Catching the Right Thing, in the Right Order

Multiple catch blocks run most-specific-first, exactly like an else-if chain. And catching plain Exception everywhere isn't tidy error handling -- it's error hiding.

Automation Foundations · ashishautomationlabs.store

You can write more than one `catch` for a single `try`, and Java uses the first one whose type matches:

```
int[] codes = {200, 404};
try {
    System.out.println(codes[5]);
} catch (ArrayIndexOutOfBoundsException e) {
    System.out.println("Index problem : " + e.getMessage());
} catch (Exception e) {
    System.out.println("Something else: " + e.getMessage());
}

Index problem : Index 5 out of bounds for length 2
```

Order matters, and it works exactly like the `else if` chain from Chapter 3: most specific first, most general last. `Exception` is the general type that matches almost anything, so it belongs at the bottom. Put it first and the specific catches below it could never run -- the same dead-code trap from Chapter 3, wearing new clothes.

There is a temptation worth naming early: why not just catch `Exception` everywhere and be done with it? Because a failing test is the product. Your suite exists to report problems, and an exception you catch and hide is a problem nobody hears about. Catching plain `Exception` everywhere is easy and looks tidy, and it is usually a mistake, because it catches problems you did not anticipate and were not ready to handle. Catch the specific thing you expect and know how to report. If something else goes wrong, you generally want to know loudly.

Where this discipline actually gets tested

A data-driven suite reads three hundred rows. Each row is parsed inside `try { ... } catch (Exception e) { }` -- an empty catch block. The suite is green and it never crashes on bad data, which looks like exactly what was wanted. But it is green because it cannot report anything: if forty rows have bad data, the suite skips forty rows silently and still prints PASS.

The fix is not to stop catching, and it is not to let one bad row end the whole run either -- those aren't the only two options. Catch the specific parse failure, log the row number and the value, count the failures, and fail the suite at the end if the count is above zero. You keep running, and

you still tell the truth. An empty catch block is not error handling -- it is error hiding. Catch narrowly, report every skip with its row and value, and make the run fail when data was skipped. A suite that hides failures is worse than no suite, because people trust it.

Catching precisely handles the failures you expect. Some work, though, has to happen no matter what -- expected failure, unexpected failure, or none at all. That is a different guarantee, and it has its own keyword.