

}

public static void main(String[] args) {

int[] arr = {64, 25, 12, 22, 11};

// Selection Sort

selectionSort(arr);

System.out.println("Selection Sort: " + Arrays.toString(arr));

int[] arr2 = {64, 25, 12, 22, 11};

// Insertion Sort

insertionSort(arr2);

System.out.println("Insertion Sort: " + Arrays.toString(arr2));

// Sliding Window

int[] nums = {1, 2, 3, 4, 5, 6, 7, 8, 9};

System.out.println("Max Subarray Sum (size 3): " + maxSubarraySum(nums, 3));

// Two-Pointer Approach

System.out.println("Has Pair with Sum 10: " + hasPairWithSum(nums, 10));

}

}

**2.Maximum Element:**

import java.util.Arrays;

public class MaxElementSearch {

// Finding Maximum Element in an Array

public static int findMaxElement(int[] arr) {

int max = arr[0];

for (int num : arr) {

if (num > max) {

max = num;

}

}

return max;

}

// Sliding Window Approach - Maximum in Subarrays of Size K

public static int[] maxSlidingWindow(int[] arr, int k) {

int n = arr.length;

int[] maxValues = new int[n - k + 1];

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

int max = arr[i];

for (int j = i; j < i + k; j++) {

if (arr[j] > max) {

max = arr[j];

}

}

maxValues[i] = max;

}

return maxValues;

}

public static void main(String[] args) {

int[] nums = {1, 2, 3, 4, 5, 6, 7, 8, 9};

// Finding Maximum Element

System.out.println("Max Element: " + findMaxElement(nums));

// Sliding Window Max in Subarrays

System.out.println("Max in Subarrays of Size 3: " + Arrays.toString(maxSlidingWindow(nums, 3)));

}

}

**3.Superior or Leader elements in an array**

import java.util.ArrayList;

import java.util.List;

public class LeaderElements {

// Function to find leader elements in an array

public static List<Integer> findLeaders(int[] arr) {

List<Integer> leaders = new ArrayList<>();

int n = arr.length;

int maxFromRight = arr[n - 1];

leaders.add(maxFromRight);

for (int i = n - 2; i >= 0; i--) {

if (arr[i] > maxFromRight) {

maxFromRight = arr[i];

leaders.add(maxFromRight);

}

}

return leaders;

}

public static void main(String[] args) {

int[] arr = {16, 17, 4, 3, 5, 2};

System.out.println("Leader Elements: " + findLeaders(arr));

}

}

**4.Expected Time complexity:**

import java.util.Arrays;

public class TimeComplexityAnalysis {

// Bubble Sort - O(n^2)

public static void bubbleSort(int[] arr) {

int n = arr.length;

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

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

if (arr[j] > arr[j + 1]) {

int temp = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = temp;

}

}

}

}

public static void main(String[] args) {

int[] sizes = {100, 500, 1000, 5000};

for (int size : sizes) {

int[] arr = new int[size];

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

arr[i] = (int) (Math.random() \* 10000);

}

int[] arrCopy = Arrays.copyOf(arr, arr.length);

long start = System.nanoTime();

bubbleSort(arr);

long end = System.nanoTime();

System.out.println("Bubble Sort Time for " + size + " elements: " + (end - start) / 1e6 + " ms");

}

}

}

**5.Reverse Array:**

import java.util.Arrays;

public class ReverseArray {

// Method to reverse an array

public static void reverse(int[] arr) {

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

while (left < right) {

int temp = arr[left];

arr[left] = arr[right];

arr[right] = temp;

left++;

right--;

}

}

public static void main(String[] args) {

int[] arr = {1, 2, 3, 4, 5};

System.out.println("Original Array: " + Arrays.toString(arr));

reverse(arr);

System.out.println("Reversed Array: " + Arrays.toString(arr));

}

}

1. **Reverse an array in a sub groups given k(k is multiple of n)**

import java.util.Arrays;

public class ReverseSubGroups {

// Function to reverse subarrays of size k

public static void reverseSubGroups(int[] arr, int k) {

int n = arr.length;

if (n % k != 0) {

System.out.println("Invalid k: It should be a multiple of array length");

return;

}

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

reverseArray(arr, i, i + k - 1);

}

}

// Helper function to reverse an array from start to end index

private static void reverseArray(int[] arr, int start, int end) {

while (start < end) {

int temp = arr[start];

arr[start] = arr[end];

arr[end] = temp;

start++;

end--;

}

}

public static void main(String[] args) {

int[] arr = {1, 2, 3, 4, 5, 6, 7, 8};

int k = 4; // Should be a multiple of n

System.out.println("Original Array: " + Arrays.toString(arr));

reverseSubGroups(arr, k);

System.out.println("Array after reversing in sub-groups of " + k + ": " + Arrays.toString(arr));

}

}

1. **which is not a multiple of k**

import java.util.ArrayList;

import java.util.List;

public class NotMultipleOfK {

// Function to find numbers that are not multiples of k

public static List<Integer> findNotMultiples(int[] arr, int k) {

List<Integer> result = new ArrayList<>();

for (int num : arr) {

if (num % k != 0) {

result.add(num);

}

}

return result;

}

public static void main(String[] args) {

int[] arr = {10, 15, 20, 25, 30, 35, 40};

int k = 5;

System.out.println("Numbers not multiples of " + k + ": " + findNotMultiples(arr, k));

}

}

1. **Sort an Array of 0s and 1s (Binary Sort)**

import java.util.Arrays;

public class BinarySort {

// Function to sort an array of 0s and 1s

public static void binarySort(int[] arr) {

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

while (left < right) {

while (arr[left] == 0 && left < right) {

left++;

}

while (arr[right] == 1 && left < right) {

right--;

}

if (left < right) {

arr[left] = 0;

arr[right] = 1;

left++;

right--;

}

}

}

public static void main(String[] args) {

int[] arr = {1, 0, 1, 0, 1, 0, 0, 1, 1, 0};

System.out.println("Original Array: " + Arrays.toString(arr));

binarySort(arr);

System.out.println("Sorted Array: " + Arrays.toString(arr));

}

}

**Assessment-11:**