Cucumber Cookbook

Today, behaviour-driven development (BDD) is gaining popularity around the world. Cucumber-JVM is one of the fastest growing tools and offers a cutting-edge platform for conceptualizing and implementing BDD. The variety of features available within Cucumber enhances the experience of implementing BDD for both business and development teams.

This book provides you with the skills you need to successfully create, customize, and configure the Cucumber framework. You will discover how to create Feature files, and Step Definitions, and enable various configurations, such as "Before" and "After" functions, with fun exercises. You’ll round off your learning by creating automation frameworks to automate Web, REST, and Mobile apps.

What this book will do for you...

- Explore the usage of the Gherkin Language to write meaningful and smart Feature files
- Understand Scenario, Steps, Backgrounds, Scenario Outlines, and Data Tables
- Discover the concepts of Glue Code and Step Definitions in detail
- Gain insights into the different types of Step Definitions, Regular Expressions, Doc Strings, Data Table transformations, and Capture Groups
- Master the advanced concepts of implementing Tags and Hooks
- Override default Cucumber options and settings along with different output report formats
- Run Jenkins and Cucumber from Terminal while running various Cucumber Scenarios in parallel

Inside the Cookbook...

- A straightforward and easy-to-follow format
- A selection of the most important tasks and problems
- Carefully organized instructions for solving problems efficiently
- Clear explanations of what you did
- Apply solutions to other situations

Over 35 hands-on recipes to efficiently master the art of behaviour-driven development using Cucumber-JVM
In this package, you will find:

- The author biography
- A preview chapter from the book, Chapter 1 'Writing Feature Files'
- A synopsis of the book’s content
- More information on Cucumber Cookbook

About the Author

Shankar Garg is an Agile enthusiast with expertise in automation testing. Currently, he is working as a senior consultant, testing with Xebia IT Architects, India.

He started as a Java developer but his love for breaking things got him into testing. He has worked on the automation of many projects for web, mobile, and SOA technologies. Right now, he is in love with Cucumber, Selenium, Appium, and Groovy.

Prior to working with Xebia, he worked for Jabong.com, Honeywell, and Tata Consultancy Services (TCS).

He is a Certified Scrum Master (CSM), Certified Tester (ISTQB), Certified Programmer for Java (SCJP 5.0) and Oracle 9i (OCA).
Cucumber Cookbook

Cucumber JVM is one of the fastest-growing tools that offer a cutting-edge platform to conceptualize and implement behaviour-driven development (BDD). The variety of features available within Cucumber bolsters and enhances experiences of implementing BDD for both business and development teams.

This cookbook has around 40 recipes. It takes you on a learning journey, where you start from basic concepts such as Feature files, Step Definitions, and then moves on to advanced concepts such as Hooks, Tags, configuration, and integration with Jenkins and test automation frameworks. Each chapter has multiple recipes, with the first recipe introducing the main concept of that chapter; the complexity level of each recipe increases as you progress through the chapter. The book has sufficient topics for product owners, business analysts, testers, developers, and everyone who wants to implement BDD.

This book is written with an assumption that the reader already has some idea about Cucumber. If you are new to Cucumber, it is advisable to go over my blog first:

- **Blog 1**: *How to integrate Eclipse with Cucumber plugin*  

- **Blog 2**: *Create a Cucumber Project by Integrating Maven-Cucumber-Selenium-Eclipse*  

These two blogs will help you integrate Cucumber and Eclipse and help you create and run a basic project.

All of the code explained in this book is committed on GitHub. Here is the URL of the code repository: [https://github.com/ShankarGarg/CucumberBook.git](https://github.com/ShankarGarg/CucumberBook.git).

This repository has five projects:

- **Cucumber-book-blog**: This project is used in the blogs mentioned earlier to start with Cucumber, Maven, and Eclipse
- **CucumberCookbook**: This project is used in chapters 1 to 5
- **CucumberWebAutomation, CucumberMobileAutomation, and CucumberRESTAutomation**: This project is used in *Chapter 6, Building Cucumber Frameworks*
What This Book Covers

Chapter 1, Writing Feature Files, covers unique aspect of Cucumber – the Gherkin language and usage of Gherkin language to write meaningful and smart Feature files. This chapter will also cover different keywords, such as files Scenario, Steps, Scenario Outlines, and Data Tables.

