Class 01: Introduction (BDD & Cucumber)

Agenda

- Introduction
  - TDD
  - BDD
  - Cucumber
- Feature files & Gherkin language
  - Feature file
  - Feature
  - Scenario
  - Given-when-then structure / Gherkin
- Java step def & Runner class
  - writing a step def
  - Running a feature file
  - Executing with test runner class

Old school approach

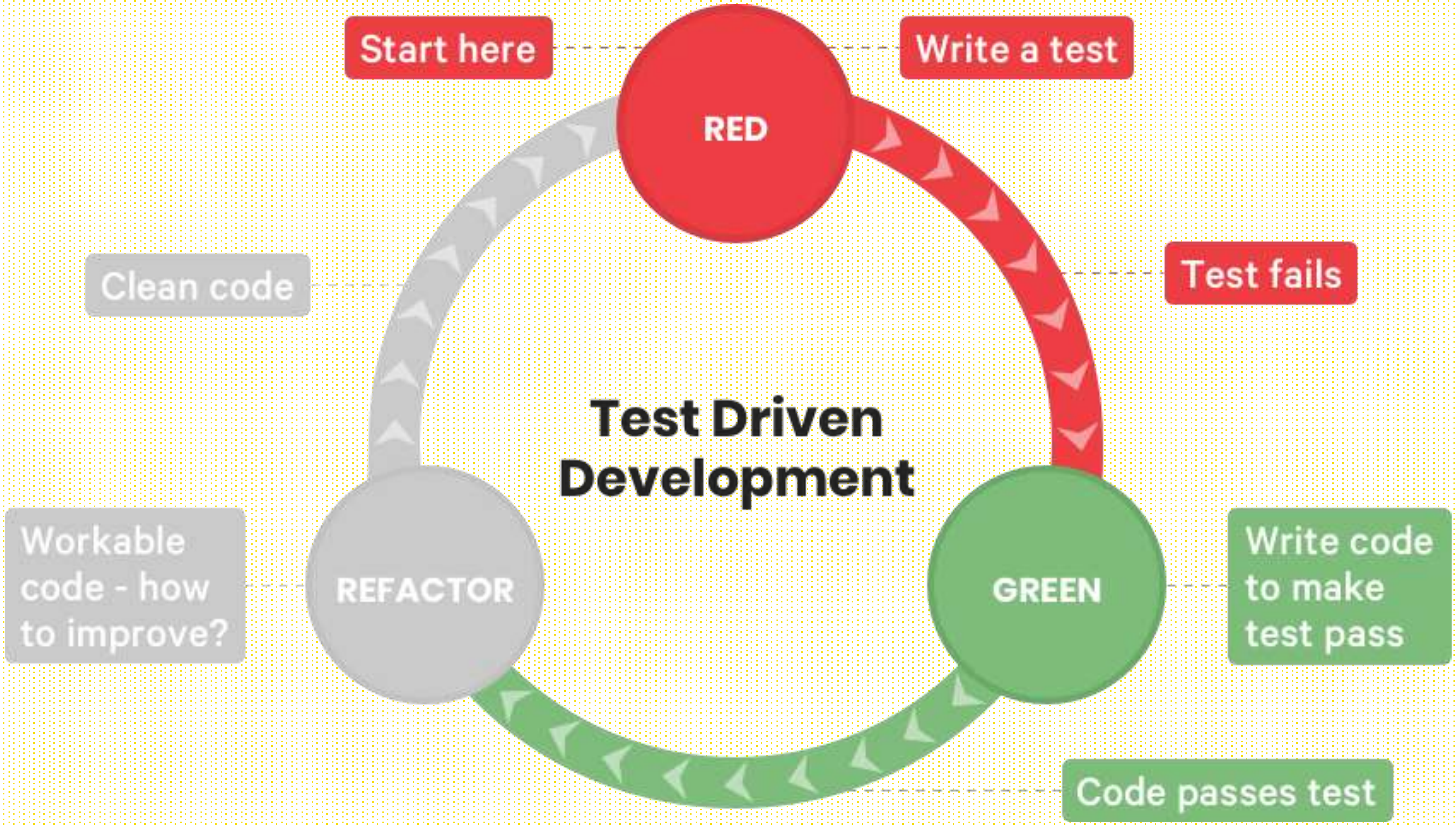
$$\begin{tikzcd}\text{Design} \arrow[r] & \text{Code} \arrow[r] & \text{Test} \arrow[ll, bend left]\end{tikzcd}$$

New school approach

$$\begin{tikzcd}
\text{Design} \arrow[r] & \text{Test} \arrow[r] & \text{Code} \arrow[ll, bend left]
\end{tikzcd}$$

TDD mantra

- Red—write a little test that doesn't work, perhaps doesn't even compile at first
- Green—make the test work quickly, committing whatever sins necessary in the process
- Refactor—eliminate all the duplication and smells created in just getting the test to work


