Name: Vaibhav Jha

Roll Number: 118

Subject: Java-Practical

Course: 5 Years M.Sc CS

Semester: 5

---

# Assignment- 1

---

---

**1. Write a Java program to swap the values of two variables using a third variable. Also, modify your program to perform the swap without using a third variable.**

---

```
import java.util.Scanner;

class SwapWithThirdVariable {
   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);

      System.out.print("Enter first number: ");
      int a = sc.nextInt();
      System.out.print("Enter second number: ");
      int b = sc.nextInt();

            int temp = a;
      a = b;
      b = temp;

      System.out.println("After swapping: a = " + a + ", b = " + b);
```

```
        a = a + b;
    b = a - b;
    a = a - b;

    System.out.println("After swapping without third variable: a = " + a + ", b = " + b);
  }
}
```

---

**2. Write a Java program that accepts an integer from the user and checks whether the given number is even or odd. Use the modulus operator to determine the result.**

---

```
import java.util.Scanner;

class EvenOdd {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);

    System.out.print("Enter a number: ");
    int num = sc.nextInt();

    if (num % 2 == 0) {
      System.out.println(num + " is Even");
    } else {
      System.out.println(num + " is Odd");
    }
  }
```

```
}
```

---

**3. Write a Java program to calculate the factorial of a given number. Accept the number from the user and use a loop (for or while) to perform the calculation.**

---

```
import java.util.Scanner;

class Factorial {
   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);

      System.out.print("Enter a number: ");
      int num = sc.nextInt();
      int fact = 1;

      for (int i = 1; i <= num; i++) {
         fact *= i;
      }

      System.out.println("Factorial of " + num + " is " + fact);
   }
}
```

---

**4. Write a Java program to print the Fibonacci series up to n terms. Accept the value of n from the user and display the series in proper format.**

**Output: 0 1 1 2 3 5 8 13 21 ,**

---

```
import java.util.Scanner;

class Fibonacci {
   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);

      System.out.print("Enter number of terms: ");
      int n = sc.nextInt();

      int a = 0, b = 1;

      System.out.print("Fibonacci Series: ");
      for (int i = 1; i <= n; i++) {
         System.out.print(a + " ");
         int next = a + b;
         a = b;
         b = next;
      }
   }
}
```

---

**5. Write a Java program to check whether a given number is a palindrome or not. A palindrome is a number that reads the same backward as forward (e.g., 121, 1331).**

---

```
import java.util.Scanner;

class Palindrome {
```

```
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter a number: ");
        int num = sc.nextInt();
        int originalNum = num;
        int reversedNum = 0;

        while (num != 0) {
            reversedNum = reversedNum * 10 + num % 10;
            num /= 10;
        }

        if (originalNum == reversedNum) {
            System.out.println(originalNum + " is a palindrome.");
        } else {
            System.out.println(originalNum + " is not a palindrome.");
        }
    }
}
```

---

**6. Write a Java program to check whether a given number is an Armstrong or not.**

**Armstrong number if the sum of the cubes of its digits is equal to the number itself (e.g., 153, 370, 371,407)**

---

```
import java.util.Scanner;

class Armstrong {
    public static void main(String[] args) {
```

```
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter a number: ");
        int num = sc.nextInt();
        int originalNum = num;
        int sum = 0;

        while (num != 0) {
            int digit = num % 10;
            sum += digit * digit * digit;
            num /= 10;
        }

        if (sum == originalNum) {
            System.out.println(originalNum + " is an Armstrong number.");
        } else {
            System.out.println(originalNum + " is not an Armstrong number.");
        }
    }
}
```

---

**7. Write a Java program to check whether a given number is prime or not. Accept the number from the user and display the result accordingly.**

---

```
import java.util.Scanner;

class PrimeNumber {
    public static void main(String[] args) {
```

```
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter a number: ");
        int num = sc.nextInt();

        boolean isPrime = true;

        for (int i = 2; i <= num / 2; i++) {
            if (num % i == 0) {
                isPrime = false;
                break;
            }
        }

        if (isPrime && num > 1) {
            System.out.println(num + " is a prime number.");
        } else {
            System.out.println(num + " is not a prime number.");
        }
    }
}
```

---

**8. Write a Java program to check whether a given year is a leap year or not.**

**A year is a leap year if it is divisible by 4, but not by 100, unless it is also divisible by 400.**

---

```
import java.util.Scanner;
```

```
class LeapYear {
   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);

      System.out.print("Enter a year: ");
      int year = sc.nextInt();

      if ((year % 4 == 0 && year % 100 != 0) || (year % 400 == 0)) {
         System.out.println(year + " is a leap year.");
      } else {
         System.out.println(year + " is not a leap year.");
      }
   }
}
```

---

**9. Write a Java program to calculate the power of a number. Accept base and exponent from the user and calculate using a loop.**

**Example: a**

**b**

**, e.g. 25 = 32**

---

```
import java.util.Scanner;

class Power {
   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);
```

```
        System.out.print("Enter base: ");
        int base = sc.nextInt();
        System.out.print("Enter exponent: ");
        int exponent = sc.nextInt();

        int result = 1;

        for (int i = 1; i <= exponent; i++) {
            result *= base;
        }

        System.out.println("Result: " + result);
    }
}
```

---

**10. Write a Java program to check whether a number is a perfect number or not.**

**A perfect number is a number whose sum of proper divisors equals the number itself.**

**Example: 6 ? 1+2+3=6**

---

```
import java.util.Scanner;

class PerfectNumber {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter a number: ");
        int num = sc.nextInt();
        int sum = 0;
```

```
        for (int i = 1; i <= num / 2; i++) {
            if (num % i == 0) {
                sum += i;
            }
        }

        if (sum == num) {
            System.out.println(num + " is a perfect number.");
        } else {
            System.out.println(num + " is not a perfect number.");
        }
    }
}
```

---

# Assignment- 2

---

---

**1. Create a Class and Object**

**o Define a Student class with attributes (name, rollNo), create objects and display**

---

```
details.class Student {
    String name;
    int rollNo;

    void display() {
```

```
        System.out.println("Student Name: " + name + ", Roll No: " + rollNo);
    }
}
```

---

**2. Constructor Example**

**o Create a Book class with a constructor to initialize book name and author, and a method to display them.**

---

```
class Book {
    String bookName;
    String author;

    Book(String name, String author) {
        this.bookName = name;
        this.author = author;
    }

    void display() {
        System.out.println("Book: " + bookName + ", Author: " + author);
    }
}
```

---

**3. Default and Parameterized Constructor**

**o Car class with two constructors: one default and one parameterized.**

---

```
class Car {
   String model;

   Car() {
      model = "Default Model";
   }

   Car(String modelName) {
      model = modelName;
   }

   void display() {
      System.out.println("Car Model: " + model);
   }
}
```

---

**4. Function Overloading**

**o Calculator class with multiple add() methods:**

- **add(int, int), add(double, double), add(int, int, int)**

---

```
class Calculator {
   int add(int a, int b) {
      return a + b;
   }

   double add(double a, double b) {
```

```
    return a + b;
  }

  int add(int a, int b, int c) {
    return a + b + c;
  }
}
```

---

**5. Constructor Overloading**

**o Employee class with overloaded constructors to initialize with different sets of**

- **data (e.g., name only, name and id, name, id, and salary).**

---

```
class Employee {
  String name;
  int id;
  double salary;

  Employee(String n) {
    name = n;
  }

  Employee(String n, int i) {
    name = n;
    id = i;
  }

  Employee(String n, int i, double s) {
    name = n;
```

```
        id = i;
        salary = s;
    }

    void display() {
        System.out.println("Employee: " + name + ", ID: " + id + ", Salary: " + salary);
    }
}
```

---

**6. Class with Method to Calculate Area**

**o Create a Rectangle class with length and width, and a method**

- **calculateArea().**

---

```
class Rectangle {
    int length, width;

    Rectangle(int l, int w) {
        length = l;
        width = w;
    }

    int calculateArea() {
        return length * width;
    }
}
```

---

**7. Student Class with Marks and Average**

**o Accept marks of 3 subjects using constructor, calculate average using method.**

---

```
class MarksStudent {
   int m1, m2, m3;

   MarksStudent(int a, int b, int c) {
      m1 = a;
      m2 = b;
      m3 = c;
   }

   double average() {
      return (m1 + m2 + m3) / 3.0;
   }
}
```

---

**8. Bank Account Class**

**o Class BankAccount with deposit, withdraw, and showBalance methods; use constructors to initialize account.**

---

```
class BankAccount {
   int balance;

   BankAccount(int initial) {
      balance = initial;
```

```
    }

    void deposit(int amount) {
        balance += amount;
    }

    void withdraw(int amount) {
        if (amount <= balance)
            balance -= amount;
        else
            System.out.println("Insufficient balance!");
    }

    void showBalance() {
        System.out.println("Balance: " + balance);
    }
}
```

---

**9. Class with Object as a Member**

**o Create Address and Employee classes. Employee has an Address object as a member.**

---

```
class Address {
    String city, state;

    Address(String c, String s) {
        city = c;
        state = s;
    }
```

```
}

class EmpWithAddress {
   String name;
   Address addr;

   EmpWithAddress(String n, Address a) {
      name = n;
      addr = a;
   }

   void display() {
      System.out.println("Name: " + name + ", City: " + addr.city + ", State: " + addr.state);
   }
}
```

---

**10.Function Overloading in Constructor and Method**

- **Shape class with overloaded constructors for circle and rectangle. Also overload area() method to handle both shapes.**

---

```
class Shape {
   double radius, length, breadth;

   Shape(double r) {
      radius = r;
   }

   Shape(double l, double b) {
```

```
    length = l;
    breadth = b;
  }

  double area() {
    if (radius != 0)
      return 3.14 * radius * radius;
    else
      return length * breadth;
  }
}
```

---

**11.Class with Private Members and Public Getters/Setters**

- **Student class with private fields (name, age) and public methods to access them using getter/setter methods. Use constructor to initialize.**

---

```
class StudentEncap {
  private String name;
  private int age;

  StudentEncap(String n, int a) {
    name = n;
    age = a;
  }

  public String getName() { return name; }
  public void setName(String n) { name = n; }
```

```
    public int getAge() { return age; }
    public void setAge(int a) { age = a; }
}
```

---

**12.Array of Objects**

- **Create a Product class and an array of Product objects. Accept data and display all products using loop.**

---

```
class Product {
    int id;
    String name;

    void setData(int i, String n) {
        id = i;
        name = n;
    }

    void display() {
        System.out.println("Product ID: " + id + ", Name: " + name);
    }
}
```

---

**13.Constructor with Validation using Exception**

- **Employee constructor throws an exception if salary is negative.**

---

```
class EmpValidate {
   String name;
   double salary;

   EmpValidate(String n, double s) throws Exception {
      if (s < 0) throw new Exception("Salary cannot be negative");
      name = n;
      salary = s;
   }

   void display() {
      System.out.println("Employee: " + name + ", Salary: " + salary);
   }
}
```

---

**14.Custom Exception Handling**

- **Create a custom exception InvalidAgeException. Throw it if age < 18 in a method checkEligibility().**

