Cucumber à

-          Cucumber can be used with Java, Ruby, Scala, Groovy.

-          JBehave is another BDD Framework.

-          Cucumber consist of three files,

o   Feature file (say login.feature) à It contains all the Gherkin keywards (Given, When, Then, And, As, But)

o   Step Definition à It contains Selenium + Java + Annotations

o   TestRunner à It is used to run our feature file and generate output/reports

**Installing Cucumber à**

-          Cucumber should be created in Maven project.

-          Following are dependencies of cucumber maven project (that should be provided in pom.xml file)

o   cucumber –java

o   cucumber –jvm

o   cucumber –junit

o   cucumber –jvm-deps

o   cucumber –reporting

o   gherkin

o   JUnit

o   selenium

* Following plugin are need to be installed in pom.xml
  + maven-compiler-plugin
  + maven-resources-plugin
  + maven-surefire-plugin // It is used to run all code in cucumber
  + maven-cucumber-reporting

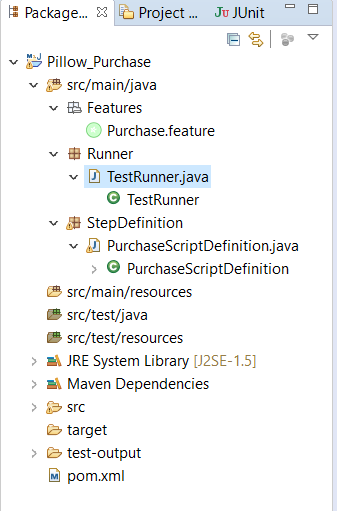
-          In order to know the Gherkin keywords by eclipse, we need to use ‘Natural’ plugin from eclipse market place.

-          Feature class à Create new package in src/main and create new file in it (say login.feature).

-          Definition class à Create new package in src/main and create new class (say LoginStepDefinition.java)

-          Runner class à Create new package in src/main and create new class (say TestRunner.java)

-          To run cucumber project, we always go to ‘TestRunner.class’ file -> right click and select RunAs -> Junit Test.



Note – Complete project is available at C:\Users\VJ\Mar14

* Whenever we want to use data driven testing (parameterization) with ‘Keyword’ approach, we need to use ‘scenario outline’.
* One feature file can contain multiple scenarios.

Cucumber focuses on ‘what to test’ and not ‘how to test’

Cucumber Options à

-          dryRun

-          Features

-          Glue

-          Tags

-          Monochrome

-          Format

-          Strict

-          dryRun à dryRun = **true**

-          It should be provided in @CucumberOptions in TestRunner.java

-          It won’t run the scenario but it will confirm whether for every feature (keyword) in .feature file, corresponding method is present in definition file or not.

-          If any method is missing then in console it will show blank space for corresponding step in .feature file

-          To run the scenario after confirmation, it should be modified to dryRun = **false**

-          Features  à It is used to provide feature file / folder / package path

-          Glue à It is used to provide ‘step definition’ file / folder / package path

-          Monochrome à monochrome = **true**

-          Display the console output in proper readable format

-          Format à It is used to generate different types of reporting

~~format~~= {"pretty", "html:test-output", "json:json\_output/cucumber.json", "junit:junit\_xml/cucumber.xml"},

-          Strict  à strict = **true**

-          For every step in feature file, if corresponding method is not present in definition class then it will give ‘pendingException’ after execution.

-          Tag  à tags= {"@SmokeTest"}

-          Tag is used to run some specific scenarios instead of running the complete feature file (like Smoke, Regression, End to end tests).

-          To implement this, we need to assign tags to scenarios in .feature file before writing scenario.

*@SmokeTest*

**Scenario:** Free CRM Login test Scenario

-  Tags can be worked with OR, AND operators.

                - OR à tags= {"@SmokeTest, @RegressionTest"}

              - AND à tags= {"@SmokeTest", "@RegressionTest"}

      - ignore à tags= {"~@SmokeTest"}

TestRunner.java  à

**package** Runner;

**import** org.junit.runner.RunWith;

**import** cucumber.api.CucumberOptions;

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

@RunWith(Cucumber.**class**)

@CucumberOptions(

              features = "C:\\Users\\vtt1\\eclipse\_Mar10\\MavenPro\\src\\main\\java\\Features\\login.feature",

              glue= {"StepDefinition"},

~~format~~= {"pretty", "html:test-output", "json:json\_output/cucumber.json", "junit:junit\_xml/cucumber.xml"},

              monochrome = **true**,

tags= {"@SmokeTest"},

              strict = **true**,

              dryRun = **false**

              )

**public** **class** TestRunner {

}

**Hooks in Cucumber   à**

-          Hooks are defined to specify preconditions (like launch browser, enter URL, delete cookies) and post conditions (close browser).

