while and do-while

When you only know 'until what,' use while -- with a safety limit so a retry cannot become a hang. Use do-while when you must check at least once, like polling a service.

Automation Foundations · ashishautomationlabs.store

Sometimes you cannot answer "how many times?" -- you can only answer "until what?" Wait until the page loads. Retry until the login succeeds. Poll until the service is up. That is a `while` loop: it repeats as long as a condition stays true, and it is the shape behind every wait in test automation.

```
public class L4 {
    public static void main(String[] args) {
        int attempts = 0;
        boolean loggedIn = false;
        while (!loggedIn && attempts < 3) {
            attempts++;
            System.out.println("Login attempt " + attempts);
            loggedIn = (attempts == 3);
        }
        System.out.println("Logged in: " + loggedIn + " after " + attempts + "
attempts");
    }
}
```

```
Login attempt 1
Login attempt 2
Login attempt 3
Logged in: true after 3 attempts
```

Notice the two-part condition: `!loggedIn && attempts < 3`. In plain English: keep trying while we are not yet logged in **and** we still have attempts left. That second part is a **safety limit**. It is not optional -- it is the difference between a retry and a hang. A `while` that can never make its condition false runs forever.

do-while: when you must run at least once

A plain `while` checks its condition first, so it can run **zero** times -- if the condition is already false, the body never runs. Occasionally you need the opposite: run the body once, then decide whether to repeat. That is `do-while`, and polling a service is the natural example -- you must check at least once to know anything:

```
public class L5 {
    public static void main(String[] args) {
        int response;
        int poll = 0;
        do {
```

```
            poll++;
            response = (poll < 2) ? 503 : 200;
            System.out.println("Poll " + poll + ": HTTP " + response);
        } while (response != 200);
        System.out.println("Service is up after " + poll + " polls");
    }
}
```

```
Poll 1: HTTP 503
Poll 2: HTTP 200
Service is up after 2 polls
```

The condition sits at the bottom, so the body always runs before it is tested.

> *"Check, then maybe do" is a* `while`*. "Do, then check whether to repeat" is a* `do-while`*. In day-to-day automation you will reach for* `while` *far more often, but the day you need "run at least once,"* `do-while` *is exactly right.*

Choosing without overthinking

Ask: do I know the number of passes? If yes -- a count, a fixed list, every row -- use `for`. If you only know the stopping condition -- until it loads, until it succeeds -- use `while`. When both would work, `for` is usually clearer because its counter and limit sit together on one line.

You can retry with a safety limit and poll until a service is up. The next lesson is the two bugs that cost testers the most time -- off-by-one and the infinite loop -- plus `break` and `continue` for fail-fast and skip-row control.