---

```
class InvalidAgeException extends Exception {
   InvalidAgeException(String msg) {
      super(msg);
   }
}

class Voter {
   int age;
```

```
  Voter(int a) {
    age = a;
  }

  void checkEligibility() throws InvalidAgeException {
    if (age < 18)
      throw new InvalidAgeException("Not eligible to vote!");
    else
      System.out.println("Eligible to vote.");
  }
}
```

---

**15.Static vs Non-static Members**

- **University class with static universityName and non-static studentName.**

**Demonstrate calling static vs non-static members.**

---

```
class University {
  static String universityName = "ABC University";
  String studentName;

  University(String sName) {
    studentName = sName;
  }

  void display() {
    System.out.println("Student: " + studentName + ", University: " + universityName);
  }
```

```
}
```

---

**16.Multiple Classes with Relationships**

- **Department and Professor class. Each Professor is linked to a Department object.**

---

```
class Department {
   String deptName;

   Department(String name) {
      deptName = name;
   }
}

class Professor {
   String name;
   Department dept;

   Professor(String n, Department d) {
      name = n;
      dept = d;
   }

   void display() {
      System.out.println("Professor: " + name + ", Department: " + dept.deptName);
   }
}
```

---

**17.Array of Objects with Total Calculation**

- **Marks class having subject marks, use array of students to calculate and display total and average marks.**

---

```
class Marks {
   int s1, s2, s3;

   void setMarks(int a, int b, int c) {
      s1 = a;
      s2 = b;
      s3 = c;
   }

   int total() {
      return s1 + s2 + s3;
   }

   double average() {
      return total() / 3.0;
   }
}
```

---

**18.Banking System with Exception and Access Modifiers**

- **Create a BankAccount class with private balance, public deposit() and withdraw(). Throw exception if withdrawal amount > balance.**

---

```
class SafeBank {
   private int balance;

   SafeBank(int bal) {
      balance = bal;
   }

   public void deposit(int amt) {
      balance += amt;
   }

   public void withdraw(int amt) throws Exception {
      if (amt > balance)
         throw new Exception("Insufficient funds!");
      balance -= amt;
   }

   public void showBalance() {
      System.out.println("Balance: " + balance);
   }
}
```

---

**19.Constructor Calling Another Constructor (this())**

- **Use this() to chain constructors inside a Customer class.**

---

```
class Customer {
   String name;
   int age;

   Customer() {
      this("Default", 0);
   }

   Customer(String n) {
      this(n, 0);
   }

   Customer(String n, int a) {
      name = n;
      age = a;
   }

   void display() {
      System.out.println("Customer: " + name + ", Age: " + age);
   }
}
```

---

**20.Library Management with Object Array and Search**

- **Book class with ID, title, author. Store multiple books and allow searching by book title.**

---

```
class LibBook {
   int id;
```

```
    String title;
    String author;

    void setData(int i, String t, String a) {
        id = i;
        title = t;
        author = a;
    }

    void display() {
        System.out.println("ID: " + id + ", Title: " + title + ", Author: " + author);
    }
}

public class Main {
    public static void main(String[] args) {
        // You can create test cases here to check each class if needed.
        System.out.println("All classes are implemented.");
    }
}
```

---

# Assignment- 3

---

---

**Q-1 : Write a Java program using the Object-Oriented Programming (OOP) concept to find the Greatest Common Divisor (GCD) of two positive integers using recursion.**

**The program should:**

**1. Define a class GCDExample that contains:**

**o Two instance variables to store the numbers.**

**o A constructor to initialize these numbers.**

**o A public method calculateGCD() that calls a private recursive method**

**gcdRecursive(int a, int b)**

**implementing Euclid's Algorithm.**

**2. In the main() method:**

**o Create an object of GCDExample by passing two positive integers.**

**o Display the GCD of the two numbers.**

---

```
class GCDExample {
private int num1;
private int num2;

public GCDExample(int num1, int num2) {
this.num1 = num1;
this.num2 = num2;
}

public int calculateGCD() {
return gcdRecursive(num1, num2);
}

private int gcdRecursive(int a, int b) {
if (b == 0)
return a;

return gcdRecursive(b, a % b);
```

**}**

**public static void main(String[] args) {**

**GCDExample gcdExample = new GCDExample(48, 18);**

**int gcd = gcdExample.calculateGCD();**

**System.out.println("GCD of 48 and 18 is: " + gcd);**

**}**

**}**

---

**Q-2 : Write a Java program using the Object-Oriented Programming (OOP) approach to compute the**

**sum of the**

**following series:**

**S=1/1+1/2+1/3+⋯+1/n**

**S1 = 1/1 + 1/2 +1/22 + ….. 1/2n**

**where n is a positive integer entered by the user.**

**1. Create a class Rational to represent a rational number (fraction) with:**

**o Two instance variables: numerator and denominator.**

**o A constructor to initialize the rational number and simplify it.**

**o A method add(Rational other) to add two rational numbers and return the result**

**as a new Rational**

**object.**

**o A toString() method to return the rational number in numerator/denominator**

**format.**

**2. In the main class Summation Series:**

**o Prompt the user to enter the value of n.**

**o Compute the sum of the series using Rational objects without converting fractions**

**to decimals.**

**o Display the sum as a simplified fraction.**

```
---------------------------------------------------------------------------------------------------------------------------------------

import java.util.Scanner;

class Rational {
private int numerator;
private int denominator;

public Rational(int numerator, int denominator) {
if (denominator == 0) {
throw new IllegalArgumentException(&amp;quot;Denominator cannot be zero.&amp;quot;);
}
int gcd = gcd(Math.abs(numerator), Math.abs(denominator));
this.numerator = numerator / gcd;
this.denominator = denominator / gcd;

if (this.denominator &amp;lt; 0) {
this.denominator = -this.denominator;
this.numerator = -this.numerator;
}
}

public Rational add(Rational other) {
int num = this.numerator * other.denominator + other.numerator * this.denominator;
int den = this.denominator * other.denominator;
return new Rational(num, den);
}

private int gcd(int a, int b) {
if (b == 0)

return a;
```

```
return gcd(b, a % b);
}

@Override
public String toString() {
if (denominator == 1) return numerator + &quot;&quot;;
else return numerator + &quot;/&quot; + denominator;
}
}

public class SummationSeries {
public static void main(String[] args) {
Scanner sc = new Scanner(System.in);
System.out.print(&quot;Enter positive integer n: &quot;);
int n = sc.nextInt();

Rational sumS = new Rational(0, 1);
for (int i = 1; i &lt;= n; i++) {
sumS = sumS.add(new Rational(1, i));
}

Rational sumS1 = new Rational(0, 1);
for (int i = 0; i &lt; n; i++) {
sumS1 = sumS1.add(new Rational(1, (int) Math.pow(2, i)));
}

System.out.println(&quot;Sum S = 1/1 + 1/2 + ... + 1/n = &quot; + sumS);
System.out.println(&quot;Sum S1 = 1/1 + 1/2 + 1/4 + ... + 1/2^n = &quot; + sumS1);

}
}
```

---

**3. Bank Account Management System**

**Write a Java program using inheritance and polymorphism to model a banking system:**

**o Create an abstract class BankAccount with attributes accountNumber, accountHolderName, and balance.**

**o Include abstract methods deposit(double amount) and withdraw(double amount).**

**o Create subclasses SavingsAccount and CurrentAccount with specific withdrawal rules.**

**o In the main method, store multiple accounts in an array and perform deposits/withdrawals using dynamic method dispatch.**

---

```
public class Car {
private String brand;
private String model;
private int year;

public Car() {
brand = &quot;Unknown&quot;;
model = &quot;Unknown&quot;;
year = 0;
}

public Car(String brand, String model, int year) {
this.brand = brand;
this.model = model;
this.year = year;
}

public void displayDetails() {
System.out.println(&quot;Brand: &quot; + brand);

System.out.println(&quot;Model: &quot; + model);
```

System.out.println("Year: " + year);

}

public static void main(String[] args) {

Car car1 = new Car();

System.out.println("Car 1 details:");

car1.displayDetails();

System.out.println();

Car car2 = new Car("Toyota", "Corolla", 2022);

System.out.println("Car 2 details:");

car2.displayDetails();

}

}

---

4. Library Management with Interfaces

Design a system for managing books in a library:

o Create an interface LibraryOperations with methods addBook(Book b), removeBook(int bookId), and searchBook(String title).

o Implement this interface in a Library class that maintains an ArrayList<Book>.

o Use encapsulation to keep book details private.

o Demonstrate adding, removing, and searching books in the main method.

---

public class Calculator {

public int add(int a, int b) {

return a + b;

}

```
// Add two doubles
public double add(double a, double b) {
return a + b;
}

public int add(int a, int b, int c) {
return a + b + c;
}

public static void main(String[] args) {
Calculator calc = new Calculator();

System.out.println(&quot;Add 2 ints (5 + 10): &quot; + calc.add(5, 10));
System.out.println(&quot;Add 2 doubles (3.5 + 4.5): &quot; + calc.add(3.5, 4.5));
System.out.println(&quot;Add 3 ints (1 + 2 + 3): &quot; + calc.add(1, 2, 3));
}
}
```

---

**5. Employee Payroll Calculation**

**Using inheritance and method overriding:**

**o Create a base class Employee with fields name, id, and salary.**

**o Create subclasses FullTimeEmployee and PartTimeEmployee with different salary calculation methods.**

**o Store multiple employees in an ArrayList and use polymorphism to calculate salaries.**

---

```
class Employee {
private String name;
private int id;

private double salary;
```

```
public Employee(String name) {

this.name = name;

this.id = 0;

this.salary = 0;

}


public Employee(String name, int id) {

this.name = name;

this.id = id;

this.salary = 0;

}


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

this.name = name;

this.id = id;

this.salary = salary;

}


public void display() {

System.out.println(&quot;Name: &quot; + name + &quot;, ID: &quot; + id + &quot;, Salary: &quot;
+ salary);

}


public static void main(String[] args) {

Employee e1 = new Employee(&quot;Alice&quot;);

Employee e2 = new Employee(&quot;Bob&quot;, 102);

Employee e3 = new Employee(&quot;Charlie&quot;, 103, 50000);


e1.display();

e2.display();
```

```
e3.display();
}
}
```

---

**6. Student Result Processing with Abstract Class**

**o Create an abstract class Student with fields rollNumber, name and abstract method calculateResult().**

**o Create a subclass UndergraduateStudent and PostgraduateStudent with different pass percentage criteria.**

**o Take input for multiple students and display pass/fail results using runtime polymorphism.**

---

```
class Rectangle {
private double length;
private double width;

public Rectangle(double length, double width) {
this.length = length;
this.width = width;
}

public double calculateArea() {
return length * width;
}

public static void main(String[] args) {
Rectangle rect = new Rectangle(5, 3);
System.out.println(&amp;quot;Area of rectangle: &amp;quot; + rect.calculateArea());
}

}
```

---

**7. Online Shopping Cart with Generics**

**o Create a generic class Cart<T> that stores items of type T.**

**o Add methods addItem(T item), removeItem(T item), and displayItems().**

**o Create a Product class with name and price.**

**o Use the cart with Product objects and display the total cost.**

---

```
class Student {
private int mark1, mark2, mark3;
public Student(int mark1, int mark2, int mark3) {
this.mark1 = mark1;
this.mark2 = mark2;
this.mark3 = mark3;
}

public double calculateAverage() {
return (mark1 + mark2 + mark3) / 3.0;
}
public static void main(String[] args) {
Student s = new Student(80, 90, 85);
System.out.println(&quot;Average marks: &quot; + s.calculateAverage());
}
}
```