-          @Before and @After used for that purpose.

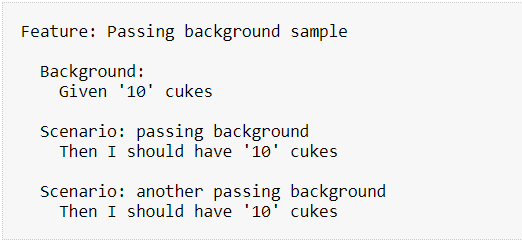
-          Methods with @Before (to initialize) and @After (to teardown) are defined in step definition class. No need to do anything in .feature file.

-          @Before method is executed before starting of any scenario from .feature file.

-          @After method is executed after completion of any scenario from .feature file.

**Q.) What is the use of ‘Background’ keyword in cucumber ?**

**-** Background keyword allows to specify steps that should be run before each scenario in the feature.

****

**Q.) When cucumber is used in real time ?**

- It is generally used in real time to write acceptance tests for an application. It is generally used by non technical people such as Business Analyst, Function Tester etc.

**Q.) What are the different keywords of Gherkin language ?**

Feature

Scenario

Scenario Outline

Given

When

Then

And

\*

Examples

Background

Data Driven Testing in Cucumber  à

1)      By passing parameters from single line from feature file.

Feature File –

*Then* User enters "Vijay" and "pass@123"

Step Definition –

@Then("^User enters \"([^\"]\*)\" and \"([^\"]\*)\"$")

**public** **void** user\_enter\_username\_and\_password (String username, String password) {

                     driver.findElement(By.*xpath*("//input[@placeholder=\"MM / YY\"]")).sendKeys(username);

                     driver.findElement(By.*xpath*("//input[@placeholder=\"123\"]")).sendKeys(password);

       }

2)      If user want to pass multiple set of data for iterative testing  à Here user need to use ‘Scenario Outine’ with ‘Examples‘ keyword.

Feature file -

**Feature:** Midtrans Website Automation Testing

**Scenario Outline:** Pillow Purchase Using Credit Card

*When* Selects payment option as Credit Card

*Then* Enters the credit card details "*<Card Number>*", "*<Expiry Date>*", "*<CVV Number>*" and bank's OTP "*<Bank’s OTP>*"

**Examples:**

| Card Number | Expiry Date | CVV Number | Bank’s OTP |

| 4811 1111 1111 1114 | 03/20 | 123 | 112233 |

| 4911 1111 1111 1113 | 03/20 | 123 | 112233 |

Definition File –

       @Then("^Enters the credit card details \"([^\"]\*)\", \"([^\"]\*)\", \"([^\"]\*)\" and bank's OTP \"([^\"]\*)\"$")

**public** **void** enters\_the\_credit\_card\_details\_and\_bank\_s\_OTP(String card\_number, String expiry, String CVV, String OTP) {

              driver.findElement(By.*name*("cardnumber")).sendKeys(card\_number);

              driver.findElement(By.*xpath*("//input[@placeholder=\"MM / YY\"]")).sendKeys(expiry);

              driver.findElement(By.*xpath*("//input[@placeholder=\"123\"]")).sendKeys(CVV);

driver.findElement(By.id("PaRes")).sendKeys(OTP);

       }

3)  **Using Test Data Table (List Object)** à In this approach, we write test data immediately after the step in feature file (as the data is applicable for only that step and not for complete scenario like we use for Scenario outline). Here we use ‘Scenario’ and not ‘Scenario Outline’.

Feature File –

*Then* User enters username and password

| vijay | pass@123 |

Definition File –

              @Then("^User enters username and password$")

**public** **void** user\_enter\_username\_and\_password (DataTable credentials) {     // here credentials is variable which refers to data provided in feature file

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

                     driver.findElement(By.xpath("//input[@placeholder=\"MM / YY\"]")).sendKeys(data.get(0).get(0));

                    driver.findElement(By.xpath("//input[@placeholder=\"123\"]")).sendKeys(data.get(0).get(1));

              }

4)  **Using Map**  à Issue while working with above list object is that the data we were passing were not having column name. So for third person, it becomes difficult to understand. And if I want to iterate one specific step multiple times (not scenario) like creating multiple users in one big scenario then that was not possible. To overcome this, we use Map concept. Here we use ‘Scenario’ and not ‘Scenario Outline’.

