

## **OOPS Assignment – JAVA Tutorials2**

### Assignment-1 (OOPs (CS2003)) Unit-1

**Q-1] Explain the concept of Object-Oriented Programming (OOP) and its benefits. What is inheritance? Provide a code example in Java**

#### **What is Object-Oriented Programming (OOP)?**

- Object-Oriented Programming (OOP) is a programming standard that revolves around the concept of objects and classes.
- It's a way of designing and organizing code that simulates real-world objects and systems.
- OOP provides a framework for creating reusable, modular, and maintainable software systems.
- So in a nutshell, OOP is a way of writing code that helps us organize and structure our programs in a more logical and efficient way.

#### **Key Concepts:-**

##### **1. Classes and Objects:**

- A class is a blueprint or template that defines the properties and behaviors of an object.

- An object is an instance of a class, with its own set of attributes (data) and methods (functions).

## **2. Inheritance: (what is inheritance? – asked in quest.)**

- Inheritance is the mechanism by which one class can inherit the properties and behaviors of another class.
- The child class inherits all the fields and methods of the parent class and can also add new fields and methods or override the ones inherited from the parent class.

## **3. Polymorphism:**

- Polymorphism is the ability of an object to take on multiple forms.
- This can be achieved through method overriding (where a subclass provides a different implementation of a method already defined in its superclass) or method overloading (where multiple methods with the same name can be defined, but with different parameters).

## **4. Encapsulation:**

- Encapsulation is the concept of bundling data and methods that operate on that data within a single unit (the class).
- This helps to hide the implementation details of an object from the outside world and provides a way to control access to the object's state.

## **5. Abstraction:**

- Abstraction is the process of exposing only the necessary information to the outside world while hiding the implementation details.
- Abstraction helps to reduce complexity and improve modularity by providing a simplified interface to an object's functionality.

## **Benefits of OOP:-**

### **1. Modularity:**

OOP promotes modular code, making it easier to develop, test, and maintain large software systems.

### **2. Reusability:**

OOP enables code reuse through inheritance and polymorphism, reducing the amount of code that needs to be written and maintained.

### **3. Easier Maintenance:**

OOP's modular and reusable nature makes it easier to modify and extend existing code without affecting other parts of the system.

### **4. Improved Readability:**

OOP's focus on objects and classes makes it easier to understand and visualize the relationships between different components of a system.

### **Q) What is Inheritance?**

A child class can inherit properties and behaviors from a parent class.

### **Code Example in Java:-**

```
Welcome    J Factorial.java 2    J Calculator.java 2    J sample.java 1 ●
C: > Users > Admin > Desktop > J sample.java > ...
1  // Animal class (parent)
2  public class Animal {
3      void sound() {
4          System.out.println(x:"The animal makes a sound.");
5      }
6  }
7
8  // Dog class (child) that inherits from Animal
9  public class Dog extends Animal {
10     void sound() {
11         System.out.println(x:"The dog barks.");
12     }
13 }
14
15 public class Main {
16     Run | Debug
17     public static void main(String[] args) {
18         Dog myDog = new Dog();
19         myDog.sound(); // Output: The dog barks.
20     }
21 }
```

In this example, the **Dog** class inherits the **sound()** method from the **Animal** class and provides its own implementation. This demonstrates the concept of inheritance in OOP.

## Hello world in java:- (Another Sample Java program)

Main.java	Output
<pre>1 class HelloWorld { 2     public static void main(String[] args) { 3         System.out.println("hello world!"); 4     } 5 }</pre>	<pre>java -cp /tmp/ZarcgwXeYs/HelloWorld hello world!  === Code Execution Successful ===</pre>

## Looping statements – An example:-

Main.java	Output
<pre>1 public class Main { 2     public static void main(String[] args) { 3 4         System.out.printf("hello world!"); 5 6         for (int i = 1; i &lt;= 5; i++) { 7 8             System.out.println("i = " + i); 9         } 10    } 11 }</pre>	<pre>java -cp /tmp/eb0uyGJOy/Main hello world! i = 1 i = 2 i = 3 i = 4 i = 5  === Code Execution Successful ===</pre>

**Q-2] Write a Java program to calculate the factorial of a number.**

### AIM:-

To write a program in Java to calculate factorial of a number

### Algorithm: Factorial

Input: n, a positive integer using Scanner, a class in java.util package used for obtaining the input of the primitive types like int, double, etc. and strings

Output: factorial, the factorial of n

Steps:

- Initialize factorial to 1.
- For each integer i from 1 to n:
- Multiply factorial by i.
- Return factorial.

