|  |  |  |
| --- | --- | --- |
| **Sr.No** | **Lab program** | **Remark** |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**Problem Statement No. 1**

Define a class to represent a bank account. Include the following details like name of the depositor, account number, type of account, balance amount in the account. Write methods to assign initial values, to deposit an amount, withdraw an amount after checking the balance, to display name, account number, account type and balance.

**Class: accounter**

This class represents a bank account. It has **four attributes**:

* name → Account holder’s name
* accountNumber → Unique account number
* accountType → Type of account (e.g., saving, current)
* balance → Stores the balance amount

**Constructors:**

1. accounter() → Default constructor, initializes all attributes to default values.
2. accounter(String name, String accountNumber, String accountType) → Initializes account with name, account number, and type but sets balance to 0.
3. accounter(String name, String accountNumber, String accountType, double balance) → Initializes all attributes, including balance.

**Methods:**

1. display\_accounter() → Returns account details as a formatted string. (Syntax error: Should have { return "String"; })
2. print\_balance() → Prints the current balance.
3. deposit(double amount) → Adds money to the balance (rejects negative values).
4. withdraw(double amount) → Deducts money if sufficient balance is available.
5. checkBalance(double amount) → Checks if the balance is enough for withdrawal and returns true/false.

**Class: program1**

This is the **main class** that runs the program.

**main(String[] args) method:**

1. Creates three accounter objects with different names and account numbers.
2. Displays their details using display\_accounter().
3. Deposits money in two accounts (a1 and a2).
4. Displays updated details.
5. Withdraws 500 from a1 and displays updated details.
6. Checks if a2 has sufficient balance for withdrawal amounts (500 and 1500).

Harshwardhan Mane

612303112  
Program N0. 1

class accounter{

private String name;

private String accountNumber;

private String accountType;

private double balance;

public accounter() {

name = "";

accountNumber = "";

accountType = "";

balance = 0;

}

public accounter(String name, String accountNumber, String accountType){

this.name = name;

this.accountNumber = accountNumber;

this.accountType = accountType;

this.balance = 0;

}

public accounter(String name, String accountNumber, String accountType, double balance){

this.name = name;

this.accountNumber = accountNumber;

this.accountType = accountType;

this.balance = balance;

}

public String display\_accounter()

return "Name: " + name + "\nAccount Number: " + accountNumber + "\nAccount Type: " + accountType + "\nBalance: " + balance;

}

public void print\_balance(){

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

}

public void deposit(double amount){

if(amount >= 0){

this.balance += amount;

System.out.println("Deposited: " + amount);

}

else

System.out.println("negtive amount");

}

public void withdraw(double amount){

if(amount <= balance){

this.balance -= amount;

System.out.println("Withdrew: " + amount);

}

else

System.out.println("Insufficient balance");

}

public boolean checkBalance(double amount){

if(amount <= balance){

return true;

}

else return false;

}

}

public class program1{

public static void main(String[] args) {

accounter a1 = new accounter("a1", "1", "saving");

accounter a2 = new accounter("a2", "2", "saving");

accounter a3 = new accounter("a3", "3", "saving");

System.out.println(a1.display\_accounter());

System.out.println(a2.display\_accounter());

System.out.println(a3.display\_accounter());

a1.deposit(1000);

a2.deposit(1001);

System.out.println(a1.display\_accounter());

System.out.println(a2.display\_accounter());

a1.withdraw(500);

System.out.println(a1.display\_accounter());

System.out.println(a2.checkBalance(500));

System.out.println(a2.checkBalance(1500));

}

}

**Output:**

Name: a1

Account Number: 1

Account Type: saving

Balance: 0.0

Name: a2

Account Number: 2

Account Type: saving

Balance: 0.0

Name: a3

Account Number: 3

Account Type: saving

Balance: 0.0

Deposited: 1000.0

Deposited: 1001.0

Name: a1

Account Number: 1

Account Type: saving

Balance: 1000.0

Name: a2

Account Number: 2

Account Type: saving

Balance: 1001.0

Withdrew: 500.0

Name: a1

Account Number: 1

Account Type: saving

Balance: 500.0

true

false

**Problem Statement No. 2**

Write a program to implement following. Create a base class called person consisting of name and code. Create 2 child classes a. Account with member pay and b. Admin with experience and inherit the base class. Create a class Employee with name, code, experience and pay by inheriting the above class.

a. Write Python script to display

b. Current date and time

c. Current year

d. Month of year

e. Week number of the year

f. Weekend of the week

g. Day of year

h. Day of the month and Day of week.

