**What is Cucumber Options ?**

In layman language ***@CucumberOptions*** are like property file or settings for your test. Basically *@CucumberOptions* enables us to do all the things that we could have done if we have used cucumber command line. This is very helpful and of utmost importance if we are using IDE such eclipse only to execute our project. You must have noticed that we set few options in the ‘***TestRunner’*** class in the previous chapter.

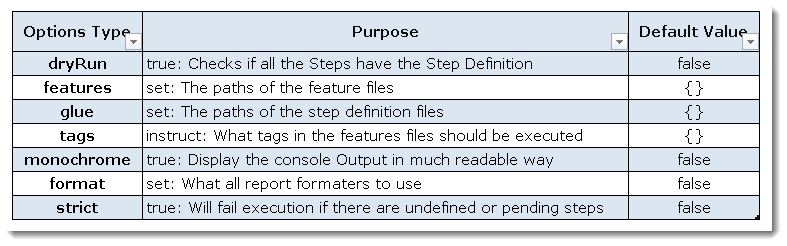
***TestRunner Class***



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | package cucumberTest;    import org.junit.runner.RunWith;  import cucumber.api.CucumberOptions;  import cucumber.api.junit.Cucumber;    @RunWith(Cucumber.class)  @CucumberOptions(  features = "Feature"  ,glue={"stepDefinition"}  )    public class TestRunner {    } |

So in the above example we have just set two different *Cucumber* *Options*. One is for *Feature File* and other is for *Step Definition* file. We will talk about it in detail now but with this we can say that *@CucumberOptions* are used to set some specific properties for the *Cucumber* test.

Following Main Options are available in Cucumber:



***Dry Run***

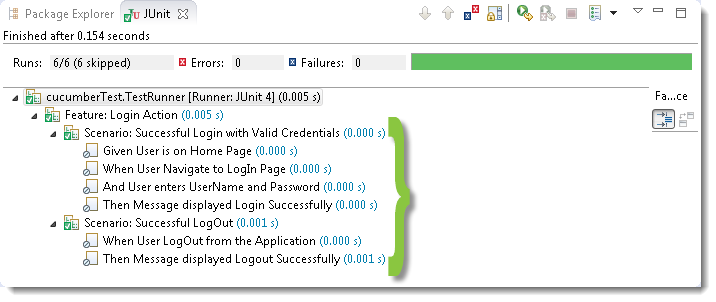
***dryRun*** option can either set as ***true***or ***false***. If it is set as *true*, it means that *Cucumber* will only checks that every *Step* mentioned in the *Feature File* have corresponding code written in *Step Definition* file or not. So in case any of the function is missed in the *Step Definition* for any *Step* in *Feature File*, it will give us the message. For practice just add the code ‘***dryRun = true***‘ in ***TestRunner*** class:

***TestRunner Class***

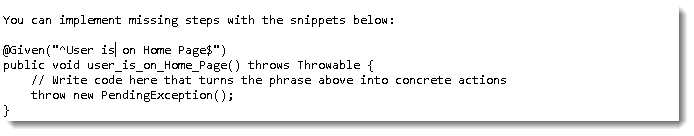


|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | package cucumberTest;    import org.junit.runner.RunWith;  import cucumber.api.CucumberOptions;  import cucumber.api.junit.Cucumber;    @RunWith(Cucumber.class)  @CucumberOptions(  features = "Feature"  ,glue={"stepDefinition"}  ,dryRun = true  )    public class TestRunner {    } |

*Now give it a run by Right Click* on ***TestRunner*** class and Click ***Run As****>****JUnit Test.****Cucumber*will run the script and the result will be shown in the left hand side *project explorer window* in *JUnit* tab.



Take a look at the time duration at the end of the every *Steps*, it is (***0.000s***). It means none of the *Step* is executed but still *Cucumber* has made sure that every Step have the corresponding method available in the *Step Definition* file. Give it a try, remove the ‘***@Given(“^User is on Home Page$”)***‘ statement from the ***Test\_Steps*** class and run the ***TestRunner***class again. You would get the following message:



***Monochrome***

This option can either set as ***true***or ***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:

***TestRunner Class***

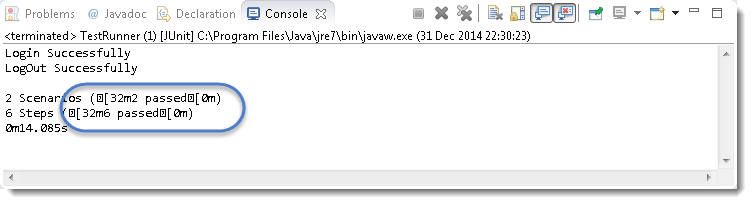


|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | package cucumberTest;    import org.junit.runner.RunWith;  import cucumber.api.CucumberOptions;  import cucumber.api.junit.Cucumber;    @RunWith(Cucumber.class)  @CucumberOptions(  features = "Feature"  ,glue={"stepDefinition"}  ,monochrome = false  )    public class TestRunner {    } |

*Now give it a run by Right Click* on ***TestRunner*** class and Click ***Run As****>****JUnit Test.****Cucumber*will run the script and Console Output will display like this:



This time change the value from *true* to *false* and run the ***TestRunner*** class again. This time the *Console Output* will look like this:



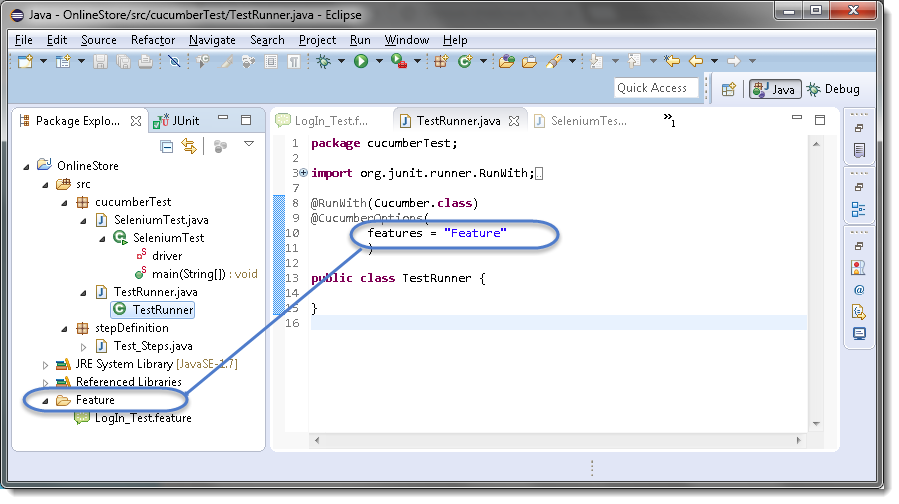
***Features***

***Features Options***helps *Cucumber* to locate the *Feature file* in the project folder structure. You must have notices that we have been specifying the *Feature Option* in the ***TestRunner***class since the first chapter. All we need to do is to specify the folder path and *Cucumber* will automatically find all the ‘***.features***‘ extension files in the folder. It can be specified like:

***features = “Feature“***

*Or if the Feature file is in the deep folder structure*

***features = “src/test/features“***



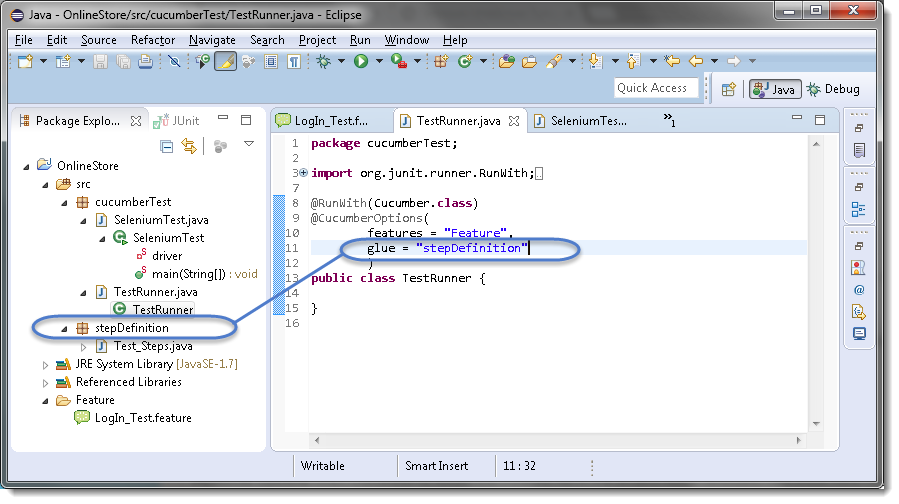
***Glue***

It is almost the same think as *Features Option* but the only difference is that it helps *Cucumber* to locate the ***Step 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 specified like:

***glue = “stepDefinition“***

*Or if the Step Definition file is in the deep folder structure*

***glue = “src/test/stepDeinition“***



***Format***

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

***Pretty:***Prints the *Gherkin* source with additional colours and stack traces for errors. Use below code:

***format = {“pretty“}***

***HTML:***This will generate a HTML report at the location mentioned in the for-matter itself. Use below code:

***format = {“html:Folder\_Name“}***

***JSON:***This report contains all the information from the gherkin source in JSON Format. This report is meant to be post-processed into another visual format by 3rd party tools such as Cucumber Jenkins. Use the below code:

***format = {“json:Folder\_Name/cucumber.json“}***

***JUnit:*** This report generates XML files just like Apache Ant’s JUnit report task. This XML format is understood by most Continuous Integration servers, who will use it to generate visual reports. use the below code:

***format = { “junit:Folder\_Name/cucumber.xml“}***

Gherkin

***Gherkin*** is not necessarily used to write automated tests. Gherkin is primarily used to write ***structured*** tests which can later be used as project documentation. The property of being structured gives us the ability to automate them. This automation is done by ***Cucumber/SpecFlow***. In the [***Gherkin – Business Driven Development***](http://toolsqa.wpengine.com/cucumber/gherkin-business-driven-development-bdd-language/) we saw a simple Gherkin Keyword test and why Gherkin is important to use.

***Note:***Cucumber/SpecFlow understands Gherkin hence we can say that this is a Cucumber/SpecFlow test.

***Feature***: LogIn Action Test  
Description: This feature will test a LogIn and LogOut functionality

***Scenario***: 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

You will quickly notice that there are some colored words. These words are Gherkin keywords and each keyword holds a meaning. Now we will discuss these keywords one by one. Here is the list of keywords that Gherkin supports:

* ***Feature***
* ***Background***
* ***Scenario***
* ***Given***
* ***When***
* ***Then***
* ***And***
* ***But***
* **\***

## Feature: Keyword

Each Gherkin file begins with a **Feature** keyword. Feature defines the logical test functionality you will test in this feature file. For e.g if you are testing a payment gateway your Feature will become Payment Gateway or if you are testing the LogIn functionality then the Feature will become Login. The idea of having a feature file is to put down a summary of what you will be testing. This will serve as the documentation for your tests as well as a good point to start for a new team member. Note that a feature keyword is present at the starting of the feature file.

***Feature***: LogIn Action Test

Or

***Feature***: LogIn Action Test  
Description: This feature will test a LogIn and LogOut functionality

Or

***Feature***: LogIn Action Test  
This feature will test a LogIn and LogOut functionality

Notice that whatever comes after the ***Feature: keyword,*** will be considered as the feature description. Feature description can span across multiple lines like shown above in second example. Everything after Feature: till the next Keyword is encountered is considered as feature description.

***Note:*** Description is not a keyword of Gherkin.

Take a look at the example of [***Cucumber Feature***](http://toolsqa.com/cucumber/cucumber-jvm-feature-file/) file and [***SpecFlow Feature***](http://toolsqa.com/specflow/feature-file/) file

## Background: Keyword

***Background***keyword is used to define steps which are common to all the tests in the feature file. For example to purchase a product, you need to do following steps:

* Navigate to Home Page
* Click on the LogIn link
* Enter UserName and Password
* Click on Submit button

After these steps only you will be able to add a product to your cart/basket and able to perform the payment. Now as we are in a feature file where we will be testing only the Add to Cart functionality, these tests become common for all tests. So instead of writing them again and again for all tests we can move it under the background keyword. This is how it will look like:

***Feature***: Add to Cart  
This feature will test functionality of adding different products to the User basket from different flow

***Background:*** User is Logged In

***Scenario***: Search a product and add the first result/product to the User basket  
***Given***User searched for Lenovo Laptop  
***When*** Add the first laptop that appears in the search result to the basket  
***Then*** User basket should display with 1 item

Take a look at the example of [***Cucumber Background***](http://toolsqa.com/cucumber/background-in-cucumber/)

## Scenario: Keyword

Each Feature will contain some number of tests to test the feature. Each test is called a ***Scenario*** and is described using the Scenario: keyword.

