Matrix Multiplication — Final Report (v3.1)

Generated: 2025-11-12 09:09:27Z

This document contains simulated execution of 48 tests (39 parsed from source + 10 extras). Results show executed System.out.println outputs per test (with source line numbers) and comparison of 'my-code' vs 'expected' results.

Annotated Original Code (inline comments)

```
//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: prints diagnostic or result messages to console
System.out.println("Hence can not perform multiplication with Matrix("+(counter+1)+")" + "   row:" + (rowCounterNextMatrix+1)+"\t"
    // ANNOTATION: prints diagnostic or result messages to console
+  "  index["+z+"]: " + rowInfo.get(z));


System.out.println("1Matrix("+(counter)+")" + "   row:" + rowCounterFirstMatrix+"\t" +  "value at index["+z+"]: " + ee.get(posPrevMatrixRowLevel)
    // ANNOTATION: prints diagnostic or result messages to console
+"  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,
```

```
    //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));
// ANNOTATION: prints diagnostic or result messages to console
}

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: prints diagnostic or result messages to console

    do
    {
        matrix=matrices.get(numMatrixes);

        System.out.println("\nMatrix" +"("+numMatrixes+"):");
// ANNOTATION: prints diagnostic or result messages to console

        for (List<Integer> row: matrix)
        {
            System.out.println(row);
// ANNOTATION: prints diagnostic or result messages to console
        }
        numMatrixes++;

    }while(numMatrixes<matrices.size());

    System.out.println("\n\n\n");
// ANNOTATION: prints diagnostic or result messages to console
    numMatrixes=0;

    do
    {
        matrix=matrices.get(counter);

        if (counter>0 && hasStartedMultiplication)
        {
            System.out.println("\n***SUMMARY****");
// ANNOTATION: prints diagnostic or result messages to console
            System.out.println("Matrix multiplication: " + "Matrix ("+(counter-2)+") x Matrix ("+(counter-1)+")");
// ANNOTATION: prints diagnostic or result messages to console

            if (matrices.size()!=2)
            {
                System.out.println("ADDED Multiplication Matrix into the chain at index: " + (counter));
// ANNOTATION: prints diagnostic or result messages to console
                matrices.add(counter,matrixMultiplicationList);
                hasStartedMultiplication=false;
                System.out.println("Matrix("+counter+")");
// ANNOTATION: prints diagnostic or result messages to console

                for (Integer []m: matrixMultiplication)
```

```
                {
                    System.out.println(Arrays.toString(m));
// ANNOTATION: prints diagnostic or result messages to console

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

                System.out.println("\n***CALCULATION STEPS*******");
// ANNOTATION: prints diagnostic or result messages to console

                for (int q=0; q<matrixMultiplicationCalculations.length; q++)
// ANNOTATION: iterates indices; part of matrix traversal/accumulation
                {
                    matrixRowCalculations = new StringJoiner("   ");

                    for (int r=0; r<matrixMultiplicationCalculations[0].length; r++)
// ANNOTATION: iterates indices; part of matrix traversal/accumulation
                    {
                        matrixRowCalculations.add(" ["+matrixMultiplicationCalculations[q][r]+"]");
                    }
                    System.out.println(matrixRowCalculations);
// ANNOTATION: prints diagnostic or result messages to console
                }
                System.out.println("\nIt will perform: " + "Matrix("+counter+")   x  Matrix("+(counter+1)+")");
// ANNOTATION: prints diagnostic or result messages to console

                for (List<Integer> w: matrices.get((counter+1)))
                {
                    System.out.println("\t\t\t\t"+String.valueOf(w));
// ANNOTATION: prints diagnostic or result messages to console
                }

                matricesSize=matrices.size();
                matrix=matrices.get(counter);
            }
        }
        System.out.println("\n\n------Matrix" +"("+counter+"):");
// ANNOTATION: prints diagnostic or result messages to console
        matrix=matrices.get(counter);

        for (List<Integer> row: matrix)
        {
            System.out.println(row);
// ANNOTATION: prints diagnostic or result messages to console
        }

        matrixMultiplicationList = new ArrayList<>();

        try
        {
            nextMatrix=matrices.get(counter+1);
        }
        catch (IndexOutOfBoundsException e)
        {
```

```
            System.out.println("Single matrix only");
// ANNOTATION: prints diagnostic or
```

All Test Case Simulations (full)

Test #1 — source: source_line_49 (source line 49)

Number of matrices: 2

System.out.println outputs (executed for this test):

```
[L16] 2Matrix(
[L17] Hence can not perform multiplication with Matrix(
[L21] 1Matrix(
[L168] \n\n***FINAL OUTCOME***: \n
[L212] ***ALL MATRIX*******:
[L237] \n***SUMMARY****
[L238] Matrix multiplication:
[L242] ADDED Multiplication Matrix into the chain at index:
[L245] Matrix(
[L254] \n***CALCULATION STEPS*******
[L293] Single matrix only
[L295] \n***CALCULATION STEPS*******
[L308] Size of Matrix multiplication row:
[L309] Size of Matrix multiplication columns:
[L325] THIS IS ROW COUNTER MATRIX(
[L326] ----------------------------------------THIS IS ROW DATA MATRIX(
[L473] 1Accessing matrix(
[L497] 2Accessing matrix(
[L513] 2Matrix(
[L514] Hence can not perform multiplication with Matrix(
[L521] 1Matrix(
[L540] 1Matrix(
[L580] Processed matrix(
[L600] VERIFICATION SUCCESSFUL(MATRIX(
[L649] \n***CALCULATION STEPS*******
```

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: source_line_52 (source line 52)

Number of matrices: 1

System.out.println outputs (executed for this test):

```
[L212] ***ALL MATRIX*******:
[L293] Single matrix only
No multiplication steps executed for this test.
```

Test #3 — source: source_line_55 (source line 55)

Number of matrices: 3

System.out.println outputs (executed for this test):

```
[L16] 2Matrix(
[L17] Hence can not perform multiplication with Matrix(
[L21] 1Matrix(
[L168] \n\n***FINAL OUTCOME***: \n
[L212] ***ALL MATRIX*******:
[L237] \n***SUMMARY****
[L238] Matrix multiplication:
[L242] ADDED Multiplication Matrix into the chain at index:
[L245] Matrix(
[L254] \n***CALCULATION STEPS*******
[L293] Single matrix only
[L295] \n***CALCULATION STEPS*******
[L308] Size of Matrix multiplication row:
[L309] Size of Matrix multiplication columns:
[L325] THIS IS ROW COUNTER MATRIX(
[L326] ----------------------------------------THIS IS ROW DATA MATRIX(
[L473] 1Accessing matrix(
[L497] 2Accessing matrix(
[L513] 2Matrix(
[L514] Hence can not perform multiplication with Matrix(
[L521] 1Matrix(
[L540] 1Matrix(
[L580] Processed matrix(
[L600] VERIFICATION SUCCESSFUL(MATRIX(
[L649] \n***CALCULATION STEPS*******
```

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: source_line_58 (source line 58)

Number of matrices: 3

System.out.println outputs (executed for this test):

```
[L16] 2Matrix(
[L17] Hence can not perform multiplication with Matrix(
[L21] 1Matrix(
[L168] \n\n***FINAL OUTCOME***: \n
[L212] ***ALL MATRIX*******:
[L237] \n***SUMMARY****
[L238] Matrix multiplication:
[L242] ADDED Multiplication Matrix into the chain at index:
[L245] Matrix(
[L254] \n***CALCULATION STEPS*******
[L293] Single matrix only
[L295] \n***CALCULATION STEPS*******
[L308] Size of Matrix multiplication row:
[L309] Size of Matrix multiplication columns:
[L325] THIS IS ROW COUNTER MATRIX(
[L326] ----------------------------------------THIS IS ROW DATA MATRIX(
[L473] 1Accessing matrix(
[L497] 2Accessing matrix(
[L513] 2Matrix(
[L514] Hence can not perform multiplication with Matrix(
[L521] 1Matrix(
[L540] 1Matrix(
[L580] Processed matrix(
[L600] VERIFICATION SUCCESSFUL(MATRIX(
[L649] \n***CALCULATION STEPS*******
```

Step 1: Multiply A(2, 5) x B(3, 2)

```
  My-code result:
[
  [ 58, 64 ],
  [ 139, 154 ]
]
  Expected (strict) error: Incompatible dims colsA=5 rowsB=3
  Possible reason for difference: Possible reason: jagged-row skipping or missing entries treated differently by your logic (skipped) vs strict zero-fill.
```

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: source_line_61 (source line 61)

Number of matrices: 2

System.out.println outputs (executed for this test):

```
[L16] 2Matrix(
[L17] Hence can not perform multiplication with Matrix(
[L21] 1Matrix(
[L168] \n\n***FINAL OUTCOME***: \n
[L212] ***ALL MATRIX*******:
[L237] \n***SUMMARY****
[L238] Matrix multiplication:
[L242] ADDED Multiplication Matrix into the chain at index:
[L245] Matrix(
[L254] \n***CALCULATION STEPS*******
[L293] Single matrix only
[L295] \n***CALCULATION STEPS*******
[L308] Size of Matrix multiplication row:
[L309] Size of Matrix multiplication columns:
[L325] THIS IS ROW COUNTER MATRIX(
[L326] ----------------------------------------THIS IS ROW DATA MATRIX(
[L473] 1Accessing matrix(
[L497] 2Accessing matrix(
[L513] 2Matrix(
[L514] Hence can not perform multiplication with Matrix(
[L521] 1Matrix(
[L540] 1Matrix(
[L580] Processed matrix(
[L600] VERIFICATION SUCCESSFUL(MATRIX(
[L649] \n***CALCULATION STEPS*******
```

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

```
  My-code result:
[
  [ 70 ]
]
  Expected (strict):
[
  [ 70 ]
]
```

Test #6 — source: source_line_64 (source line 64)

Number of matrices: 3

System.out.println outputs (executed for this test):

```
[L16] 2Matrix(
[L17] Hence can not perform multiplication with Matrix(
[L21] 1Matrix(
[L168] \n\n***FINAL OUTCOME***: \n
[L212] ***ALL MATRIX*******:
[L237] \n***SUMMARY****
[L238] Matrix multiplication:
[L242] ADDED Multiplication Matrix into the chain at index:
[L245] Matrix(
[L254] \n***CALCULATION STEPS*******
[L293] Single matrix only
[L295] \n***CALCULATION STEPS*******
[L308] Size of Matrix multiplication row:
[L309] Size of Matrix multiplication columns:
[L325] THIS IS ROW COUNTER MATRIX(
[L326] ----------------------------------------THIS IS ROW DATA MATRIX(
[L473] 1Accessing matrix(
[L497] 2Accessing matrix(
[L513] 2Matrix(
[L514] Hence can not perform multiplication with Matrix(
[L521] 1Matrix(
[L540] 1Matrix(
[L580] Processed matrix(
[L600] VERIFICATION SUCCESSFUL(MATRIX(
[L649] \n***CALCULATION STEPS*******
```

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: source_line_67 (source line 67)

