

NAME _____

DATE _____

PERIOD _____

UNIT 4 · TOPIC 4.5 · WORKSHEET

The Array Workshop

Find the best, count the matches, spot the repeats, and rotate a row without losing anyone.

About 30 minutes

10 questions

Cumulative through 4.5

TOPIC

4.5

**DIRECTIONS**

Work each puzzle in pencil. For code, write exactly what prints. You may use the Java Quick Reference. Some puzzles include pieces you won't need.

PART A · 3 QUESTIONS

A

Warm up your brain

1

YOU'RE THE JVM

To find the warmest of four winter temperatures, this code starts max at 0, since surely every value beats it. Run the code in your head. What prints?

```
int[] temps = {-4, -9, -2, -7};
int max = 0;
for (int t : temps) {
    if (t > max) {
        max = t;
    }
}
System.out.println(max);
```

A 0**B** -2**C** -9**D** -4



FAIR QUESTION

So what should a maximum start at?

At an element of the array, usually the first one. Then the best-so-far is always a real value, and the loop can only replace it with a bigger real value. The same goes for a minimum: start it at the first element, not at 0 or some big number you hope is big enough.

2 PENCIL CHECK

Each row runs a standard algorithm on `int[] v = {3, 8, 8, 1, 6};`. Fill in the result.

The algorithm finds...	Result
The sum of all the elements, stored in <code>int sum</code>	
<code>sum / v.length</code>	
<code>(double) sum / v.length</code>	
The index of the minimum	
How many elements are greater than 5	
How many neighbor pairs go up, where <code>v[i] < v[i + 1]</code>	
Whether any value appears more than once	

3 YOU'RE THE JVM

Hopper's lap counts for five days are in `Laps`. The code keeps the **index** of the best day. What prints?

```
int[] laps = {48, 52, 47, 52, 50};
int best = 0;
for (int i = 1; i < laps.length; i++) {
    if (laps[i] > laps[best]) {
        best = i;
    }
}
System.out.println(best + " " + laps[best]);
```

A 1 52

B 3 52

C 52 52

D 4 50

PART B · 3 PUZZLES

B Puzzles

4 SNIPPET POOL

`allPositive` should return `true` only when **every** element is greater than `0`. `main` calls it on `{3, 1, 4}`, then `{2, 0, 5}`, then `{-6}`, and should print `true`, `false`, `false`. Fill each hole from the pool. Each snippet is used at most once.

```
public static boolean allPositive(int[] nums) {
    for (int n : nums) {
        if ( (1) ) {
            return (2);
        }
    }
    return (3);
}
```

THE POOL

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

<code>n >= 0</code>	<code>true</code>	<code>n < 0</code>	<code>nums[n] <= 0</code>
<code>false</code>	<code>n <= 0</code>		

5 CODE TILES

`String[] row = {"Ann", "Ben", "Cy", "Dee"};` At the end of each round, everyone moves one seat to the left, and the person in seat 0 walks around to the last seat. Rotate `row` left by one using five tiles. Two tiles are decoys.

<code>row[i] = row[i - 1];</code>	<code>}</code>
<code>String first = row[0];</code>	<code>for (int i = 0; i < row.length; i++) {</code>
<code>row[row.length - 1] = first;</code>	<code>row[i - 1] = row[i];</code>
<code>for (int i = 1; i < row.length; i++) {</code>	

row afterward

`"Ben" "Cy" "Dee" "Ann"`

YOUR CODE

6 MATCH-UP

Each loop runs on a fresh array `int[] a = {1, 2, 3, 4, 5};`. Match each loop to what `a` holds afterward. Each answer is used once.

Loop	Code	a holds
1	<pre>for (int i = 0; i < a.length - 1; i++) { a[i] = a[i + 1]; }</pre>	
2	<pre>for (int i = 0; i < a.length - 1; i++) { a[i + 1] = a[i]; }</pre>	
3	<pre>for (int i = a.length - 1; i > 0; i--) { a[i] = a[i - 1]; }</pre>	
4	<pre>for (int i = 0; i < a.length; i++) { int temp = a[i]; a[i] = a[a.length - 1 - i]; a[a.length - 1 - i] = temp; }</pre>	

P 1 1 2 3 4 **Q** 2 3 4 5 5 **R** 1 2 3 4 5 **S** 1 1 1 1 1

PART C · 4 QUESTIONS

C Catch the mistake

7 ONE-TOKEN FIX

This should print 3, the number of rolls that match the roll before them. It crashes instead. Change exactly one token to fix it.

```
1 int[] rolls = {4, 4, 2, 5, 5, 5};
2 int repeats = 0;
3 for (int i = 0; i < rolls.length; i++) {
4     if (rolls[i] == rolls[i - 1]) { repeats++; }
5 }
6 System.out.println(repeats);
```

LINE, OLD TOKEN, NEW TOKEN, AND WHY IT CRASHED

8 TWO TRUTHS AND A LIE

This should check whether any locker code repeats. Two claims are true and one is a lie. Circle the lie and rewrite it.

```
int[] codes = {4, 1, 8, 3};
boolean dup = false;
for (int i = 0; i < codes.length; i++) {
    for (int j = 0; j < codes.length; j++) {
        if (codes[i] == codes[j]) { dup = true; }
    }
}
System.out.println(dup);
```

- **1** It prints true, even though no code repeats.
- **2** Its very first comparison checks codes[0] against itself.
- **3** Starting j at i instead of at 0 would fix it.

THE LIE, REWRITTEN

9 WHO AM I?

The standard array algorithms are playing a party game. Each gives a clue about itself. Name the algorithm.

Clue	I am...
My flag starts at false, and one element that passes flips it to true.	
My flag starts at true, and a single failure can end me early.	
I look at a[i] and a[i + 1], so I stop at length - 1.	
I need a loop inside a loop, and my inner index starts at i + 1.	
I save the first element, slide everyone over, then put it at the end.	
I swap the ends toward the middle and stop halfway.	

**WATCH OUT**

In an indexed loop, `i` and `nums[i]` are both `ints`, so mixing them up compiles fine. `if (i % 2 == 0)` asks whether the **position** is even; `if (nums[i] % 2 == 0)` asks whether the **element** is. Read each `i` aloud as "position" and each `nums[i]` as "the value there".

10**WRAP-UP****REFLECTION**

In one sentence: why does shifting an array to the right need a loop that runs backward?

YOUR ANSWER