

Behavior Driven Development

BDD :

- Behavior-Driven Development (BDD) is the software development process
- BDD is all about collaboration between the teams.
- Generates the documents that can be understood by all teams and stakeholders.

What is CUCUMBER :

- Cucumber is a tool that supports BDD.
- It understands the documents of the requirements and turns it into automated tests.

Traditional Development Process :

- Business owner tells Business Analyst (or) Product owner about his requirements.
- Product owner writes the requirements document. This should share to
 - a) Developer converts requirements into code.
 - b) Tester converts requirements into test cases.
- Test execution, bug fixing, retesting & regression.

Drawbacks :

Dev and QA team will not be involved in requirements discussion, user story creation and examples creation.

BDD PROCESS :

- Business owner tells Business Analyst (or) Product owner about his requirements.
- PO, DEV and QA meet to discuss the requirements and explore with example to define the system behavior.
- Examples are documented using specifications language like Gherkin that can be used for development and automation.
- Test execution, bug fixing, retesting, regression & release.

What is Gherkin ?

Gherkin is like a special language that helps you write down instructions in plain English for testing software.

Gherkin keywords

Feature :

- The purpose of the Feature keyword is to provide a high-level description of a software feature, and to group related scenarios.
- The first primary keyword in a Gherkin document must always be Feature, followed by a : and a short text that describes the feature.

Eg : **Feature: Guess the word**

Rule :

- It provides additional information for a feature.
- The purpose of the Rule keyword is to represent one business rule that should be implemented.

Eg : **Feature: Highlander**

Rule: There can be only One

Steps

- Each step starts with Given, When, Then, And, or But.

Given

- Given steps are used to describe the initial context of the system - the scene of the scenario.

Examples:

- Mickey and Minnie have started a game
- I am logged in
- Joe has a balance of £42

When

- When steps are used to describe an event, or an action. This can be a person interacting with the system, or it can be an event triggered by another system

Examples:

- Guess a word
- Invite a friend
- Withdraw money

Then

- Then steps are used to describe an expected outcome, or result

Examples:

- See that the guessed word was wrong
- Receive an invitation
- Card should be swallowed

And, But

- If you have successive Given's or Then's, you could write:

[Example: Multiple Givens](#)

Given one thing

Given another thing

Given yet another thing

When I open my eyes

Then I should see something

Then I shouldn't see something else

- Or, you could make the example more fluidly structured by replacing the successive Given's or Then's with And's and But's:

[Example: Multiple Givens](#)

Given one thing

And another thing

And yet another thing

When I open my eyes

Then I should see something

But I shouldn't see something else

- Gherkin also supports using an asterisk (*) in place of any of the normal step keywords. This can be helpful when you have some steps that are effectively a list of things

Scenario: All done

Given I am out shopping

* I have eggs

* I have milk

* I have butter

When I check my list

Then I don't need anything

Scenario:

- Definition: A Scenario is a single test case that describes a specific situation or feature to be tested.
- Purpose: It helps verify that a part of the software works as expected.

Scenario: Successful login

Given the user is on the login page

When the user enters valid credentials

Then the user is redirected to the dashboard

Scenario Outline:

- Definition: A Scenario Outline is used to run the same Scenario multiple times with different sets of data.
- Purpose: It helps test the same feature with various inputs efficiently.
- Scenario outlines allow us to more concisely express these scenarios through the use of a template with <>-delimited parameters:

Scenario Outline: Login with multiple users

Given the user is on the login page

When the user enters "<username>" and "<password>"

Then the user is redirected to the dashboard

Examples:

username password
user1 pass1
user2 pass2
user3 pass3

What are Step Definitions ?

- They link Gherkin instructions (written in plain English) to the actual code
- When Gherkin gives an instruction, step definitions tell the code what action to take.

Feature File:

Definition: A feature file is a plain text file containing one or more scenarios written using Gherkin language.

Purpose: It describes the functionality of the software feature in a way that is easy for both non-technical stakeholders and developers to understand.

Structure: Each feature file starts with the Feature keyword, followed by a brief description of the feature. It then contains one or more Scenario or Scenario Outline sections.

Feature: User login

Scenario: Successful login with valid credentials

Given the user is on the login page

When the user enters valid username and password

Then the user is redirected to the dashboard

Scenario Outline: Unsuccessful login with invalid credentials

Given the user is on the login page

When the user enters "<username>" and "<password>"

Then an error message is displayed

Examples:

| username | password |

| invalid1 | wrongpass |

| invalid2 | wrongpass |

Maven project creation and cucumber installation :

- Create new maven project
- Add maven dependencies

```
<dependency> // <!-- https://mvnrepository.com/artifact/junit/junit -->
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.12</version>
    <scope>test</scope>
</dependency>
```

```

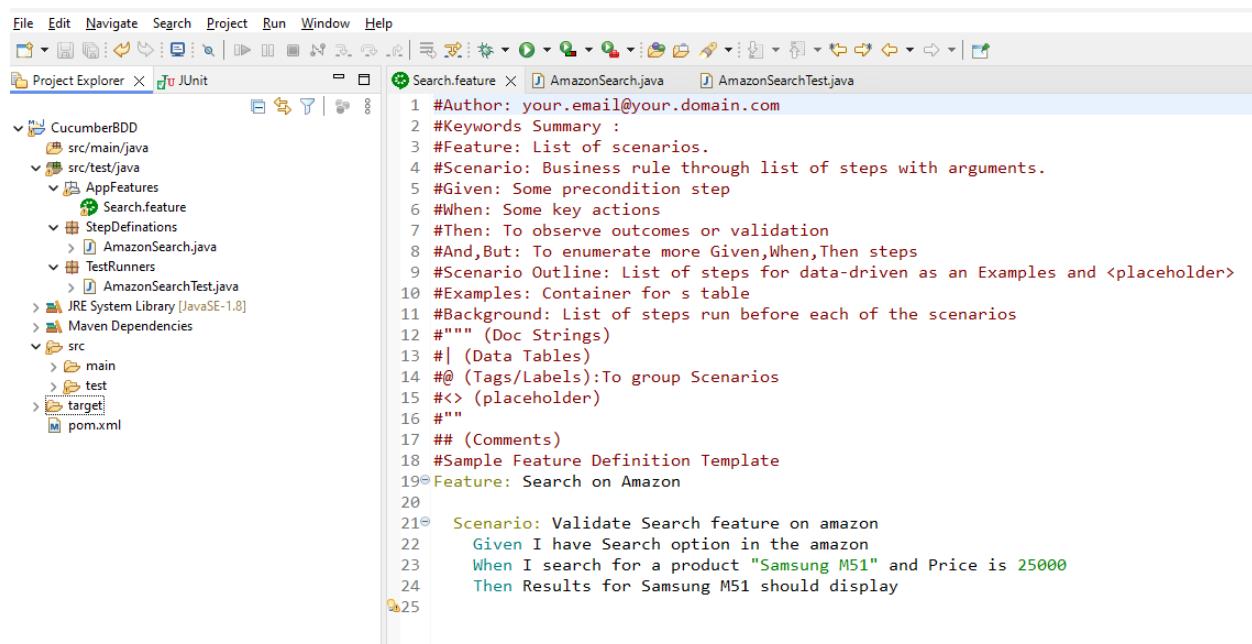
<!-- https://mvnrepository.com/artifact/io.cucumber/cucumber-java -->
<dependency>
    <groupId>io.cucumber</groupId>
    <artifactId>cucumber-java</artifactId>
    <version>7.15.0</version>
</dependency>
<!-- https://mvnrepository.com/artifact/io.cucumber/cucumber-junit -->
<dependency>
    <groupId>io.cucumber</groupId>
    <artifactId>cucumber-junit</artifactId>
    <version>7.16.1</version>
    <scope>test</scope>
</dependency>

