

NAME _____

DATE _____

PERIOD _____

UNIT 4 · TOPIC 4.10 · WORKSHEET

List Surgery

Count, compare, cut, insert, and flip ArrayLists without dropping a single element.

About 30 minutes

9 questions

Cumulative through 4.10

TOPIC

4.10

**DIRECTIONS**

Work each puzzle in pencil. Every program has `import java.util.ArrayList;` at the top. Printing a list with `println` shows its elements in square brackets, like `[4, 7, 2]`. Some puzzles include pieces you won't need.

**THINK FIRST**

You call `remove(1)` on a list holding `[10, 20, 30, 40]`. Which value sits at index 1 afterward? Guess in the margin; question 2 puts that answer to work.

PART A · 3 QUESTIONS

A Warm up your brain

1 PENCIL CHECK

Hopper logged five morning temperatures. Run each standard algorithm on `temps` in your head and write its result.

```
ArrayList<Integer> temps = new ArrayList<Integer>();
temps.add(6); temps.add(-3); temps.add(9); temps.add(4);
temps.add(9);
```

Algorithm on <code>temps</code>	Result
The sum of all the elements	
The average, computed as <code>(double) sum / temps.size()</code>	
The maximum, starting from <code>temps.get(0)</code>	
The minimum, starting from <code>temps.get(0)</code>	
How many elements are greater than 5	
Is at least one element negative? (true or false)	
Are all the elements even? (true or false)	
How many consecutive pairs have a larger second element	

2 YOU'RE THE JVM

This program is supposed to remove every pet name shorter than 4 letters. Run it in your head. What does it really print?

```
ArrayList<String> pets = new ArrayList<String>();
pets.add("ox"); pets.add("emu"); pets.add("yak");
pets.add("lion"); pets.add("ant");
for (int i = 0; i < pets.size(); i++) {
    if (pets.get(i).length() < 4) {
        pets.remove(i);
    }
}
System.out.println(pets);
```

- A** [lion]

C [emu, yak, lion]

B [emu, lion]

D A ConcurrentModificationException

3 FACE-OFF

Tonight: `add(i, value)` and `remove(i)` argue about who's harder to use in a loop. Fill in the blanks.

ADD(I, VALUE)	I squeeze a new element in at index <code>i</code> and push every later element one place to the _____.
REMOVE(I)	I pull out the element at index <code>i</code> and slide every later element one place to the _____. I even _____ the element I took, so you can add it back at the other end and rotate the list.
ADD(I, VALUE)	My trap: insert in front of the element you're checking, and it moves to <code>i + 1</code> . The loop's <code>i++</code> lands right back on it, so the loop runs _____ unless you add an extra _____.
REMOVE(I)	Mine's sneakier. After I remove, the next element slides into index <code>i</code> , and a loop that always does <code>i++</code> _____ it.
ADD(I, VALUE)	And inside an enhanced for loop, neither of us is welcome. Change the list's size there and Java throws a _____.

PART B · 2 PUZZLES

B Puzzles

4 CODE TILES

Coach keeps runner names in an array and lap counts in an ArrayList, matched by index. Eli hasn't run yet, so he has no lap count. Put the tiles in order so the program prints the runners with at least 10 laps. One tile is a decoy.

```
String[] names = {"Ana", "Ben", "Cy", "Dee", "Eli"};
ArrayList<Integer> laps = new ArrayList<Integer>();
laps.add(12); laps.add(8); laps.add(15); laps.add(10);
```

```
System.out.println(fast);
for (int i = 0; i < names.length; i++) {
}
ArrayList<String> fast = new ArrayList<String>();
if (laps.get(i) >= 10) { fast.add(names[i]); }
for (int i = 0; i < laps.size(); i++) {
```

Target output

```
[Ana, Cy, Dee]
```

YOUR PROGRAM

5 SNIPPET POOL

Fill each hole from the pool so the program removes every score below 60 and prints [72, 90]. Each snippet is used at most once, and some aren't needed.

```
ArrayList<Integer> scores = new ArrayList<Integer>();
scores.add(45); scores.add(72); scores.add(58);
scores.add(50); scores.add(90);
for (int i = (1); (2); i--) {
    if (scores.get(i) < 60) {
        scores.(3);
    }
}
System.out.println(scores);
```

THE POOL

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

scores.size()	i >= 0	remove(0)
scores.size() - 1	i > 0	remove(i)
set(i, 0)		



FAIR QUESTION

Why didn't Java warn me that an element was skipped?

