Put the Known Value First

Calling .equals on a variable that might be null crashes with a NullPointerException. Calling it on a literal never can -- so the literal goes first, always.

Automation Foundations · ashishautomationlabs.store

There is a second defect hiding near the equality trap, and it's a crash rather than a false pass. `.equals` is called **on** an object, so if that object is `null`, the call fails with the `NullPointerException` you met in Chapter 3. Watch the order carefully:

```
public class Eq4 {
    public static void main(String[] args) {
        String input = null;
        System.out.println("Safe check: " + "YES".equals(input));
    }
}
```

```
Safe check: false
```

No crash. The literal `"YES"` is never `null`, so calling `.equals` on it is always safe, and it simply reports that `null` is not equal to `"YES"`. Now reverse the order, with `input` on the left:

```
System.out.println("Result: " + input.equals("YES"));
```

```
Exception in thread "main" java.lang.NullPointerException:
Cannot invoke "String.equals(Object)" because "<local1>" is null
```

Same comparison, reversed order, and now a crash -- because you called `.equals` on a `null` arrow. The fix is a free habit: when you compare a variable against a known value, put the known value on the left. `"YES".equals(input)` can never crash, whatever `input` turns out to be. This is the same null-first discipline from short-circuit evaluation in Chapter 3, wearing new clothes.

Why this isn't paranoia

A test reads a value from a config file and checks it: `configValue.equals("enabled")`. That reads fine -- if the config says `enabled`, the check passes. Until the key is missing and `configValue` comes back `null`. Then `.equals` is called on `null` and the whole test crashes, before it can even report a real result.

"The config always has that key" is true today. The day someone ships a config without it, you'd rather the test say "not enabled" than throw a `NullPointerException` that looks like a framework bug. Flip it: `"enabled".equals(configValue)`. Same check, and it can never crash. It costs nothing and removes a whole class of crash -- interviewers notice when you write it that way by habit.

The four questions this chapter answers

All frequent, all fair game in an interview:

- **What's the difference between `==` and `.equals`?** `==` compares references -- object identity for objects, values for primitives. `.equals` compares contents. Use `.equals` for text.
- **Why does `==` sometimes return `true` for two Strings?** The String pool reuses identical literals, so both arrows point at one object.
- **What does `new String("x")` do differently from `"x"`?** It forces a separate object outside the pool, so `==` against the literal is `false`.
- **How do you avoid a `NullPointerException` when comparing a possibly-null String to a literal?** Put the literal first: `"literal".equals(value)`.

Said plainly, those four answers mark you as someone who understands the language, not just uses it.