```

- Download cucumber plugin from eclipse marketplace
- Create a folder for “Features” under src/test/resources
- Under features folder create new feature file with **.feature** extension
- In feature file add contents → Feature, Scenario, Steps, Scenario Outline, Examples, Tags, Comments
- Try to Run Feature file
- Add step Definition/Glue code under src/test/java package

Automation Test Case Creation Process :

1) Create feature file → 2) Create Step Definition class for feature scenarios → 3) create runner class



The screenshot shows the Eclipse IDE interface with the following details:

- File Bar:** File, Edit, Navigate, Search, Project, Run, Window, Help.
- Project Explorer View:** Shows a project named "CucumberBDD" with the following structure:
 - src/main/java
 - src/test/java
 - AppFeatures
 - Search.feature
 - StepDefinitions
 - AmazonSearch.java
 - TestRunners
 - AmazonSearchTest.java
 - JRE System Library [JavaSE-1.8]
 - Maven Dependencies
 - src
 - main
 - test
 - target
 - pom.xml
- Editor View:** Displays the content of the "Search.feature" file.

```

1 #Author: your.email@your.domain.com
2 #Keywords Summary :
3 #Feature: List of scenarios.
4 #Scenario: Business rule through list of steps with arguments.
5 #Given: Some precondition step
6 #When: Some key actions
7 #Then: To observe outcomes or validation
8 #And,But: To enumerate more Given,When,Then steps
9 #Scenario Outline: List of steps for data-driven as an Examples and <placeholder>
10 #Examples: Container for s table
11 #Background: List of steps run before each of the scenarios
12 """ (Doc Strings)
13 #| (Data Tables)
14 #@ (Tags/Labels):To group Scenarios
15 #<> (placeholder)
16 """
17 ## (Comments)
18 #Sample Feature Definition Template
19@ Feature: Search on Amazon
20
21@ Scenario: Validate Search feature on amazon
22    Given I have Search option in the amazon
23    When I search for a product "Samsung M51" and Price is 25000
24    Then Results for Samsung M51 should display
25

```

File Edit Source Refactor Navigate Search Project Run Window Help

Project Explorer X JUnit

```

1 package StepDefinitions;
2
3 import io.cucumber.java.en.Given;
4 import io.cucumber.java.en.Then;
5 import io.cucumber.java.en.When;
6
7 public class AmazonSearch {
8
9     @Given("I have Search option in the amazon")
10    public void i_have_search_option_in_the_amazon() {
11        // Write code here that turns the phrase above into concrete actions
12        System.out.println("Search Option is present on amazon page");
13    }
14
15    @When("I search for a product {string} and Price is {int}")
16    public void i_search_for_a_product_and_price_is(String product, Integer price) {
17        // Write code here that turns the phrase above into concrete actions
18        System.out.println("Searched for a product " + product + " and price " + price);
19    }
20
21    @Then("Results for Samsung M51 should display")
22    public void results_for_samsung_m51_should_display() {
23        // Write code here that turns the phrase above into concrete actions
24        System.out.println("Results for search product is displayed");
25    }
26
27

```

File Edit Source Refactor Navigate Search Project Run Window Help

Project Explorer X JUnit

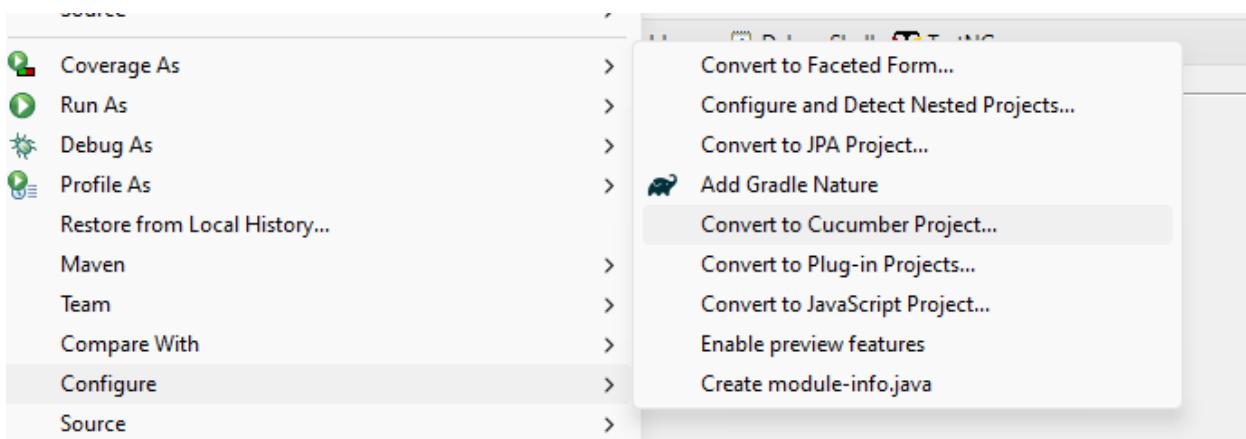
```

1 package TestRunners;
2
3 import org.junit.runner.RunWith;
4 import io.cucumber.junit.Cucumber;
5 import io.cucumber.junit.CucumberOptions;
6
7 @RunWith(Cucumber.class)
8 @CucumberOptions(features = { "src\\test\\java\\AppFeatures" }, // provide feature files path
9                 glue = { "StepDefinitions" }, // stepdefinition package path
10                plugin = {"pretty", "html:target/HtmlReports/report.html"}
11
12                // The Pretty plugin provides a verbose output,
13                // making it easier to understand what's happening during test execution
14 public class AmazonSearchTest {
15
16 }
17

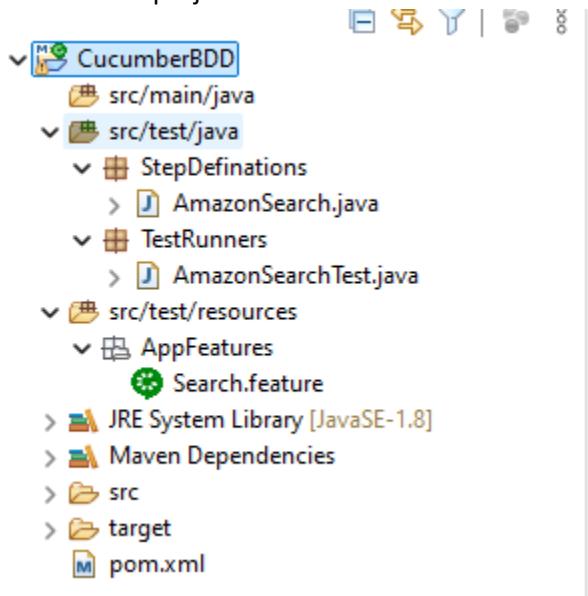
```