---

**8. Matrix Operations using OOP**

**o Create a class Matrix with a 2D array and methods for addition, subtraction,**

**multiplication of matrices.**

**o Implement method overloading for scalar multiplication.**

**o Overload the toString() method to display matrices neatly.**

**o Write a main program to demonstrate all operations.**

---

```
class BankAccount {
private String accountNumber;
private double balance;

public BankAccount(String accountNumber, double initialBalance) {
this.accountNumber = accountNumber;
this.balance = initialBalance;
}

public void deposit(double amount) {
if(amount &gt; 0)
balance += amount;
else
System.out.println(&quot;Invalid deposit amount&quot;);
}

public void withdraw(double amount) {
if (amount &gt; 0 &amp;&amp; amount &lt;= balance)
balance -= amount;
else
System.out.println(&quot;Insufficient funds or invalid amount&quot;);
}

public void showBalance() {
System.out.println(&quot;Account &quot; + accountNumber + &quot; balance: &quot; + balance);
```

**}**

**public static void main(String[] args) {**

**BankAccount acc = new BankAccount("12345", 1000);**

**acc.deposit(500);**

**acc.withdraw(200);**

**acc.showBalance();**

**}**

**}**

---

**9. Shape Area & Perimeter Calculation**

**o Create an abstract class Shape with abstract methods calculateArea() and calculatePerimeter().**

**o Create subclasses Circle, Rectangle, and Triangle.**

**o Use method overriding to implement each formula.**

**o Store multiple shapes in an array and calculate areas/perimeters using a single loop (polymorphism).**

---

**class Address {**

**private String city;**

**private String state;**

**private String country;**

**public Address(String city, String state, String country) {**

**this.city = city;**

**this.state = state;**

**this.country = country;**

```
}

public String toString() {
return city + &quot;, &quot; + state + &quot;, &quot; + country;
}
}

public class Employee {
private String name;
private Address address;

public Employee(String name, Address address) {
this.name = name;
this.address = address;
}

public void display() {
System.out.println(&quot;Name: &quot; + name);
System.out.println(&quot;Address: &quot; + address);
}

public static void main(String[] args) {
Address addr = new Address(&quot;New York&quot;, &quot;NY&quot;, &quot;USA&quot;);
Employee emp = new Employee(&quot;John Doe&quot;, addr);
emp.display();
}
}
```

---

**10. Polynomial Operations**

o Create a Polynomial class that represents a polynomial equation (e.g., 3x2+2x+53x^2

**+ 2x + 5).**

**o Use an ArrayList to store coefficients.**

**o Implement methods to add, subtract, and multiply two polynomials.**

**o Override toString() to display them in algebraic form.**

**o In main(), take two polynomials and perform all operations.**

---

```
class Shape {
private double radius;
private double length, width;
private boolean isCircle;

public Shape(double radius) {
this.radius = radius;

this.isCircle = true;
}

public Shape(double length, double width) {
this.length = length;
this.width = width;
this.isCircle = false;
}

public double area() {
if (isCircle) {
return Math.PI * radius * radius;
} else {
return length * width;
}
}
```

**public static void main(String[] args) {**

**Shape circle = new Shape(5);**

**Shape rectangle = new Shape(4, 6);**

**System.out.println("Circle area: " + circle.area());**

**System.out.println("Rectangle area: " + rectangle.area());**

**}**

**}**

---

**11. University Course Enrollment**

**o Create classes Course and Student.**

**o A student can enroll in multiple courses, and a course can have multiple students (many-to-many relationship).**

**o Use composition with ArrayList to store enrollments.**

**o Implement methods to add/remove students from a course and display enrolled students.**

---

**class Student {**

**private String name;**

**private int age;**

**public Student(String name, int age) {**

**this.name = name;**

**setAge(age);**

**}**

**public String getName() {**

```
return name;
}

public void setName(String name) {
this.name = name;
}

public int getAge() {
return age;
}

public void setAge(int age) {
if (age &gt;= 0) this.age = age;
else System.out.println(&quot;Invalid age&quot;);

}

public static void main(String[] args) {
Student s = new Student(&quot;Alice&quot;, 20);
System.out.println(&quot;Name: &quot; + s.getName());
System.out.println(&quot;Age: &quot; + s.getAge());
s.setAge(22);
System.out.println(&quot;Updated Age: &quot; + s.getAge());
}
}
```

---

**12. Vehicle Rental System**

**Design a Java program to manage a vehicle rental system using OOP principles.**

**1. Create an abstract class Vehicle with:**

**o Fields: registrationNumber, brand, dailyRentalRate.**

**o Constructor to initialize these fields.**

**o Abstract method calculateRentalCost(int days).**

**o toString() method to display vehicle details.**

**2. Create subclasses:**

**o Car (extra field: seatingCapacity).**

**o Motorbike (extra field: engineCapacity).**

**o Truck (extra field: loadCapacity).**

**3. Each subclass overrides calculateRentalCost() to apply different rules:**

**o Car: base cost = days × dailyRentalRate. If days > 7, apply 10% discount.**

**o Motorbike: base cost = days × dailyRentalRate. If engineCapacity > 500cc, add ₹200/day.**

**o Truck: base cost = days × dailyRentalRate. If loadCapacity > 5000kg, add 15% surcharge.**

**4. In the main() method:**

**o Store multiple vehicles in an ArrayList<Vehicle>.**

**o Let the user input days for each rental.**

**o Display the total rental cost for each vehicle using runtime polymorphism.**

**Example Output:**

**Vehicle: Car, Reg No: GJ05AB1234, Brand: Toyota, Days: 10, Cost: ₹13500.0**

---

```
import java.util.Scanner;

class Product {
private String name;
private double price;

public Product(String name, double price) {
this.name = name;
this.price = price;
```

```
}

public void display() {
System.out.println(&quot;Product: &quot; + name + &quot;, Price: &quot; + price);
}
}

public class ProductArray {
public static void main(String[] args) {
Scanner sc = new Scanner(System.in);

System.out.print(&quot;Enter number of products: &quot;);
int n = sc.nextInt();
sc.nextLine();
Product[] products = new Product[n];

for (int i = 0; i &lt; n; i++) {
System.out.print(&quot;Enter name of product &quot; + (i+1) + &quot;: &quot;);
String name = sc.nextLine();
System.out.print(&quot;Enter price of product &quot; + (i+1) + &quot;: &quot;);
double price = sc.nextDouble();
sc.nextLine();

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

System.out.println(&quot;\nProducts list:&quot;);
for (Product p : products) {
p.display();
}
}
```

}

---

**13. Inventory System with Method Overloading**

o Create an Inventory class to store products.

o Implement overloaded methods addProduct() that can:

o Add by name and price.

o Add by name, price, and quantity.

o Add by a Product object.

o Ensure no duplicate products (use equals() and hashCode() in Product).

---

```
class NegativeSalaryException extends Exception {
public NegativeSalaryException(String message) {
super(message);
}
}

public class Employee {
private String name;
private double salary;

public Employee(String name, double salary) throws NegativeSalaryException {
if (salary &lt; 0)
throw new NegativeSalaryException(&quot;Salary cannot be negative&quot;);
this.name = name;
this.salary = salary;
}
public void display() {
```

System.out.println("Name: " + name + ", Salary: " + salary);

}

public static void main(String[] args) {

try {

Employee e1 = new Employee("John", 50000);

e1.display();

Employee e2 = new Employee("Jane", -10000);

e2.display();

} catch (NegativeSalaryException e) {

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

}

}

}

---

**14. Complex Number Calculator**

o Create a ComplexNumber class with real and imaginary parts.

o Implement methods for addition, subtraction, multiplication, and division of

complex numbers.

o Override toString() for proper display (e.g., 3 + 4i).

o Demonstrate in main() with multiple operations.

---

class InvalidAgeException extends Exception {

public InvalidAgeException(String message) {

super(message);

}

}

public class EligibilityChecker {

```
public void checkEligibility(int age) throws InvalidAgeException {
if (age &lt; 18) {
throw new InvalidAgeException(&quot;Age must be at least 18&quot;);
} else {
System.out.println(&quot;Eligible.&quot;);
}
}

public static void main(String[] args) {
EligibilityChecker checker = new EligibilityChecker();

try {
checker.checkEligibility(20);
checker.checkEligibility(16);
} catch (InvalidAgeException e) {
System.out.println(&quot;Exception: &quot; + e.getMessage());
}
}
}
```

---

**15. Bank Loan Eligibility System**

o Create an abstract class Loan with method isEligible().
o Create subclasses HomeLoan, CarLoan, and EducationLoan with different eligibility rules.
o Take customer details and display whether they are eligible for a loan type.
o Use runtime polymorphism for the check.

```
class University {
static String universityName = &quot;ABC University&quot;;
String studentName;

public University(String studentName) {
this.studentName = studentName;
}

public static void main(String[] args) {
System.out.println(&quot;University Name (static): &quot; + University.universityName);

University student1 = new University(&quot;Alice&quot;);
University student2 = new University(&quot;Bob&quot;);

System.out.println(&quot;Student 1: &quot; + student1.studentName);
System.out.println(&quot;Student 2: &quot; + student2.studentName);
}
}
```

---

**16. Online Exam Grading System**

- Create classes Question and Exam.
- Store multiple choice questions and answers in ArrayList.
- Use encapsulation for storing the correct answer.
- In main(), allow the user to attempt the exam and display their score.

---

```
class Department {
private String deptName;

public Department(String deptName) {
```

```
this.deptName = deptName;
}

public String getDeptName() {
return deptName;
}
}

public class Professor {
private String profName;
private Department department;

public Professor(String profName, Department department) {
this.profName = profName;
this.department = department;
}

public void display() {
System.out.println(&quot;Professor: &quot; + profName + &quot;, Department: &quot; +
department.getDeptName());
}

public static void main(String[] args) {
Department cs = new Department(&quot;Computer Science&quot;);
Department math = new Department(&quot;Mathematics&quot;);

Professor p1 = new Professor(&quot;Dr. Smith&quot;, cs);
Professor p2 = new Professor(&quot;Dr. Jones&quot;, math);

p1.display();
p2.display();
```

**}**

**}**

---

**17. Travel Booking Fare Calculator**

**o Create an abstract class TravelBooking with method calculateFare().**

**o Subclasses: BusBooking, TrainBooking, and FlightBooking.**

**o Each has a different fare formula (base fare + tax + discount rules).**

**o In main(), create different bookings and calculate fares using polymorphism.**

---

```
class Marks {
private int[] subjectMarks;

public Marks(int[] marks) {
this.subjectMarks = marks;
}

public int totalMarks() {
int sum = 0;
for (int m : subjectMarks) {
sum += m;
}
return sum;
}

public double averageMarks() {
return totalMarks() / (double) subjectMarks.length;
}

public static void main(String[] args) {
```

Marks[] students = new Marks[3];

students[0] = new Marks(new int[]{70, 80, 90});

students[1] = new Marks(new int[]{60, 75, 85});

students[2] = new Marks(new int[]{90, 95, 100});

int totalAllStudents = 0;

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

System.out.println("Student " + (i+1) + " total marks: " + students[i].totalMarks());

totalAllStudents += students[i].totalMarks();

}

double averageAllStudents = totalAllStudents / (double)(students.length * 3);

System.out.println("Average marks of all students: " + averageAllStudents);

}

}

---

18. E-Commerce Discount System

o Create an interface Discountable with method getDiscountPrice().

o Implement it in Electronics, Clothing, and Grocery classes.

o Each category has different discount rules.

o Use interface polymorphism to display final prices for different products.

