Matrix Multiplication — Final Report (v3.2) Generated: 2025-11-12 22:55:27Z Executive Summary (included requirements): - Do NOT zero-fill missing values. Use available matching elements for dot-products; continue if surplus values exist. - Terminate/discard the multiplication only when required elements are missing for a cell (strict discard rule). - Simulate Java logic (Integer type). - Include verbatim literal, Java-like list, executed prints ([L###]), my-code result, expected strict result, and notes per test.

Annotated Source Code (excerpt)

```
//Finished and removed single nested loop (advised by chatGPT to reduce time complexity). Also removed if statement identifed previously.
//Tested code but not extensively.
//It was created based on refinement from several attempts (notably obtaining positive changes in Test case 28)
//And feeding it back into earlier iteration
//All Programiz challenge test cases have passed

//ALL FORMATTED INDENTED CORRECTLY
//TERMS Matrix A and Matrix B will be used to discuss the pairing of the first and second matrix in transaction

//ALSO NOTE, in my comments, there is reference to Matrix A, Matrix B, Matrix C
//In absence of a Matrix C (references are to Matrix A and Matrix B)
//In presence of Matrix C (references are to Matrix A, Matrix B, Matrix C, Matrix, B, Matrix C........)

//ANY CATCH STATEMENT WHICH REQUIRES A MORE USER FRIENDLY MESSAGE CAN INCLUDE ONE OF THESE (IF REQUIRED)
/*
System.out.println("2Matrix("+(counter)+")" + "   row:" + ee +"\thas no content at index["+m+"]");    // ANNOTATION: print at L16
System.out.println("Hence can not perform multiplication with Matrix("+(counter+1)+")" + "   row:" + (rowCounterNextMatrix+1)+"\t"     // ANNOTATION: print at L17
+  "  index["+z+"]: " + rowInfo.get(z));



System.out.println("1Matrix("+(counter)+")" + "   row:" + rowCounterFirstMatrix+"\t" +  "value at index["+z+"]: " + ee.get(posPrevMatrixRowLevel)     // ANNOTATION: print
+"  has no available supporting entry in \nMatrix("+(counter+1)+")" + "   row:" + (rowCounterNextMatrix+1)+"\t" +  "\t index["+z+"]: " + " row content:" + rowInfo);
*/



/*
//**********USAGE************************
BEFORE CONFIGURING TEST CASES REMEMBER THESE CRITERIA AS MINIMUM:

Numbers columms in Matrix A/C  = Number rows in Matrix B
Jagged arrays supported under validation rules
Size first row (number columns) in Matrix A/C and number rows in Matrix B will ALWAYS determine matrixMultiplication size
//*********************************
*/

import java.util.*;

public class Solution
{
    static List<List<Integer>> matrix = new ArrayList<>();
    static List<List <Integer>> nextMatrix;

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

```
{
    List <List<List<Integer>>> testMatrix = new ArrayList<>();

    //TEST CASES
    //We have to ensure that the test case is a 3d array as oppose to a 2d array to contain all the matrixes
    //Integer [][][]test = new Integer[][][] {   {{1},{5}},{{9,88}}};

    //Only 1 matrix defined
    //Integer [][][]test = new Integer[][][] {   {{1,2,3},{4,5,6}}};

    //As per challenge  - Test Case 1
    //Integer [][][]test = new Integer[][][] {   {{1,2,3},{4,5,6}},  {{7,8},{9,10},{11,12}},  { {13,14}, {15,16} } };

    //Adaptation  Test Case 1
    //Integer [][][]test = new Integer[][][] {   {{1,2,3,5,6},{4,5,6}},  {{7,8},{9,10},{11,12}},  { {13,14}, {15,16} } };

    //As per challenge  - Test Case 2
    //Integer [][][]test = new Integer[][][] {   {{1,2,3,4}},{{5},{6},{7},{8}}};

    //As per challenge  - Test Case 3/4  (both same)
    //Integer [][][]test = new Integer[][][] {   {{1, 2}, {3, 4}}, {{5, 6}, {7, 8}}, {{9, 10}, {11, 12}}  };

    //TEST CASES EXPLORING JAGGED ARRAYS (variants of TEST CASE 28 IN Document)
    //Integer [][][]test = new Integer[][][] {   {{1,2,3},{4,5,6}},  {{7,8},{10},{11,12}},  { {13,14}, {15,16} } };

    //TEST CASES EXPLORING JAGGED ARRAYS (variants of TEST CASE 28 IN Document)
    //This relies on the matrixMultiplication initialisation
    Integer [][][]test = new Integer[][][] {   {{1,2,3},{4,5,6}},  {{7,8,999},{10,11},{11,12}},  { {13,14}, {15,16} } };

    //TEST CASES EXPLORING JAGGED ARRAYS (variants of TEST CASE 28 IN Document)
    //Integer [][][]test = new Integer[][][] {   {{1,2,3},{4,5}},  {{7,8},{10,11},{11,12}},  { {13,14,17}, {15,16} } };

    //TEST CASES EXPLORING JAGGED ARRAYS (variants of TEST CASE 28 IN Document)
    //Integer [][][]test = new Integer[][][] {   {{1,2},{4,5,6}},  {{7,8},{10,11},{11,12}},  { {13,14,17}, {15,16,17} } };

    //TEST CASES EXPLORING JAGGED ARRAYS (variants of TEST CASE 28 IN Document)
    //This will get picked up on area of code I had written towards bottom part of code
    //Integer [][][]test = new Integer[][][] {   {{},{4,5,6}},  {{7,8,6},{10,14},{11,12}},  { {13}, {15,16} } };

    //HERE!!!!!!!!!!!!!!!!!!!!!! - NEED TO CHECK THIS!!!!
    //Integer [][][]test = new Integer[][][] {   {{1,2},{3,4},{5,6}},  {{7,8},{10,14},{0}},  { {13}, {15,16} } };

    //With this test case it has not performed jump to new column
    //Integer [][][]test = new Integer[][][] {   {{1,2,3},{4,5,6}},  {{7,8},{9,10},{11,12}},  { {0}, {16,17},{5}} };
    //Integer [][][]test = new Integer[][][] {   {{1,2,3},{4,5,6}},  {{7,8},{9,10},{11}},  { {13,14}, {16,17} } };
```

```
//TEST CASES EXPLORING JAGGED ARRAYS (variants of TEST CASE 28 IN Document)
//Integer [][][]test = new Integer[][][] {   {{1,2,3},{4,5,6}},  {{7,8},{9,2},{11,12}},  { {13,14}, {16,87},{3,4} } };

//TEST CASES EXPLORING JAGGED ARRAYS (variants of TEST CASE 28 IN Document)
//Integer [][][]test = new Integer[][][] {   {{1,2,3},{4,5,6}},  {{},{10,14},{11,12}},  { {13}, {15,16} } };

//TEST CASES EXPLORING JAGGED ARRAYS (variants of TEST CASE 28 IN Document)
//Integer [][][]test = new Integer[][][] {   {{1,2,3},{4,5,6}},  {{1,7},{25,14},{}},  { {13,15}, {15,16} } };

//TEST CASES EXPLORING JAGGED ARRAYS (variants of TEST CASE 28 IN Document)
//This will be handled validation in later part of code
//Integer [][][]test = new Integer[][][] {   {{1,2,3},{}},  {{1,7},{25,14},{4,8}},  { {13,15}, {15,16} } };

//TEST CASES EXPLORING JAGGED ARRAYS (variants of TEST CASE 28 IN Document
//Integer [][][]test = new Integer[][][] {   {{1,2,3},{4,5,6}},  {{1,7},{25,14},{4,8}},  { {}, {15,16} } };

//TEST CASES EXPLORING JAGGED ARRAYS (variants of TEST CASE 28 IN Document)
//Integer [][][]test = new Integer[][][] {   {{1,2,3},{4,5,6}},  {{1,7},{25,14},{4,8}},  { {34,45}, {} } };

//TEST CASES EXPLORING JAGGED ARRAYS (TEST CASE 28 IN Document exactly)
//Integer [][][]test = new Integer[][][] {   {{1,2,3},{4,5,6}},  {{7,8,6},{10,14},{11,12}},  { {13,14}, {15,16} } };

//TEST CASES EXPLORING JAGGED ARRAYS (TEST CASE 28 IN Document exactly)
//But it is error free, so informed end user excess numbers
//Integer [][][]test = new Integer[][][] {   {{1,2,3},{4,5,6,9}},  {{7,8},{9,10},{11,12}},  { {13,14}, {15,16} } };

//TEST CASES EXPLORING JAGGED ARRAYS BUT EXPLORING CHANGES IN MATRIX A (TEST CASE 28 IN Document exactly)
//Integer [][][]test = new Integer[][][] {   {{1,2,3},{4}},  {{7,8},{9,10},{11,12}},  { {13,14}, {15,16} } };

//TEST CASES EXPLORING JAGGED ARRAYS BUT EXPLORING CHANGES IN MATRIX A (TEST CASE 28 IN Document exactly)
//But it is error free, so informed end user excess numbers
//Integer [][][]test = new Integer[][][] {   {{1,2,3},{4,7,2,6}},  {{7,8},{9,10},{11,12}},  { {13,14}, {15,16} } };

//TEST CASES EXPLORING JAGGED ARRAYS (TEST CASE 28 IN Document exactly)
//Integer [][][]test = new Integer[][][] {   {{1,2,3},{4,5,6}},  {{},{10,14},{11,12}},  { {13,14}, {15,16} } };

//Integer [][][]test = new Integer[][][] {   {{1,2,3},{4,5,6}},  {{11,14},{},{19,12}},  { {13,14}, {15,16} } };
//Integer [][][]test = new Integer[][][] {   {{1,2,3},{4,5,6}},  {{11,14},{44,55},{19,12}},  { {13,14}, {} } };

//Integer [][][]test = new Integer[][][] {   {{1,2,3},{4,5,6}},  {{11,14},{44,55},{19,12}},  { {13,14}, {0} } };

//Integer [][][]test = new Integer[][][] {   {{1,2,3},{4,5,6}},  {{11,14},{44,55},{19,12}},  { {13,14}, {0,5},{0} } };

//TEST CASES EXPLORING JAGGED ARRAYS (TEST CASE 28 IN Document exactly)
//Integer [][][]test = new Integer[][][] {   {{1,2,3},{4,5,6}},  {{7,8},{10,14},{11,12}},  { {13}, {15,16} } };
```

```
    //ADDITIONAL TEST CASES

    //Integer [][][]test = new Integer[][][] {   {{1,2,3},{4,5,6}},  {{11,14},{44,55},{19,12}},  { {13,14}, {0} } };

    //Integer [][][]test = new Integer[][][] {   {{1,2,3},{4,5,6}},  {{7,8},{10,14},{15,12,65,55}},  { {13,99}, {15,16} } };

    //Integer [][][]test = new Integer[][][] {   {{1,2,3},{4,5,6},{7,8,9}},  {{10,11,12},{13,14,15},{16,17,18}},  { {}, {22,23,24}} };

    //Integer [][][]test = new Integer[][][] {   {{1,2,3},{4,5,6}},  {{7,8},{9,10},{11,12}},  { {13,14}, {15,16} },    { {17,18}, {19,20} },{ {17,18}, {19,20} }};