Map feature file steps with step definition methods in step definition class :

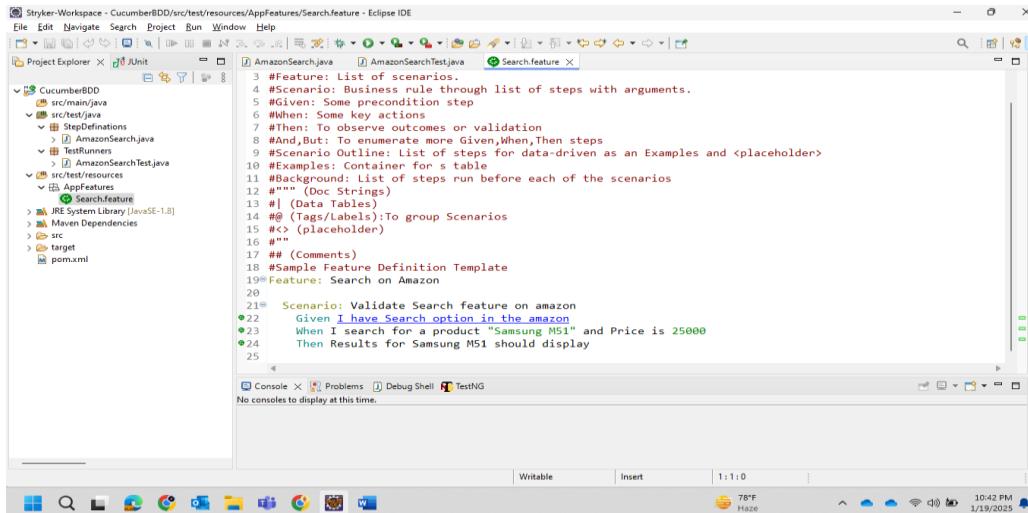
- Right click on project -> configure -> Convert to Cucumber Project



- Now maven project converted in to maven cucumber project



- Control + click to navigate to step definition methods of feature file scenarios.



The screenshot shows the Eclipse IDE with the 'AmazonSearch.java' file open in the editor. The code is as follows:

```

import io.cucumber.java.en.Given;
import io.cucumber.java.en.Then;
import io.cucumber.java.en.When;

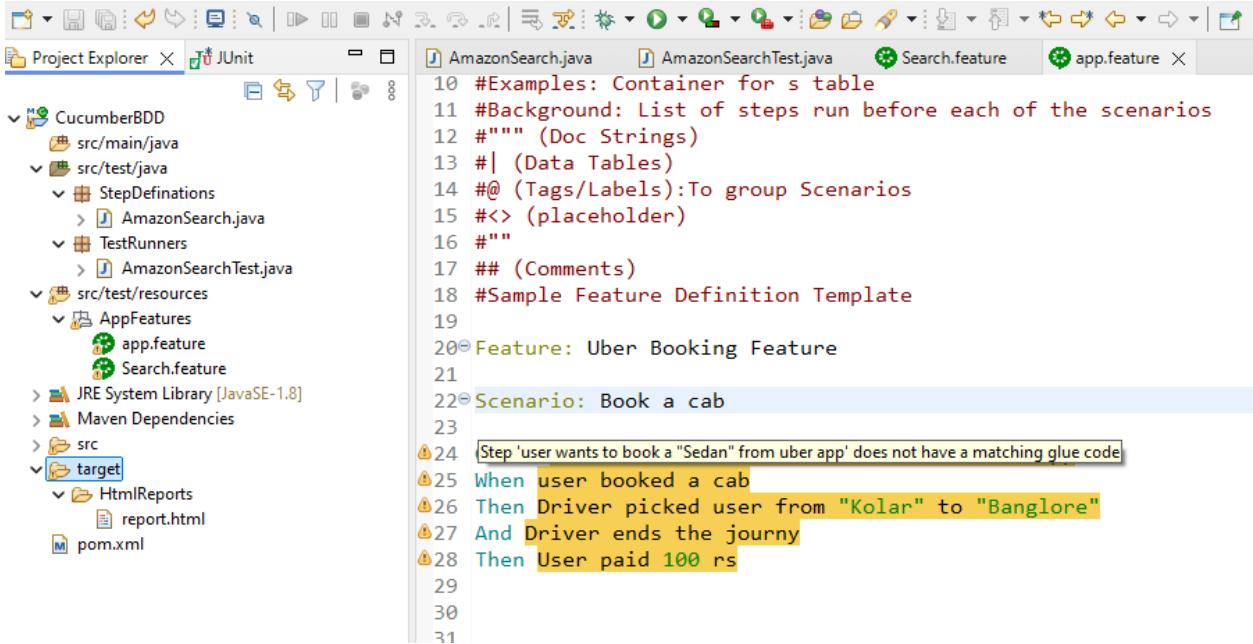
public class AmazonSearch {

    @Given("I have Search option in the amazon")
    public void i_have_search_option_in_the_amazon() {
        // Write code here that turns the phrase above into concrete actions
        System.out.println("Search Option is present on amazon page");
    }

    @When("I search for a product {string} and Price is {int}")
    public void i_search_for_a_product_and_price_is(String product, Integer price) {
        // Write code here that turns the phrase above into concrete actions
        System.out.println("Searched for a product " + product + " and price " + price);
    }
}

```

- The moment you created feature file with scenarios, cucumber plug in warns you to create step Definition or Glue code



The screenshot shows the Eclipse IDE interface with the Project Explorer view on the left and the Editor view on the right. The Project Explorer shows a Java project named 'CucumberBDD' with packages like 'src/main/java', 'src/test/java', and 'src/test/resources'. In the Editor view, there are two tabs: 'Search.feature' and 'app.feature'. The 'Search.feature' tab contains a Cucumber feature definition:

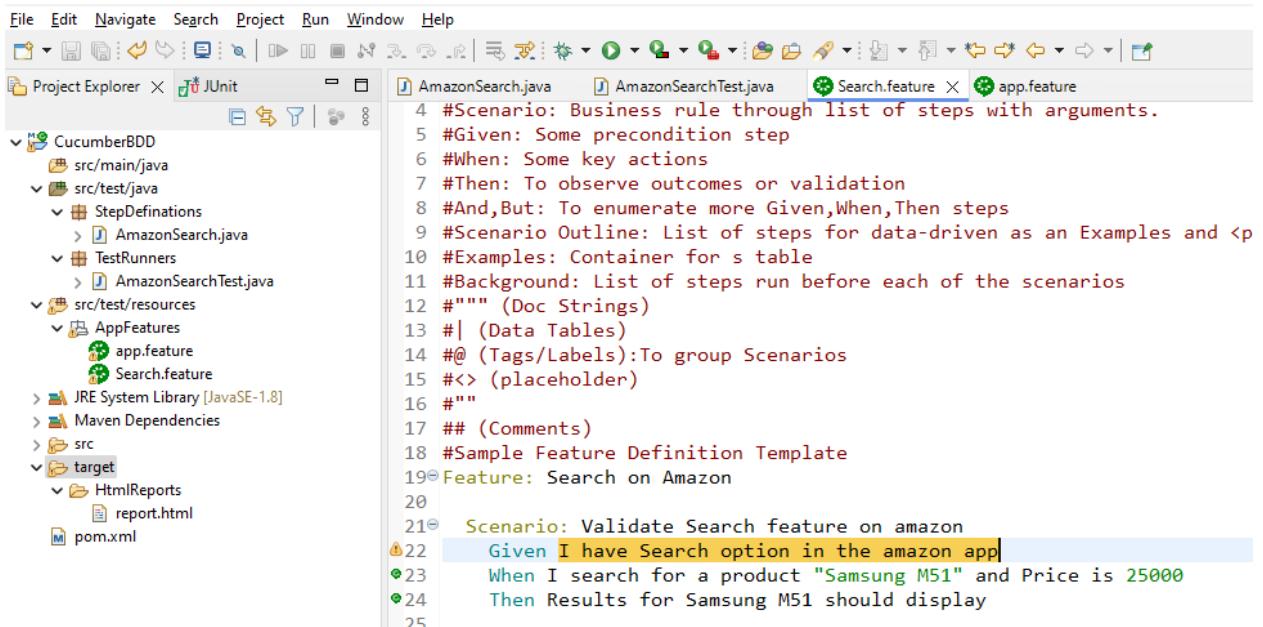
```

10 #Examples: Container for s table
11 #Background: List of steps run before each of the scenarios
12 """
13 #| (Data Tables)
14 #@ (Tags/Labels):To group Scenarios
15 #<> (placeholder)
16 """
17 ## (Comments)
18 #Sample Feature Definition Template
19
20@Feature: Uber Booking Feature
21
22@Scenario: Book a cab
23
24 Step 'user wants to book a "Sedan" from uber app' does not have a matching glue code
25 When user booked a cab
26 Then Driver picked user from "Kolar" to "Banglore"
27 And Driver ends the journey
28 Then User paid 100 rs
29
30
31

```

A yellow warning box highlights the step 'Step 'user wants to book a "Sedan" from uber app'' with the message 'does not have a matching glue code'.

- If you modify any scenario then also we will get the warning



The screenshot shows the Eclipse IDE interface with the Project Explorer view on the left and the Editor view on the right. The Project Explorer shows a Java project named 'CucumberBDD' with packages like 'src/main/java', 'src/test/java', and 'src/test/resources'. In the Editor view, there are two tabs: 'Search.feature' and 'app.feature'. The 'Search.feature' tab contains a Cucumber feature definition:

```

4 #Scenario: Business rule through list of steps with arguments.
5 #Given: Some precondition step
6 #When: Some key actions
7 #Then: To observe outcomes or validation
8 #And,But: To enumerate more Given,When,Then steps
9 #Scenario Outline: List of steps for data-driven as an Examples and <
10 #Examples: Container for s table
11 #Background: List of steps run before each of the scenarios
12 """
13 #| (Data Tables)
14 #@ (Tags/Labels):To group Scenarios
15 #<> (placeholder)
16 """
17 ## (Comments)
18 #Sample Feature Definition Template
19@Feature: Search on Amazon
20
21@ Scenario: Validate Search feature on amazon
22 Given I have Search option in the amazon app
23 When I search for a product "Samsung M51" and Price is 25000
24 Then Results for Samsung M51 should display
25

```

A yellow warning box highlights the step 'Given I have Search option in the amazon app' with the message 'does not have a matching glue code'.

Regular Expression In Cucumber :

- Regular Expressions → [0-9]+, (\d+)
- Cucumber Expressions → {String}, {int}

Rules:

- Step def file will be generating cucumber expression by default.
- But we can also use Regular expression in step def file.
- We can mix both cucumber and regular expression in step def file.
- We can't mix both expression in a step def method.

Regular Expressions:

Quantifiers in Reg Exp : + , * , ? , {n}

- How many times a character needs to be occurred

([0-9]+) : 0 to 9 digits appear (once or more)

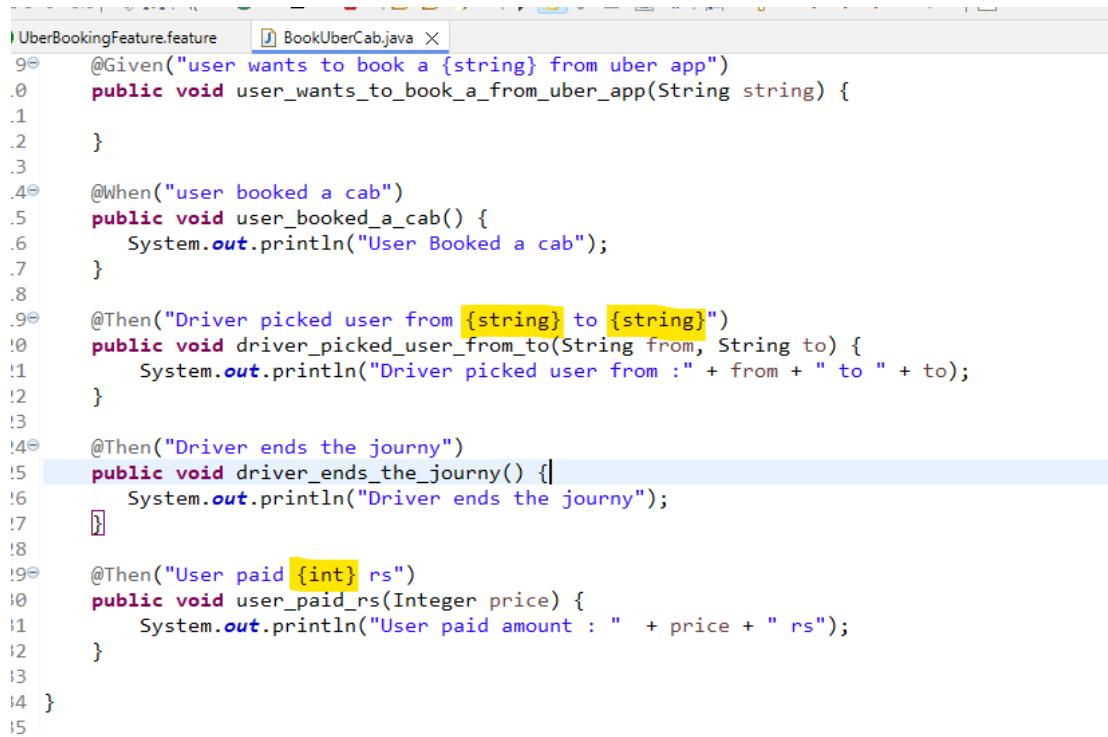
([0-9]{4}) : 0000 , 9999, 1212, 3456, 1234, 8888

([0-9]*) : zero or more

([0-9]?) : zero or once

Predefined shorthand character classes - Java 9 Regular Expressions [Book]

