The Hidden Zero and the Case-Sensitive Search

One program, two clean-looking results, both wrong: a padded array quietly skews an average, and a contains check fails on a value that's clearly there.

Automation Foundations · ashishautomationlabs.store

This program compiles, runs, and prints two believable results. Both are wrong, and both are the silent kind -- no error, no crash. Read first, then run.

```
import java.util.ArrayList;
import java.util.List;

public class BugHunt5 {
    public static void main(String[] args) {
        int[] scores = new int[5];
        scores[0] = 80;
        scores[1] = 90;
        scores[2] = 100;
        int total = 0;
        for (int i = 0; i < scores.length; i++) {
            total += scores[i];
        }
        double average = (double) total / scores.length;
        System.out.println("Average score: " + average);

        List<String> ran = new ArrayList<>();
        ran.add("LoginTest");
        ran.add("CartTest");
        System.out.println("Ran logintest? " + ran.contains("logintest"));
    }
}

Average score: 54.0
Ran logintest? false
```

Two clean-looking lines, two defects:

1. The array was created with `new int[5]`, but only three scores were filled. The other two boxes still hold their default `0`. The loop dutifully sums all five, giving 270, and divides by `scores.length`, which is 5 -- so the average is 54.0. The real average of the three actual scores is 90.0. The two hidden zeros dragged it down, and nothing warned you. This is the exact danger of default values: an empty box is not the same as no box, and `.length` counts the boxes, not the real data. A list would never do this, because it only ever contains what you added.

2. `ran.contains("logintest")` **returns** `false`**, and the run did include** `LoginTest`**.** The problem is case: the list holds `"LoginTest"` with capitals, and `contains` compares text exactly, letter for letter. `"logintest"` and `"LoginTest"` are different text, so the honest answer is `false`. This is the same case-sensitivity that has been true since Chapter 1, now hiding inside a list search. When a `contains` check surprises you, suspect the exact spelling and case first.

Three lessons in one program, and zero errors. A program that runs and prints is not a program that is correct -- and the only defense is to know your data well enough to predict the answer before you read it.

Checkpoint

- What are the three fixed properties of an array, and which one sends you to a list?
- You write `int[] counts = new int[3];` and print `counts[0]` before assigning

  anything. What prints, and why?

- Give the list method that matches each array idea: the count; reading position

  `i`; asking "is this value present?"

- In `List<String> names = new ArrayList<>();`, what does the `<String>` part do?
- State the one-line rule for choosing an array versus an `ArrayList`.
- A list search with `contains(...)` returns `false` for a value you're sure is

  there. Name the two things to check first.

You can hold many values now and work with them as a set, not one variable at a time. You know the array completely -- its numbered boxes, its default values, its fixed size -- and the running-value pattern that sums, counts, and finds the largest. More importantly, you know when the array runs out of room, and you can reach for an `ArrayList` that grows as your test run does.

There is a crack in your knowledge, though, and this chapter widened it on purpose. Twice now -- the `.equals` in `results[i].equals("FAIL")`, and the case-sensitive `contains` in the bug hunt above -- your work has quietly depended on how Java decides that two pieces of text are equal. You've been told since Chapter 2 to use `.equals` for text and warned that `==` has a trap, always with a promise that the full picture was coming. That picture is about what a `String` really is, where it lives, and why two values that look identical can still be treated as different. It's one of the most asked questions in any Java interview -- and it's Chapter 6.