Program Explanation

Class: Person

The Person class acts as the base class for different types of people in the system, such as employees and administrators. It contains two attributes: name to store the person's name and code to store a unique identifier. The constructor initializes these attributes when a Person object is created.

Class: Account

The Account class extends Person and represents individuals who have a salary or payment information. It inherits name and code from Person and introduces a new attribute, pay, which stores the person's salary. The constructor calls the parent class constructor to initialize name and code and then sets the pay attribute.

Class: Admin

The Admin class extends Person and represents administrative personnel. It adds an experience attribute to store the number of years of experience. The constructor calls the Person constructor to initialize name and code, then assigns a value to experience.

Class: Employee

The Employee class extends Account, meaning it inherits the attributes and methods from both Person and Account. In addition to name, code, and pay, it introduces the experience attribute to store years of experience. It also defines a method called displayDetails, which prints the employee's details, including name, code, pay, and experience.

Class: Manager

The Manager class extends Admin, meaning it inherits name, code, and experience from the Person and Admin classes. It introduces a new attribute, department, which stores the department the manager oversees. The class provides two methods: displayDetails, which prints the manager's name, code, experience, and department, and displayDepartmentDetails, which prints the department and calls displayDetails to show the manager's details.

Class: program2

This is the main class where objects of Employee and Manager are created. It first creates an Employee object, then displays its details using the displayDetails method. Next, it creates a Manager object and displays its details using both displayDetails and displayDepartmentDetails. Finally, it calls displayDateInfo, a method that retrieves and prints the current date, time, year, month, week number, and other related information.

Method: displayDateInfo

This method retrieves and prints various details about the current date and time, including the formatted timestamp, year, month, week number, day of the year, day of the month, and day of the week. It uses Java's LocalDateTime and DateTimeFormatter classes to format and display this information.

**Imported Packages**

java.time.LocalDateTime

This class represents the current date and time without a time zone.

It provides methods to retrieve and manipulate date-time values.

java.time.format.DateTimeFormatter

This class is used to format and parse date-time objects into human-readable strings.

It allows customization of date and time formatting.

java.time.temporal.WeekFields

This class provides access to week-based fields, such as the week number of the year.

It helps determine the current week number based on locale settings.

java.util.Locale

This class represents a specific geographical, cultural, or linguistic region.

It is used to format date and time according to the user's region.

Date and Time Operations in displayDateInfo Method

Getting the Current Date and Time

The method uses LocalDateTime.now() to get the system's current date and time.

Formatting the Date and Time

DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss") creates a pattern to format the date and time in YYYY-MM-DD HH:MM:SS format.

The format method is then used to display the formatted date and time.

Retrieving Date Components

now.getYear() retrieves the current year.

now.getMonth() gets the name of the current month.

now.getDayOfYear() returns the day number in the year.

now.getDayOfMonth() returns the current day of the month.

now.getDayOfWeek() returns the name of the day (e.g., Sunday, Monday).

Retrieving the Week Number of the Year

WeekFields.of(Locale.getDefault()) retrieves locale-specific week numbering rules.

now.get(weekFields.weekOfWeekBasedYear()) calculates the current week number

**Class Hierarchy Representation**

Harshwardhan Mane

612303112  
Program N0. 2

import java.time.LocalDateTime;

import java.time.format.DateTimeFormatter;

import java.time.temporal.WeekFields;

import java.util.Locale;

class Person {

String name;

String code;

public Person(String name, String code) {

this.name = name;

this.code = code;

}

}

class Account extends Person {

double pay;

public Account(String name, String code, double pay) {

super(name, code);

this.pay = pay;

}

}

class Admin extends Person {

int experience;

public Admin(String name, String code, int experience) {

super(name, code);

this.experience = experience;

}

}

class Employee extends Account {

int experience;

public Employee(String name, String code, double pay, int experience) {

super(name, code, pay);

this.experience = experience;

}

public void displayDetails() {

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

System.out.println("Code: " + code);

System.out.println("Pay: " + pay);

System.out.println("Experience: " + experience + " years");

}

}

class Manager extends Admin {

String department;

public Manager(String name, String code, int experience, String department) {

super(name, code, experience);

this.department = department;

}

public void displayDetails() {

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

System.out.println("Code: " + code);

System.out.println("Experience: " + experience + " years");

System.out.println("Department: " + department);

