Assignment - 1

CS0557 - Cryptography Laboratory

**Name:** Vikas Pundir

**Roll No:** 25203112

**Section:** MTech CSE(IS) First Year

**Date:** 26 August 2025

***Question 1: Write a program to print all prime numbers between 1 and 1000.***

public class Prime {

public static void main(String[] args) { for (int i = 2; i <= 1000; i++) {

boolean isPrime = true;

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

isPrime = false; break;

}

}

if (isPrime) { System.out.print(i + " ");

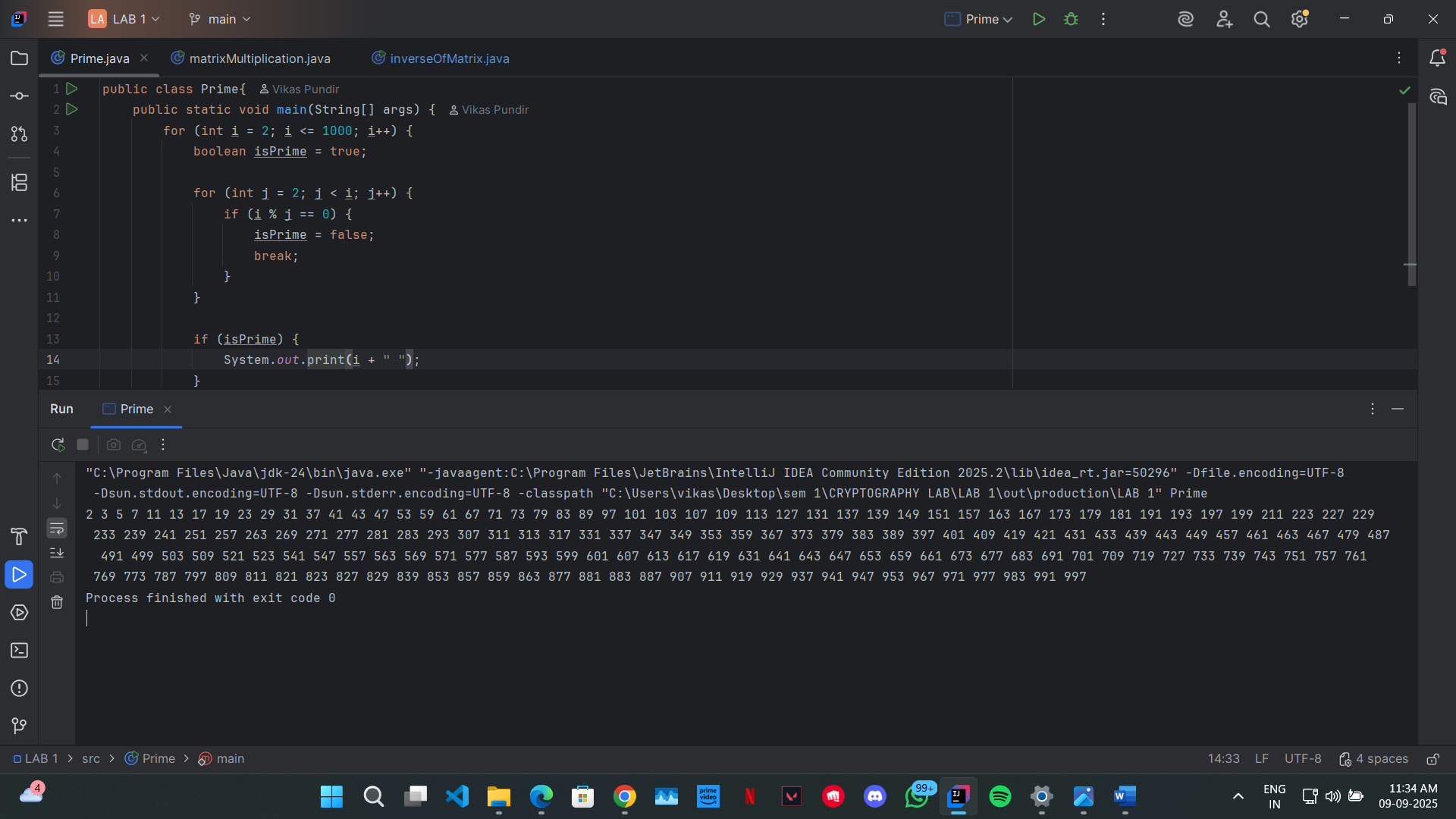
}

}

}

}

**Output :**



***Question 2: Write a program to perform matrix multiplication.***

public class matrixMultiplication {