■ Cucumber Expressions



The screenshot shows a Java code editor with a tab bar containing 'UberBookingFeature.feature' and 'BookUberCab.java'. The code is a Cucumber step definition class named 'BookUberCab.java'. It contains five annotated methods: @Given("user wants to book a {string} from uber app"), @When("user booked a cab"), @Then("Driver picked user from {string} to {string}"), @Then("Driver ends the journey"), and @Then("User paid {int} rs"). The code uses System.out.println to print the step details. Lines 1 through 15 are numbered on the left.

```
UberBookingFeature.feature BookUberCab.java
9@ Given("user wants to book a {string} from uber app")
10 public void user_wants_to_book_a_from_uber_app(String string) {
11
12 }
13
14@ When("user booked a cab")
15 public void user_booked_a_cab() {
16     System.out.println("User Booked a cab");
17 }
18
19@ Then("Driver picked user from {string} to {string}")
20 public void driver_picked_user_from_to(String from, String to) {
21     System.out.println("Driver picked user from :" + from + " to " + to);
22 }
23
24@ Then("Driver ends the journey")
25 public void driver_ends_the_journey() {
26     System.out.println("Driver ends the journey");
27 }
28
29@ Then("User paid {int} rs")
30 public void user_paid_rs(Integer price) {
31     System.out.println("User paid amount : " + price + " rs");
32 }
33
34 }
```

■ Regular Expressions

If we need to use Regex in step definition we need to starts with ^ and ends with \$

The screenshot shows a Java IDE interface with three tabs at the top: 'UberBookingFeature.feature', 'BookUberCab.java', and 'BookUberCabTest.java'. The 'BookUberCab.java' tab is currently selected. Below it, the code is displayed:

```
14@ When("user booked a cab")
15 public void user_booked_a_cab() {
16     System.out.println("User Booked a cab");
17 }
18
19@ Then("^Driver picked user from \"([^\"]+)\" to \"([^\"]+)\\"$")
20 public void driver_picked_user_from_to(String from, String to) {
21     System.out.println("Driver picked user from :" + from + " to " + to);
22 }
23
24@ Then("Driver ends the journey")
25 public void driver_ends_the_journey() {
26     System.out.println("Driver ends the journey");
27 }
28
29@ Then("User paid {int} rs")
30 public void user_paid_rs(Integer price) {
31     System.out.println("User paid amount : " + price + " rs");
32 }
33
34}
35
```

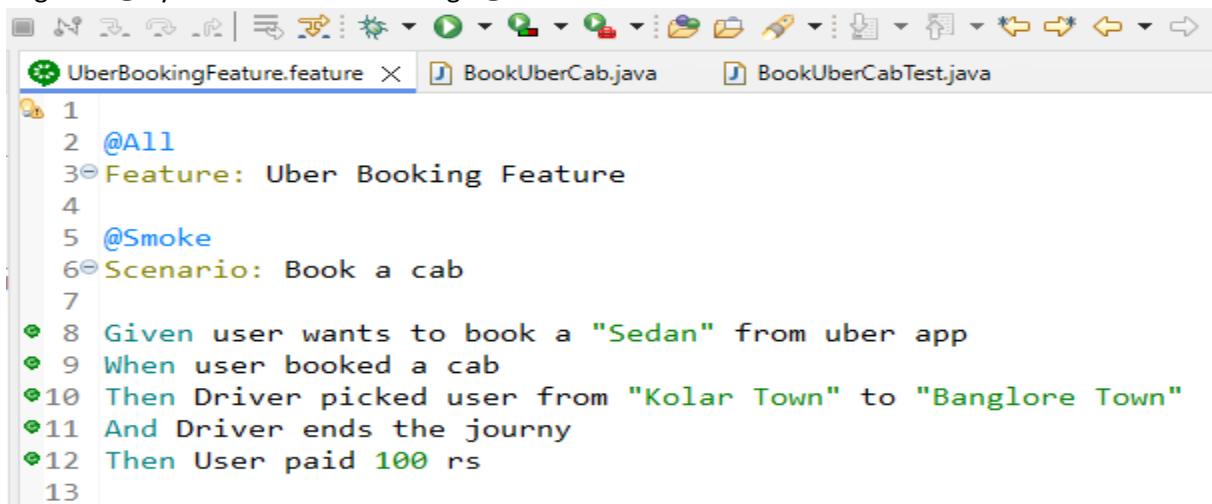
Below the code, the feature file content is shown:

```
# Sample Feature Definition Template
#
@Feature: Uber Booking Feature

@Scenario: Book a cab
#
| Given user wants to book a "Sedan" from uber app
| When user booked a cab
| Then Driver picked user from "Kolar Town" to "Banglore Town"
| And Driver ends the journey
| Then User paid 100 rs
|
```

TAGS IN CUCUMBER :

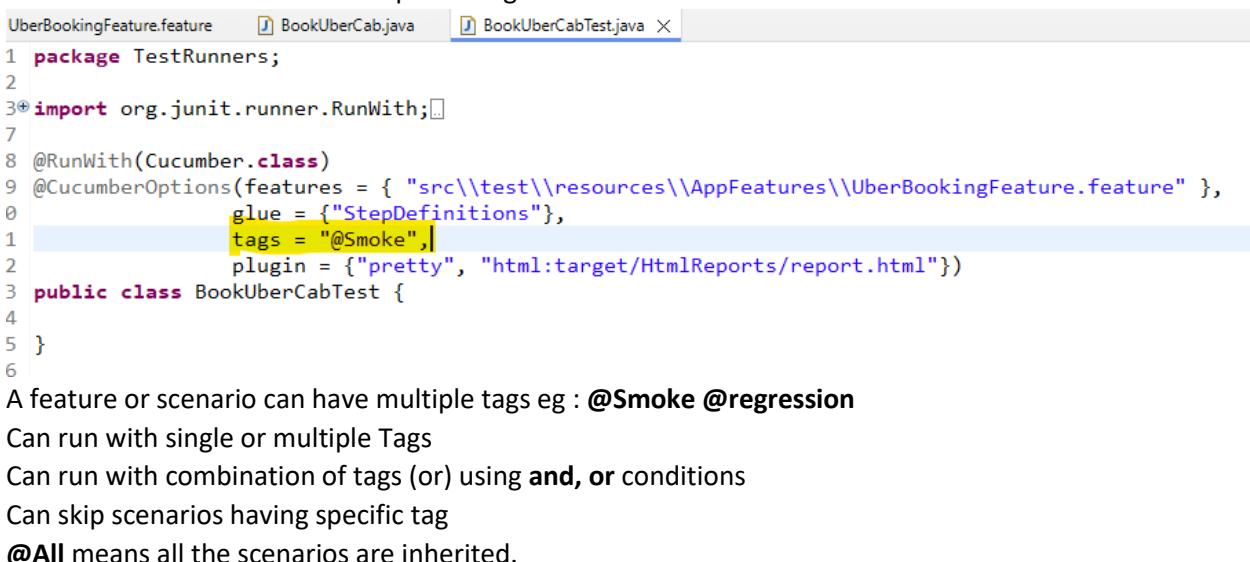
- Features & Scenarios can be marked with Tags
- Tags use @ symbol with some text e.g : @Smoke