Number of matrices: 3

System.out.println outputs (executed for this test):

```
[L16] 2Matrix(
[L17] Hence can not perform multiplication with Matrix(
[L21] 1Matrix(
[L168] \n\n***FINAL OUTCOME***: \n
[L212] ***ALL MATRIX*******:
[L237] \n***SUMMARY****
[L238] Matrix multiplication:
[L242] ADDED Multiplication Matrix into the chain at index:
[L245] Matrix(
[L254] \n***CALCULATION STEPS*******
[L293] Single matrix only
[L295] \n***CALCULATION STEPS*******
[L308] Size of Matrix multiplication row:
[L309] Size of Matrix multiplication columns:
[L325] THIS IS ROW COUNTER MATRIX(
[L326] ----------------------------------------THIS IS ROW DATA MATRIX(
[L473] 1Accessing matrix(
[L497] 2Accessing matrix(
[L513] 2Matrix(
[L514] Hence can not perform multiplication with Matrix(
[L521] 1Matrix(
[L540] 1Matrix(
[L580] Processed matrix(
[L600] VERIFICATION SUCCESSFUL(MATRIX(
[L649] \n***CALCULATION STEPS*******
```

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

```
  My-code result:
[
  [ 60, 44 ],
  [ 144, 104 ]
]
   - NOTE: Skipped A[0][1] or B[1][1] missing
   - NOTE: Skipped A[1][1] or B[1][1] missing
  Expected (strict):
[
  [ 60, 44 ],
  [ 144, 104 ]
]
```

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: source_line_71 (source line 71)

Number of matrices: 3

System.out.println outputs (executed for this test):

```
[L16] 2Matrix(
[L17] Hence can not perform multiplication with Matrix(
[L21] 1Matrix(
[L168] \n\n***FINAL OUTCOME***: \n
[L212] ***ALL MATRIX*******:
[L237] \n***SUMMARY****
[L238] Matrix multiplication:
[L242] ADDED Multiplication Matrix into the chain at index:
[L245] Matrix(
[L254] \n***CALCULATION STEPS*******
[L293] Single matrix only
[L295] \n***CALCULATION STEPS*******
[L308] Size of Matrix multiplication row:
[L309] Size of Matrix multiplication columns:
[L325] THIS IS ROW COUNTER MATRIX(
[L326] ----------------------------------------THIS IS ROW DATA MATRIX(
[L473] 1Accessing matrix(
[L497] 2Accessing matrix(
[L513] 2Matrix(
[L514] Hence can not perform multiplication with Matrix(
[L521] 1Matrix(
[L540] 1Matrix(
[L580] Processed matrix(
[L600] VERIFICATION SUCCESSFUL(MATRIX(
[L649] \n***CALCULATION STEPS*******
```

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

```
  My-code result:
[
  [ 60, 66, 999 ],
  [ 144, 159, 3996 ]
]
   - NOTE: Skipped A[0][1] or B[1][2] missing
   - NOTE: Skipped A[0][2] or B[2][2] missing
   - NOTE: Skipped A[1][1] or B[1][2] missing
   - NOTE: Skipped A[1][2] or B[2][2] missing
  Expected (strict):
[
  [ 60, 66, 999 ],
  [ 144, 159, 3996 ]
]
```

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

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

Possible reason for difference: Possible reason: jagged-row skipping or missing entries treated differently by your logic (skipped) vs strict zero-fill.

Test #9 — source: source_line_74 (source line 74)

Number of matrices: 3

System.out.println outputs (executed for this test):

```
[L16] 2Matrix(
[L17] Hence can not perform multiplication with Matrix(
[L21] 1Matrix(
[L168] \n\n***FINAL OUTCOME***: \n
[L212] ***ALL MATRIX*******:
[L237] \n***SUMMARY****
[L238] Matrix multiplication:
[L242] ADDED Multiplication Matrix into the chain at index:
[L245] Matrix(
[L254] \n***CALCULATION STEPS*******
[L293] Single matrix only
[L295] \n***CALCULATION STEPS*******
[L308] Size of Matrix multiplication row:
[L309] Size of Matrix multiplication columns:
[L325] THIS IS ROW COUNTER MATRIX(
[L326] ----------------------------------------THIS IS ROW DATA MATRIX(
[L473] 1Accessing matrix(
[L497] 2Accessing matrix(
[L513] 2Matrix(
[L514] Hence can not perform multiplication with Matrix(
[L521] 1Matrix(
[L540] 1Matrix(
[L580] Processed matrix(
[L600] VERIFICATION SUCCESSFUL(MATRIX(
[L649] \n***CALCULATION STEPS*******
```

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

```
  My-code result:
[
  [ 60, 66 ],
  [ 78, 87 ]
]
   - NOTE: Skipped A[1][2] or B[2][0] missing
   - NOTE: Skipped A[1][2] or B[2][1] missing
  Expected (strict):
[
  [ 60, 66 ],
  [ 78, 87 ]
]
```

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

```
  My-code result:
[
  [ 1770, 1896, 1020 ],
  [ 2319, 2484, 1326 ]
]
   - NOTE: Skipped A[0][1] or B[1][2] missing
   - NOTE: Skipped A[1][1] or B[1][2] missing
  Expected (strict):
```

```
[
  [ 1770, 1896, 1020 ],
  [ 2319, 2484, 1326 ]
]
```

Test #10 — source: source_line_77 (source line 77)

Number of matrices: 3

System.out.println outputs (executed for this test):

```
[L16] 2Matrix(
[L17] Hence can not perform multiplication with Matrix(
[L21] 1Matrix(
[L168] \n\n***FINAL OUTCOME***: \n
[L212] ***ALL MATRIX*******:
[L237] \n***SUMMARY****
[L238] Matrix multiplication:
[L242] ADDED Multiplication Matrix into the chain at index:
[L245] Matrix(
[L254] \n***CALCULATION STEPS*******
[L293] Single matrix only
[L295] \n***CALCULATION STEPS*******
[L308] Size of Matrix multiplication row:
[L309] Size of Matrix multiplication columns:
[L325] THIS IS ROW COUNTER MATRIX(
[L326] ----------------------------------------THIS IS ROW DATA MATRIX(
[L473] 1Accessing matrix(
[L497] 2Accessing matrix(
[L513] 2Matrix(
[L514] Hence can not perform multiplication with Matrix(
[L521] 1Matrix(
[L540] 1Matrix(
[L580] Processed matrix(
[L600] VERIFICATION SUCCESSFUL(MATRIX(
[L649] \n***CALCULATION STEPS*******
```

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

```
  My-code result:
[
  [ 27, 30 ],
  [ 78, 87 ]
]
  Expected (strict):
[
  [ 27, 30 ],
  [ 144, 159 ]
]
  Possible reason for difference: Possible reason: jagged-row skipping or missing entries treated differently by your logic (skipped) vs strict zero-fill.
```

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

```
  My-code result:
[
  [ 801, 858, 969 ],
  [ 2319, 2484, 2805 ]
]
  Expected (strict):
[
  [ 801, 858, 969 ],
  [ 2319, 2484, 2805 ]
```

]

Test #11 — source: source_line_81 (source line 81)

Number of matrices: 3

System.out.println outputs (executed for this test):

```
[L16] 2Matrix(
[L17] Hence can not perform multiplication with Matrix(
[L21] 1Matrix(
[L168] \n\n***FINAL OUTCOME***: \n
[L212] ***ALL MATRIX*******:
[L237] \n***SUMMARY****
[L238] Matrix multiplication:
[L242] ADDED Multiplication Matrix into the chain at index:
[L245] Matrix(
[L254] \n***CALCULATION STEPS*******
[L293] Single matrix only
[L295] \n***CALCULATION STEPS*******
[L308] Size of Matrix multiplication row:
[L309] Size of Matrix multiplication columns:
[L325] THIS IS ROW COUNTER MATRIX(
[L326] ----------------------------------------THIS IS ROW DATA MATRIX(
[L473] 1Accessing matrix(
[L497] 2Accessing matrix(
[L513] 2Matrix(
[L514] Hence can not perform multiplication with Matrix(
[L521] 1Matrix(
[L540] 1Matrix(
[L580] Processed matrix(
[L600] VERIFICATION SUCCESSFUL(MATRIX(
[L649] \n***CALCULATION STEPS*******
```

Step 1: Multiply A(2, 0) x B(3, 3)

```
  My-code result:
[
  [ 0, 0, 0 ],
  [ 0, 0, 0 ]
]
  Expected (strict):
[
  [ 0, 0, 0 ],
  [ 144, 174, 24 ]
]
  Possible reason for difference: Possible reason: jagged-row skipping or missing entries treated differently by your logic (skipped) vs strict zero-fill.
```

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

```
  My-code result:
[
  [ 0 ],
  [ 0 ]
]
  Expected (strict) error: Incompatible dims colsA=3 rowsB=2
  Possible reason for difference: Possible reason: jagged-row skipping or missing entries treated differently by your logic (skipped) vs strict zero-fill.
```

Test #12 — source: source_line_84 (source line 84)

Number of matrices: 3

System.out.println outputs (executed for this test):

```
[L16] 2Matrix(
[L17] Hence can not perform multiplication with Matrix(
[L21] 1Matrix(
[L168] \n\n***FINAL OUTCOME***: \n
[L212] ***ALL MATRIX*******:
[L237] \n***SUMMARY****
[L238] Matrix multiplication:
[L242] ADDED Multiplication Matrix into the chain at index:
[L245] Matrix(
[L254] \n***CALCULATION STEPS*******
[L293] Single matrix only
[L295] \n***CALCULATION STEPS*******
[L308] Size of Matrix multiplication row:
[L309] Size of Matrix multiplication columns:
[L325] THIS IS ROW COUNTER MATRIX(
[L326] ----------------------------------------THIS IS ROW DATA MATRIX(
[L473] 1Accessing matrix(
[L497] 2Accessing matrix(
[L513] 2Matrix(
[L514] Hence can not perform multiplication with Matrix(
[L521] 1Matrix(
[L540] 1Matrix(
[L580] Processed matrix(
[L600] VERIFICATION SUCCESSFUL(MATRIX(
[L649] \n***CALCULATION STEPS*******
```

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

```
  My-code result:
[
  [ 27, 36 ],
  [ 61, 80 ],
  [ 95, 124 ]
]
  Expected (strict) error: Incompatible dims colsA=2 rowsB=3
  Possible reason for difference: Possible reason: jagged-row skipping or missing entries treated differently by your logic (skipped) vs strict zero-fill.
```

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

```
  My-code result:
[
  [ 891 ],
  [ 1993 ],
  [ 3095 ]
]
  Expected (strict):
[
  [ 891, 576 ],
  [ 1993, 1280 ],
  [ 3095, 1984 ]
]
```

Possible reason for difference: Possible reason: jagged-row skipping or missing entries treated differently by your logic (skipped) vs strict zero-fill.

