Java finalMatrix Debug – Explanation and Fix

This document explains the cause of the overwriting issue in the provided Java code and presents a minimal, annotated fix without refactoring. It preserves the user's original logic and debugging output.

Annotated and Corrected Code

```
import java.util.*;

public class Main
{
    // This is fine as a class-level variable; not the source of the overwrite issue.
    static List<List<Integer>> finalMatrix = new ArrayList<List<Integer>>();

    public static void main (String[] args)
    {
        Integer [][]test = new Integer[][] {{1,2,3},{4,5,6},{7,8,9}};
        //Integer [][]test = new Integer[][] {{10,20},{30,40}};
        //Integer [][]test = new Integer[][] {{11},{22},{33}};
        //Integer [][]test = new Integer[][] {{1,2},{3,4}};

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

        for (Integer [] row: test)
        {
            matrix.add(Arrays.asList(row));
        }

        transposeMatrix(matrix);
    }

    public static List<List<Integer>> transposeMatrix(List<List<Integer>> matrix)
    {
        // ■■ Original bug: temp was defined here once and reused.
        // List<Integer> temp = new ArrayList<Integer>();  // ■ Remove or comment this out

        int size=0;
        int matrixSize=matrix.size();
        int posInList=0;
        int numElementsInList=0;
        int counter=0;
        int i;
        int position=0;
        boolean hasTransposed=false;

        // Determine number of elements in a row (columns count)
        for (List<Integer> ee: matrix)
        {
            for (int k: ee)
            {
                numElementsInList++;
            }
            break;
        }

        for (int j=0; j<numElementsInList;j++)
        {
            // ■ FIX: Create a new temp list INSIDE the loop each time.
            // Each iteration needs its *own* list, otherwise they all reference the same memory.
            List<Integer> temp = new ArrayList<Integer>();  // ■ Correct placement

            System.out.println("--------Commencing transposing at index: " + position);
            counter=0;
            System.out.println("1CURRENT TRANSFORMED MATRIX: " + finalMatrix);

            for (int k=0; k<matrixSize; k++)
            {
                hasTransposed=false;
                System.out.println("INNER LIST: " + matrix.get(k));
```

```
                    for (int l: matrix.get(k))
                    {
                        if (!hasTransposed)
                        {
                            System.out.println("position: " + position + "\tcounter: " + counter);

                            if (counter==position)
                            {
                                temp.add(l);
                                System.out.println("CURRENT LIST: " + temp);
                                hasTransposed=true;
                                counter=0;
                            }
                            else
                            {
                                counter++;
                            }
                        }
                    }
                }

                position++;

                System.out.println("CURRENT TRANSFORMED MATRIX: " + finalMatrix);
                finalMatrix.add(temp);  // ■ Each 'temp' is now a unique list
                System.out.println("NEWLY TRANSFORMATION: " + finalMatrix);

                // ■■ Do NOT clear temp here anymore! Each iteration has its own list.
                // temp.clear();  // ■ Removing this prevents the lists from being emptied.
            }

            return finalMatrix;
        }
    }
```

Summary of Changes: - ■ Moved `List temp = new ArrayList<>()` inside the outer loop to give each iteration its own list. - ■ Removed `temp.clear()` to avoid erasing data referenced by `finalMatrix`. - ■■ Added explanatory comments marking the fixed and original bug areas. - ■ All debug print statements and original logic are preserved.

Behavior Explanation: Before Fix: - The same `temp` list object was reused across iterations. - Each `finalMatrix.add(temp)` added a reference to the same list. - `temp.clear()` emptied that list, making all entries in `finalMatrix` appear blank. After Fix: - A new list is created each iteration. - Each `temp` is independent and remains populated. - The final output for the 3x3 test matrix is: [[1, 4, 7], [2, 5, 8], [3, 6, 9]].