Chapter 2, Creating Step Definitions, covers basic concepts and usage of Glue Code/Step Definitions, and regular expressions to come up with efficient and optimized Step Definitions. This chapter will also elaborate the concept of String and Data Table transformations.

Chapter 3, Enabling Fixtures, covers the advanced concepts of implementing fixtures by Tags and Hooks. Here not only the individual concepts of Tags and Hooks are explained, but practical example of using Tags and Hooks combination is also explained.

Chapter 4, Configuring Cucumber, deals with integration of Cucumber with JUnit and the concept of Cucumber Options. Here you will learn various practical examples of using Cucumber Options and different types of reports that can be generated with Cucumber.

Chapter 5, Running Cucumber, covers topics of running Cucumber from the Terminal and from Jenkins. You will learn Cucumber integration with Jenkins and GitHub to implement Continuous Integration and Continuous Deployment (CICD) pipelines. Then you will learn parallel execution to take full advantage of Cucumber.

Chapter 6, Building Cucumber Frameworks, covers detailed Steps to create robust test automation frameworks to automate web applications, mobile applications, and REST services.
In this chapter, we will cover the following topics:

- Writing your first Feature file with one Scenario
- Creating Scenarios with different Steps
- Creating a Scenario with the And and But keywords
- Writing a Feature file with multiple Scenarios
- Adding Background to Feature files
- Sending multiple arguments in Steps
- Using complex data types to store data
- Implementing Scenario Outlines
- Creating a Feature file in a language other than English
- Combining Scenarios, Background, and Scenario Outlines

**Introduction**

In Cucumber Framework, business requirements are specified in Feature files, which are written in the Gherkin Language. So it becomes very important for us to understand the power and usage of the Gherkin language to come up with efficient and optimized Feature files.

This chapter will cover the usage of the Gherkin language to write meaningful and smart Feature files. We will start with some simple recipes to create a Feature file with one Scenario and will gradually move to recipes that are more complex where we create Feature files with multiple Scenarios, Backgrounds, and Scenario Outlines. We will also cover concepts and keywords, such as Feature, Scenario, Steps, Background, Scenario Outline and Data Tables.
In this chapter, we will only focus on Feature files. Step Definitions and automation libraries will be covered in later chapters. Initially, you may not understand everything about the concepts in this chapter, but things will become clearer as you read on.

Writing your first Feature file with one Scenario

Let's assume you are a Product Owner (PO) or a Business Analyst (BA). Your team is creating a web application and you need to write specifications for that application. A very simple and basic specification for that web application is when we enter the URL of that application in a browser, the application should load. So how do we write this specification in Cucumber? We will be covering this in this recipe.

How to do it...

In this recipe, we are going to create a simple Feature file with only one Scenario that tests whether the web page has loaded or not.

Let's create a page_load.feature file:

```gherkin
Feature: Test Git web Application
In order to Test Git web Application
As a user
I want to specify the application flow

Scenario: Web Site loads
application page load should be quick

Given application URL is ready with the user
When user enters the URL in browser
Then application page loads
```

How it works...

In Cucumber we write our requirements in plain English like Language, Gherkin. Gherkin is a domain-specific language that has a very well-defined syntax. It works on the basis of some predefined keywords. In the preceding example, the highlighted portions of the text are Gherkin's keywords and the rest is dependent on the application under test.

Let's understand each keyword in more detail.
Feature

In Cucumber, Feature files contain business requirements. The text that immediately follows the Feature keyword, and is in the same line, is the Title of the Feature file. Three (optional) Text lines that follow the Feature keyword line are Intent of the Feature file and intent text is whatever we want to write, up until the first Scenario. Feature file should contain either Scenario or Scenario Outline. The naming conventions for Feature files should be lowercase with underscores, for example, login.feature and home_page.feature. The names of Scenarios and Feature files must be unique.