Test #13 — source: source_line_87 (source line 87)

Number of matrices: 3

System.out.println outputs (executed for this test):

```
[L16] 2Matrix(
[L17] Hence can not perform multiplication with Matrix(
[L21] 1Matrix(
[L168] \n\n***FINAL OUTCOME***: \n
[L212] ***ALL MATRIX*******:
[L237] \n***SUMMARY****
[L238] Matrix multiplication:
[L242] ADDED Multiplication Matrix into the chain at index:
[L245] Matrix(
[L254] \n***CALCULATION STEPS*******
[L293] Single matrix only
[L295] \n***CALCULATION STEPS*******
[L308] Size of Matrix multiplication row:
[L309] Size of Matrix multiplication columns:
[L325] THIS IS ROW COUNTER MATRIX(
[L326] ----------------------------------------THIS IS ROW DATA MATRIX(
[L473] 1Accessing matrix(
[L497] 2Accessing matrix(
[L513] 2Matrix(
[L514] Hence can not perform multiplication with Matrix(
[L521] 1Matrix(
[L540] 1Matrix(
[L580] Processed matrix(
[L600] VERIFICATION SUCCESSFUL(MATRIX(
[L649] \n***CALCULATION STEPS*******
```

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, 1)

```
  My-code result:
[
  [ 1024 ],
  [ 2464 ]
]
  Expected (strict) error: Incompatible dims colsA=2 rowsB=3
  Possible reason for difference: Possible reason: jagged-row skipping or missing entries treated differently by your logic (skipped) vs strict zero-fill.
```

Test #14 — source: source_line_89 (source line 89)

Number of matrices: 3

System.out.println outputs (executed for this test):

```
[L16] 2Matrix(
[L17] Hence can not perform multiplication with Matrix(
[L21] 1Matrix(
[L168] \n\n***FINAL OUTCOME***: \n
[L212] ***ALL MATRIX*******:
[L237] \n***SUMMARY****
[L238] Matrix multiplication:
[L242] ADDED Multiplication Matrix into the chain at index:
[L245] Matrix(
[L254] \n***CALCULATION STEPS*******
[L293] Single matrix only
[L295] \n***CALCULATION STEPS*******
[L308] Size of Matrix multiplication row:
[L309] Size of Matrix multiplication columns:
[L325] THIS IS ROW COUNTER MATRIX(
[L326] ----------------------------------------THIS IS ROW DATA MATRIX(
[L473] 1Accessing matrix(
[L497] 2Accessing matrix(
[L513] 2Matrix(
[L514] Hence can not perform multiplication with Matrix(
[L521] 1Matrix(
[L540] 1Matrix(
[L580] Processed matrix(
[L600] VERIFICATION SUCCESSFUL(MATRIX(
[L649] \n***CALCULATION STEPS*******
```

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

```
  My-code result:
[
  [ 58, 28 ],
  [ 139, 82 ]
]
   - NOTE: Skipped A[0][2] or B[2][1] missing
   - NOTE: Skipped A[1][2] or B[2][1] missing
  Expected (strict):
[
  [ 58, 28 ],
  [ 139, 82 ]
]
```

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: source_line_92 (source line 92)

Number of matrices: 3

System.out.println outputs (executed for this test):

```
[L16] 2Matrix(
[L17] Hence can not perform multiplication with Matrix(
[L21] 1Matrix(
[L168] \n\n***FINAL OUTCOME***: \n
[L212] ***ALL MATRIX*******:
[L237] \n***SUMMARY****
[L238] Matrix multiplication:
[L242] ADDED Multiplication Matrix into the chain at index:
[L245] Matrix(
[L254] \n***CALCULATION STEPS*******
[L293] Single matrix only
[L295] \n***CALCULATION STEPS*******
[L308] Size of Matrix multiplication row:
[L309] Size of Matrix multiplication columns:
[L325] THIS IS ROW COUNTER MATRIX(
[L326] ----------------------------------------THIS IS ROW DATA MATRIX(
[L473] 1Accessing matrix(
[L497] 2Accessing matrix(
[L513] 2Matrix(
[L514] Hence can not perform multiplication with Matrix(
[L521] 1Matrix(
[L540] 1Matrix(
[L580] Processed matrix(
[L600] VERIFICATION SUCCESSFUL(MATRIX(
[L649] \n***CALCULATION STEPS*******
```

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) error: Incompatible dims colsA=2 rowsB=3
  Possible reason for difference: Possible reason: jagged-row skipping or missing entries treated differently by your logic (skipped) vs strict zero-fill.
```

Test #16 — source: source_line_95 (source line 95)

Number of matrices: 3

System.out.println outputs (executed for this test):

```
[L16] 2Matrix(
[L17] Hence can not perform multiplication with Matrix(
[L21] 1Matrix(
[L168] \n\n***FINAL OUTCOME***: \n
[L212] ***ALL MATRIX*******:
[L237] \n***SUMMARY****
[L238] Matrix multiplication:
[L242] ADDED Multiplication Matrix into the chain at index:
[L245] Matrix(
[L254] \n***CALCULATION STEPS*******
[L293] Single matrix only
[L295] \n***CALCULATION STEPS*******
[L308] Size of Matrix multiplication row:
[L309] Size of Matrix multiplication columns:
[L325] THIS IS ROW COUNTER MATRIX(
[L326] ----------------------------------------THIS IS ROW DATA MATRIX(
[L473] 1Accessing matrix(
[L497] 2Accessing matrix(
[L513] 2Matrix(
[L514] Hence can not perform multiplication with Matrix(
[L521] 1Matrix(
[L540] 1Matrix(
[L580] Processed matrix(
[L600] VERIFICATION SUCCESSFUL(MATRIX(
[L649] \n***CALCULATION STEPS*******
```

Step 1: Multiply A(2, 3) x B(3, 0)

```
  My-code result:
[
  [  ],
  [  ]
]
  Expected (strict):
[
  [ 53, 64 ],
  [ 116, 142 ]
]
  Possible reason for difference: Possible reason: jagged-row skipping or missing entries treated differently by your logic (skipped) vs strict zero-fill.
```

Step 2: Multiply A(2, 0) x B(2, 1)

```
  My-code result:
[
  [ 0 ],
  [ 0 ]
]
  Expected (strict) error: Incompatible dims colsA=0 rowsB=2
  Possible reason for difference: Possible reason: jagged-row skipping or missing entries treated differently by your logic (skipped) vs strict zero-fill.
```

Test #17 — source: source_line_98 (source line 98)

Number of matrices: 3

System.out.println outputs (executed for this test):

```
[L16] 2Matrix(
[L17] Hence can not perform multiplication with Matrix(
[L21] 1Matrix(
[L168] \n\n***FINAL OUTCOME***: \n
[L212] ***ALL MATRIX*******:
[L237] \n***SUMMARY****
[L238] Matrix multiplication:
[L242] ADDED Multiplication Matrix into the chain at index:
[L245] Matrix(
[L254] \n***CALCULATION STEPS*******
[L293] Single matrix only
[L295] \n***CALCULATION STEPS*******
[L308] Size of Matrix multiplication row:
[L309] Size of Matrix multiplication columns:
[L325] THIS IS ROW COUNTER MATRIX(
[L326] ----------------------------------------THIS IS ROW DATA MATRIX(
[L473] 1Accessing matrix(
[L497] 2Accessing matrix(
[L513] 2Matrix(
[L514] Hence can not perform multiplication with Matrix(
[L521] 1Matrix(
[L540] 1Matrix(
[L580] Processed matrix(
[L600] VERIFICATION SUCCESSFUL(MATRIX(
[L649] \n***CALCULATION STEPS*******
```

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

```
  My-code result:
[
  [ 51, 35 ],
  [ 129, 98 ]
]
   - NOTE: Skipped A[0][2] or B[2][0] missing
   - NOTE: Skipped A[0][2] or B[2][1] missing
   - NOTE: Skipped A[1][2] or B[2][0] missing
   - NOTE: Skipped A[1][2] or B[2][1] missing
  Expected (strict):
[
  [ 51, 35 ],
  [ 129, 98 ]
]
```

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: source_line_102 (source line 102)

Number of matrices: 3

System.out.println outputs (executed for this test):

```
[L16] 2Matrix(
[L17] Hence can not perform multiplication with Matrix(
[L21] 1Matrix(
[L168] \n\n***FINAL OUTCOME***: \n
[L212] ***ALL MATRIX*******:
[L237] \n***SUMMARY****
[L238] Matrix multiplication:
[L242] ADDED Multiplication Matrix into the chain at index:
[L245] Matrix(
[L254] \n***CALCULATION STEPS*******
[L293] Single matrix only
[L295] \n***CALCULATION STEPS*******
[L308] Size of Matrix multiplication row:
[L309] Size of Matrix multiplication columns:
[L325] THIS IS ROW COUNTER MATRIX(
[L326] ----------------------------------------THIS IS ROW DATA MATRIX(
[L473] 1Accessing matrix(
[L497] 2Accessing matrix(
[L513] 2Matrix(
[L514] Hence can not perform multiplication with Matrix(
[L521] 1Matrix(
[L540] 1Matrix(
[L580] Processed matrix(
[L600] VERIFICATION SUCCESSFUL(MATRIX(
[L649] \n***CALCULATION STEPS*******
```

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

```
  My-code result:
[
  [ 63, 59 ],
  [ 0, 0 ]
]
   - NOTE: Skipped A[1][0] or B[0][0] missing
   - NOTE: Skipped A[1][1] or B[1][0] missing
   - NOTE: Skipped A[1][2] or B[2][0] missing
   - NOTE: Skipped A[1][0] or B[0][1] missing
   - NOTE: Skipped A[1][1] or B[1][1] missing
   - NOTE: Skipped A[1][2] or B[2][1] missing
  Expected (strict):
[
  [ 63, 59 ],
  [ 0, 0 ]
]
```

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

```
  My-code result:
[
  [ 1704, 1889 ],
  [ 0, 0 ]
```

```
]
  Expected (strict):
[
  [ 1704, 1889 ],
  [ 0, 0 ]
]
```

Test #19 — source: source_line_105 (source line 105)

Number of matrices: 3

System.out.println outputs (executed for this test):

```
[L16] 2Matrix(
[L17] Hence can not perform multiplication with Matrix(
[L21] 1Matrix(
[L168] \n\n***FINAL OUTCOME***: \n
[L212] ***ALL MATRIX*******:
[L237] \n***SUMMARY****
[L238] Matrix multiplication:
[L242] ADDED Multiplication Matrix into the chain at index:
[L245] Matrix(
[L254] \n***CALCULATION STEPS*******
[L293] Single matrix only
[L295] \n***CALCULATION STEPS*******
[L308] Size of Matrix multiplication row:
[L309] Size of Matrix multiplication columns:
[L325] THIS IS ROW COUNTER MATRIX(
[L326] ----------------------------------------THIS IS ROW DATA MATRIX(
[L473] 1Accessing matrix(
[L497] 2Accessing matrix(
[L513] 2Matrix(
[L514] Hence can not perform multiplication with Matrix(
[L521] 1Matrix(
[L540] 1Matrix(
[L580] Processed matrix(
[L600] VERIFICATION SUCCESSFUL(MATRIX(
[L649] \n***CALCULATION STEPS*******
```

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, 0)

```
  My-code result:
[
  [  ],
  [  ]
]
  Expected (strict):
[
  [ 885, 944 ],
  [ 2190, 2336 ]
]
```

Possible reason for difference: Possible reason: jagged-row skipping or missing entries treated differently by your logic (skipped) vs strict zero-fill.

Test #20 — source: source_line_108 (source line 108)

Number of matrices: 3

System.out.println outputs (executed for this test):

