Optional: A Value That Might Not Be There

Optional makes a possibly-missing value visible to the compiler and the reader instead of letting it arrive as null at some distant, unrelated line.

Automation Foundations · ashishautomationlabs.store

Chapter 6 taught you that a missing value comes back as `null`, and that using it produces a `NullPointerException` at some distant line. `Optional` is a container that makes the possibility explicit, so the compiler and the reader both see it. Stream searches return one:

```
Optional<String> found = tests.stream().filter(t -> t.startsWith("Cart")).findFirst();
Optional<String> missing = tests.stream().filter(t -> t.startsWith("Pay")).findFirst();

present? : true
value    : CartTest
present? : false
orElse   : no matching test
done
```

The useful methods are few. `isPresent()` asks whether there is a value. `get()` returns it -- and throws if there is not, so never call it without checking. `orElse(fallback)` returns the value or your default, which is usually what you want. And `ifPresent(...)` runs a `Consumer` only when a value exists, which is why the "never printed" line in that output was never printed.

The discipline `Optional` buys you is a habit, not a guarantee: it forces the question "what if this is absent?" at the moment you write the code, instead of at 2 a.m. when a pipeline fails. Use it as a return type for a search that may find nothing. Do not use it for fields or method parameters, where it adds ceremony without much benefit.

Every tool in this chapter so far has assumed the stream or the pipeline is the right shape for the job. It usually is -- but not always, and knowing when to say no is worth as much as knowing the syntax.