Cucumber BDD

tutorialspoint

SIMPLY EASY LEARNING

www.tutorialspoint.com

https://www.facebook.com/tutorialspointindia

https://twitter.com/tutorialspoint

About the Tutorial

Cucumber is a testing tool that supports Behavior Driven Development (BDD) framework. It defines application behavior using simple English text, defined by a language called Gherkin.

Cucumber allows automation functional validation that is easily read and understood. Cucumber was initially implemented in Ruby and then extended to Java framework. Both the tools support native JUnit.

This tutorial is fairly comprehensive and covers all the necessary aspects on Cucumber using examples for easy understanding.

Audience

This tutorial is designed for software professionals such as analysts, developers, and testers who are keen on learning the fundamentals of Cucumber and want to put it into practice.

Prerequisites

Before proceeding with this tutorial, you need to have a basic knowledge on testing as well as some hands-on experience of some testing tools. You should have a commanding knowledge on Java, and some familiarity with JUnit and Ruby.

Copyright & Disclaimer

Table of Contents

1. Cucumber – Overview

In order to get better advantage of the software testing, organizations are nowadays taking a step forward. They implement important acceptance test scenarios while development is in-progress. This approach is commonly known as **Behavior Driven Development** (BDD).

Behavior Driven Development gives us an opportunity to create test scripts from both the developer's and the customer's prospective as well. So in the beginning, developers, project managers, QAs, user acceptance testers and the product owner (stockholder), all get together and brainstorm about which test scenarios should be passed in order to call this software/application successful. This way they come up with a set of test scenarios. All these test scripts are in simple English language, so it serves the purpose of documentation also.

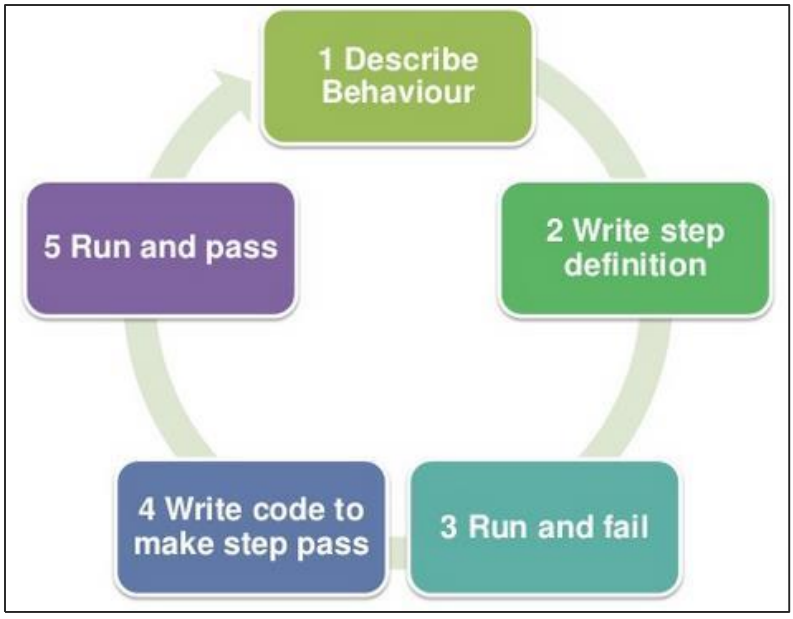
Example

If we are developing a user authentication feature, then the following can be few key test scenarios, which needs to get passed in order to call it a success.

- The user should be able to login with correct username and correct password.
- The user should not be able to login with incorrect username and correct password.
- The user should not be able to login with correct username and incorrect password.

How it Works

By the time the code is ready, test scripts are ready too. The code has to pass the test scripts defined in BDD. If it does not happen, code refactoring will be needed. Code gets freezed only after successful execution of defined test scripts.



It is a very simple notion, but what we need in order to get this concept implemented. The answer is, Behavior Driven Development (BDD) Framework. Cucumber is one such open source tool, which supports behavior driven development. To be more precise, Cucumber can be defined as a testing framework, driven by plain English text. It serves as documentation, automated tests, and a development aid – all in one.

So what does Cucumber do? It can be described in the following steps:

Cucumber reads the code written in plain English text (Language Gherkin – to be introduced later in this tutorial) in the feature file (to be introduced later).

It finds the exact match of each step in the step definition (a code file - details provided later in the tutorial).

The piece of code to be executed can be different software frameworks like **Selenium**, **Ruby on Rails**, etc. Not every BDD framework tool supports every tool.