public static void main(String[] args) { int [][] A = {{1,2}, {3,4}};

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

int rowA = A.length; int colA = A[0].length; int rowB = B.length; int colB = B[0].length;

if(colA != rowB){

System.out.println("PLEASE ENTER 2 VALID MATRIX FOR MULTIPLICATION");

return;

}

int [][] result = new int[rowA][colB]; for (int i = 0; i < rowA; i++) {

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

for (int k = 0; k < colA; k++) { result[i][j] += A[i][k] \* B[k][j];

}

}

}

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

for (int j = 0; j< colB; j++){ System.out.print(result[i][j] + " ");

}

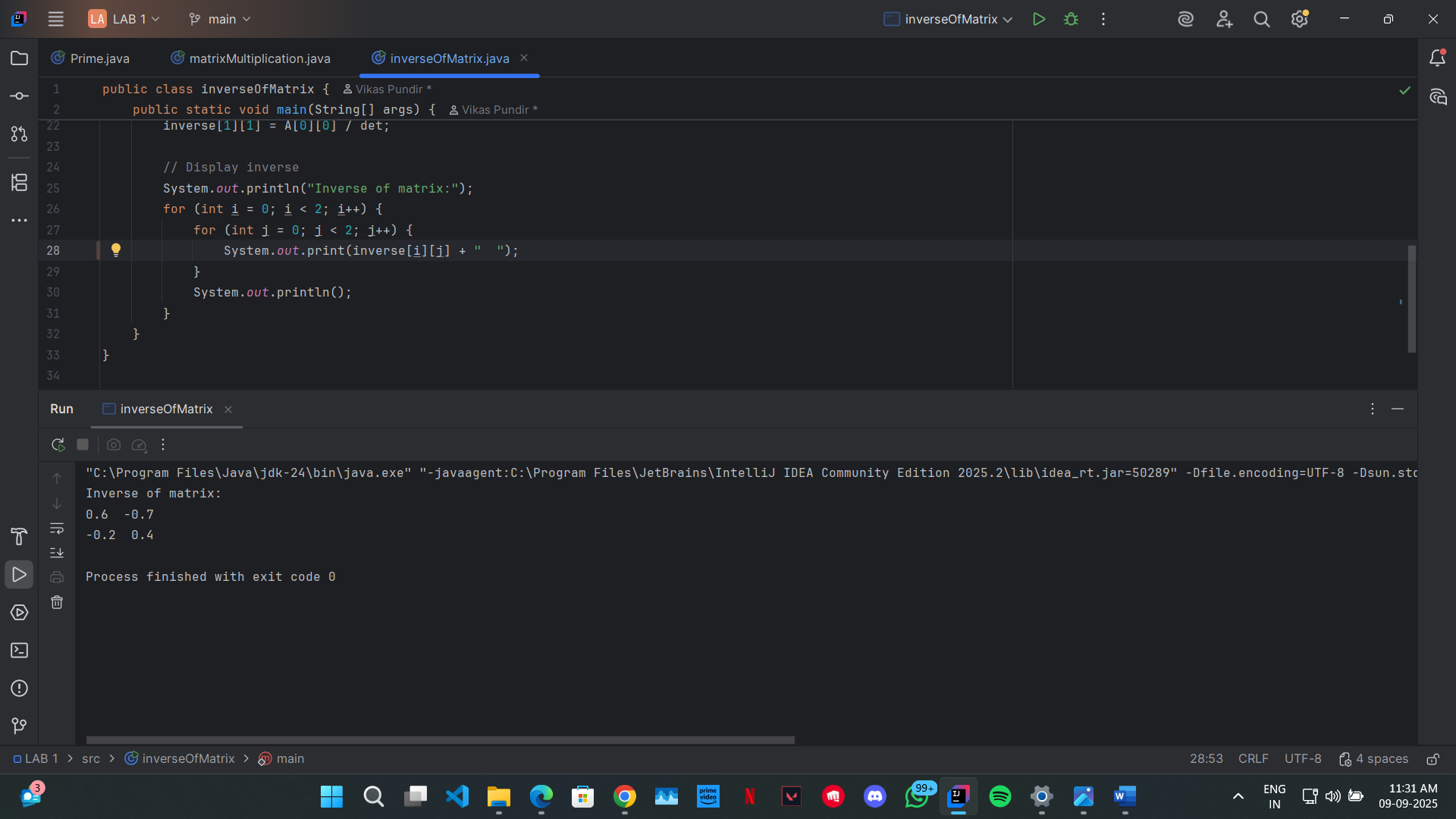
System.out.println();