Scenarios

Scenarios are like test cases and start with the Scenario keyword in a new line (different from the Feature intent). The text that immediately follows the Scenario keyword, and is on the same line, is the Scenario Title. Three (optional) Text lines that follow the Scenario keyword line are Intent of the Scenario. All Scenarios perform following:

- Get the system into a particular state
- Poke it (perform some action)
- Examine the new state

Steps

Scenarios contain Steps which are equivalent to test Steps and use the following keywords to denote them: Given, When, Then, But, and And (case sensitive).

When you save the Feature files mentioned in this chapter and run them, in the first run, Cucumber is going to give errors for the missing Step Definition files, along with suggestions for Step Definitions. To resolve these errors, copy the suggestions given by Cucumber and paste them into a default Step Definition file.

Creating Scenarios with different Steps

When we specify a business requirement, we need to specify the pre-conditions, user actions, and expected output. Let's first understand what each of these mean:

- **Pre-condition:** This sets the Application Under Test (AUT) in a state where the test case can be executed, or establishing the application context.
- **User action:** This refers to the action that a user performs that is in line with the Scenario objective.
- **Expected output:** This refers to the application's response after the user action.

So let's have this specification written in Cucumber in this recipe.
How to do it...

In this recipe, we are going to update the Feature file we created in the previous recipe by using the keywords Given, When and Then

Feature: login Page
   In order to test login page
   As a Registered user
   I want to specify the login conditions

Scenario: checking pre-condition, action and results
   Given user is on Application landing page
   When user clicks Sign in button
   Then user is on login screen

How it works...

A Cucumber Scenario consists of Steps identified with keywords such as Given, When, Then, And, But, and so on. These have been defined as follows:

- **Given**: Preconditions are mentioned in the `Given` keyword. The Steps of the `Given` keyword put the system in to a known state, which is necessary for the user action. Avoid talking about user interaction in `Given` Steps.
- **When**: The purpose of the `When` Steps is to describe the user action.
- **Then**: The purpose of `Then` Steps is to observe the expected output. The observations should be related to the business value/benefit of your Feature description.

Creating a Scenario with the And and But keywords

When we specify a business requirement, sometimes there are multiple pre-conditions, user actions, and expected outcomes. So how do we write these specifications in Cucumber?

Getting ready...

Based on what we have learned so far we know how to create Scenarios with one `Given`, `When`, and `Then` keyword. Now, if we need to add multiple `Steps`, then we can update our Feature file like this:

Feature: login Page
   In order to test login page
   As a Registered user
   I want to specify the login conditions
Scenario: without and & but
  Given user is on Application landing page
  Given Sign in button is present on screen
  When user clicks on Sign in button
  Then user can see login screen
  When user enters "ShankarGarg" in username field
  When user enters "123456" in password field
  When user clicks Sign in button
  Then user is on home page
  Then title of home page is "GitHub"

The problem here is that the keywords Given, When, and Then are repeated and the readability is thus affected. Having readable Feature files is one of biggest advantages of Cucumber. So how do we maintain the readability of Feature files? Let's figure this out in this recipe.

How to do it...

In this recipe, we are going to add one more Scenario and will use the And and But keywords:

Feature: login Page
  In order to test login page
  As a Registered user
  I want to specify the login conditions

Scenario: with and & but
  Given user is on Application landing page
  And Sign in button is present on screen
  When user clicks on Sign in button
  Then user is displayed login screen
  When user enters "ShankarGarg" in username field
  And user enters "123456" in password field
  And user clicks Sign in button
  Then user is on home page
  And title of home page is "GitHub"
  But Sign in button is not present

How it works...