This has become the reason for Cucumber's popularity over other frameworks, like **JBehave**, **JDave**, **Easyb**, etc.

Cucumber supports over a dozen different software platforms like:

- Ruby on Rails
- Selenium
- PicoContainer
- Spring Framework
- Watir

Advantages of Cucumber Over Other Tools

- Cucumber supports different languages like Java.net and Ruby.
- It acts as a bridge between the business and technical language. We can accomplish this by creating a test case in plain English text.
- It allows the test script to be written without knowledge of any code, it allows the involvement of non-programmers as well.
- It serves the purpose of end-to-end test framework unlike other tools.
- Due to simple test script architecture, Cucumber provides code reusability.

2. Cucumber – Environment

In this chapter, we will see the environment setup for Cucumber with Selenium WebDriver and Java, on Windows Machine.

Prerequisites for Environment Setup

Following are the prerequisites required to set up with:

Java

Why we need: Java is a robust programming language. Cucumber supports Java platform for the execution.

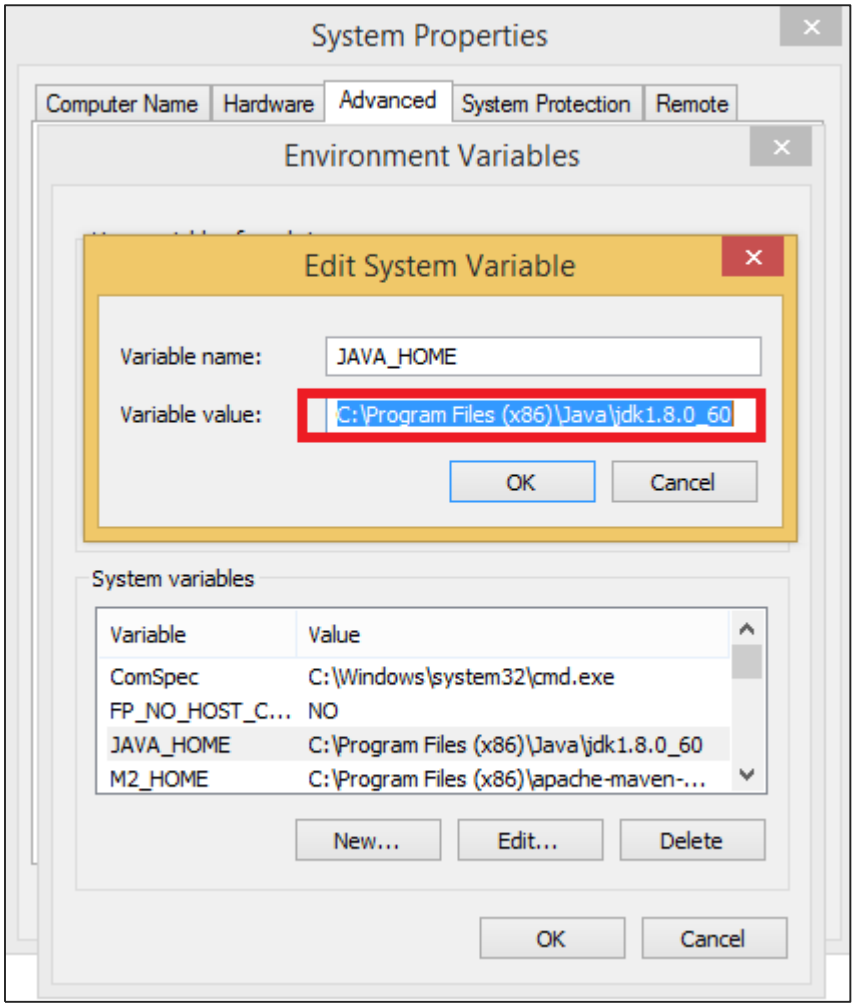
How to install:

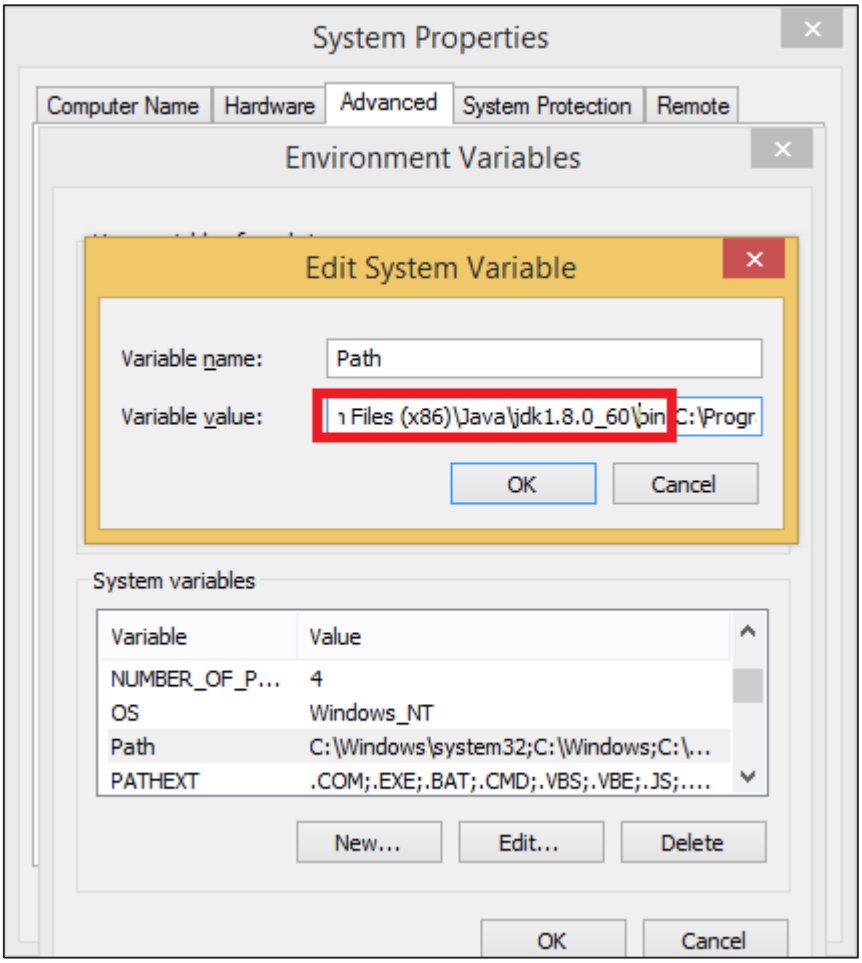
Step (1): Download jdk and jre from the following link
http://www.oracle.com/technetwork/java/javase/downloads/index.html

Step (2): Accept license agreement.

Step (3): Install JDK and JRE.

Step (4): Set the environment variable as shown in the following screenshots.


System Properties
Computer Name
Hardware
Advanced
System Protection
Remote
Environment Variables
Edit System Variable
Variable name:
JAVA_HOME
Variable value:
C:\Program Files (x86)\Java\jdk1.8.0_60
OK
Cancel
System variables
Variable
Value
ComSpec
C:\Windows\system32\cmd.exe
FP_NO_HOST_C...
NO
JAVA_HOME
C:\Program Files (x86)\Java\jdk1.8.0_60
M2_HOME
C:\Program Files (x86)\apache-maven-...
New...
Edit...
Delete
OK
Cancel



Eclipse

- **Why we need**: Eclipse is an Integrated Development Environment (IDE). It contains a base workspace and an extensible plug-in system for customizing the environment.

  How to install:

  Step (1): Make sure JAVA is installed on your machine.

  Step (2): Download Eclipse from http://www.eclipse.org/downloads.

  Step (3): Unzip and Eclipse is installed.

Maven

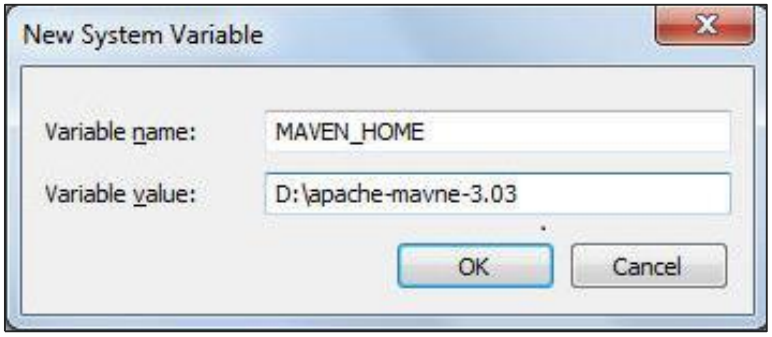
- **Why we need**: Maven is a build automation tool used primarily for Java projects. It provides a common platform to perform activities like generating source code, compiling code, packaging code to a jar, etc. Later if any of the software versions gets changed, Maven provides an easy way to modify the test project accordingly.