}

}

}

**Output:**



**Question 3: Write a program to find the inverse of a 3x3 matrix using adjoint and determinant method.**

public class InverseMatrix3x3 {

public static void main(String[] args) { double[][] A = {

{1, 2, 3},

{0, 1, 4},

{5, 6, 0}

};

double det = determinant(A);

if (det == 0) {

System.out.println("Matrix is singular, inverse does not exist."); return;

}

double[][] adj = adjoint(A); double[][] inverse = new double[3][3];

for (int i = 0; i < 3; i++) {

for (int j = 0; j < 3; j++) { inverse[i][j] = adj[i][j] / det;

}

}

System.out.println("Inverse Matrix:"); for (int i = 0; i < 3; i++) {

for (int j = 0; j < 3; j++) { System.out.printf("%.2f ", inverse[i][j]);

}

System.out.println();

}

}

static double determinant(double A[][]) {

return A[0][0]\*(A[1][1]\*A[2][2] - A[1][2]\*A[2][1])

- A[0][1]\*(A[1][0]\*A[2][2] - A[1][2]\*A[2][0])

+ A[0][2]\*(A[1][0]\*A[2][1] - A[1][1]\*A[2][0]);

}

static double[][] adjoint(double A[][]) { double[][] adj = new double[3][3];

adj[0][0] = (A[1][1]\*A[2][2] - A[1][2]\*A[2][1]);

adj[0][1] = -(A[1][0]\*A[2][2] - A[1][2]\*A[2][0]);

adj[0][2] = (A[1][0]\*A[2][1] - A[1][1]\*A[2][0]);

adj[1][0] = -(A[0][1]\*A[2][2] - A[0][2]\*A[2][1]);

adj[1][1] = (A[0][0]\*A[2][2] - A[0][2]\*A[2][0]);

adj[1][2] = -(A[0][0]\*A[2][1] - A[0][1]\*A[2][0]);

adj[2][0] = (A[0][1]\*A[1][2] - A[0][2]\*A[1][1]);

adj[2][1] = -(A[0][0]\*A[1][2] - A[0][2]\*A[1][0]);

adj[2][2] = (A[0][0]\*A[1][1] - A[0][1]\*A[1][0]);

// transpose of cofactor matrix

for (int i = 0; i < 3; i++) {

for (int j = i+1; j < 3; j++) { double temp = adj[i][j]; adj[i][j] = adj[j][i]; adj[j][i] = temp;