***Scenario***: Search a product and add the first result/product to the User basket

Or

***Scenario***: Successful LogIn with Valid Credentials

A scenario is equivalent to a test in our regular development process. Each scenario/test can be basically broken down into three parts:

* ***Precondition*** to the test, which represent with (***Given***) keyword
* ***Test step*** execution, which represent with (***When***) keyword
* ***Verification*** of the output with expected result, which represent with (***Then***)

## Given Keyword

***Given*** defines a precondition to the test. For e.g. In shopping website, assume that the LogIn page link is only present on the Home Page, so the precondition for clicking the LogIn link is that the user is at the Home Page. If user is not at the Home Page, user would not be able to enter Username & Password. This precondition can be expressed in Gherkin like this:

***Scenario***: Successful LogIn with Valid Credentials

***Given*** User is on Home Page  
***When*** User Navigate to LogIn Page

## When Keyword

***When*** keyword defines the test action that will be executed. By test action we mean the user input action.

***Scenario***: Successful LogIn with Valid Credentials

***Given*** User is on Home Page  
***When*** User Navigate to LogIn Page

Here user is performing some action using When keyword, clicking on the LogIn link. We can see that when defines the action taken by the user. It’s the event that will cause the actual change in state of the application.

## Then Keyword

***Then***keyword defines the Outcome of previous steps. We can understand it best by looking at the test above and adding a Then step there.

***Feature***: LogIn Action Test  
Description: This feature will test a LogIn and LogOut functionality

***Scenario***: 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

Here we can see that ***Then*** is the outcome of the steps above. The reader of this test would easily be able to relate to Then step and would understand that when the above conditions are fulfilled then the Then step will be executed.

## And Keyword

***And*** keyword is used to add conditions to your steps. Let’s look at it by modifying our example a little

***Feature***: LogIn Action Test  
Description: This feature will test a LogIn and LogOut functionality

***Scenario***: 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

Or

***Feature***: LogIn Action Test  
Description: This feature will test a LogIn and LogOut functionality

***Scenario***: Successful Login with Valid Credentials  
***Given*** User is on Home Page  
***And*** LogIn Link displayed  
***When*** User Navigate to LogIn Page  
***And*** User enters UserName and Password  
***Then*** Message displayed Login Successfully  
***And***LogOut Link displayed

Here you would see that And is being used to add more details to the Given step, it’s simply adding more conditions. We have just added three conditions. Use it when you have specified more than one condition. And is used to add more conditions to Given, When and Then statements.

## But Keyword

***But*** keyword is used to add negative type comments. It is not a hard & fast rule to use but only for negative conditions. It makes sense to use But when you will try to add a condition which is opposite to the premise your test is trying to set. Take a look at the example below:

***Feature***: LogIn Action Test  
Description: This feature will test a LogIn and LogOut functionality

***Scenario***: Unsuccessful Login with InValid Credentials  
***Given*** User is on Home Page  
***When*** User Navigate to LogIn Page  
***And*** User enters UserName and Password  
***But***The user credentials are wrong  
***Then***Message displayed Wrong UserName & Password

Here you can see how adding ***But*** has helped define a negative test, in this test we will try to test failure conditions. Where a wrong credentials are a failure condition.

## \* Keyword

This keyword is very special. This keyword defies the whole purpose of having Given, When, Then and all the other keywords. Basically Cucumber doesn’t care about what Keyword you use to define test steps, all it cares about what code it needs to execute for each step. That code is called a ***step definition*** and we will discuss about it in the next section. At this time just remember that all the keywords can be replaced by the ***\* keyword*** and your test will just work fine. Let’s see with example, we had this test earlier:

***Feature***: LogIn Action Test  
Description: This feature will test a LogIn and LogOut functionality

***Scenario***: 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

***Using \* Keyword***

***Feature***: LogIn Action Test  
Description: This feature will test a LogIn and LogOut functionality

***Scenario***: Successful Login with Valid Credentials  
***\****User is on Home Page  
***\****User Navigate to LogIn Page  
***\****User enters UserName and Password  
***\****Message displayed Login Successfully

Here we conclude the tutorial on different keywords of Cucumber. I hope you like it. Let’s now jump deep into how to execute these steps with the help of Step Definition file.