```
[L16] 2Matrix(
[L17] Hence can not perform multiplication with Matrix(
[L21] 1Matrix(
[L168] \n\n***FINAL OUTCOME***: \n
[L212] ***ALL MATRIX*******:
[L237] \n***SUMMARY****
[L238] Matrix multiplication:
[L242] ADDED Multiplication Matrix into the chain at index:
[L245] Matrix(
[L254] \n***CALCULATION STEPS*******
[L293] Single matrix only
[L295] \n***CALCULATION STEPS*******
[L308] Size of Matrix multiplication row:
[L309] Size of Matrix multiplication columns:
[L325] THIS IS ROW COUNTER MATRIX(
[L326] ----------------------------------------THIS IS ROW DATA MATRIX(
[L473] 1Accessing matrix(
[L497] 2Accessing matrix(
[L513] 2Matrix(
[L514] Hence can not perform multiplication with Matrix(
[L521] 1Matrix(
[L540] 1Matrix(
[L580] Processed matrix(
[L600] VERIFICATION SUCCESSFUL(MATRIX(
[L649] \n***CALCULATION STEPS*******
```

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 ]
]
   - NOTE: Skipped A[0][1] or B[1][0] missing
   - NOTE: Skipped A[0][1] or B[1][1] missing
   - NOTE: Skipped A[1][1] or B[1][0] missing
   - NOTE: Skipped A[1][1] or B[1][1] missing
  Expected (strict):
```

```
[
  [ 2142, 2835 ],
  [ 5202, 6885 ]
]
```

Test #21 — source: source_line_111 (source line 111)

Number of matrices: 3

System.out.println outputs (executed for this test):

```
[L16] 2Matrix(
[L17] Hence can not perform multiplication with Matrix(
[L21] 1Matrix(
[L168] \n\n***FINAL OUTCOME***: \n
[L212] ***ALL MATRIX*******:
[L237] \n***SUMMARY****
[L238] Matrix multiplication:
[L242] ADDED Multiplication Matrix into the chain at index:
[L245] Matrix(
[L254] \n***CALCULATION STEPS*******
[L293] Single matrix only
[L295] \n***CALCULATION STEPS*******
[L308] Size of Matrix multiplication row:
[L309] Size of Matrix multiplication columns:
[L325] THIS IS ROW COUNTER MATRIX(
[L326] ----------------------------------------THIS IS ROW DATA MATRIX(
[L473] 1Accessing matrix(
[L497] 2Accessing matrix(
[L513] 2Matrix(
[L514] Hence can not perform multiplication with Matrix(
[L521] 1Matrix(
[L540] 1Matrix(
[L580] Processed matrix(
[L600] VERIFICATION SUCCESSFUL(MATRIX(
[L649] \n***CALCULATION STEPS*******
```

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

```
  My-code result:
[
  [ 60, 72, 6 ],
  [ 144, 174, 24 ]
]
   - NOTE: Skipped A[0][1] or B[1][2] missing
   - NOTE: Skipped A[0][2] or B[2][2] missing
   - NOTE: Skipped A[1][1] or B[1][2] missing
   - NOTE: Skipped A[1][2] or B[2][2] missing
  Expected (strict):
[
  [ 60, 72, 6 ],
  [ 144, 174, 24 ]
]
```

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

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

Possible reason for difference: Possible reason: jagged-row skipping or missing entries treated differently by your logic (skipped) vs strict zero-fill.

Test #22 — source: source_line_115 (source line 115)

Number of matrices: 3

System.out.println outputs (executed for this test):

```
[L16] 2Matrix(
[L17] Hence can not perform multiplication with Matrix(
[L21] 1Matrix(
[L168] \n\n***FINAL OUTCOME***: \n
[L212] ***ALL MATRIX*******:
[L237] \n***SUMMARY****
[L238] Matrix multiplication:
[L242] ADDED Multiplication Matrix into the chain at index:
[L245] Matrix(
[L254] \n***CALCULATION STEPS*******
[L293] Single matrix only
[L295] \n***CALCULATION STEPS*******
[L308] Size of Matrix multiplication row:
[L309] Size of Matrix multiplication columns:
[L325] THIS IS ROW COUNTER MATRIX(
[L326] ----------------------------------------THIS IS ROW DATA MATRIX(
[L473] 1Accessing matrix(
[L497] 2Accessing matrix(
[L513] 2Matrix(
[L514] Hence can not perform multiplication with Matrix(
[L521] 1Matrix(
[L540] 1Matrix(
[L580] Processed matrix(
[L600] VERIFICATION SUCCESSFUL(MATRIX(
[L649] \n***CALCULATION STEPS*******
```

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

```
  My-code result:
[
  [ 58, 64 ],
  [ 139, 154 ]
]
  Expected (strict) error: Incompatible dims colsA=4 rowsB=3
  Possible reason for difference: Possible reason: jagged-row skipping or missing entries treated differently by your logic (skipped) vs strict zero-fill.
```

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: source_line_118 (source line 118)

Number of matrices: 3

System.out.println outputs (executed for this test):

```
[L16] 2Matrix(
[L17] Hence can not perform multiplication with Matrix(
[L21] 1Matrix(
[L168] \n\n***FINAL OUTCOME***: \n
[L212] ***ALL MATRIX*******:
[L237] \n***SUMMARY****
[L238] Matrix multiplication:
[L242] ADDED Multiplication Matrix into the chain at index:
[L245] Matrix(
[L254] \n***CALCULATION STEPS*******
[L293] Single matrix only
[L295] \n***CALCULATION STEPS*******
[L308] Size of Matrix multiplication row:
[L309] Size of Matrix multiplication columns:
[L325] THIS IS ROW COUNTER MATRIX(
[L326] ----------------------------------------THIS IS ROW DATA MATRIX(
[L473] 1Accessing matrix(
[L497] 2Accessing matrix(
[L513] 2Matrix(
[L514] Hence can not perform multiplication with Matrix(
[L521] 1Matrix(
[L540] 1Matrix(
[L580] Processed matrix(
[L600] VERIFICATION SUCCESSFUL(MATRIX(
[L649] \n***CALCULATION STEPS*******
```

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

```
  My-code result:
[
  [ 58, 64 ],
  [ 28, 32 ]
]
   - NOTE: Skipped A[1][1] or B[1][0] missing
   - NOTE: Skipped A[1][2] or B[2][0] missing
   - NOTE: Skipped A[1][1] or B[1][1] missing
   - NOTE: Skipped A[1][2] or B[2][1] missing
  Expected (strict):
[
  [ 58, 64 ],
  [ 28, 32 ]
]
```

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: source_line_122 (source line 122)

Number of matrices: 3

System.out.println outputs (executed for this test):

```
[L16] 2Matrix(
[L17] Hence can not perform multiplication with Matrix(
[L21] 1Matrix(
[L168] \n\n***FINAL OUTCOME***: \n
[L212] ***ALL MATRIX*******:
[L237] \n***SUMMARY****
[L238] Matrix multiplication:
[L242] ADDED Multiplication Matrix into the chain at index:
[L245] Matrix(
[L254] \n***CALCULATION STEPS*******
[L293] Single matrix only
[L295] \n***CALCULATION STEPS*******
[L308] Size of Matrix multiplication row:
[L309] Size of Matrix multiplication columns:
[L325] THIS IS ROW COUNTER MATRIX(
[L326] ----------------------------------------THIS IS ROW DATA MATRIX(
[L473] 1Accessing matrix(
[L497] 2Accessing matrix(
[L513] 2Matrix(
[L514] Hence can not perform multiplication with Matrix(
[L521] 1Matrix(
[L540] 1Matrix(
[L580] Processed matrix(
[L600] VERIFICATION SUCCESSFUL(MATRIX(
[L649] \n***CALCULATION STEPS*******
```

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

```
  My-code result:
[
  [ 58, 64 ],
  [ 113, 126 ]
]
  Expected (strict) error: Incompatible dims colsA=4 rowsB=3
  Possible reason for difference: Possible reason: jagged-row skipping or missing entries treated differently by your logic (skipped) vs strict zero-fill.
```

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: source_line_125 (source line 125)

Number of matrices: 3

System.out.println outputs (executed for this test):

```
[L16] 2Matrix(
[L17] Hence can not perform multiplication with Matrix(
[L21] 1Matrix(
[L168] \n\n***FINAL OUTCOME***: \n
[L212] ***ALL MATRIX*******:
[L237] \n***SUMMARY****
[L238] Matrix multiplication:
[L242] ADDED Multiplication Matrix into the chain at index:
[L245] Matrix(
[L254] \n***CALCULATION STEPS*******
[L293] Single matrix only
[L295] \n***CALCULATION STEPS*******
[L308] Size of Matrix multiplication row:
[L309] Size of Matrix multiplication columns:
[L325] THIS IS ROW COUNTER MATRIX(
[L326] ----------------------------------------THIS IS ROW DATA MATRIX(
[L473] 1Accessing matrix(
[L497] 2Accessing matrix(
[L513] 2Matrix(
[L514] Hence can not perform multiplication with Matrix(
[L521] 1Matrix(
[L540] 1Matrix(
[L580] Processed matrix(
[L600] VERIFICATION SUCCESSFUL(MATRIX(
[L649] \n***CALCULATION STEPS*******
```

Step 1: Multiply A(2, 3) x B(3, 0)

```
  My-code result:
[
  [  ],
  [  ]
]
  Expected (strict):
[
  [ 53, 64 ],
  [ 116, 142 ]
]
  Possible reason for difference: Possible reason: jagged-row skipping or missing entries treated differently by your logic (skipped) vs strict zero-fill.
```

Step 2: Multiply A(2, 0) x B(2, 2)

```
  My-code result:
[
  [ 0, 0 ],
  [ 0, 0 ]
]
  Expected (strict) error: Incompatible dims colsA=0 rowsB=2
  Possible reason for difference: Possible reason: jagged-row skipping or missing entries treated differently by your logic (skipped) vs strict zero-fill.
```

Test #26 — source: source_line_127 (source line 127)

Number of matrices: 3

System.out.println outputs (executed for this test):

```
[L16] 2Matrix(
[L17] Hence can not perform multiplication with Matrix(
[L21] 1Matrix(
[L168] \n\n***FINAL OUTCOME***: \n
[L212] ***ALL MATRIX*******:
[L237] \n***SUMMARY****
[L238] Matrix multiplication:
[L242] ADDED Multiplication Matrix into the chain at index:
[L245] Matrix(
[L254] \n***CALCULATION STEPS*******
[L293] Single matrix only
[L295] \n***CALCULATION STEPS*******
[L308] Size of Matrix multiplication row:
[L309] Size of Matrix multiplication columns:
[L325] THIS IS ROW COUNTER MATRIX(
[L326] ----------------------------------------THIS IS ROW DATA MATRIX(
[L473] 1Accessing matrix(
[L497] 2Accessing matrix(
[L513] 2Matrix(
[L514] Hence can not perform multiplication with Matrix(
[L521] 1Matrix(
[L540] 1Matrix(
[L580] Processed matrix(
[L600] VERIFICATION SUCCESSFUL(MATRIX(
[L649] \n***CALCULATION STEPS*******
```

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

```
  My-code result:
[
  [ 68, 50 ],
  [ 158, 128 ]
]
   - NOTE: Skipped A[0][1] or B[1][0] missing
   - NOTE: Skipped A[0][1] or B[1][1] missing
   - NOTE: Skipped A[1][1] or B[1][0] missing
   - NOTE: Skipped A[1][1] or B[1][1] missing
  Expected (strict):
[
  [ 68, 50 ],
  [ 158, 128 ]
]
```

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: source_line_128 (source line 128)

