Cucumber with Java – Build Automation Framework with Minimal code

What is Framework?

In any Real time project whenever Automation scripts are developed, One should come up with an Execution system called framework to run and maintain Automated tests

What is Cucumber?

Cucumber is the BDD Framework for running automated tests.   
Cucumber does not Automate your testcases!

When my tests are already automated and can run, what cucumber does?

Data driven, Parameterization, Execution controls, Hooks, Reports, Automation utilities and many more….

When you say automated tests, what type of Automation testcases does cucumber Support?

Any Test (Web, Mobile, API, Unit Testing) which is written in Java/Ruby supported by Cucumber

How cucumber is unique and Best from other Test Frameworks (Keyword, Datadriven,Hybrid) in the Market?

Because Testcases/Requirements are defined with on BDD methodology (Gherkin syntax)

No coding is required to implement Framework functionalities

Cucumber Terminologies:

What is Gherkin?   
 It is a [Business Readable, Domain Specific Language](https://martinfowler.com/bliki/BusinessReadableDSL.html) that lets you describe software's behavior.

Example: Pop up messaged is displayed when buttons are clicked and errors are gone

Keywords Used in Cucumber: Scenario, Feature, Feature file, Scenario outline, Step Definition

Scenarios:

In Cucumber Testcases are represented as Scenarios.

Scenarios contain Steps which are equivalent to test Steps and use the following keywords (Gherkin syntax) to denote them: Given, When, Then, But, and And (case sensitive).

* **Given**: Preconditions are mentioned in the Given keyword
* **When**: The purpose of the When Steps is to describe the user action.
* **Then**: The purpose of Then Steps is to observe the expected output. The observations should be related to the business value/benefit of your Feature description.

When we specify a business requirement, sometimes there are multiple pre-conditions, user actions, and expected outcomes

we are going to add one more Scenario and will use the And and But keywords:

* **And**: This is used for statements that are an addition to the previous Steps and represent positive statements.
* **But**: This is used for statements that are an addition to previous Steps and represent negative statements.

Feature and Feature File:

Feature represents Business requirement.

Feature File acts as a Test Suite which consists of all Scenarios.

In Cucumber, Feature files contain Scenarios. We can simply create feature file with. feature extension

Scenarios belonging to specific area of Application will be grouped into one Feature file

The text that immediately follows the Feature keyword, and is in the same line, is the Title of the Feature file

Feature file should contain either Scenario or Scenario Outline. The naming conventions for Feature files should be lowercase with. feature extension

**Feature**: Credit card payment

In order to test Credit Card Payment functionality

As a CC user

I want to complete the payment through online

**Scenario**: Make Minimum Due payment

**Given** user is on Pay credit card page

**Then** user fills all details and select Minimum amount option

**And** User clicks on Pay button

**Then** Credit Card confirmation page is displayed

**Scenario**: Pay Statement Balance

**Given** user is on Pay credit card page

**Then** user fills all details and select Statement Balance option

**And** User clicks on Pay button

**Then** Credit Card confirmation page is displayed

**Scenario**: Enter another Amount as 0

**Given** user is on Pay credit card page

**Then** user fills all details and select other Amount and enter 0

**And** User clicks on Pay button

**Then** Credit Card confirmation page is not displayed

**But** error message is displayed

**Feature File:**

#Author: your.email@your.domain.com

#Keywords Summary :

#Feature: List of scenarios.

#Scenario: Business rule through list of steps with arguments.

#Given: Some precondition step

#When: Some key actions

#Then: To observe outcomes or validation

#And,But: To enumerate more Given,When,Then steps

#Scenario Outline: List of steps for data-driven as an Examples and <placeholder>

#Examples: Container for s table

#Background: List of steps run before each of the scenarios

#""" (Doc Strings)

#| (Data Tables)

#@ (Tags/Labels):To group Scenarios

#<> (placeholder)

#""

## (Comments)

#Sample Feature Definition Template

#if we need to parameteriazing the values we need to use scenario outline

#we can use multiple tgss for 1 scenrios also

#Background: List of steps run before each of the scenarios

Examples: #it is keyword to pass the parameterized vaues

| name | value | status |

| name1 | 5 | success |

| name2 | 7 | Fail |

**Example:**

#Sample Feature Definition Template

@tag

Feature: Title of your feature I want to use this template for my feature file

#Background: List of steps run before each of the scenarios

#Background:

#Given validate the browser

#When browser is trigged

#Then check if browser is started

@tag1

Scenario: Basic scenario

Given I want to write a step with precondition

And some other precondition

When I complete action with "string"