---

class InsufficientFundsException extends Exception {

public InsufficientFundsException(String message) {

super(message);

}

}

public class BankAccount {

private double balance;

```
public BankAccount(double initialBalance) {
this.balance = initialBalance;

}

public void deposit(double amount) {
if(amount &gt; 0) balance += amount;
}

public void withdraw(double amount) throws InsufficientFundsException {
if (amount &gt; balance) {
throw new InsufficientFundsException(&quot;Withdrawal amount exceeds balance&quot;);
}
balance -= amount;
}

public void showBalance() {
System.out.println(&quot;Balance: &quot; + balance);
}

public static void main(String[] args) {
BankAccount acc = new BankAccount(1000);

acc.deposit(500);
try {
acc.withdraw(200);
acc.showBalance();

acc.withdraw(2000); // throws exception
} catch (InsufficientFundsException e) {
```

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

}

}

}

---

19. Sports Tournament Points Table

o Create a Team class with fields name, matchesPlayed, wins, losses, points.

o Use methods to update match results and auto-update points.

o Store teams in an ArrayList and sort them by points using Comparator.

---

class Customer {

private String name;

private int id;

private double balance;

public Customer(String name) {

this(name, 0);

}

public Customer(String name, int id) {

this(name, id, 0);

}

public Customer(String name, int id, double balance) {

this.name = name;

this.id = id;

this.balance = balance;

}

**public void display() {**

**System.out.println("Name: " + name + ", ID: " + id + ", Balance: " + balance);**

**}**

**public static void main(String[] args) {**

**Customer c1 = new Customer("Alice");**

**Customer c2 = new Customer("Bob", 101);**

**Customer c3 = new Customer("Charlie", 102, 1000);**

**c1.display();**

**c2.display();**

**c3.display();**

**}**

**}**

---

**20. Bank Transaction History (Without Files)**

**o Create a Transaction class with fields type (Deposit/Withdraw), amount, and**

**date.**

**o Create a BankAccount class that maintains an ArrayList<Transaction>.**

**o Implement deposit/withdraw methods that automatically record each**

**transaction in history.**

**o Display account details and transaction history in main().**

---

**import java.util.ArrayList;**

**import java.util.Scanner;**

**class Book {**

```
private int id;

private String title;

private String author;


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

this.id = id;

this.title = title;


this.author = author;

}


public int getId() { return id; }

public String getTitle() { return title; }


public void display() {

System.out.println(&amp;quot;ID: &amp;quot; + id + &amp;quot;, Title: &amp;quot; + title + &amp;quot;, Author: &amp;quot; +
author);

}

}


public class LibraryManagement {

private ArrayList&amp;lt;Book&amp;gt; books = new ArrayList&amp;lt;&amp;gt;();


public void addBook(Book b) {

books.add(b);

}


public void searchByTitle(String title) {

boolean found = false;

for (Book b : books) {

if (b.getTitle().equalsIgnoreCase(title)) {
```

**b.display();**

**found = true;**

**}**

**}**

**if (!found) {**

**System.out.println("Book not found.");**

**}**

**}**

**public void displayBooks() {**

**for (Book b : books) {**

**b.display();**

**}**

**}**

**public static void main(String[] args) {**

**LibraryManagement**

---

---

# Assignment- 4

---

**Q1. Character Frequency Counter (using charAt() and length())**

**Concepts: class, methods, charAt(), loops**

**Problem:**

**Create a class CharFrequency with methods to count the frequency of each character**

**(case-insensitive) in a string without using arrays or collections.**

**Print each unique character and its count.**

**Hint: use nested loops and charAt() inside a class method.**

**Code:-**

```
import java.util.Scanner;
public class CharFrequency {
   public static void frequencyCounter(String str) {
      str = str.toLowerCase();
      for (char alphabet = 'a'; alphabet <= 'z'; alphabet++) {
         int alphabetCounter = 0;
         for (int i = 0; i < str.length(); i++) {
            if (str.charAt(i) == alphabet) {
               alphabetCounter++;
            }
         }
         System.out.println("Count of '" + alphabet + "' : " + alphabetCounter);
      }
   }
   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);
      System.out.print("Enter a string: ");
      String str = scanner.nextLine();
      scanner.close();

      frequencyCounter(str);
   }
}
```

**Output:-**


**Q2. String Comparator (using equals() and equalsIgnoreCase())**

**Concepts: class, constructor, object comparison**

**Problem:**

**Design a class StringCompare with two string attributes initialized using a constructor.**

**Write methods:**

**compareExact() → compares with equals()**

**compareIgnoreCase() → compares with equalsIgnoreCase()**

**Print appropriate messages with the result.**

**Add-on: Count the number of differing characters if they're not equal.**

**Code:-**

```
import java.util.Scanner;


public class StringCompare {
   private String str1;
   private String str2;


   public StringCompare(String str1, String str2) {
      this.str1 = str1;
      this.str2 = str2;
   }


   public void compareExact() {
      if (str1.equals(str2)) {
```

```
        System.out.println("Both the string are equal!");
    } else {
        System.out.println("The two strings are not equal!");
    }
    int len1 = str1.length();
    int len2 = str2.length();
    int minLength = Math.min(len1, len2);
    int lengthDifference = Math.abs(len1 - len2);
    int differenceCount = lengthDifference;

    for (int i = 0; i < minLength; i++) {
        if (str1.charAt(i) != str2.charAt(i)) {
            differenceCount++;
        }
    }

    System.out.println("Number of different character: " + differenceCount);
}

public void compareIgnoreCase() {
    if (str1.equalsIgnoreCase(str2)) {
        System.out.println("Both the string are equal!");
    } else {
        System.out.println("The two strings are not equal!");
    }
    int len1 = str1.length();
    int len2 = str2.length();
    String s1 = str1.toLowerCase();
    String s2 = str2.toLowerCase();
    int minLength = Math.min(len1, len2);
    int lengthDifference = Math.abs(len1 - len2);
```

```
        int differenceCount = lengthDifference;

        for (int i = 0; i < minLength; i++) {
            if (s1.charAt(i) != s2.charAt(i)) {
                differenceCount++;
            }
        }

        System.out.println("Number of different character: " + differenceCount);
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter 2 Strings");
        String s1 = scanner.nextLine();
        String s2 = scanner.nextLine();
        scanner.close();

        StringCompare sc = new StringCompare(s1, s2);
        System.out.println("Compare Exact:-");
        sc.compareExact();
        System.out.println("\nCompare Ignore Case:-");
        sc.compareIgnoreCase();
    }
}
```

**Output:-**


**Q3. Dictionary Sorter (using compareTo())**

**Concepts: class, array of objects, sorting logic**

**Problem:**

**Define a class DictionarySorter that accepts multiple words through a constructor.**

**Implement a method sortWords() that sorts them alphabetically using compareTo().**

**Display:**

**The sorted list**

**The smallest and largest word**

**Code:-**

```
public class DictionarySorter {
   private String[] words;

   public DictionarySorter(String[] words) {
      this.words = words;
   }

   public void sortWords() {
      int n = words.length;

      for (int i = 0; i < n - 1; i++) {
         for (int j = 0; j < n - i - 1; j++) {
            if (words[j].compareTo(words[j+1]) > 0) {
               String temp = words[j];
               words[j] = words[j+1];
               words[j+1] = temp;
            }
         }
      }
   }

   public void display() {
      sortWords();
```

```
        for (String word : words) {
            System.out.print(word + "\t");
        }
        System.out.println("\nSmallest Word: " + words[0]);
        System.out.println("Longest Word: " + words[words.length-1]);
    }

    public static void main(String[] args) {
        String[] words = {
            "Car",
            "Aeroplane",
            "Truck",
            "Motorbike"
        };

        DictionarySorter ds = new DictionarySorter(words);
        ds.display();
    }
}
```

**Output:-**


**Q4. Palindrome Substrings Finder (using substring())**

**Concepts: class, methods, nested loops, string manipulation**

**Problem:**

**Create a class PalindromeSubstrings that:**

**Accepts a string through constructor**

**Finds and prints all substrings that are palindromes**

**Prints the total count of palindrome substrings**

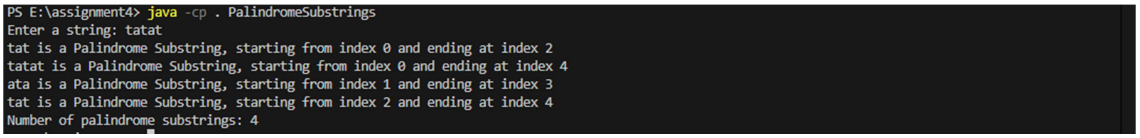
**(Hint: use nested loops + substring())**

**Code:-**

```
import java.util.Scanner;


public class PalindromeSubstrings {
   private String str;


   public PalindromeSubstrings(String str) {
      this.str = str;
   }


   private boolean isPalindrome(String str) {
      String reversed = new StringBuilder(str).reverse().toString();
      return str.equals(reversed);
   }


   public void printPalindromeSubstrings() {
      int len = str.length();
      int count = 0;
      for (int i = 0; i < len; i++) {
         for (int j = i + 1; j < len; j++) {
            String tempStr = str.substring(i, j+1);
            if (isPalindrome(tempStr)) {
               System.out.println(tempStr + " is a Palindrome Substring, starting from index " + i + " and
ending at index " + j);
               count++;
            }
         }
      }
      System.out.println("Number of palindrome substrings: " + count);
   }
```

```
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter a string: ");
        String str = scanner.nextLine();
        scanner.close();
        PalindromeSubstrings ps = new PalindromeSubstrings(str);
        ps.printPalindromeSubstrings();
    }
}
```

**Output:-**




**Q5. Word Occurrence Finder (using indexOf() and lastIndexOf())**

**Concepts: class, encapsulation, indexOf logic**

**Problem:**

**Create a class SubstringFinder with a method findOccurrences(String main, String sub)**

**→ Prints all starting and ending indices of every occurrence of sub in main.**

**Handle overlapping substrings as well.**

**Code:-**

```
import java.util.Scanner;

public class SubstringFinder {
    public void findOccurrences(String main, String sub) {
        int index = 0;

        while (index < main.length()) {
            int foundIndex = main.indexOf(sub, index);
            if (foundIndex == -1) {
                return;
```

```
        }
        int endIndex = foundIndex + sub.length() - 1;
        index = foundIndex + 1;

        System.out.println("Substring found at start index " + foundIndex + " and end index " +
endIndex);
      }
   }

   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);
      System.out.print("Enter the main string: ");
      String main = scanner.nextLine();
      System.out.print("Enter the sub string: ");
      String sub = scanner.nextLine();
      scanner.close();
      SubstringFinder sf = new SubstringFinder();
      sf.findOccurrences(main, sub);
   }
}
```

