```
/*-----------------------------------------------------------------------------------
File name       : Matrix.java
Author(s)       : Kevin Farine, Timothée Van Hove
Created         : 3 nov. 2022
Description     : Class who will create matrix with an array of array of int
Remark(s)       :
JDK             : OpenJDK Runtime Environment Temurin-17.0.5+8 (build 17.0.5+8)
-----------------------------------------------------------------------------------*/
package ch.heigvd.poo.labo5.Matrix;

import ch.heigvd.poo.labo5.operations.Operation;
import java.util.Random;
import ch.heigvd.poo.labo5.util.Util;

public class Matrix {

    private final int[][] internalValue;

    private int modulus;

    private static final Random random = new Random();

    /**
     * Constructor who take a 2D array of int and a modulus
     * @param values 2D array of int who is the base of the matrix
     * @param modulus The modulus who will be used for the matrix
     * @throws RuntimeException If modulus, nbRows, nbColumns <= 0, or if a matrix
     * value is >= modulus
     * @throws  NullPointerException if the given int[][], or one of its inner array
     * is null*/
    public Matrix(int[][] values, int modulus) throws RuntimeException {
        checkAndSetModulus(modulus);
        internalValue = new int[values.length][values[0].length];
        for (int i = 0; i < values.length; ++i) {
            if(values[i].length != values[0].length){
                throw new RuntimeException("The given 2d array must have the same " +
                        "number of elements j for each rows i");
            }
            for (int j = 0; j < values[0].length; ++j) {
                if (values[i][j] < 0 || values[i][j] >= this.modulus) {
                    throw new RuntimeException(
                            "The given values must be > 0 and < " + (modulus - 1));
                }
                internalValue[i][j] = values[i][j];
            }
        }
    }

    /**
     * Constructor who take the number of rows, columns and a modulus.
     * @param nbRows The number of rows of the matrix
     * @param nbColumns The number of columns of the matrix
     * @param modulus The modulus who will be used for the matrix
     * @throws RuntimeException if modulus, nbRows, nbColumns <= 0
     */
    public Matrix(int nbRows, int nbColumns, int modulus) throws RuntimeException {
        checkAndSetModulus(modulus);
        if(nbRows <= 0 || nbColumns <= 0){
            throw new RuntimeException("The number of rows / columns must be > 0");
        }
        internalValue = new int[nbRows][nbColumns];
        for (int i = 0; i < nbRows; ++i) {
            for (int j = 0; j < nbColumns; ++j) {
                internalValue[i][j] = random.nextInt(modulus);
            }
        }
    }
```

```
    /**
     * Returns the formatted representation of the matrix with auto-align
     * @return  The formatted representation of the matrix as String
     */
    @Override
    public String toString() {
        StringBuilder result = new StringBuilder();
        int[] maxDigits = new int[internalValue[0].length];
        for (int i = 0; i < internalValue[0].length; ++i) {
            //Find the max value of each column
            int maxValue = 0;
            for (int[] row : internalValue) {
                maxValue = Math.max(maxValue, row[i]);
            }
            //Get the number of digits of the max number of each column
            maxDigits[i] = Util.nbDigits(maxValue);
        }
        for (int[] row : internalValue) {
            for (int j = 0; j < internalValue[0].length; ++j) {
                // Get the number of space characters to add after each value
                int nbSpace = maxDigits[j] + 1 - Util.nbDigits(row[j]);
                result.append(row[j]).append(" ".repeat(nbSpace));
            }
            result.append("\n");
        }
        return result.toString();
    }

    /**
     * Executes the given operation and return a matrix as result
     * @param rhs The other matrix used as right operand
     * @param op The operation used to calculate the new matrix
     * @return The operation result as a new matrix object
     * @throws RuntimeException if the modulus of the 2 matrices are different
     * @throws NullPointerException if the given Matrix or Operation is null
     */
    public Matrix executeOperation(Matrix rhs, Operation op) throws RuntimeException
    {
        if (this.modulus != rhs.modulus){
            throw new RuntimeException("The modulus of the 2 matrices must be " +
                    "identical");
        }
        int maxRows = Math.max(internalValue.length, rhs.internalValue.length);
        int maxColumns = Math.max(internalValue[0].length, rhs.internalValue[0].
        length);
        int[][] result = new int[maxRows][maxColumns];

        for (int i = 0; i < maxRows; ++i) {
            for (int j = 0; j < maxColumns; ++j) {
                int valM1 = checkBounds(i, j), valM2 = rhs.checkBounds(i, j);
                result[i][j] = Math.floorMod(op.execute(valM1, valM2), modulus);
            }
        }
        return new Matrix(result, modulus);
    }

    /**
     * Checks the value of the modulus and set it to the member attribute
     * @param modulus The modulus to check and set
     */
    private void checkAndSetModulus(int modulus){
        if (modulus <= 0)
            throw new RuntimeException("The modulus must be > 0");
        this.modulus = modulus;
    }
```

```
    /**
     * Controls if the indexes are within the bounds of the matrix
     * @param rowIndex Index of the row which is checked
     * @param columnIndex Index of the column which is checked
     * @return internalValue[rowIndex][columnIndex] if the indexes are in the bounds
     * of the matrix, else 0
     */
    private int checkBounds(int rowIndex, int columnIndex) {
        if(rowIndex <= internalValue.length - 1
                && columnIndex <= internalValue[0].length - 1){
            return internalValue[rowIndex][columnIndex];
        }
        return 0;
    }
}
```