And some other action

And yet another action

Then I validate the outcomes

And check more outcomes

#if we need to parameteriazing the values we need to use scenario outline

#we can use multiple tgss for 1 scenrios also

# @tag1

@tag2

Scenario Outline: scenario with values

Given I want to write a step with name <name>

#write the key in <> to pass the value

When I check for the <value> in step

Then I verify the <status> in step

Examples: #it is keyword to pass the parameterized vaues

| name | value | status |

| name1 | 5 | success |

| name2 | 7 | Fail |

@tag3

Scenario: data driven scenario

Given user is on signup page

When user signup with following details

|First name | Last name | email@yopmail.com | Address | 50324 |

**Step Defination:**

1. **Data Table:**

//code for datadriven table

**public** **void** user\_signup\_with\_following\_details(DataTable data) **throws** Throwable { //all the data stored in data object

//data is stored in rows and columns (index)

List<List<String>> obj = data.asLists(); //to get/retrieve the list of data in data object

//get(0).get(0) means first row, first column, index starts with 0

String firstName = obj.get(0).get(0); //firstName will be stored in firstname object

String lastName = obj.get(0).get(1); //lastName will be stored in lastName object

String email = obj.get(0).get(2); //email will be stored in email object

String address = obj.get(0).get(3); //address will be stored in address object

String zipcode = obj.get(0).get(4); //zipcode will be stored in zipcode object

**Example:**

**package** stepDefinations;

**import** java.util.List;

**import** io.cucumber.datatable.DataTable;

**import** io.cucumber.java.en.And;

**import** io.cucumber.java.en.Given;

**import** io.cucumber.java.en.Then;

**import** io.cucumber.java.en.When;

**public** **class** stepDefination

{

@Given("^validate the browser$")

**public** **void** validate\_the\_browser() **throws** Throwable {

System.***out***.println("validate the browser");

}

@When("^browser is trigged$")

**public** **void** browser\_is\_trigged() **throws** Throwable {

System.***out***.println("browser is trigged");

}

@Then("^check if browser is started$")

**public** **void** check\_if\_browser\_is\_started() **throws** Throwable {

System.***out***.println("check if browser is started");

}

@Given("^I want to write a step with precondition$")

**public** **void** i\_want\_to\_write\_a\_step\_with\_precondition() **throws** Throwable {

System.***out***.println("Given");

}

@When("^I complete action with \"([^\"]\*)\"$")

**public** **void** i\_complete\_action\_with\_something(String strArg1) **throws** Throwable {

System.***out***.println("When with "+strArg1 );

}

@Then("^I validate the outcomes$")

**public** **void** i\_validate\_the\_outcomes() **throws** Throwable {

System.***out***.println("Then");

}

@And("^some other precondition$")

**public** **void** some\_other\_precondition() **throws** Throwable {

System.***out***.println("And");

}

@And("^some other action$")

**public** **void** some\_other\_action() **throws** Throwable {

System.***out***.println("And");

}

@And("^yet another action$")

**public** **void** yet\_another\_action() **throws** Throwable {

System.***out***.println("And");

}

@And("^check more outcomes$")

**public** **void** check\_more\_outcomes() **throws** Throwable {

System.***out***.println("And");

}

@Given("^I want to write a step with name (.+)$") // code for values

**public** **void** i\_want\_to\_write\_a\_step\_with\_name(String name) **throws** Throwable {

System.***out***.println(name);

}

@When("^I check for the (.+) in step$")

**public** **void** i\_check\_for\_the\_in\_step(String value) **throws** Throwable {

System.***out***.println(value);

}

@Then("^I verify the (.+) in step$")

**public** **void** i\_verify\_the\_in\_step(String status) **throws** Throwable {

System.***out***.println(status);

}

@Given("^user is on signup page$")

**public** **void** user\_is\_on\_signup\_page() **throws** Throwable {

}

@When("^user signup with following details$") //code for datadriven table

**public** **void** user\_signup\_with\_following\_details(DataTable data) **throws** Throwable { //all the data stored in data object

//data is stored in rows and columns (index)

List<List<String>> obj = data.asLists(); //to get/retrieve the list of data in data object

//get(0).get(0) means first row, first column, index starts with 0

String firstName = obj.get(0).get(0); //firstName will be stored in firstname object

String lastName = obj.get(0).get(1); //lastName will be stored in lastName object

String email = obj.get(0).get(2); //email will be stored in email object

String address = obj.get(0).get(3); //address will be stored in address object

String zipcode = obj.get(0).get(4); //zipcode will be stored in zipcode object

System.***out***.println(firstName);

System.***out***.println(lastName);

System.***out***.println(email);

System.***out***.println(address);

System.***out***.println(zipcode);

}

}

**Test Runner:**

1. **Junit:**

@RunWith(Cucumber.**class**) //openwith used to execute in Junit

@CucumberOptions(

features = "src/test/java/features",

glue="stepDefinations", stepNotifications = **true**, tags ="@tag3 or @tag1 or tag2", dryRun=**true**, monochrome=**true**,

plugin= {"pretty", "html:target/cucumber.html", "json:target/cucumber.json", "junit:target/cucumber.xml"}) //to generate reports

//tags is used to execute only particular tags what we mentioned

//"not @tag" is used to run except that tag other tags should run

// if a scenario has two tags we need to use "@tag1 and @tag2"