**Output:-**


**Q6. Email Validator (using matches() and regex)**

**Concepts: class, constructor, regex validation**

**Problem:**

**Design a class EmailValidator which takes an email as input and checks if it's valid**

**using the matches() method.**

**If invalid, print the reason (missing @, invalid domain, etc.).**

**(Hint: Use regex and if conditions for descriptive output.)**

**Code:-**

```
import java.util.regex.*;

public class EmailValidator {
   private String email;

   public EmailValidator(String email) {
      this.email = email;
   }

   public void validate() {
      String regex = "^[A-Za-z0-9+_.-]+@[A-Za-z0-9.-]+\\.[A-Za-z]{2,}$";

      if (!email.contains("@")) {
         System.out.println("Invalid: Missing '@' symbol.");
         return;
      }

      if (email.startsWith("@") || email.endsWith("@")) {
         System.out.println("Invalid: '@' cannot be at the start or end.");
         return;
      }

      int atIndex = email.indexOf('@');
      String domainPart = email.substring(atIndex + 1);
      if (!domainPart.contains(".")) {
         System.out.println("Invalid: Missing domain extension (e.g., '.com').");
         return;
      }

      if (email.matches(regex)) {
         System.out.println("Valid email address!");
```

```
        } else {
            System.out.println("Invalid: Email format is incorrect.");
        }
    }

    public static void main(String[] args) {
        EmailValidator e1 = new EmailValidator("john.doe@gmail.com");
        e1.validate();

        EmailValidator e2 = new EmailValidator("john.doe@gmail");
        e2.validate();

        EmailValidator e3 = new EmailValidator("johndoegmail.com");
        e3.validate();

        EmailValidator e4 = new EmailValidator("@example.com");
        e4.validate();
    }
}
```

**Output:-**


**Q7. Anagram Checker (using toCharArray() and sorting)**

**Concepts: class, methods, arrays, logic**

**Problem:**

**Create a class AnagramCheck with two string members.**

**Write a method isAnagram() that checks if both strings are anagrams of each other (case-insensitive).**

**Use toCharArray(), sort, and compare.**

**Code:-**

```
import java.util.Arrays;
```

```
public class AnagramCheck {
   private String str1;
   private String str2;

   public AnagramCheck(String str1, String str2) {
      this.str1 = str1;
      this.str2 = str2;
   }

   public boolean isAnagram() {
      String s1 = str1.toLowerCase();
      String s2 = str2.toLowerCase();

      char []c1 = s1.toCharArray();
      char []c2 = s2.toCharArray();

      Arrays.sort(c1);
      Arrays.sort(c2);

      return Arrays.equals(c1, c2);
   }

   public static void main(String args[]) {
      AnagramCheck ac = new AnagramCheck("Listen", "Silent");

      if (ac.isAnagram()) {
         System.out.println("Listen and Silent are anagram");
      } else {
         System.out.println("Listen and Silent are not anagram");
      }
```

```
    }
}
```

**Output:-**

```
PS E:\assignment4> java -cp . AnagramCheck
Listen and Silent are anagram
```

**Q8. Filename Filter (using startsWith() and endsWith())**

**Concepts: class, arrays, filtering**

**Problem:**

**Define a class FileFilter which stores an array of filenames.**

**Implement a method filterFiles() that prints only those starting with "temp" and ending with ".txt".**

**Code:-**

```
public class FileFilter {
    private String[] fileNames;

    public FileFilter(String []fileNames) {
        this.fileNames = fileNames;
    }

    public void filterFiles() {
        System.out.println("Files that start with temp and ends with .txt:-");
        for (String file : fileNames) {
            if (file.startsWith("temp") && file.endsWith(".txt")) {
                System.out.print(file + " ");
            }
        }
    }


    public static void main(String args[]) {
        String []fileNames = {"temp1.txt", "temp2.txt", "abr.mp3", "man.txt"};
        FileFilter ff = new FileFilter(fileNames);
```

```
        System.out.print("All the files: ");
        for (String file : fileNames) {
            System.out.print(file + " ");
        }
        System.out.println();
        ff.filterFiles();
    }
}
```

**Output:-**

```
PS E:\assignment4> java -cp . FileFilter
All the files: temp1.txt temp2.txt abr.mp3 man.txt
Files that start with temp and ends with .txt:-
temp1.txt temp2.txt
```

**Q9. String Cleaner (using replaceAll() with regex)**

**Concepts: class, data sanitization, regex**

**Problem:**

**Create a class StringCleaner with a method cleanText(String data) that**

**removes:**

**Digits**

**Punctuation marks**

**Multiple spaces**

**Use regex with replaceAll() and print the cleaned text.**

**Code:-**

```
public class StringCleaner {


    public void cleanText(String data) {
        String cleaned = data
            .replaceAll("\\d", "")
            .replaceAll("\\p{Punct}", "")
            .replaceAll("\\s+", " ")
            .trim();
        System.out.println(cleaned);
    }
```

```
    public static void main(String args[]) {
        String str = "Hello!!! 123   world... How   are you??";
        StringCleaner sc = new StringCleaner();
        System.out.println("Uncleaned String: " + str);
        sc.cleanText(str);
    }
}
```

**Output:-**


**Q10. Performance Test (String vs StringBuilder – Immutability)**

**Concepts: class, performance comparison, immutability**

**Problem:**

**Make a class PerformanceTest that:**

**Concatenates 10,000 words using String**

**Repeats using StringBuilder**

**Prints time taken by each**

**Demonstrate how immutability affects speed.**

**Code:-**

```
import java.time.LocalDateTime;
import java.time.Duration;

public class PerformanceTest {
    public void timeTakenByString() {
        LocalDateTime startTime = LocalDateTime.now();
        String testStr = "";
        for (int i = 0; i < 10000; i++) {
            testStr = testStr + i;
        }
        Duration diff = Duration.between(startTime, LocalDateTime.now());
        System.out.println("Milli Seconds ellapsed for string concatenation: " + diff.toMillis());
```

```
    }

    public void timeTakenByStringBuilder() {
        LocalDateTime startTime = LocalDateTime.now();
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < 10000; i++) {
            sb.append(i);
        }
        Duration diff = Duration.between(startTime, LocalDateTime.now());
        System.out.println("Milli Seconds ellapsed for concatenation via string builder: " +
diff.toMillis());
    }

    public static void main(String args[]) {
        PerformanceTest pt = new PerformanceTest();
        pt.timeTakenByString();
        pt.timeTakenByStringBuilder();
    }
}
```

**Output:-**


**Q11. Password Strength Analyzer (using multiple String methods)**

**Concepts: class design, encapsulation, condition checks**

**Problem:**

**Create a class PasswordAnalyzer with:**

**A constructor to take password input**

**Method analyze() that checks:**

**Minimum 8 characters (length())**

**Contains upper & lower case (matches())**

**Contains digits and special chars (matches("[0-9]"), matches("[^a-zA-Z0-9]"))**

**Print strength level: Weak / Moderate / Strong**

**Code:-**

```
public class PasswordAnalyzer {
   private String password;
   public PasswordAnalyzer(String password) {
      this.password = password;
   }
   public void analyze() {
      int minLenght = 0, hasUpperAndLower = 0, hasSpecialChars = 0;

      if (password.length() >= 8) {
         minLenght = 1;
      }

      if (password.matches("(?=.*[A-Z])(?=.*[a-z]).*")) {
         hasUpperAndLower = 1;
      }

      if (password.matches("[0-9]") && password.matches("[^a-zA-Z0-9]")) {
         hasSpecialChars = 1;
      }

      int score = minLenght + hasUpperAndLower + hasSpecialChars;

      if (score == 0) {
         System.out.println("Password is invalid (doesnt matches any of the required criteria)!");
      } else if (score == 1) {
         System.out.println("Password Strength: Weak!");
      } else if (score == 2) {
         System.out.println("Password Strength: Moderate!");
      } else {
```

```
            System.out.println("Password Strength: Strong!");
        }
    }
    public static void main(String args[]) {
        PasswordAnalyzer pa = new PasswordAnalyzer("rishabh1");
        pa.analyze();
    }
}
```

**Output:-**


**Q12. Unique Word Extractor (using split() and equalsIgnoreCase())**

**Concepts: class, string splitting, duplication removal**

**Problem:**

**Build a class UniqueWordExtractor that:**

**Splits a sentence into words using split()**

**Removes duplicates (case-insensitive)**

**Prints the unique words in order of appearance**

**Code:-**

```
import java.util.Scanner;

public class UniqueWordExtractor {
    private String str;

    public UniqueWordExtractor(String str) {
        this.str = str;
    }

    public void extractUniqueWords() {
        String []uniqueWords = new String[str.length()];
        int uniqueCount = 0;
        for (String word : str.split(" ")) {
```

```
            boolean notInArray = true;
            for (int i = 0; i < uniqueCount; i++) {
                if (word.equalsIgnoreCase(uniqueWords[i])) {
                    notInArray = false;
                    break;
                }
            }
            if (notInArray) {
                uniqueWords[uniqueCount] = word;
                uniqueCount++;
            }
        }
        System.out.println("Unique words in encountered order:-");
        for (int i =0; i < uniqueCount; i++) {
            System.out.println(uniqueWords[i]);
        }
    }

    public static void main(String []args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter a sentence: ");
        UniqueWordExtractor uwe = new UniqueWordExtractor(scanner.nextLine());
        scanner.close();
        uwe.extractUniqueWords();
    }
}
```

**Output:-**


**Q13. Sentence Pattern Reverser**

**Concepts: class, multiple String functions, logic**

**Problem:**

**Write a class SentencePattern that:**

**Takes a full sentence**

**Reverses each word individually (not the sentence order)**

**Removes extra spaces**

**Capitalizes the first letter of each word**

**(Uses: split(), trim(), substring(), toUpperCase(), toLowerCase())**

**Code:-**

```
import java.util.Scanner;

public class SentencePattern {
   private String str;

   public SentencePattern(String str) {
      this.str = str;
   }

   private String reverseAndCapitalizeFirstLetter(String s) {
      String word = new StringBuilder(s).reverse().toString();
      String result = word.substring(0, 1).toUpperCase() + word.substring(1);
      return result;
   }

   public void sentencePattern() {
      String[] sentenceParts = str.trim().toLowerCase().split("\\s+");
      for (String word : sentenceParts) {
         System.out.print(reverseAndCapitalizeFirstLetter(word) + " ");
      }
   }
   public static void main(String[] args) {
```

```
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter a sentence: ");
        SentencePattern sp = new SentencePattern(scanner.nextLine());
        sp.sentencePattern();
        scanner.close();
    }
}
```

**Output:-**

```
PS E:\assignment4> java -cp . SentencePattern
Enter a sentence: nac uoy dnif nrettap
Can You Find Pattern
```

---

# Assignment – 5

---

1. Thread Creation using Thread class

Write a Java program to create two threads:

o Thread 1 prints numbers from 1 to 10

o Thread 2 prints alphabets from A to J

Run both threads simultaneously.

