## Cucumber Introduction:

* Cucumber is a framework which supports BDD-Behavior Driven Development
* In BDD Automation programs are created based on the behavior of the application
* Cucumber was initially implemented with Ruby, later it was extended to Java, C#

## Cucumber Feature File:

In Cucumber the automation programs are created based on a file called Feature File.

In this feature file the task to be automated is written in plain English statements which are connected to the selenium code which performs those activities. The advantage of this process is the flow of the automation programs can be understood by Non-Technical people like Stake-Holders or Business Analyst.

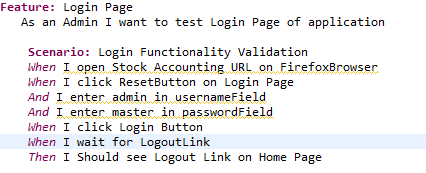
Note: Cucumber uses its own language called as Gherkin

## Cucumber Keywords:

Feature: This represents the module or functionality that is under tests

Scenario: This represents the test case that is been automated in a particular feature

Note: one Feature have one or more Scenarios.



Given: This represents the pre-condition of the test case

When: This represents the exact action that is performed in the test case

And: This represents any additional actions that should be performed on the test case

Then: This represents the outcome of the test case

## Cucumber Test Runner Class:

With a cucumber-based framework, you cannot run a feature file on its own. You will need to create a java class, which in turn will run this cucumber feature file. We call this class as **cucumber test runner class**.

Cucumber test runner class is one of the many mechanisms using which you can run Cucumber feature file. The test runner class also acts as an inter-link between feature files and **step definition classes**. It is in test runner class, that you provide the path for both feature file and **step defs class**.

With a test runner class, you have the option to run either a single feature file, or multiple feature files as well. For now, we will focus on running a single feature file

Example TestRunner:

**import** org.junit.runner.RunWith;

**import** cucumber.api.CucumberOptions;

**import** cucumber.api.junit.Cucumber;

**import** cucumber.api.testng.AbstractTestNGCucumberTests;

@RunWith(Cucumber.**class**)

@CucumberOptions(features="FeatureFiles",

glue= "stepDefinitions",

tags= {"@CreationTest"},

plugin = {"com.cucumber.listener.ExtentCucumberFormatter:Reports/report.html","pretty", "html:target/cucumber-reports"},

//monochrome = **true**

//,dryRun=true

)

**public** **class** Runner **extends** AbstractTestNGCucumberTests {

}

Create FeatureFiles folder at the project level and create one feature file in it

Create testRunner package and create a Runner.java under \src\test\java

Now run the test runner and observe the missing steps generated by cucumber

Create stepDefinitions package and create a StepDefinitions.java class under \src\test\java

Add the missing steps generated by cucumber to StepDefinitions.java class

Create a new source folder and name it as \src\test\resources and put driver files in it

Create Reports folder at the project level to store extent reports

Update the pom.xml file with Apache poi dependencies

Create TestData excel workBook in \src\test\resources

Create utility package in \src\test\java and create an excel Handler Class to get data from a cell and sheet

Create DataHelper class to transfer data to cucumber line in Create utility package in \src\test\java

**public** **class** DataHelper {

**public** **static** String getData(String rowSheetIndex,String colHdr) **throws** Throwable {

ExcelHandler tdtf=**new** ExcelHandler();

//String hypen="11-Login";

**int** hypenIndex=rowSheetIndex.indexOf("-");

String number=rowSheetIndex.substring(0, hypenIndex);

**int** rowNumber=Integer.*parseInt*(number);

String sheet=rowSheetIndex.substring(hypenIndex+1, rowSheetIndex.length());

//System.out.println(number+" "+sheet);

**int** rqColNumber=-1;

**for**(**int** i=0;i<tdtf.colCount(sheet, 0);i++) {

**if**(tdtf.getData(sheet,0,i).equalsIgnoreCase(colHdr)) {

rqColNumber=i;

}

}

**return** tdtf.getData(sheet, rowNumber, rqColNumber);

}

}

Create a scenario like below

Scenario Outline: Login Functionality Validation

When I open Stock Accounting URL on "Browser" from "<Row-SheetIndex>"

When I click ResetButton

And I enter "UserName" in usernameField from "<Row-SheetIndex>"

And I enter "Password" in passwordField from "<Row-SheetIndex>"

When I click Login Button

When I wait for LogoutLink

Then I Should see Logout Link

When I close the browser

Examples:

|Row-SheetIndex|

|2-Login|

|3-Login|

|4-Login|