1. **TestNG:**

@CucumberOptions(features = "src/test/java/features",

glue = "stepDefinations")

**public** **class** testngTestRunner **extends** AbstractTestNGCucumberTests{ //extends used to execute in testing

**Hooks:**

// hooks used to execute before particular tags

// to run hooks need to remove background from features

**Example:**

package stepDefinations;

import io.cucumber.java.After;

import io.cucumber.java.Before;

public class hooks{

// hooks used to execute before particular tags

// to run hooks need to remove background from features

@Before("@tag3")

public void beforeScenario() {

System.out.println("before tag3 scenario, hooks");

}

@After("@tag3")

public void afterScenario() {

System.out.println("after tag3 scenario, hooks");

}

@Before("@tag1")

public void before\_tag1\_Scenario() {

System.out.println("before tag1 scenario, hooks");

}

@After("@tag1")

public void after\_tag1\_Scenario() {

System.out.println("after tag1 scenario, hooks");

}

}

Sample Selenium Cucumber Project

<https://github.com/selenium-cucumber/selenium-cucumber-java-maven-example>

Sample Appium Cucumber Project

<https://github.com/mubbashir/appium-java-cucumber-gradle>

**MAVEN:**

Maven is a software project management and build management tool for java frameworks

* Central repository to get dependencies
* Maintaining common structure across the organization -- Jars
* Flexibility in integration in CI tools (Jenkins)
* Plugins for test framework execution

**Maven Installation:**

Download maven apache zip file from this link[**https://maven.apache.org/download.cgi**](https://maven.apache.org/download.cgi)

Extract zip and set path in environment table as **MAVEN\_HOME** & **%MAVEN\_HOME%\bin**

**Maven terminologies :**

**Artifact:** it is file, usually a jar, that helps deployed to maven repository

**GroupID:** groupid will identify your project uniquely across all projects

**archetype:generate** Generates a new project from an archetype

**Creating Maven project:**

mvn -B archetype:generate -DgroupId=com.mycompany.app -DartifactId=my-app -DarchetypeArtifactId=maven-archetype-quickstart -DarchetypeVersion=1.4

To import maven project in eclipse we need to integrate

**integrate maven with eclipse:** in command prompt **cd <Project Name>** then write **mvn eclipse:eclipse**

by integrating maven with eclipse .project and .classpath will create in project

Import maven project to eclipse

**Write test cases:**

Create java files and write test cases for selenium, appium, API etc..

Java file should be end with Test keyword

And create test runner file

**Import Dependencies:**

To run selenium, appium, API, TestNG (testRunner) we need jars, to get JARS add dependency in POM.xml file

Get dependency from maven repository (selenium java, appium (java client) API (rest assured), TestNG )

**Add maven sure fire plugin to eclipse:**

To execute all our test cases in maven project we need **Maven sure fire plugin**

Get plugin code from below link

<https://maven.apache.org/surefire/maven-surefire-plugin/usage.html>

copy code from build/plugin and add code below project/under plugin tag resp

add code in pom.xml file

**Maven commands : command prompt**

To run in command prompt change directory to project location

Mvn Clean: to avoid build errors

Mvn complie: to check complie error/ syntax errors

Mvn test: trigger our test execution, if we given test directly test before clean and complile by default it will run clean and compile

**Running cucumber project with Maven commands:**

Similarly we can able to run Our cucumber automation project through command prompt,   
in command prompt navigate to project location and run with maven commands, make sure our cucumber project is maven project

Maven will run tests which file name start with **TestRunner** in the project

We have a command to run single test runner file through maven “mvn –Dtest=TestRunner(file name) test ”

We have a command to run single **tag** scenarios **mvn test–Dcucumber.options= “--tags @smoketest”**

**Jenkins:**

Installation of Jenkins.war file:

Go to cmd prompt and set path to folder where has Jenkins.war file

java -jar jenkins.war –httpPort 8080

**OR**

install Jenkins.exe file

After install download necessary plugins and create username and password and set path of Java, Maven, Ant, Git etc…

Click on manage Jenkins then global tool configuration after set jdk, maven paths and save

**Creation of Maven project in Jenkins**

[**http://localhost:8080**](http://localhost:8080)

Firstly our maven project should be placed in .jenkins folder, that folder path we get from manage Jenkins🡪 system information🡪 under environment variables JENKINS\_HOME path, Paste the maven project in .jenkins file



Click on new item in dashboard

Enter project name and Click on free style project

In general section click on advance the click on use custom workspace **${JENKINS\_HOME}/MavenJava** for maven project

In Build environment section click on add build step then select invoke top-level maven targets, then set maven version and give command (**test**) in goals then save

After configuration of settings to run the test click on Build now

To get the results in Jenkins of automation project, in TestRunner file we have to generate test results file and those files are set in target folder

The path of test results file should be placed in Jenkins

Project 🡪 Configure🡪add post build actions 🡪 select publish junit test result report and paste the path and save

Now run the build and check results in Latest test result in Jenkins project