The And and But keywords will be useful here. These keywords help to increase the expressiveness and readability of the Feature file:

- **And**: This is used for statements that are an addition to the previous Steps and represent positives statements.
- **But**: This is used for statements that are an addition to previous Steps and represent negative statements.
In a Step Definitions file, And and But are listed as Given/When/Then, the keyword that they appear after. There are no And and But keywords in Step Definitions.

Writing a Feature file with multiple Scenarios

Feature files contain possible Scenarios for a particular functionality. This is like writing all possible requirements that a Feature should meet when it is implemented. So let's write these specifications in Cucumber in the following section.

How to do it...

We will create a new Feature file called home_page.feature, which will cover the functionality of the default content of https://github.com/, the Bootcamp section, and the top banner content. We will create a different Scenario for each functionality. Take a look at the following screenshot for more clarity:

Feature: Home Page
In order to test Home Page of application
As a Registered user
I want to specify the features of home page

Scenario: Home Page Default content
Given user is on Github home page
Then user gets a GitHub bootcamp section
And username is also displayed on right corner
Scenario: GitHub Bootcamp Section
Given user is on GitHub home page
When user focuses on GitHub Bootcamp Section
Then user gets an option to setup git
And user gets an option to create repository
And user gets an option to Fork Repository
And user gets an option to work together

Scenario: Top Banner content
Given user is on GitHub home page
When user focuses on Top Banner
Then user gets an option of home page
And user gets an option to search
And user gets settings options
And user gets an option to logout

How it works...

A Cucumber Feature file can have any number of Scenarios as required. Some points to keep in mind are as follows:

- One Feature file normally focuses on one functionality of the application, such as login page, home page, and so on.
- One Scenario refers to one sub-Feature of that functionality, such as the new customer page, delete customer page, and so on.

When we have multiple Scenarios in a Feature file, we should always follow the Stateless Scenarios Guideline. Let's understand this guideline better—each Scenario must make sense and should be executed independently of any other Scenario. The result of one Scenario/Feature should not affect the other Scenario.

These are the benefits of independent Scenarios:

- Feature files are easier and fun to understand
- You can only run a subset of Scenarios, as all the required Steps of a Scenario are mentioned in the Scenario itself
- In comparison to dependent Scenarios, independent Scenarios will be more eligible candidates for parallel execution

Download the example code
You can download the example code files from your account at http://www.packtpub.com for all the Packt Publishing books you have purchased. If you purchased this book elsewhere, you can visit http://www.packtpub.com/support and register to have the files e-mailed directly to you.
Adding Backgrounds to Feature files

When we write Feature files, we write multiple Scenarios. Now all of these Scenarios start from one particular point. If I'm writing home page Scenarios, then I need to start the flow from the login functionality. So it is better to write the repetitive Steps at one place rather than in all Scenarios. Let's understand how to do this in the next Section.

Getting ready

Based on what we have learned so far, this is what our Feature file will look like:

Feature: Home Page
   In order to test Home Page of application
   As a Registered user
   I want to specify the features of home page

Scenario: Home Page Default content
   Given a registered user exists
   Given user is on GitHub login page
   When user enters username
   And user enters password
   And user clicks on login button
   Then user is on Application home page
   And user gets a GitHub bootcamp section

Scenario: GitHub Bootcamp Section
   Given user is on GitHub login page
   When user enters username
   And user enters password
   And user clicks on login button
   Then user is on Application home page
   When user focuses on GitHub Bootcamp Section
   Then user gets an option to setup git

Scenario: Top Banner content
   Given user is on GitHub login page
   When user enters username
   And user enters password
   And user clicks on login button
   Then user is on Application home page
   When user focuses on Top Banner
   Then user gets a logout option
The problem here is that first five statements are repeated in all the Scenarios. This affects the readability of the Feature files, and there is a lot of duplicated effort.

The problems with this way of writing Feature files are:

- **Repetition**: Many statements are repeated in all the Scenarios
- **Readability**: The readability of the Feature files is affected.
- **Duplication**: Copying these Steps to all the Scenarios is redundant.
- **Maintainability**: Even if a single Step changes, we have to change all occurrences.

