

NAME

DATE

PERIOD

UNIT 4 · TOPIC 4.13 · WORKSHEET

Work the Grid

Aim your loops at a row, a column, or the whole grid, then total, count, flag, rotate, and reverse.

About 30 minutes

10 questions

Cumulative through 4.13

TOPIC

4.13



DIRECTIONS

Work each puzzle in pencil. For code, write exactly what prints. Every 2D array here is rectangular, with rows first: `grid[r][c]`. Some puzzles include pieces you won't need.

PART A · 3 QUESTIONS

A

Warm up your brain

1

YOU'RE THE JVM

You are the Java Virtual Machine. Run this code in your head. What prints?

```
int[][] pts = { {9, 4, 7}, {6, 8, 5},
                {3, 9, 8}, {7, 2, 6} };
int sum = 0;
for (int r = 0; r < pts.length; r++) {
    sum += pts[r][1];
}
System.out.println(sum / pts.length + " " + (double) sum / pts.length);
```

A

 5 5.75

B

 5.75 5.75

C

 5 5.0

D

 7 7.666666666666667

2 PENCIL CHECK

Each loop header below uses the element shown, one pass at a time. Write the values it visits, in order.

```
int[][] g = { {3, 8, 1, 6}, {5, 2, 9, 4}, {7, 0, 4, 2} };
```

Loop header	Element used	Values visited
for (int c = 0; c < g[1].length; c++)	g[1][c]	
for (int r = 0; r < g.length; r++)	g[r][2]	
for (int i = 0; i < g.length; i++)	g[i][i]	
for (int c = 1; c < g[0].length - 1; c++)	g[0][c]	
for (int r = g.length - 1; r >= 0; r--)	g[r][0]	
for (int c = 0; c < g[2].length - 1; c++)	g[2][c + 1]	

3 YOU'RE THE COMPILER

Each snippet uses the array g below. Decide whether it compiles. If it does, write what it prints, or say that it crashes and after printing what. If it doesn't compile, say why.

```
int[][] g = { {4, 1, 7},
              {2, 9, 3} };
```

Snippet	Compiles?	Output, crash, or why not
1 for (int r = 0; r < g[0].length; r++) { System.out.println(g[r][2]); }		
2 for (int c = 0; c < g[1].length - 1; c++) { System.out.println(g[1][c + 1] - g[1][c]); }		
3 int count = g.length() * g[0].length();		
4 for (int c = 0; c < g[0].length; c++) { System.out.println(g[c][c]); }		



WATCH OUT

Ask which index the loop variable stands for. If it's the **row** index, its bound is the number of rows, `grid.length`. If it's the **column** index, its bound is the length of a row, `grid[r].length`. Mixing them up works on a square grid and crashes on the others.

PART B · 3 PUZZLES

B

Puzzles

4

SNIPPET POOL

Fill each numbered hole with a snippet from the pool so the program reverses row 1 and prints 5 3 8 1 6. Hole 2 appears twice and gets the same snippet both times. Not every snippet is used.

```
int[][] rack = { {2, 4, 6, 8, 10},
                 {6, 1, 8, 3, 5} };

int row = 1;
int len = rack[row].length;
for (int c = 0; c < (1); c++) {
    int temp = rack[row][c];
    rack[row][c] = rack[row][(2)];
    rack[row][(2)] = (3);
}
for (int v : rack[row]) {
    System.out.print(v + " ");
}
```

THE POOL

Each snippet can be used only once, and not every snippet is needed.

len	len - 1 - c	rack[row][c]	len / 2
temp	len - 1		



FAIR QUESTION

Why does the swap need temp at all?

Because an assignment erases what was there. After `a = b;`, the old value of `a` is gone, so `b = a;` just copies `b` back into itself. Saving one value in a temporary variable first is the only way to trade two values.



THINK FIRST

A column of blocks drops down one row. Which value gets overwritten first, the top one or the bottom one? Guess in the margin; question 5 settles it.