Start here
Write a test
RED
Test fails
Clean code
Test Driven Development
Workable code - how to improve?
REFACTOR
GREEN
Write code to make test pass
Code passes test

Advantage of TDD

- Early bug notification
- Better Designed, cleaner and more extensible code
- Almost 100% test coverage
- It helps to clarify requirements

Behavior Driven Development - BDD

- Behaviors are written first, ideally by the business analyst, tester and developer
- Everyone understand how the application is supposed to be behaving, because it is basically in English
- AFTER everyone agreeing on the behaviors of the application, then the team starts building the functionalities.
- Therefore this minimizes the misunderstandings
  - Helping team to build the **right** application faster
  - Causing business to save money

BDD is like an extension of TDD

- TDD is more technical
- In TDD unit tests are written in the programming language of choice
- In BDD tests are written in Gherkin language, which makes it easier to understand to everyone
- But in both of these you are writing the tests first

Some tools used for TDD

- JUnit
- TestNG
- Mockito
- Karma

Some tools used for BDD

- Cucumber
- Jbehave
- SpecFlow

WHAT IS GHERKIN?

- Gherkin is the **language** that Cucumber understands. It is a **Business Readable**, Domain Specific Language, that lets you describe software’s behavior without detailing how that behavior is implemented.
- Gherkin is a business readable language used to express the **system’s behavior**. The language can be understood by an automation tool called Cucumber

Gherkin Keyword

- Given
- When
- Then
- And
- Scenario
- Feature
- Background
- Scenario Outline
- Examples

What’s Cucumber

- Cucumber is a software tool used by the testers to develop test cases for the testing of behaviour of the software
- In the Cucumber testing, the test cases are written in a simple English text, which anybody can understand without any technical knowledge. This simple English text is called the Gherkin language

Feature File

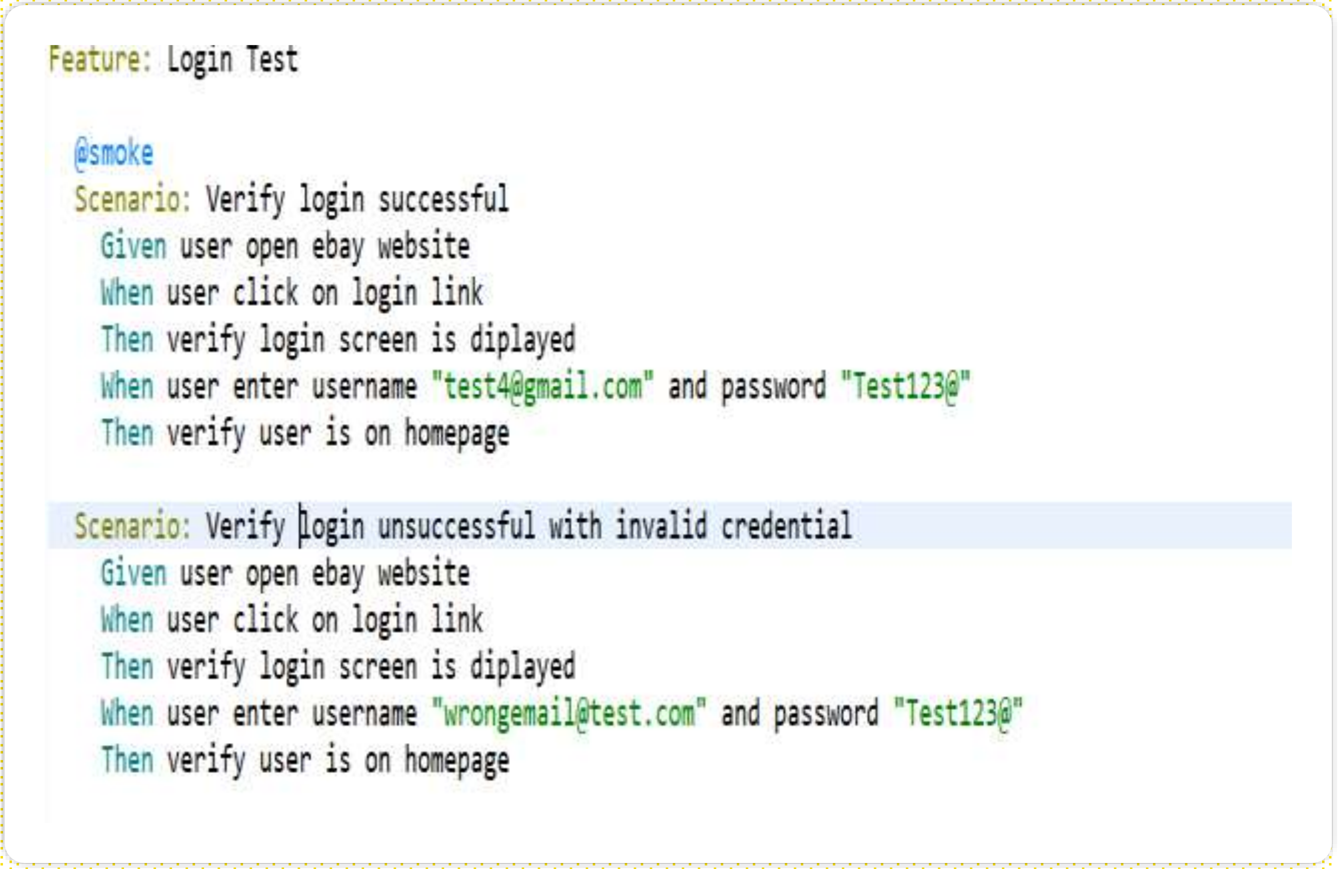
- A *Feature File* is an entry point to the *Cucumber* tests.
- This is a file where you will describe your tests in Descriptive language (Like English).
- It is an essential part of Cucumber, as it serves as an automation test script as well as live documents.
- A feature file can contain a scenario or can contain many scenarios in a single feature file but it usually contains a list of scenarios.