**Program:-**

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

```
public class Factorial {
```

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

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

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

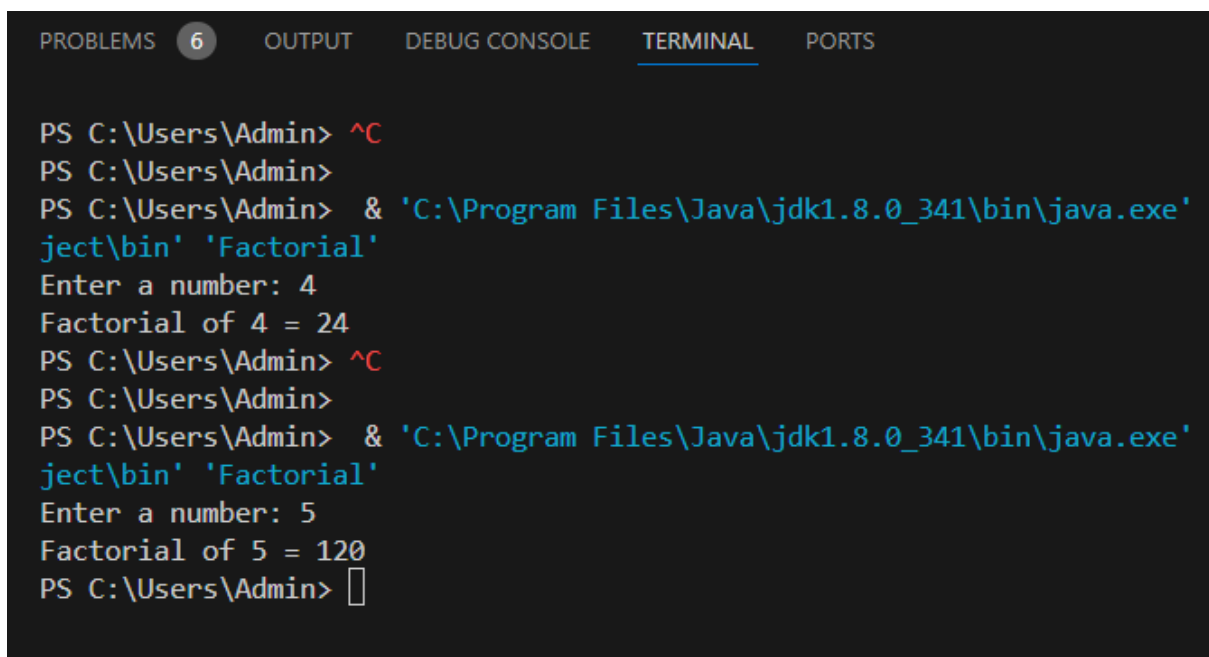
```
        int num = scanner.nextInt();
```

```
        long factorial = 1;
```

```
        for(int i = 1; i <= num; ++i)
```

```
{  
  
    factorial *= i;  
  
}  
  
    System.out.printf("Factorial of %d = %d", num, factorial);  
  
}  
  
}
```

### Output:-



The screenshot shows an IDE terminal window with the following content:

```
PROBLEMS 6 OUTPUT DEBUG CONSOLE TERMINAL PORTS  
  
PS C:\Users\Admin> ^C  
PS C:\Users\Admin>  
PS C:\Users\Admin> & 'C:\Program Files\Java\jdk1.8.0_341\bin\java.exe'  
ject\bin' 'Factorial'  
Enter a number: 4  
Factorial of 4 = 24  
PS C:\Users\Admin> ^C  
PS C:\Users\Admin>  
PS C:\Users\Admin> & 'C:\Program Files\Java\jdk1.8.0_341\bin\java.exe'  
ject\bin' 'Factorial'  
Enter a number: 5  
Factorial of 5 = 120  
PS C:\Users\Admin> 
```

### Result:-



Thus a Java program to find factorial of a number is executed and the output is verified successfully.

