I am starting the test cases extremely early since there is significant difficulty in simply obtaining content out of the matrices (consisting of different dimension matrix)

TEST CASE 1: Creating code to simply output all the matrixes and also output the numbers within the matrix... FOR ALL the matrix in the matrices

Input [[[1, 2, 3], [4, 5, 6]], [[7, 8], [9, 10], [11, 12]], [[13, 14], [15, 16]]]
//This is the test case

Output (for Debugging)
Matrix: [[1, 2, 3], [4, 5, 6]] //I have clearly obtained the matrix
1 //I have obtained the integers within the matrix
2 //I have obtained the integers within the matrix
3 //I have obtained the integers within the matrix
4 //I have obtained the integers within the matrix
5 //I have obtained the integers within the matrix
6 //I have obtained the integers within the matrix
//Although it is clear that these are part of the same matrix, we can be entirely sure since in the input, it has introduced]]. I have clearly marked this boundary in the input above for verification.

//all information is correct as per the input
Matrix: [[7, 8], [9, 10], [11, 12]]
7
8
9
10
11
12

//all information is correct as per the input
Matrix: [[13, 14], [15, 16]]
13
14
15
16
num elements: 16

Your Result [] //We know that I have not performed calculations so I expect incorrect result

Expected Result [[1714, 1836], [4117, 4410]]

If the three matrix are created, it would be represented as such

So I am presented with matrixes as follows. This clearly varies to that presented in question. It also presents extreme difficulty since I have very little idea on the principles behind the formula but it requires MOVING COLUMN ACROSS IN FIRST MATRIX AND ROW ACROSS IN NEXT. The NUMBER OF COLUMNS AND ROWS MUST MATCH

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

B
[7, 8]
[9, 10]
[11, 12]

=(1x7)+(2x9)+(3x11),
(1x8)+(2x10)+(3x12),
(4x7)+(5x9)+(6x11),
(4x8)+(5x10)+(6x12)

58,64
139, 154

C
[13, 14]
[15, 16]

= C x 58,64
139, 154

=(58x13) + (64x15),
(58x14) + (64x16)
(139x13) + (154x15)
(139x14) + (154x16)

1714,1836
4117,4410

I have found it extremely difficult to grasp such a relatively straight forward process. I had to complete calculations several times not due to data entry, but due to moving in wrong direction of the matrix.

This is something I need to be extra careful of during testing also since the multiplications can render huge numbers..... And also I am applying the principles correctly.

I know that integers are displayed on the screen one at a time. The first point for completing multiplication would be at number 7.
I could perform necessary calculations as follows...
But it is also essential I check to see if there is validation in the matrix sizes prior to multiplication. I suspect Programiz have given valid cases..

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

B [7, 8]
[9, 10]
[11, 12]

C [13, 14]
[15, 16]

= C x 58,64
139, 154

We know when storing value in array, we can simply create a 2d array [rowSize][colSize]

=(1x7)+(2x9)+(3x11),
(1x8)+(2x10)+(3x12),
(4x7)+(5x9)+(6x11),
(4x8)+(5x10)+(6x12)

We need to ensure validation as below in order to create the correct storage array:

4️⃣ 2×3 × 2×2 → ❌ **Not possible**
Because the number of columns in A (3) doesn't match the number of rows in B (2).

💡 **In short:**

A size	B size	Result size	Valid?
2×2	2×2	2×2	✅
2×3	3×2	2×2	✅
3×2	2×4	3×4	✅
2×3	2×2	—	❌

This creates a primary challenge in development. If I decide to perform multiplications as soon as 7 arrives, I would continue successfully up to number 12..

But based on research, we can see that if there is another row below 12, it would invalidate the multiplication

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

[7, 8]
[9, 10]
[11, 12]
[13, 14] //There are four rows compared to three columns above

I will continue to develop my code and perhaps use a try and catch technique to handle situation if situation occurs. Otherwise I would require repeat loops again before main code execution.