    //Integer [][][]test = new Integer[][][] {   {{1,2,3},{4,5,6}},  {{7,8},{9,10},{11,12}},  { {13,14}, {15,16}},  { {17,18}, {19,20},{99,18}, {19,20}}, { {21,22,23,2

    //Integer [][][]test = new Integer[][][] {   {{1,2,3},{4,5,6},{7,8,9}},  {{10,11,12},{13,14,15},{16,17,18}},  { {19,20,21}, {22,23,24}} };

    //active
    //Integer [][][]test = new Integer[][][] {   {{1,2,3},{4,5,6},{7,8,9}},  {{11,12},{13,14,15},{16,17,18}} };

    //Integer [][][]test = new Integer[][][] {   {{},{4,5,6},{7,8,9}},  {{10,11,12},{13,14,15},{16,17,18}} };


    for (Integer [][] item: test)    // ANNOTATION: loop over indices
    {
        matrix = new ArrayList<>();

        for (Integer [] row: item)    // ANNOTATION: loop over indices
        {
            matrix.add(Arrays.asList(row));
        }
        testMatrix.add(matrix);
    }

    System.out.println("\n\n***FINAL OUTCOME***: \n" + matrixMultiplication(testMatrix));    // ANNOTATION: print at L168
}

public static List<List<Integer>> matrixMultiplication(List<List<List<Integer>>> matrices)
{
    List<Integer> rowInfo = new ArrayList<>();

    int rowCounterNextMatrix;
    int seekPosition=0;
    int numWritesAtIndexWithinMultiplicationMatrix;

    StringJoiner matrixRowCalculations = new StringJoiner("");
    StringJoiner excessRowItems=new StringJoiner(",");
    String rowMatrix=null;
```

```
String currentEntry=null;
String prevEntry=null;

StringJoiner excessRowReport = new StringJoiner("\n");
StringJoiner excessColReport = new StringJoiner("\n");
StringJoiner [][] matrixMultiplicationCalculations=null;
int numWritesToLastIndexOnRowMultiplicationMatrix;
boolean hasReachedEndRow=false;
boolean hasStartedMultiplication=false;
Integer multiplication=0;

int rowCounterFirstMatrix=0;
int rowsNextMatrix=0;
int matricesSize=matrices.size();

int counter=0;

int numElementsInRow=0;
int posPrevMatrixRowLevel=0;
int numMatrixes=0;

Integer [][]matrixMultiplication = null;

List <List<Integer>> matrixMultiplicationList = new ArrayList<>();

String currentColumnEntry;

StringJoiner excessColItems=new StringJoiner(",");

System.out.println("***ALL MATRIX*******: " + matrices.size());    // ANNOTATION: print at L212

do
{
    matrix=matrices.get(numMatrixes);

    System.out.println("\nMatrix" +"("+numMatrixes+"):");    // ANNOTATION: print at L218

    for (List<Integer> row: matrix)    // ANNOTATION: loop over indices
    {
        System.out.println(row);    // ANNOTATION: print at L222
    }
    numMatrixes++;

}while(numMatrixes<matrices.size());
System.out.println("\n\n\n");    // ANNOTATION: print at L228
```

```
numMatrixes=0;

do
{
    matrix=matrices.get(counter);

    if (counter>0 && hasStartedMultiplication)    // ANNOTATION: condition
    {
        System.out.println("\n***SUMMARY****");    // ANNOTATION: print at L237
        System.out.println("Matrix multiplication: " + "Matrix ("+(counter-2)+") x Matrix ("+(counter-1)+")");    // ANNOTATION: print at L238

        if (matrices.size()!=2)    // ANNOTATION: condition
        {
            System.out.println("ADDED Multiplication Matrix into the chain at index: " + (counter));    // ANNOTATION: print at L242
            matrices.add(counter,matrixMultiplicationList);
            hasStartedMultiplication=false;
            System.out.println("Matrix("+counter+")");    // ANNOTATION: print at L245

            for (Integer []m: matrixMultiplication)    // ANNOTATION: loop over indices
            {
                System.out.println(Arrays.toString(m));    // ANNOTATION: print at L249

                matrixMultiplicationList.add(Arrays.asList(m));
            }

            System.out.println("\n***CALCULATION STEPS*******");    // ANNOTATION: print at L254

            for (int q=0; q<matrixMultiplicationCalculations.length; q++)    // ANNOTATION: loop over indices
            {
                matrixRowCalculations = new StringJoiner("   ");

                for (int r=0; r<matrixMultiplicationCalculations[0].length; r++)    // ANNOTATION: loop over indices
                {
                    matrixRowCalculations.add(" ["+matrixMultiplicationCalculations[q][r]+"]");
                }
                System.out.println(matrixRowCalculations);    // ANNOTATION: print at L264
            }
            System.out.println("\nIt will perform: " + "Matrix("+counter+")   x  Matrix("+(counter+1)+")");    // ANNOTATION: print at L266

            for (List<Integer> w: matrices.get((counter+1)))    // ANNOTATION: loop over indices
            {
                System.out.println("\t\t\t\t"+String.valueOf(w));    // ANNOTATION: print at L270
            }

            matricesSize=matrices.size();
```

```
        matrix=matrices.get(counter);
    }
}
System.out.println("\n\n------Matrix" +"("+counter+"):");    // ANNOTATION: print at L277
matrix=matrices.get(counter);

for (List<Integer> row: matrix)    // ANNOTATION: loop over indices
{
    System.out.println(row);    // ANNOTATION: print at L282
}

matrixMultiplicationList = new ArrayList<>();

try
{
    nextMatrix=matrices.get(counter+1);
}
catch (IndexOutOfBoundsException e)
{
    System.out.println("Single matrix only");    // ANNOTATION: print at L293
    matrix=matrices.get(counter);
    System.out.println("\n***CALCULATION STEPS*******");    // ANNOTATION: print at L295

    for (List<Integer> row: matrix)    // ANNOTATION: loop over indices
    {
        System.out.println(row);    // ANNOTATION: print at L299
    }
    System.exit(0);
}

System.out.println("^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^");    // ANNOTATION: print at L304

matrixMultiplicationCalculations=new StringJoiner[matrix.size()][nextMatrix.get(0).size()];

System.out.println("Size of Matrix multiplication row: " + matrix.size());    // ANNOTATION: print at L308
System.out.println("Size of Matrix multiplication columns: " + nextMatrix.get(0).size());    // ANNOTATION: print at L309

System.out.println("*******Customising Storage Grid for Multiplication Matrix**********["+matrix.size()+"]["+nextMatrix.get(0).size()+"]***************");    /

for (int r=0; r<matrix.size(); r++)    // ANNOTATION: loop over indices
{
    for (int q=0; q<nextMatrix.get(0).size(); q++)    // ANNOTATION: loop over indices
    {
        System.out.println("Configuring for index: " + r + "," + q);    // ANNOTATION: print at L317
        matrixMultiplicationCalculations[r][q] = new StringJoiner("+");
```

```
    }
}

for (List<Integer> ee: matrix)    // ANNOTATION: loop over indices
{
    rowMatrix = String.valueOf(ee);
    System.out.println("THIS IS ROW COUNTER MATRIX("+counter+"): " + rowCounterFirstMatrix);    // ANNOTATION: print at L325
    System.out.println("----------------------------------------THIS IS ROW DATA MATRIX("+counter+"): " + ee);    // ANNOTATION: print at L326

    numWritesToLastIndexOnRowMultiplicationMatrix=0;
    hasReachedEndRow=false;

    for (int k=0; k<ee.size(); k++)    // ANNOTATION: loop over indices
    {
        if(counter!=matricesSize-1)
        {
            rowCounterNextMatrix=0;

            if (!hasStartedMultiplication)    // ANNOTATION: condition
            {
                System.out.println("**************************CONFIGURING SIZE FOR BLANK MULTIPLICATION MATRIX*****************************************");
                matrixMultiplication = new Integer[matrix.size()][nextMatrix.get(0).size()];

                System.out.println("Multiplication matrix (W=" + matrix.size()+")  x  (H=" + nextMatrix.get(0).size()+")"     // ANNOTATION: print at L342
                + " configured to store Matrix " + counter + " x  Matrix " + (counter+1));

                try
                {
                    for (int a = 0; a < matrix.size(); a++)     // ANNOTATION: loop over indices
                    {
                        for (int b = 0; b < nextMatrix.get(0).size(); b++)     // ANNOTATION: loop over indices
                        {
                            matrixMultiplication[a][b] = 0;
                        }
                    }
                }

                catch (ArrayIndexOutOfBoundsException e)
                {
                    System.out.println("\n33The current dimension of matrixMultiplication: "     // ANNOTATION: print at L358
                    + "["+matrix.size()+"]"+ "["+nextMatrix.get(0).size()+"]");
                }
                hasStartedMultiplication=true;
            }
            if (nextMatrix.get(0).size()==0)    // ANNOTATION: condition
```

```
{
    System.out.println("8The current dimension of matrixMultiplication: "     // ANNOTATION: print at L366
    + "["+matrix.size()+"]"+"["+nextMatrix.get(0).size()+"]");
    System.out.println("2It EXPECTS: " + nextMatrix.get(0).size() + " elements per row");    // ANNOTATION: print at L368
    System.out.println("Hence multiplication matrix has no width");    // ANNOTATION: print at L369
    System.out.println("matrix" + "("+(counter+1)+")" + " row:" + rowCounterNextMatrix +  " "     // ANNOTATION: print at L370
    + nextMatrix.get(0) + " size: " + nextMatrix.get(0).size());
    System.out.println(matrices);    // ANNOTATION: print at L372
    System.exit(0);
}

for (int z=0; z<nextMatrix.get(0).size(); z++)    // ANNOTATION: loop over indices
{
    System.out.println("\n*****************Column number: " + z);    // ANNOTATION: print at L378
    posPrevMatrixRowLevel=0;
    numWritesAtIndexWithinMultiplicationMatrix=0;

    for (int m=0; m<nextMatrix.size(); m++)    // ANNOTATION: loop over indices
    {
        for (int j=0;j<nextMatrix.get(m).size();j++)    // ANNOTATION: loop over indices
        {

            if (numWritesAtIndexWithinMultiplicationMatrix==nextMatrix.size())    // ANNOTATION: condition
            {
                System.out.println("*************************************************************");    // ANNOTATION: print at L389
                break;
            }
            else
            {
                System.out.println("retrieving row: " + rowCounterNextMatrix     // ANNOTATION: print at L394
                + " from matrix (" + (counter+1)+")");
            }
            rowInfo=new ArrayList<>(nextMatrix.get(m));

            if (!hasReachedEndRow)    // ANNOTATION: condition
            {
                seekPosition=0;
                excessRowItems = new StringJoiner(",");

                if (rowInfo.size()!=matrixMultiplication[0].length)    // ANNOTATION: condition
                {
                    currentEntry = "2Row length:" + rowInfo.size()+" (matrix" + "("+(counter+1)+") row:"
                    +rowCounterNextMatrix+ "   " + nextMatrix.get(m)+") "
                    + " inconsistent with length expectations"
                    + "("+matrixMultiplication[0].length+") in multiplcation matrix";
```

