

NAME

DATE

PERIOD

UNIT 4 · TOPIC 4.9 · WORKSHEET

Mind the Gap

Walk a list forward, backward, and for-each, and catch the loop that skips.

About 30 minutes

9 questions

Cumulative through 4.9

TOPIC

4.9

**DIRECTIONS**

Work each puzzle in pencil. When code prints, write exactly what it prints; a printed list looks like `[frog, pond]`. Every program starts with `import java.util.ArrayList;`, even where it isn't shown. Some puzzles include pieces you won't need.

PART A · 2 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?

```
ArrayList<String> trail = new ArrayList<String>();
trail.add("oak");
trail.add("elm");
trail.add("yew");
String marks = "";
for (int i = trail.size() - 1; i >= 0; i -= 2) {
    marks = marks + trail.get(i).substring(0, 1);
}
System.out.println(marks);
```

A yo**B** oy**C** yeo**D** y

2 PENCIL CHECK

`pets` holds 5 elements. Each loop body calls `pets.get(i)`. Write the indexes each loop visits, in order. If it crashes, write *crash* where it happens.

Loop	Indexes visited
<code>for (int i = 0; i < pets.size(); i++)</code>	
<code>for (int i = 0; i <= pets.size(); i++)</code>	
<code>for (int i = pets.size() - 1; i >= 0; i--)</code>	
<code>for (int i = 1; i < pets.size(); i += 2)</code>	
<code>for (int i = pets.size(); i > 0; i--)</code>	
<code>int i = 0; then while (i < pets.size()), with i += 3; at the end of the body</code>	



THINK FIRST

Your loop is at `i = 2` and removes the element there. Which element is at index 2 now? Will the loop ever check it after `i++`? Guess in the margin; question 4 settles it.



WATCH OUT

An enhanced for loop over a list of objects hands you a **reference** to each object. So calling a method on the loop variable, like `f.hop()` on each `Frog f`, really does change the objects in the list.

PART B · 4 PUZZLES

B Puzzles**3** CODE TILES

Hopper's snack bar is doubling every price. Put the tiles in order so the program changes the list itself and prints the output below. Three tiles are decoys and aren't used.

```
ArrayList<Integer> prices = new ArrayList<Integer>();  
prices.add(4);  
prices.add(10);  
prices.add(7);
```

```
System.out.println(prices);  
  
}  
  
for (int p : prices) {  
    prices.set(i, prices.get(i) * 2);  
    for (int i = 0; i <= prices.size(); i++) {  
        for (int i = 0; i < prices.size(); i++) {  
            p = p * 2;
```

Target output

```
[8, 20, 14]
```

YOUR PROGRAM

4 SNIPPET POOL

Hopper wants to drop every word shorter than 3 letters, so the program prints [frog , pond]. Fill each numbered hole with a snippet from the pool. Each snippet is used at most once.

```
ArrayList<String> words = new ArrayList<String>();
words.add("go");
words.add("a");
words.add("frog");
words.add("is");
words.add("in");
words.add("pond");
for (int i = (1); (2); (3)) {
    if (words.get(i).length() < 3) {
        words.remove(i);
    }
}
System.out.println(words);
```

THE POOL

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

0	words.size() - 1	words.size()
i < words.size()	i >= 0	i <= words.size()
i++	i--	



FAIR QUESTION

Is a while loop ever better than a for loop for a list?

Yes, when the index shouldn't move the same way on every pass. A while loop can stop as soon as it finds what it's looking for, or move `i` forward only on some passes. Any for loop can be written as a while loop, so pick whichever says what you mean most clearly.

5 FACE-OFF

Tonight: the indexed for loop and the enhanced for loop argue about who's the better way to walk a list. Fill in the blanks.

ENHANCED	I hand you each element in order, so I never run off the _____.
INDEXED	But I know <i>where</i> I am, so <code>set(i, ...)</code> lets me _____ elements.
ENHANCED	True. Assigning to my loop variable changes _____ in the list.
INDEXED	And add or remove while you walk, and you throw a _____.
ENHANCED	Remove going forward and the next element slides _____, so your <code>i++</code> _____ it.
INDEXED	That's why I walk _____ when I remove.

6 YOU'RE THE COMPILER

You are the compiler, then the JVM. `nums` starts as `[3, 6, 9]` each time. Compiles? If not, why not? If so, what prints, and what exception, if any?

Snippet	Compiles?	Output, crash, or why not
<pre>1 for (int i = 0; i <= nums.size(); i++) { System.out.println(nums.get(i)); }</pre>		
<pre>2 for (int n : nums) { n = 0; } System.out.println(nums);</pre>		
<pre>3 for (int n : nums) { nums.add(n); }</pre>		
<pre>4 for (int i = 0; i < nums.length; i++) { System.out.println(nums.get(i)); }</pre>		

PART C · 3 QUESTIONS

C

Catch the mistake

7

TWO TRUTHS AND A LIE

This loop is supposed to remove every temperature below 32. Two of the claims are true and one is a lie. Circle the lie, then rewrite it so it's true.

```
1 ArrayList<Integer> temps = new ArrayList<Integer>();
2 temps.add(40);
3 temps.add(31);
4 temps.add(28);
5 temps.add(55);
6 for (int i = 0; i < temps.size(); i++) {
7     if (temps.get(i) < 32) {
8         temps.remove(i);
9     }
10 }
```

- **1** The loop body runs 3 times.
- **2** The if never checks the 28.
- **3** The test `i < temps.size()` uses the current size on every pass, so nothing is skipped and temps ends as [40, 55].

THE LIE, REWRITTEN

8 ONE-TOKEN FIX

This loop should print the playlist backward: Tides, Glow, Rain. It prints only Tides and Glow. Change exactly one token to fix it.

```
1 ArrayList<String> songs = new ArrayList<String>();
2 songs.add("Rain");
3 songs.add("Glow");
4 songs.add("Tides");
5 for (int i = songs.size() - 1; i > 0; i--) {
6     System.out.println(songs.get(i));
7 }
```

WHAT WENT WRONG, AND YOUR FIX

9 WHO AM I?

Some traversal ideas are playing a party game. Each gives a clue about itself. Name it.

Clue	I am...
I visit every element, in order, with no index variable.	
I'm the last valid index of a list.	
I'm thrown when your index reaches <code>size()</code> .	
I'm thrown when a list changes size during an enhanced for loop.	
I'm the method that replaces the element at index <code>i</code> as you go.	
Walk this way when you remove, so shifts only move elements you've already checked.	