```java
/*
 ------------------------------------------------------------------------------------
 Nom du fichier  : Main.java
 Nom du labo     : Laboratoire 5 POO : Matrice
 Auteur(s)       : Baume Oscar, Slimani Walid
 Date creation   : 12.10.2022

 Description     : L'objectif du labo est de définir une classe permettant de représenter des matrices
                   de taille quelconque (M × N) contenant des éléments entre 0 et n - 1 pour un
                   entier n (les entiers sont modulo n). Ces matrices doivent pouvoir
                   être affichée. Il est nécessaire que ces matrices puissent faire des operations ensemble,
                   addition, soustraction et multiplication sont implémenter ici.

 Remarque(s)     :

 Modification(s) : / aucune modification
 ------------------------------------------------------------------------------------
*/
public class Main {
    public static void main(String[] args) {

        final int MODULO = 5;

        Matrice one = new Matrice(new int[][] {{1,3,1,1}, {3,2,4,2}, {1,0,1,0}}, MODULO);
        Matrice two = new Matrice(new int[][] {{1,4,2,3,2}, {0,1,0,4,2}, {0,0,2,0,2}}, MODULO);

        System.out.println("The modulus is " + MODULO);

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

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

        System.out.println("one + two");
        System.out.println(one.add(two));

        System.out.println("one - two");
        System.out.println(one.substract(two));

        System.out.println("one x two");
        System.out.println(one.multiply(two));
    }
}
```

```
/*
 ------------------------------------------------------------------------------------
 Nom du fichier  : Matrice.java
 Nom du labo     : Laboratoire 5 POO : Matrice
 Auteur(s)       : Baume Oscar, Slimani Walid
 Date creation   : 12.10.2022

 Description     : Ce fichier défini la classe Matrice qui modelise une matrice de taille
                   ligne par colonne avec un modulo donnée. Cette classe a l'aide des sous-classes
                   d'Operator
                   nous permet d'addtitionner, soustraire et multiplier des matrices.
                   Il soit possible de créer une matrice soit en générant son contenu aléatoirement
                   (une fois sa taille et son modulo connus),soit en passant ses valeurs en paramètre.
                   • Il soit possible d'afficher le contenu de la matrice.
                   • Il soit possible d'effectuer les opérations suivantes entre deux matrices: l'addition,
                     la soustraction et le produit composante par composante. Toutes les opérations
                     doivent être
                     effectuées modulo n.

 Remarque(s)     : Si l'on effectue une opération entre une matrice A de taille M1 × N1 et
                   une matrice B de taille M2 × N2 et que les tailles ne correspondent pas, le résultat
                   est une matrice de taille max(M1, M2) × max(N1, N2) où les Ai,j et Bi,j manquants ont
                   été remplacés
                   par des 0.
                   • Si les modules n des deux matrices ne correspondent pas, lever une RuntimeException.

 Modification(s) : / aucune modification
 ------------------------------------------------------------------------------------
*/
public class Matrice {
    // region Ctor
    public Matrice(int ligne, int colonne, int modulo) {
        this.ligne   = ligne;
        this.colonne = colonne;
        this.modulo  = modulo;
        generateRandomMatrice(ligne, colonne, modulo);
    }

    public Matrice(int[][] matrice, int modulo) {
        this.modulo = modulo;
        try {
            generateMatriceWhitModel(matrice, modulo);
        }
        catch (RuntimeException e) {
            throw e;
        }
    }
    //endregion

    // region Paramètres privés
    static final int REPLACE_EMPTY_CASE = 0;
    private int ligne;
    private int colonne;
    private int modulo;
    private int[][] matrice;
    // endregion

    // region Méthodes publiques

    /**
     * Nom          : add
     * Description  : Additionne la matrice this à la matrice right
     * @param right : La matrice avec laquelle on va s'additionner
     * @return      : La matrice résultant de l'addition
     */
    public Matrice add(Matrice right) {
        if(right == null){
            throw new RuntimeException("La matrice de droite est null");
        }
        try{
            return applyOperation(this,right, new Addition());
        }
```

```
        catch (RuntimeException e) {
            throw e;
        }
    }

    /**
     * Nom          : substract
     * Description  : Soustrait la matrice right à la matrice this