- **How to install:**

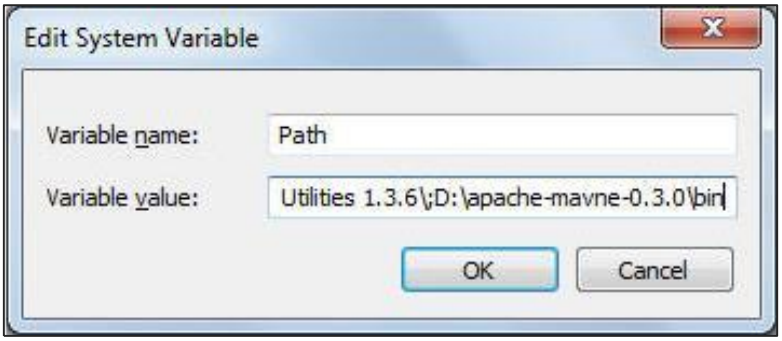
  Step (1): Download Maven from the following link:
  https://maven.apache.org/download.cgi

  Step (2): Unzip the file and remember the location.

  Step (3): Create environment variable MAVEN_HOME as shown in the following screenshot.



  Step (4): Edit Path variable and include Maven as shown in the following screenshot.



  Step (5): Download MAVEN plugin from Eclipse.

  Step (6): Open Eclipse.

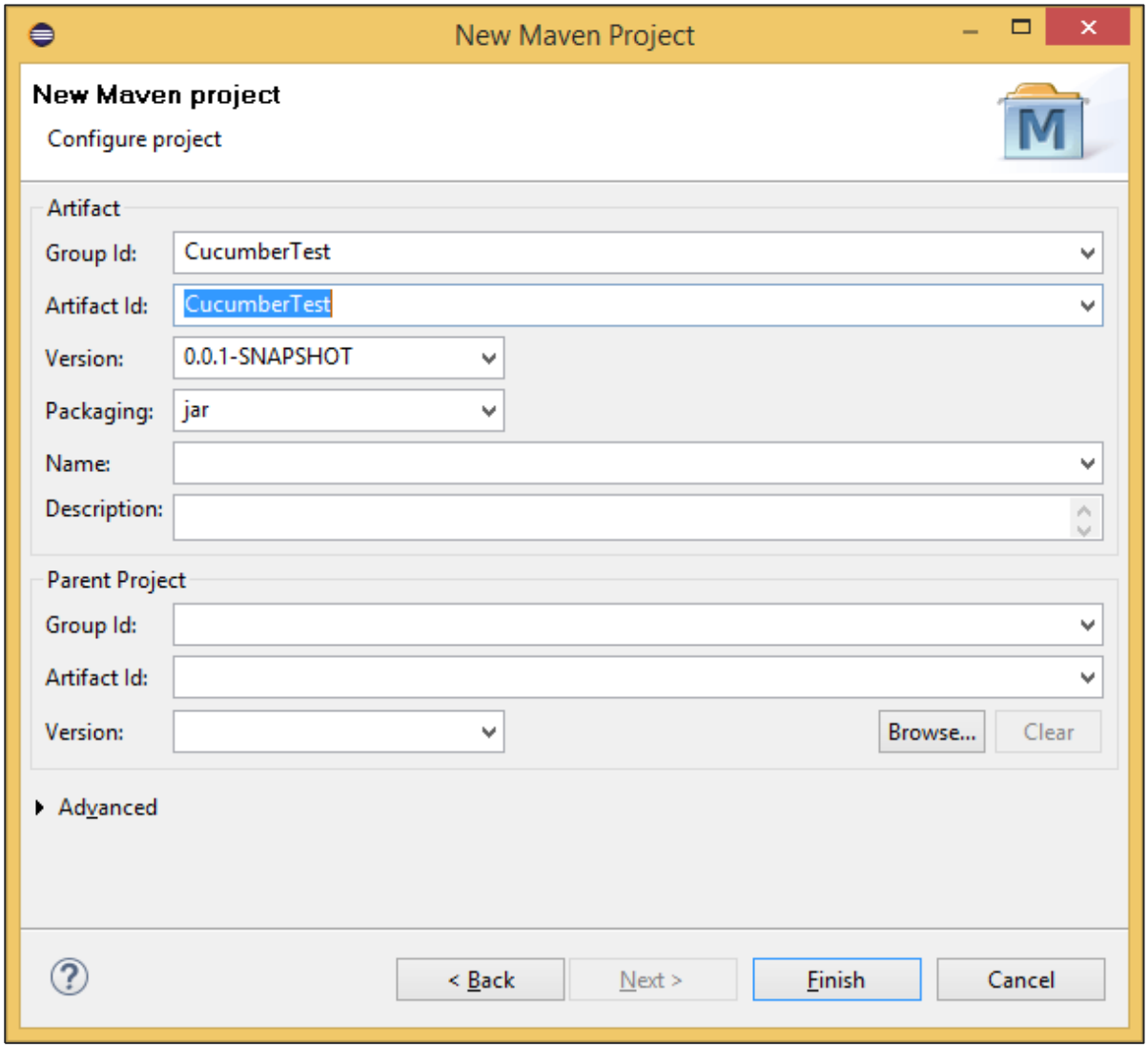
  Step (7): Go to Help -> Eclipse Marketplace -> Search Maven -> Maven Integration for Eclipse -> INSTALL.