**Q-3] Write a Java program to implement a simple calculator.**

**Aim:-**

To write a Java program to implement a simple calculator

**Algorithm:-**

- Ask user to choose an operation (addition, subtraction, multiplication, or division)
- Get two numbers from the user
- Perform the chosen operation on the two numbers
- Use a while loop or implement a switch case condition to demonstrate the arithmetic operation and calculation in a menu driven interface
- Display the result
- Handle errors (e.g. division by zero)

### Pseudocode:

```
INPUT operation
INPUT num1
INPUT num2

IF operation = 1 THEN
    result = num1 + num2
ELSE IF operation = 2 THEN
    result = num1 - num2
ELSE IF operation = 3 THEN
    result = num1 * num2
ELSE IF operation = 4 THEN
    IF num2 ≠ 0 THEN
        result = num1 / num2
    ELSE
        PRINT "Error: Division by zero!"
        EXIT
ELSE
    PRINT "Invalid choice!"
    EXIT

PRINT "Result: " + result
```

### Program:-

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

```
public class Calculator {
```

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

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

```
        System.out.println("Simple Calculator");
```

```
        System.out.println("1. Addition");
```

```
        System.out.println("2. Subtraction");
```

```
        System.out.println("3. Multiplication");
```

```
        System.out.println("4. Division");
```

```
        System.out.println("Choose an operation (1-4): ");
```

```
        int choice = scanner.nextInt();
```

```
        System.out.print("Enter first number: ");
```

```
        double num1 = scanner.nextDouble();
```

```
System.out.print("Enter second number: ");
```

```
double num2 = scanner.nextDouble();
```

```
double result = 0;
```

```
switch (choice) {
```

```
    case 1:
```

```
        result = num1 + num2;
```

```
        break;
```

```
    case 2:
```

```
        result = num1 - num2;
```

```
        break;
```

```
    case 3:
```

```
        result = num1 * num2;
```

```
        break;
```

```
    case 4:
```

```
    if (num2 != 0) {  
        result = num1 / num2;  
    } else {  
        System.out.println("Error: Division by zero!");  
        return;  
    }  
    break;  
default:  
    System.out.println("Invalid choice!");  
    return;  
}  
  
System.out.println("Result: " + result);  
}  
}
```

## Output:-

```
ject\bin' 'Calculator'
Simple Calculator
1. Addition
2. Subtraction
3. Multiplication
4. Division
Choose an operation (1-4):
1
Enter first number: 1
Enter second number: 2
Result: 3.0
PS C:\Users\Admin> ^C
PS C:\Users\Admin>
PS C:\Users\Admin> & 'C:\Program Files\Java\jdk1.8.0_341\bin\
ject\bin' 'Calculator'
Simple Calculator
1. Addition
2. Subtraction
3. Multiplication
4. Division
Choose an operation (1-4):
4
Enter first number: 1
Enter second number: 0
Error: Division by zero!
PS C:\Users\Admin> 
```

## **Result:-**

Thus a Java Program to implement a Simple Calculator is executed and the output is verified successfully.

**Q-4] Write a Java program to perform the multiplication of two matrices.**

## **Aim:-**

**To write a Java program to perform the multiplication of two matrices**

**Sample example “JUST FOR LEARNING PURPOSES”**

**(case-1 : the inputs are passed as parameters)**

```
public class MatrixMultiplication {  
  
    public static void main(String[] args) {  
  
        int[][] matrixA = {{1, 2, 3}, {4, 5, 6}};  
  
        int[][] matrixB = {{7, 8}, {9, 10}, {11, 12}};  
  
  
        int[][] result = multiplyMatrices(matrixA, matrixB);  
  
  
        System.out.println("Matrix A:");
```

```
printMatrix(matrixA);

System.out.println("Matrix B:");

printMatrix(matrixB);

System.out.println("Result:");

printMatrix(result);

}
```

```
public static int[][] multiplyMatrices(int[][] matrixA, int[][]
matrixB) {

    int rowsA = matrixA.length;

    int colsA = matrixA[0].length;

    int rowsB = matrixB.length;

    int colsB = matrixB[0].length;

    if (colsA!= rowsB) {

        System.out.println("Matrices cannot be multiplied!");

        return null;

    }

}
```



```
}
```

```
int[][] result = new int[rowsA][colsB];
```

```
for (int i = 0; i < rowsA; i++) {
```

```
    for (int j = 0; j < colsB; j++) {
```

```
        for (int k = 0; k < colsA; k++) {
```

```
            result[i][j] += matrixA[i][k] * matrixB[k][j];
```

```
        }
```

```
    }
```

```
}
```

```
return result;
```

```
}
```

```
public static void printMatrix(int[][] matrix) {
```