Feature File –

*Then* User enters deal details

| title | amount | probability | commission |

| test deal1 | 1000 | 50 | 10 |

| test deal1 | 2000 | 60 | 20 |

| test deal3 | 3000 | 70 | 30 |

Definition File –

                     @Then("^User enters deal details$")

**public** **void** user\_enter\_deal\_details (DataTable dealData) {

**for**(Map<String, String> data : dealData.asMap(String.**class**, String.**class**))

                           driver.findElement(By.xpath("//input[@placeholder=\"MM / YY\"]")).sendKeys(data.get("title"));

                         driver.findElement(By.xpath("//input[@placeholder=\"123\"]")).sendKeys(data.get("amount"));

                         driver.findElement(By.xpath("//a[@class=\"button-main-content\"]")).sendKeys(data.get("probability"));

                         driver.findElement(By.xpath("//input[@placeholder=\"werf\"]")).sendKeys(data.get("commission"));

              }

pom.xml  à

<project xmlns=*"*[*http://maven.apache.org/POM/4.0.0*](http://maven.apache.org/POM/4.0.0)*"* xmlns:xsi=*"*[*http://www.w3.org/2001/XMLSchema-instance*](http://www.w3.org/2001/XMLSchema-instance)*"* xsi:schemaLocation=*"*[*http://maven.apache.org/POM/4.0.0*](http://maven.apache.org/POM/4.0.0)[*https://maven.apache.org/xsd/maven-4.0.0.xsd*](https://maven.apache.org/xsd/maven-4.0.0.xsd)*"*>

  <modelVersion>4.0.0</modelVersion>

  <groupId>MavenPro</groupId>

  <artifactId>MavenPro</artifactId>

  <version>0.0.1-SNAPSHOT</version>

  <dependencies>

  <dependency>

    <groupId>info.cukes</groupId>

    <artifactId>cucumber-java</artifactId>

    <version>1.2.5</version>

  </dependency>

   <dependency>

    <groupId>info.cukes</groupId>

    <artifactId>cucumber-jvm-deps</artifactId>

    <version>1.0.5</version>

    <scope>provided</scope>

   </dependency>

  <dependency>

    <groupId>info.cukes</groupId>

    <artifactId>gherkin</artifactId>

    <version>2.12.2</version>

    <scope>provided</scope>

  </dependency>

  <dependency>

    <groupId>org.seleniumhq.selenium</groupId>

    <artifactId>selenium-java</artifactId>

    <version>3.141.59</version>

  </dependency>

  <dependency>

    <groupId>info.cukes</groupId>

    <artifactId>cucumber-junit</artifactId>

    <version>1.2.5</version>

  </dependency>

  <dependency>

    <groupId>junit</groupId>

    <artifactId>junit</artifactId>

    <version>4.13</version>

</dependency>

</dependencies>

</project>

login.feature  à

**Feature:** Free CRM Login Feature

**Scenario:** Free CRM Login test Scenario

*Given* user is already on Login Page

*When* title of login page is Free CRM

*Then* user enters username and password

*And* user clicks on login button

LoginStepDefinition.java  à

**package** StepDefinition;

**import** [org.openqa.selenium.By](http://org.openqa.selenium.by/);

**import** org.openqa.selenium.WebDriver;

**import** org.openqa.selenium.chrome.ChromeDriver;

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

**import** cucumber.api.java.en.Then;

**import** cucumber.api.java.en.When;

**import** junit.framework.~~Assert~~;

**public** **class** LoginStepDefinition {

       WebDriver driver;

       @Given("^user is already on Login Page$")

**public** **void** user\_already\_on\_login\_page() {

              System.*setProperty*("webdriver.chrome.driver", "D:/chromedriver.exe");

              driver = **new** ChromeDriver();

              driver.get("<https://www.facebook.com/>");

              }

       @When("^title of login page is Free CRM$")

**public** **void** title\_of\_login\_page() {

              String title = driver.getTitle();

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

~~Assert~~.~~assertEquals~~("Facebook – log in or sign up", title);

       }

       @Then("^user enters username and password$")

**public** **void** user\_enters\_username\_and\_password() {

              driver.findElement(By.*id*("email")).sendKeys("9420289001");;

              driver.findElement(By.*id*("pass")).sendKeys("dfsswe");

       }

       @Then("^user clicks on login button$")

**public** **void** user\_clicks\_on\_login\_button() {

              driver.findElement(By.*id*("u\_0\_b")).click();

       }

}

TestRunner.java  à

**package** Runner;

**import** org.junit.runner.RunWith;

**import** cucumber.api.CucumberOptions;

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

@RunWith(Cucumber.**class**)

@CucumberOptions(

              features = "C:\\Users\\vtt1\\eclipse\_Mar10\\MavenPro\\src\\main\\java\\Features\\login.feature",

              glue= {"StepDefinition"},

~~format~~= {"pretty", "html:test-output"}

              )

**public** **class** TestRunner {

}