The screenshot shows a Java IDE interface with three tabs at the top: 'UberBookingFeature.feature', 'BookUberCab.java', and 'BookUberCabTest.java'. The 'UberBookingFeature.feature' tab is active, displaying the following Cucumber code:

```
1
2 @All
3 Feature: Uber Booking Feature
4
5 @Smoke
6 Scenario: Book a cab
7
8 Given user wants to book a "Sedan" from uber app
9 When user booked a cab
10 Then Driver picked user from "Kolar Town" to "Banglore Town"
11 And Driver ends the journey
12 Then User paid 100 rs
13
```

- In Test Runner we can run with specific tags



The screenshot shows a Java IDE interface with three tabs: 'UberBookingFeature.feature', 'BookUberCab.java', and 'BookUberCabTest.java'. The 'BookUberCabTest.java' tab is active, showing the following Java code:

```
1 package TestRunners;
2
3 import org.junit.runner.RunWith;
4
5 @RunWith(Cucumber.class)
6 @CucumberOptions(features = { "src\\test\\resources\\AppFeatures\\UberBookingFeature.feature" },
7                 glue = {"StepDefinitions"},
8                 tags = "@Smoke",
9                 plugin = {"pretty", "html:target/HtmlReports/report.html"})
10 public class BookUberCabTest {
```

The 'tags = "@Smoke"' line is highlighted in yellow.

- A feature or scenario can have multiple tags eg : @Smoke @regression
- Can run with single or multiple Tags
- Can run with combination of tags (or) using **and**, **or** conditions
- Can skip scenarios having specific tag
- **@All** means all the scenarios are inherited.



The screenshot shows a Java IDE interface with three tabs: 'UberBookingFeature.feature', 'BookUberCab.java', and 'BookUberCabTest.java'. The 'BookUberCabTest.java' tab is active, showing the following Java code:

```
1 package TestRunners;
2
3 import org.junit.runner.RunWith;
4
5 @RunWith(Cucumber.class)
6 @CucumberOptions(features = { "src\\test\\resources\\AppFeatures\\UberBookingFeature.feature" },
7                 glue = {"StepDefinitions"},
8                 tags = "@Smoke or @Regression",
9                 plugin = {"pretty", "html:target/HtmlReports/report.html"})
10 public class BookUberCabTest {
```

The 'tags = "@Smoke or @Regression"' line is highlighted in yellow.

```
UberBookingFeature.feature X BookUberCab.java BookUberCabTest.java
1
2 @All
3@ Feature: Uber Booking Feature
4
5 @Smoke @Regression
6@ Scenario: Book a cab Sedan
7
8 Given user wants to book a "Sedan" from uber app
9 When user booked a cab
10 Then Driver picked user from "Kolar Town" to "Banglore Town"
11 And Driver ends the journey
12 Then User paid 100 rs
13
14 @Smoke @important
15@ Scenario: Book a cab SUV
16
17 Given user wants to book a "SUV" from uber app
18 When user booked a cab
19 Then Driver picked user from "Kolar Town" to "Banglore Town"
20 And Driver ends the journey
21 Then User paid 100 rs
```

```
UberBookingFeature.feature X BookUberCab.java BookUberCabTest.java X
1 package TestRunners;
2
3@ import org.junit.runner.RunWith;...
7
8 @RunWith(Cucumber.class)
9 @CucumberOptions(features = { "src\\test\\resources\\AppFeatures\\UberBookingFeature.feature" },
0         glue = {"StepDefinitions"}, ...
1         tags = "@Smoke and @Regression",
2         plugin = {"pretty", "html:target/HtmlReports/report.html"})
3 public class BookUberCabTest {
4
5 }
```

```
UberBookingFeature.feature X BookUberCab.java BookUberCabTest.java X
1 package TestRunners;
2
3@ import org.junit.runner.RunWith;...
7
8 @RunWith(Cucumber.class)
9 @CucumberOptions(features = { "src\\test\\resources\\AppFeatures\\UberBookingFeature.feature" },
10         glue = {"StepDefinitions"}, ...
11         tags = "@Smoke or @Regression) and not @Smoke",
12         plugin = {"pretty", "html:target/HtmlReports/report.html"})
13 public class BookUberCabTest {
14
15 }
```

```
UberBookingFeature.feature X BookUberCab.java BookUberCabTest.java X
1 package TestRunners;
2
3@ import org.junit.runner.RunWith;...
7
8 @RunWith(Cucumber.class)
9 @CucumberOptions(features = { "src\\test\\resources\\AppFeatures\\UberBookingFeature.feature" },
10         glue = {"StepDefinitions"}, ...
11         tags = "@All",
12         plugin = {"pretty", "html:target/HtmlReports/report.html"})
13 public class BookUberCabTest {
14
15 }
```

Background Keyword

- The Background keyword in Cucumber is used to define a set of steps that are common to all scenarios in a feature file.
- This helps avoid repetition by allowing you to specify these common steps once, and they will be executed before each scenario in the feature file.

```
*Orders.feature ×
1 Feature: Home Page
2   In order to check my order details
3   As a registered user
4   I want to specify the features of the order details page
5
6 Background:
7   Given a registered user exists
8   Then user is on amazon login page
9   When user enters username
10  And user enters password
11  And user clicks on login button
12  Then user navigates to order page
13  When user clicks on order link
14
15 Scenario: Check previous order details
16   Then user check previous order details
17
18 Scenario: Check open order details
19   Then user check open order details
20
21 Scenario: Check cancelled order details
22   Then user check cancelled order details
23

Scenario: Check open order details # src/test/resources/AppFeatures/Orders.feature:18
Given a registered user exists # StepDefinitions.OrderSteps.a_registered_user_exists()
Then user is on amazon login page # StepDefinitions.OrderSteps.user_is_on_amazon_login_page()
When user enters username # StepDefinitions.OrderSteps.user_enters_username()
And user enters password # StepDefinitions.OrderSteps.user_enters_password()
And user clicks on login button # StepDefinitions.OrderSteps.user_clicks_on_login_button()
Then user navigates to order page # StepDefinitions.OrderSteps.user_navigates_to_order_page()
When user clicks on order link # StepDefinitions.OrderSteps.user_clicks_on_order_link()
Then user check open order details # StepDefinitions.OrderSteps.user_check_open_order_details()

Scenario: Check cancelled order details # src/test/resources/AppFeatures/Orders.feature:21
Given a registered user exists # StepDefinitions.OrderSteps.a_registered_user_exists()
Then user is on amazon login page # StepDefinitions.OrderSteps.user_is_on_amazon_login_page()
When user enters username # StepDefinitions.OrderSteps.user_enters_username()
And user enters password # StepDefinitions.OrderSteps.user_enters_password()
And user clicks on login button # StepDefinitions.OrderSteps.user_clicks_on_login_button()
Then user navigates to order page # StepDefinitions.OrderSteps.user_navigates_to_order_page()
When user clicks on order link # StepDefinitions.OrderSteps.user_clicks_on_order_link()
Then user check cancelled order details # StepDefinitions.OrderSteps.user_check_cancelled_order_details()

3 Scenarios (3 passed)
24 Steps (24 passed)
```