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

```
        for (int j = 0; j < matrix[0].length; j++) {  
            System.out.print(matrix[i][j] + " ");  
        }  
        System.out.println();  
    }  
}  
}
```

**case-1 output:-**

```
PROBLEMS 7 OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\Admin> & 'C:\Program Files\Java\jdk1.8.0_3
xMultiplication'
Matrix A:
1 2 3
4 5 6
Matrix A:
1 2 3
4 5 6
4 5 6
Matrix B:
7 8
7 8
9 10
9 10
11 12
Result:
58 64
139 154
PS C:\Users\Admin>
```

## Case-2: Taking user-defined inputs

### Aim:-

To write a Java program to perform the multiplication of two matrices.

### Algorithm: Matrix Multiplication:-

#### Get Matrix A:

- Ask user for number of rows and columns

- Ask user to input each element of Matrix A

### **Get Matrix B:**

- Ask user for number of rows and columns
- Ask user to input each element of Matrix B

### **Check if matrices can be multiplied: (put a condition) $m \times n \times n \times q$**

If number of columns in Matrix A is not equal to number of rows in Matrix B, stop

### **Multiply Matrices:**

- Create a result matrix with same number of rows as Matrix A and same number of columns as Matrix B
- For each element in result matrix, multiply corresponding elements of Matrix A and Matrix B and add them up

**Print Results** (Display): Print Matrix A, Matrix B, and result matrix

### **Program:-**

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

```
public class MatrixMultiplication {
```

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

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

```
System.out.println("Enter the number of rows for Matrix A:");

int rowsA = scanner.nextInt();

System.out.println("Enter the number of columns for Matrix A:");

int colsA = scanner.nextInt();


int[][] matrixA = new int[rowsA][colsA];

System.out.println("Enter the elements of Matrix A:");

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

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

        System.out.print("Enter element [" + i + "][" + j + "]: ");

        matrixA[i][j] = scanner.nextInt();

    }

}
```

```
System.out.println("Enter the number of rows for Matrix B:");

int rowsB = scanner.nextInt();

System.out.println("Enter the number of columns for Matrix B:");

int colsB = scanner.nextInt();
```

```
int[][] matrixB = new int[rowsB][colsB];

System.out.println("Enter the elements of Matrix B:");

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

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

        System.out.print("Enter element [" + i + "][" + j + "]: ");

        matrixB[i][j] = scanner.nextInt();

    }

}
```

```
int[][] result = multiplyMatrices(matrixA, matrixB);
```

```
System.out.println("Matrix A:");
```

```
printMatrix(matrixA);
```

```
System.out.println("Matrix B:");
```

```
printMatrix(matrixB);
```

```
System.out.println("Result:");
```

```
printMatrix(result);
```

```
}
```

```
public static int[][] multiplyMatrices(int[][] matrixA, int[][] matrixB) {  
  
    int rowsA = matrixA.length;  
  
    int colsA = matrixA[0].length;  
  
    int rowsB = matrixB.length;  
  
    int colsB = matrixB[0].length;  
  
    if (colsA != rowsB) {  
  
        System.out.println("Matrices cannot be multiplied!");  
  
        return null;  
    }  
  
    int[][] result = new int[rowsA][colsB];  
  
    for (int i = 0; i < rowsA; i++) {  
  
        for (int j = 0; j < colsB; j++) {  
  
            for (int k = 0; k < colsA; k++) {  
  
                result[i][j] += matrixA[i][k] * matrixB[k][j];  
  
            }  
        }  
    }  
}
```

```
}
```

```
}
```

```
return result;
```

```
}
```

```
public static void printMatrix(int[][] matrix) {
```

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

```
        for (int j = 0; j < matrix[0].length; j++) {
```

```
            System.out.print(matrix[i][j] + " ");
```

```
        }
```

```
        System.out.println();
```

```
    }
```

```
}
```

```
}
```



**Output:-**

## Output

```
java -cp /tmp/lVU1MbugId/MatrixMultiplication
Enter the number of rows for Matrix A:
2
Enter the number of columns for Matrix A:
2
Enter the elements of Matrix A:
Enter element [0][0]: 1
Enter element [0][1]: 2
Enter element [1][0]: 3
Enter element [1][1]: 4
Enter the number of rows for Matrix B:
2
Enter the number of columns for Matrix B:
2
Enter the elements of Matrix B:
Enter element [0][0]: 5
Enter element [0][1]: 6
Enter element [1][0]: 7
Enter element [1][1]: 8
Matrix A:
1 2
3 4
Matrix B:
5 6
7 8
Result:
19 22
43 50

=== Code Execution Successful ===
```

**Result:-** Thus a Java program to implement multiplication of 2 matrices is executed and the output is verified successfully.