```
    if (!currentEntry.equals(prevEntry))    // ANNOTATION: condition
    {
        System.out.println(currentEntry);    // ANNOTATION: print at L413
        excessRowReport.add(currentEntry);

        for (int x: nextMatrix.get(m))    // ANNOTATION: loop over indices
        {
            seekPosition++;

            if (seekPosition>matrixMultiplication[0].length)    // ANNOTATION: condition
            {
                excessRowItems.add(String.valueOf(x));
            }
        }
    }
}

prevEntry = currentEntry;

if (seekPosition!=0)    // ANNOTATION: condition
{
    System.out.println("["+excessRowItems+"] are exempt from multiplcation matrix");    // ANNOTATION: print at L432
    excessRowReport.add("["+excessRowItems+"] are exempt from multiplcation matrix");

    if(excessRowItems.toString().length()==0)
    {
        System.out.println("10The current dimension of matrixMultiplication: "     // ANNOTATION: print at L437
        + "["+matrix.size()+"]"+"["+nextMatrix.get(0).size()+"]");
        System.out.println("3It EXPECTS: " + nextMatrix.get(0).size() + " elements per row");    // ANNOTATION: print at L439

        System.out.println("matrix" + "("+(counter+1)+")" + " row:" + rowCounterNextMatrix     // ANNOTATION: print at L441
        +  " " + "(content:"+nextMatrix.get(m) +")" + " size:" + nextMatrix.get(m).size()
        + " invalidates this");
        System.out.println(matrices);    // ANNOTATION: print at L444
    }
}
try
{
    System.out.println("Matrix["+(counter+1)+"] \trow element["+z+"]: " + rowInfo.get(z) +"\t\t" + nextMatrix.get(m));    // ANNOTA

}
catch (IndexOutOfBoundsException e)
{
    System.out.println("\n24The current dimension of matrixMultiplication: "     // ANNOTATION: print at L454
    + "["+nextMatrix.get(0).size()+"]"+"["+matrix.size()+"]");
```

```
                                    System.out.println("4It EXPECTS: " + nextMatrix.get(0).size() + " elements per row");    // ANNOTATION: print at L456
                                    System.out.println("matrix" + "("+(counter+1)+")" + " row:" + rowCounterNextMatrix     // ANNOTATION: print at L457
                                    +  " " + nextMatrix.get(m) + " size: " + nextMatrix.get(m).size() + " invalidates this");
                                    System.out.println(matrices);    // ANNOTATION: print at L459
                                }

                                try
                                {
                                    System.out.println("Matrix["+counter+"] \trow element["+posPrevMatrixRowLevel+"]: " +ee.get(posPrevMatrixRowLevel) +"\t\t" + ee
                                }
                                catch (ArrayIndexOutOfBoundsException e)
                                {
                                    System.out.println("\n1The current dimension of matrixMultiplication: "     // ANNOTATION: print at L468
                                    + "["+matrixMultiplication.length+"]"+"["+matrixMultiplication[0].length+"]");

                                    if (rowInfo.size()>matrixMultiplication.length)    // ANNOTATION: condition
                                    {
                                        System.out.println("1Accessing matrix(" + counter+") row index: "     // ANNOTATION: print at L473
                                        + rowCounterNextMatrix + " from " + nextMatrix.get(m) + " invalidates this");
                                        System.out.println("Remove " + nextMatrix.get(m) + " and rows beyond from matrix("     // ANNOTATION: print at L475
                                        + (counter+1)+") row index: " + m);
                                    }
                                ■
■■■■■    if(ee.size()>matrixMultiplication.length)
                                    {
                                        System.out.println("5It EXPECTS: " + matrix.size() + " elements per row");    // ANNOTATION: print at L481
                                        System.out.println("matrix" + "("+(counter)+")" + " row:" + rowCounterFirstMatrix     // ANNOTATION: print at L482
                                        +  " " + matrix.get(rowCounterFirstMatrix) + " size: "
                                        + matrix.get(rowCounterFirstMatrix).size() + " invalidates this");
                                    }
                                ■■■
                                    if(nextMatrix.get(m).size()>matrixMultiplication[0].length)
                                    {
                                        System.out.println("6It EXPECTS: " + matrix.size() + " elements per row");    // ANNOTATION: print at L489
                                        System.out.println("matrix" + "("+(counter+1)+")"     // ANNOTATION: print at L490
                                        + " row:" + rowCounterNextMatrix +  " " + nextMatrix.get(m) + " size: "
                                        + nextMatrix.get(m).size() + " invalidates this");
                                    }

                                    if (ee.size()<rowCounterNextMatrix)    // ANNOTATION: condition
                                    {
                                        System.out.println("2Accessing matrix(" + counter+") row index: "     // ANNOTATION: print at L497
                                        + rowCounterNextMatrix + " from " + ee + " invalidates this");
                                        System.out.println("Remove " + nextMatrix.get(m)     // ANNOTATION: print at L499
                                        + " and rows beyond from matrix(" + (counter+1)+") row index: " + m);
```

```
        System.out.println(matrices);    // ANNOTATION: print at L501
        System.exit(0);
    }
}

try
{
    multiplication =  ee.get(posPrevMatrixRowLevel) *rowInfo.get(z);
}

catch (ArrayIndexOutOfBoundsException e)
{
    System.out.println("2Matrix("+(counter)+")" + "   row:" + ee +"\thas no content at index["+m+"]");    // ANNOTATION: print at L
    System.out.println("Hence can not perform multiplication with Matrix("+(counter+1)+")" + "   row:" + (rowCounterNextMatrix+1)+"
    +  "  index["+z+"]: " + rowInfo.get(z));
    System.exit(0);
}

catch (IndexOutOfBoundsException e)
{
    System.out.println("1Matrix("+(counter)+")" + "   row:" + rowCounterFirstMatrix+"\t" +  "value at index["+z+"]: " + ee.get(posP
    +"  has no available supporting entry in \nMatrix("+(counter+1)+")" + "   row:" + (rowCounterNextMatrix+1)+"\t" +  "\t index["+
    System.exit(0);
}
System.out.println(ee.get(posPrevMatrixRowLevel) + " X " + rowInfo.get(z) + "= " + multiplication);    // ANNOTATION: print at L525

try
{
    matrixMultiplication[rowCounterFirstMatrix][z] = matrixMultiplication[rowCounterFirstMatrix][z]
    + multiplication;
}
catch (ArrayIndexOutOfBoundsException e)
{
    System.out.println("\n2The current dimension of matrixMultiplication: "     // ANNOTATION: print at L534
    + "["+matrix.size()+"]"+"["+nextMatrix.get(0).size()+"]");
    System.out.println("1It EXPECTS: " + nextMatrix.get(0).size() + " elements per row");    // ANNOTATION: print at L536
    System.out.println("matrix" + "("+(counter+1)+")" + " row:" + rowCounterNextMatrix +  " "     // ANNOTATION: print at L537
    + nextMatrix.get(m) + " size: " + nextMatrix.get(m).size() + " invalidates this");

    System.out.println("1Matrix("+(counter)+")" + "   row:" + rowCounterFirstMatrix+"\t" +  "value at index["+z+"]: " + ee.get(posP
    +"  has no available supporting entry in \nMatrix("+(counter+1)+")" + "   row:" + (rowCounterNextMatrix+1)+"\t" +  "\t index["+
    System.exit(0);
}
System.out.println("Value in multiplication matrix: "     // ANNOTATION: print at L544
+ matrixMultiplication[rowCounterFirstMatrix][z] + "\t\t"
```

```
                        + "["+rowCounterFirstMatrix+"],["+z+"]" );
                        numWritesAtIndexWithinMultiplicationMatrix++;
                        matrixMultiplicationCalculations[rowCounterFirstMatrix][z].add("(" + ee.get(posPrevMatrixRowLevel)
                        + "x" +rowInfo.get(z) +" ="+String.valueOf(multiplication)+")");

                        System.out.println(matrixMultiplicationCalculations[rowCounterFirstMatrix][z]+"\n");    // ANNOTATION: print at L551

                        if (z==(matrixMultiplication.length-1))    // ANNOTATION: condition
                        {
                            numWritesToLastIndexOnRowMultiplicationMatrix++;

                            if (numWritesToLastIndexOnRowMultiplicationMatrix==nextMatrix.size())    // ANNOTATION: condition
                            {
                                hasReachedEndRow=true;
                            }
                        }
                        posPrevMatrixRowLevel++;

                        break;
                    }
                    else
                    {
                        System.out.println("-------------------------------------------------");    // ANNOTATION: print at L568
                        System.out.println("previous written to each index location of multiplication matrix: "     // ANNOTATION: print at L569
                        + nextMatrix.size() + "  MAXIMUM times");
                        System.out.println("-------------------------------------------------");    // ANNOTATION: print at L571

                        j=nextMatrix.get(m).size();
                        m=nextMatrix.size()-1;
                    }
                }
                rowCounterNextMatrix=0;
            }
        }
        System.out.println("Processed matrix("+(counter+1)+")  column("+k+")"+  " AGAINST " + "MATRIX("+counter+")  row("+rowCounterFirstMatrix+"): " + ee
    }
    numElementsInRow++;
}
if(counter!=(matricesSize-1))
{
    for (List<Integer> rows: matrices.get((counter+1)))    // ANNOTATION: loop over indices
    {
        rowsNextMatrix++;
    }
    if ((numElementsInRow!=rowsNextMatrix && numElementsInRow<matrices.get(counter).size()) || numElementsInRow==0)    // ANNOTATION: condition
```

```
    {
        System.out.println("INVALID VERIFICATION=>  columns " + "("+numElementsInRow+") in Matrix"  + "("+counter+")"     // ANNOTATION: print at L592
        + "(Row:" +rowCounterFirstMatrix + " content: " +  rowMatrix +  ") should match "
        +  rowsNextMatrix + " rows in matrix("+(counter+1)+")");
        System.out.println(matrices);    // ANNOTATION: print at L595
        System.exit(0);
    }
    else
    {
        System.out.println("VERIFICATION SUCCESSFUL(MATRIX("+counter+")  row("+rowCounterFirstMatrix+")) =>  ***COLUMNS" + "("+numElementsInRow+")\t   Matr

        excessColItems = new StringJoiner(",");

        if (numElementsInRow>nextMatrix.size())    // ANNOTATION: condition
        {
            currentColumnEntry = "columns" + "("+numElementsInRow+") in Matrix"  + "("+counter+")"
        + "(Row:" +rowCounterFirstMatrix + " content: " +  rowMatrix +  ") inconsistent with "
        +  rowsNextMatrix + " rows in matrix("+(counter+1)+").";

            System.out.println(currentColumnEntry);    // ANNOTATION: print at L610

            for (int x: matrix.get(rowCounterFirstMatrix))    // ANNOTATION: loop over indices
            {
                seekPosition++;

                if (seekPosition>rowsNextMatrix)    // ANNOTATION: condition
                {
                    excessColItems.add(String.valueOf(x));
                }
            }
            if (seekPosition!=0)    // ANNOTATION: condition
            {
                System.out.println("["+excessColItems+"] are exempt from multiplcation matrix");    // ANNOTATION: print at L623

                excessColReport.add(currentColumnEntry);
                excessColReport.add("["+excessColItems+"] are exempt from multiplcation matrix"+"\n");
            }
            seekPosition=0;
        }

