Lesson 6 Answers

Configuration

1 - Reflection questions

1.1 - Filtering scenarios

What are the three main ways to filter the scenarios that Cucumber will run?

1. The `--tags` command-line argument
2. The `file:line` syntax when specifying features files at the command-line
3. The `--name` command-line argument

1.2 - Why filter scenarios?

Why would you want to filter scenarios? What's wrong with running all of them, all the time?

There are various reasons. The most common one is that you want to focus on a specific scenario that you're working on right now. You're only interested in feedback from that specific scenario, and seeing the results of others would be distracting.

Another reason is time. If your whole test suite takes a long time to run, one solution can be to use tags to select a sub-set of your scenarios to give you quicker (but less complete) feedback about the quality of a build.

Another reason is reliability. If you have a scenario that fails intermittently, you might want to use a tag to exclude that scenario from your regular build.

One more reason might be if you're using the same set of scenarios to test multiple devices. For example, we know a TV network who use Cucumber to test their smart-TV apps. Their app has to work on many different TV and set-top box devices, each with their own quirks. Not all their scenarios work on every device, so they use tags like `@not-samsung-model-xyz` tags to *tag out* scenarios that won't work on certain devices. This way, they can use the same set of feature files to describe the behaviour for all the different devices.

1.3 - Tag Scope

What's the difference between tagging one scenario in a feature, and tagging a whole feature?

A tag on the whole feature is the same as tagging each of the scenarios in that feature.

1.3 - Tag Expressions

How do you tell Cucumber not to run scenarios with a specific tag?

You use a tilde (~) character, like this:

```
cucumber --tags ~@todo
```

1.4 - Abbreviated options

Many of the command-line options have a short and a long version. For example `cucumber --tags @focus` and `cucumber -t @focus` are identical.

Which other options that we described can be shortened? Use `--help` to guide you.

We leave this as an exercise for the reader.

1.5 - Formatter Plugins

Cucumber can print reports in many different formats. How might this help you build trust with the non-technical stakeholders on your team?

From a non-technical person's point of view, feature files can disappear and become *just something for the testers* or *just something for the programmers* once they are checked-in to souce control. When you're a technical person still working with the features every day, it's easy not to notice this. Making an effort to keep feature files visible, especially seeing the results as they run as tests against your software, helps everyone to understand why your team are making the investment of time in sitting together to write scenarios before coding.

1.6 - Strict mode

What's the purpose of strict mode? What kind of mistakes could it catch?

Strict mode is designed to help catch mistakes you might have made. These mistakes typically take two different forms:

1. You’ve changed the wording in a Gherkin step, updated the corresponding step definition to match, but forgotten to look for any other Gherkin steps that were using the same step definition. Those other steps, still using the old wording, are now undefined.
2. You’ve accidentally committed and pushed a half-finished scenario (with a pending step) into your team’s master or trunk branch.

Strict mode will catch either of these mistakes.