Feature

- Each feature file of Cucumber testing starts with a feature keyword
- It is a standalone unit or functionality to be tested. For example, login feature, payment transfer feature, registration feature, etc.

Scenario

- Each feature contains the required number of tests to test the feature. Each test is named as a Scenario.
- For example, feature login functionality can contain two scenarios, first for a successful login and second for unsuccessful login



```
Feature: Login Test

  @smoke
  Scenario: Verify login successful
    Given user open ebay website
    When user click on login link
    Then verify login screen is diplayed
    When user enter username "test4@gmail.com" and password "Test123@"
    Then verify user is on homepage

  Scenario: Verify login unsuccessful with invalid credential
    Given user open ebay website
    When user click on login link
    Then verify login screen is diplayed
    When user enter username "wrongemail@test.com" and password "Test123@"
    Then verify user is on homepage
```

What are steps?

- Using “Given, When, Then, And” keywords you will create your steps.
- Always write reusable steps
- Always re-use as many steps as you can

Keyword used in writing scenario

- **Given**
  - This keyword refers to the pre-condition of the test
  - For example, to access any web application, the first requirement or precondition is to navigate its home page

- **When**
  - It usually refers to the actions of a user that is to be executed

- **Then**
  - This keyword refers to the outcome of the previous step or upcoming action

- **And**
  - This keyword is used to add more conditions into your steps.

Step Definition

- A Step Definition is a Java method with an expression that links it to one or more Gherkin steps. When Cucumber executes a Gherkin step in a scenario, it will look for a matching step definition to execute.
- We can use cucumber annotation to map cucumber step with java method
- Annotations are - @Given, @When, @Then, @And

Runner Class

- Cucumber use junit runner class to run the tests.
- It uses @RunWith junit annotation that tells junit to run the tests as cucumber tests
- Cucumber options or configuration of Feature files, Step Definitions and Reporting will also be specified in the runner class
- **@RunWith**: This annotation tells junit to run the tests as Cucumber tests
- **@CucumberOptions**: Cucumber options is used to specify the features to executed, step definitions location, plugins, and etc.

@CucumberOptions

- **Features**
  - Features Options helps Cucumber to locate the Feature file in the project
  - All we need to do is to specify the folder path and Cucumber will automatically find all the '.features 'extension files in the folder
- **Glue**
  - It is almost the same think as Features Option but the only difference is that it helps Cucumber to locate the Step Definition file
- **Monochrome**
  - If it is set as true, it means that the console output for the Cucumber test are much more readable
- **DryRun**
  - If it is set as true, it means that Cucumber will only checks that every Step mentioned in the Feature File have corresponding code written in Step Definition file or not

Hooks

- In Cucumber, the hook is the block of code which can be defined with each scenario in step definition file by using the annotation **@Before** and **@After**.
- These **@Before** and **@After** annotations create a block in which we can write the code.