    }
    rowsNextMatrix=0;
}
numElementsInRow=0;
rowCounterFirstMatrix++;
```

```
            }
            rowCounterFirstMatrix=0;

            counter=counter+2;
            rowsNextMatrix=0;

        }while (counter<matricesSize);

        for (Integer []m: matrixMultiplication)    // ANNOTATION: loop over indices
        {
            System.out.println(Arrays.toString(m));    // ANNOTATION: print at L646
        }

        System.out.println("\n***CALCULATION STEPS*******");    // ANNOTATION: print at L649

        for (int q=0; q<matrixMultiplicationCalculations.length; q++)    // ANNOTATION: loop over indices
        {
            matrixRowCalculations = new StringJoiner("");

            for (int r=0; r<matrixMultiplicationCalculations[0].length; r++)    // ANNOTATION: loop over indices
            {
                matrixRowCalculations.add(" ["+matrixMultiplicationCalculations[q][r]+"]");

            }
            System.out.println(matrixRowCalculations);    // ANNOTATION: print at L660
        }

        for (Integer []m: matrixMultiplication)    // ANNOTATION: loop over indices
        {
            matrixMultiplicationList.add(Arrays.asList(m));
        }

        System.out.println("\n-----MATRIX A   Analysis------");    // ANNOTATION: print at L668
        System.out.println(excessColReport);    // ANNOTATION: print at L669

        System.out.println("\n\n-----MATRIX B   Analysis------");    // ANNOTATION: print at L671
        System.out.println(excessRowReport);    // ANNOTATION: print at L672

        return matrixMultiplicationList;
    }
}
```

Test #1 (source line: 49)

Raw literal:
```
{   {{1},{5}},{{9,88}}}
```

Actual Java Lists executed:
```
[
  [[1], [5]],
  [[9, 88]]
]
```

Step 1: Multiply A(2, 1) x B(1, 2)
```
  My-code result: [
    [9, 88],
    [45, 440]
  ]
  Expected (strict): [
    [9, 88],
    [45, 440]
  ]
```

Test #2 (source line: 52)

Raw literal:

```
{   {{1,2,3},{4,5,6}}}
```

Actual Java Lists executed:

```
[
  [[1, 2, 3], [4, 5, 6]]
]
```

```
No multiplication steps for this test.
```

Test #3 (source line: 55)

Raw literal:

```
{   {{1,2,3},{4,5,6}},  {{7,8},{9,10},{11,12}},  { {13,14}, {15,16} } }
```

Actual Java Lists executed:

```
[
  [[1, 2, 3], [4, 5, 6]],
  [[7, 8], [9, 10], [11, 12]],
  [[13, 14], [15, 16]]
]
```

Step 1: Multiply A(2, 3) x B(3, 2)

```
  My-code result: [
    [58, 64],
    [139, 154]
  ]
  Expected (strict): [
    [58, 64],
    [139, 154]
  ]
```

Step 2: Multiply A(2, 2) x B(2, 2)

```
  My-code result: [
    [1714, 1836],
    [4117, 4410]
  ]
  Expected (strict): [
    [1714, 1836],
    [4117, 4410]
  ]
```

Test #4 (source line: 58)

Raw literal:
```
{   {{1,2,3,5,6},{4,5,6}},  {{7,8},{9,10},{11,12}},  { {13,14}, {15,16} } }
```

Actual Java Lists executed:
```
[
  [[1, 2, 3, 5, 6], [4, 5, 6]],
  [[7, 8], [9, 10], [11, 12]],
  [[13, 14], [15, 16]]
]
```

Step 1: Multiply A(2, 5) x B(3, 2)
```
  My-code result: [
    [58, 64],
    [139, 154]
  ]
  Expected (strict): Invalid/Discarded (Incompatible dims colsA=5 rowsB=3)
```

Step 2: Multiply A(2, 2) x B(2, 2)
```
  My-code result: [
    [1714, 1836],
    [4117, 4410]
  ]
  Expected (strict): [
    [1714, 1836],
    [4117, 4410]
  ]
```

Test #5 (source line: 61)

Raw literal:
```
{   {{1,2,3,4}},{{5},{6},{7},{8}}}
```

Actual Java Lists executed:
```
[
  [[1, 2, 3, 4]],
  [[5], [6], [7], [8]]
]
```

Step 1: Multiply A(1, 4) x B(4, 1)
```
  My-code result: [
    [70]
  ]
  Expected (strict): [
    [70]
  ]
```

Test #6 (source line: 64)

Raw literal:

```
{   {{1, 2}, {3, 4}}, {{5, 6}, {7, 8}}, {{9, 10}, {11, 12}}  }
```

Actual Java Lists executed:

```
[
  [[1, 2], [3, 4]],
  [[5, 6], [7, 8]],
  [[9, 10], [11, 12]]
]
```

Step 1: Multiply A(2, 2) x B(2, 2)

```
  My-code result: [
    [19, 22],
    [43, 50]
  ]
  Expected (strict): [
    [19, 22],
    [43, 50]
  ]
```

Step 2: Multiply A(2, 2) x B(2, 2)

```
  My-code result: [
    [413, 454],
    [937, 1030]
  ]
  Expected (strict): [
    [413, 454],
    [937, 1030]
  ]
```

Test #7 (source line: 67)

Raw literal:

```
{   {{1,2,3},{4,5,6}},  {{7,8},{10},{11,12}},  { {13,14}, {15,16} } }
```

Actual Java Lists executed:

```
[
  [[1, 2, 3], [4, 5, 6]],
  [[7, 8], [10], [11, 12]],
  [[13, 14], [15, 16]]
]
```

Step 1: Multiply A(2, 3) x B(3, 2)

```
  My-code result: [
    [60, 44],
    [144, 104]
  ]
  Expected (strict): Invalid/Discarded (Invalid/Discarded due to missing B element)
```

Step 2: Multiply A(2, 2) x B(2, 2)

```
  My-code result: [
    [1440, 1544],
    [3432, 3680]
  ]
  Expected (strict): [
    [1440, 1544],
    [3432, 3680]
  ]
```

Test #8 (source line: 71)

Raw literal:

```
{   {{1,2,3},{4,5,6}},  {{7,8,999},{10,11},{11,12}},  { {13,14}, {15,16} } }
```

Actual Java Lists executed:

```
[
  [[1, 2, 3], [4, 5, 6]],
  [[7, 8, 999], [10, 11], [11, 12]],
  [[13, 14], [15, 16]]
]
```

Step 1: Multiply A(2, 3) x B(3, 3)

```
  My-code result: [
    [60, 66, 999],
    [144, 159, 3996]
  ]
  Expected (strict): Invalid/Discarded (Invalid/Discarded due to missing B element)
```

Step 2: Multiply A(2, 3) x B(2, 2)

```
  My-code result: [
    [1770, 1896],
    [4257, 4560]
  ]
  Expected (strict): Invalid/Discarded (Incompatible dims colsA=3 rowsB=2)
```

Test #9 (source line: 74)

Raw literal:
```
{   {{1,2,3},{4,5}},  {{7,8},{10,11},{11,12}},  { {13,14,17}, {15,16} } }
```

Actual Java Lists executed:
```
[
  [[1, 2, 3], [4, 5]],
  [[7, 8], [10, 11], [11, 12]],
  [[13, 14, 17], [15, 16]]
]
```

Step 1: Multiply A(2, 3) x B(3, 2)
```
  My-code result: [
    [60, 66],
    [78, 87]
  ]
  Expected (strict): Invalid/Discarded (Invalid/Discarded due to missing A element)
```

Step 2: Multiply A(2, 2) x B(2, 3)
```
  My-code result: [
    [1770, 1896, 1020],
    [2319, 2484, 1326]
  ]
  Expected (strict): Invalid/Discarded (Invalid/Discarded due to missing B element)
```

Test #10 (source line: 77)

Raw literal:

```
{   {{1,2},{4,5,6}},  {{7,8},{10,11},{11,12}},  { {13,14,17}, {15,16,17} } }
```

Actual Java Lists executed:

```
[
  [[1, 2], [4, 5, 6]],
  [[7, 8], [10, 11], [11, 12]],
  [[13, 14, 17], [15, 16, 17]]
]
```

Step 1: Multiply A(2, 3) x B(3, 2)

```
  My-code result: [
    [27, 30],
    [144, 159]
  ]
  Expected (strict): Invalid/Discarded (Invalid/Discarded due to missing A element)
```

Step 2: Multiply A(2, 2) x B(2, 3)

```
  My-code result: [
    [801, 858, 969],
    [4257, 4560, 5151]
  ]
  Expected (strict): [
    [801, 858, 969],
    [4257, 4560, 5151]
  ]
```

Test #11 (source line: 81)

Raw literal:

```
{   {{},{4,5,6}},  {{7,8,6},{10,14},{11,12}},  { {13}, {15,16} } }
```

Actual Java Lists executed:

```
[
  [[], [4, 5, 6]],
  [[7, 8, 6], [10, 14], [11, 12]],
  [[13], [15, 16]]
]
```

Step 1: Multiply A(2, 3) x B(3, 3)

```
  My-code result: [
    [null, null, null],
    [144, 174, 24]
  ]
  Expected (strict): Invalid/Discarded (Invalid/Discarded due to missing A element)
```

Step 2: Multiply A(2, 3) x B(2, 2)

```
  My-code result: [
    [null, null],
    [4482, 2784]
  ]
  Expected (strict): Invalid/Discarded (Incompatible dims colsA=3 rowsB=2)
```

Test #12 (source line: 84)

Raw literal:

```
{   {{1,2},{3,4},{5,6}},  {{7,8},{10,14},{0}},  { {13}, {15,16} } }
```

Actual Java Lists executed:

```
[
  [[1, 2], [3, 4], [5, 6]],
  [[7, 8], [10, 14], [0]],
  [[13], [15, 16]]
]
```

Step 1: Multiply A(3, 2) x B(3, 2)

```
  My-code result: [
    [27, 36],
    [61, 80],
    [95, 124]
  ]
  Expected (strict): Invalid/Discarded (Incompatible dims colsA=2 rowsB=3)
```

Step 2: Multiply A(3, 2) x B(2, 2)

```
  My-code result: [
    [891, 576],
    [1993, 1280],
    [3095, 1984]
  ]
  Expected (strict): Invalid/Discarded (Invalid/Discarded due to missing B element)
```

Test #13 (source line: 87)

Raw literal:

```
{   {{1,2,3},{4,5,6}},  {{7,8},{9,10},{11,12}},  { {0}, {16,17},{5}} }
```

Actual Java Lists executed:

```
[
  [[1, 2, 3], [4, 5, 6]],
  [[7, 8], [9, 10], [11, 12]],
  [[0], [16, 17], [5]]
]
```

Step 1: Multiply A(2, 3) x B(3, 2)

```
  My-code result: [
    [58, 64],
    [139, 154]
  ]
  Expected (strict): [
    [58, 64],
    [139, 154]
  ]
```

Step 2: Multiply A(2, 2) x B(3, 2)

