

I have now managed to reconstruct the code but this time using my knowledge a bit more better of the List and for loop structures.

I have executed the code and this is the output:

TEST CASE 1: RUNNING THE REDESIGNED CODE

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

```
CURRENT TRANSFORMATION: [[1, 4, 7], [1, 4, 7], [1, 4, 7]]
```

```
Your Result
[[], [], []]
```

At this moment, I am unsure of the end outcome given the above transformation... But this will be final concern

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

There are several positive aspects. The List<Integer> are populated with correct number elements. Also the first List<Integer> is correct.

BUT clearly the area that requires addressing is that it has not progressed onto next index location for matrix.get(k).

I will simply analyse the debugging information as per usual:

Output (for Debugging)

-----Commencing transposing at index: 0

INNER LIST: [1, 2, 3] //This is correct

position: 0 //This is correct

counter: 0 //This is correct

EVER REACH

CURRENT LIST: [1]

PERFORMING BREAK

INNER LIST: [4, 5, 6] //This is correct

position: 1 //This has surprised me, I expected that position remained same, and counter would have started from 0... Although it functions in order to populate the first transposed List<Integer> the logic is not as expected

counter: 1
EVER REACH
CURRENT LIST: [1, 4]
PERFORMING BREAK
INNER LIST: [7, 8, 9]
position: 2 counter: 2
EVER REACH
CURRENT LIST: [1, 4, 7]
PERFORMING BREAK
IS IT HERE AFTER BREAK
CURRENT TRANSFORMATION: [[1, 4, 7]] //this is correct
-----Commencing transposing at index: 1
//this whole section performs a repeat of the above
INNER LIST: [1, 2, 3]
position: 1 counter: 1
EVER REACH
CURRENT LIST: [1]
PERFORMING BREAK
INNER LIST: [4, 5, 6]
position: 2 counter: 2
EVER REACH
CURRENT LIST: [1, 4]
PERFORMING BREAK
INNER LIST: [7, 8, 9]
position: 3 counter: 3
EVER REACH
CURRENT LIST: [1, 4, 7]
PERFORMING BREAK
IS IT HERE AFTER BREAK
CURRENT TRANSFORMATION: [[1, 4, 7], [1, 4, 7]]
-----Commencing transposing at index: 1
//this whole section performs a repeat of the above
INNER LIST: [1, 2, 3]
position: 1 counter: 1
EVER REACH
CURRENT LIST: [1]
PERFORMING BREAK
INNER LIST: [4, 5, 6]
position: 2 counter: 2
EVER REACH CURRENT LIST: [1, 4]

PERFORMING BREAK

INNER LIST: [7, 8, 9]

position: 3 counter:

3 EVER REACH

CURRENT LIST: [1, 4, 7]

PERFORMING BREAK

IS IT HERE AFTER BREAK CURRENT TRANSFORMATION: [[1, 4, 7], [1, 4, 7], [1, 4, 7]]

//hence this is complete repeat transposition for each column and row. It has only action first index!

TEST CASE 2: Tidied up some of the messages above, notably identifying the correct location of code upon performing a break.....

-----Commencing transposing at index: 0

IS IT HERE AFTER BREAK

INNER LIST: [1, 2, 3]

position: 0

counter: 0

EVER REACH

CURRENT LIST: [1]

1position: 0 counter: 0

HERE AFTER BREAK

IS IT HERE AFTER BREAK

INNER LIST: [4, 5, 6]

position: 0

counter: 0

EVER REACH

CURRENT LIST: [1, 4]

1position: 0 counter: 0

HERE AFTER BREAK

IS IT HERE AFTER BREAK

INNER LIST: [7, 8, 9]

position: 0

counter: 0

EVER REACH

CURRENT LIST: [1, 4, 7]

1position: 0 counter: 0

HERE AFTER BREAK

CURRENT TRANSFORMATION: [[1, 4, 7]]

//It is perfectly fine up to here

-----Commencing transposing at index: 1 //this is correct message

IS IT HERE AFTER BREAK

INNER LIST: [1, 2, 3]

position: 1

counter: 0

position: 1

counter: 1 //we can see the counter has been increased in a newly implemented else statement... This will ensure that it always reaches the correct index location in List<Integer>

EVER REACH

CURRENT LIST: [2]

1position: 1 counter: 1 //at this point in time, we expected the counter to be set again to 0. Otherwise position will evaluate as counter... And it will automatically pick index 0 again.