Number of matrices: 3

System.out.println outputs (executed for this test):

```
[L16] 2Matrix(
[L17] Hence can not perform multiplication with Matrix(
[L21] 1Matrix(
[L168] \n\n***FINAL OUTCOME***: \n
[L212] ***ALL MATRIX*******:
[L237] \n***SUMMARY****
[L238] Matrix multiplication:
[L242] ADDED Multiplication Matrix into the chain at index:
[L245] Matrix(
[L254] \n***CALCULATION STEPS*******
[L293] Single matrix only
[L295] \n***CALCULATION STEPS*******
[L308] Size of Matrix multiplication row:
[L309] Size of Matrix multiplication columns:
[L325] THIS IS ROW COUNTER MATRIX(
[L326] ----------------------------------------THIS IS ROW DATA MATRIX(
[L473] 1Accessing matrix(
[L497] 2Accessing matrix(
[L513] 2Matrix(
[L514] Hence can not perform multiplication with Matrix(
[L521] 1Matrix(
[L540] 1Matrix(
[L580] Processed matrix(
[L600] VERIFICATION SUCCESSFUL(MATRIX(
[L649] \n***CALCULATION STEPS*******
```

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 ]
]
   - NOTE: Skipped A[0][1] or B[1][0] missing
   - NOTE: Skipped A[0][1] or B[1][1] missing
   - NOTE: Skipped A[1][1] or B[1][0] missing
   - NOTE: Skipped A[1][1] or B[1][1] missing
  Expected (strict):
```

```
[
  [ 2028, 2184 ],
  [ 4914, 5292 ]
]
```

Test #28 — source: source_line_130 (source line 130)

Number of matrices: 3

System.out.println outputs (executed for this test):

```
[L16] 2Matrix(
[L17] Hence can not perform multiplication with Matrix(
[L21] 1Matrix(
[L168] \n\n***FINAL OUTCOME***: \n
[L212] ***ALL MATRIX*******:
[L237] \n***SUMMARY****
[L238] Matrix multiplication:
[L242] ADDED Multiplication Matrix into the chain at index:
[L245] Matrix(
[L254] \n***CALCULATION STEPS*******
[L293] Single matrix only
[L295] \n***CALCULATION STEPS*******
[L308] Size of Matrix multiplication row:
[L309] Size of Matrix multiplication columns:
[L325] THIS IS ROW COUNTER MATRIX(
[L326] ----------------------------------------THIS IS ROW DATA MATRIX(
[L473] 1Accessing matrix(
[L497] 2Accessing matrix(
[L513] 2Matrix(
[L514] Hence can not perform multiplication with Matrix(
[L521] 1Matrix(
[L540] 1Matrix(
[L580] Processed matrix(
[L600] VERIFICATION SUCCESSFUL(MATRIX(
[L649] \n***CALCULATION STEPS*******
```

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 ]
]
   - NOTE: Skipped A[0][1] or B[1][1] missing
   - NOTE: Skipped A[1][1] or B[1][1] missing
  Expected (strict):
[
  [ 2028, 2184 ],
```

```
  [ 4914, 5292 ]
]
```

Test #29 — source: source_line_132 (source line 132)

Number of matrices: 3

System.out.println outputs (executed for this test):

```
[L16] 2Matrix(
[L17] Hence can not perform multiplication with Matrix(
[L21] 1Matrix(
[L168] \n\n***FINAL OUTCOME***: \n
[L212] ***ALL MATRIX*******:
[L237] \n***SUMMARY****
[L238] Matrix multiplication:
[L242] ADDED Multiplication Matrix into the chain at index:
[L245] Matrix(
[L254] \n***CALCULATION STEPS*******
[L293] Single matrix only
[L295] \n***CALCULATION STEPS*******
[L308] Size of Matrix multiplication row:
[L309] Size of Matrix multiplication columns:
[L325] THIS IS ROW COUNTER MATRIX(
[L326] ----------------------------------------THIS IS ROW DATA MATRIX(
[L473] 1Accessing matrix(
[L497] 2Accessing matrix(
[L513] 2Matrix(
[L514] Hence can not perform multiplication with Matrix(
[L521] 1Matrix(
[L540] 1Matrix(
[L580] Processed matrix(
[L600] VERIFICATION SUCCESSFUL(MATRIX(
[L649] \n***CALCULATION STEPS*******
```

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) error: Incompatible dims colsA=2 rowsB=3
  Possible reason for difference: Possible reason: jagged-row skipping or missing entries treated differently by your logic (skipped) vs strict zero-fill.
```

Test #30 — source: source_line_135 (source line 135)

Number of matrices: 3

System.out.println outputs (executed for this test):

```
[L16] 2Matrix(
[L17] Hence can not perform multiplication with Matrix(
[L21] 1Matrix(
[L168] \n\n***FINAL OUTCOME***: \n
[L212] ***ALL MATRIX*******:
[L237] \n***SUMMARY****
[L238] Matrix multiplication:
[L242] ADDED Multiplication Matrix into the chain at index:
[L245] Matrix(
[L254] \n***CALCULATION STEPS*******
[L293] Single matrix only
[L295] \n***CALCULATION STEPS*******
[L308] Size of Matrix multiplication row:
[L309] Size of Matrix multiplication columns:
[L325] THIS IS ROW COUNTER MATRIX(
[L326] ----------------------------------------THIS IS ROW DATA MATRIX(
[L473] 1Accessing matrix(
[L497] 2Accessing matrix(
[L513] 2Matrix(
[L514] Hence can not perform multiplication with Matrix(
[L521] 1Matrix(
[L540] 1Matrix(
[L580] Processed matrix(
[L600] VERIFICATION SUCCESSFUL(MATRIX(
[L649] \n***CALCULATION STEPS*******
```

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, 1)

```
  My-code result:
[
  [ 1860 ],
  [ 4482 ]
]
  Expected (strict):
[
  [ 1860, 1152 ],
  [ 4482, 2784 ]
]
```

Possible reason for difference: Possible reason: jagged-row skipping or missing entries treated differently by your logic (skipped) vs strict zero-fill.

Test #31 — source: source_line_139 (source line 139)

Number of matrices: 3

System.out.println outputs (executed for this test):

```
[L16] 2Matrix(
[L17] Hence can not perform multiplication with Matrix(
[L21] 1Matrix(
[L168] \n\n***FINAL OUTCOME***: \n
[L212] ***ALL MATRIX*******:
[L237] \n***SUMMARY****
[L238] Matrix multiplication:
[L242] ADDED Multiplication Matrix into the chain at index:
[L245] Matrix(
[L254] \n***CALCULATION STEPS*******
[L293] Single matrix only
[L295] \n***CALCULATION STEPS*******
[L308] Size of Matrix multiplication row:
[L309] Size of Matrix multiplication columns:
[L325] THIS IS ROW COUNTER MATRIX(
[L326] ----------------------------------------THIS IS ROW DATA MATRIX(
[L473] 1Accessing matrix(
[L497] 2Accessing matrix(
[L513] 2Matrix(
[L514] Hence can not perform multiplication with Matrix(
[L521] 1Matrix(
[L540] 1Matrix(
[L580] Processed matrix(
[L600] VERIFICATION SUCCESSFUL(MATRIX(
[L649] \n***CALCULATION STEPS*******
```

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 ]
]
   - NOTE: Skipped A[0][1] or B[1][1] missing
   - NOTE: Skipped A[1][1] or B[1][1] missing
  Expected (strict):
[
  [ 2028, 2184 ],
```

```
  [ 4914, 5292 ]
]
```

Test #32 — source: source_line_141 (source line 141)

Number of matrices: 3

System.out.println outputs (executed for this test):

```
[L16] 2Matrix(
[L17] Hence can not perform multiplication with Matrix(
[L21] 1Matrix(
[L168] \n\n***FINAL OUTCOME***: \n
[L212] ***ALL MATRIX*******:
[L237] \n***SUMMARY****
[L238] Matrix multiplication:
[L242] ADDED Multiplication Matrix into the chain at index:
[L245] Matrix(
[L254] \n***CALCULATION STEPS*******
[L293] Single matrix only
[L295] \n***CALCULATION STEPS*******
[L308] Size of Matrix multiplication row:
[L309] Size of Matrix multiplication columns:
[L325] THIS IS ROW COUNTER MATRIX(
[L326] ----------------------------------------THIS IS ROW DATA MATRIX(
[L473] 1Accessing matrix(
[L497] 2Accessing matrix(
[L513] 2Matrix(
[L514] Hence can not perform multiplication with Matrix(
[L521] 1Matrix(
[L540] 1Matrix(
[L580] Processed matrix(
[L600] VERIFICATION SUCCESSFUL(MATRIX(
[L649] \n***CALCULATION STEPS*******
```

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

```
  My-code result:
[
  [ 72, 72 ],
  [ 168, 174 ]
]
  Expected (strict):
[
  [ 72, 72, 195, 165 ],
  [ 168, 174, 390, 330 ]
]
  Possible reason for difference: Possible reason: jagged-row skipping or missing entries treated differently by your logic (skipped) vs strict zero-fill.
```

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

```
  My-code result:
[
  [ 2016, 8280 ],
  [ 4794, 19416 ]
]
  Expected (strict):
[
  [ 2016, 8280 ],
  [ 4794, 19416 ]
```

]

Test #33 — source: source_line_143 (source line 143)

Number of matrices: 3

System.out.println outputs (executed for this test):

```
[L16] 2Matrix(
[L17] Hence can not perform multiplication with Matrix(
[L21] 1Matrix(
[L168] \n\n***FINAL OUTCOME***: \n
[L212] ***ALL MATRIX*******:
[L237] \n***SUMMARY****
[L238] Matrix multiplication:
[L242] ADDED Multiplication Matrix into the chain at index:
[L245] Matrix(
[L254] \n***CALCULATION STEPS*******
[L293] Single matrix only
[L295] \n***CALCULATION STEPS*******
[L308] Size of Matrix multiplication row:
[L309] Size of Matrix multiplication columns:
[L325] THIS IS ROW COUNTER MATRIX(
[L326] ----------------------------------------THIS IS ROW DATA MATRIX(
[L473] 1Accessing matrix(
[L497] 2Accessing matrix(
[L513] 2Matrix(
[L514] Hence can not perform multiplication with Matrix(
[L521] 1Matrix(
[L540] 1Matrix(
[L580] Processed matrix(
[L600] VERIFICATION SUCCESSFUL(MATRIX(
[L649] \n***CALCULATION STEPS*******
```

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, 0)

```
  My-code result:
[
  [  ],
  [  ],
  [  ]
]
  Expected (strict) error: Incompatible dims colsA=3 rowsB=2
  Possible reason for difference: Possible reason: jagged-row skipping or missing entries treated differently by your logic (skipped) vs strict zero-fill.
```

Test #34 — source: source_line_145 (source line 145)

Number of matrices: 5

System.out.println outputs (executed for this test):

```
[L16] 2Matrix(
[L17] Hence can not perform multiplication with Matrix(
[L21] 1Matrix(
[L168] \n\n***FINAL OUTCOME***: \n
[L212] ***ALL MATRIX*******:
[L237] \n***SUMMARY****
[L238] Matrix multiplication:
[L242] ADDED Multiplication Matrix into the chain at index:
[L245] Matrix(
[L254] \n***CALCULATION STEPS*******
[L293] Single matrix only
[L295] \n***CALCULATION STEPS*******
[L308] Size of Matrix multiplication row:
[L309] Size of Matrix multiplication columns:
[L325] THIS IS ROW COUNTER MATRIX(
[L326] ----------------------------------------THIS IS ROW DATA MATRIX(
[L473] 1Accessing matrix(
[L497] 2Accessing matrix(
[L513] 2Matrix(
[L514] Hence can not perform multiplication with Matrix(
[L521] 1Matrix(
[L540] 1Matrix(
[L580] Processed matrix(
[L600] VERIFICATION SUCCESSFUL(MATRIX(
[L649] \n***CALCULATION STEPS*******
```