```
  My-code result: [
    [1024, 1088],
    [2464, 2618]
  ]
  Expected (strict): Invalid/Discarded (Incompatible dims colsA=2 rowsB=3)
```

Test #14 (source line: 89)

Raw literal:
```
{   {{1,2,3},{4,5,6}},  {{7,8},{9,10},{11}},  { {13,14}, {16,17} } }
```

Actual Java Lists executed:
```
[
  [[1, 2, 3], [4, 5, 6]],
  [[7, 8], [9, 10], [11]],
  [[13, 14], [16, 17]]
]
```

Step 1: Multiply A(2, 3) x B(3, 2)
```
  My-code result: [
    [58, 28],
    [139, 82]
  ]
  Expected (strict): Invalid/Discarded (Invalid/Discarded due to missing B element)
```

Step 2: Multiply A(2, 2) x B(2, 2)
```
  My-code result: [
    [1202, 1288],
    [3119, 3340]
  ]
  Expected (strict): [
    [1202, 1288],
    [3119, 3340]
  ]
```

Test #15 (source line: 92)

Raw literal:

```
{   {{1,2,3},{4,5,6}},  {{7,8},{9,2},{11,12}},  { {13,14}, {16,87},{3,4} } }
```

Actual Java Lists executed:

```
[
  [[1, 2, 3], [4, 5, 6]],
  [[7, 8], [9, 2], [11, 12]],
  [[13, 14], [16, 87], [3, 4]]
]
```

Step 1: Multiply A(2, 3) x B(3, 2)

```
  My-code result: [
    [58, 48],
    [139, 114]
  ]
  Expected (strict): [
    [58, 48],
    [139, 114]
  ]
```

Step 2: Multiply A(2, 2) x B(3, 2)

```
  My-code result: [
    [1522, 4988],
    [3631, 11864]
  ]
  Expected (strict): Invalid/Discarded (Incompatible dims colsA=2 rowsB=3)
```

Test #16 (source line: 95)

Raw literal:
```
{   {{1,2,3},{4,5,6}},  {{},{10,14},{11,12}},  { {13}, {15,16} } }
```

Actual Java Lists executed:
```
[
  [[1, 2, 3], [4, 5, 6]],
  [[], [10, 14], [11, 12]],
  [[13], [15, 16]]
]
```

Step 1: Multiply A(2, 3) x B(3, 2)
```
  My-code result: [
    [53, 64],
    [116, 142]
  ]
  Expected (strict): Invalid/Discarded (Invalid/Discarded due to missing B element)
```

Step 2: Multiply A(2, 2) x B(2, 2)
```
  My-code result: [
    [1649, 1024],
    [3638, 2272]
  ]
  Expected (strict): Invalid/Discarded (Invalid/Discarded due to missing B element)
```

Test #17 (source line: 98)

Raw literal:
```
{   {{1,2,3},{4,5,6}},  {{1,7},{25,14},{}},  { {13,15}, {15,16} } }
```

Actual Java Lists executed:
```
[
  [[1, 2, 3], [4, 5, 6]],
  [[1, 7], [25, 14], []],
  [[13, 15], [15, 16]]
]
```

Step 1: Multiply A(2, 3) x B(3, 2)
```
  My-code result: [
    [51, 35],
    [129, 98]
  ]
  Expected (strict): Invalid/Discarded (Invalid/Discarded due to missing B element)
```

Step 2: Multiply A(2, 2) x B(2, 2)
```
  My-code result: [
    [1188, 1325],
    [3147, 3503]
  ]
  Expected (strict): [
    [1188, 1325],
    [3147, 3503]
  ]
```

Test #18 (source line: 102)

Raw literal:
```
{   {{1,2,3},{}},  {{1,7},{25,14},{4,8}},  { {13,15}, {15,16} } }
```

Actual Java Lists executed:
```
[
  [[1, 2, 3], []],
  [[1, 7], [25, 14], [4, 8]],
  [[13, 15], [15, 16]]
]
```

Step 1: Multiply A(2, 3) x B(3, 2)
```
  My-code result: [
    [63, 59],
    [null, null]
  ]
  Expected (strict): Invalid/Discarded (Invalid/Discarded due to missing A element)
```

Step 2: Multiply A(2, 2) x B(2, 2)
```
  My-code result: [
    [1704, 1889],
    [null, null]
  ]
  Expected (strict): Invalid/Discarded (Invalid/Discarded due to missing A element)
```

Test #19 (source line: 105)

Raw literal:

```
{   {{1,2,3},{4,5,6}},  {{1,7},{25,14},{4,8}},  { {}, {15,16} } }
```

Actual Java Lists executed:

```
[
  [[1, 2, 3], [4, 5, 6]],
  [[1, 7], [25, 14], [4, 8]],
  [[], [15, 16]]
]
```

Step 1: Multiply A(2, 3) x B(3, 2)

```
  My-code result: [
    [63, 59],
    [153, 146]
  ]
  Expected (strict): [
    [63, 59],
    [153, 146]
  ]
```

Step 2: Multiply A(2, 2) x B(2, 2)

```
  My-code result: [
    [885, 944],
    [2190, 2336]
  ]
  Expected (strict): Invalid/Discarded (Invalid/Discarded due to missing B element)
```

Test #20 (source line: 108)

Raw literal:
```
{   {{1,2,3},{4,5,6}},  {{1,7},{25,14},{4,8}},  { {34,45}, {} } }
```

Actual Java Lists executed:
```
[
  [[1, 2, 3], [4, 5, 6]],
  [[1, 7], [25, 14], [4, 8]],
  [[34, 45], []]
]
```

Step 1: Multiply A(2, 3) x B(3, 2)
```
  My-code result: [
    [63, 59],
    [153, 146]
  ]
  Expected (strict): [
    [63, 59],
    [153, 146]
  ]
```

Step 2: Multiply A(2, 2) x B(2, 2)
```
  My-code result: [
    [2142, 2835],
    [5202, 6885]
  ]
  Expected (strict): Invalid/Discarded (Invalid/Discarded due to missing B element)
```

Test #21 (source line: 111)

Raw literal:

```
{   {{1,2,3},{4,5,6}},  {{7,8,6},{10,14},{11,12}},  { {13,14}, {15,16} } }
```

Actual Java Lists executed:

```
[
  [[1, 2, 3], [4, 5, 6]],
  [[7, 8, 6], [10, 14], [11, 12]],
  [[13, 14], [15, 16]]
]
```

Step 1: Multiply A(2, 3) x B(3, 3)

```
  My-code result: [
    [60, 72, 6],
    [144, 174, 24]
  ]
  Expected (strict): Invalid/Discarded (Invalid/Discarded due to missing B element)
```

Step 2: Multiply A(2, 3) x B(2, 2)

```
  My-code result: [
    [1860, 1992],
    [4482, 4800]
  ]
  Expected (strict): Invalid/Discarded (Incompatible dims colsA=3 rowsB=2)
```

Test #22 (source line: 115)

Raw literal:

```
{   {{1,2,3},{4,5,6,9}},  {{7,8},{9,10},{11,12}},  { {13,14}, {15,16} } }
```

Actual Java Lists executed:

```
[
  [[1, 2, 3], [4, 5, 6, 9]],
  [[7, 8], [9, 10], [11, 12]],
  [[13, 14], [15, 16]]
]
```

Step 1: Multiply A(2, 4) x B(3, 2)

```
  My-code result: [
    [58, 64],
    [139, 154]
  ]
  Expected (strict): Invalid/Discarded (Incompatible dims colsA=4 rowsB=3)
```

Step 2: Multiply A(2, 2) x B(2, 2)

```
  My-code result: [
    [1714, 1836],
    [4117, 4410]
  ]
  Expected (strict): [
    [1714, 1836],
    [4117, 4410]
  ]
```

Test #23 (source line: 118)

Raw literal:

```
{   {{1,2,3},{4}},  {{7,8},{9,10},{11,12}},  { {13,14}, {15,16} } }
```

Actual Java Lists executed:

```
[
  [[1, 2, 3], [4]],
  [[7, 8], [9, 10], [11, 12]],
  [[13, 14], [15, 16]]
]
```

Step 1: Multiply A(2, 3) x B(3, 2)

```
  My-code result: [
    [58, 64],
    [28, 32]
  ]
  Expected (strict): Invalid/Discarded (Invalid/Discarded due to missing A element)
```

Step 2: Multiply A(2, 2) x B(2, 2)

```
  My-code result: [
    [1714, 1836],
    [844, 904]
  ]
  Expected (strict): [
    [1714, 1836],
    [844, 904]
  ]
```

Test #24 (source line: 122)

Raw literal:

```
{   {{1,2,3},{4,7,2,6}},  {{7,8},{9,10},{11,12}},  { {13,14}, {15,16} } }
```

Actual Java Lists executed:

```
[
  [[1, 2, 3], [4, 7, 2, 6]],
  [[7, 8], [9, 10], [11, 12]],
  [[13, 14], [15, 16]]
]
```

Step 1: Multiply A(2, 4) x B(3, 2)

```
  My-code result: [
    [58, 64],
    [113, 126]
  ]
  Expected (strict): Invalid/Discarded (Incompatible dims colsA=4 rowsB=3)
```

Step 2: Multiply A(2, 2) x B(2, 2)

```
  My-code result: [
    [1714, 1836],
    [3359, 3598]
  ]
  Expected (strict): [
    [1714, 1836],
    [3359, 3598]
  ]
```

Test #25 (source line: 125)

Raw literal:

```
{   {{1,2,3},{4,5,6}},  {{},{10,14},{11,12}},  { {13,14}, {15,16} } }
```

Actual Java Lists executed:

```
[
  [[1, 2, 3], [4, 5, 6]],
  [[], [10, 14], [11, 12]],
  [[13, 14], [15, 16]]
]
```

Step 1: Multiply A(2, 3) x B(3, 2)

```
  My-code result: [
    [53, 64],
    [116, 142]
  ]
  Expected (strict): Invalid/Discarded (Invalid/Discarded due to missing B element)
```

Step 2: Multiply A(2, 2) x B(2, 2)

```
  My-code result: [
    [1649, 1766],
    [3638, 3896]
  ]
  Expected (strict): [
    [1649, 1766],
    [3638, 3896]
  ]
```

Test #26 (source line: 127)

Raw literal:
```
{   {{1,2,3},{4,5,6}},  {{11,14},{},{19,12}},  { {13,14}, {15,16} } }
```

Actual Java Lists executed:
```
[
  [[1, 2, 3], [4, 5, 6]],
  [[11, 14], [], [19, 12]],
  [[13, 14], [15, 16]]
]
```

Step 1: Multiply A(2, 3) x B(3, 2)
```
  My-code result: [
    [68, 50],
    [158, 128]
  ]
  Expected (strict): Invalid/Discarded (Invalid/Discarded due to missing B element)
```

Step 2: Multiply A(2, 2) x B(2, 2)
```
  My-code result: [
    [1634, 1752],
    [3974, 4260]
  ]
  Expected (strict): [
    [1634, 1752],
    [3974, 4260]
  ]
```

Test #27 (source line: 128)

Raw literal:

```
{   {{1,2,3},{4,5,6}},  {{11,14},{44,55},{19,12}},  { {13,14}, {} } }
```

Actual Java Lists executed:

```
[
  [[1, 2, 3], [4, 5, 6]],
  [[11, 14], [44, 55], [19, 12]],
  [[13, 14], []]
]
```

Step 1: Multiply A(2, 3) x B(3, 2)

```
  My-code result: [
    [156, 160],
    [378, 403]
  ]
  Expected (strict): [
    [156, 160],
    [378, 403]
  ]
```

Step 2: Multiply A(2, 2) x B(2, 2)

```
  My-code result: [
    [2028, 2184],
    [4914, 5292]
  ]
  Expected (strict): Invalid/Discarded (Invalid/Discarded due to missing B element)
```

Test #28 (source line: 130)

Raw literal:

```
{   {{1,2,3},{4,5,6}},  {{11,14},{44,55},{19,12}},  { {13,14}, {0} } }
```

Actual Java Lists executed:

```
[
  [[1, 2, 3], [4, 5, 6]],
  [[11, 14], [44, 55], [19, 12]],
  [[13, 14], [0]]
]
```

Step 1: Multiply A(2, 3) x B(3, 2)

```
  My-code result: [
    [156, 160],
    [378, 403]
  ]
  Expected (strict): [
    [156, 160],
    [378, 403]
  ]
```

Step 2: Multiply A(2, 2) x B(2, 2)

```
  My-code result: [
    [2028, 2184],
    [4914, 5292]
  ]
  Expected (strict): Invalid/Discarded (Invalid/Discarded due to missing B element)
```

Test #29 (source line: 132)

Raw literal:

```
{   {{1,2,3},{4,5,6}},  {{11,14},{44,55},{19,12}},  { {13,14}, {0,5},{0} } }
```

Actual Java Lists executed:

```
[
  [[1, 2, 3], [4, 5, 6]],
  [[11, 14], [44, 55], [19, 12]],
  [[13, 14], [0, 5], [0]]
]
```

Step 1: Multiply A(2, 3) x B(3, 2)

```
  My-code result: [
    [156, 160],
    [378, 403]
  ]
  Expected (strict): [
    [156, 160],
    [378, 403]
  ]
```

Step 2: Multiply A(2, 2) x B(3, 2)

```
  My-code result: [
    [2028, 2984],
    [4914, 7307]
  ]
  Expected (strict): Invalid/Discarded (Incompatible dims colsA=2 rowsB=3)
```

Test #30 (source line: 135)

Raw literal:
```
{   {{1,2,3},{4,5,6}},  {{7,8},{10,14},{11,12}},  { {13}, {15,16} } }
```

Actual Java Lists executed:
```
[
  [[1, 2, 3], [4, 5, 6]],
  [[7, 8], [10, 14], [11, 12]],
  [[13], [15, 16]]
]
```

Step 1: Multiply A(2, 3) x B(3, 2)
```
  My-code result: [
    [60, 72],
    [144, 174]
  ]
  Expected (strict): [
    [60, 72],
    [144, 174]
  ]
```

Step 2: Multiply A(2, 2) x B(2, 2)
```
  My-code result: [
    [1860, 1152],
    [4482, 2784]
  ]
  Expected (strict): Invalid/Discarded (Invalid/Discarded due to missing B element)
```

Test #31 (source line: 139)

Raw literal:

```
{   {{1,2,3},{4,5,6}},  {{11,14},{44,55},{19,12}},  { {13,14}, {0} } }
```

Actual Java Lists executed:

```
[
  [[1, 2, 3], [4, 5, 6]],
  [[11, 14], [44, 55], [19, 12]],
  [[13, 14], [0]]
]
```

Step 1: Multiply A(2, 3) x B(3, 2)

```
  My-code result: [
    [156, 160],
    [378, 403]
  ]
  Expected (strict): [
    [156, 160],
    [378, 403]
  ]
```

Step 2: Multiply A(2, 2) x B(2, 2)

```
  My-code result: [
    [2028, 2184],
    [4914, 5292]
  ]
  Expected (strict): Invalid/Discarded (Invalid/Discarded due to missing B element)
```

Test #32 (source line: 141)

Raw literal:

```
{   {{1,2,3},{4,5,6}},  {{7,8},{10,14},{15,12,65,55}},  { {13,99}, {15,16} } }
```

Actual Java Lists executed:

```
[
  [[1, 2, 3], [4, 5, 6]],
  [[7, 8], [10, 14], [15, 12, 65, 55]],
  [[13, 99], [15, 16]]
]
```

Step 1: Multiply A(2, 3) x B(3, 4)

```
  My-code result: [
    [72, 72, 195, 165],
    [168, 174, 390, 330]
  ]
  Expected (strict): Invalid/Discarded (Invalid/Discarded due to missing B element)
```

Step 2: Multiply A(2, 4) x B(2, 2)

```
  My-code result: [
    [2016, 8280],
    [4794, 19416]
  ]
  Expected (strict): Invalid/Discarded (Incompatible dims colsA=4 rowsB=2)
```

Test #33 (source line: 143)

Raw literal:

```
{   {{1,2,3},{4,5,6},{7,8,9}},  {{10,11,12},{13,14,15},{16,17,18}},  { {}, {22,23,24}} }
```

Actual Java Lists executed:

```
[
  [[1, 2, 3], [4, 5, 6], [7, 8, 9]],
  [[10, 11, 12], [13, 14, 15], [16, 17, 18]],
  [[], [22, 23, 24]]
]
```

Step 1: Multiply A(3, 3) x B(3, 3)

```
  My-code result: [
    [84, 90, 96],
    [201, 216, 231],
    [318, 342, 366]
  ]
  Expected (strict): [
    [84, 90, 96],
    [201, 216, 231],
    [318, 342, 366]
  ]
```

Step 2: Multiply A(3, 3) x B(2, 3)

```
  My-code result: [
    [1980, 2070, 2160],
    [4752, 4968, 5184],
    [7524, 7866, 8208]
  ]
  Expected (strict): Invalid/Discarded (Incompatible dims colsA=3 rowsB=2)
```

Test #34 (source line: 145)

Raw literal:

```
{   {{1,2,3},{4,5,6}},  {{7,8},{9,10},{11,12}},  { {13,14}, {15,16} },    { {17,18}, {19,20} },{ {17,18}, {19,20} }}
```

Actual Java Lists executed:

```
[
  [[1, 2, 3], [4, 5, 6]],
  [[7, 8], [9, 10], [11, 12]],
  [[13, 14], [15, 16]],
  [[17, 18], [19, 20]],
  [[17, 18], [19, 20]]
]
```

Step 1: Multiply A(2, 3) x B(3, 2)

```
  My-code result: [
    [58, 64],
    [139, 154]
  ]
  Expected (strict): [
    [58, 64],
    [139, 154]
  ]
```

Step 2: Multiply A(2, 2) x B(2, 2)

```
  My-code result: [
    [1714, 1836],
    [4117, 4410]
  ]
  Expected (strict): [
    [1714, 1836],
    [4117, 4410]
  ]
```

Step 3: Multiply A(2, 2) x B(2, 2)

```
  My-code result: [
    [64022, 67572],
    [153779, 162306]
  ]
  Expected (strict): [
    [64022, 67572],
    [153779, 162306]
  ]
```

Step 4: Multiply A(2, 2) x B(2, 2)

```
  My-code result: [
    [2372242, 2503836],
    [5698057, 6014142]
```

```
]
Expected (strict): [
  [2372242, 2503836],
  [5698057, 6014142]
]
```

Test #35 (source line: 147)

Raw literal:
```
{   {{1,2,3},{4,5,6}},  {{7,8},{9,10},{11,12}},  { {13,14}, {15,16}},  { {17,18}, {19,20},{99,18}, {19,20}}, { {21,22,23,24}, {25,26,27,28},{29,30,31,32}} }
```

Actual Java Lists executed:
```
[
  [[1, 2, 3], [4, 5, 6]],
  [[7, 8], [9, 10], [11, 12]],
  [[13, 14], [15, 16]],
  [[17, 18], [19, 20], [99, 18], [19, 20]],
  [[21, 22, 23, 24], [25, 26, 27, 28], [29, 30, 31, 32]]
]
```

Step 1: Multiply A(2, 3) x B(3, 2)
```
  My-code result: [
    [58, 64],
    [139, 154]
  ]
  Expected (strict): [
    [58, 64],
    [139, 154]
  ]
```

Step 2: Multiply A(2, 2) x B(2, 2)
```
  My-code result: [
    [1714, 1836],
    [4117, 4410]
  ]
  Expected (strict): [
    [1714, 1836],
    [4117, 4410]
  ]
```

Step 3: Multiply A(2, 2) x B(4, 2)
```
  My-code result: [
    [64022, 67572],
    [153779, 162306]
  ]
  Expected (strict): Invalid/Discarded (Incompatible dims colsA=2 rowsB=4)
```

Step 4: Multiply A(2, 2) x B(3, 4)
```
  My-code result: [
    [3033762, 3165356, 3296950, 3428544],
    [7287009, 7603094, 7919179, 8235264]
  ]
  Expected (strict): Invalid/Discarded (Incompatible dims colsA=2 rowsB=3)
```

Test #36 (source line: 149)

Raw literal:

```
{   {{1,2,3},{4,5,6},{7,8,9}},  {{10,11,12},{13,14,15},{16,17,18}},  { {19,20,21}, {22,23,24}} }
```

Actual Java Lists executed:

```
[
  [[1, 2, 3], [4, 5, 6], [7, 8, 9]],
  [[10, 11, 12], [13, 14, 15], [16, 17, 18]],
  [[19, 20, 21], [22, 23, 24]]
]
```

Step 1: Multiply A(3, 3) x B(3, 3)

```
  My-code result: [
    [84, 90, 96],
    [201, 216, 231],
    [318, 342, 366]
  ]
  Expected (strict): [
    [84, 90, 96],
    [201, 216, 231],
    [318, 342, 366]
  ]
```

Step 2: Multiply A(3, 3) x B(2, 3)

```
  My-code result: [
    [3576, 3750, 3924],
    [8571, 8988, 9405],
    [13566, 14226, 14886]
  ]
  Expected (strict): Invalid/Discarded (Incompatible dims colsA=3 rowsB=2)
```

Test #37 (source line: 152)

Raw literal:

```
{   {{1,2,3},{4,5,6},{7,8,9}},  {{11,12},{13,14,15},{16,17,18}} }
```

Actual Java Lists executed:

```
[
  [[1, 2, 3], [4, 5, 6], [7, 8, 9]],
  [[11, 12], [13, 14, 15], [16, 17, 18]]
]
```

Step 1: Multiply A(3, 3) x B(3, 3)

```
  My-code result: [
    [85, 91, 84],
    [205, 220, 183],
    [325, 349, 282]
  ]
  Expected (strict): Invalid/Discarded (Invalid/Discarded due to missing B element)
```

Test #38 (source line: 154)

Raw literal:
```
{   {{},{4,5,6},{7,8,9}},  {{10,11,12},{13,14,15},{16,17,18}} }
```

Actual Java Lists executed:
```
[
  [[], [4, 5, 6], [7, 8, 9]],
  [[10, 11, 12], [13, 14, 15], [16, 17, 18]]
]
```

Step 1: Multiply A(3, 3) x B(3, 3)
```
  My-code result: [
    [null, null, null],
    [201, 216, 231],
    [318, 342, 366]
  ]
  Expected (strict): Invalid/Discarded (Invalid/Discarded due to missing A element)
```

Test #39 (source line: None)

Raw literal:

```
[[[1], [2, 3], [4, 5, 6]], [[7, 8], [9, 10], [11, 12]]]
```

Actual Java Lists executed:

```
[
  [[1], [2, 3], [4, 5, 6]],
  [[7, 8], [9, 10], [11, 12]]
]
```

Step 1: Multiply A(3, 3) x B(3, 2)

```
  My-code result: [
    [7, 8],
    [41, 46],
    [139, 154]
  ]
  Expected (strict): Invalid/Discarded (Invalid/Discarded due to missing A element)
```

Test #40 (source line: None)

Raw literal:

```
[[[1, 2, 3]], [[4], [5], [6]], [[7, 8, 9]]]
```

Actual Java Lists executed:

```
[
  [[1, 2, 3]],
  [[4], [5], [6]],
  [[7, 8, 9]]
]
```

Step 1: Multiply A(1, 3) x B(3, 1)

```
  My-code result: [
    [32]
  ]
  Expected (strict): [
    [32]
  ]
```

Step 2: Multiply A(1, 1) x B(1, 3)

```
  My-code result: [
    [224, 256, 288]
  ]
  Expected (strict): [
    [224, 256, 288]
  ]
```

Test #41 (source line: None)

Raw literal:

```
[[[1, 2], [3]], [[4], [5, 6]], [[7, 8], [9]]]
```

Actual Java Lists executed:

```
[
  [[1, 2], [3]],
  [[4], [5, 6]],
  [[7, 8], [9]]
]
```

Step 1: Multiply A(2, 2) x B(2, 2)

```
  My-code result: [
    [14, 12],
    [12, null]
  ]
  Expected (strict): Invalid/Discarded (Invalid/Discarded due to missing A element)
```

Step 2: Multiply A(2, 2) x B(2, 2)

```
  My-code result: [
    [206, 112],
    [84, 96]
  ]
  Expected (strict): Invalid/Discarded (Invalid/Discarded due to missing A element)
```

Coverage Summary

Parsed tests found: 38

Simulated tests total: 41

Loops visited: 7

Loops not visited: 1, 2, 3, 4, 5, 6

Appendix: Original Main.java (commented)

```
// //Finished and removed single nested loop (advised by chatGPT to reduce time complexity). Also removed if statement identifed previously.
// //Tested code but not extensively.
// //It was created based on refinement from several attempts (notably obtaining positive changes in Test case 28)
// //And feeding it back into earlier iteration
// //All Programiz challenge test cases have passed
//
// //ALL FORMATTED INDENTED CORRECTLY
// //TERMS Matrix A and Matrix B will be used to discuss the pairing of the first and second matrix in transaction
//
// //ALSO NOTE, in my comments, there is reference to Matrix A, Matrix B, Matrix C
// //In absence of a Matrix C (references are to Matrix A and Matrix B)
// //In presence of Matrix C (references are to Matrix A, Matrix B, Matrix C, Matrix, B, Matrix C........)
//
// //ANY CATCH STATEMENT WHICH REQUIRES A MORE USER FRIENDLY MESSAGE CAN INCLUDE ONE OF THESE (IF REQUIRED)
// /*
// System.out.println("2Matrix("+(counter)+")" + "   row:" + ee +"\thas no content at index["+m+"]");
// System.out.println("Hence can not perform multiplication with Matrix("+(counter+1)+")" + "   row:" + (rowCounterNextMatrix+1)+"\t"
// +  "  index["+z+"]: " + rowInfo.get(z));
//
//
// System.out.println("1Matrix("+(counter)+")" + "   row:" + rowCounterFirstMatrix+"\t" +  "value at index["+z+"]: " + ee.get(posPrevMatrixRowLevel)
// +"  has no available supporting entry in \nMatrix("+(counter+1)+")" + "   row:" + (rowCounterNextMatrix+1)+"\t" +  "\t index["+z+"]: " + " row content:" + rowInfo);
// */
//
//
// /*
// //**********USAGE************************
// BEFORE CONFIGURING TEST CASES REMEMBER THESE CRITERIA AS MINIMUM:
//
// Numbers columms in Matrix A/C  = Number rows in Matrix B
// Jagged arrays supported under validation rules
// Size first row (number columns) in Matrix A/C and number rows in Matrix B will ALWAYS determine matrixMultiplication size
// //*********************************
// */
//
// import java.util.*;
//
// public class Solution
// {
//     static List<List<Integer>> matrix = new ArrayList<>();
//     static List<List <Integer>> nextMatrix;
//
//     public static void main (String [] args)
```

```
//     {
//         List <List<List<Integer>>> testMatrix = new ArrayList<>();
//
//         //TEST CASES
//         //We have to ensure that the test case is a 3d array as oppose to a 2d array to contain all the matrixes
//         //Integer [][][]test = new Integer[][][] {   {{1},{5}},{{9,88}}};
//
//         //Only 1 matrix defined
//         //Integer [][][]test = new Integer[][][] {   {{1,2,3},{4,5,6}}};
//
//         //As per challenge  - Test Case 1
//         //Integer [][][]test = new Integer[][][] {   {{1,2,3},{4,5,6}},  {{7,8},{9,10},{11,12}},  { {13,14}, {15,16} } };
//
//         //Adaptation  Test Case 1
//         //Integer [][][]test = new Integer[][][] {   {{1,2,3,5,6},{4,5,6}},  {{7,8},{9,10},{11,12}},  { {13,14}, {15,16} } };
//
//         //As per challenge  - Test Case 2
//         //Integer [][][]test = new Integer[][][] {   {{1,2,3,4}},{{5},{6},{7},{8}}};
//
//         //As per challenge  - Test Case 3/4  (both same)
//         //Integer [][][]test = new Integer[][][] {   {{1, 2}, {3, 4}}, {{5, 6}, {7, 8}}, {{9, 10}, {11, 12}}  };
//
//         //TEST CASES EXPLORING JAGGED ARRAYS (variants of TEST CASE 28 IN Document)
//         //Integer [][][]test = new Integer[][][] {   {{1,2,3},{4,5,6}},  {{7,8},{10},{11,12}},  { {13,14}, {15,16} } };
//
//         //TEST CASES EXPLORING JAGGED ARRAYS (variants of TEST CASE 28 IN Document)
//         //This relies on the matrixMultiplication initialisation
//         Integer [][][]test = new Integer[][][] {   {{1,2,3},{4,5,6}},  {{7,8,999},{10,11},{11,12}},  { {13,14}, {15,16} } };
//
//         //TEST CASES EXPLORING JAGGED ARRAYS (variants of TEST CASE 28 IN Document)
//         //Integer [][][]test = new Integer[][][] {   {{1,2,3},{4,5}},  {{7,8},{10,11},{11,12}},  { {13,14,17}, {15,16} } };
//
//         //TEST CASES EXPLORING JAGGED ARRAYS (variants of TEST CASE 28 IN Document)
//         //Integer [][][]test = new Integer[][][] {   {{1,2},{4,5,6}},  {{7,8},{10,11},{11,12}},  { {13,14,17}, {15,16,17} } };
//
//         //TEST CASES EXPLORING JAGGED ARRAYS (variants of TEST CASE 28 IN Document)
//         //This will get picked up on area of code I had written towards bottom part of code
//         //Integer [][][]test = new Integer[][][] {   {{},{4,5,6}},  {{7,8,6},{10,14},{11,12}},  { {13}, {15,16} } };
//
//         //HERE!!!!!!!!!!!!!!!!!!!!!! - NEED TO CHECK THIS!!!!
//         //Integer [][][]test = new Integer[][][] {   {{1,2},{3,4},{5,6}},  {{7,8},{10,14},{0}},  { {13}, {15,16} } };
//
//         //With this test case it has not performed jump to new column
//         //Integer [][][]test = new Integer[][][] {   {{1,2,3},{4,5,6}},  {{7,8},{9,10},{11,12}},  { {0}, {16,17},{5}} };
//
```

```
//          //Integer [][][]test = new Integer[][][] {   {{1,2,3},{4,5,6}},  {{7,8},{9,10},{11}},  { {13,14}, {16,17} } };
//
//          //TEST CASES EXPLORING JAGGED ARRAYS (variants of TEST CASE 28 IN Document)
//          //Integer [][][]test = new Integer[][][] {   {{1,2,3},{4,5,6}},  {{7,8},{9,2},{11,12}},  { {13,14}, {16,87},{3,4} } };
//
//          //TEST CASES EXPLORING JAGGED ARRAYS (variants of TEST CASE 28 IN Document)
//          //Integer [][][]test = new Integer[][][] {   {{1,2,3},{4,5,6}},  {{},{10,14},{11,12}},  { {13}, {15,16} } };
//
//          //TEST CASES EXPLORING JAGGED ARRAYS (variants of TEST CASE 28 IN Document)
//          //Integer [][][]test = new Integer[][][] {   {{1,2,3},{4,5,6}},  {{1,7},{25,14},{}},  { {13,15}, {15,16} } };
//
//          //TEST CASES EXPLORING JAGGED ARRAYS (variants of TEST CASE 28 IN Document)
//          //This will be handled validation in later part of code
//          //Integer [][][]test = new Integer[][][] {   {{1,2,3},{}},  {{1,7},{25,14},{4,8}},  { {13,15}, {15,16} } };
//
//          //TEST CASES EXPLORING JAGGED ARRAYS (variants of TEST CASE 28 IN Document
//          //Integer [][][]test = new Integer[][][] {   {{1,2,3},{4,5,6}},  {{1,7},{25,14},{4,8}},  { {}, {15,16} } };
//
//          //TEST CASES EXPLORING JAGGED ARRAYS (variants of TEST CASE 28 IN Document)
//          //Integer [][][]test = new Integer[][][] {   {{1,2,3},{4,5,6}},  {{1,7},{25,14},{4,8}},  { {34,45}, {} } };
//
//          //TEST CASES EXPLORING JAGGED ARRAYS (TEST CASE 28 IN Document exactly)
//          //Integer [][][]test = new Integer[][][] {   {{1,2,3},{4,5,6}},  {{7,8,6},{10,14},{11,12}},  { {13,14}, {15,16} } };
//
//          //TEST CASES EXPLORING JAGGED ARRAYS (TEST CASE 28 IN Document exactly)
//          //But it is error free, so informed end user excess numbers
//          //Integer [][][]test = new Integer[][][] {   {{1,2,3},{4,5,6,9}},  {{7,8},{9,10},{11,12}},  { {13,14}, {15,16} } };
//
//          //TEST CASES EXPLORING JAGGED ARRAYS BUT EXPLORING CHANGES IN MATRIX A (TEST CASE 28 IN Document exactly)
//          //Integer [][][]test = new Integer[][][] {   {{1,2,3},{4}},  {{7,8},{9,10},{11,12}},  { {13,14}, {15,16} } };
//
//          //TEST CASES EXPLORING JAGGED ARRAYS BUT EXPLORING CHANGES IN MATRIX A (TEST CASE 28 IN Document exactly)
//          //But it is error free, so informed end user excess numbers
//          //Integer [][][]test = new Integer[][][] {   {{1,2,3},{4,7,2,6}},  {{7,8},{9,10},{11,12}},  { {13,14}, {15,16} } };
//
//          //TEST CASES EXPLORING JAGGED ARRAYS (TEST CASE 28 IN Document exactly)
//          //Integer [][][]test = new Integer[][][] {   {{1,2,3},{4,5,6}},  {{},{10,14},{11,12}},  { {13,14}, {15,16} } };
//
//          //Integer [][][]test = new Integer[][][] {   {{1,2,3},{4,5,6}},  {{11,14},{},{19,12}},  { {13,14}, {15,16} } };
//          //Integer [][][]test = new Integer[][][] {   {{1,2,3},{4,5,6}},  {{11,14},{44,55},{19,12}},  { {13,14}, {} } };
//
//          //Integer [][][]test = new Integer[][][] {   {{1,2,3},{4,5,6}},  {{11,14},{44,55},{19,12}},  { {13,14}, {0} } };
//
//          //Integer [][][]test = new Integer[][][] {   {{1,2,3},{4,5,6}},  {{11,14},{44,55},{19,12}},  { {13,14}, {0,5},{0} } };
//
```

