Configuration in a Properties File

A properties file supplies the values that change per environment, and a missing key returns null exactly like a Map -- which is why the two-argument fallback form is almost always what you want.

Automation Foundations · ashishautomationlabs.store

A `.properties` file is Java's built-in format for configuration: one `key=value` per line. It is the standard home for the values that change between environments and machines.

```
Properties config = new Properties();
try (InputStream in = Files.newInputStream(file)) {
    config.load(in);
}
```

```
base.url : https://uat.shopcart.internal
browser  : chrome
timeout  : 30s
missing  : null
fallback : http://localhost:4444
```

Three things to take from that output. `getProperty` returns the value as text, so a timeout must be converted with `Integer.parseInt` -- the safe-parsing discipline from Chapter 14 applies here too. A key that is not present returns `null`, exactly like the Map lookup in Chapter 10, and with the same consequence if you use it without checking. And the two-argument form supplies a fallback, which is almost always what you want for optional settings.

The keys use dots to group related settings, which is convention rather than syntax. Combine this with Chapter 13's enums and constants and the picture is complete: the enum defines the fixed set of environments, the properties file supplies the values for the one you are running against, and no test class contains a URL.

Should test data ever be hard-coded, after all this? Yes, sometimes. A unit-style check with two obvious inputs is clearer written in place. Externalize when the data is a set that grows, when someone outside the team maintains it, or when it changes per environment. Moving two values into a properties file or CSV adds ceremony without benefit.

Configuration answers "what environment am I running against." The next question is different: how do many rows of test data -- not one setting -- turn into the objects your test methods actually use.