     * @param right : La matrice qu'on va soustraire à this
     * @return      : La matrice résultant de la soustraction
     */
    public Matrice substract(Matrice right) {
        if(right == null){
            throw new RuntimeException("La matrice de droite est null");
        }
        try{
            return applyOperation(this,right, new Substraction());
        }
        catch (RuntimeException e) {
            throw e;
        }
    }

    /**
     * Nom          : multiply
     * Description  : Multiplie la matrice this avec la matrice right
     * @param right : La matrice avec laquelle on va multiplier this
     * @return      : La matrice résultant de la multiplication
     */
    public Matrice multiply(Matrice right) {
        if(right == null){
            throw new RuntimeException("La matrice de droite est null");
        }
        try{
            return applyOperation(this,right, new Multiplication());
        }
        catch (RuntimeException e) {
            throw e;
        }
    }

    /**
     * Nom          : toString
     * Description  : Fonction retournant la matrice sous forme de string
     * @return      : La matrice sous forme de string
     */
    @Override
    public String toString() {
        String m = "";

        for (int i = 0; i < ligne; ++i) {
            for (int j = 0; j < colonne; ++j) {
                m += matrice[i][j];
            }
            m += "\n";
        }
        return m;
    }
    // endregion

    // region Méthodes privées

    /**
     * Nom              : generateRandomMatrice
     * Description      : génère une matrice de taille ligne x colonne, n'ayant que des entiers inférieur
     à modulo.
     * @param ligne     : Nombre de ligne de la matrice souhaiter
     * @param colonne   : Nombre de colonne de la matrice souhaiter
     * @param modulo    : Modulo de la matrice souhaiter
     */
    private void generateRandomMatrice(int ligne, int colonne, int modulo) {
        if(ligne == 0 || colonne == 0 || modulo == 0){
```

```
            throw new RuntimeException("Les paramètres d'entré ne peuvent pas avoir la valeur de 0");
        }
        matrice = new int[ligne][colonne];
        for (int i = 0; i < ligne; ++i) {
            for (int j = 0; j < colonne; ++j) {
                matrice[i][j] = (int) (Math.random() * modulo);
            }
        }
    }

    /**
     * Nom              : generateMatriceWhitModel
     * Description      : Génère une matrice avec un tableau à 2 dimensions en entré,
     * @param matrice   : Tableau 2d representant la matrice
     * @param modulo    : modulo souhaiter pour la matrice
     * Remarque         : Si un entier de matrice est plus grand ou égal à modulo on throw une
     RunTimeException
     */
    private void generateMatriceWhitModel(int[][] matrice, int modulo) {
        if(matrice == null || modulo == 0){
            throw new RuntimeException("La matrice entrée est null ou le modulo est égal à 0");
        }
        if (matrice.length > 0)
            ligne = matrice.length;

        else throw new RuntimeException("Nombre de ligne égal à 0");

        if (matrice[0].length > 0)
            colonne = matrice[0].length;

        else throw new RuntimeException("Nombre de colonne égal à 0");

        for (int[] l : matrice) {
            for(int i : l){
                if( i < 0) {
                    throw new RuntimeException("Valeur inférieur à 0 ");
                }
                if(i >= modulo ) {
                    throw new RuntimeException("Valeur suppérieur ou égal au modulo");
                }
            }
        }
        this.matrice = matrice;
    }

    /**
     * Nom          : reformatMatriceWithModulo
     * Description  : Cette fonction est utilisé après les operations pour que la matrice respecte la
     condition du modulo
     */
    private void reformatMatriceWithModulo() {
        for(int i = 0; i < ligne; i++) {
            for(int j = 0; j < colonne; j++) {
                matrice[i][j] = Math.floorMod(matrice[i][j], modulo);
            }
        }
    }

    /**
     * Nom              : applyOperation
     * Description      : Cette fonction va faire l'opération souhaiter entre 2 matrice
     * @param left      : Matrice de gauche
     * @param right     : Matrice de droite
     * @param operator  : L'opération souhaiter
     * @return          : La matrice resultant de l'opération
     * Remarque         : Si le modulo des 2 matrices sont différents, alors la fonction throw une
     RunTimeException
     */
    private Matrice applyOperation(Matrice left, Matrice right, Operator operator) {
        if(left == null){
            throw new RuntimeException("La matrice de gauche est null");
        }
```

```
        if(right == null){