```
//         //TEST CASES EXPLORING JAGGED ARRAYS (TEST CASE 28 IN Document exactly)
//         //Integer [][][]test = new Integer[][][] {   {{1,2,3},{4,5,6}},  {{7,8},{10,14},{11,12}},  { {13}, {15,16} } };
//
//         //ADDITIONAL TEST CASES
//
//         //Integer [][][]test = new Integer[][][] {   {{1,2,3},{4,5,6}},  {{11,14},{44,55},{19,12}},  { {13,14}, {0} } };
//
//         //Integer [][][]test = new Integer[][][] {   {{1,2,3},{4,5,6}},  {{7,8},{10,14},{15,12,65,55}},  { {13,99}, {15,16} } };
//
//         //Integer [][][]test = new Integer[][][] {   {{1,2,3},{4,5,6},{7,8,9}},  {{10,11,12},{13,14,15},{16,17,18}},  { {}, {22,23,24}} };
//
//         //Integer [][][]test = new Integer[][][] {   {{1,2,3},{4,5,6}},  {{7,8},{9,10},{11,12}},  { {13,14}, {15,16} },    { {17,18}, {19,20} },{ {17,18}, {19,20} }};
//
//         //Integer [][][]test = new Integer[][][] {   {{1,2,3},{4,5,6}},  {{7,8},{9,10},{11,12}},  { {13,14}, {15,16}},  { {17,18}, {19,20},{99,18}, {19,20}}, { {21,22,2
//
//         //Integer [][][]test = new Integer[][][] {   {{1,2,3},{4,5,6},{7,8,9}},  {{10,11,12},{13,14,15},{16,17,18}},  { {19,20,21}, {22,23,24}} };
//
//         //active
//         //Integer [][][]test = new Integer[][][] {   {{1,2,3},{4,5,6},{7,8,9}},  {{11,12},{13,14,15},{16,17,18}} };
//
//         //Integer [][][]test = new Integer[][][] {   {{},{4,5,6},{7,8,9}},  {{10,11,12},{13,14,15},{16,17,18}} };
//
//
//         for (Integer [][] item: test)
//         {
//             matrix = new ArrayList<>();
//
//             for (Integer [] row: item)
//             {
//                 matrix.add(Arrays.asList(row));
//             }
//             testMatrix.add(matrix);
//         }
//
//         System.out.println("\n\n***FINAL OUTCOME***: \n" + matrixMultiplication(testMatrix));
//     }
//
//     public static List<List<Integer>> matrixMultiplication(List<List<List<Integer>>> matrices)
//     {
//         List<Integer> rowInfo = new ArrayList<>();
//
//         int rowCounterNextMatrix;
//         int seekPosition=0;
//         int numWritesAtIndexWithinMultiplicationMatrix;
//
```

```
//         StringJoiner matrixRowCalculations = new StringJoiner("");
//         StringJoiner excessRowItems=new StringJoiner(",");
//
//         String rowMatrix=null;
//         String currentEntry=null;
//         String prevEntry=null;
//
//         StringJoiner excessRowReport = new StringJoiner("\n");
//         StringJoiner excessColReport = new StringJoiner("\n");
//         StringJoiner [][] matrixMultiplicationCalculations=null;
//         int numWritesToLastIndexOnRowMultiplicationMatrix;
//         boolean hasReachedEndRow=false;
//         boolean hasStartedMultiplication=false;
//         Integer multiplication=0;
//
//         int rowCounterFirstMatrix=0;
//         int rowsNextMatrix=0;
//         int matricesSize=matrices.size();
//
//         int counter=0;
//
//         int numElementsInRow=0;
//         int posPrevMatrixRowLevel=0;
//         int numMatrixes=0;
//
//         Integer [][]matrixMultiplication = null;
//
//         List <List<Integer>> matrixMultiplicationList = new ArrayList<>();
//
//         String currentColumnEntry;
//
//         StringJoiner excessColItems=new StringJoiner(",");
//
//         System.out.println("***ALL MATRIX*******: " + matrices.size());
//
//         do
//         {
//             matrix=matrices.get(numMatrixes);
//
//             System.out.println("\nMatrix" +"("+numMatrixes+"):");
//
//             for (List<Integer> row: matrix)
//             {
//                 System.out.println(row);
//             }
```

```
//             numMatrixes++;
//
//         }while(numMatrixes<matrices.size());
//
//         System.out.println("\n\n\n");
//         numMatrixes=0;
//
//         do
//         {
//             matrix=matrices.get(counter);
//
//             if (counter>0 && hasStartedMultiplication)
//             {
//                 System.out.println("\n***SUMMARY****");
//                 System.out.println("Matrix multiplication: " + "Matrix ("+(counter-2)+") x Matrix ("+(counter-1)+")");
//
//                 if (matrices.size()!=2)
//                 {
//                     System.out.println("ADDED Multiplication Matrix into the chain at index: " + (counter));
//                     matrices.add(counter,matrixMultiplicationList);
//                     hasStartedMultiplication=false;
//                     System.out.println("Matrix("+counter+")");
//
//                     for (Integer []m: matrixMultiplication)
//                     {
//                         System.out.println(Arrays.toString(m));
//
//                         matrixMultiplicationList.add(Arrays.asList(m));
//                     }
//
//                     System.out.println("\n***CALCULATION STEPS*******");
//
//                     for (int q=0; q<matrixMultiplicationCalculations.length; q++)
//                     {
//                         matrixRowCalculations = new StringJoiner("   ");
//
//                         for (int r=0; r<matrixMultiplicationCalculations[0].length; r++)
//                         {
//                             matrixRowCalculations.add(" ["+matrixMultiplicationCalculations[q][r]+"]");
//                         }
//                         System.out.println(matrixRowCalculations);
//                     }
//                     System.out.println("\nIt will perform: " + "Matrix("+counter+")   x  Matrix("+(counter+1)+")");
//
//                     for (List<Integer> w: matrices.get((counter+1)))
```

```
//                      {
//                          System.out.println("\t\t\t\t"+String.valueOf(w));
//                      }
//
//                      matricesSize=matrices.size();
//                      matrix=matrices.get(counter);
//                  }
//              }
//              System.out.println("\n\n------Matrix" +"("+counter+"):");
//              matrix=matrices.get(counter);
//
//              for (List<Integer> row: matrix)
//              {
//                  System.out.println(row);
//              }
//
//              matrixMultiplicationList = new ArrayList<>();
//
//              try
//              {
//                  nextMatrix=matrices.get(counter+1);
//              }
//              catch (IndexOutOfBoundsException e)
//              {
//                  System.out.println("Single matrix only");
//                  matrix=matrices.get(counter);
//                  System.out.println("\n***CALCULATION STEPS*******");
//
//                  for (List<Integer> row: matrix)
//                  {
//                      System.out.println(row);
//                  }
//                  System.exit(0);
//              }
//
//              System.out.println("^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^");
//
//              matrixMultiplicationCalculations=new StringJoiner[matrix.size()][nextMatrix.get(0).size()];
//
//              System.out.println("Size of Matrix multiplication row: " + matrix.size());
//              System.out.println("Size of Matrix multiplication columns: " + nextMatrix.get(0).size());
//
//              System.out.println("*******Customising Storage Grid for Multiplication Matrix**********["+matrix.size()+"]["+nextMatrix.get(0).size()+"]***************");
//
//              for (int r=0; r<matrix.size(); r++)
```

```
//             {
//                 for (int q=0; q<nextMatrix.get(0).size(); q++)
//                 {
//                     System.out.println("Configuring for index: " + r + "," + q);
//                     matrixMultiplicationCalculations[r][q] = new StringJoiner("+");
//                 }
//             }
//
//             for (List<Integer> ee: matrix)
//             {
//                 rowMatrix = String.valueOf(ee);
//                 System.out.println("THIS IS ROW COUNTER MATRIX("+counter+"): " + rowCounterFirstMatrix);
//                 System.out.println("----------------------------------------THIS IS ROW DATA MATRIX("+counter+"): " + ee);
//
//                 numWritesToLastIndexOnRowMultiplicationMatrix=0;
//                 hasReachedEndRow=false;
//
//                 for (int k=0; k<ee.size(); k++)
//                 {
//                     if(counter!=matricesSize-1)
//                     {
//                         rowCounterNextMatrix=0;
//
//                         if (!hasStartedMultiplication)
//                         {
//                             System.out.println("**************************CONFIGURING SIZE FOR BLANK MULTIPLICATION MATRIX*****************************************"
//                             matrixMultiplication = new Integer[matrix.size()][nextMatrix.get(0).size()];
//
//                             System.out.println("Multiplication matrix (W=" + matrix.size()+")  x  (H=" + nextMatrix.get(0).size()+")"
//                             + " configured to store Matrix " + counter + " x  Matrix " + (counter+1));
//
//                             try
//                             {
//                                 for (int a = 0; a < matrix.size(); a++)
//                                 {
//                                     for (int b = 0; b < nextMatrix.get(0).size(); b++)
//                                     {
//                                         matrixMultiplication[a][b] = 0;
//                                     }
//                                 }
//                             }
//
//                             catch (ArrayIndexOutOfBoundsException e)
//                             {
//                                 System.out.println("\n33The current dimension of matrixMultiplication: "
```

```
//                              + "["+matrix.size()+"]"+ "["+nextMatrix.get(0).size()+"]");
//                          }
//                          hasStartedMultiplication=true;
//                      }
//
//                      if (nextMatrix.get(0).size()==0)
//                      {
//                          System.out.println("8The current dimension of matrixMultiplication: "
//                          + "["+matrix.size()+"]"+"["+nextMatrix.get(0).size()+"]");
//                          System.out.println("2It EXPECTS: " + nextMatrix.get(0).size() + " elements per row");
//                          System.out.println("Hence multiplication matrix has no width");
//                          System.out.println("matrix" + "("+(counter+1)+")" + " row:" + rowCounterNextMatrix +  " "
//                          + nextMatrix.get(0) + " size: " + nextMatrix.get(0).size());
//                          System.out.println(matrices);
//
```