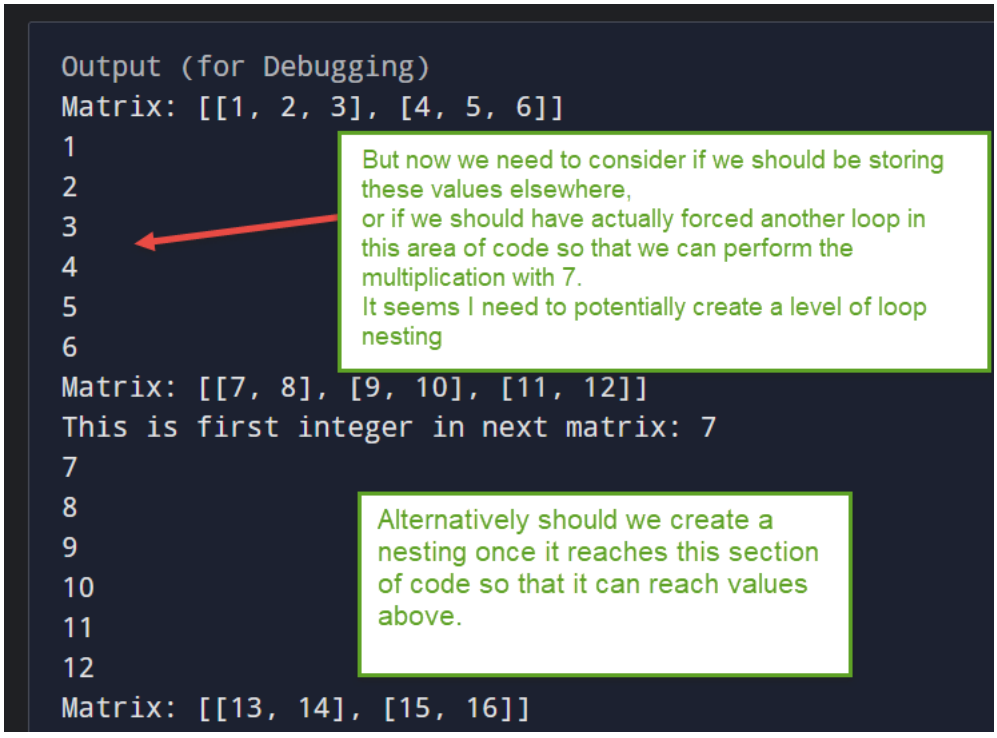
TEST CASE 2: Create point in code when the first digit of the second matrix is outputted to the screen

I simply created following core logic

```
for (List<Integer> ee: matrix)
{
    for (int k: ee)
    {
        if (matrixNumber<counter && hasStartMatrix)
        {
            System.out.println("This is first integer in
            next matrix: " + k);
            hasStartMatrix=false;
        }

        numElementsInList++;
        System.out.println(k);
    }
```

I can now start the main multiplications.
However when visualising the screen output again, it makes me suggest that I should have really been looking at next matrix as soon as the first appeared on the screen and attempted calculations. This is in conjunction with reducing intermediate variables.



Output (for Debugging)

Matrix(0): [[1, 2, 3], [4, 5, 6]]

1

2

3

columns in Matrix A (3) rows Matrix B: (3) //WE CAN SEE THAT WE CAN ONLY OFFICIALLY VALIDATE THE FORTHCOMING MATRIX ONCE IT HAS OBTAINED COLUMNS FOR EXISTING ONE... I AM NOT GOING TO EXTEND MY LOGIC TO CHECK IF THE LIST CONSISTS OF JAGGED ARRAY OF INTEGERS... I HAVE ONLY PERFORMED VALIDATION UPON RETRIEVING THE FIRST ROW. ALSO I AM CURRENTLY INCLUDING VALIDATION WHICH IS NOT PROPOSED IN THE CHALLENGE...
AT SOME POINT IN MY CODE, I AM CERTAIN I CAN TRY TO USE EXCEPTION HANDLING SHOULD AN ERROR OCCUR WITH IRREGULAR END USER INPUTS. WE KNOW IT IS VERY POSSIBLE

4

5

6

Matrix(1): [[7, 8], [9, 10], [11, 12]]

This is first integer in next matrix: 7

7

8

columns in Matrix A (2) rows Matrix B: (2) //Indication no issues found

9

10

11

12

Matrix(2): [[13, 14], [15, 16]] //and as expected no check against forthcoming matrix since it is the last one

This is first integer in next matrix: 13

13

14

15

16

num elements: 16

I managed to implement these changes and from now on saving code in repository matching Test case.