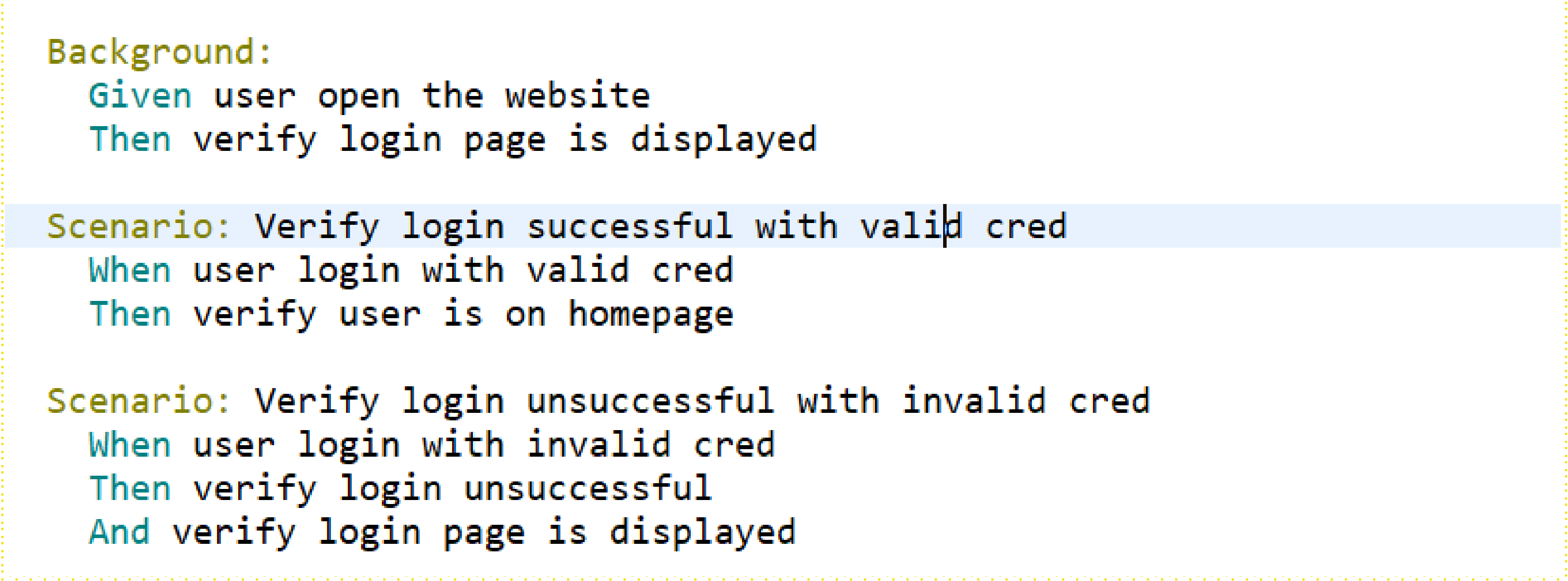
Need of Hooks

- Before Hook
  - To Start a web driver
  - Set up of Data Base connections
  - Set up of test data
  - Set up of browser cookies
  - Navigation to a certain page

- After Hook
  - To stop the web driver
  - To Close DB connections
  - To Clear the test data
  - To Clear browser cookies
  - To Log out from the application
  - Printing reports or logs
  - Taking the screenshots of error

Background Keyword

- Often you find that several scenarios in the same feature start with a common context.
- Cucumber provides a mechanism for this, by providing a Background keyword where you can specify steps that should be run before each scenario in the feature



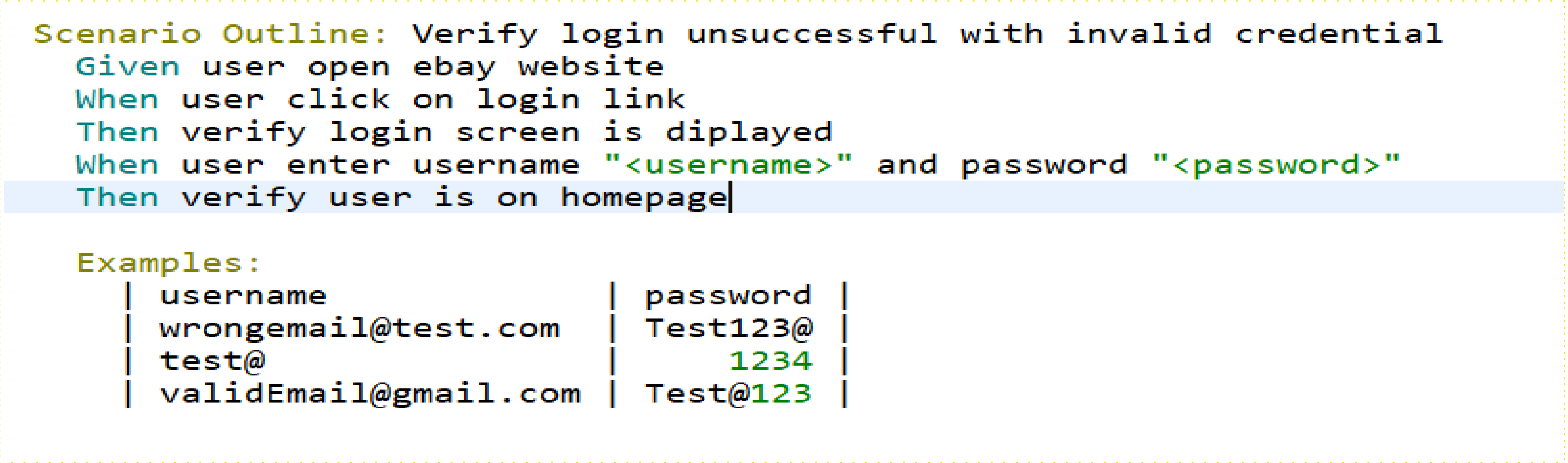
```
Background:
  Given user open the website
  Then verify login page is displayed

Scenario: Verify login successful with valid cred
  When user login with valid cred
  Then verify user is on homepage

Scenario: Verify login unsuccessful with invalid cred
  When user login with invalid cred
  Then verify login unsuccessful
  And verify login page is displayed
```