Code:

```
class NumberThread extends Thread {
    public void run() {
        for (int i = 1; i <= 10; i++) {
            System.out.println("Number Thread: " + i);
        }
    }
}
```

```
class AlphabetThread extends Thread {
    public void run() {
        for (char c = 'A'; c <= 'J'; c++) {
            System.out.println("Alphabet Thread: " + c);
        }
    }
}

public class Program1 {
    public static void main(String[] args) {
        new NumberThread().start();
        new AlphabetThread().start();
    }
}
```

2. Thread Creation using Runnable interface

Create a program where one thread displays even numbers and another thread displays odd numbers

up to 20 using the Runnable interface.

Code:

```
class EvenRunnable implements Runnable {
    public void run() {
        for (int i = 2; i <= 20; i += 2) {
            System.out.println("Even: " + i);
        }
    }
}
```

```
class OddRunnable implements Runnable {
    public void run() {
        for (int i = 1; i <= 20; i += 2) {
            System.out.println("Odd: " + i);
        }
    }
}

public class Program2 {
    public static void main(String[] args) {
        Thread even = new Thread(new EvenRunnable());
        Thread odd = new Thread(new OddRunnable());

        even.start();
        odd.start();
    }
}
```

3. Thread Priority Example

Create three threads with different priorities (MIN_PRIORITY, NORM_PRIORITY, and MAX_PRIORITY) and observe the order of execution.

Code:

```
class Task extends Thread {
    public Task(String name) {
        super(name);
    }

    public void run() {
        System.out.println("Running: " + this.getName());
    }
```

```
}

public class Program3 {
    public static void main(String[] args) {
        Task t1 = new Task("Low Priority");
        t1.setPriority(Thread.MIN_PRIORITY);

        Task t2 = new Task("Normal Priority");
        t2.setPriority(Thread.NORM_PRIORITY);

        Task t3 = new Task("High Priority");
        t3.setPriority(Thread.MAX_PRIORITY);

        t1.start();
        t2.start();
        t3.start();
    }
}
```

4. Thread Sleep and Join Example

Create a thread that counts from 1 to 5 with a delay of 1 second between counts using Thread.sleep().

Use join() in the main thread to wait until the counting thread finishes.

Code:

```
class CounterThread extends Thread {
    public void run() {
        for (int i = 1; i <= 5; i++) {
            System.out.println("Count: " + i);
            try {
```

```
            Thread.sleep(1000);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
  }
}

public class Program4 {
  public static void main(String[] args) {
    CounterThread t = new CounterThread();
    t.start();

    try {
      t.join(); // main waits
    } catch (Exception e) {
      e.printStackTrace();
    }

    System.out.println("Counting completed!");
  }
}
```

5. Synchronization Example – Banking System

Write a Java program where multiple threads try to withdraw money from a shared bank account object.

Use synchronization to prevent race conditions.

Code:

```
class Bank {
  private int balance = 1000;
```

```
    public synchronized void withdraw(int amount) {
        if (balance >= amount) {
            System.out.println(Thread.currentThread().getName() + " withdrawing " + amount);
            balance -= amount;
            System.out.println("Remaining balance: " + balance);
        } else {
            System.out.println(Thread.currentThread().getName() + " insufficient balance!");
        }
    }
}

class WithdrawTask extends Thread {
    Bank bank;

    public WithdrawTask(Bank bank) {
        this.bank = bank;
    }

    public void run() {
        bank.withdraw(700);
    }
}

public class Program5 {
    public static void main(String[] args) {
        Bank bank = new Bank();

        Thread t1 = new WithdrawTask(bank);
        Thread t2 = new WithdrawTask(bank);
```

```
        t1.start();
        t2.start();
    }
}
```

6. Producer-Consumer Problem

Implement the classic producer-consumer problem using wait() and notify().

o Producer adds items to a shared buffer

o Consumer removes items from the buffer

Ensure proper thread synchronization.

Code:

```
class Buffer {
    int item;
    boolean available = false;

    public synchronized void produce(int value) {
        while (available) {
            try { wait(); } catch (Exception e) {}
        }
        item = value;
        available = true;
        System.out.println("Produced: " + value);
        notify();
    }

    public synchronized int consume() {
        while (!available) {
            try { wait(); } catch (Exception e) {}
        }
        available = false;
```

```
        System.out.println("Consumed: " + item);
        notify();
        return item;
    }
}

class Producer extends Thread {
    Buffer b;
    public Producer(Buffer b) { this.b = b; }
    public void run() {
        for (int i = 1; i <= 5; i++) {
            b.produce(i);
            try { Thread.sleep(500); } catch (Exception e) {}
        }
    }
}

class Consumer extends Thread {
    Buffer b;
    public Consumer(Buffer b) { this.b = b; }
    public void run() {
        for (int i = 1; i <= 5; i++) {
            b.consume();
            try { Thread.sleep(500); } catch (Exception e) {}
        }
    }
}

public class Program6 {
    public static void main(String[] args) {
        Buffer b = new Buffer();
```

```
        new Producer(b).start();
        new Consumer(b).start();
    }
}
```

7. Thread Communication Example

Create two threads:

o One thread prints numbers from 1 to 10

o Another thread prints “Halfway there” when count reaches 5

Use inter-thread communication for coordination.

Code:

```
class NumberPrinter extends Thread {
    int count = 0;
    boolean halfway = false;

    public void run() {
        for (int i = 1; i <= 10; i++) {
            count = i;
            System.out.println(i);

            if (i == 5) {
                halfway = true;
                synchronized (this) {
                    notify();
                }
            }

            try { Thread.sleep(300); } catch (Exception e) {}
        }
    }
```

```
}

class MessageThread extends Thread {
   NumberPrinter np;
   public MessageThread(NumberPrinter np) { this.np = np; }

   public void run() {
      synchronized (np) {
         try { np.wait(); } catch (Exception e) {}
      }
      System.out.println("Halfway there!");
   }
}

public class Program7 {
   public static void main(String[] args) {
      NumberPrinter np = new NumberPrinter();
      MessageThread mt = new MessageThread(np);

      np.start();
      mt.start();
   }
}
```

8. Multithreading with File Handling

Write a Java program that reads data from one file using one thread and writes the data to another file using another thread simultaneously.

Code:

```
import java.io.*;
```

```
class FileReaderThread extends Thread {
   String content = "";
   public void run() {
      try {
         BufferedReader br = new BufferedReader(new FileReader("input.txt"));
         String line;
         while ((line = br.readLine()) != null) {
            content += line + "\n";
         }
         br.close();
      } catch (Exception e) { e.printStackTrace(); }
   }
}

class FileWriterThread extends Thread {
   FileReaderThread reader;

   public FileWriterThread(FileReaderThread reader) {
      this.reader = reader;
   }

   public void run() {
      try {
         reader.join();
         FileWriter fw = new FileWriter("output.txt");
         fw.write(reader.content);
         fw.close();
         System.out.println("Data written to output file");
      } catch (Exception e) { e.printStackTrace(); }
   }
}
```

```
public class Program8 {
   public static void main(String[] args) {
      FileReaderThread fr = new FileReaderThread();
      FileWriterThread fw = new FileWriterThread(fr);

      fr.start();
      fw.start();
   }
}
```

9. Matrix Multiplication using Threads

Perform matrix multiplication using multiple threads — one thread per row or per cell.

Code:

```
class MultiplyRow extends Thread {
   int row;
   int[][] A, B, C;

   public MultiplyRow(int row, int[][] A, int[][] B, int[][] C) {
      this.row = row;
      this.A = A;
      this.B = B;
      this.C = C;
   }

   public void run() {
      for (int j = 0; j < B[0].length; j++) {
         C[row][j] = 0;
         for (int k = 0; k < B.length; k++) {
            C[row][j] += A[row][k] * B[k][j];
```

```
        }
      }
    }
}

public class Program9 {
    public static void main(String[] args) {
        int[][] A = { {1, 2}, {3, 4} };
        int[][] B = { {5, 6}, {7, 8} };
        int[][] C = new int[2][2];

        Thread[] threads = new Thread[2];

        for (int i = 0; i < 2; i++) {
            threads[i] = new MultiplyRow(i, A, B, C);
            threads[i].start();
        }

        try {
            for (Thread t : threads) t.join();
        } catch (Exception e) {}

        System.out.println("Result Matrix:");
        for (int[] row : C) {
            for (int x : row) System.out.print(x + " ");
            System.out.println();
        }
    }
}
```

10. Thread Pool Example (Executor Framework)

Create a fixed thread pool of 3 threads using Executors.newFixedThreadPool().

Submit 5 tasks that print the thread name and task number.

Code:

```
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

class Task implements Runnable {
   int taskNumber;
   public Task(int n) { taskNumber = n; }

   public void run() {
      System.out.println("Task " + taskNumber + " executed by " +
                  Thread.currentThread().getName());
   }
}

public class Program10 {
   public static void main(String[] args) {

      ExecutorService pool = Executors.newFixedThreadPool(3);

      for (int i = 1; i <= 5; i++) {
         pool.execute(new Task(i));
      }

      pool.shutdown();
   }
}
```

11. Daemon Thread Example

Create a daemon thread that continuously prints “Background task running...” while the main thread

executes a separate task.

Code:

```
class DaemonTask extends Thread {
   public void run() {
      while (true) {
         System.out.println("Background task running...");
         try { Thread.sleep(500); } catch (Exception e) {}
      }
   }
}

public class Program11 {
   public static void main(String[] args) {
      DaemonTask d = new DaemonTask();
      d.setDaemon(true); // daemon thread
      d.start();

      System.out.println("Main thread doing some work...");
      try { Thread.sleep(2000); } catch (Exception e) {}

      System.out.println("Main thread finished.");
   }
}
```

12. Deadlock Simulation

Write a Java program that demonstrates a deadlock situation between two threads, and then modify it

to resolve the deadlock using proper synchronization order.

Code:

```
class ResourceA {}
class ResourceB {}

class Thread1 extends Thread {
   ResourceA a;
   ResourceB b;

   Thread1(ResourceA a, ResourceB b) { this.a = a; this.b = b; }

   public void run() {
      synchronized (a) {
         System.out.println("Thread1 locked A");
         try { Thread.sleep(100); } catch (Exception e) {}

         synchronized (b) {
            System.out.println("Thread1 locked B");
         }
      }
   }
}

class Thread2 extends Thread {
   ResourceA a;
   ResourceB b;

   Thread2(ResourceA a, ResourceB b) { this.a = a; this.b = b; }

   public void run() {
      synchronized (b) {
         System.out.println("Thread2 locked B");
```

```
            try { Thread.sleep(100); } catch (Exception e) {}

            synchronized (a) {
                System.out.println("Thread2 locked A");
            }
        }
    }
}

public class Program12_Deadlock {
    public static void main(String[] args) {
        ResourceA a = new ResourceA();
        ResourceB b = new ResourceB();

        new Thread1(a, b).start();
        new Thread2(a, b).start();
    }
}
```

B:

```
class FixedThread1 extends Thread {
    ResourceA a;
    ResourceB b;

    FixedThread1(ResourceA a, ResourceB b) { this.a = a; this.b = b; }

    public void run() {
        synchronized (a) {
            System.out.println("Thread1 locked A");
            synchronized (b) {
```

```
            System.out.println("Thread1 locked B");
        }
      }
    }
}

class FixedThread2 extends Thread {
    ResourceA a;
    ResourceB b;

    FixedThread2(ResourceA a, ResourceB b) { this.a = a; this.b = b; }

    public void run() {
        synchronized (a) {
            System.out.println("Thread2 locked A");
            synchronized (b) {
                System.out.println("Thread2 locked B");
            }
        }
    }
}

public class Program12_Resolved {
    public static void main(String[] args) {
        ResourceA a = new ResourceA();
        ResourceB b = new ResourceB();

        new FixedThread1(a, b).start();
        new FixedThread2(a, b).start();
    }
}
```

13. Multithreading with Anonymous Class / Lambda Expression

Use anonymous inner classes or lambda expressions to create and run threads that perform simple calculations (like factorial or sum of array).