TEST CASE 3: Commencing the multiplication and implementing logic to support this

I have performed several nesting of loops gaining strong understanding through the documentation. This will be a perfect chance to try and explore the debug messages.

Output (for Debugging)

Matrix(0): [[1, 2, 3], [4, 5, 6]]

RESET first row on forthcoming matrix //This is performed to set numElementInNextMatrixRow=0 back to 0.. This is the location in the column that will be processed. I still have doubts if this is in correct location, but once I introduce more screen outputs, I should be able to remediate

Element: 1 //Indication of element in current matrix

REACH!!!!: row next matrix: 0num elem: 0

****

VALUE: 7 1x7 //it performs 1 x 7 as expected

INCREASE ROW NUMBER NEXT MATRIX //it now has to move a row down... But it has to remembered that it should stay on same column of forthcoming matrix UNTIL it reaches the end of row for Matrix A

REACH!!!!: row next matrix: 1num elem: 0

REACH!!!!: row next matrix: 1num elem: 0

Element: 2

REACH!!!!: row next matrix: 1num elem: 1

****

VALUE: 23 2x8 //we can see this is incorrect, we expected it to perform 2 x 9. This is critical error, so I will investigate this. At some point, I might consider outputting the matrix onto the screen so that end user can relate to the operation better

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

[7, 8]
[9, 10]
[11, 12]

[13,14]
[15,16]

INCREASE ROW NUMBER NEXT MATRIX

REACH!!!!: row next matrix: 2num elem: 1

REACH!!!!: row next matrix: 2num elem: 1

Element: 3

REACH!!!!: row next matrix: 2num elem: 2

****

VALUE: 47 3x8

INCREASE ROW NUMBER NEXT MATRIX

REACH!!!!: row next matrix: 3num elem: 2

REACH!!!!: row next matrix: 3num elem: 2

columns in Matrix A (3) rows Matrix B: (3)

RESET first row on forthcoming matrix

Element: 4

REACH!!!!: row next matrix: 3num elem: 0

REACH!!!!: row next matrix: 3num elem: 0

REACH!!!!: row next matrix: 3num elem: 0

Element: 5

REACH!!!!: row next matrix: 3num elem: 1

REACH!!!!: row next matrix: 3num elem: 1

REACH!!!!: row next matrix: 3num elem: 1

Element: 6

REACH!!!!: row next matrix: 3num elem: 2

REACH!!!!: row next matrix: 3num elem: 2

REACH!!!!: row next matrix: 3num elem: 2

Matrix(1): [[7, 8], [9, 10], [11, 12]]

RESET first row on forthcoming matrix

Element: 7

REACH!!!!: row next matrix: 0num elem: 0

****

VALUE: 138 7x13

INCREASE ROW NUMBER NEXT MATRIX

REACH!!!!: row next matrix: 1num elem: 0

This is first integer in next matrix: 7

Element: 8

REACH!!!!: row next matrix: 1num elem: 1

****

VALUE: 250 8x14

INCREASE ROW NUMBER NEXT MATRIX

REACH!!!!: row next matrix: 2num elem: 1

columns in Matrix A (2) rows Matrix B: (2)

RESET first row on forthcoming matrix

Element: 9

REACH!!!!: row next matrix: 2num elem: 0

REACH!!!!: row next matrix: 2num elem: 0

Element: 10

REACH!!!!: row next matrix: 2num elem: 1

REACH!!!!: row next matrix: 2num elem: 1

RESET first row on forthcoming matrix

Element: 11

REACH!!!!: row next matrix: 2num elem: 0

REACH!!!!: row next matrix: 2num elem: 0

Element: 12

REACH!!!!: row next matrix: 2num elem: 1

REACH!!!!: row next matrix: 2num elem: 1

Matrix(2): [[13, 14], [15, 16]]

RESET first row on forthcoming matrix

Element: 13

This is first integer in next matrix: 13

Element: 14

RESET first row on forthcoming matrix

Element: 15

Element: 16

num elements: 16

I have managed to create remediation, but it is an extremely tricky code to manipulate. This will be test case 4

TEST CASE 4: several changes above. It seems to be following a closer system, of traversing correct row of Matrix B. But it is not doing all actions for each element in row on Matrix A