Configure Cucumber with Maven

Step (1): Create a Maven project.

- Go to File -> New -> Others -> Maven -> Maven Project -> Next.
- Provide group Id (group Id will identify your project uniquely across all projects).

- Provide artifact Id (artifact Id is the name of the jar without version. You can choose any name, which is in lowercase). Click on Finish.



Step (2): Open pom.xml:

- Go to package explorer on the left hand side of Eclipse.
- Expand the project **CucumberTest**.
- Locate **pom.xml** file.
- Right-click and select the option, open with "Text Editor".

Step (3): Add dependency for selenium: This will indicate Maven which Selenium jar files are to be downloaded from the central repository to the local repository.

- Open **pom.xml** is in the edit mode, create dependencies tag (<dependencies></dependencies>), inside the project tag.

- Inside the dependencies tag, create dependency tag (<dependency></dependency>).

- Provide the following information within the dependency tag.

```
<dependency>
<groupId>org.seleniumhq.selenium</groupId>
<artifactId>selenium-java</artifactId>
<version>2.47.1</version>
</dependency>
```

Step (4): Add dependency for Cucumber-Java: This will indicate Maven, which Cucumber files are to be downloaded from the central repository to the local repository.

- Create one more dependency tag.
- Provide the following information within the dependency tag

```
<dependency>
<groupId>info.cukes</groupId>
<artifactId>cucumber-java</artifactId>
<version>1.0.2</version>
<scope>test</scope>
</dependency>
```

Step (5): Add dependency for Cucumber-JUnit: This will indicate Maven, which Cucumber JUnit files are to be downloaded from the central repository to the local repository.

- Create one more dependency tag.
- Provide the following information within the dependency tag

```
<dependency>
<groupId>info.cukes</groupId>
<artifactId>cucumber-junit</artifactId>
<version>1.0.2</version>
<scope>test</scope>
</dependency>
```

Step (6): Add dependency for JUnit: This will indicate Maven, which JUnit files are to be downloaded from the central repository to the local repository.

