\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Cucumber

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1. keywords used inside CucumberOptions

feature, glue, dryrun,tags, monochrome, strict, plugin

There are different type of options present under @CucumberOptions. Lets look at each option one by one to understand it better.

Plugin: plugin Option is used to specify different formatting options for the output reports. Various options that can be used as for-matters are:

Note – Format option is deprecated . Use Plugin in place of that.

A- Pretty: Prints the Gherkin source with additional colors and stack traces for errors.

@RunWith(Cucumber.class)

@CucumberOptions(plugin = {"pretty" ,"html:Folder\_Name" ,

"json:Folder\_Name/cucumber.json" ,

"junit:Folder\_Name/cucumber.xml"})

public class RunYoursTest

{

// This class will be empty

}

DryRun: This option can either set as true or false (default value is false). If it is set as true, it means that Cucumber will only checks that every Step mentioned in the Feature File have corresponding code written in Step Definition file or not. So in case any of the function is missed in the Step Definition for any Step in Feature File, it will give us the message. So If you writing scenarios first and then implementing step definitions then add dryRun = true.

Strict: if strict option is set to false then at execution time if cucumber encounters any undefined/pending steps then cucumber does not fail the execution and undefined steps are skipped and BUILD is SUCCESSFUL.

and if Strict option is set to true then at execution time if cucumber encounters any undefined/pending steps then cucumber does fails the execution and undefined steps are marked as fail and BUILD is FAILURE. This is what the Console output looks like:

Monochrome: This option can either set as true or false (default value is false). If it is set as true, it means that the console output for the Cucumber test are much more readable. And if it is set as false, then the console output is not as readable as it should be. For practice just add the code ‘monochrome = true‘ in TestRunner class.

Features: Features Options helps Cucumber to locate the Feature file in the project folder structure.All we need to do is to specify the folder path and Cucumber will automatically find all the ‘.features‘ extension files in the folder.

Glue: It is almost the same think as Features Option but the only difference is that it helps Cucumber to locate theStep Definition file. Whenever Cucumber encounters a Step, it looks for a Step Definition inside all the files present in the folder mentioned in Glue Option.

It can be defined like-

Snippet Style: Cucumber generates code snippets in Underscore style by default. If you want to change the format of cucumber snippets then you can set snippet type in your cucumber options. There are two types of snippets,

1- SnippetType.CAMELCASE

2- SnippetType.UNDERSCORE

and if Strict option is set to true then at execution time if cucumber encounters any undefined/pending steps then cucumber does fails the execution and undefined steps are marked as fail and BUILD is FAILURE. This is what the Console output looks like:

2. Difference between DryRun and Strict in cucumberOptions?

dryRun: true. checks if all the steps have the stepdefinition

strict: true: will fail execution if there are indefined or pending tags.

3. And/OR operation in Tags

Separate a list of tags by commas for a Logical OR tag expression.

tags={"@first,@second"}= OR

Specifying multiple tag arguments creates a logical AND between each tag expression.

tags={"@first","@second"}= AND

tags={"~@Regression"}= NOT

4. Annotations in Cucumber

Cucumber hooks are blocks of code that runs before or after each scenario. It can be defined anywhere in project or step definition layers using methods @Before, @After. Cucumber hooks Annotations allow us to manage better code workflow and help in reducing code redundancy. Cucumber hooks are used in a situations where prerequisite steps before testing any test scenario is performed.

1. @Before Hook: It will execute before every scenario.

2. @After Hook: It will execute after every scenario.

5. Use of order function inside cucumber annotations

@Before(order = int) : This runs in increment order, means value 0 would run first and 1 would be after 0.

@After(order = int) : This runs in decrements order, means apposite of @Before. Value 1 would run first and 0 would be after 1.

6. Use of background keyword

Background in Cucumber is used to define a step or series of steps that are common to all the tests in the feature file. It allows you to add some context to the scenarios for a feature where it is defined. A Background is much like a scenario containing a number of steps. But it runs before each and every scenario were for a feature in which it is defined.

7. Keywords used in cucumber

Feature, Scenario, Scenario Outline, Given, When, Then, Examples

8. How to perform singleline and multiline comment in cucumber

single line comment: use #

multi line comment: press Ctrl+/

9. How data is passed in cucumber

Data-Driven Testing in Cucumber

Parameterization without Example Keyword

Scenario: Successful Login with Valid Credentials

Given User is on Home Page

When User Navigate to LogIn Page

And User enters "testuser\_1" and "Test@123"

Then Message displayed Login Successfully

Data-Driven Testing in Cucumber using Scenario Outline

Parameterization with Example Keyword

Scenario Outline: Successful Login with Valid Credentials

Given User is on Home Page

When User Navigate to LogIn Page

And User enters "<username>" and "<password>"

Then Message displayed Login Successfully

Examples:

| username | password |

| testuser\_1 | Test@153 |

| testuser\_2 | Test@153 |

Parameterization using Tables

Scenario: Successful Login with Valid Credentials

Given User is on Home Page

When User Navigate to LogIn Page

And User enters Credentials to LogIn

| testuser\_1 | Test@153 |

Then Message displayed Login Successfully

@When("^User enters Credentials to LogIn$")

public void user\_enters\_testuser\_\_and\_Test(DataTable usercredentials) throws Throwable {

//Write the code to handle Data Table

List<List<String>> data = usercredentials.raw();

//This is to get the first data of the set (First Row + First Column)

driver.findElement(By.id("log")).sendKeys(data.get(0).get(0));

//This is to get the first data of the set (First Row + Second Column)

driver.findElement(By.id("pwd")).sendKeys(data.get(0).get(1));

Maps in Data Tables with Header

In the previous chapter of Data Tables in Cucumber, we pass Username & Password without Header, due to which the test was not much readable. What if there will be many columns. The basic funda of BDD test is to make the Test in Business readable format, so that business users can understand it easily. Setting Header in Test data is not a difficult task in Cucumber. take a look at a below Scenario.

Feature File Scenario

Scenario: Successful Login with Valid Credentials

Given User is on Home Page

When User Navigate to LogIn Page

And User enters Credentials to LogIn

| Username | Password |

| testuser\_1 | Test@153 |

Then Message displayed Login Successfully

Data-Driven Testing in Cucumber using External Files

Parameterization using Excel Files

Parameterization using Json

Parameterization using XML

10. Use of ScenarioOutline and example

Scenario Outline is used to perform same scenario testing on different data set. Scenario Outline must have “Examples” keyword and in example we pass test data. In above feature file login scenario performed for four users (mercury, mercury1, mercury2 and mercury3).

11. What is dataTable

dataTable is used to parameterization in cucumber

12. Can user provide multiple examples for one scenario outline

yes

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*