TEST CASE 3: Executing the code again taken into consideration the above changes

-----Commencing transposing at index: 0

IS IT HERE AFTER BREAK

INNER LIST: [1, 2, 3]

position: 0

counter: 0

EVER REACH

CURRENT LIST: [1]

1position: 0 counter: 0

HERE AFTER BREAK

IS IT HERE AFTER BREAK

INNER LIST: [4, 5, 6]

position: 0

counter: 0

EVER REACH

CURRENT LIST: [1, 4]

1position: 0 counter: 0

HERE AFTER BREAK

IS IT HERE AFTER BREAK

INNER LIST: [7, 8, 9]

position: 0

counter: 0

EVER REACH

CURRENT LIST: [1, 4, 7] [//THIS IS CORRECT](#)

1position: 0 counter: 0

HERE AFTER BREAK

CURRENT TRANSFORMATION: [[1, 4, 7]]

-----Commencing transposing at index: 1

IS IT HERE AFTER BREAK

INNER LIST: [1, 2, 3]

position: 1

counter: 0

position: 1

counter: 1

EVER REACH

CURRENT LIST: [2]

1position: 1 counter: 1

HERE AFTER BREAK

IS IT HERE AFTER BREAK

INNER LIST: [4, 5, 6]

position: 1

counter: 0

position: 1

counter: 1

EVER REACH

CURRENT LIST: [2, 5]

1position: 1 counter: 1

HERE AFTER BREAK

IS IT HERE AFTER BREAK

INNER LIST: [7, 8, 9]

position: 1

counter: 0

position: 1

counter: 1

EVER REACH

CURRENT LIST: [2, 5, 8] [//THIS IS CORRECT](#)

1position: 1 counter: 1

HERE AFTER BREAK

CURRENT TRANSFORMATION: [[2, 5, 8], [2, 5, 8]] //But this is the first area that is surprising since it has written the same List<Integer> twice and overwritten previous

-----Commencing transposing at index: 2

IS IT HERE AFTER BREAK

INNER LIST: [1, 2, 3]

position: 2

counter: 0

position: 2

counter: 1

position: 2

counter: 2

EVER REACH

CURRENT LIST: [3]

1position: 2 counter: 2

HERE AFTER BREAK

IS IT HERE AFTER BREAK

INNER LIST: [4, 5, 6]

position: 2

counter: 0

position: 2

counter: 1

position: 2

counter: 2

EVER REACH

CURRENT LIST: [3, 6]

1position: 2 counter: 2

HERE AFTER BREAK

IS IT HERE AFTER BREAK

INNER LIST: [7, 8, 9]

position: 2

counter: 0

position: 2

counter: 1

position: 2

counter: 2

EVER REACH

CURRENT LIST: [3, 6, 9] **//THIS IS CORRECT**

1position: 2 counter: 2

HERE AFTER BREAK

CURRENT TRANSFORMATION: [[3, 6, 9], [3, 6, 9], [3, 6, 9]]

//But this is the first area that is surprising since it has written the same List<Integer> three times and overwritten previous

I am fairly close to the solution now

```
INNER LIST: [7, 8, 9]
position: 0
counter: 0
EVER REACH
CURRENT LIST: [1, 4, 7]
1position: 0 counter: 0
HERE AFTER BREAK
2CURRENT TRANSFORMED MATRIX: []
NEWLY TRANSFORMATION: [[1, 4, 7]]
-----Commencing transposing at index: 1
1CURRENT TRANSFORMED MATRIX: [[]]
IS IT HERE AFTER BREAK
INNER LIST: [1, 2, 3]
position: 1
counter: 0
position: 1
counter: 1
EVER REACH
```

It can be seen that by time it reaches the next iteration of the most outer for loop, it has lost the value in finalMatrix.


```
-----Commencing transposing at index: 0
```

```
2CURRENT TRANSFORMED MATRIX: []  
NEWLY TRANSFORMATION: [[1, 4, 7]]
```

```
-----Commencing transposing at index: 1
```

```
counter: 0  
position: 1  
counter: 1  
EVER REACH  
CURRENT LIST: [2, 5, 8]  
1position: 1 counter: 1  
HERE AFTER BREAK  
2CURRENT TRANSFORMED MATRIX: [[2, 5, 8]]  
NEWLY TRANSFORMATION: [[2, 5, 8], [2, 5, 8]]  
-----Commencing transposing at index: 2  
1CURRENT TRANSFORMED MATRIX: [[1, 4, 7], [2, 5, 8], [2, 5, 8]]
```

It is first time code has entered this section since it has finished transposing at index 0 above

There seems to be an error in the IDE. There is absolutely no possibility the finalMatrix can be overwritten. I can attempt to declare it as class level, although it logically makes no sense since all logic is local to the method

TEST CASE 4: Attempting to change the finalMatrix to class level and also attempting to change it to static

```
1 import java.util.*;  
2 public class Solution {  
3  
4     static List<List<Integer>> finalMatrix = new  
       ArrayList<List<Integer>>();  
5     -----  
6  
7     public static List<List<Integer>> transposeMatrix  
       (List<List<Integer>> matrix) {  
8  
9         List<Integer> ll = new ArrayList<Integer>();  
10
```

It is now class level variable but outcomes are still the same