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: source_line_147 (source line 147)

Number of matrices: 5

System.out.println outputs (executed for this test):

```
[L16] 2Matrix(
[L17] Hence can not perform multiplication with Matrix(
[L21] 1Matrix(
[L168] \n\n***FINAL OUTCOME***: \n
[L212] ***ALL MATRIX*******:
[L237] \n***SUMMARY****
[L238] Matrix multiplication:
[L242] ADDED Multiplication Matrix into the chain at index:
[L245] Matrix(
[L254] \n***CALCULATION STEPS*******
[L293] Single matrix only
[L295] \n***CALCULATION STEPS*******
[L308] Size of Matrix multiplication row:
[L309] Size of Matrix multiplication columns:
[L325] THIS IS ROW COUNTER MATRIX(
[L326] ----------------------------------------THIS IS ROW DATA MATRIX(
[L473] 1Accessing matrix(
[L497] 2Accessing matrix(
[L513] 2Matrix(
[L514] Hence can not perform multiplication with Matrix(
[L521] 1Matrix(
[L540] 1Matrix(
[L580] Processed matrix(
[L600] VERIFICATION SUCCESSFUL(MATRIX(
[L649] \n***CALCULATION STEPS*******
```

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) error: Incompatible dims colsA=2 rowsB=4
  Possible reason for difference: Possible reason: jagged-row skipping or missing entries treated differently by your logic (skipped) vs strict zero-fill.
```

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

```
  My-code result:
[
  [ 3033762, 3165356, 3296950, 3428544 ],
  [ 7287009, 7603094, 7919179, 8235264 ]
]
  Expected (strict) error: Incompatible dims colsA=2 rowsB=3
  Possible reason for difference: Possible reason: jagged-row skipping or missing entries treated differently by your logic (skipped) vs strict zero-fill.
```

Test #36 — source: source_line_149 (source line 149)

Number of matrices: 3

System.out.println outputs (executed for this test):

```
[L16] 2Matrix(
[L17] Hence can not perform multiplication with Matrix(
[L21] 1Matrix(
[L168] \n\n***FINAL OUTCOME***: \n
[L212] ***ALL MATRIX*******:
[L237] \n***SUMMARY****
[L238] Matrix multiplication:
[L242] ADDED Multiplication Matrix into the chain at index:
[L245] Matrix(
[L254] \n***CALCULATION STEPS*******
[L293] Single matrix only
[L295] \n***CALCULATION STEPS*******
[L308] Size of Matrix multiplication row:
[L309] Size of Matrix multiplication columns:
[L325] THIS IS ROW COUNTER MATRIX(
[L326] ----------------------------------------THIS IS ROW DATA MATRIX(
[L473] 1Accessing matrix(
[L497] 2Accessing matrix(
[L513] 2Matrix(
[L514] Hence can not perform multiplication with Matrix(
[L521] 1Matrix(
[L540] 1Matrix(
[L580] Processed matrix(
[L600] VERIFICATION SUCCESSFUL(MATRIX(
[L649] \n***CALCULATION STEPS*******
```

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) error: Incompatible dims colsA=3 rowsB=2
  Possible reason for difference: Possible reason: jagged-row skipping or missing entries treated differently by your logic (skipped) vs strict zero-fill.
```

Test #37 — source: source_line_152 (source line 152)

Number of matrices: 2

System.out.println outputs (executed for this test):

```
[L16] 2Matrix(
[L17] Hence can not perform multiplication with Matrix(
[L21] 1Matrix(
[L168] \n\n***FINAL OUTCOME***: \n
[L212] ***ALL MATRIX*******:
[L237] \n***SUMMARY****
[L238] Matrix multiplication:
[L242] ADDED Multiplication Matrix into the chain at index:
[L245] Matrix(
[L254] \n***CALCULATION STEPS*******
[L293] Single matrix only
[L295] \n***CALCULATION STEPS*******
[L308] Size of Matrix multiplication row:
[L309] Size of Matrix multiplication columns:
[L325] THIS IS ROW COUNTER MATRIX(
[L326] ----------------------------------------THIS IS ROW DATA MATRIX(
[L473] 1Accessing matrix(
[L497] 2Accessing matrix(
[L513] 2Matrix(
[L514] Hence can not perform multiplication with Matrix(
[L521] 1Matrix(
[L540] 1Matrix(
[L580] Processed matrix(
[L600] VERIFICATION SUCCESSFUL(MATRIX(
[L649] \n***CALCULATION STEPS*******
```

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

```
  My-code result:
[
  [ 85, 91 ],
  [ 205, 220 ],
  [ 325, 349 ]
]
  Expected (strict):
[
  [ 85, 91, 84 ],
  [ 205, 220, 183 ],
  [ 325, 349, 282 ]
]
  Possible reason for difference: Possible reason: jagged-row skipping or missing entries treated differently by your logic (skipped) vs strict zero-fill.
```

Test #38 — source: source_line_154 (source line 154)

Number of matrices: 2

System.out.println outputs (executed for this test):

```
[L16] 2Matrix(
[L17] Hence can not perform multiplication with Matrix(
[L21] 1Matrix(
[L168] \n\n***FINAL OUTCOME***: \n
[L212] ***ALL MATRIX*******:
[L237] \n***SUMMARY****
[L238] Matrix multiplication:
[L242] ADDED Multiplication Matrix into the chain at index:
[L245] Matrix(
[L254] \n***CALCULATION STEPS*******
[L293] Single matrix only
[L295] \n***CALCULATION STEPS*******
[L308] Size of Matrix multiplication row:
[L309] Size of Matrix multiplication columns:
[L325] THIS IS ROW COUNTER MATRIX(
[L326] ----------------------------------------THIS IS ROW DATA MATRIX(
[L473] 1Accessing matrix(
[L497] 2Accessing matrix(
[L513] 2Matrix(
[L514] Hence can not perform multiplication with Matrix(
[L521] 1Matrix(
[L540] 1Matrix(
[L580] Processed matrix(
[L600] VERIFICATION SUCCESSFUL(MATRIX(
[L649] \n***CALCULATION STEPS*******
```

Step 1: Multiply A(3, 0) x B(3, 3)

```
  My-code result:
[
  [ 0, 0, 0 ],
  [ 0, 0, 0 ],
  [ 0, 0, 0 ]
]
  Expected (strict):
[
  [ 0, 0, 0 ],
  [ 201, 216, 231 ],
  [ 318, 342, 366 ]
]
  Possible reason for difference: Possible reason: jagged-row skipping or missing entries treated differently by your logic (skipped) vs strict zero-fill.
```

Test #39 — source: extra

Number of matrices: 2

System.out.println outputs (executed for this test):

```
[L16] 2Matrix(
[L17] Hence can not perform multiplication with Matrix(
[L21] 1Matrix(
[L168] \n\n***FINAL OUTCOME***: \n
[L212] ***ALL MATRIX*******:
[L237] \n***SUMMARY****
[L238] Matrix multiplication:
[L242] ADDED Multiplication Matrix into the chain at index:
[L245] Matrix(
[L254] \n***CALCULATION STEPS*******
[L293] Single matrix only
[L295] \n***CALCULATION STEPS*******
[L308] Size of Matrix multiplication row:
[L309] Size of Matrix multiplication columns:
[L325] THIS IS ROW COUNTER MATRIX(
[L326] ----------------------------------------THIS IS ROW DATA MATRIX(
[L473] 1Accessing matrix(
[L497] 2Accessing matrix(
[L513] 2Matrix(
[L514] Hence can not perform multiplication with Matrix(
[L521] 1Matrix(
[L540] 1Matrix(
[L580] Processed matrix(
[L600] VERIFICATION SUCCESSFUL(MATRIX(
[L649] \n***CALCULATION STEPS*******
```

Step 1: Multiply A(2, 0) x B(2, 1)

```
  My-code result:
[
  [ 0 ],
  [ 0 ]
]
  Expected (strict) error: Incompatible dims colsA=0 rowsB=2
  Possible reason for difference: Possible reason: jagged-row skipping or missing entries treated differently by your logic (skipped) vs strict zero-fill.
```

Test #40 — source: extra

Number of matrices: 3

System.out.println outputs (executed for this test):

```
[L16] 2Matrix(
[L17] Hence can not perform multiplication with Matrix(
[L21] 1Matrix(
[L168] \n\n***FINAL OUTCOME***: \n
[L212] ***ALL MATRIX*******:
[L237] \n***SUMMARY****
[L238] Matrix multiplication:
[L242] ADDED Multiplication Matrix into the chain at index:
[L245] Matrix(
[L254] \n***CALCULATION STEPS*******
[L293] Single matrix only
[L295] \n***CALCULATION STEPS*******
[L308] Size of Matrix multiplication row:
[L309] Size of Matrix multiplication columns:
[L325] THIS IS ROW COUNTER MATRIX(
[L326] ----------------------------------------THIS IS ROW DATA MATRIX(
[L473] 1Accessing matrix(
[L497] 2Accessing matrix(
[L513] 2Matrix(
[L514] Hence can not perform multiplication with Matrix(
[L521] 1Matrix(
[L540] 1Matrix(
[L580] Processed matrix(
[L600] VERIFICATION SUCCESSFUL(MATRIX(
[L649] \n***CALCULATION STEPS*******
```

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

```
  My-code result:
[
  [ 41 ],
  [ 20 ]
]
   - NOTE: Skipped A[1][1] or B[1][0] missing
   - NOTE: Skipped A[1][2] or B[2][0] missing
  Expected (strict):
[
  [ 41, 41 ],
  [ 20, 0 ]
]
  Possible reason for difference: Possible reason: jagged-row skipping or missing entries treated differently by your logic (skipped) vs strict zero-fill.
```

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

```
  My-code result:
[
  [ 41 ],
  [ 20 ]
]
  Expected (strict) error: Incompatible dims colsA=1 rowsB=2
  Possible reason for difference: Possible reason: jagged-row skipping or missing entries treated differently by your logic (skipped) vs strict zero-fill.
```

Test #41 — source: extra

Number of matrices: 2

System.out.println outputs (executed for this test):

```
[L16] 2Matrix(
[L17] Hence can not perform multiplication with Matrix(
[L21] 1Matrix(
[L168] \n\n***FINAL OUTCOME***: \n
[L212] ***ALL MATRIX*******:
[L237] \n***SUMMARY****
[L238] Matrix multiplication:
[L242] ADDED Multiplication Matrix into the chain at index:
[L245] Matrix(
[L254] \n***CALCULATION STEPS*******
[L293] Single matrix only
[L295] \n***CALCULATION STEPS*******
[L308] Size of Matrix multiplication row:
[L309] Size of Matrix multiplication columns:
[L325] THIS IS ROW COUNTER MATRIX(
[L326] ----------------------------------------THIS IS ROW DATA MATRIX(
[L473] 1Accessing matrix(
[L497] 2Accessing matrix(
[L513] 2Matrix(
[L514] Hence can not perform multiplication with Matrix(
[L521] 1Matrix(
[L540] 1Matrix(
[L580] Processed matrix(
[L600] VERIFICATION SUCCESSFUL(MATRIX(
[L649] \n***CALCULATION STEPS*******
```