}

public void displayDepartmentDetails() {

System.out.println("Department: " + department);

System.out.println("Manager Details:");

displayDetails();

}

}

public class program2 {

public static void main(String[] args) {

// Create Employee object

Employee employee = new Employee("John Doe", "E123", 5000.00, 5);

// Display employee details

employee.displayDetails();

Manager manager = new Manager("harsh", "HR123", 3, "java devlop");

// Display manager details

manager.displayDetails();

manager.displayDepartmentDetails();

// Display date and time information

displayDateInfo();

}

public static void displayDateInfo() {

LocalDateTime now = LocalDateTime.now();

// Format the current date and time

DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");

System.out.println("\nDate and Time Information:");

System.out.println("Current Date and Time: " + now.format(formatter));

// Display current year

System.out.println("Current Year: " + now.getYear());

// Display month of year

System.out.println("Month of Year: " + now.getMonth());

// Get the week number of the year

WeekFields weekFields = WeekFields.of(Locale.getDefault());

int weekNumber = now.get(weekFields.weekOfWeekBasedYear());

System.out.println("Week Number of the Year: " + weekNumber);

// Display the day of the week (Weekend of the week)

System.out.println("Weekend of the Week: " + now.getDayOfWeek());

// Display day of the year

System.out.println("Day of the Year: " + now.getDayOfYear());

// Display day of the month

System.out.println("Day of the Month: " + now.getDayOfMonth());

// Display the day of the week

System.out.println("Day of the Week: " + now.getDayOfWeek());

}

}

**Output:**

Name: John Doe

Code: E123

Pay: 5000.0

Experience: 5 years

Name: harsh

Code: HR123

Experience: 3 years

Department: java devlop

Department: java devlop

Manager Details:

Name: harsh

Code: HR123

Experience: 3 years

Department: java devlop

Date and Time Information:

Current Date and Time: 2025-02-02 21:13:09

Current Year: 2025

Month of Year: FEBRUARY

Week Number of the Year: 6

Weekend of the Week: SUNDAY

Day of the Year: 33

Day of the Month: 2

Day of the Week: SUNDAY

**Problem Statement No. 3**

Create a Java program that defines and calculates the area and perimeter (or curved path) of various geometric shapes. The shapes include rectangles, triangles, circles, squares, pentagons, regular polygons, and arcs. The program should allow the user to input dimensions for each shape and then calculate and display the corresponding area and perimeter (or curved path).

**Explanation :**

1. **Abstract Classes**:
   * Polygon: An abstract class that defines the methods getDimensions(), calculateArea(), and calculatePerimeter().
   * curvedshape: An abstract class that defines the methods getDimensions(), calculateArea(), and calculateCurvedPath().
2. **Concrete Classes**:
   * Rectangle: Inherits from Polygon and implements methods to get dimensions, calculate area, and calculate perimeter.
   * Triangle: Inherits from Polygon and implements methods to get dimensions, calculate area, and calculate perimeter.
   * Circle: Inherits from curvedshape and implements methods to get dimensions, calculate area, and calculate the length of the curved path.
   * Square: Inherits from Rectangle and overrides methods to get dimensions, calculate area, and calculate perimeter.
   * pentagon: Inherits from Polygon and implements methods to get dimensions, calculate area, and calculate perimeter.
   * RegularPolygon: Inherits from Polygon and implements methods to get dimensions, calculate area, and calculate perimeter.
   * Arc: Inherits from curvedshape and implements methods to get dimensions, calculate area, and calculate the length of the curved path.
3. **Main Class (program3)**:
   * Contains a main method that provides a menu for the user to select a shape.
   * Based on user input, the program creates an instance of the corresponding shape class.
   * Calls the getDimensions() method to get the necessary dimensions from the user.
   * Calculates and displays the area and perimeter (or curved path) of the selected shape.
4. **Methods**:
   * calculateArea() and calculatePerimeter() are methods defined in the Polygon class and its subclasses.
   * Each shape class (e.g., Rectangle, Triangle, etc.) provides its own implementation of these methods.
   * Another method getDimensions() is used to get the dimensions of the shape from the user via input.
5. **Abstraction**:
   * Polygon is an abstract class with abstract methods calculateArea() and calculatePerimeter().
   * curvedshape is another abstract class with methods calculateArea() and calculateCurvedPath().
   * These abstract classes allow for the creation of different shapes like Rectangle, Triangle, Circle, etc., where each subclass must provide concrete implementations for the abstract methods.
   * This use of abstraction helps to define a general structure for different shape calculations while allowing the specific details to be handled by concrete subclasses, promoting code flexibility and reusability.