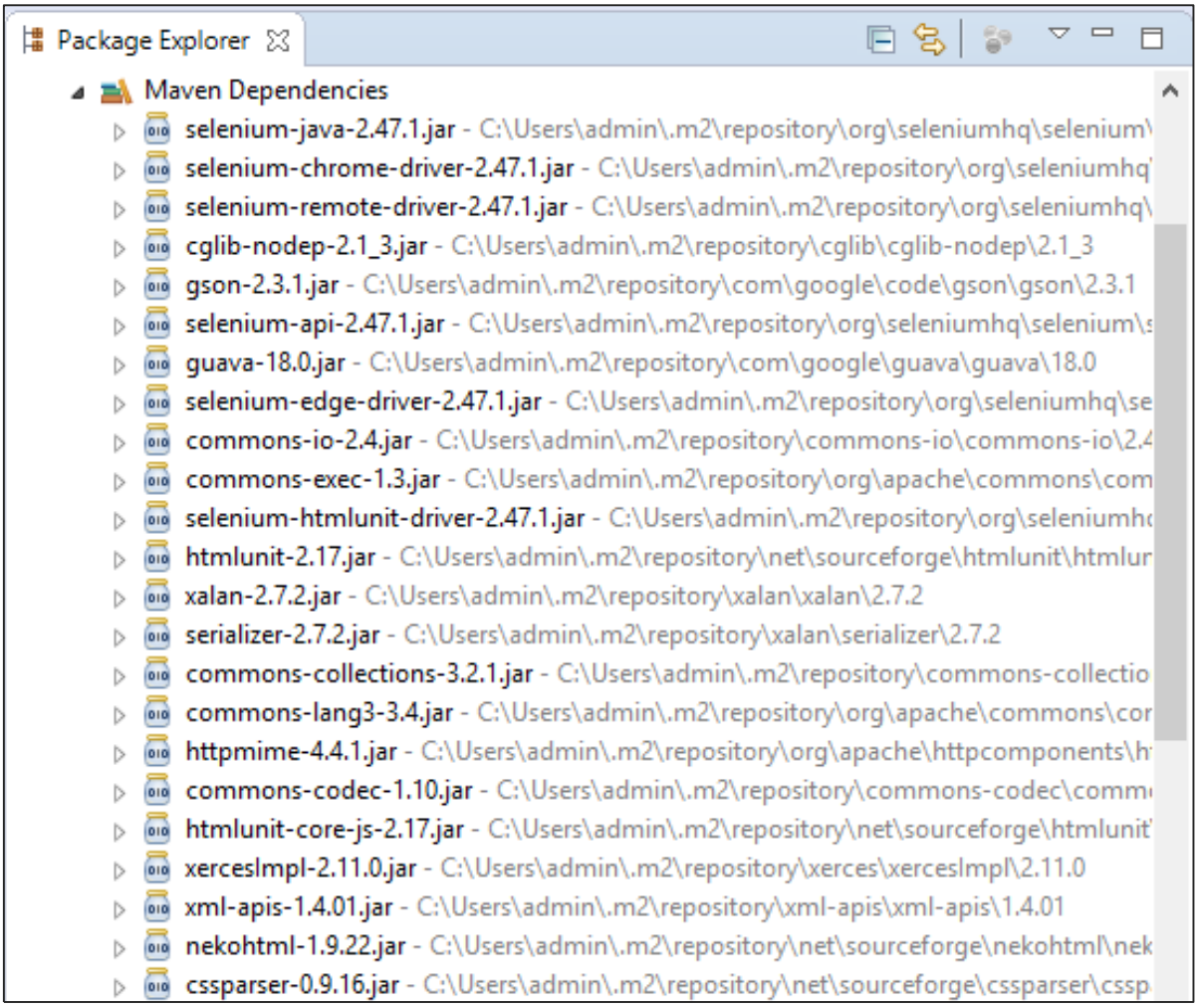
- Create one more dependency tag.
- Provide the following information within the dependency tag.

```
<dependency>
<groupId>junit</groupId>
<artifactId>junit</artifactId>
<version>4.10</version>
<scope>test</scope>
</dependency>
```

Step (7): Verify binaries.

- Once **pom.xml** is edited successfully, save it.
- Go to Project -> Clean: It will take a few minutes.

You will be able to see a Maven repository like shown in the following screenshot.



- Create a feature file (to be covered later).
- Create a step definition file (to be covered later).
- Create a JUnit runner to run the test (to be covered later).

3. Cucumber – Gherkins

So far, we have got an understanding of Cucumber and what it does. It executes the test scripts, which have been defined in the feature file (to be covered in subsequent chapters). The language, in which this executable feature files is written, is known as **Gherkin**. Gherkin is a plain English text language, which helps the tool - Cucumber to interpret and execute the test scripts.

One may think that, it has been discussed many times that Cucumber supports simple English text then why we need a separate language - Gherkins. The answer lies in the concept of the Behavior Driven Development.

As discussed earlier, we had seen that BDD incorporates different prospectives while creating test scripts. It can be development prospective, business prospective, etc. That said, we will need people from different community like developers, project managers, product owners, and testers while developing test scripts. As these people do not belong to the same category, there is a risk of not using the common language for test script conceptualizing. This was the evolution point for Gherkins.

Gherkin provides the common set of keywords in English text, which can be used by people amongst the different community and yet get the same output in the form of test scripts.

Example

Feature: Login functionality for a social networking site. Given I am a social networking site user. When I enter username as username1. And I enter password as password1. Then I should be redirected to the home page of the site.