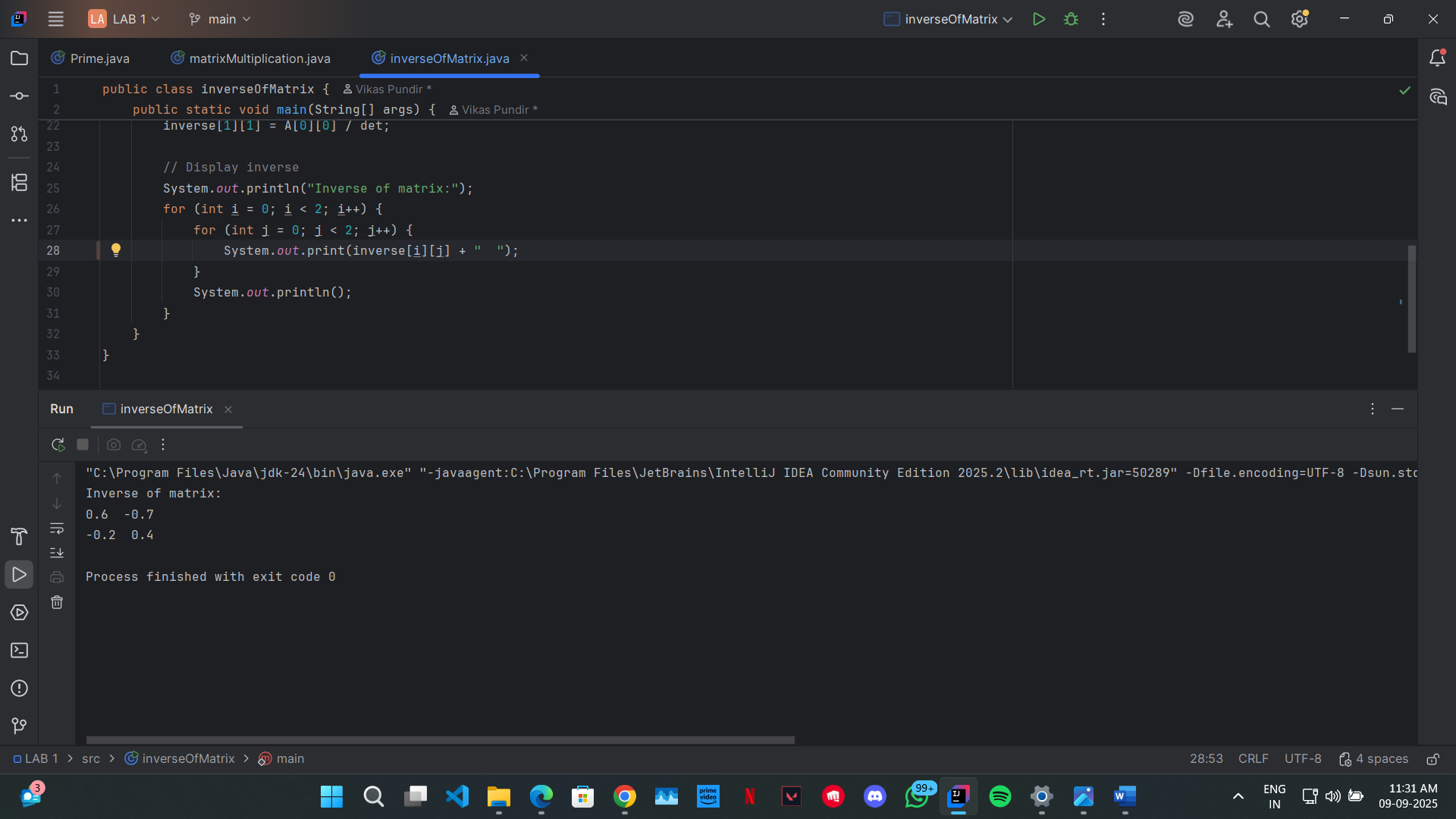
}

}

return adj;

}

**Output:**



\*\*