Booleans Are the Shape of Every Verdict

Every test case ends in a judgment. In Java that judgment is a boolean -- true or false, nothing else -- and every assertion library is built on that one fact.

Automation Foundations · ashishautomationlabs.store

Think about what a tester actually does, reduced to its simplest form.

You set something up. You perform an action. You observe a result. And then you **judge**. Pass or fail. Matches or does not match. Go or no-go.

Every test case you have ever written ends in a judgment. Every defect you have ever raised began with one. That judgment is the product of your job. The setup steps are only the cost of getting there.

Here is the good news, and it is bigger than it first sounds. In Java, that judgment has a name, a shape, and exactly one rule. The name is the **boolean**. The shape is the `if` statement. And the rule is this: a judgment is any expression that produces `true` or `false` -- nothing else.

That is the whole idea. Everything after this lesson is you learning to write judgments so exact that a machine can make them for you, a thousand times a night, while you sleep.

A condition produces a boolean

You met `boolean` in the types chapter as a type that holds only `true` or `false`. Now you see why it is the most important type in testing.

A **condition** is any expression that produces a boolean. You build conditions with comparison operators:

Operator | Reads as | Example

`==` | is equal to | `statusCode == 200`

`!=` | is not equal to | `statusCode != 500`

`>` | is greater than | `openDefects > 5`

`<` | is less than | `responseMs < 400`

`>=` | greater than or equal | `attempts >= 3`

`<=` | less than or equal | `pageLoad <= 2.0`

Every one of those produces a boolean. Which means you can store the result in a variable and print it:

```
public class A {
    public static void main(String[] args) {
        String expected = "Welcome, Priya!";
        String actual = "Welcome, Priya!";
        boolean passed = expected.equals(actual);
        System.out.println("Check passed? " + passed);
        int statusCode = 200;
        System.out.println("Is 200? " + (statusCode == 200));
        System.out.println("Not 500? " + (statusCode != 500));
        System.out.println("Under 400? " + (statusCode < 400));
    }
}
```

Predict the four lines, then run it:

```
Check passed? true
Is 200? true
Not 500? true
Under 400? true
```

Look closely at `boolean passed = expected.equals(actual);`. That single line is a test result -- not a report of a test result, the result itself, held in a variable, ready to be judged. Every assertion library in every automation framework is, at heart, doing exactly this and then deciding what to do about it.

> *Predict each printed boolean, given* `int openDefects = 7;`*: (a)* `openDefects > 5` *(b)* `openDefects == 5` *(c)* `openDefects != 0`

>

> *Answers: true, false, true. Every comparison produces a boolean, and nothing else.*

You can hold a verdict in a variable. The next lesson is the deciding: `if` / `else`, and the one-character trap that runs your suite on a red build with no error at all.