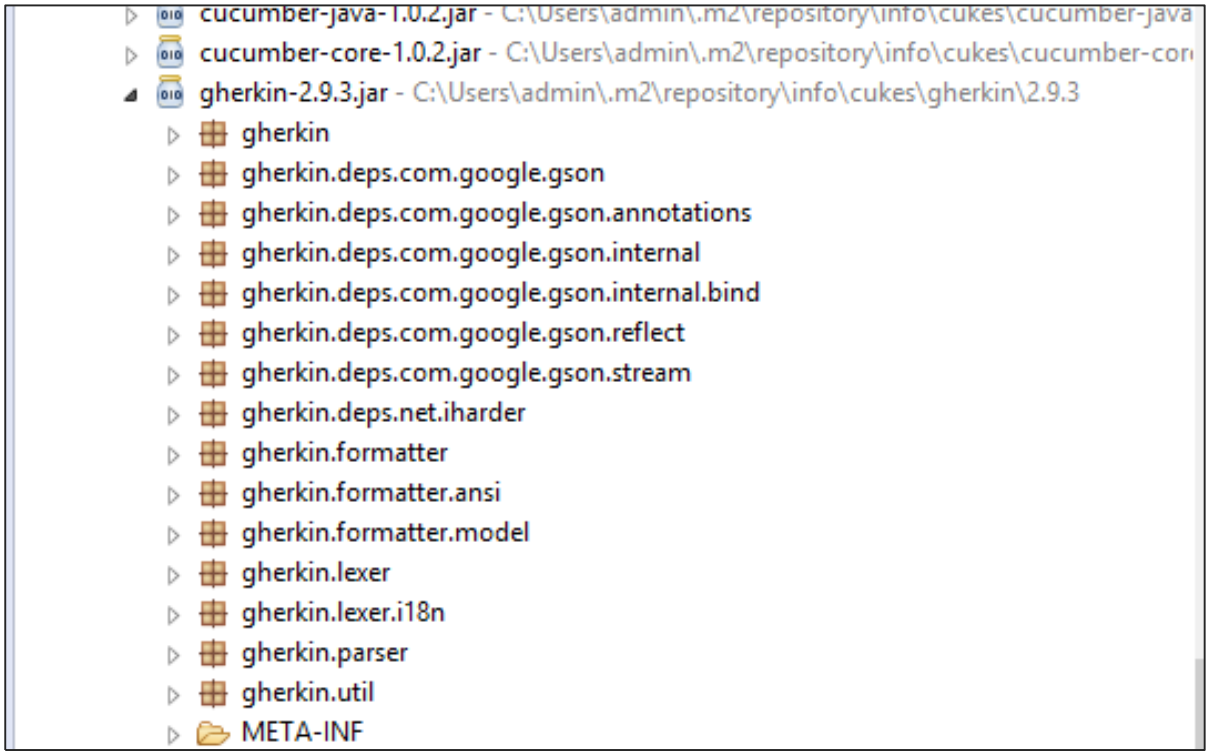
The above-mentioned scenario is of a feature called user login. All the words highlighted in bold are Gherkin keywords.

Example of few other keywords:

- Background
- But
- *
- Scenario Outline
- Examples

Gherkin will parse each step written in step definition file (to be covered later). So the steps mentioned in the feature file and the step definition file (to be covered later) should match.

You can locate the Gherkin jars in the Maven Dependency folder in the Package Explorer. It gets downloaded along with the other Cucumber jars. It will look like the following screenshot:



Another interesting fact about Gherkin is, it supports not only English but many other native languages such as French, Finnish, Indonesian, Hungarian, Hindi, Urdu, Gujarati, etc.

4. Cucumber – Features

A **Feature** can be defined as a standalone unit or functionality of a project. Let's take a very common example of a social networking site. How does the feature of this product/project look like? Few basic features can be determined as:

- Create and remove the user from the social networking site.
- User login functionality for the social networking site.
- Sharing photos or videos on the social networking site.
- Sending a friend request.
- Logout.

By now, it is clear that, each independent functionality of the product under test can be termed as a feature when we talk about Cucumber. It is a best practice later when you start testing, that before deriving the test scripts, we should determine the features to be tested.

A feature usually contains a list of scenarios to be tested for that feature. A file in which we store features, description about the features and scenarios to be tested is known as **Feature File**. We will see more about feature files in the following chapter.

The keyword to represent a feature under test in Gherkins is "Feature". The suggested best practice is, to write a small description of the feature beneath the feature title in the feature file. This will fulfill the need of a good documentation as well.

Example

Feature: Login functionality for a social networking site.

The user should be able to login into the social networking site if the username and the password are correct.

The user should be shown the error message if the username and the password are incorrect.

The user should be navigated to home page, if the username and password are correct.

Feature Files

The file, in which Cucumber tests are written, is known as **feature file**s. It is advisable that there should be a separate feature file, for each feature under test. The extension of the feature file needs to be ".feature".

One can create as many feature files as needed. To have an organized structure, each feature should have one feature file.

For Example:

Sr. No.	Feature	Feature File name
1	User Login	userLogin.feature
2	Share the Post	sharePost.feature
3	Create Account	createAccount.feature
4	Delete Account	deleteAccount.feature

The naming convention to be used for feature name, feature file name depends on the individual's choice. There is no ground rule in Cucumber about names.

