Interview Questions: Strings

Six questions on String comparison, immutability, StringBuilder, safe number conversion, and the split(".") trap -- the strong answers a tester gives after actually running the code once.

Automation Foundations · ashishautomationlabs.store

Q23 · ●●● -- How do you compare two Strings in Java?

Weak answer: "With `.equals()`, not `==`."

Strong answer: "With `.equals()` for exact comparison, or `equalsIgnoreCase()` when case should not matter -- which is often the right choice for statuses and labels from a page. `==` compares whether they are the same object, so it can return `true` for identical literals and `false` for text built at run time, which makes it dangerously unreliable. The other habit I keep is putting the expected value first -- `"PASS".equals(actual)` -- because that survives `actual` being `null` instead of throwing."

Follow-up: *"Why does the expected-first version not throw on `null`?"*

Where in this book: Chapter 6.

Q24 · ●●■ -- What does it mean that Strings are immutable?

Weak answer: "They cannot be changed."

Strong answer: "Once a String exists, its contents can never change. Every method that looks like it modifies one -- `trim`, `toUpperCase`, `replace` -- actually returns a brand new String and leaves the original untouched. The practical consequence catches people constantly: writing `value.trim();` on its own line does nothing at all, because the result was discarded. You have to assign it. The benefit is that Strings are safe to share between threads and safe as Map keys, since nothing can alter them underneath."

Follow-up: *"So what do you use when you are building text in a loop?"*

Where in this book: Chapter 14.

Q25 · ●●■ -- Which String methods do you use most in test automation?

Weak answer: "`trim`, `split`, and `contains`."

Strong answer: "`trim` first, because text scraped from a page almost always carries whitespace you cannot see. Then `contains`, `startsWith`, and `equalsIgnoreCase` for assertions, `split` for

parsing CSV rows, `replace` for stripping currency symbols and separators, and `toLowerCase` for normalizing before comparison. I also print values inside quote marks while debugging, so invisible spaces show up -- `' DEF-101 '` and `'DEF-101'` look identical in a log otherwise."

Follow-up: *"You scrape* `Total: Rs 1,299.00` *and need the number* `1299`*. Walk me through it."*

Where in this book: Chapter 14.

Q26 · ●●■ -- What is `StringBuilder` for?

Weak answer: "It is a mutable String."

Strong answer: "It is a mutable text buffer, for when you are assembling text rather than just joining two values. Because Strings are immutable, every `+` in a loop creates and discards another object -- at three rows that is irrelevant, at three thousand it is real waste. `StringBuilder.append` modifies one buffer in place and `toString()` produces the finished text at the end. My rule is simple: joining a couple of values, use `+` because it reads better; building inside a loop, use a `StringBuilder`."

Follow-up: *"Is* `StringBuffer` *different?"*

Where in this book: Chapter 14.

Q27 · ●●■ -- How would you convert a String to a number safely?

Weak answer: "Use `Integer.parseInt()`."

Strong answer: "`Integer.parseInt` or `Double.parseDouble`, but never on raw test data without protection, because anything that is not a clean number throws `NumberFormatException` and ends the run. Real data has blank cells and values like `'n/a'`. So I check for `null` or blank first, clean the text -- strip commas and currency symbols -- then parse inside a `try`/`catch`. The important part is that the catch reports the value it could not use and does not silently return zero, otherwise the suite computes a wrong total and looks fine."

Follow-up: *"What is wrong with returning* `0` *in the catch block?"*

Where in this book: Chapter 14, with exception handling in Chapter 9.

Q28 · ●■■ -- Why does `"a.b.c".split(".")` return nothing?

Weak answer: "Because the dot needs escaping."

Strong answer: "Because `split` takes a regular expression rather than a plain separator, and in that language a dot matches any character. So it splits on every character and there is nothing left, giving zero fields. Escaping it as `"\\."` means a literal dot and gives the three fields expected. It is worth knowing because the failure is silent -- you get an empty array, then an array

index error somewhere else, and the real cause is three lines earlier. The same applies to a pipe separator, which is common in test data files."

Follow-up: *"Which other characters would you need to escape?"*

Where in this book: Chapter 14.