**How to do it...**

We are going to update the `home_page.feature` file and we are going to use the `Background` keyword to put the common Steps across all the Scenarios in one place:

Feature: Home Page

In order to test Home Page of application
As a Registered user
I want to specify the features of home page

Background: flow till home page

Given user is on Application home page
When user enters username
   And user enters password
   And user clicks on login button
   Then user is on Application home page

Scenario: Home Page Default content

   Then user gets a GitHub bootcamp section

Scenario: GitHub Bootcamp Section

   When user focuses on GitHub Bootcamp Section
   Then user gets an option to setup git

Scenario: Top Banner content

   When user focuses on Top Banner
   Then user gets an option of home page
How it works...

Here, we have used the Background keyword. All the Steps mentioned in the Background keyword will be executed before each Scenario or Scenario Outline in a Feature file. Let's understand this keyword in greater detail:

- There can be only one Background in one Feature file and it allows us to set a precondition for all Scenarios in a Feature file.
- A Background is like a Scenario, containing a number of Steps.
- Background is run before each Scenario, but after the BeforeScenario Hooks. (We will read about Hooks in Chapter 3, Enabling Fixtures).
- The title and multiline description / intent of Background are optional.
- Since the Steps mentioned in Background will be run for all Scenarios in a Feature file, we need to be careful when adding the Steps to Background. For example, we should not add a Step that is not common to all Scenarios.

This is what the output of the preceding file looks like:
Don't use **Background** to set up a complicated state unless that state is actually something the client needs to know.

- Keep your **Background** section short because you expect a person to remember these Steps when you are adding a new Scenario
- Make your **Background** section vivid, because that way it will be easier for a person to remember it

## Sending multiple arguments in Steps

When we talk about testing, data-driven testing is a very famous approach. Until now, we have focused on what our Steps intend to do. The questions that now come to mind are as follows:

- Can our Steps also send test data?
- What kind of test data can our Steps send?
- Can we send mixed data types in one single Step?

Put on a BA's shoes and let's write some Scenarios for the GitHub user registration page and login functionality.

### How to do it...

We are going to update the *login.feature* file and add two Scenarios, where we are going to pass arguments in Steps:

```feature
Feature: login Page
In order to test login page
As a Registered user
I want to specify the login conditions

Scenario: New User Registration
Given user is on Application landing page
When user enters "ShankarGarg" in username field
And user enters "sgarg@gmail.com" in password field
And user enters "123456" in password field
And user clicks on Signup for GitHub button
Then user is successfully registered

Scenario: login
Given user is on Application landing page
And Sign in button is present on screen
When user clicks on Sign in button
Then user is displayed login screen
When user enters "ShankarGarg" in username field
```

- `Feature: login Page
  In order to test login page
  As a Registered user
  I want to specify the login conditions

Scenario: New User Registration
  Given user is on Application landing page
  When user enters "ShankarGarg" in username field
  And user enters "sgarg@gmail.com" in password field
  And user enters "123456" in password field
  And user clicks on Signup for GitHub button
  Then user is successfully registered

Scenario: login
  Given user is on Application landing page
  And Sign in button is present on screen
  When user clicks on Sign in button
  Then user is displayed login screen
  When user enters "ShankarGarg" in username field`
Writing Feature Files

And user enters "123456" in password field
And user clicks Sign in button
Then user is on home page
And title of home page is "GitHub"

How it works...

In the preceding Feature file, focus on the text written in " ". This is our test data. The text mentioned in between " " in Steps is associated to Capture groups in Step Definition files.

An example of Step Definition for one of the Steps is:

```java
@When("^user enters "(.*?)" in username field$")
public void user_enters_in_username_field(String userName) {
    //print the value of data passed from Feature file
    System.out.println(userName);
}
```

The output of the preceding `System.out.println` will be ShankarGarg (test data that we have passed in the Feature file).

Now, since you have learned how to pass test data in Steps, try your hand at the following:

- Send String and integer data in the same Step.
- Send a List in a Step; for example: "Monday, Tuesday, Wednesday".