**Q-5] Write a Java method to compute the determinant of an  $N \times N$  matrix using recursion.**

**AIM:-**

**To write a Program in Java to compute the determinant of a  $N \times N$  matrix using recursion.**

**Algorithm:-**

1. Get Matrix Size: Ask user for the size of the matrix (N).
2. Get Matrix Elements: Ask user to input elements of the matrix.
3. Calculate Determinant:
  - If matrix size is 1, return the single element.
  - If matrix size is 2, calculate determinant using the formula:  $a*b - c*d$ .
  - For larger matrices:
    - For each element in the first row:
      - Create a smaller sub-matrix by removing the current row and column.

- Calculate the determinant of the sub-matrix recursively.
- Multiply the result by the current element and a sign factor  $(-1)^i$ .
- Add up all the results to get the final determinant.

4. Print Result: Display the calculated determinant to the user.

**Program :-**

```
import java.util.Scanner;

public class MatrixDeterminant {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        System.out.println("Enter the dimension of the matrix (N): ");

        int n = scanner.nextInt();

        int[][] matrix = new int[n][n];

        System.out.println("Enter the elements of the matrix: ");

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

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

                matrix[i][j] = scanner.nextInt();
```

```
}
```

```
}
```

```
int determinant = findDeterminant(matrix, n);
```

```
System.out.println("The determinant of the matrix is: " +  
determinant);
```

```
}
```

```
public static int findDeterminant(int[][] matrix, int n) {
```

```
if (n == 1) {
```

```
return matrix[0][0];
```

```
}
```

```
if (n == 2) {
```

```
return matrix[0][0] * matrix[1][1] - matrix[0][1] * matrix[1][0];
```

```
}
```

```
int determinant = 0;
```

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

```
int[][] subMatrix = new int[n - 1][n - 1];
```

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

```
for (int k = 0, col = 0; k < n; k++) {  
  
    if (k != i) {  
  
        subMatrix[j - 1][col++] = matrix[j][k];  
  
    }  
  
    }  
  
    }  
  
    determinant += Math.pow(-1, i) * matrix[0][i] *  
    findDeterminant(subMatrix, n - 1);  
  
    }  
  
    return determinant;  
  
    }  
  
    }
```

**Output:-**

## Output

```
java -cp /tmp/97Xkuc0JWJ/MatrixDeterminant
Enter the dimension of the matrix (N):
2
Enter the elements of the matrix:
4
5
3
2
The determinant of the matrix is: -7

=== Code Execution Successful ===
```

### Result:-

Thus a Java program to compute the determinant of a  $N \times N$  matrix using recursion is executed and the output is verified successfully.

**Q-6] Write a Java program to solve a system of linear equations using Cramer's rule.**

**Aim:-**

To Write a program in Java to solve a system of linear equations using Cramer's rule.

**Algorithm:-**

Cramer's Rule Algorithm

1. Get Number of Variables: Ask user for the number of variables (n).
2. Get Coefficients and Constants: Ask user to input coefficients of the matrix and constants.
3. Calculate Determinant: Calculate the determinant of the coefficient matrix using recursion.
4. Check for Unique Solution: If the determinant is 0, print "The system has no unique solution." and exit.
5. Calculate Solutions: For each variable, create a modified matrix by replacing the coefficients of the current variable with the constants, and calculate its determinant.



6. Calculate Solution Values: Calculate the solution value for each variable by dividing the determinant of the modified matrix by the determinant of the coefficient matrix.

7. Print Solutions: Print the solution values for each variable.

Note: This algorithm assumes that the input is a system of linear equations, and Cramer's Rule is used to solve it.