+---------------------+

| Abstract Class: |

| Polygon |

|---------------------|

| - getDimensions() |

| - calculateArea() |

| - calculatePerimeter|

+---------------------+

|

|

+-------------------+------------------+

| | |

+---------------------+ | +---------------------+

| Rectangle | | | Triangle |

|---------------------| | |---------------------|

| - length | | | - base |

| - breadth | | | - height |

|---------------------| | |---------------------|

| + getDimensions() | | | + getDimensions() |

| + calculateArea() | | | + calculateArea() |

| + calculatePerimeter()| | | + calculatePerimeter()|

+---------------------+ | +---------------------+

--------|-------------------+

| |

+---- +------------------+

| |

+---------------------+ +---------------------+

| Pentagon | | Square |

|---------------------| |---------------------|

| - sideLength | | - side |

+---------------------| +---------------------+

| + getDimensions() | | + getDimensions() |

| + calculateArea() | | + calculateArea() |

| + calculatePerimeter| | + calculatePerimeter|

+---------------------| +---------------------+

+---------------------+

| Abstract Class: |

| curvedshape |

|---------------------|

| - getDimensions() |

| - calculateArea() |

| - calculateCurvedPath|

+---------------------+

|

|

+-------------------+------------------+

| |

+---------------------+ +---------------------+

| Circle | | Arc |

|---------------------| |---------------------|

| - radius | | - radius |

| - PI | | - angle |

|---------------------| |---------------------|

| + getDimensions() | | + getDimensions() |

| + calculateArea() | | + calculateArea() |

| + calculateCurvedPath| | + calculateCurvedPath|

+---------------------+ +---------------------+

Harshwardhan Mane

612303112  
Program N0. 3

import *java.util.Scanner*;

*abstract* *class* Polygon {

*abstract* void getDimensions();

*abstract* double calculateArea();

*abstract* double calculatePerimeter();

}

*abstract* *class* curvedshape {

*abstract* void getDimensions();

*abstract* double calculateArea();

*abstract* double calculateCurvedPath();

}

*class* Rectangle *extends* Polygon {

*private* double length, breadth;

    @Override

    void getDimensions() {

        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter the length of the rectangle: ");

        length = scanner.nextDouble();

        System.out.print("Enter the breadth of the rectangle: ");

        breadth = scanner.nextDouble();

    }

    @Override

    double calculateArea() {

        return length \* breadth;

    }

    @Override

    double calculatePerimeter() {

        return 2 \* (length + breadth);

    }

}

*class* pentagon *extends* Polygon{

*private* double side;

*private* double sideLength;

    @Override

    void getDimensions() {

        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter the side length of the pentagon: ");

        sideLength = scanner.nextDouble();

    }

    @Override

    double calculateArea() {

        return (5 \* Math.pow(sideLength, 2)) / (4 \* Math.tan(Math.PI / 5));

    }

    @Override

    double calculatePerimeter() {

        return 5 \* sideLength;

    }

}

*class* RegularPolygon *extends* Polygon {

*private* double side;

*private* int numberOfSides;

    @Override

    void getDimensions() {

        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter the side length of the regular polygon: ");

        side = scanner.nextDouble();

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

        numberOfSides = scanner.nextInt();

    }

    @Override

    double calculateArea() {

        return (numberOfSides \* Math.pow(side, 2)) / (4 \* Math.tan(Math.PI / numberOfSides));

    }

    @Override

    double calculatePerimeter() {

        return numberOfSides \* side;

    }

}

*class* Circle *extends* curvedshape {

*private* double radius;

*private*  double PI = 3.14159;

    @Override

    void getDimensions() {

        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter the radius of the circle: ");

        radius = scanner.nextDouble();

    }

    @Override

    double calculateArea() {

        return PI \* Math.pow(radius, 2);

    }

    @Override

    double calculateCurvedPath() {

        return PI \* radius \* (360 / 360);

    }

}

*class* Arc *extends* curvedshape{

*private* double radius;

*private* double angle;

*private* double PI = 3.14159;

    @Override

    void getDimensions() {

            Scanner scanner = new Scanner(System.in);

            System.out.print("Enter the radius of the circle: ");

            radius = scanner.nextDouble();

            System.out.print("Enter the angle in degrees: ");

            angle = scanner.nextDouble();