Using complex data types to store data

In the previous recipe, we learnt how we can send data in Steps, which can be used by the application for processing. The data that we have sent up to this point has been either Strings or integers. But what if we want to send data structures that are more complex and span across multiple lines?

Getting ready

Let's write a Scenario for this functionality—we want to verify whether various users exist or not:

Scenario: Existing user Verification
Given user is on Application landing page
Then we verify user "Shankar" with password "P@ssword123", phone "999" exists
   Then we verify user "Ram" with password "P@ssword456", phone "888" exists
Then we verify user "Sham" with password "P@ssword789", phone "666" exists

The problem with this approach of writing Feature files is that Feature files are not expressive enough and there is a lot of repetition.

**How to do it...**

We are going to add one more Scenario to the login.feature file, and we are going to use Data Table to send a large set of test data along a Step:

**Scenario: Existing user Verification**

Given user is on Application landing page  
Then we verify following user exists

<table>
<thead>
<tr>
<th>name</th>
<th>email</th>
<th>phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shankar</td>
<td><a href="mailto:sgarg@email.com">sgarg@email.com</a></td>
<td>999</td>
</tr>
<tr>
<td>Ram</td>
<td><a href="mailto:ram@email.com">ram@email.com</a></td>
<td>888</td>
</tr>
<tr>
<td>Sham</td>
<td><a href="mailto:sham@email.org">sham@email.org</a></td>
<td>666</td>
</tr>
</tbody>
</table>

**How it works...**

Here we have used Data Tables. Tables as arguments to Steps are handy for specifying larger datasets. Let's understand Data Tables in more detail:

- Tables as arguments to Steps are handy to specify larger datasets.
- The first row of a Data Table is always the header row, where we specify the headers for each column. All the other rows in a Data Table are data rows, which contain the actual data that will be used by the application.
- Data tables will be passed to the Step Definition as the last argument.
- Don't confuse Data Tables with Scenario Outline tables.
- Data tables are very easy to handle in Step Definition files as well. This is what a sample Step Definition code looks like:

```java
@Then("we verify following user exists$")
public void we_verify_following_user_exists(DataTable userDetails){
    //Write the code to handle Data Table
    List<List<String>> data = userDetails.raw();
    System.out.println(data.get(1).get(1));
}
```

In the preceding code sample, the Data Table has been converted into a List of String and can be handled very easily thereafter.
Data table transformation has been explained in detail in the Transforming Data Tables to parse test data recipe in Chapter 2, Creating Step Definitions.

Implementing Scenario Outlines

In the previous recipe, we learnt how we can send test data in Steps itself, which can be used by the application for processing. Until now, the data was associated with one particular Step (implemented by Data Tables); but what if I want to send data which is related to the whole Scenario, and what if I want to repeat all the Steps of a Scenario again and again for different sets of data? This is a classic case of data-driven testing. This will be implemented by using a Scenario Outline.

Getting ready

Let's create a Scenario for a login functionality where we want to test all the possible Scenarios where the login will fail. Based on what we have learned so far, this is how our Scenario will look:

Scenario: login fail - wrong username
  Given user is on Application landing page
  When user clicks on Sign in button
  Then user is displayed login screen
  When user enters "wrongusername" in username field
  And user enters "123456" in password field
  And user clicks Sign in button
  Then user gets login failed error message

Scenario: login fail - wrong password
  Given user is on Application landing page
  When user clicks on Sign in button
  Then user is displayed login screen
  When user enters "ShankarGarg" in username field
  And user enters "wrongpassword" in password field
  And user clicks Sign in button
  Then user gets login failed error message

In terms of syntax, there is no problem in this code. Cucumber will treat it as well as any other, but the problem is for the person writing the Feature file. If you look closely, only the dataset is changing and all the other Steps are the same. These are the following problems with this approach to creating Feature files:
Copying and pasting Scenarios to use different values can quickly become tedious and repetitive.

