COURSEWARE

Setting up Cucumber

Contents

Overview

There is no Overview for this module.

Tutorial

In the following steps, we'll be looking at installing Cucumber and producing your first test runner using a feature file.

Installation

Maven Dependency

`Cucumber` is a dependency that can work in conjunction with other dependencies that require the same version.

With this in mind it is worth setting up a property to help us track our version more centrally in our pom.xml:

```
<properties>
    <cucumber.version>6.8.1</cucumber.version>
</properties>
```

We can now reference this variable above in our POM file and import Cucumber version `6.8.1` by inputting the following dependency.

```
<dependency>
    <groupId>io.cucumber</groupId>
    <artifactId>cucumber-java</artifactId>
    <version>${cucumber.version}</version>
    <scope>test</scope>
</dependency>
```

> Notice the use of `${cucumber.version}` in the above two examples.

Junit Integration

Whilst we have pulled in the `Cucumber` framework, we also need another artifact so that Cucumber can integrate with JUnit.

To integrate Cucumber with Junit, add the following dependency:

```
<dependency>
    <groupId>io.cucumber</groupId>
    <artifactId>cucumber-junit</artifactId>
    <version>${cucumber.version}</version>
    <scope>test</scope>
</dependency>
```

`cucumber-java`/`cucumber-java8` and `cucumber-junit` should be all we need to set up our project. To support development with Cucumber, we want to upgrade eclipse with a plugin to help our IDE understand the `feature` files associated with Cucumber.

> Optional Lambda Import:

To use `lambda functionality` in your tests, consider using the following import:

▶ expand import

Eclipse Plugin

To provide Eclipse with support for Cucumber, follow these steps:

Find the `Help` option in Eclipse at the top of the application and select the following:

> Help -> Eclipse Marketplace -> Search 'Cucumber' ->

Plugin: `Cucumber Eclipse Plugin 1.0.0.202005150629 (or higher)`.

After installing the application restart your Eclipse and proceed onward to create a feature file.

Setting up Our First Cucumber Test

When we create a Cucumber test suite we need three distinct requirements:

- Feature File/s
- Junit Runner
- Step definition - *(like Junit Test Suites)*

In the content below, we will look at each requirement and how to implement them.

Create a Feature File

We typically create feature files in the `test resources` folder, so let us create a `cuke` sub-folder here to hold your feature files.

Inside the `cuke` folder, `right click -> create new file` and name this file `test.feature`.

> Copy the block below and save it in the feature file we just created.

```
Feature: Test

This is an example cucumber-gherkin feature file.

  Scenario: Google Search term Kitten
    Given a circumstance
    When I perform an action
    Then I should be able to observe a tangible result
```

You should be able to right-click this file and `run as a cucumber feature file`; receiving the following output:

▶ expand

Creating a Junit Test Runner

When creating Cucumber tests, we can create a singular test that will allow us to run all the associated features.

Let us create the java class `CukeTestRunner` in `com.qa.app.cuke` of the `src/test/java` folder. This test can be used to execute all our future Cucumber tests.

```
@RunWith(Cucumber.class)
public class CukeTestRunner {

}
```

Using `@RunWith` will let us tell Junit to utilise Cucumber and follow its guidance on test execution. Unfortunately, we still need to add a little more.

We need three major features to get Cucumber tests to work:

- A Cuke Runner *(see above)*
- Feature File *(with at least one Scenario)*
- Step Definitions *(JUnit tests with @Given, @When, and @Then methods)*

We can now run the `CukeTestRunner` class; tying together the `Feature Files` with their respective `Step Definitions` tests and generating some recommended methods.

Linking Our Test Runner with Features

Using the annotation `@CucumberOptions()`; we can locate the feature files and step definitions to run.

To target the feature file we can use `@CucumberOptions(features = "src/test/resources/cuke")`, this should allow us to run the feature files as JUnit tests and generate methods as part of the following console output:

```
io.cucumber.junit.UndefinedStepException: The step "a circumstance" is undefined.
You can implement it using the snippet(s) below:
...
...
<<<your generated step methods here>>>
...
...
```

All we need to do now is tie our runner and feature file with our tests; for that, we need to write our `Step Definitions` using the methods generated above.

Creating a Step Definition

When we run the `CukeTestRunner` class it will automatically attempt to scan for step definitions stored in the adjacent or child package of the current `com.qa.app.cuke` package.

It is generally good practice to create a `cuke` package, then create a package inside of `cuke` called `stepdefs`:

```
src/test/java:
    ---> com.qa.app.cuke
       | - > CukeTestRunner.java
       |
       | - > com.qa.app.cuke.stepdefs
           | - > ExampleFeature.java
```

We can then store all our tests in the `stepdefs` tying them to our feature files using our test runner.

> There is no need to write `Step Definitions` from scratch as Cucumber can generate this for us!

Let us create a test class called `ExampleFeature` and copy generated steps created by Cucumber in our failure trace or from running our feature file, they should appear similar to the following:

For example:

```
@Given("a circumstance")
public void a_circumstance() {
    // Write code here that turns the phrase above into concrete actions
    throw new io.cucumber.java.PendingException();
}
```

Notice that each method created includes a comment and exception; Cucumber generates these with the hope that you will implement the code needed to fulfill the test step requirement.

> These methods are similar to tests in Junit, however, these tests are driven by the steps outlined in their respective feature files.

Linking Our Test Runner with Steps

Occasionally there are circumstances where the `CukeTestRunner` might not be in the same structure as the `stepdefs`, as such we can provide the package structure Cucumber should scan to find these respective test steps using the `glue` parameter as follows:

```
@CucumberOptions(
    features = "src/test/resources/cuke",
    glue = "com.qa.cuke.custompackagename")
public class CukeTestRunner {}
```

Try running the `CukeTestRunner`, then afterwards, remove these exceptions and add "Hello World" print statement to test how the test class runs!

> You should now be able to successfully run your first Cucumber test!

Exercises

- Refactor your/"Create a" Selenium Google Kittens test to work in a Scenario using the following feature file:

```
Feature: Google Search Kittens

Scenario: Google Search term Kitten
  Given I can access Google
  When I search for Kittens
  And submit my search term
  Then I can navigate to images
  And see pictures of Kittens
```