    }

    @Override

    double calculateArea() {

        return (PI \* Math.pow(radius, 2)) \* (angle / 360);

    }

    @Override

    double calculateCurvedPath() {

        return (PI \* radius \* angle) / 360;

    }

}

*class* Square *extends* Rectangle {

*private* double side;

    @Override

    void getDimensions() {

        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter the side length of the square: ");

        side = scanner.nextDouble();

    }

    @Override

    double calculateArea() {

        return Math.pow(side, 2);

    }

    @Override

    double calculatePerimeter() {

        return 4 \* side;

    }

}

*class* Triangle *extends* Polygon {

*private* double base, height;

    @Override

    void getDimensions() {

        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter the base of the triangle: ");

        base = scanner.nextDouble();

        System.out.print("Enter the height of the triangle: ");

        height = scanner.nextDouble();

    }

    @Override

    double calculateArea() {

        return 0.5 \* base \* height;

    }

    @Override

    double calculatePerimeter() {

        return Math.sqrt(Math.pow(base, 2) + Math.pow(height, 2));

    }

}

*public*  *class* program3{

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

        Scanner scanner = new Scanner(System.in)

        while(true){

        System.out.println("Enter 0 for exit. Choose a shape:\n 1. Rectangle\n 2. Triangle\n 3.Circle\n 4.Square\n 5.Pentagon\n 6.Any regularly n side polygon\n 7.Arc");

        int choice = scanner.nextInt();

        Polygon polygon;

        curvedshape  arc;

        if (choice == 0) {

            break;

        }

        else if (choice == 1) {

            polygon = new Rectangle();

        } else if (choice == 2) {

            polygon = new Triangle();

        }

        else if (choice == 3) {

            arc = new Circle();

            arc.getDimensions();

            System.out.println("Area: " + arc.calculateArea());

            System.out.println("lenth of curved path: " + arc.calculateCurvedPath());

            continue;

        }

        else if (choice == 4) {

            polygon = new Square();

        }

        else if (choice == 5) {

            polygon = new pentagon();

        }

        else if (choice == 6) {

            polygon = new RegularPolygon();

        }

        else if (choice == 7) {

            arc = new Arc();

            arc.getDimensions();

            System.out.println("Area: " + arc.calculateArea());

            System.out.println("lenth of curved path: " + arc.calculateCurvedPath());

            continue;

        }

        else {

            System.out.println("Invalid choice! Please enter 1 t0 4.");

            continue;

        }

        polygon.getDimensions();

        System.out.println("Area: " + polygon.calculateArea());

        System.out.println("Perimeter: " + polygon.calculatePerimeter());

    }

    }

}

**OUTPUT :**

**Enter 0 for exit. Choose a shape:**

**1. Rectangle**

**2. Triangle**

**3.Circle**

**4.Square**

**5.Pentagon**

**6.Any regularly n side polygon**

**7.Arc**

**1**

**Enter the length of the rectangle: 23**

**Enter the breadth of the rectangle: 24**

**Area: 552.0**

**Perimeter: 94.0**

**Enter 0 for exit. Choose a shape:**

**1. Rectangle**

**2. Triangle**

**3.Circle**

**4.Square**

**5.Pentagon**

**6.Any regularly n side polygon**

**7.Arc**

**2**

**Enter the base of the triangle: 56**

**Enter the height of the triangle: 54**

**Area: 1512.0**

**Perimeter: 77.79460135510689**

**Enter 0 for exit. Choose a shape:**

**1. Rectangle**

**2. Triangle**

**3.Circle**

**4.Square**

**5.Pentagon**

**6.Any regularly n side polygon**

**7.Arc**

**3**

**Enter the radius of the circle: 5**

**Area: 78.53975**

**lenth of curved path: 15.70795**

**Enter 0 for exit. Choose a shape:**

**1. Rectangle**

**2. Triangle**

**3.Circle**

**4.Square**

**5.Pentagon**

**6.Any regularly n side polygon**

**7.Arc**

**7**

**Enter the radius of the circle: 65**

**Enter the angle in degrees: 60**

**Area: 2212.2029583333333**

**lenth of curved path: 34.03389166666667**

**Enter 0 for exit. Choose a shape:**

**1. Rectangle**

**2. Triangle**

**3.Circle**

**4.Square**

**5.Pentagon**

**6.Any regularly n side polygon**

**7.Arc**

**6**

**Enter the side length of the regular polygon: 123**

**Enter the number of sides: 15**

**Area: 266911.3084736233**

**Perimeter: 1845.0**

**Enter 0 for exit. Choose a shape:**

**1. Rectangle**

**2. Triangle**

**3.Circle**

**4.Square**

**5.Pentagon**

**6.Any regularly n side polygon**

**7.Arc**

**0**

**Problem Statement No. 4**

Defines a Matrix class that allows for the creation and manipulation of matrices. The class includes constructors for initializing matrices either with user input or with a constant value. The main functionality of the code is to facilitate basic matrix operations such as inputting matrix elements and initializing matrices with a specific value.