Scenario outline

- The Scenario Outline keyword can be used to run the same Scenario multiple times, with different combinations of values
- Scenario Outline will run scenario once for each row in the Examples section
- Cucumber automatically runs the complete test the number of times equal to the number of data in the Test Set
- Example tables always have a header row, because the compiler needs to match the header columns to the placeholders in the Scenario Outline’s steps

Scenario outline..

- When comparing a regular Scenario Definition with Scenario Outline, values no longer need to be hard-coded in step definitions. Values are replaced with parameters as <parameter_name> in step-definition itself.



```
Scenario Outline: Verify login unsuccessful with invalid credential
  Given user open ebay website
  When user click on login link
  Then verify login screen is diplayed
  When user enter username "<username>" and password "<password>"
  Then verify user is on homepage

  Examples:
    | username             | password |
    | wrongemail@test.com  | Test123@ |
    | test@                |     1234 |
    | validEmail@gmail.com | Test@123 |
```

Cucumber Tags

- Tag fulfils the following purposes:
  - If we have many scenarios in the feature file, to keep them in one group, we use tags in Cucumber, through which we will be able to prepare reports for specific scenarios under the same tag.
  - By default, Cucumber executes all the scenarios inside the feature file, but if we need to execute or skip any specific scenario under a specific test, so we can declare scenarios within a tag.
- Syntax:

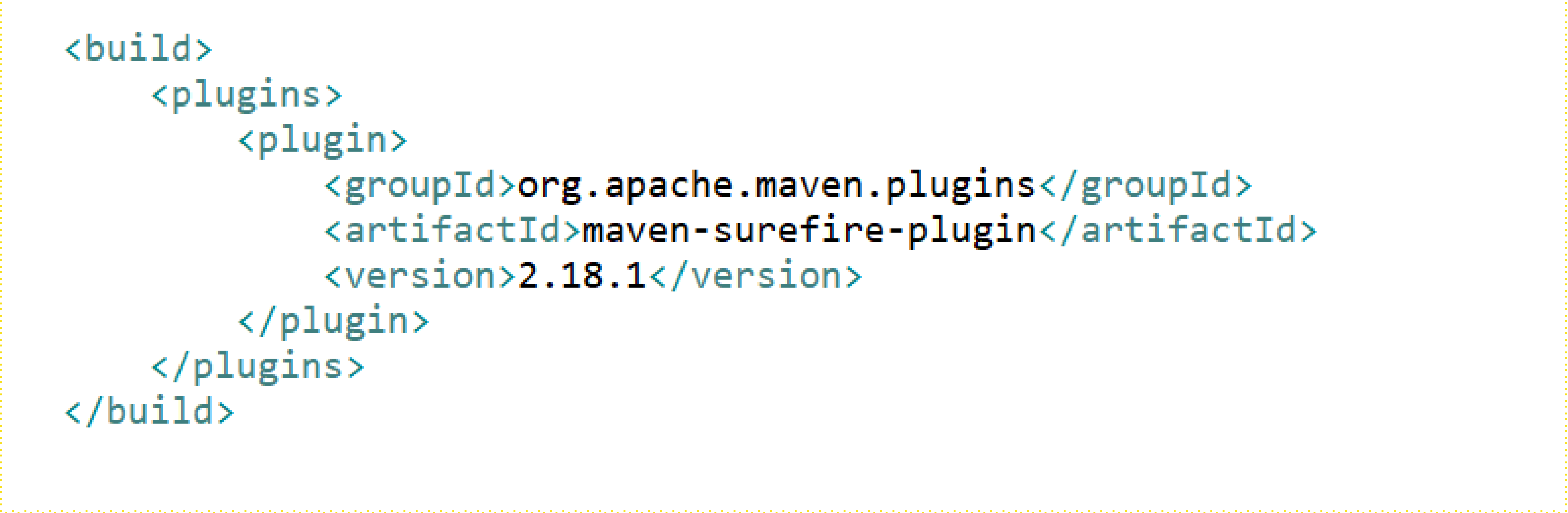
```
@TagName
Scenario: Mention the Scenario
```

Cucumber Report (HTML & Json) – Using Runner Class

- Runner class Plug-in - plug-in Option is used to specify different formatting options for the output reports.
    - HTML
    - JSON

Maven sure-fire plug-in

- The Surefire Plug-in is used during the test phase of the build lifecycle to execute the unit tests of an application