The screenshot shows a code editor with two tabs: 'Orders.feature' and '*OrderSteps.java'. The '*OrderSteps.java' tab is active, displaying the following Java code:

```
9@ Given("a registered user exists")
10 public void a_registered_user_exists() {
11 }
12@ Then("user is on amazon login page")
13 public void user_is_on_amazon_login_page() {
14 }
15@ When("user enters username")
16 public void user_enters_username() {
17 }
18@ When("user enters password")
19 public void user_enters_password() {
20 }
21@ When("user clicks on login button")
22 public void user_clicks_on_login_button() {
23 }
24@ Then("user navigates to order page")
25 public void user_navigates_to_order_page() {
26 }
27@ When("user clicks on order link")
28 public void user_clicks_on_order_link() {
29 }
30@ Then("user check previous order details")
31 public void user_check_previous_order_details() {
32 }
33@ Then("user check open order details")
34 public void user_check_open_order_details() {
35 }
36@ Then("user check cancelled order details")
37 public void user_check_cancelled_order_details() {
38 }
```

HOOKS

- Blocks of code that runs before or after each scenario.
- Hooks in cucumber are like listeners in TestNg
- Can define Hook using **@Before @After**

Scenario Hooks

- Runs Before and after each scenario

```
@Before
public void SetUp() {
    System.out.println("Browser Launched");
}
@After
public void tearDown() {
    System.out.println("Close Browser");
}
```

Provide hook package path in glue

The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer:** Shows the project structure under "CucumberBDD". It includes "src/main/java" (empty), "src/test/java" (containing "MyHooks" and "TestRunners" packages), "src/test/resources" (containing "AppFeatures" folder with "Orders.feature", "Search.feature", and "UberBookingFeature.feature"), and "src/test/resources" (empty). It also lists "JRE System Library [JavaSE-1.8]", "Maven Dependencies", "src", "target", and "pom.xml".
- JUnit View:** Shows the current file is "Search.feature". Below it is the code for "AmazonSearchTest.java":

```
1 package TestRunners;
2
3 import org.junit.runner.RunWith;
4
5 @RunWith(Cucumber.class)
6 @CucumberOptions(features = { "src\test\resources\AppFeatures\Search.feature" },
7                     glue = { "StepDefinitions", "MyHooks" }, // stepdefinition package p
8                     plugin = {"pretty", "html:target/HtmlReports/report.html"})
9
10 // The Pretty plugin provides a verbose output,
11 // making it easier to understand what's happening during test execu
12
13 public class AmazonSearchTest {
14
15 }
16
17
```

- Console View:** Displays the terminal output of the test execution:

```
<terminated> AmazonSearchTest (1) [JUnit] C:\Users\Chethank2\p2\pool\plugins\org.eclipse.jdt.core\openjdk.hotspot.jre.full.win32.x86_64_17.0.13.v20241
Scenario: Validate Search feature on amazon
# src/test/resources/AppFe
Browser Launched
Search Option is present on amazon page
Given I have Search option in the amazon
Searched for a product Samsung M51 and price 25000
When I search for a product "Samsung M51" and Price is 25000
Results for search product is displayed
Then Results for Samsung M51 should display
Close Browser
```

Step Hooks

- Runs Before and after each step

```
public class Hooks {

    @Before
    public void setUp() {
        System.out.println("Browser Launched");
    }

    @After
    public void tearDown() {
        System.out.println("Close Browser");
    }

    @BeforeStep
    public void refreshPage() {
        System.out.println("refresh Page");
    }

    @AfterStep
    public void takeScreenShot() {
        System.out.println("Capture Screenshot");
    }
}
```

Execution with Order

```
7
8 public class Hooks {
9
10    @Before(order = 1)
11    public void launchBrowser() {
12        System.out.println("Browser Launched");
13    }
14
15    @Before(order = 2)
16    public void launchUrl() {
17        System.out.println("url Launched");
18    }
19
20    @After(order = 1)
21    public void logout() {
22        System.out.println("user logout");
23    }
24
25    @After(order = 2)
26    public void tearDown() {
27        System.out.println("Close Browser");
28    }
}
```

Conditional Hooks

- Hooks associated with tags for conditional execution

Tags can be used with @Before, @After, @BeforeStep, @AfterStep

```
public class Hooks {
    @Before(value="@Smoke or @Regression")
    public void launchBrowser() {
        System.out.println("Browser Launched");
    }

    @After("@Smoke")
    public void logout() {
        System.out.println("user logout");
    }

    @After(value="@Smoke", order = 1)
    public void closeBrowser() {
        System.out.println("close browser");
    }
}
```

Generate Different Reports With Cucumber

The screenshot shows the Eclipse IDE interface. On the left is the Project Explorer view, which lists the project structure. It includes a 'src' folder containing 'main/java' (with 'CucumberBDD' and 'MyHooks' packages), 'test/java' (with 'StepDefinitions' package containing 'AmazonSearch.java', 'BookUberCab.java', 'OrderSteps.java', and 'TestRunners' package containing 'AmazonSearchTest.java', 'BookUberCabTest.java', 'CucumberReportsTest.java', and 'OrdersTest.java'), and 'test/resources' (with 'AppFeatures' folder containing 'Orders.feature', 'Search.feature', and 'UberBookingFeature.feature'). It also shows 'JRE System Library [JavaSE-1.8]' and 'Maven Dependencies'. On the right is the Java editor view, showing the code for 'CucumberReportsTest.java'. The code imports org.junit.runner.RunWith and org.junit.runners.Parameterized. It uses @RunWith(Cucumber.class) and @CucumberOptions to run 'Search.feature'. The 'glue' parameter specifies the step definition package path as 'StepDefinitions' and 'MyHooks'. The 'tags' parameter is set to '@Smoke or @Regression'. The 'plugin' parameter is set to 'pretty', 'html:target/HtmlReports/report.html', 'junit:target/JUnitReports/report.xml', and 'json:target/JsonReports/report.json'. The class definition for 'CucumberReportsTest' is shown. Below the editor is the 'Console' view, which shows the output of running the test class 'AmazonSearchTest'. The output indicates the test was terminated successfully.

```
1 package TestRunners;
2
3 import org.junit.runner.RunWith;
4
5 import cucumber.api.junit.Cucumber;
6 import cucumber.api.junit.CucumberOptions;
7
8 @RunWith(Cucumber.class)
9 @CucumberOptions(features = { "src\\test\\resources\\AppFeatures\\Search.feature" }, // provide feature
10                 glue = { "StepDefinitions", "MyHooks" }, // stepdefinition package path
11                 tags = "@Smoke or @Regression",
12                 plugin = {"pretty", "html:target/HtmlReports/report.html",
13                            "junit:target/JUnitReports/report.xml",
14                            "json:target/JsonReports/report.json"})
15
16 public class CucumberReportsTest {
17
18 }
```

Data Table (asLists):