A simple feature file consists of the following keywords/parts:

- **Feature**: Name of the feature under test.
- **Description** (optional): Describe about feature under test.
- **Scenario**: What is the test scenario.
- **Given**: Prerequisite before the test steps get executed.
- **When**: Specific condition which should match in order to execute the next step.
- **Then**: What should happen if the condition mentioned in WHEN is satisfied.

Example

Feature: User login on social networking site.
The user should be able to login into the social networking site when the username and the password are correct.
The user should be shown an error message when the username and the password are incorrect.
The user should be navigated to the home page if the username and the password are correct.

Outline: Login functionality for a social networking site.

The given user navigates to Facebook. When I enter Username as "<username>" and Password as "<password>". Then, login should be unsuccessful.

```
|username |password    |
|username1 |password1   |
```

* **AND** keyword is used to show conjunction between two conditions. **AND** can be used with any other keywords like **GIVEN**, **WHEN** and **THEN**.

There are no logic details written in the feature file.

Steps Definitions

We have got our feature file ready with the test scenarios defined. However, this is not the complete job done. Cucumber doesn't really know which piece of code is to be executed for any specific scenario outlined in a feature file.

This calls the need of an intermediate – Step Definition file. Steps definition file stores the mapping between each step of the scenario defined in the feature file with a code of function to be executed.

So, now when Cucumber executes a step of the scenario mentioned in the feature file, it scans the step definition file and figures out which function is to be called.

Example of Step Definition File

```
public void goToFacebook()
{ driver = new
FirefoxDriver();
driver.navigate().to("https://www.facebook.com/");
}

@When "^user logs in using Username as \"([^\"]*)\" and Password as
\"([^\"]*)\"$"
public void I_enter_Username_as_and_Password_as(String arg1, String arg2)
{ driver.findElement(By.id("email")).sendKeys(arg1);
driver.findElement(By.id("pass")).sendKeys(arg2);
driver.findElement(By.id("u_0_v")).click();
}

@Then"^login should be unsuccessful$"
public void validateRelogin() {
if(driver.getCurrentUrl().equalsIgnoreCase("https://www.facebook.com/login.php?
login_attempt=1&lwv=110")){
System.out.println("Test
Pass"); }
else {
System.out.println("Test
Failed"); }

driver.close();
}
```

So with each function, whatever code you want to execute with each test step (i.e. GIVEN/THEN/WHEN), you can write it within Step Definition file. Make sure that code/function has been defined for each of the steps.

This function can be Java functions, where we can use both Java and Selenium commands in order to automate our test steps.

5. Cucumber – Scenarios

Scenario is one of the core Gherkin structures. Every scenario starts with the keyword "Scenario:" (or localized one) and is followed by an optional scenario title. Each feature can have one or more scenarios and every scenario consists of one or more steps. A very simple example of scenario can be:

Scenario: Verify Help Functionality.
Given user navigates to Facebook.
When the user clicks on Help, then the Help page opens.

Consider a case, where we need to execute a test scenario more than once. Suppose, we need to make sure that the login functionality is working for all types of subscription holders. That requires execution of login functionality scenario multiple times. Copy paste the same steps in order to just re-execute the code, does not seem to be a smart idea. For this, Gherkin provides one more structure, which is scenario outline.

Scenario outline is similar to scenario structure; the only difference is the provision of multiple inputs. As you can see in the following example, the test case remains the same and non-repeatable. At the bottom we have provided multiple input values for the variables "Username" and "Password". While running the actual test, Cucumber will replace the variable with input values provided and it will execute the test. Once pass-1 has been executed, the test will rerun for second iteration with another input value. Such variable or placeholders can be represented with "<>" while mentioning with gherkin statements.

Example

Scenario Outline: Login functionality for a social networking site.
The given user navigates to Facebook.

When the user logs in using the Username as "<username>" and the Password as "<password>", then login should be successful.

```
|username    |password       |
|user1       |password1      |
|user2       |password2      |
```

There are a few tips and tricks to smartly define the Cucumber scenarios.

- Each step should be clearly defined, so that it does not create any confusion for the reader.

- Do not repeat the test scenario, if needed use scenario outline to implement repetition.

- Develop a test step in a way that, it can be used within multiple scenarios and scenario outlines.

- As far as possible, keep each step completely independent. For example: "Given the user is logged in". This can be divided into two steps:
    - Given the user enters the user name.
    - Clicks on login.

End of ebook preview
If you liked what you saw…
Buy it from our store @ **https://store.tutorialspoint.com**