```java
package POO.Labo5;

public class Main {
    public static void main(String[] args) {
        int m1Lines = 0;
        int m1Columns = 0;
        int m2Lines = 0;
        int m2Columns = 0;
        int mod = 0;

        try {
            m1Lines = Integer.parseInt(args[0]);
            m1Columns = Integer.parseInt(args[1]);
            m2Lines = Integer.parseInt(args[2]);
            m2Columns = Integer.parseInt(args[3]);
            mod = Integer.parseInt(args[4]);
        }
        catch (RuntimeException e) {
            throw new RuntimeException("Please enter valid numbers in the parameters.\n" + e);
        }

        if(m1Lines > 500 || m1Columns > 500 || m2Lines > 500 || m2Columns > 500)
            throw new RuntimeException("Please enter a matrix smaller than 500x500.");

        Matrix m1 = new Matrix(m1Lines, m1Columns, mod);
        Matrix m2 = new Matrix(m2Lines, m2Columns, mod);

        System.out.println("Modulo is : " + mod + "\n");
        System.out.println("Matrix 1 : \n" + m1);
        System.out.println("Matrix 2 : \n" + m2);

        System.out.println("Addition : \n" + m1.add(m2));
        System.out.println("Substraction : \n" + m1.sub(m2));
        System.out.println("Multiplication : \n" + m1.multiply(m2));

        // Test avec les mêmes matrices que la donnée
        /*
        int[][] tbl = new int[][] {
                {1, 3, 1, 1},
                {3, 2, 4, 2},
                {1, 0, 1, 0},
        };
        int[][] tbl2 = new int[][] {
                {1, 4, 2, 3, 2},
                {0, 1, 0, 4, 2},
                {0, 0, 2, 0, 2}
        };

        Matrix m3 = new Matrix(tbl);
        Matrix m4 = new Matrix(tbl2);
        Matrix res;

        System.out.println("The modulos is : 5");
        System.out.println("Matrix 3 : \n" + m3);
        System.out.println("Matrix 4 : \n" + m4);

        System.out.println("Addition : \n" + m3.add(m4));
        System.out.println("Substraction : \n" + m3.sub(m4));
        System.out.println("Multiplication : \n" + m3.multiply(m4));
         */
    }
}
```

```java
package POO.Labo5;

import java.lang.Math;

public class Matrix {

    private final int NB_LINE, NB_COLUMN, MODULO;
    private final int[][] VALUES;

    /**
     * Builds a matrix from a 2-dimensional table.
     * The modulo is calculated by taking the largest element of the matrix + 1.
     *
     * @param values 2-dimensional array of integers representing the values of the matrix.
     * @throws RuntimeException If the array is uninitialised, empty or contains
     *                          negative elements.
     */
    public Matrix(int[][] values) {
        if (values == null || values.length == 0 || values[0].length == 0)
            throw new RuntimeException("The matrix must be at least 1x1 in size.");

        this.NB_LINE = values.length;
        this.NB_COLUMN = values[0].length;
        this.VALUES = values;

        int moduloMax = 0;
        for (int i = 0; i < NB_LINE; ++i) {
            for (int j = 0; j < NB_COLUMN; ++j) {
                if(this.VALUES[i][j] < 0)
                    throw new RuntimeException("Matrix elements must be equal or greater than 0.");
                moduloMax = Math.max(moduloMax, this.VALUES[i][j]);
            }
        }
        this.MODULO = moduloMax + 1;
    }

    /**
     * Builds a matrix of randomly chosen elements.
     *
     *  @param nbLine Number of lines in the matrix.
     *  @param nbColumn Number of columns in the matrix.
     *  @param modulo The modulo limits the value of the elements of the matrix.
     *  @throws RuntimeException If the number of lines or columns is less than 1,
     *                           or if the modulo is less than 1.
     *  */
    public Matrix(int nbLine, int nbColumn, int modulo) {
        if(nbLine < 1 || nbColumn < 1) throw new RuntimeException("The matrix must be at least 1x1 in size.");
        else if (modulo < 1) throw new RuntimeException("The modulo must be equal or greater than 1.");

        this.NB_LINE = nbLine;
        this.NB_COLUMN = nbColumn;
        this.MODULO = modulo;

        VALUES = new int[NB_LINE][NB_COLUMN];
        for(int i = 0; i < NB_LINE; ++i) {
            for(int j = 0; j < NB_COLUMN; ++j) {
                double randomDouble = Math.random();
                VALUES[i][j] = (int)(randomDouble * Integer.MAX_VALUE) % this.MODULO;
            }
        }
    }

    private Matrix calculateMatrix(Matrix m, Operation op) {
        if(MODULO != m.MODULO) throw new RuntimeException("The two modulos must be equal.");

        Matrix res = new Matrix(Math.max(NB_LINE, m.NB_LINE), Math.max(NB_COLUMN, m.NB_COLUMN), MODULO);
        for(int i = 0; i < res.NB_LINE; ++i) {
            for(int j = 0; j < res.NB_COLUMN; ++j) {
                int ope1 = 0;
                int ope2 = 0;
                if(i < NB_LINE && j < NB_COLUMN) ope1 = VALUES[i][j];
                if (i < m.NB_LINE && j < m.NB_COLUMN) ope2 = m.VALUES[i][j];
                res.VALUES[i][j] = op.operator(ope1, ope2, MODULO);
            }
        }
        return res;
    }


    /**
     * Adds the elements of 2 matrices component by component.
     *
     * @param otherMatrix Second matrix to add.
     * */
    public Matrix add(Matrix otherMatrix) {
        return calculateMatrix(otherMatrix, new Addition());
    }

    /**
     * Subtracts the elements of 2 matrices component by component.
     *
     * @param otherMatrix Second matrix to subtract.
     *  */
    public Matrix sub(Matrix otherMatrix) {
        return calculateMatrix(otherMatrix, new Substraction());
    }

    /**
     * Multiplies  the elements of 2 matrices component by component.
     *
     *  @param otherMatrix Second matrix to multriply.
     *  */
    public Matrix multiply(Matrix otherMatrix) {
        return calculateMatrix(otherMatrix, new Multiplication());
    }

    /**
     * Displays the elements of a matrix
     *  */
    @Override
    public String toString() {
        String res = "";
        for (int i = 0; i < NB_LINE; i++) {
            for (int j = 0; j < NB_COLUMN; j++) {
                res += VALUES[i][j] + " ";
            }
            res += "\n";
        }
        return res;
    }
}
```

```java
package POO.Labo5;

public class Addition implements Operation{

    /**
     * Allows you to add two elements together and apply a modulo.
     *
     *  @param firstOperator First element for the addition
     *  @param secondOperator Second element for the addition
     *  @param modulo Modulo use for the addition
     *  @throws RuntimeException The result exceeds the maximum value of an int
     *  */
    @Override
    public int operator(int firstOperator, int secondOperator, int modulo) {
        try {
            Math.addExact(firstOperator, secondOperator);
        } catch (ArithmeticException e) {
            throw new RuntimeException("The numbers in your matrices are too " +
                        "large to be added together and cause capacity overflow.\n" + e);
        }
        return (firstOperator + secondOperator) % modulo;
    }
}

