Light Regular Expressions

matches and replaceAll cover most of what a tester needs from regular expressions -- validating an id's shape, or stripping everything that isn't a digit.

Automation Foundations · ashishautomationlabs.store

A regular expression is a pattern language for text. You do not need to master it, and for test automation a small vocabulary covers most needs. `matches` asks whether a whole String fits a pattern:

```
String[] inputs = {"DEF-101", "def-101", "DEFECT-101", "DEF-9999"};
for (String in : inputs) {
    boolean ok = in.matches("DEF-\\d{3}");
    System.out.println(in + " -> " + ok);
}
```

```
DEF-101 -> true
def-101 -> false
DEFECT-101 -> false
DEF-9999 -> false
```

The pattern `DEF-\d{3}` reads as: the literal text `DEF-`, then exactly three digits. `\d` means a digit, and `{3}` means exactly three of them. In Java source you write `\\d` because the backslash itself must be escaped. All three rejections are correct: wrong case, wrong prefix, four digits instead of three.

`replaceAll` uses the same pattern language to strip or transform:

```
String text = "Order ORD-4471 shipped";
String digits = text.replaceAll("[^0-9]", "");
```

```
digits only: 4471
```

`[^0-9]` means "any character that is not a digit," so replacing all of them with nothing leaves the number. That one line handles a great many scraping problems.

A word of restraint: regular expressions are powerful and become unreadable quickly. For validating an id format or pulling digits out of a label they are ideal. For parsing something with real structure, such as JSON or HTML, use a proper parser instead. A regular expression will appear to work, then fail on the first input you did not imagine.

Splitting and pattern-matching both assume the text is well-formed enough to process. Real test data is not always that cooperative -- sometimes the cell is blank, or holds `"n/a"` instead of a number.