The screenshot shows the Eclipse IDE interface. On the left is the Project Explorer view, which lists the project structure. It includes a 'src' folder containing 'main/java' (with 'UserRegistration.java') and 'test/java' (with 'UserRegistration.feature'). On the right is the Java editor view, showing the code for 'UserRegistration.java'. The code defines a feature 'User Registration' with a scenario 'user registration with diffrent data'. The scenario has given, when, and then steps. The 'when' step uses a data table with two rows of user details. The 'then' step asserts successful registration. Below the editor is the Java code for the 'user_enters_following_user_details' method. It uses the 'DataTable.asLists' method to convert the data table into a list of lists of strings. It then iterates over the list and prints each row to the console using System.out.println.

```
1 Feature: User Registration
2
3 Scenario: user registration with diffrent data
4   Given User is on registration page
5   When User enters following user details
6     | chethan | chnr@gmail.com | 7654457 | Kolar |
7     | Ramesh | rash@gmail.com | 7886533 | Malur |
8   Then user registration should be successful
9
10  @When("User enters following user details")
11  public void user_enters_following_user_details(DataTable dataTable) {
12
13      List<List<String>> userDetails = dataTable.asLists(String.class);
14
15      for(List<String> list : userDetails) {
16          System.out.println(list);
17      }
18  }
```

Data Table (asMaps):

Scenario: user registration with diffrent data with columns

Given User is on registration page

When User enters following user details with columns

user	gmail	number	city
chethan	chnr@gmail.com	7654457	Kolar
Ramesh	rash@gmail.com	7886533	Malur

Then user registration should be successful

```
@When("User enters following user details with columns")
public void user_enters_following_user_details_with_columns(DataTable dataTable) {

    List<Map<String, String>> userDetails = dataTable.asMaps(String.class, String.class);
    userDetails.get(0).get("user");
    userDetails.get(1).get("user");

    for (Map<String, String> e : userDetails) {
        System.out.println(e.get("user"));
        System.out.println(e.get("gmail"));
        System.out.println(e.get("number"));
        System.out.println(e.get("city"));
    }
}

chethan
chnr@gmail.com
7654457
Kolar
Ramesh
rash@gmail.com
7886533
Malur
When User enters following user details with columns      # StepDefinitions.UserRegistration.user_enters_following_
| user | gmail | number | city |
| chethan | chnr@gmail.com | 7654457 | Kolar |
| Ramesh | rash@gmail.com | 7886533 | Malur |
```

Scenario Outline with Examples Keyword (Data Driven Testing in cucumber)

The screenshot shows a code editor with two files open:

- login.feature**: A Cucumber feature file containing a scenario outline and examples.
- login.java**: A Java step definition file corresponding to the feature file.

login.feature content:

```
1 Feature: login feature
2
3 Scenario Outline: login Test
4   Given user is on login page
5   When Enter the user name "<username>"
6   And Enter password "<password>"
7   Then click on login button
8
9 Examples:
10   | username | password |
11   | chethan  | 5757   |
12   | ramesh   | 7847   |
13
```

login.java content:

```
1 package com.cucumber.stepdefinition;
2
3 import io.cucumber.java.en.Given;
4 import io.cucumber.java.en.Then;
5 import io.cucumber.java.en.When;
6
7 public class login {
8     @Given("user is on login page")
9     public void user_is_on_login_page() {
10
11     }
12
13     @When("Enter the user name \"<username>\"")
14     public void enter_the_user_name(String username) {
15
16     }
17
18     @And("Enter password \"<password>\"")
19     public void enter_password(String password) {
20
21     }
22
23     @Then("click on login button")
24     public void click_on_login_button() {
25
26     }
27 }
```

```

public class login {

    @Given("user is on login page")
    public void user_is_on_login_page() {
        System.out.println("user is on login page");
    }

    @When("Enter the user name {string}")
    public void enter_the_user_name(String userName) {
        System.out.println("User enters the username is " + userName);
    }

    @When("Enter password {string}")
    public void enter_password(String password) {
        System.out.println("User enters the password is " + password);
    }

    @Then("click on login button")
    public void click_on_login_button() {
        System.out.println("Clicked on login button");
    }
}

```



The screenshot shows the Eclipse IDE interface with the 'Console' tab selected. The console output displays the results of running a JUnit test named 'LoginStep'. It lists two scenarios: one for 'user is on login page' and one for 'user is on login page'. Each scenario has its steps and corresponding Java code snippets.

```

Console X Problems Debug Shell Results of running class TC_994451Test
<terminated> LoginStep [JUnit] C:\Users\ChethanK2\p2\pool\plugins\org.eclipse.jdt.openjdk.hotspot.jre.full.win32.x86_64_17.0.13.v20241023-1126\jre\bin\javaw.exe (Jan 26, 2025, 12:07:33 AM)

Scenario Outline: login Test      # src/test/resources/AppFeatures/login.feature:11
user is on login page
  Given user is on login page    # StepDefinitions.login.user_is_on_login_page()
  Use enters the username is chethan
    When Enter the user name "chethan" # StepDefinitions.login.enter_the_user_name(java.lang.String)
  Use enters the password is 5757
    And Enter password "5757"       # StepDefinitions.login.enter_password(java.lang.String)
  Clicked on login button
    Then click on login button     # StepDefinitions.login.click_on_login_button()

Scenario Outline: login Test      # src/test/resources/AppFeatures/login.feature:12
user is on login page
  Given user is on login page    # StepDefinitions.login.user_is_on_login_page()
  Use enters the username is ramesh
    When Enter the user name "ramesh" # StepDefinitions.login.enter_the_user_name(java.lang.String)
  Use enters the password is 7847
    And Enter password "7847"       # StepDefinitions.login.enter_password(java.lang.String)
  Clicked on login button
    Then click on login button     # StepDefinitions.login.click_on_login_button()

```

Hybrid Framework with Page Object Model (POM) with Cucumber BDD & Selenium (Initial Design).

Designing different components in this Framework:

- 1. Feature Files**
- 2. Step Definition Classes**
- 3. Configuration Files**
- 4. Cucumber Hooks with before and after**
- 5. Element Utilities/Libraries/Generic Functions**
- 6. Cucumber 6 Extent Report Adaptor for Spark HTML / PDF Reports**
- 7. Test Runners in JUnit**
- 8. Page Classes for POM**
- 9. Maven with pom.xml with different dependencies and plugins**
- 10. Parallel Execution**
- 11. Cucumber 6 Web HTML Reports**
- 12. Screenshot for Failure scenarios**
- 13. Integration with GIT Repo**
- 14. Running test cases from Jenkins**
- 15. Running test cases on Dockerized Selenium GRID**

And much more.....

Technologies Used:

- 1. Selenium WebDriver with Java Language binding**
- 2. Cucumber 6.x JVM library**
- 3. WebDriverManager**
- 4. JDK 1.8**
- 5. Maven (Build tool)**
- 6. Maven Plugins**
- 7. Cucumber extent report 6 adapter**
- 8. JUnit 4.x library**
- 9. Log4j**

10. GIT HUB - Git Repo

11. Docker

12. Jenkins

13. Eclipse (IDE)