Because nothing illegal happened. Every index the loop used was in range, so every call worked. Java checks the rules of the language, not whether your algorithm does what you meant. A skipped element is a logic error, and only a careful trace catches it.

PART C · 4 QUESTIONS

C

Catch the mistake

6

RUNS, CRASHES, OR NEVER ENDS?

Each snippet starts fresh with `nums` holding `[4, 7, 7, 2]`. Decide what happens: write the output, name the exception it throws, or write *never ends*.

Snippet	What happens?
<pre>1 nums.add(nums.remove(0)); System.out.println(nums);</pre>	
<pre>2 for (int n : nums) { if (n > 5) { nums.add(n); } } System.out.println(nums);</pre>	
<pre>3 int rises = 0; for (int i = 0; i < nums.size(); i++) { if (nums.get(i) < nums.get(i + 1)) { rises++; } } System.out.println(rises);</pre>	
<pre>4 for (int i = 0; i < nums.size(); i++) { if (nums.get(i) > 5) { nums.add(i, 0); } } System.out.println(nums);</pre>	
<pre>5 nums.remove(nums.size() - 1); nums.add(0, 0); System.out.println(nums);</pre>	

7 TWO TRUTHS AND A LIE

This loop reverses a list in place. Two claims about it are true and one is a lie. Circle the lie, then rewrite it so it's true.

```
ArrayList<String> rainbow = new ArrayList<String>();
rainbow.add("red"); rainbow.add("orange");
rainbow.add("yellow"); rainbow.add("green");
rainbow.add("blue");
for (int i = 0; i < rainbow.size() / 2; i++) {
    String temp = rainbow.get(i);
    rainbow.set(i, rainbow.get(rainbow.size() - 1 - i));
    rainbow.set(rainbow.size() - 1 - i, temp);
}
System.out.println(rainbow);
```

- **1** The loop body runs exactly 2 times.
- **2** "yellow" is still at index 2 after the loop.
- **3** Changing `rainbow.size() / 2` to `rainbow.size()` would still reverse the list; it would just do a few extra swaps.

THE LIE, REWRITTEN**WATCH OUT**

`==` on two Integer objects compares *references*, not values. If a list holds two 500s, `big.get(0) == big.get(1)` can be false. Copy each element into an `int` variable first (`int a = big.get(0);`), and `==` compares the numbers.

8 ONE-TOKEN FIX

This duplicate check crashes with an `IndexOutOfBoundsException`. Change exactly one token so it prints `Duplicate: true` for this list and would still print `Duplicate: false` for a list with no repeats.

```
1 ArrayList<Integer> ids = new ArrayList<Integer>();
2 ids.add(301); ids.add(118); ids.add(254); ids.add(118);
3 boolean found = false;
4 for (int i = 0; i < ids.size(); i++) {
5     for (int j = i + 1; j <= ids.size(); j++) {
6         int a = ids.get(i);
7         int b = ids.get(j);
8         if (a == b) {
9             found = true;
10        }
11    }
12 }
13 System.out.println("Duplicate: " + found);
```

LINE, OLD TOKEN, NEW TOKEN

9 WHO AM I?

Some `ArrayList` algorithms are playing a party game. Each gives a clue about itself. Name it.

Clue	I am...
I move every element one place, and the one that falls off one end comes back in at the other.	
I swap pairs from the outside in, and I stop halfway.	
I'm a brand-new list of only the elements that passed a test. The original is untouched.	
I start <code>false</code> and flip to <code>true</code> the first time I see a match. Nothing flips me back.	
I'm the loop bound that keeps <code>get(i + 1)</code> in range when you visit consecutive pairs.	