### **Program:-**

```
import java.util.Scanner;

public class CramersRule {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        System.out.println("Enter the number of variables: ");

        int n = scanner.nextInt();

        int[][] matrix = new int[n][n];

        int[] constants = new int[n];

        System.out.println("Enter the coefficients of the matrix: ");

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

            for (int j = 0; j < n; j++) {
```

```
matrix[i][j] = scanner.nextInt();
```

```
}
```

```
}
```

```
System.out.println("Enter the constants: ");
```

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

```
constants[i] = scanner.nextInt();
```

```
}
```

```
int determinant = findDeterminant(matrix, n);
```

```
if (determinant == 0) {
```

```
System.out.println("The system has no unique solution.");
```

```
return;
```

```
}
```

```
int[] solutions = new int[n];
```

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

```
int[][] modifiedMatrix = new int[n][n];
```

```
for (int j = 0; j < n; j++) {
```

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

```
if (k == i) {

    modifiedMatrix[j][k] = constants[j];

} else {

    modifiedMatrix[j][k] = matrix[j][k];

}

}

}

solutions[i] = findDeterminant(modifiedMatrix, n) / determinant;

}

System.out.println("The solutions are: ");

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

    System.out.println("x" + (i + 1) + " = " + solutions[i]);

}

}

public static int findDeterminant(int[][] matrix, int n) {

    if (n == 1) {

        return matrix[0][0];

    }

}
```

```
}
```

```
if (n == 2) {
```

```
    return matrix[0][0] * matrix[1][1] - matrix[0][1] * matrix[1][0];
```

```
}
```

```
int determinant = 0;
```

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

```
    int[][] subMatrix = new int[n - 1][n - 1];
```

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

```
        for (int k = 0, col = 0; k < n; k++) {
```

```
            if (k != i) {
```

```
                subMatrix[j - 1][col++] = matrix[j][k];
```

```
            }
```

```
        }
```

```
    }
```

```
    determinant += Math.pow(-1, i) * matrix[0][i] *
```

```
    findDeterminant(subMatrix, n - 1);
```

```
}
```

```
return determinant;
```

```
}
```

```
}
```

**Output:-**

## Output

```
java -cp /tmp/Qe4wm1njgW/CramersRule
Enter the number of variables:
3
Enter the coefficients of the matrix:
1
2
3
4
5
6
7
8
9
Enter the constants:
1
2
3
The system has no unique solution.

=== Code Execution Successful ===
```

## Result:-

Thus a Java program to solve a system of linear equations using Cramer's rule is executed and the output is verified successfully.

**Q-7] Write a Java program to find the eigenvalues of a 2x2 matrix and verify the determinant as the product of eigenvalues.**

**Aim:-**

To Write a Java program to find the eigenvalues of a 2x2 matrix and verify the determinant as the product of eigenvalues.

**Algorithm:-**

### **Eigenvalues Algorithm**

1. **Get Matrix Elements:** Ask user to input elements of a 2x2 matrix.
2. **Calculate Coefficients:** Calculate coefficients **a**, **b**, and **c** for the quadratic equation.
3. **Check for Complex Eigenvalues:** If the discriminant ( $b^2 - 4ac$ ) is negative, print "The matrix has complex eigenvalues." and exit.
4. **Calculate Eigenvalues:** Calculate two eigenvalues using the quadratic formula.
5. **Calculate Determinant:** Calculate the determinant of the matrix.

6. **Verify Determinant:** Verify that the determinant is equal to the product of the eigenvalues.

7. **Print Results:** Print the eigenvalues, determinant, and verification result.

Note: This algorithm assumes that the input is a 2x2 matrix, and it calculates the eigenvalues and determinant using the quadratic formula and matrix operations.

**Program:-**

```
import java.util.Scanner;

public class Eigenvalues {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        double[][] matrix = new double[2][2];

        System.out.println("Enter the elements of the 2x2 matrix: ");

        for (int i = 0; i < 2; i++) {

            for (int j = 0; j < 2; j++) {

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



```
}  
  
}
```

```
double a = 1;  
  
double b = -(matrix[0][0] + matrix[1][1]);  
  
double c = matrix[0][0] * matrix[1][1] - matrix[0][1] *  
matrix[1][0];
```

```
double discriminant = b * b - 4 * a * c;  
  
if (discriminant < 0) {  
  
    System.out.println("The matrix has complex  
eigenvalues.");  
  
    return;  
  
}
```

```
double eigenvalue1 = (-b + Math.sqrt(discriminant)) / (2 * a);  
  
double eigenvalue2 = (-b - Math.sqrt(discriminant)) / (2 * a);
```

```
double determinant = matrix[0][0] * matrix[1][1] - matrix[0][1]
* matrix[1][0];
```