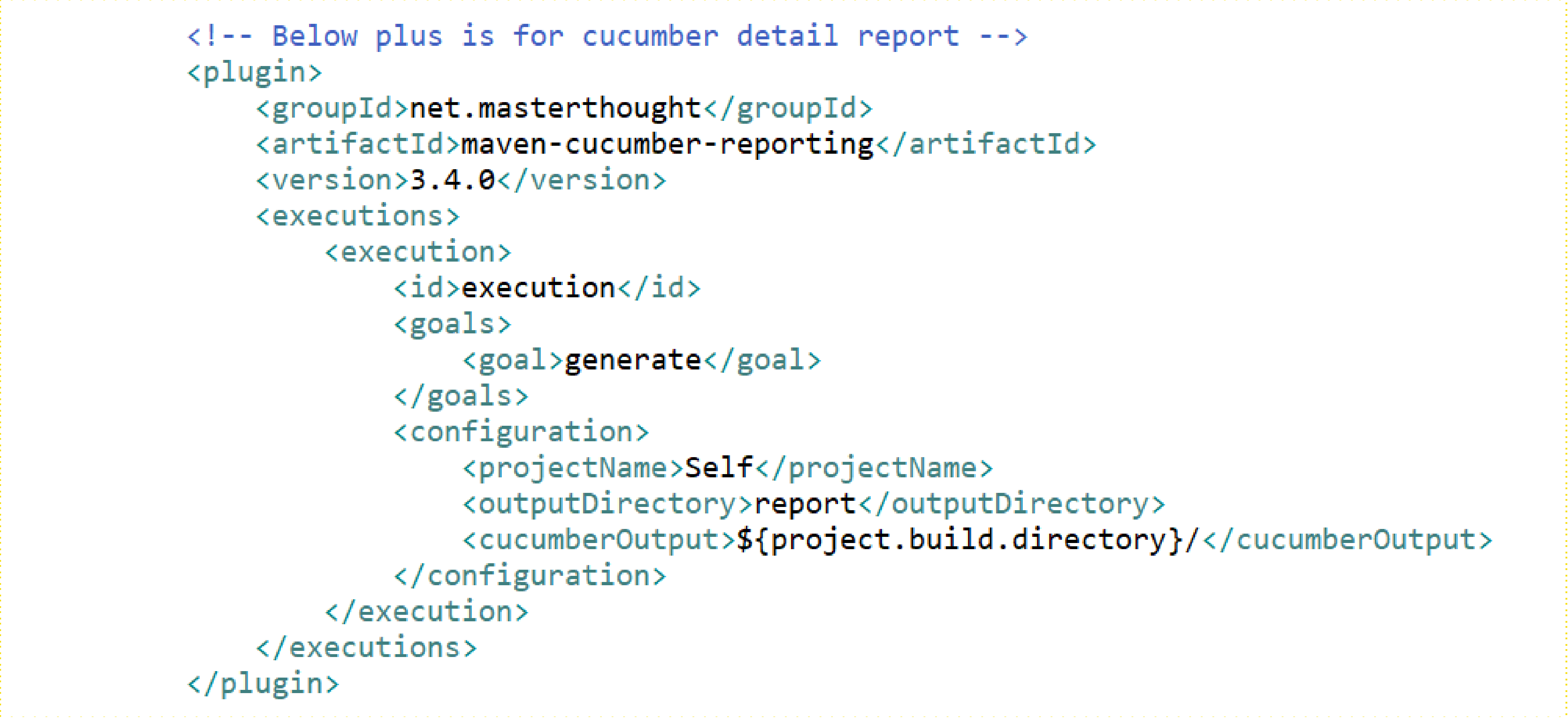


```xml
<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-surefire-plugin</artifactId>
            <version>2.18.1</version>
        </plugin>
    </plugins>
</build>
```

Cucumber Reporting

- Reporting plays a crucial component in a well-designed test automation framework
- Test automation report is the blue print that shows how the script was executed. It shows the scripts that were executed, steps in the test script which were executed, execution time, check points that were passed or failed, if failed what was the actual results and failure screenshot.