            throw new RuntimeException("La matrice de droite est null");
        }
        if(operator == null){
            throw new RuntimeException("L'operateur est null");
        }
        if(right.modulo != left.modulo) {
            throw new RuntimeException("Modulo pas identique entre les 2 éléments de l'operation");
        }

        int outL = Math.max(right.ligne,left.ligne);
        int outC = Math.max(right.colonne,left.colonne);

        Matrice out = new Matrice(outL,outC, left.modulo);
        for(int i = 0; i < outL; ++i) {
            for(int j = 0; j < outC; ++j) {
                // si la case n'existe pas pour la matrice de gauche
                if( i >= left.ligne  || j >= left.colonne) {
                    // on effectue le calcul avec 0 pour valeur de gauche
                    out.matrice[i][j] = operator.apply(REPLACE_EMPTY_CASE, right.matrice[i][j]);
                    // on peut passer à la case suivante
                    continue;
                }
                // si la case n'existe pas pour la matrice de droite
                if(i >= right.ligne  || j >= right.colonne ) {
                    // on effectue le calcul avec 0 pour valeur droite
                    out.matrice[i][j] = operator.apply(left.matrice[i][j],REPLACE_EMPTY_CASE);
                    // on peut passer à la case suivante
                    continue;
                }

                // si les 2 cases existe on fait le calcul normalement.
                out.matrice[i][j] = operator.apply(left.matrice[i][j], right.matrice[i][j]);
            }
        }
        out.reformatMatriceWithModulo();
        return out;
    }
    // endregion
}
```

```java
/*
 ------------------------------------------------------------------------------------
 Nom du fichier  : Operator.java
 Nom du labo     : Laboratoire 5 POO : Matrice
 Auteur(s)       : Baume Oscar, Slimani Walid
 Date creation   : 12.10.2022

 Description     : Ce fichier défini la classe abstraite Operator, qui a pour seul élément la
                   méthode abstraite apply.

 Remarque(s)     : La méthode apply retourne le resultat de l'operation modéliser entre 2 entiers.

 Modification(s) : / aucune modification
 ------------------------------------------------------------------------------------
*/
public abstract class  Operator {
    /**
     * Nom          : apply
     * Description  : Fonction effectuant l'operation
     * @param left  : entier de gauche
     * @param right : entier de droite
     * @return      : Résultat de l'opération
     */
    public abstract int apply(int left, int right);
}
```

```java
/*
 ------------------------------------------------------------------------------------
 Nom du fichier  : Addition.java
 Nom du labo     : Laboratoire 5 POO : Matrice
 Auteur(s)       : Baume Oscar, Slimani Walid
 Date creation   : 12.10.2022

 Description     : Ce fichier défini la classe Addtion héritant de Operator.

 Remarque(s)     : La méthode apply retourne le resultat de l'addition entre 2 entiers.

 Modification(s) : / aucune modification
 ------------------------------------------------------------------------------------
*/
public class Addition extends Operator {
    @Override
    public int apply(int left, int right) {
        return left + right;
    }
}
```

```java
/*
 ------------------------------------------------------------------------------------
 Nom du fichier  : Substraction.java
 Nom du labo     : Laboratoire 5 POO : Matrice
 Auteur(s)       : Baume Oscar, Slimani Walid
 Date creation   : 12.10.2022

 Description     : Ce fichier défini la classe Substraction héritant de Operator.

 Remarque(s)     : La méthode apply retourne le resultat de la soustraction entre 2 entiers.

 Modification(s) : / aucune modification
 ------------------------------------------------------------------------------------
*/
public class Substraction extends Operator {
    @Override
    public int apply(int left, int right) {
        return left - right;
    }
}
```

```java
/*
 ------------------------------------------------------------------------------------
 Nom du fichier  : Multiplcation.java
 Nom du labo     : Laboratoire 5 POO : Matrice
 Auteur(s)       : Baume Oscar, Slimani Walid
 Date creation   : 12.10.2022

 Description     : Ce fichier défini la classe Multiplcation héritant de Operator.

 Remarque(s)     : La méthode apply retourne le resultat de la multiplication entre 2 entiers.

 Modification(s) : / aucune modification
 ------------------------------------------------------------------------------------
*/
public class Multiplication extends Operator {
    @Override
    public int apply(int left, int right) {
        return left * right;
    }
}
```