Code:

```
public class Program13 {
   public static void main(String[] args) {

      // Lambda: factorial
      Thread factorial = new Thread(() -> {
         int n = 5, fact = 1;
         for (int i = 1; i <= n; i++) fact *= i;
         System.out.println("Factorial: " + fact);
      });

      // Anonymous class: sum of array
      Thread sumThread = new Thread(new Runnable() {
         public void run() {
            int[] arr = {1, 2, 3, 4, 5};
            int sum = 0;
            for (int x : arr) sum += x;
            System.out.println("Sum of array: " + sum);
         }
      });

      factorial.start();
      sumThread.start();
   }
}
```

Q14. Write a program to implement a stack using arrays with the following operations:

 push() – to insert an element into the stack

Code:

```
import java.util.Scanner;

class ArrayStack {
   int top = -1;
   int size;
   int stack[];

   ArrayStack(int size) {
      this.size = size;
      stack = new int[size];
   }

   void push(int x) {
      if (top == size - 1) {
         System.out.println("Stack Overflow");
         return;
      }
      stack[++top] = x;
   }

   int pop() {
      if (top == -1) {
         System.out.println("Stack Underflow");
         return -1;
      }
      return stack[top--];
   }
```

```
    int peek() {
        if (top == -1) return -1;
        return stack[top];
    }

    void display() {
        if (top == -1) {
            System.out.println("Stack is empty.");
            return;
        }
        System.out.println("Stack elements:");
        for (int i = top; i >= 0; i--)
            System.out.println(stack[i]);
    }
}

public class Program14 {
    public static void main(String[] args) {
        ArrayStack s = new ArrayStack(5);

        s.push(10);
        s.push(20);
        s.push(30);
        s.display();

        System.out.println("Popped: " + s.pop());
        System.out.println("Peek: " + s.peek());
        s.display();
    }
}
```

 pop() – to remove the top element

 peek() – to view the top element

 display() – to display all elements of the stack

Q15. Implement a stack using Linked List and perform push and pop operations.

Code:

```
class Node {
   int data;
   Node next;
   Node(int d) { data = d; }
}

class LinkedListStack {
   Node top = null;

   void push(int x) {
      Node n = new Node(x);
      n.next = top;
      top = n;
   }

   int pop() {
      if (top == null) {
         System.out.println("Stack Underflow");
         return -1;
      }
      int val = top.data;
      top = top.next;
      return val;
   }
```

```
    void display() {
        Node temp = top;
        if (temp == null) {
            System.out.println("Stack is empty");
            return;
        }
        while (temp != null) {
            System.out.println(temp.data);
            temp = temp.next;
        }
    }
}

public class Program15 {
    public static void main(String[] args) {
        LinkedListStack s = new LinkedListStack();

        s.push(5);
        s.push(10);
        s.push(15);

        s.display();

        System.out.println("Popped: " + s.pop());
        s.display();
    }
}
```

Q16. Write a Java program to check whether a given string is a palindrome using a stack.

Code:

```
import java.util.Stack;

public class Program16 {
   public static void main(String[] args) {
      String str = "MADAM";
      Stack<Character> stack = new Stack<>();

      for (char c : str.toCharArray())
         stack.push(c);

      String reversed = "";
      while (!stack.isEmpty())
         reversed += stack.pop();

      if (str.equals(reversed))
         System.out.println(str + " is a palindrome");
      else
         System.out.println(str + " is NOT a palindrome");
   }
}
```

Q17. Write a program to implement a queue using arrays with operations:

- enqueue() – insert element
- dequeue() – remove element
- display() – show all elements

Code:

```
class ArrayQueue {
   int front = 0, rear = -1, size = 0;
   int capacity;
   int arr[];
```

```
ArrayQueue(int capacity) {
    this.capacity = capacity;
    arr = new int[capacity];
}

void enqueue(int x) {
    if (size == capacity) {
        System.out.println("Queue Overflow");
        return;
    }
    rear = (rear + 1) % capacity;
    arr[rear] = x;
    size++;
}

int dequeue() {
    if (size == 0) {
        System.out.println("Queue Underflow");
        return -1;
    }
    int val = arr[front];
    front = (front + 1) % capacity;
    size--;
    return val;
}

void display() {
    if (size == 0) {
        System.out.println("Queue is empty.");
        return;
```

```
        }
        System.out.println("Queue elements:");
        for (int i = 0; i < size; i++)
            System.out.println(arr[(front + i) % capacity]);
    }
}

public class Program17 {
    public static void main(String[] args) {
        ArrayQueue q = new ArrayQueue(5);
        q.enqueue(10);
        q.enqueue(20);
        q.enqueue(30);
        q.display();

        System.out.println("Dequeued: " + q.dequeue());
        q.display();
    }
}
```

Q18. Implement a circular queue using arrays. Perform insertion, deletion, and display operations.

Code:

```
class CircularQueue {
    int front = -1, rear = -1;
    int size;
    int arr[];

    CircularQueue(int size) {
        this.size = size;
        arr = new int[size];
    }
```

```
void enqueue(int x) {
    if ((rear + 1) % size == front) {
        System.out.println("Queue is full");
        return;
    }
    if (front == -1) front = 0;
    rear = (rear + 1) % size;
    arr[rear] = x;
}

int dequeue() {
    if (front == -1) {
        System.out.println("Queue is empty");
        return -1;
    }
    int val = arr[front];

    if (front == rear)
        front = rear = -1;
    else
        front = (front + 1) % size;

    return val;
}

void display() {
    if (front == -1) {
        System.out.println("Queue empty");
        return;
    }
    System.out.println("Circular Queue:");
```

```
        int i = front;
        while (true) {
            System.out.println(arr[i]);
            if (i == rear) break;
            i = (i + 1) % size;
        }
    }
}

public class Program18 {
    public static void main(String[] args) {
        CircularQueue cq = new CircularQueue(5);

        cq.enqueue(1);
        cq.enqueue(2);
        cq.enqueue(3);
        cq.display();

        System.out.println("Dequeued: " + cq.dequeue());
        cq.display();
    }
}
```

Q19. Write a Java program to implement a priority queue where higher numbers have higher priority.

Code:

```
class PriorityQueue {
    int size;
    int[] arr;
    int count = 0;
```

```
PriorityQueue(int size) {
   this.size = size;
   arr = new int[size];
}

void insert(int x) {
   if (count == size) {
      System.out.println("Queue Full");
      return;
   }
   int i;
   for (i = count - 1; i >= 0 && arr[i] < x; i--) {
      arr[i + 1] = arr[i];
   }
   arr[i + 1] = x;
   count++;
}

int remove() {
   if (count == 0) {
      System.out.println("Queue Empty");
      return -1;
   }
   return arr[--count];
}

void display() {
   System.out.println("Priority Queue:");
   for (int i = 0; i < count; i++)
      System.out.println(arr[i]);
```

```
    }
}

public class Program19 {
    public static void main(String[] args) {
        PriorityQueue pq = new PriorityQueue(5);
        pq.insert(30);
        pq.insert(10);
        pq.insert(50);
        pq.insert(40);

        pq.display();
    }
}
```

Q20. Write a program to implement a singly linked list with the following operations:

- Insert at beginning
- Insert at end
- Delete from beginning
- Delete from end
- Display the list

Code:

```
class Node {
    int data;
    Node next;
    Node(int d) { data = d; }
}

class SinglyLinkedList {
    Node head = null;
```

```
void insertAtBeginning(int x) {
   Node n = new Node(x);
   n.next = head;
   head = n;
}

void insertAtEnd(int x) {
   Node n = new Node(x);
   if (head == null) {
      head = n;
      return;
   }
   Node temp = head;
   while (temp.next != null) temp = temp.next;
   temp.next = n;
}

void deleteFromBeginning() {
   if (head == null) return;
   head = head.next;
}

void deleteFromEnd() {
   if (head == null) return;
   if (head.next == null) {
      head = null;
      return;
   }
   Node temp = head;
   while (temp.next.next != null) temp = temp.next;
```

```
        temp.next = null;
    }

    void display() {
        Node temp = head;
        while (temp != null) {
            System.out.println(temp.data);
            temp = temp.next;
        }
    }
}

public class Program20 {
    public static void main(String[] args) {
        SinglyLinkedList list = new SinglyLinkedList();

        list.insertAtBeginning(10);
        list.insertAtEnd(20);
        list.insertAtEnd(30);
        list.display();

        System.out.println("After deletion:");
        list.deleteFromBeginning();
        list.deleteFromEnd();
        list.display();
    }
}
```

Q21. Implement a doubly linked list and perform insertion and deletion operations at both ends.

Code:

```
class DNode {
    int data;
```

```
    DNode prev, next;

    DNode(int d) { data = d; }
}

class DoublyLinkedList {
    DNode head = null;

    void insertAtBeginning(int x) {
        DNode n = new DNode(x);
        if (head != null) {
            n.next = head;
            head.prev = n;
        }
        head = n;
    }

    void insertAtEnd(int x) {
        DNode n = new DNode(x);
        if (head == null) {
            head = n;
            return;
        }
        DNode temp = head;
        while (temp.next != null) temp = temp.next;
        temp.next = n;
        n.prev = temp;
    }

    void deleteFromBeginning() {
        if (head == null) return;
```

```
        head = head.next;
        if (head != null) head.prev = null;
    }

    void deleteFromEnd() {
        if (head == null) return;
        if (head.next == null) {
            head = null;
            return;
        }
        DNode temp = head;
        while (temp.next != null) temp = temp.next;
        temp.prev.next = null;
    }

    void display() {
        DNode temp = head;
        while (temp != null) {
            System.out.println(temp.data);
            temp = temp.next;
        }
    }
}

public class Program21 {
    public static void main(String[] args) {
        DoublyLinkedList dl = new DoublyLi
```

Q22. Write a program to implement a circular linked list and perform insertion and deletion.

Code:

```
class CNode {
    int data;
```

```
    CNode next;
    CNode(int d) { data = d; }
}

class CircularLinkedList {
    CNode head = null;

    void insert(int x) {
        CNode n = new CNode(x);
        if (head == null) {
            head = n;
            n.next = head;
            return;
        }
        CNode temp = head;
        while (temp.next != head) temp = temp.next;
        temp.next = n;
        n.next = head;
    }

    void delete(int x) {
        if (head == null) return;

        CNode temp = head, prev = null;

        // Deleting head node
        if (head.data == x) {
            while (temp.next != head) temp = temp.next;
            if (head == temp) {
                head = null;
                return;
```

```
            }
            temp.next = head.next;
            head = head.next;
            return;
        }

        // Deleting non-head node
        temp = head;
        do {
            prev = temp;
            temp = temp.next;
        } while (temp != head && temp.data != x);

        if (temp.data == x) prev.next = temp.next;
    }

    void display() {
        if (head == null) {
            System.out.println("Empty Circular List");
            return;
        }
        CNode temp = head;
        do {
            System.out.println(temp.data);
            temp = temp.next;
        } while (temp != head);
    }
}

public class Program22 {
    public static void main(String[] args) {
```

```
        CircularLinkedList cl = new CircularLinkedList();

        cl.insert(10);
        cl.insert(20);
        cl.insert(30);
        cl.display();

        System.out.println("After deletion:");
        cl.delete(20);
        cl.display();
    }
}
```

Q23. Write a Java program to reverse a linked list.