```
double productOfEigenvalues = eigenvalue1 * eigenvalue2;
```

```
System.out.println("Eigenvalue 1: " + eigenvalue1);
```

```
System.out.println("Eigenvalue 2: " + eigenvalue2);
```

```
System.out.println("Determinant: " + determinant);
```

```
System.out.println("Product of Eigenvalues: " +
productOfEigenvalues);
```

```
if (Math.abs(determinant - productOfEigenvalues) < 1e-9) {
```

```
    System.out.println("The determinant is verified to be the
product of the eigenvalues.");
```

```
    } else {
```

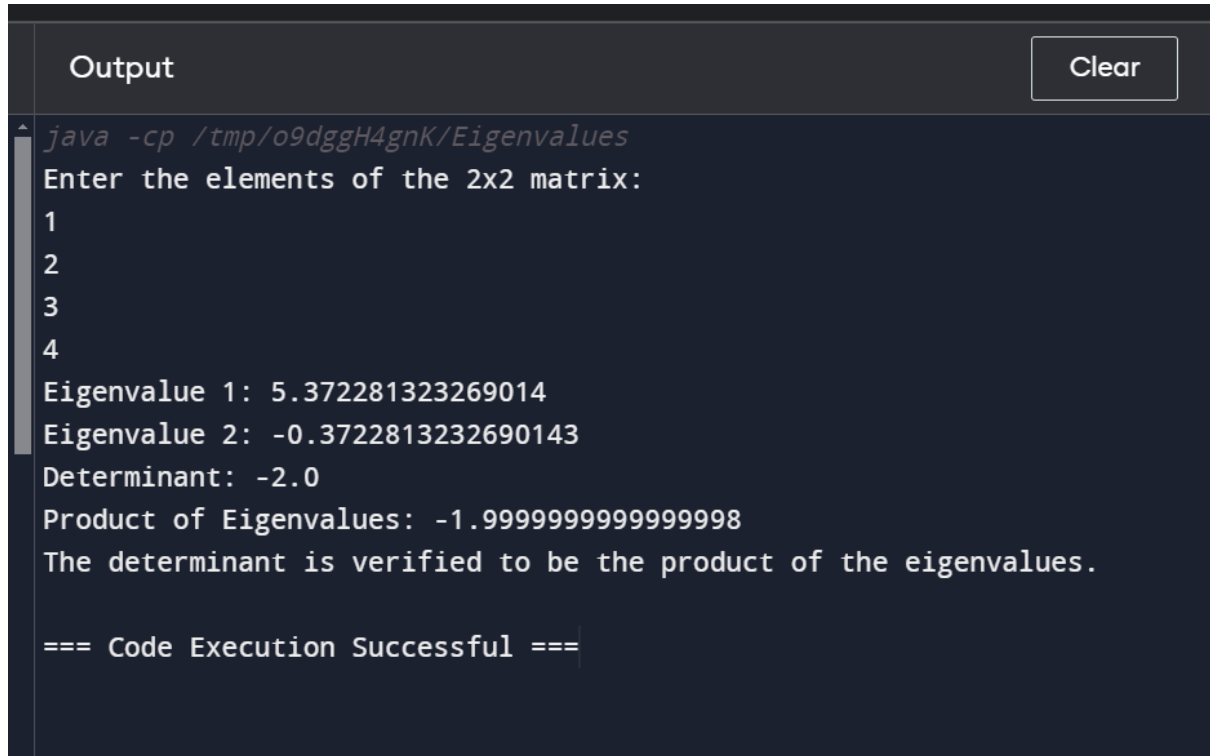
```
        System.out.println("The determinant does not match the
product of the eigenvalues.");
```

```
    }
```

```
}
```

```
}
```

## Output:-



```
Output Clear  
^ java -cp /tmp/o9dggH4gnK/Eigenvalues  
Enter the elements of the 2x2 matrix:  
1  
2  
3  
4  
Eigenvalue 1: 5.372281323269014  
Eigenvalue 2: -0.3722813232690143  
Determinant: -2.0  
Product of Eigenvalues: -1.9999999999999998  
The determinant is verified to be the product of the eigenvalues.  
  
=== Code Execution Successful ===
```

## Result:-

Thus a Java program to find the eigenvalues of a 2x2 matrix and verify the determinant as the product of eigenvalues is executed and the output is verified successfully.

## **Note:-**

- All the program codes and outputs of the assignment questions are pushed into the following repository of mine for reference, which I have given below.

<https://github.com/jayadithya-g7/OOPS-Assignment-Programs-UNIT-1-/tree/main>