If tomorrow only one Step changes, it has to be changed in all the Scenarios. So, maintainability and reusability are big issues.

To avoid these problems, let's look at the next section and understand how we can solve them.

**How to do it...**

Here, we are going to use the **Scenario Outline** keyword and add one Scenario Outline to test possible login Scenarios:

**Scenario Outline: Login fail - possible combinations**

Given user is on Application landing page
When user clicks on Sign in button
Then user is displayed login screen
When user enters "<UserName>" in username field
And user enters "<Password>" in password field
And user clicks Sign in button
Then user gets login failed error message

<table>
<thead>
<tr>
<th>UserName</th>
<th>Password</th>
</tr>
</thead>
<tbody>
<tr>
<td>wrongusername</td>
<td>123456</td>
</tr>
<tr>
<td>ShankarGarg</td>
<td>wrongpassword</td>
</tr>
<tr>
<td>wrongusername</td>
<td>wrongpassword</td>
</tr>
</tbody>
</table>

**How it works...**

Here we have used the **Scenario Outline** keyword and we have merged all three Scenarios in to one Scenario Outline. One advantage of the Scenario Outline is that our Feature file is now compact and expressive. Let's understand Scenario Outline in more detail:

- Scenario Outline allow us to send test data to Scenarios through the use of a template with placeholders.
- A Scenario Outline is run once for each row in the Examples section beneath it (not counting the first row of column headers).
- A Scenario Outline is a template that is never directly run. It uses placeholders, which are contained within `< >` in the Scenario Outline's Steps.
Think of a placeholder like a variable. It is replaced with a real value from the Examples table row, where the text between the placeholder’s angle brackets matches that of the table column header.

- In the first execution, when Cucumber encounters the first Step with placeholders, which is `When user enters <UserName> in username field in our case`, Cucumber looks for a column with the header `UserName` in the Examples table.
- If there is no column with `UserName` in the Examples table, then Cucumber does not give an error but instead considers `<UserName>` as a String and passes it to Step Definition as it is.
- If Cucumber finds a column with the header `UserName`, then it picks the first row data from this column and replaces `UserName` with that value, which is `wrongusername` in our case, and sends this value to Step Definition.
- Cucumber repeats this process for all `< >` for one round of execution.
- So, for the first execution, this is how our Scenario Outline will look:

```
Given user is on Application landing page
When user clicks on Sign in button
Then user is displayed login screen
When user enters "wrongusername" in username field
And user enters "123456" in password field
And user clicks Sign in button
Then user gets login failed error message
```

- The value substituted for the placeholder changes with each subsequent run of the Scenario Outline. The values from the second row are taken for the second execution and so on, until the end of the Examples table is reached.
- The Scenario Outline itself is useless without an Examples table, which Lists the rows of values to be substituted for each placeholder.

Now that you have learned the concept of Scenario Outline, try implementing the following:

- Write a Scenario Outline with multiple arguments in one Step.
- Can you create a Scenario Outline with multiple examples? You can group examples of positive tests and negative tests in different tables.
Creating a Feature file in a language other than English

Most of us have worked in teams spanning multiple geographies, and we would agree that some of us are more comfortable in native languages as compared to English. We are able to express ourselves better, and we are also able to express everything. So what if our BA or PO is more comfortable in Danish compared to English? Let's write the specification in a language other than English in Cucumber.

How to do it...

This is a sample English Feature file, which we will convert into different languages.

```
Feature: sample application
   In order to test login page
   As a Registered user
   I want to specify the login conditions

Scenario: sample scenario
   Given user is on application page
   When user clicks login button
   Then user is on home page
```

To create the Feature file in Danish (Danish.feature):

```
# language: da
Egenskab: prøve ansøgning
   For at teste login side
   Som registreret bruger
   Jeg ønsker at angive login betingelser

Scenarie: prøve scenario
   Givet brugeren er på ansøgning side
   Når brugeren klikker login knap
   SÅ Derefter bruger er på hjemmesiden
```
How it works...