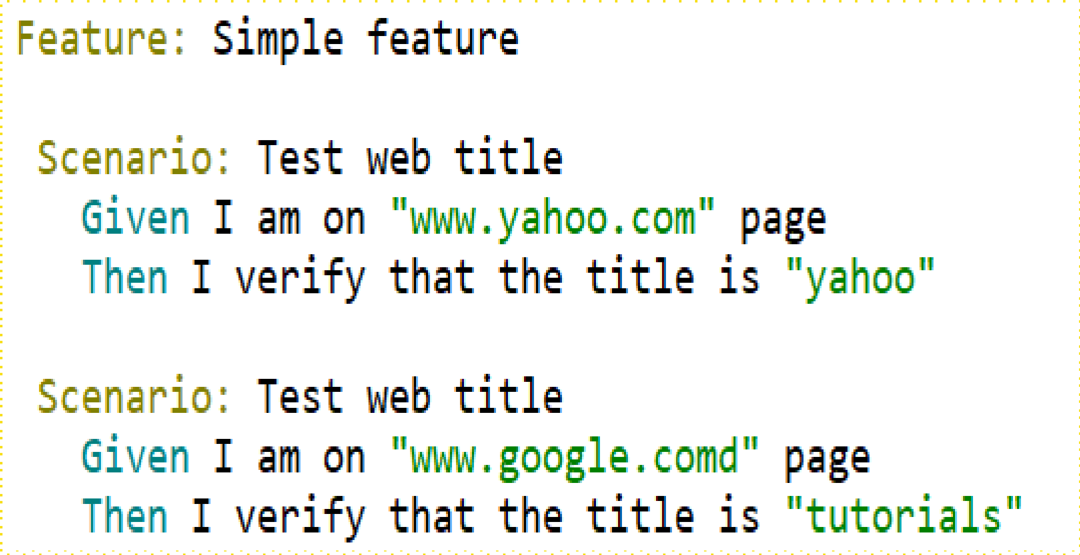
Cucumber Reporting plug-in



```xml
<!-- Below plus is for cucumber detail report -->
<plugin>
    <groupId>net.masterthought</groupId>
    <artifactId>maven-cucumber-reporting</artifactId>
    <version>3.4.0</version>
    <executions>
        <execution>
            <id>execution</id>
            <goals>
                <goal>generate</goal>
            </goals>
            <configuration>
                <projectName>Self</projectName>
                <outputDirectory>report</outputDirectory>
                <cucumberOutput>${project.build.directory}/</cucumberOutput>
            </configuration>
        </execution>
    </executions>
</plugin>
```

Parameter Through Scenario Step

- We can pass the parameters to the step methods from feature file as shown in below scenario. In below scenario, we have passed the name of website in Given step.
- Advantage of passing the parameters is that we can re-use same step method in different scenarios with different parameters



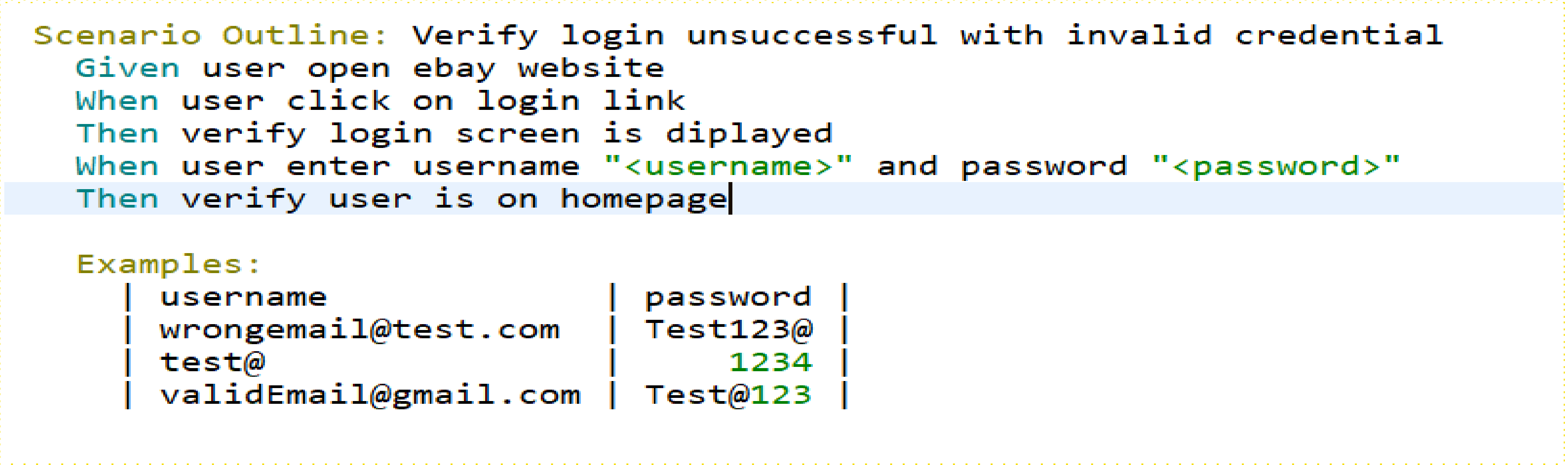
```
Feature: Simple feature

 Scenario: Test web title
   Given I am on "www.yahoo.com" page
   Then I verify that the title is "yahoo"

 Scenario: Test web title
   Given I am on "www.google.comd" page
   Then I verify that the title is "tutorials"
```

Scenario outline

- The Scenario Outline keyword can be used to run the same Scenario multiple times, with different combinations of values
- Scenario Outline will run scenario once for each row in the Examples section
- Cucumber automatically runs the complete test the number of times equal to the number of data in the Test Set
- Example tables always have a header row, because the compiler needs to match the header columns to the placeholders in the Scenario Outline’s steps

Scenario outline..

