accenture
Latvia ATC

Test Automation Engineering Fundamentals: Cucumber

Tags and Step definition

Cucumber tags

accenture
Latvia ATC

Test Automation Engineering Fundamentals: Cucumber

Tags

accenture
Latvia ATC

It looks simple when we just have one, two, or maybe five scenarios in a feature file. However, in real life it does not happen. For each feature under test, we may have 10, 20, or may be more number of scenarios in a single feature file. They may represent different purpose (Smoke test/Regression test), different prospectives (Developer/QA/BA), different status (Ready for execution/Work in progress), etc. How to manage execution for such a mass?

For this, Cucumber has already provided a way to organize your scenario execution by using tags in feature file. We can define each scenario with a useful tag. Later, in the runner file, we can decide which specific tag (and so as the scenario(s)) we want Cucumber to execute. Tag starts with "@". After "@" you can have any relevant text to define your tag. Let's understand this with an example.

@tags - scenarios

accenture
Latvia ATC

Suppose, there are two or more scenarios in a feature file. We want to execute only one scenario as part of smoke test. So first thing is to identify that scenario and second is to tag it with “@SmokeTest” text at the beginning of the scenario.

```
Feature: Verify billing
@SmokeTest
  Scenario: Missing product description
  Scenario: Several products
```

@tags - features

accenture
Latvia ATC

Tag can also be defined at a feature level. Once you define a tag at the feature level, it ensures that all the scenarios within that feature file inherits that tag. Depending on the nature of the scenario, we can use more than one tag for the single feature. Whenever Cucumber finds an appropriate call, a specific scenario will be executed.

```
@RegressionTest
Feature: Verify billing
```

Change the runner class

accenture
Latvia ATC

Then all that's left is update runner with tags for run and remove specific feature to just folder with all the features

```
@RunWith(Cucumber.class)
@CucumberOptions(
          features = "src/test/resources/features/",
          tags = {"@SmokeTest"},
          plugin = { "pretty", "html:cucumber-report/html-report",
                "junit:cucumber-report/junit-report.xml",
                "json:cucumber-report/json-report.json" },
          glue = { "stepDefinitions" })
public class CucumberRunner {}
```

Negative tags

accenture
Latvia ATC

Cucumber also provides a way to inverse the choice of tags. Consider that out of 25 defined scenarios, 10 are marked as smoke test. We are required to execute only regression test scenarios.
For this, we can use “~” in JUnit runner class to exclude smoke test scenario.
It will look like the following:

```
tags = {"~@SmokeTest"}
```

Multiple tags

accenture
Latvia ATC

While defining multiple tags, we can also define logical or/and logical and operation.
Defining logical or in runner class – `{“@dev, @wip”}` **– It says that scenarios matching any of this tag needs to be executed.**
Defining logical and in runner class – `{“@dev”, “@wip”}` **– It says that scenarios matching both these tag needs to be executed.**

Tring out tags

accenture
Latvia ATC

1) **Feature “tags“ has 3 tags:**
   - @smokeTest
   - @regressionTest
   - @sanityTest
2) **Change the runner to execute:**
   1) Test(s) with “@smokeTest” [1 test]
   2) Test(s) with “@regressionTest” [3 test]
   3) Test(s) with “@sanityTest” or “@smokeTest” [2 test]
   4) Test(s) with “@sanityTest” and “@smokeTest” [0 test]
   5) Test(s) with “@regressionTest”, but not “@sanityTest” [2 test]
   6) Tests with from all the features [3+ test]

Step definition

accenture
Latvia ATC

Test Automation Engineering Fundamentals: Cucumber

Exact match

accenture
Latvia ATC

```
@Given("I have a cucumber step")
public void i_have_a_cucumber_step() throws Throwable {
    System.out.println("Step definition exact match");
}
```

```
Given I have a cucumber step
Given in the beginning all I have a cucumber step and a salad
```

Use anchors
(^) for the beginning and ($) to show end of expression:

```
@Given("^I have a cucumber step$")
```

Capture integers and strings

accenture
Latvia ATC

```
@Given("^I have (\\d+) (.*) in my basket$")
public void i_have_in_my_basket(int number, String veg) throws Throwable {
    System.out.println(format("I have {0} {1} in my basket", number, veg));
}
```

By using round brackets a part of the expression is marked as a capture group so that Cucumber can map it to a method parameter. In case above this means following patterns are in place:

- **\d+** matching at least one digit, **d** represents a digit, **+** is a quantifier and means one or more times; the expression is escaped with a backslash, because it also is the escape character in Java it is needed to escape it with another backslash, therefor end result is **\\d+**
- **.+** matching at least one character, **.** (dot) represents any character

Use non-capturing groups

accenture
Latvia ATC

It may be useful to have a bit of flexibility and add words in the step which are not matched. This is what non-capturing groups can be used for. There is a ?:operator (question mark and colon) and if it is present at the beginning of the group it will not be mapped to method parameters.

```
@Given("^I have a (?:tasty|nasty|rusty) cucumber step$")
public void i_have_a_X_cucumber_step() throws Throwable {
    System.out.println("Step definition with a non-capturing group");
}
```

