Cucumber:

A cucumber is a tool based on Behavior Driven Development (BDD) framework which is used to write acceptance tests for the web application. It allows automation of functional validation in easily readable and understandable format (like plain English) to Business Analysts, Developers, Testers, etc.

Cucumber feature files can serve as a good document for all. There are many other tools like JBehave which also support BDD framework. Initially, Cucumber was implemented in Ruby and then extended to Java framework. Both the tools support native JUnit.

Behavior Driven Development is an extension of Test Driven Development and it is used to test the system rather than testing the particular piece of code. We will discuss more the BDD and style of writing BDD tests.

Cucumber can be used along with Selenium, Watir, and Capybara etc. Cucumber supports many other languages like Perl, PHP, Python, Net etc. In this tutorial, we will concentrate on Cucumber with Java as a language

What are the benefits?

1. It is helpful to involve business stakeholders who can't easily read code.

2. Cucumber focuses on end-user experience .

3. Style of writing tests allow for easier reuse of code in the tests

4. Quick and easy set up and execution

5. Efficient tool for testing

Cucumber Basics: In order to understand cucumber, we need to know all the features of cucumber and its usage

1) Feature Files:

Feature files are the essential part of cucumber which is used to write test automation steps or acceptance tests. This can be used as the live document. The steps are the application specification. All the feature files end with .feature extension.

2) Feature:

This gives information about the high-level business functionality (Refer to the previous example) and the purpose of Application under test. Everybody should be able to understand the intent of feature file by reading the first Feature step. This part is basically kept brief.

3) Scenario:

Basically, a scenario represents a particular functionality which is under test. By seeing the scenario user should be able to understand the intent behind the scenario and what the test is all about. Each scenario should follow given, when and then format. This language is called as “gherkin”.

Given: As mentioned above, given specifies the pre-conditions. It is basically a known state.

When: This is used when some action is to be performed. As in above example, we have seen when the user tries to log in using username and password, it becomes an action.

Then: The expected outcome or result should be placed here. For Instance: verify the login is successful, successful page navigation.

Background: Whenever any step is required to perform in each scenario then those steps need to be placed in Background. For Instance: If a user needs to clear database before each scenario then those steps can be put in a background.

And: And is used to combine two or more same type of action.

4) Scenario Outline:

Scenario outlines are used when the same test has to be performed with different data set. Let’s take the same example. We have to test login functionality with multiple different sets of username and password.

Example Program for Cucumber:

1. We have to create one feature file with the extension of .feature(LoginBB.feature)

```
Feature: Login Functionality

Scenario: Login to Big Basket application
   Given Launch the browser
   And navigate to "https://www.bigbasket.com/"
   When click on login button
   And enter the mobile number as "9150660240"
   And click on continue
   Then enter OTP
   And click on Verify and continue
   Then close the browser
```

2)We have to create Base class in which we declare variables and Objects

```
package com.BaseClass;

import org.openqa.selenium.WebDriver;

public class BaseTest {
public static WebDriver driver;
public LoginToBigBasket login;
}
```

Here we can declare and initialize variables and also can be create objects of different classes and This class should inherited in the Steps definitions class

3) We have to Create Page factory class in that we have to write element locators and actions of elements as shown below

```
public class LoginToBigBasket extends BaseTest {
    WebDriver driver;

    public LoginToBigBasket(WebDriver driver) {
        this.driver = driver;
        PageFactory.initElements(driver, this);
    }

    @FindBy(xpath = "//button[text()='Login/ Sign Up']")
    public WebElement loginSignUpButton;
    @FindBy(xpath = "//input[@id='multiform']")
    public WebElement enterMobileNumber;
    @FindBy(xpath = "//button[text()='Continue']")
    public WebElement continueButton;
    @FindBy(xpath = "//button[text()='Verify & Continue']")
    public WebElement verifyAndContinueButton;

    public void clickOnLoginSignUpButton() {

        try {
            Thread.sleep(3000);
            loginSignUpButton.click();

        } catch (Exception e) {
            e.printStackTrace();
        }
        System.out.println("Login/SignIn Buton is successfully clicked");
    }
```

```java
public void enterUserLoginData(String userLoginData) {
    Actions action = new Actions(driver);
    action.sendKeys(Keys.TAB.TAB).build().perform();
    try {
        Thread.sleep(3000);
        enterMobileNumber.sendKeys(userLoginData);
    } catch (Exception e) {
        e.printStackTrace();
    }
    System.out.println("User data is entered");
}

public void clickOnContinueButton() {

    try {
        Thread.sleep(3000);
        continueButton.click();
    } catch (Exception e) {
        e.printStackTrace();
    }
    System.out.println("Continue button is clicked successfully");
}
```

```java
public void waitToEnterOTP() {
    try {
        Thread.sleep(20000);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    System.out.println("OTP is entered successfully");
}

public void clickOnVerifyAndCountButton() {

    try {
        Thread.sleep(3000);
        verifyAndContinueButton.click();

    } catch (Exception e) {
        e.printStackTrace();
    }
    System.out.println("successfully navigated to Home page");
}
```

4)Create a class which is used to implement all the methods generated(Snippets) when you run the feature file

```java
package com.StepDefinitions;

import org.openqa.selenium.chrome.ChromeDriver;
import com.BaseClass.BaseTest;
import com.PageFactory.LoginToBigBasket;
import io.cucumber.java.en.Given;
import io.cucumber.java.en.Then;
import io.cucumber.java.en.When;
import io.github.bonigarcia.wdm.WebDriverManager;

public class Steps extends BaseTest {
    @Given("Launch the browser")
    public void launch_the_browser() {
        WebDriverManager.chromedriver().setup();
        driver = new ChromeDriver();
    }

    @Given("navigate to {string}")
    public void navigate_to(String appUrl) {
        driver.get(appUrl);
        driver.manage().window().maximize();
    }

    @When("click on login button")
    public void click_on_login_button() {
        login = new LoginToBigBasket(driver);
        login.clickOnLoginSignUpButton();
    }

    @When("enter the mobile number as {string}")
    public void enter_the_mobile_number_as(String userMobileNumber) {
        login.enterUserLoginData(userMobileNumber);
    }

    @When("click on continue")
    public void click_on_continue() {
        login.clickOnContinueButton();
    }

    @Then("enter OTP")
    public void enter_otp() {
        login.waitToEnterOTP();
    }

    @Then("click on Verify and continue")
    public void click on verify and continue() {
        login.clickOnVerifyAndCountButton();
    }

    @Then("close the browser")
    public void close_the_browser() {
        driver.quit();
    }
}
```

5)To execute the script we have to create a Test Runner class

```
package com.TestRunner;

import org.junit.runner.RunWith;
import io.cucumber.junit.Cucumber;
import io.cucumber.junit.CucumberOptions;


@RunWith(Cucumber.class)
@CucumberOptions(features="./src/test/java/featureFiles/loginToBB.feature",glue="com.StepDefinitions",
plugin= {"pretty", "html:target/cucumber-reports"})

public class TestRunner {


}
```

We can run or execute our script in two ways

- With Feature File and also
- With Test Runner class

Attributes of Cucumber Options Annotation :

features : Give path of your feature file

glue : Give Package name which contains steps class where you have implemented all snippets

plugin : It is used to generate cucumber reports