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

```
  My-code result:
[
  [ 40, 44 ],
  [ 139, 154 ]
]
   - NOTE: Skipped A[0][1] or B[1][0] missing
   - NOTE: Skipped A[0][1] or B[1][1] missing
  Expected (strict):
[
  [ 40, 44 ],
  [ 139, 154 ]
]
```

Test #42 — source: extra

Number of matrices: 3

System.out.println outputs (executed for this test):

```
[L16] 2Matrix(
[L17] Hence can not perform multiplication with Matrix(
[L21] 1Matrix(
[L168] \n\n***FINAL OUTCOME***: \n
[L212] ***ALL MATRIX*******:
[L237] \n***SUMMARY****
[L238] Matrix multiplication:
[L242] ADDED Multiplication Matrix into the chain at index:
[L245] Matrix(
[L254] \n***CALCULATION STEPS*******
[L293] Single matrix only
[L295] \n***CALCULATION STEPS*******
[L308] Size of Matrix multiplication row:
[L309] Size of Matrix multiplication columns:
[L325] THIS IS ROW COUNTER MATRIX(
[L326] ----------------------------------------THIS IS ROW DATA MATRIX(
[L473] 1Accessing matrix(
[L497] 2Accessing matrix(
[L513] 2Matrix(
[L514] Hence can not perform multiplication with Matrix(
[L521] 1Matrix(
[L540] 1Matrix(
[L580] Processed matrix(
[L600] VERIFICATION SUCCESSFUL(MATRIX(
[L649] \n***CALCULATION STEPS*******
```

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

```
  My-code result:
[
  [ 1, 2, 3 ]
]
  Expected (strict) error: Incompatible dims colsA=2 rowsB=1
  Possible reason for difference: Possible reason: jagged-row skipping or missing entries treated differently by your logic (skipped) vs strict zero-fill.
```

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

```
  My-code result:
[
  [ 5 ]
]
  Expected (strict) error: Incompatible dims colsA=3 rowsB=2
  Possible reason for difference: Possible reason: jagged-row skipping or missing entries treated differently by your logic (skipped) vs strict zero-fill.
```

Test #43 — source: extra

Number of matrices: 8

System.out.println outputs (executed for this test):

```
[L16] 2Matrix(
[L17] Hence can not perform multiplication with Matrix(
[L21] 1Matrix(
[L168] \n\n***FINAL OUTCOME***: \n
[L212] ***ALL MATRIX*******:
[L237] \n***SUMMARY****
[L238] Matrix multiplication:
[L242] ADDED Multiplication Matrix into the chain at index:
[L245] Matrix(
[L254] \n***CALCULATION STEPS*******
[L293] Single matrix only
[L295] \n***CALCULATION STEPS*******
[L308] Size of Matrix multiplication row:
[L309] Size of Matrix multiplication columns:
[L325] THIS IS ROW COUNTER MATRIX(
[L326] ----------------------------------------THIS IS ROW DATA MATRIX(
[L473] 1Accessing matrix(
[L497] 2Accessing matrix(
[L513] 2Matrix(
[L514] Hence can not perform multiplication with Matrix(
[L521] 1Matrix(
[L540] 1Matrix(
[L580] Processed matrix(
[L600] VERIFICATION SUCCESSFUL(MATRIX(
[L649] \n***CALCULATION STEPS*******
```

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

```
  My-code result:
[
  [ 2 ]
]
  Expected (strict):
[
  [ 2 ]
]
```

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

```
  My-code result:
[
  [ 6 ]
]
  Expected (strict):
[
  [ 6 ]
]
```

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

```
  My-code result:
[
  [ 24 ]
```

```
]
  Expected (strict):
[
  [ 24 ]
]
```

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

```
  My-code result:
[
  [ 120 ]
]
  Expected (strict):
[
  [ 120 ]
]
```

Step 5: Multiply A(1, 1) x B(1, 1)

```
  My-code result:
[
  [ 720 ]
]
  Expected (strict):
[
  [ 720 ]
]
```

Step 6: Multiply A(1, 1) x B(1, 1)

```
  My-code result:
[
  [ 5040 ]
]
  Expected (strict):
[
  [ 5040 ]
]
```

Step 7: Multiply A(1, 1) x B(1, 1)

```
  My-code result:
[
  [ 40320 ]
]
  Expected (strict):
[
  [ 40320 ]
]
```

Test #44 — source: extra

Number of matrices: 4

System.out.println outputs (executed for this test):

```
[L16] 2Matrix(
[L17] Hence can not perform multiplication with Matrix(
[L21] 1Matrix(
[L168] \n\n***FINAL OUTCOME***: \n
[L212] ***ALL MATRIX*******:
[L237] \n***SUMMARY****
[L238] Matrix multiplication:
[L242] ADDED Multiplication Matrix into the chain at index:
[L245] Matrix(
[L254] \n***CALCULATION STEPS*******
[L293] Single matrix only
[L295] \n***CALCULATION STEPS*******
[L308] Size of Matrix multiplication row:
[L309] Size of Matrix multiplication columns:
[L325] THIS IS ROW COUNTER MATRIX(
[L326] ----------------------------------------THIS IS ROW DATA MATRIX(
[L473] 1Accessing matrix(
[L497] 2Accessing matrix(
[L513] 2Matrix(
[L514] Hence can not perform multiplication with Matrix(
[L521] 1Matrix(
[L540] 1Matrix(
[L580] Processed matrix(
[L600] VERIFICATION SUCCESSFUL(MATRIX(
[L649] \n***CALCULATION STEPS*******
```

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

```
  My-code result:
[
  [ 5, 6, 7 ],
  [ 15, 18, 21 ]
]
  Expected (strict) error: Incompatible dims colsA=2 rowsB=1
  Possible reason for difference: Possible reason: jagged-row skipping or missing entries treated differently by your logic (skipped) vs strict zero-fill.
```

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

```
  My-code result:
[
  [ 94 ],
  [ 282 ]
]
  Expected (strict) error: Incompatible dims colsA=3 rowsB=2
  Possible reason for difference: Possible reason: jagged-row skipping or missing entries treated differently by your logic (skipped) vs strict zero-fill.
```

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

```
  My-code result:
[
  [ 1034, 1128 ],
  [ 3102, 3384 ]
]
```

Expected (strict) error: Incompatible dims colsA=1 rowsB=2
Possible reason for difference: Possible reason: jagged-row skipping or missing entries treated differently by your logic (skipped) vs strict zero-fill.

Test #45 — source: extra

Number of matrices: 2

System.out.println outputs (executed for this test):

```
[L16] 2Matrix(
[L17] Hence can not perform multiplication with Matrix(
[L21] 1Matrix(
[L168] \n\n***FINAL OUTCOME***: \n
[L212] ***ALL MATRIX*******:
[L237] \n***SUMMARY****
[L238] Matrix multiplication:
[L242] ADDED Multiplication Matrix into the chain at index:
[L245] Matrix(
[L254] \n***CALCULATION STEPS*******
[L293] Single matrix only
[L295] \n***CALCULATION STEPS*******
[L308] Size of Matrix multiplication row:
[L309] Size of Matrix multiplication columns:
[L325] THIS IS ROW COUNTER MATRIX(
[L326] ----------------------------------------THIS IS ROW DATA MATRIX(
[L473] 1Accessing matrix(
[L497] 2Accessing matrix(
[L513] 2Matrix(
[L514] Hence can not perform multiplication with Matrix(
[L521] 1Matrix(
[L540] 1Matrix(
[L580] Processed matrix(
[L600] VERIFICATION SUCCESSFUL(MATRIX(
[L649] \n***CALCULATION STEPS*******
```

Step 1: Multiply A(1, 0) x B(2, 2)

```
  My-code result:
[
  [ 0, 0 ]
]
  Expected (strict) error: Incompatible dims colsA=0 rowsB=2
  Possible reason for difference: Possible reason: jagged-row skipping or missing entries treated differently by your logic (skipped) vs strict zero-fill.
```

Test #46 — source: extra

Number of matrices: 3

System.out.println outputs (executed for this test):

```
[L16] 2Matrix(
[L17] Hence can not perform multiplication with Matrix(
[L21] 1Matrix(
[L168] \n\n***FINAL OUTCOME***: \n
[L212] ***ALL MATRIX*******:
[L237] \n***SUMMARY****
[L238] Matrix multiplication:
[L242] ADDED Multiplication Matrix into the chain at index:
[L245] Matrix(
[L254] \n***CALCULATION STEPS*******
[L293] Single matrix only
[L295] \n***CALCULATION STEPS*******
[L308] Size of Matrix multiplication row:
[L309] Size of Matrix multiplication columns:
[L325] THIS IS ROW COUNTER MATRIX(
[L326] ----------------------------------------THIS IS ROW DATA MATRIX(
[L473] 1Accessing matrix(
[L497] 2Accessing matrix(
[L513] 2Matrix(
[L514] Hence can not perform multiplication with Matrix(
[L521] 1Matrix(
[L540] 1Matrix(
[L580] Processed matrix(
[L600] VERIFICATION SUCCESSFUL(MATRIX(
[L649] \n***CALCULATION STEPS*******
```

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 #47 — source: extra

Number of matrices: 3

System.out.println outputs (executed for this test):

```
[L16] 2Matrix(
[L17] Hence can not perform multiplication with Matrix(
[L21] 1Matrix(
[L168] \n\n***FINAL OUTCOME***: \n
[L212] ***ALL MATRIX*******:
[L237] \n***SUMMARY****
[L238] Matrix multiplication:
[L242] ADDED Multiplication Matrix into the chain at index:
[L245] Matrix(
[L254] \n***CALCULATION STEPS*******
[L293] Single matrix only
[L295] \n***CALCULATION STEPS*******
[L308] Size of Matrix multiplication row:
[L309] Size of Matrix multiplication columns:
[L325] THIS IS ROW COUNTER MATRIX(
[L326] ----------------------------------------THIS IS ROW DATA MATRIX(
[L473] 1Accessing matrix(
[L497] 2Accessing matrix(
[L513] 2Matrix(
[L514] Hence can not perform multiplication with Matrix(
[L521] 1Matrix(
[L540] 1Matrix(
[L580] Processed matrix(
[L600] VERIFICATION SUCCESSFUL(MATRIX(
[L649] \n***CALCULATION STEPS*******
```

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

```
  My-code result:
[
  [ 14 ],
  [ 12 ]
]
   - NOTE: Skipped A[1][1] or B[1][0] missing
  Expected (strict):
[
  [ 14, 12 ],
  [ 12, 0 ]
]
  Possible reason for difference: Possible reason: jagged-row skipping or missing entries treated differently by your logic (skipped) vs strict zero-fill.
```

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

```
  My-code result:
[
  [ 98, 112 ],
  [ 84, 96 ]
]
  Expected (strict) error: Incompatible dims colsA=1 rowsB=2
  Possible reason for difference: Possible reason: jagged-row skipping or missing entries treated differently by your logic (skipped) vs strict zero-fill.
```