Use non-capturing groups examples

accenture
Latvia ATC

```
@Given("^I have a (?:tasty|nasty|rusty) cucumber step$")
```

```
Scenario: Non-capturing group         # cucumber/regex.feature:9
  Given I have a tasty cucumber step # CucumberSteps.i_have_a_X_cucumber_step()
  Given I have a nasty cucumber step # CucumberSteps.i_have_a_X_cucumber_step()
  Given I have a rusty cucumber step # CucumberSteps.i_have_a_X_cucumber_step()
```

Singular and plural

accenture
Latvia ATC

Use ? qualifier to match words in both singular and plural. ? at the end of a word makes the last letter optional. We can also use the logical alternative operator |(pipe) to support correct grammar as well as irregular plurals which will make sentence read better.

```
@Given("^There (?:is|are) (\\d+) (?:cats?|ox|oxen) fed by (\\d+)
(?:persons?|people)$")
public void animals_fed_by_people(int animals, int persons) throws Throwable {
    System.out.println(format("{0} animal(s) fed by {1} person(s)",
                    animals, persons));
}
```

Singular and plural examples

accenture
Latvia ATC

```
@Given("^There (?:is|are) (\\d+) (?:cats?|ox|oxen) fed by (\\d+)
(?:persons?|people)$")
```

```
Given There is 1 cat fed by 1 person
Given There are 2 cats fed by 1 person
Given There are 2 cats fed by 2 persons
Given There are 2 cats fed by 3 people
Given There is 1 ox fed by 4 persons
Given There are 3 oxen fed by 5 people
```

Use Data Tables

accenture
Latvia ATC

Cucumber has a nice feature that will help to use tables in scenarios. The table can easily be converted to a list or a map that can be used in steps.

```
Scenario: Data tables example 1
  Given I have the following order
    | vegetable | amount | cost |
    | cucumber  |   4    |  10  |
    | carrot    |   5    |   6  |
    | potato    |   6    |   4  |

Scenario: Data tables example 2
  Given a list of numbers
    | 17   |
    | 42   |
    | 4711 |
```

One column to List

accenture
Latvia ATC

```
Scenario: The sum of a list of numbers should be calculated
  When I summarize numbers
    | 17   |
    | 42   |
    | 4711 |
  Then should I get 4770
```

```
private int sum;
@Given("^I summarize numbers$")
public void i_summarize_numbers(List<Integer> numbers) throws Throwable {
    for (Integer number : numbers) {
        sum += number;
    }
}
```

2 columns to Map

accenture
Latvia ATC

```
Scenario: Coffee can be ordered when available
  Given the price list for a coffee shop
    | coffee | 1 |
    | donut  | 2 |
  When I order 1 coffee
```

```
private Map<String, Integer> priceList;
@Given("^the price list for a coffee shop$")
public void the_price_list_for_a_coffee_shop(Map<String, Integer> priceList) throws Throwable {
    this.priceList = priceList;
}

@When("^I order (\\d+) (.*)$")
public void i_order_item(int numberOfItems, String item) throws Throwable {
    int price = priceList.get(item);
    totalSum += price * numberOfItems;
}
```

3 columns to Map

accenture
Latvia ATC

```
Given I have the following order
  | vegetable | amount | cost |
  | cucumber  |   4    |  10  |
  | carrot    |   5    |   6  |
  | potato    |   6    |   4  |
```

```
@Given ("^I have the following order$")
public void i_have_the_following_order (DataTable table) throws Throwable {
    for (Map<String, String> map : table.asMaps(String.class, String.class)) {
        String vegetable = map.get("vegetable");
        String amount = map.get("amount");
        String cost = map.get("cost");
        System.out.println(format("Order of {0} {1}s at the cost of {2}",
                        amount, vegetable, cost));
    }
}
```

Map data tables to domain objects

accenture
Latvia ATC

Luckily there are easier ways to access your data than DataTable. For instance you can create a domain object and have Cucumber map your data in a table to a list of these.

```
@Given("^I have another order$")
public void i_have_another_order(List<OrderItem> list) throws Throwable {
    for (OrderItem orderItem : list) {
        String vegetable = orderItem.getVegetable ();
        int amount = orderItem.getAmount();
        int cost = orderItem.getCost ();
        System.out.println(format("Order of {0} {1}s at the cost of {2}",
                        amount, vegetable, cost));
    }
}
```

Domain object – OrderItem

accenture
Latvia ATC

```
public class OrderItem {

    private String vegetable;
    private int amount;
    private int cost;

    public String getVegetable () {
        return vegetable;
    }

    public void setVegetable (String vegetable) {
        this.vegetable = vegetable;
    }

    public int getAmount () {
        return amount;
    }

    public void setAmount (int amount) {
        this.amount = amount;
    }

    public int getCost () {
        return cost;
    }

    public void setCost (int cost) {
        this.cost = cost;
    }
}
```

QUESTIONS

accenture
Latvia ATC

Activity

accenture
Latvia ATC

1) **Create a new feature file**
2) **Write tests for the “Give us your feedback!” form**