What a Method Can (and Cannot) Change

Java always passes a copy of what's in the box. A helper that collects failures into a list you passed in will work; one that tries to reset a counter you passed in never will.

Automation Foundations · ashishautomationlabs.store

Here is a subtlety that catches almost everyone, and it decides whether your helper methods actually work. Java always passes a copy of the value in the variable. Watch what that means for a number:

```
public class M5 {
    static void tryBump(int count) {
        count = count + 1;
        System.out.println("  inside method : " + count);
    }
    public static void main(String[] args) {
        int retries = 1;
        System.out.println("before call   : " + retries);
        tryBump(retries);
        System.out.println("after call    : " + retries);
    }
}
```

```
before call   : 1
  inside method : 2
after call    : 1
```

The method changed its copy to `2`, and `retries` back in `main` is still `1`. The method received the value, not the variable. If you want a new value out of a method, you must return it and assign it -- that is what return values are for.

Now the same experiment with an object, and the result flips:

```
public class M6 {
    static void addFailure(List<String> failures, String id) {
        failures.add(id);
    }
    public static void main(String[] args) {
        List<String> failures = new ArrayList<>();
        addFailure(failures, "TC-1001");
        addFailure(failures, "TC-1004");
        System.out.println("Failures: " + failures);
        System.out.println("Count   : " + failures.size());
    }
}
```

```
Failures: [TC-1001, TC-1004]
```

```
Count   : 2
```

This time the change survived. Both results follow one rule, and the memory picture from Chapter 2 explains it: Java copies what is in the box. For an `int`, the box holds the number, so the method gets a copy of the number. For a list, the box holds an arrow, so the method gets a copy of the arrow -- and a copy of an arrow points at the very same object. The method cannot make your variable point somewhere else, but it can change the object it points at.

For a tester this is a practical rule, not trivia. A helper that collects failures into a list you passed in will work. A helper that tries to "reset" a counter you passed in will not. When in doubt, return the new value.

Copying the box also explains a word you have typed since page one without asking why: `static`. It decides whether a method needs an object at all -- or whether it can run with nothing but what you pass in.