```

```java
package POO.Labo5;

public interface Operation {
    /**
     * Enables an operation to be performed between two elements and a modulo to be applied.
     *
     * @param firstOperator First element for the operation
     * @param secondOperator Second element for the operation
     * @param modulo Modulo use for the operation
     * */
    public int operator(int firstOperator, int secondOperator, int modulo);
}

```

```java
package POO.Labo5;

public class Substraction implements Operation{

    /**
     * Allows you to subtract two elements together and apply a modulo
.
     *
     *  @param firstOperator First element for the subtraction
     *  @param secondOperator Second element for the subtraction
     *  @param modulo Modulo use for the subtraction
     * */
    @Override
    public int operator(int firstOperator, int secondOperator, int
modulo) {
        int res = (firstOperator - secondOperator);
        res =  Math.floorMod(res,modulo);
        return res;
    }
}

```

```java
package POO.Labo5;

public class Multiplication implements Operation{
    /**
     * Allows you to multiply two elements together and apply a modulo .
     *
     *  @param firstOperator First element for the multiplication
     *  @param secondOperator Second element for the multiplication
     *  @param modulo Modulo use for the multiplication
     *  @throws RuntimeException The result exceeds the maximum value of an int
     *  */
    @Override
    public int operator(int firstOperator, int secondOperator,int modulo) {
        try {
            Math.multiplyExact(firstOperator, secondOperator);
        } catch (ArithmeticException e) {
            throw new RuntimeException("The numbers in your matrices are too large to be " +
                    "multiplied and cause your capacity to be exceeded .\n" + e);
        }
        return (firstOperator * secondOperator) % modulo;
    }
}

```