5 CODE TILES

A falling-blocks game **rotates** column 1 down by one spot: every block drops one row, and the bottom block wraps around to the top. The tiles fell off the fridge. Put them back between the lines already written so the program prints the target output. One tile is a decoy.

```
int[][] grid = { {1, 2}, {3, 4}, {5, 6} };
int col = 1;
int end = grid.length - 1;
// your tiles go here
for (int[] row : grid) {
    System.out.println(row[0] + " " + row[1]);
}
```

grid[0][col] = saved;	for (int r = end; r > 0; r--) {
int saved = grid[end][col];	grid[r][col] = grid[r - 1][col];
}	for (int r = 1; r <= end; r++) {

Target output

```
1 6
3 2
5 4
```

YOUR TILES, IN ORDER

6 MATCH-UP

Two lines go into this code: `int count = 0;` and `System.out.print(count + " ");`. Each version below puts them in different spots. Match each version to what happens, from the list. Each result is used once.

```
int[][] rain = { {0, 3, 0},
                 {2, 0, 4},
                 {0, 0, 0} };

// spot A
for (int r = 0; r < rain.length; r++) {
    // spot B
    for (int c = 0; c < rain[r].length; c++) {
        if (rain[r][c] > 0) {
            count++;
        }
    }
    // spot C
}
// spot D
```

Version	<code>int count = 0;</code> goes at	<code>print</code> goes at	Result
1	spot A	spot D	
2	spot B	spot C	
3	spot A	spot C	
4	spot B	spot D	

P 1 3 3 **Q** 3 **R** It doesn't compile. **S** 1 2 0

PART C · 4 QUESTIONS

C Catch the mistake

7 ONE-TOKEN FIX

This program should print `true` when every shelf in the store has at least one of each item (no `0` anywhere). Every value is above `0`, but it prints `false`. Change exactly one token to fix it.

```
1  int[][] stock = { {3, 6, 1}, {2, 4, 5} };
2  boolean allStocked = false;
3  for (int[] shelf : stock) {
4      for (int n : shelf) {
5          if (n == 0) { allStocked = false; }
6      }
7  }
8  System.out.println(allStocked);
```

LINE, OLD TOKEN, NEW TOKEN

8 FACE-OFF

Tonight: Neighbor Pairs and Duplicate Check argue about who compares better. Fill in the blanks.

NEIGHBOR PAIRS I compare each element with the one right next to it: `row[c]` and _____.

DUPLICATE CHECK Only next door? I compare each element with every element that comes _____ it, so my inner loop starts at _____.

NEIGHBOR PAIRS Careful. Start at `j = 0` and on your first comparison you check an element against _____. Then you say `true` for every row.

DUPLICATE CHECK Fine. But you have to stop one early, at _____, or the last element looks for a neighbor that isn't there.

NEIGHBOR PAIRS True. A row of 5 has only _____ pairs of neighbors. Skip the `- 1` and I throw an _____.

9 TWO TRUTHS AND A LIE

A submarine logs depths in meters (negative is below the surface). Two of the claims about this code are true and one is a lie. Circle the lie, then rewrite it so it's true.

```
int[][] depth = { {-12, -30, -7}, {-25, -9, -18} };
int shallowest = 0;
int deepest = depth[0][0];
for (int r = 0; r < depth.length; r++) {
    for (int c = 0; c < depth[r].length; c++) {
        if (depth[r][c] > shallowest) { shallowest = depth[r][c]; }
        if (depth[r][c] < deepest) { deepest = depth[r][c]; }
    }
}
```

- **1** After the loops, deepest holds -30.
- **2** After the loops, shallowest holds -7.
- **3** The body of the first if never runs.

THE LIE, REWRITTEN

10 WHO AM I?

Some 2D array algorithm ideas are playing a party game. Each gives a clue about itself. Name the idea.

Clue	I am...
I'm a boolean. Set me once before the loops and switch me when one element decides the answer.	
I start at 0 and go up by 1 each time an element passes the test.	
I move every element one spot, and the one pushed off the end is lost.	
I move every element one spot, and the one pushed off the end wraps around.	
My row and column indexes are always equal: <code>[i][i]</code> .	