Code:

```
class Node23 {
    int data;
    Node23 next;

    Node23(int d) { data = d; }
}

class LinkedList23 {
    Node23 head;

    void insert(int x) {
        Node23 n = new Node23(x);
        n.next = head;
        head = n;
    }

    void reverse() {
```

```
        Node23 prev = null, current = head, nextNode;

        while (current != null) {
            nextNode = current.next;
            current.next = prev;
            prev = current;
            current = nextNode;
        }
        head = prev;
    }

    void display() {
        Node23 temp = head;
        while (temp != null) {
            System.out.println(temp.data);
            temp = temp.next;
        }
    }
}

public class Program23 {
    public static void main(String[] args) {
        LinkedList23 ll = new LinkedList23();

        ll.insert(10);
        ll.insert(20);
        ll.insert(30);

        System.out.println("Original List:");
        ll.display();
```

```
        ll.reverse();
        System.out.println("Reversed List:");
        ll.display();
    }
}
```

Q24. Write a program to implement Bubble Sort to sort an array of integers in ascending order.

Code:

```
public class Program24 {
    public static void main(String[] args) {
        int arr[] = {5, 1, 4, 2, 8};
        int n = arr.length;

        for (int i = 0; i < n - 1; i++) {
            for (int j = 0; j < n - 1 - i; j++) {
                if (arr[j] > arr[j + 1]) {
                    int temp = arr[j];
                    arr[j] = arr[j + 1];
                    arr[j + 1] = temp;
                }
            }
        }

        System.out.println("Sorted Array:");
        for (int x : arr)
            System.out.print(x + " ");
    }
}
```

Q25. Write a program to implement Selection Sort to sort an array of numbers in descending order.

Code:

```
public class Program25 {
    public static void main(String[] args) {
```

```
        int arr[] = {4, 2, 8, 1, 6};
        int n = arr.length;

        for (int i = 0; i < n - 1; i++) {
            int maxIndex = i;
            for (int j = i + 1; j < n; j++) {
                if (arr[j] > arr[maxIndex])
                    maxIndex = j;
            }

            int temp = arr[i];
            arr[i] = arr[maxIndex];
            arr[maxIndex] = temp;
        }

        System.out.println("Sorted in Descending Order:");
        for (int x : arr)
            System.out.print(x + " ");
    }
}
```

Q26. Write a program to implement Insertion Sort and display the number of passes required for sorting.

Code:

```
public class Program26 {
    public static void main(String[] args) {
        int arr[] = {5, 2, 9, 1, 5};
        int n = arr.length;
        int passes = 0;

        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;
            passes++;
        }

        System.out.println("Sorted Array:");
        for (int x : arr) System.out.print(x + " ");
        System.out.println("\nNumber of passes: " + passes);
    }
}
```

Q27. Write a program to implement Merge Sort for sorting an array in ascending order.

Code:

```
public class Program27 {

    static void merge(int arr[], int l, int m, int r) {
        int n1 = m - l + 1;
        int n2 = r - m;

        int L[] = new int[n1];
        int R[] = new int[n2];

        for (int i = 0; i < n1; i++) L[i] = arr[l + i];
        for (int j = 0; j < n2; j++) R[j] = arr[m + 1 + j];

        int i = 0, j = 0, k = l;
        while (i < n1 && j < n2) {
```

```
            if (L[i] <= R[j]) arr[k++] = L[i++];
            else arr[k++] = R[j++];
        }

        while (i < n1) arr[k++] = L[i++];
        while (j < n2) arr[k++] = R[j++];
    }

    static void mergeSort(int arr[], int l, int r) {
        if (l < r) {
            int m = l + (r - l) / 2;
            mergeSort(arr, l, m);
            mergeSort(arr, m + 1, r);
            merge(arr, l, m, r);
        }
    }

    public static void main(String[] args) {
        int arr[] = {12, 11, 13, 5, 6, 7};
        mergeSort(arr, 0, arr.length - 1);

        System.out.println("Sorted Array:");
        for (int x : arr) System.out.print(x + " ");
    }
}
```

Q28. Write a program to implement Quick Sort and display the pivot element for each pass.

Code:

```
public class Program28 {

    static int partition(int arr[], int low, int high) {
        int pivot = arr[high];
```

```
    System.out.println("Pivot: " + pivot);
    int i = low - 1;

    for (int j = low; j < high; j++) {
        if (arr[j] < pivot) {
            i++;
            int temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
        }
    }

    int temp = arr[i + 1];
    arr[i + 1] = arr[high];
    arr[high] = temp;

    return i + 1;
}

static void quickSort(int arr[], int low, int high) {
    if (low < high) {
        int pi = partition(arr, low, high);
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}

public static void main(String[] args) {
    int arr[] = {10, 7, 8, 9, 1, 5};
    quickSort(arr, 0, arr.length - 1);
```

```
        System.out.println("Sorted Array:");
        for (int x : arr) System.out.print(x + " ");
    }
}
```

Q29. Write a program to evaluate a postfix expression using stack.

Code:

```
import java.util.Stack;

public class Program29 {
    public static void main(String[] args) {
        String exp = "231*+9-"; // example postfix
        Stack<Integer> stack = new Stack<>();

        for (char c : exp.toCharArray()) {
            if (Character.isDigit(c))
                stack.push(c - '0');
            else {
                int b = stack.pop();
                int a = stack.pop();
                switch (c) {
                    case '+': stack.push(a + b); break;
                    case '-': stack.push(a - b); break;
                    case '*': stack.push(a * b); break;
                    case '/': stack.push(a / b); break;
                }
            }
        }

        System.out.println("Result: " + stack.pop());
    }
}
```

Q30. Write a program to convert infix expression to postfix using stack.

Code:

```
import java.util.Stack;

public class Program30 {

   static int precedence(char c) {
      if (c == '+' || c == '-') return 1;
      if (c == '*' || c == '/') return 2;
      return -1;
   }

   public static void main(String[] args) {
      String exp = "a+b*c";
      StringBuilder result = new StringBuilder();
      Stack<Character> stack = new Stack<>();

      for (char c : exp.toCharArray()) {
         if (Character.isLetterOrDigit(c))
            result.append(c);
         else if (c == '(')
            stack.push(c);
         else if (c == ')') {
            while (!stack.isEmpty() && stack.peek() != '(')
               result.append(stack.pop());
            stack.pop();
         } else {
            while (!stack.isEmpty() && precedence(c) <= precedence(stack.peek()))
               result.append(stack.pop());
            stack.push(c);
         }
```

```
        }

        while (!stack.isEmpty())
            result.append(stack.pop());

        System.out.println("Postfix Expression: " + result);
    }
}
```

Q31. Write a program to count the total number of nodes in a linked list.

Code:

```
class Node31 {
    int data;
    Node31 next;
    Node31(int d) { data = d; }
}

class LinkedList31 {
    Node31 head;

    void insert(int x) {
        Node31 n = new Node31(x);
        n.next = head;
        head = n;
    }

    int countNodes() {
        int count = 0;
        Node31 temp = head;
        while (temp != null) {
            count++;
            temp = temp.next;
```

```
        }
        return count;
    }

    void display() {
        Node31 temp = head;
        while (temp != null) {
            System.out.println(temp.data);
            temp = temp.next;
        }
    }
}

public class Program31 {
    public static void main(String[] args) {
        LinkedList31 ll = new LinkedList31();
        ll.insert(10);
        ll.insert(20);
        ll.insert(30);

        System.out.println("Linked List:");
        ll.display();

        System.out.println("Total nodes: " + ll.countNodes());
    }
}
```

Q32. Write a program to merge two sorted linked lists into one sorted list.

Code:

```
class Node32 {
    int data;
    Node32 next;
```

```
    Node32(int d) { data = d; }
}

class LinkedList32 {
    Node32 head;

    void insert(int x) {
        Node32 n = new Node32(x);
        if (head == null || x < head.data) {
            n.next = head;
            head = n;
            return;
        }

        Node32 temp = head;
        while (temp.next != null && temp.next.data < x)
            temp = temp.next;

        n.next = temp.next;
        temp.next = n;
    }

    static LinkedList32 mergeLists(LinkedList32 l1, LinkedList32 l2) {
        LinkedList32 result = new LinkedList32();
        Node32 dummy = new Node32(0);
        Node32 tail = dummy;
        Node32 a = l1.head, b = l2.head;

        while (a != null && b != null) {
            if (a.data <= b.data) {
                tail.next = new Node32(a.data);
```

```
            a = a.next;
        } else {
            tail.next = new Node32(b.data);
            b = b.next;
        }
        tail = tail.next;
    }

    while (a != null) {
        tail.next = new Node32(a.data);
        a = a.next;
        tail = tail.next;
    }

    while (b != null) {
        tail.next = new Node32(b.data);
        b = b.next;
        tail = tail.next;
    }

    result.head = dummy.next;
    return result;
}

void display() {
    Node32 temp = head;
    while (temp != null) {
        System.out.print(temp.data + " ");
        temp = temp.next;
    }
    System.out.println();
```

```
    }
}

public class Program32 {
    public static void main(String[] args) {
        LinkedList32 l1 = new LinkedList32();
        l1.insert(1);
        l1.insert(3);
        l1.insert(5);

        LinkedList32 l2 = new LinkedList32();
        l2.insert(2);
        l2.insert(4);
        l2.insert(6);

        System.out.println("List 1:");
        l1.display();
        System.out.println("List 2:");
        l2.display();

        LinkedList32 merged = LinkedList32.mergeLists(l1, l2);
        System.out.println("Merged Sorted List:");
        merged.display();
    }
}
```

Q33. Write a program to find the middle element of a linked list without using length.

Code:

```
class Node33 {
    int data;
    Node33 next;
    Node33(int d) { data = d; }
```

```
}

class LinkedList33 {
    Node33 head;

    void insert(int x) {
        Node33 n = new Node33(x);
        n.next = head;
        head = n;
    }

    void findMiddle() {
        if (head == null) {
            System.out.println("List is empty");
            return;
        }
        Node33 slow = head, fast = head;
        while (fast != null && fast.next != null) {
            slow = slow.next;
            fast = fast.next.next;
        }
        System.out.println("Middle element: " + slow.data);
    }

    void display() {
        Node33 temp = head;
        while (temp != null) {
            System.out.print(temp.data + " ");
            temp = temp.next;
        }
        System.out.println();
```

```
    }
}

public class Program33 {
    public static void main(String[] args) {
        LinkedList33 ll = new LinkedList33();
        ll.insert(10);
        ll.insert(20);
        ll.insert(30);
        ll.insert(40);
        ll.insert(50);
        System.out.println("Linked List:");
        ll.display();

        ll.findMiddle();
    }
}
```