**Class: Matrix**

The Matrix class represents a mathematical matrix and includes member variables and methods for matrix operations.

**Member Variables:**

* private int rows: The number of rows in the matrix.
* private int cols: The number of columns in the matrix.
* private double[][] matrix: A 2D array to store the elements of the matrix.

**Encapsulation in the Matrix Class**

Encapsulation is a fundamental principle of object-oriented programming that involves bundling the data (variables) and methods (functions) that operate on the data into a single unit, or class. It also involves restricting direct access to some of the object's components, which is a means of preventing accidental interference and misuse of the data.

In the provided Matrix class, encapsulation is implemented as follows:

**Member Variables**

The member variables of the Matrix class are declared as private. This means they cannot be accessed directly from outside the class. Instead, access to these variables is controlled through public methods.

private int rows; // Number of rows in the matrix

private int cols; // Number of columns in the matrix

private double[][] matrix; // 2D array to store matrix elements

**Encapsulation in the Matrix class is achieved by:**

* Declaring member variables as private to restrict direct access.
* Providing constructors to initialize the member variables.
* Offering public methods to allow controlled access and modification of the matrix data.

This approach helps to maintain the integrity of the data and prevents unintended interference from external code.

**Class: MatrixOperation**

The MatrixOperation class is designed to perform various operations on matrices. Below are the methods typically found in such a class, along with their purposes:

**Methods:**

1. **Matrix Addition**:
   * **Method Name**: add
   * **Purpose**: Adds two matrices of the same dimensions.
2. **Matrix Subtraction**:
   * **Method Name**: subtract
   * **Purpose**: Subtracts one matrix from another matrix of the same dimensions.
3. **Matrix Multiplication**:
   * **Method Name**: multiply
   * **Purpose**: Multiplies two matrices if the number of columns in the first matrix equals the number of rows in the second matrix.
4. **Matrix Transposition**:
   * **Method Name**: transpose
   * **Purpose**: Transposes a given matrix, i.e., converts rows to columns and vice versa.
5. **Matrix Inversion**:
   * **Method Name**: invert
   * **Purpose**: Finds the inverse of a given square matrix, if it exists.
6. **Matrix Determinant Calculation**:
   * **Method Name**: determinant
   * **Purpose**: Calculates the determinant of a given square matrix.

Harshwardhan Mane

612303112  
Program N0. 4

import *java.util.Scanner*;

*class* Matrix{

*private* int rows, cols;

*private* double [][] matrix;

*public* Matrix(){

                        this.rows = 0;

                        this.cols = 0;

                    }

*public* Matrix(int rows, int cols) {

                        this.rows = rows;

                        this.cols = cols;

                        matrix = new double[rows][cols];

                        System.out.println("enter matrix elements from row vise to column vise");

                        Scanner scanner = new Scanner(System.in);

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

                                for(int j = 0; j < cols; j++){

                                    System.out.println("enter at row: " + i + " col: " + j);

                                    matrix[i][j] = scanner.nextDouble();

                                }

                            }

                    }

*public* Matrix(int rows, int cols, double n) {

                        this.rows = rows;

                        this.cols = cols;

                        matrix = new double[rows][cols];

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

                            for(int j = 0; j < cols; j++){

                                matrix[i][j] = n;

                            }

                        }

                    }

*public* int getRow() {

                        return rows;

                    }

*public* int getCol() {

                        return cols;

                    }

*public*  void Put(int row, int col, double n){

                        if(row >= this.rows || col >= this.cols) return;

                        this.matrix[row][col] = n;

                    }

*public* double Get(int row, int col){

                        if(row >= this.rows || col >= this.cols) return -1;

                        return this.matrix[row][col];

                    }

                }

*class* MatrixOpretion{

*static* void PrintMatrix(Matrix matrix){

        for(int i=0; i<matrix.getRow(); i++){

        for(int j=0; j<matrix.getCol(); j++){

            System.out.print(matrix.Get(i, j) + " ");