Cucumber allows us to write Feature files in around 40 spoken languages, thus empowering the teams whose first language is not English to write Feature files which are as robust as English language Feature files. The header `# language: da` in the first line of the Feature tells Cucumber what language will be used in the Feature file. By default, the language is English. If we want to write Feature files in another language, the Feature files must be saved with "UTF-8" encoding.

In a single project, we can have Feature files in multiple languages; but for one Feature file, only one language will work.

For all languages, there is no difference in how Step definitions are generated.

Combining Scenarios, Backgrounds, and Scenario Outlines

Until now we have learned about Scenarios, Steps, Background, and Scenario Outline individually. But when a BA or a PO has to write the Feature file, they have to combine all these keywords to come up with a very efficient and expressive Feature file.

Consider writing a Feature file for a login functionality where the latter meets the following criteria:

- The user should get an option to log in from the application's home page
- To log in, a user should have a username and password
- After a successful login, the user should be redirected to the home page
- In case of an unsuccessful login, the user should get the appropriate message
- The user should also get an option to register new users on the home page
- The user should also be able to verify which users exist in the application (this Feature is not present on the GitHub landing page but has been added for to clarify concepts)

All these requirements are specific to the behavior of the application, and none of them are concerned with how you implement these requirements.
Chapter 1

How to do it...

Now we are going to use all the keywords we have explored until now, and we are going to create a login.feature file that specifies all the aforementioned requirements:

Feature: login Page
   In order to test login page
   As a Registered user
   I want to specify the login conditions

Scenario: login flow
   Given user is on Application landing page
   And Sign in button is present on screen
   When user clicks on Sign in button
   Then user is displayed login screen
   When user enters "ShankarGarg" in username field
   And user enters "123456" in password field
   And user clicks Sign in button
   Then user is on home page
   And title of home page is "GitHub"

Scenario Outline: Login fail - possible combinations
   Given user is on Application landing page
   When user clicks on Sign in button
   Then user is displayed login screen
   When user enters "<UserName>" in username field
   And user enters "<Password>" in password field
   And user clicks Sign in button
   Then user gets login failed error message

Examples:

<table>
<thead>
<tr>
<th>UserName</th>
<th>Password</th>
</tr>
</thead>
<tbody>
<tr>
<td>wrongusername</td>
<td>123456</td>
</tr>
<tr>
<td>ShankarGarg</td>
<td>wrongpassword</td>
</tr>
<tr>
<td>wrongusername</td>
<td>wrongpassword</td>
</tr>
</tbody>
</table>

Scenario: Existing user Verification
   Given user is on Application landing page
   Then we verify following user exists

<table>
<thead>
<tr>
<th>Name</th>
<th>Email</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shankar</td>
<td><a href="mailto:sgarg@email.com">sgarg@email.com</a></td>
<td>999</td>
</tr>
<tr>
<td>Ram</td>
<td><a href="mailto:ram@email.com">ram@email.com</a></td>
<td>888</td>
</tr>
<tr>
<td>Sham</td>
<td><a href="mailto:sham@email.org">sham@email.org</a></td>
<td>666</td>
</tr>
</tbody>
</table>
Scenario: New User Registration
Given user is on Application landing page
When user enters "ShankarGarg" in username field
And user enters "sgarg@gmail.com" in password field
And user enters "123456" in password field
And user clicks on Signup for GitHub button
Then user is successfully registered

How it works...

Here we have combined all the keywords and concepts discussed until now in this chapter. Let's go through each requirement one by one and analyze how and with which keyword we specified these requirements:

- User should get an option to log in from the application home page—Scenario
- For login, a user should have a username and password—Scenario
- After successful login, the user should be redirected to the home page—Scenario
- In case of unsuccessful login, the user should get the appropriate message—Scenario Outline
- The user should also get an option to register new users on the home page—Scenario
- The user should also be able to verify which users exist in the application—Data Table
Where to buy this book
You can buy Cucumber Cookbook from the Packt Publishing website.
Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.
Click here for ordering and shipping details.