Gherkin is the language to define the BDD framework.

On the basis of behaviour, we write the test cases, there are 3 components in Cucumber

1 feature file

feature file is a login.feature all the gerkins features will be used here. We use all the key word here like given, when, then as, but, and,
With the help of these key words we will define SCENARIO, SCENARIO OUTLINE FOR a specific feature Feature: is the keyword

(login.feature) file

Feature: FREE CRM LOGIN FEATURE

Scenario: FREE CRM LOGIN TEST_SCENARIO
Given User is already on login page (pre requisite)
When title of login page is free crm
Then user enters username and passwor
Then user clicks on logi button
And user is on home page

then save the login.feature file

Always save this in a **Features package**

We need to download get the **Gherkin keyword** automatically that is **"Natural Plugin"**

Our product managers/BA /manual testers will write the **acceptance criteria** in **feature file forma**t // nothing but they are defining the behaviour of the test cases and we are converting into the **step definitions**

Disadvantages of BDD: So many steps we have write in this framework its lengthy but entire agile team will contribute everybody will be on the same page

2 STEP DEFINITION

Corresponding to the feature file we will write all codes into Step definition like JAVA and selenium code and different annotations.

Create a separate **package for the step definition|**
Create a class like **LoginStepDefinitio**n (don't select the main method in java)

@Given(**"^**copy the exact line given in feature file**$"**) **^$ will understand the cucumber language**

// create a method
public void User_is_already_on_login page(){

}

3. Test Runner

We will be writing the Test Runner in JUNIT to run in your **feature** and to generate the output/report

Create a runner package and create a class

We have to add the following steps (3 steps)

@RunWith(Cucumber.class) // Run with import from JUNIT and Cucumber.class import from cucumber.apu.CucumberOptions;

@CucumberOptions(// import from cucumber.api

// in testng we use testng.xml file to control our test cases like wise we have TestRunner.java in cucumber control the executions

features = "Copy you path of features" // where exactly your feature file is available no need to write login.feature file direct package name
,**glue**= {"package name of **stepDefinition** or path "} // standard thing
,**format** = {"pretty", "html:test-output"}
// scenario will get skipped if the step definition is not written, in the console the step definition missing just copy the code snippet

}

public class TestRunner{

}

What are different cucumber options

All these **Cucumber Options** are defined in runner class expect **Tags**

Dryrun
to check whether the mapping is done fine or not **between Feature file and Step Definition.**
It gives **missing stepdefinition**

you will come to know which methods are missing in for stepdefinition the we can write the code in it with respect to steps available in Login.feature file

dryRun= true
dryRun = false

Glue
The path of StepDefinition

Features
The path of feature file/package

Tags (Feature file)

MonoChrome (v imp) always use it
Displays the output in a **readable format**

Monochrome= true, // display the console output in a proper readable format

Format
Format is used to generate different types of reporting
Html
JSON
XML format is format = {"pretty", "junit:junit_xml/cucumber.xml"}

Strict
strict=true,
it will check if any step is not defined in definition file

A proper step definition is written are not.
it will fail the execution if there are any undefined/pending steps are there.
Mapping between step definition and feature file, it will follow strictly follow the rules between them

DATA DRIVEN TESTING USING CUCUMBER:

1. Simple Data Driven- without Examples keyword
2. With examples+scenario outline
3. Using tables

Without examples keyword

Scenario: free crm test

Write in the feature file:
Then user enters "naveenk" and "ABCD"

Go to STEP DEFINITiON go to that respective method

@Then ("^ user enters **\"(.*)\"** and **\"(.*)\"$")** // this is the **\"(.*)\" regular expression it picks from feature file (no hardcoded)**

public void user_enters_usernameand_password(String username, String password){

driver.findelement(By.name("username").sendkeys("username");
driver.findelement(By.name("password").sendkeys("password");

With examples keyword
Scenario Outline: free crm test

Given User is already on login page (pre requisite)
When title of login page is free crm
Then user enters "<username>" and "<password>" // this line is parameterized and used regular expressions in step def
Then user clicks on login button
And user is on home page

Examples:

username	password
naveen	123456
tom	122122

Difference between scenario and scenario outline

If you are **using without examples keyword** then Scenario

If you are using **with examples keyword** then Scenario outline (multiple data can be pass through **pipe symbol)**

How to achieve DD approach in cucumber

To achieve data driven We have to use examples keyword with scenario outline

}

Cucumber Tags

I want to execute **smoke or sanity/ regression/ end2end scenarios then we Tags**

Only important test cases to execute, segregations will be done.

Create a **tagging.feature** file in that put all the scenarios

in that scenario mention as @SmokeTest @RegressionTest
now go to Runner class and put the feature file of tagging.feature and put dryRun=true then get the StepDefinition code snippets
Then complete the StepDefinition and come to Test Runner class and write in @CucumberOptions { as tags={"@SmokeTest, @RegressionTest"}}
OR=,
ANDed condition tags={"@SmokeTest", "@RegressionTest"}

Igonre(**~**) condition tags={" **~@**SmokeTest, @RegressionTest"}

HOOKS IN CUCUMBER

This is like @Before and @After this is used for each SCENARIO
like launching browser we can use this one time instead of repeating

hook.feature file

Given user is on deals page
When user fills the deals
The deal is created

Hooks_step_def

```
@Before
public void 5etup(){
System.out.println("launch FF");
System.out.println("Enter URL");

}

@After
public void teardown(){

System.out.println("Close browser");


}



Given ("^user is on deals page$")
public void user is on deals page{
SYSO
}



When ("^user fills the deals$")
public void user is on deals page{
SYSO
}


Then ("^deal is created$")
public void user is on deals page{
SYSO
}
```

TAGGEDHOOKS

//Global Hooks @After @Before it will be executed to all scenarion (This is @beforeClass in TestNG)

// @Before("@First") // local (this is @First you must mention in your feature file too)

This is like Before method in TestNG that means you can customize the scenarios
You can control scenarios' which has to be executed

We can also define ordering for Hooks this gives the preference

@Before(order=0)
@After(order=0)

@Before(order=1)
@After(order=1)