            }

            System.out.println();

         }

        }

*static* Matrix AddMatrix(Matrix a, Matrix b){

                     Matrix ans = new Matrix();

                     if(a.getRow() != b.getRow() || a.getCol() != b.getCol()){

                         System.out.println("Cant Perform Opretion");

                         return ans;

                     }

                     ans = new Matrix(a.getRow(), a.getCol(), 0);

                     for(int i=0; i<ans.getRow(); i++){

                         for(int j=0; j<ans.getCol(); j++){

                              ans.Put(i, j, a.Get(i, j) + b.Get(i, j));

                     }

                  }

                  return ans;

             }

*static* Matrix SubMatrix(Matrix a, Matrix b){

                Matrix ans = new Matrix();

                if(a.getRow() != b.getRow() || a.getCol() != b.getCol()){

                    System.out.println("Cant Perform Opretion");

                    return ans;

                }

                ans = new Matrix(a.getRow(), a.getCol(), 0);

                for(int i=0; i<ans.getRow(); i++){

                    for(int j=0; j<ans.getCol(); j++){

                        ans.Put(i, j, a.Get(i, j) - b.Get(i, j));

                }

             }

             return ans;

        }

*static* Matrix MulMatrix(Matrix a, Matrix b){

            Matrix ans = new Matrix();

            if(a.getCol() != b.getRow()){

                System.out.println("Cant Perform Opretion");

                return ans;

            }

            ans = new Matrix(a.getRow(), b.getCol(), 0);

            for (int i = 0; i < ans.getRow(); ++i) {

                for (int j = 0; j < ans.getCol(); ++j) {

                    int sum = 0;

                    for (int k = 0; k < a.getCol(); ++k) {

                        sum += a.Get(i, k) \* b.Get(k, j);

                    }

                    ans.Put(i, j, sum);

                }

            }

            return ans;

        }

*static* Matrix MulMatrix(double n, Matrix b){

            Matrix ans = new Matrix(b.getRow(), b.getCol(), 0);

            for(int i=0; i<ans.getRow(); i++){

                for(int j=0; j<ans.getCol(); j++){

                    ans.Put(i, j, b.Get(i, j) \* n);

                }

            }

            return ans;

        }

*static* Matrix MulMatrix(Matrix a, double n){

            Matrix ans = new Matrix(a.getRow(), a.getCol(), 0);

            for(int i=0; i<ans.getRow(); i++){

                for(int j=0; j<ans.getCol(); j++){

                    ans.Put(i, j, a.Get(i, j) \* n);

                }

            }

            return ans;

        }

*static* Matrix Transpose(Matrix A) {

            Matrix result = new Matrix(A.getCol(), A.getRow(), 1);

            for (int i = 0; i < A.getRow(); i++) {

                for (int j = 0; j < A.getCol(); j++) {

                    result.Put(j, i, A.Get(i, j));

                }

            }

            return result;

        }

*static* double det(Matrix A){

*//Determinant finding algorithm for any square matrix*

            int n = A.getRow();

    if (n != A.getCol()) {

        System.out.println("Determinant is not possible for non-square matrices.");

        return -1;

    }

    if (n == 1) return A.Get(0, 0); *// Base case for 1x1 matrix*

    if (n == 2) return A.Get(0, 0) \* A.Get(1, 1) - A.Get(0, 1) \* A.Get(1, 0); *// Base case for 2x2 matrix*

     double determinant = 0;

            for (int j = 0; j < A.getCol(); j++) {

                Matrix temp = new Matrix(A.getRow() - 1, A.getCol() - 1, 1);

                for (int i = 1; i < A.getRow(); i++) {

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

                        temp.Put(i - 1, k, A.Get(i, k));

                    }

                    for (int k = j + 1; k < A.getCol(); k++) {

                        temp.Put(i - 1, k - 1, A.Get(i, k));

                    }

                }

                determinant += Math.pow(-1, j) \* A.Get(0, j) \* det(temp);

            }

            return determinant;

        }

*static* Matrix Inverse(Matrix A) {

            if (A.getRow()!= A.getCol()) {

                System.out.println("Inverse is not possible for non-square matrices.");

                return null;

            }

            double determinant = det(A);

            if(determinant == 0){

                System.out.println("Matrix is singular, no inverse exists");

                return null;