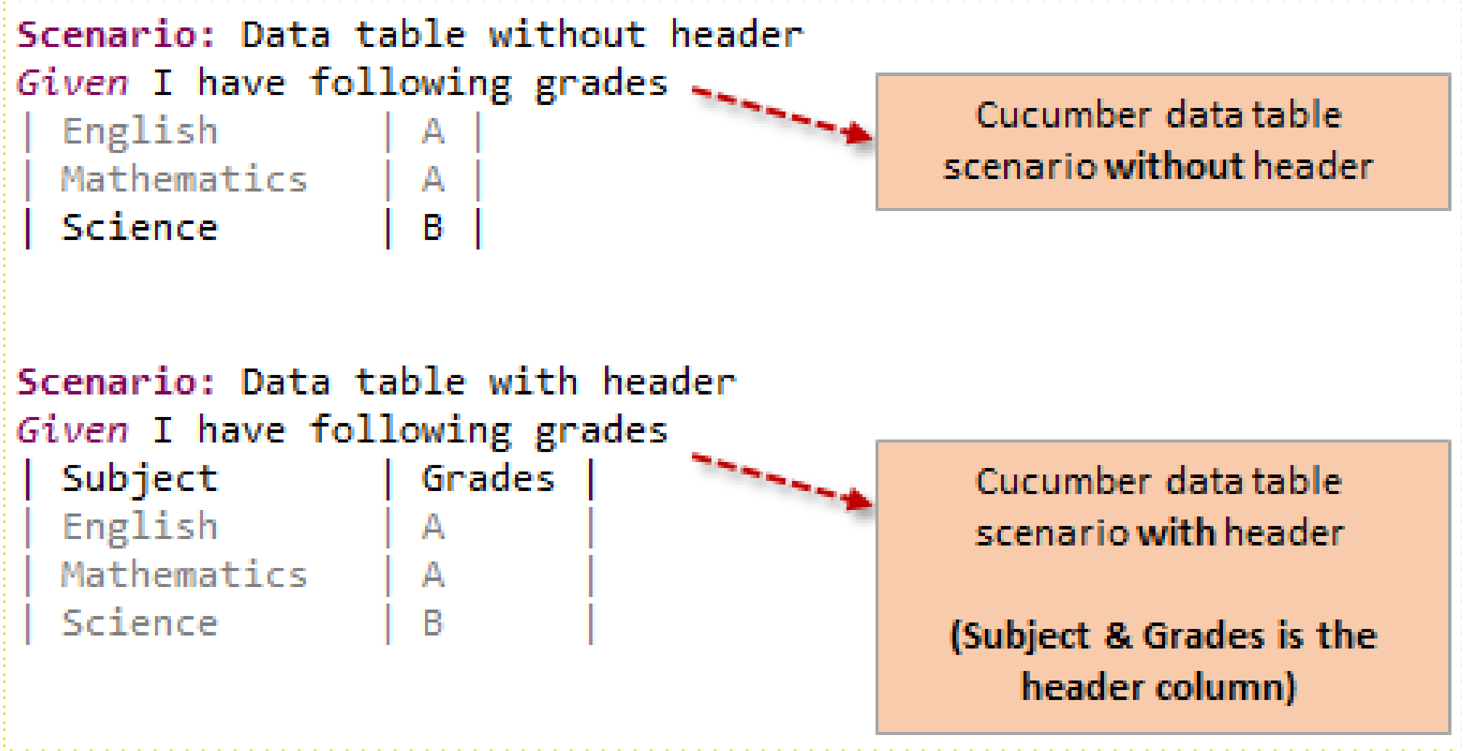
- When comparing a regular Scenario Definition with Scenario Outline, values no longer need to be hard-coded in step definitions. Values are replaced with parameters as <parameter_name> in step-definition itself.



```
Scenario Outline: Verify login unsuccessful with invalid credential
  Given user open ebay website
  When user click on login link
  Then verify login screen is diplayed
  When user enter username "<username>" and password "<password>"
  Then verify user is on homepage

  Examples:
    | username             | password |
    | wrongemail@test.com  | Test123@ |
    | test@                |     1234 |
    | validEmail@gmail.com | Test@123 |
```

Cucumber Data Tables

- You can add data tables in two different formats
  - Data table with a header (one & two dimension)
  - Data table without a header (one & two dimension)



```
Scenario: Data table without header
Given I have following grades
| English        | A |
| Mathematics    | A |
| Science        | B |


Scenario: Data table with header
Given I have following grades
| Subject        | Grades |
| English        | A      |
| Mathematics    | A      |
| Science        | B      |
```

Cucumber data table scenario without header
Cucumber data table scenario with header

(Subject & Grades is the header column)

Summary of Data tables

- Without Header
  - One column
    - *List<String>*
  - More than one column
    - *List<List<String>>*

- With header
  - One or More than one column
    - *List<Map<String,String>>*