Excel and JSON: Honest Notes

Whatever the format -- properties, CSV, Excel, JSON -- the discipline is the same: read it inside try-with-resources, convert values carefully, turn rows into objects, and fail loudly when the file is missing.

Automation Foundations · ashishautomationlabs.store

Two formats you will certainly meet need libraries that are not part of Java itself.

Excel is read with Apache POI. You add it as a Maven dependency, then open the workbook, choose a sheet, and walk rows and cells. The shape of the code is short: open a `Workbook` from a file stream, get a sheet by name or index, loop the rows, and read each cell. Convert cell types carefully, because a number typed into a spreadsheet is not text, and reading it as text throws.

JSON is handled with a mapper such as Jackson or Gson. The pattern that matters is that a mapper converts JSON directly into your own classes from Chapter 7. You define a class whose fields match the JSON keys, and one line turns a payload into an object.

A word of honesty, carried over directly from the book this site adapts: every other code sample so far was actually run, and the output you saw was real. Excel and JSON examples are the exception -- they need a network connection to fetch a library, which isn't available here. So this article describes the approach and the shape of the code rather than claiming verified output for it, on the same principle the book itself states plainly: better to say so than to print something that was never run.

What carries across from this chapter is the part that matters most: whatever the format, you read it inside `try`-with-resources, you convert the raw values with the care Chapter 14 taught, you turn rows into objects, and you fail loudly when the file is missing. The library changes; the discipline does not.

So which format, for which job? Properties for configuration -- a handful of settings, one per line. CSV for tabular test data, especially when non-developers maintain it. Excel when the business already owns the data in that form and will not move. JSON when the shape is nested, such as an API request body. Match the format to who edits it and what shape it is.

And where should any of these files actually live in a project? Under `src/test/resources`, which puts them on the classpath so they travel with the build rather than depending on a folder that happens to exist. Reading from the classpath also avoids the relative-path problem entirely, because the file is found relative to the build rather than to wherever the process was started.

Every idea in this chapter -- reading, writing, configuring, loading rows into objects, failing loudly -- compounds into one discipline. The closing bug hunt shows exactly what happens when the loudest part of that discipline gets skipped.