Test #48 — source: extra

Number of matrices: 4

System.out.println outputs (executed for this test):

```
[L16] 2Matrix(
[L17] Hence can not perform multiplication with Matrix(
[L21] 1Matrix(
[L168] \n\n***FINAL OUTCOME***: \n
[L212] ***ALL MATRIX*******:
[L237] \n***SUMMARY****
[L238] Matrix multiplication:
[L242] ADDED Multiplication Matrix into the chain at index:
[L245] Matrix(
[L254] \n***CALCULATION STEPS*******
[L293] Single matrix only
[L295] \n***CALCULATION STEPS*******
[L308] Size of Matrix multiplication row:
[L309] Size of Matrix multiplication columns:
[L325] THIS IS ROW COUNTER MATRIX(
[L326] ----------------------------------------THIS IS ROW DATA MATRIX(
[L473] 1Accessing matrix(
[L497] 2Accessing matrix(
[L513] 2Matrix(
[L514] Hence can not perform multiplication with Matrix(
[L521] 1Matrix(
[L540] 1Matrix(
[L580] Processed matrix(
[L600] VERIFICATION SUCCESSFUL(MATRIX(
[L649] \n***CALCULATION STEPS*******
```

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

```
  My-code result:
[
  [ 1, 2 ]
]
  Expected (strict) error: Incompatible dims colsA=3 rowsB=1
  Possible reason for difference: Possible reason: jagged-row skipping or missing entries treated differently by your logic (skipped) vs strict zero-fill.
```

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

```
  My-code result:
[
  [ 3 ]
]
  Expected (strict) error: Incompatible dims colsA=2 rowsB=1
  Possible reason for difference: Possible reason: jagged-row skipping or missing entries treated differently by your logic (skipped) vs strict zero-fill.
```

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

```
  My-code result:
[
  [ 12, 15 ]
]
  Expected (strict):
[
  [ 12, 15 ]
```

]

Coverage Summary

Loops visited across all runs: 1, 2, 3, 4, 5, 6, 7

Loops not visited:

Blocks Not Entered and Explanations (best-effort)

If a try/catch or if block is not listed as entered, it likely means none of the tests triggered the condition or exception protected by that block. For example, no IndexOutOfBounds or NullPointer exceptions occurred in simulation, so catches were not used.

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);
//                             System.exit(0);
//                         }
//
//                         for (int z=0; z<nextMatrix.get(0).size(); z++)
```

```
//                          {
//                              System.out.println("\n*****************Column number: " + z);
//                              posPrevMatrixRowLevel=0;
//                              numWritesAtIndexWithinMultiplicationMatrix=0;
//
//                              for (int m=0; m<nextMatrix.size(); m++)
//                              {
//                                  for (int j=0;j<nextMatrix.get(m).size();j++)
//                                  {
//
//                                      if (numWritesAtIndexWithinMultiplicationMatrix==nextMatrix.size())
//                                      {
//                                          System.out.println("*****************************************************************");
//                                          break;
//                                      }
//                                      else
//                                      {
//                                          System.out.println("retrieving row: " + rowCounterNextMatrix
//                                          + " from matrix (" + (counter+1)+")");
//                                      }
//                                      rowInfo=new ArrayList<>(nextMatrix.get(m));
//
//                                      if (!hasReachedEndRow)
//                                      {
//                                          seekPosition=0;
//                                          excessRowItems = new StringJoiner(",");
//
//                                          if (rowInfo.size()!=matrixMultiplication[0].length)
//                                          {
//                                              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))
//                                              {
//                                                  System.out.println(currentEntry);
//                                                  excessRowReport.add(currentEntry);
//
//                                                  for (int x: nextMatrix.get(m))
//                                                  {
//                                                      seekPosition++;
//
//                                                      if (seekPosition>matrixMultiplication[0].length)
//                                                      {
//                                                          excessRowItems.add(String.valueOf(x));
//                                                      }
//                                                  }
//                                              }
//                                          }
//
//                                          prevEntry = currentEntry;
//
//                                          if (seekPosition!=0)
```

```
//                                      {
//                                          System.out.println("["+excessRowItems+"] are exempt from multiplcation matrix");
//                                          excessRowReport.add("["+excessRowItems+"] are exempt from multiplcation matrix");
//
//                                          if(excessRowItems.toString().length()==0)
//                                          {
//                                              System.out.println("10The current dimension of matrixMultiplication: "
//                                              + "["+matrix.size()+"]"+"["+nextMatrix.get(0).size()+"]");
//                                              System.out.println("3It EXPECTS: " + nextMatrix.get(0).size() + " elements per row");
//
//                                              System.out.println("matrix" + "("+(counter+1)+")" + " row:" + rowCounterNextMatrix
//                                              +  " " + "(content:"+nextMatrix.get(m) +")" + " size:" + nextMatrix.get(m).size()
//                                              + " invalidates this");
//                                              System.out.println(matrices);
//                                          }
//                                      }
//                                      try
//                                      {
//                                          System.out.println("Matrix["+(counter+1)+"] \trow element["+z+"]: " + rowInfo.get(z) +"\t\t" + nextMatrix.get(m));
//
//                                      }
//                                      catch (IndexOutOfBoundsException e)
//                                      {
//                                          System.out.println("\n24The current dimension of matrixMultiplication: "
//                                          + "["+nextMatrix.get(0).size()+"]"+"["+matrix.size()+"]");
//                                          System.out.println("4It EXPECTS: " + nextMatrix.get(0).size() + " elements per row");
//                                          System.out.println("matrix" + "("+(counter+1)+")" + " row:" + rowCounterNextMatrix
//                                          +  " " + nextMatrix.get(m) + " size: " + nextMatrix.get(m).size() + " invalidates this");
//                                          System.out.println(matrices);
//                                      }
//
//                                      try
//                                      {
//                                          System.out.println("Matrix["+counter+"] \trow element["+posPrevMatrixRowLevel+"]: " +ee.get(posPrevMatrixRowLevel) +"\t\t" +
//                                      }
//                                      catch (ArrayIndexOutOfBoundsException e)
//                                      {
//                                          System.out.println("\n1The current dimension of matrixMultiplication: "
//                                          + "["+matrixMultiplication.length+"]"+"["+matrixMultiplication[0].length+"]");
//
//                                          if (rowInfo.size()>matrixMultiplication.length)
//                                          {
//                                              System.out.println("1Accessing matrix(" + counter+") row index: "
//                                              + rowCounterNextMatrix + " from " + nextMatrix.get(m) + " invalidates this");
//                                              System.out.println("Remove " + nextMatrix.get(m) + " and rows beyond from matrix("
//                                              + (counter+1)+") row index: " + m);
//                                          }
//                                  ■
// ■■■■■    if(ee.size()>matrixMultiplication.length)
//                                          {
//                                              System.out.println("5It EXPECTS: " + matrix.size() + " elements per row");
//                                              System.out.println("matrix" + "("+(counter)+")" + " row:" + rowCounterFirstMatrix
//                                              +  " " + 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");
//                                      System.out.println("matrix" + "("+(counter+1)+")"
//                                      + " row:" + rowCounterNextMatrix +  " " + nextMatrix.get(m) + " size: "
//                                      + nextMatrix.get(m).size() + " invalidates this");
//                                  }
//
//                                  if (ee.size()<rowCounterNextMatrix)
//                                  {
//                                      System.out.println("2Accessing matrix(" + counter+") row index: "
//                                      + rowCounterNextMatrix + " from " + ee + " invalidates this");
//                                      System.out.println("Remove " + nextMatrix.get(m)
//                                      + " and rows beyond from matrix(" + (counter+1)+") row index: " + m);
//                                      System.out.println(matrices);
//                                      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+"]");
//                                  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(p
//                                  +"  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);
//
//                              try
//                              {
//                                  matrixMultiplication[rowCounterFirstMatrix][z] = matrixMultiplication[rowCounterFirstMatrix][z]
//                                  + multiplication;
//                              }
//                              catch (ArrayIndexOutOfBoundsException e)
//                              {
//                                  System.out.println("\n2The current dimension of matrixMultiplication: "
//                                  + "["+matrix.size()+"]"+"["+nextMatrix.get(0).size()+"]");
//                                  System.out.println("1It EXPECTS: " + nextMatrix.get(0).size() + " elements per row");
//                                  System.out.println("matrix" + "("+(counter+1)+")" + " row:" + rowCounterNextMatrix +  " "
//                                  + nextMatrix.get(m) + " size: " + nextMatrix.get(m).size() + " invalidates this");
```

```
//
//                                        System.out.println("1Matrix("+(counter)+")" + "   row:" + rowCounterFirstMatrix+"\t" +  "value at index["+z+"]: " + ee.get(p
//                                        +"  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: "
//                                    + 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");
//
//                                    if (z==(matrixMultiplication.length-1))
//                                    {
//                                        numWritesToLastIndexOnRowMultiplicationMatrix++;
//
//                                        if (numWritesToLastIndexOnRowMultiplicationMatrix==nextMatrix.size())
//                                        {
//                                            hasReachedEndRow=true;
//                                        }
//                                    }
//                                    posPrevMatrixRowLevel++;
//
//                                    break;
//                                }
//                                else
//                                {
//                                    System.out.println("-----------------------------------------------------");
//                                    System.out.println("previous written to each index location of multiplication matrix: "
//                                    + nextMatrix.size() + "  MAXIMUM times");
//                                    System.out.println("-----------------------------------------------------");
//
//                                    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+"): " +
//                }
//                numElementsInRow++;
//            }
//            if(counter!=(matricesSize-1))
//            {
//                for (List<Integer> rows: matrices.get((counter+1)))
//                {
//                    rowsNextMatrix++;
//                }
//                if ((numElementsInRow!=rowsNextMatrix && numElementsInRow<matrices.get(counter).size()) || numElementsInRow==0)
//                {
//                    System.out.println("INVALID VERIFICATION=>  columns " + "("+numElementsInRow+") in Matrix"  + "("+counter+")"
```

```
//                          + "(Row:" +rowCounterFirstMatrix + " content: " +  rowMatrix +  ") should match "
//                          +  rowsNextMatrix + " rows in matrix("+(counter+1)+")");
//                          System.out.println(matrices);
//                          System.exit(0);
//                      }
//                      else
//                      {
//                          System.out.println("VERIFICATION SUCCESSFUL(MATRIX("+counter+")  row("+rowCounterFirstMatrix+")) =>  ***COLUMNS" + "("+numElementsInRow+")\t   N
//
//                          excessColItems = new StringJoiner(",");
//
//                          if (numElementsInRow>nextMatrix.size())
//                          {
//                              currentColumnEntry = "columns" + "("+numElementsInRow+") in Matrix"  + "("+counter+")"
//                          + "(Row:" +rowCounterFirstMatrix + " content: " +  rowMatrix +  ") inconsistent with "
//                          +  rowsNextMatrix + " rows in matrix("+(counter+1)+").";
//
//                              System.out.println(currentColumnEntry);
//
//                              for (int x: matrix.get(rowCounterFirstMatrix))
//                              {
//                                  seekPosition++;
//
//                                  if (seekPosition>rowsNextMatrix)
//                                  {
//                                      excessColItems.add(String.valueOf(x));
//                                  }
//                              }
//                              if (seekPosition!=0)
//                              {
//                                  System.out.println("["+excessColItems+"] are exempt from multiplcation matrix");
//
//                                  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)
//          {
//              System.out.println(Arrays.toString(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);
//         }
//
//         for (Integer []m: matrixMultiplication)
//         {
//             matrixMultiplicationList.add(Arrays.asList(m));
//         }
//
//         System.out.println("\n-----MATRIX A   Analysis------");
//         System.out.println(excessColReport);
//
//         System.out.println("\n\n-----MATRIX B   Analysis------");
//         System.out.println(excessRowReport);
//
//         return matrixMultiplicationList;
//     }
// }
```