

NAME

DATE

PERIOD

UNIT 4 · TOPIC 4.14 · WORKSHEET

Hide and Seek

Track down targets one element at a time, count every look, and catch the searches that quit too soon.

About 30 minutes

9 questions

Cumulative through 4.14

TOPIC

4.14

**DIRECTIONS**

Work each puzzle in pencil. For code, write exactly what prints. A *comparison* is one check of an element against the target. 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?

```
public static int findIndex(int[] values, int target) {
    for (int i = 0; i < values.length; i++) {
        if (values[i] == target) { return i; }
    }
    return -1;
}
// in main
int[] bins = {31, 18, 44, 18, 27};
System.out.println(findIndex(bins, 18) + " " + findIndex(bins, 40));
```

A 1 -1**B** 3 -1**C** 18 -1**D** 1 0



THINK FIRST

A 3-by-4 grid of numbers doesn't contain your target. How many elements must a search check before it can say "not found"? Guess in the margin; question 2 settles it.

2 PENCIL CHECK

A parking lot stores permit numbers in a 2D array. `locate` searches it row by row. For each call, write what it returns and how many comparisons it makes.

```
int[][] lot = { {12, 40, 7, 33},
                {25, 18, 61, 9},
                {50, 7, 14, 2} };
public static String locate(int[][] grid, int target) {
    for (int r = 0; r < grid.length; r++) {
        for (int c = 0; c < grid[r].length; c++) {
            if (grid[r][c] == target) { return "row " + r + ", col " + c; }
        }
    }
    return "not found";
}
```

Call	Returns	Comparisons
<code>locate(lot, 12)</code>		
<code>locate(lot, 33)</code>		
<code>locate(lot, 61)</code>		
<code>locate(lot, 7)</code>		
<code>locate(lot, 2)</code>		
<code>locate(lot, 80)</code>		

3 YOU'RE THE COMPILER

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

```
int[] a = {4, 9, 2};
```

Snippet	Compiles?	Output, crash, or why not
1 <code>int i = 0;</code> <code>while (i < a.length && a[i] != 2) {</code> <code> i++;</code> <code>}</code> <code>System.out.println(i);</code>		
2 <code>int i = 0;</code> <code>while (a[i] != 5 && i < a.length) {</code> <code> i++;</code> <code>}</code> <code>System.out.println(i);</code>		
3 <code>int i = 0;</code> <code>while (i < a.length && a[i] != 5) {</code> <code> i++;</code> <code>}</code> <code>System.out.println(i);</code>		
4 <code>int spot = a.indexOf(9);</code> <code>System.out.println(spot);</code>		



WATCH OUT

In a while search, put the index check **before** the element check: `i < list.length && list[i] != target`. Java skips the right side of `&&` when the left side is false, so the guard protects the array access.

PART B · 3 PUZZLES

B Puzzles

4 CODE TILES

`lastIndexOf` should search **from the end** and return the index of the *last* copy of `target`, or `-1` if it isn't there. Put tiles inside the method so `main` prints the target output. Two tiles are decoys.

```
public static int lastIndexOf(int[] values, int target) {
    // your tiles go here
}
// in main
int[] nums = {7, 5, 8, 5};
System.out.println(lastIndexOf(nums, 5));
System.out.println(lastIndexOf(nums, 7));
System.out.println(lastIndexOf(nums, 9));
```

```
return -1;

for (int i = values.length - 1; i >= 0; i--) {
    if (values[i] == target) { return i; }
}

else { return -1; }

for (int i = values.length - 1; i > 0; i--) {
```

Target output

3

0

-1

YOUR TILES, IN ORDER

5 SNIPPET POOL

Fill each numbered hole with a snippet from the pool so main prints 1, then 3, then -1. Not every snippet is used.

```
import java.util.ArrayList;
public static int find(ArrayList<String> names, String target) {
    for (int i = 0; i < (1); i++) {
        if ((2)) { return i; }
    }
    return (3);
}
// in main
ArrayList<String> crew = new ArrayList<String>();
crew.add("Kim");
crew.add("Omar");
crew.add("Lena");
crew.add("Ty");
System.out.println(find(crew, new String("Omar")));
System.out.println(find(crew, new String("Ty")));
System.out.println(find(crew, "Zed"));
```

THE POOL

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

crew.size()	names.length
names.size() - 1	names.get(i).equals(target)
names.size()	names.get(i) == target
0	-1



FAIR QUESTION

Then why does == sometimes work on strings?

Java may let two identical string literals, like "Zed" and "Zed", share one object, so == can be true by luck. new String(...) always makes a new object. A search that works only by luck is worse than one that always fails, so always use equals.

6 MATCH-UP

Each version below changes one thing in this program. Match each version to what it prints, from the list. Each output is used once.

```
public static int count;
public static int search(int[] v, int target) {
    count = 0;
    for (int i = 0; i < v.length; i++) {
        count++;
        if (v[i] == target) { return i; }
    }
    return -1;
}
// in main
int[] nums = {4, 7, 4, 1};
int spot = search(nums, 4);
System.out.println(spot + " " + count);
```

Version	The one change	Prints
1	None: run it as written.	
2	The loop header becomes for (int i = v.length - 1; i >= 0; i--)	
3	count++; moves to the line just after the if statement.	
4	main calls search(nums, 9) instead.	

P 2 2 **Q** -1 4 **R** 0 1 **S** 0 0

PART C · 3 QUESTIONS

C

Catch the mistake

7

ONE-TOKEN FIX

This program should print 1, then -1. It prints 1 and then crashes with an `ArrayIndexOutOfBoundsException`. Change exactly one token to fix it.

```
1 public static int find(int[] codes, int w) {  
2     for (int i = 0; i <= codes.length; i++) {  
3         if (codes[i] == w) { return i; }  
4     }  
5     return -1;  
6 }  
7 // in main  
8 int[] codes = {3, 8, 6, 1};  
9 System.out.println(find(codes, 8));  
10 System.out.println(find(codes, 5));
```

LINE, OLD TOKEN, NEW TOKEN



WATCH OUT

A search can hand you -1. Check the result before you use it as an index: `codes[find(codes, 5)]` asks for `codes[-1]`, which throws an `ArrayIndexOutOfBoundsException`. Test if `(spot != -1)` first.

8 TWO TRUTHS AND A LIE

This search looks reasonable: if it matches, say where; otherwise, say not found. Two of the claims are true and one is a lie. Circle the lie, then rewrite it so it's true.

```
public static int find(int[] v, int target) {
    for (int i = 0; i < v.length; i++) {
        if (v[i] == target) { return i; } else {
            return -1;
        }
    }
    return -1;
}

// in main
int[] tags = {21, 35, 48, 35};
```

- 1 find(tags, 21) returns 0.
- 2 find(tags, 48) returns 2.
- 3 No matter the target, find compares only one element.

THE LIE, REWRITTEN

9 FACE-OFF

Tonight: Front-to-Back Search and Back-to-Front Search argue about who's the better seeker. Fill in the blanks.

FRONT-TO-BACK I start at index 0. When a value appears twice, I report the _____ copy.

BACK-TO-FRONT I start at index _____ and count down while $i \geq 0$. So I report the _____ copy.

FRONT-TO-BACK Start at Length instead and you crash on your very first look.

BACK-TO-FRONT At least I'm honest when the target isn't there. In an array of 5, I make _____ comparisons.

FRONT-TO-BACK So do I. Neither of us can say "not found" until we've checked _____ element.