            }

            Matrix adjugate = new Matrix(A.getRow(), A.getCol(), 1);

            adjugate.Put(0,0,A.Get(1,1));

            adjugate.Put(0,1, -A.Get(1,0));

            adjugate.Put(1,0, -A.Get(0,1));

            adjugate.Put(1,1, A.Get(0,0));

            adjugate = MatrixOpretion.MulMatrix(adjugate, 1/determinant);

            return adjugate;

        }

}

*public* *class* program4{

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

                 Matrix matrix1 = new Matrix(3,3);

                 Matrix matrix2 = new Matrix(3,3);

                 System.out.println("------------Matrix1--------------");

                 MatrixOpretion.PrintMatrix(matrix1);

                 System.out.println("---------------------------------");

                 System.out.println("------------Matrix2--------------");

                 MatrixOpretion.PrintMatrix(matrix2);

                 System.out.println("---------------------------------");

                 MatrixOpretion.PrintMatrix(MatrixOpretion.AddMatrix(matrix1, matrix2));

                 System.out.println("---------------------------------");

                 MatrixOpretion.PrintMatrix(MatrixOpretion.SubMatrix(matrix1, matrix2));

                 System.out.println("---------------------------------");

                 MatrixOpretion.PrintMatrix(MatrixOpretion.MulMatrix(matrix1, matrix2));

                 System.out.println("---------------------------------");

                 MatrixOpretion.PrintMatrix(MatrixOpretion.MulMatrix(2, matrix1));

                 System.out.println("---------------------------------");

                 MatrixOpretion.PrintMatrix(MatrixOpretion.MulMatrix(matrix1, 2));

                 System.out.println("---------------------------------");

                 MatrixOpretion.PrintMatrix(MatrixOpretion.Transpose(matrix1));

                 System.out.println("---------------------------------");

                 System.out.println("Determinant: " + MatrixOpretion.det(matrix1));

                 System.out.println("---------------------------------");

                 Matrix inverse = MatrixOpretion.Inverse(matrix1);

                 if(inverse!= null){

                     MatrixOpretion.PrintMatrix(inverse);

                 }

    }

}

**OUTPUT :**

**enter matrix elements from row vise to column vise**

**enter at row: 0 col: 0**

**1**

**enter at row: 0 col: 1**

**2**

**enter at row: 0 col: 2**

**4**

**enter at row: 1 col: 0**

**6**

**enter at row: 1 col: 1**

**7**

**enter at row: 1 col: 2**

**4**

**enter at row: 2 col: 0**

**2**

**enter at row: 2 col: 1**

**8**

**enter at row: 2 col: 2**

**9**

**enter matrix elements from row vise to column vise**

**enter at row: 0 col: 0**

**5**

**enter at row: 0 col: 1**

**3**

**enter at row: 0 col: 2**

**0**

**enter at row: 1 col: 0**

**5**

**enter at row: 1 col: 1**

**2**

**enter at row: 1 col: 2**

**3**

**enter at row: 2 col: 0**

**4**

**enter at row: 2 col: 1**

**5**

**enter at row: 2 col: 2**

**6**

**------------Matrix1--------------**

**1.0 2.0 4.0**

**6.0 7.0 4.0**

**2.0 8.0 9.0**

**---------------------------------**

**------------Matrix2--------------**

**5.0 3.0 0.0**

**5.0 2.0 3.0**

**4.0 5.0 6.0**

**---------------------------------**

**6.0 5.0 4.0**

**11.0 9.0 7.0**

**6.0 13.0 15.0**

**---------------------------------**

**-4.0 -1.0 4.0**

**1.0 5.0 1.0**

**-2.0 3.0 3.0**

**---------------------------------**

**31.0 27.0 30.0**

**81.0 52.0 45.0**

**86.0 67.0 78.0**

**---------------------------------**

**2.0 4.0 8.0**

**12.0 14.0 8.0**

**4.0 16.0 18.0**

**---------------------------------**

**2.0 4.0 8.0**

**12.0 14.0 8.0**

**4.0 16.0 18.0**

**---------------------------------**

**1.0 6.0 2.0**

**2.0 7.0 8.0**

**4.0 4.0 9.0**

**---------------------------------**

**Determinant: 75.0**

**---------------------------------**

**0.09333333333333334 -0.08 0.013333333333333334**

**-0.02666666666666667 0.013333333333333334 0.013333333333333334**

**0.013333333333333334 0.013333333333333334 0.013333333333333334**