**BDD**

Behavior-driven development (BDD) is a software testing methodology that uses plain language to describe how a software application should behave..

**Cucumber** is one of the framework which support BDD...

**Why is cucumber?**

**Cucumber** a testing framework which support **BDD(Behaviour driven development)**. Using cucumber we can describe the behaviour of the application using **plain English in a file called feature file** & **the grammar that we use in that feature file is called as Gherkin.** Even though cucumber is written in ruby, we can use it with java,C# and python

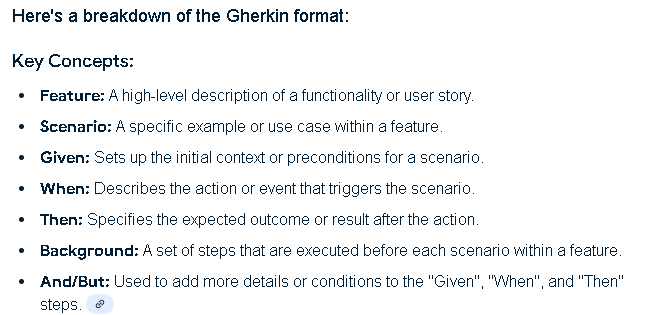
Eg..Non technical guys & client don’t understand the Java,python codes

With the help of cucumber feature file concepts, they can easily understand the code, what code is doing

We are giving the feature file code, client can execute & validate the product easly..**mainly used for client to understand their product code/structure.**

**Feature, scenario, Given,when, Then are called as Gherkin language keyword**

**A high level desp of a functionality or user story(refer below) ->called as Plain English**

****

**Create a simple maven project**

& add the Cucumber Java/ Cucumber Junit/Selenium Java/Junit jar file in maven Porn XMl

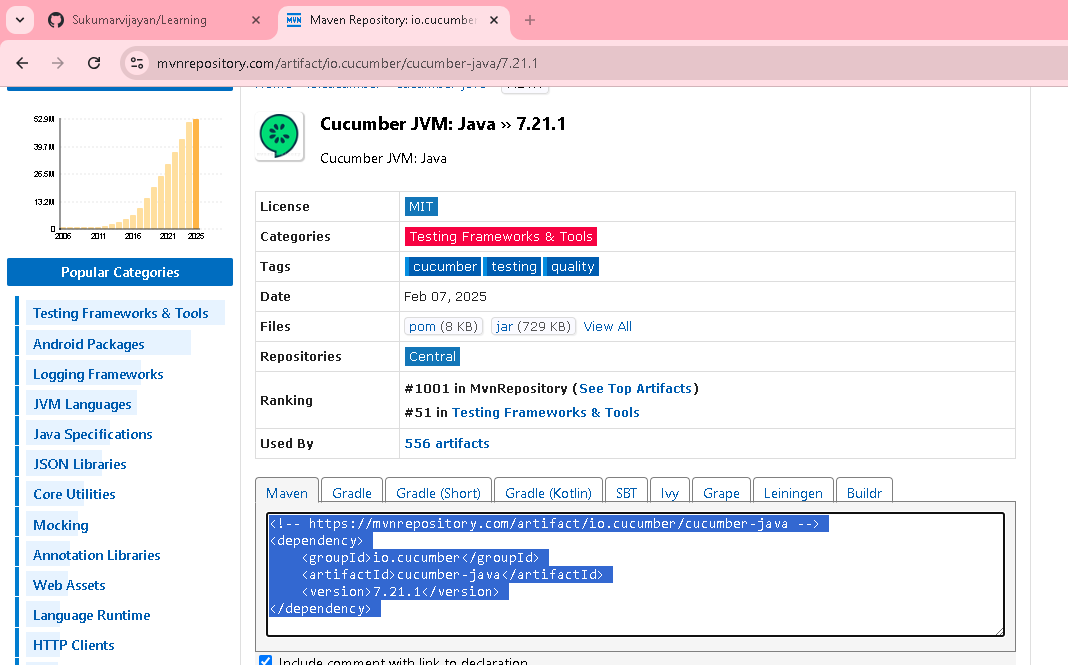
**Cucumber project Structure**

1. **Feature file->**combination of (Plain English + Gherkin) saved in (.features)
2. **Step definition ->**here only we will write actual code...

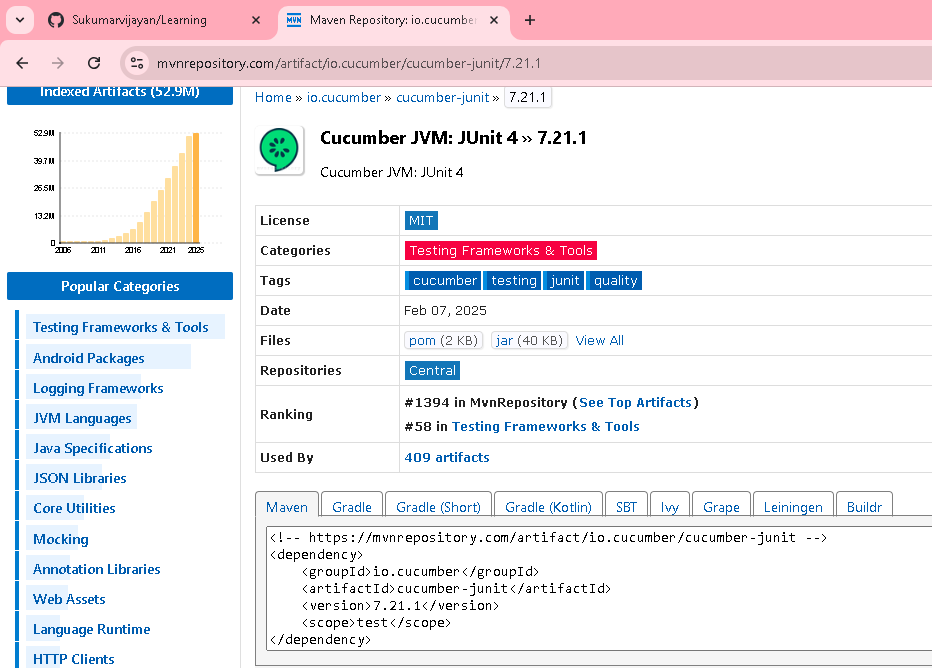
note plain English & Gherkin will not perform open google & other operation..just a plain text to End user/client for easy understanding..java code only will do all web browser operation in back layer..Feature file like a front layer

1. **Runner class** ->it will link the feature file & step definition**..**In feature file , particular file encounter , then specific step definition code execute..

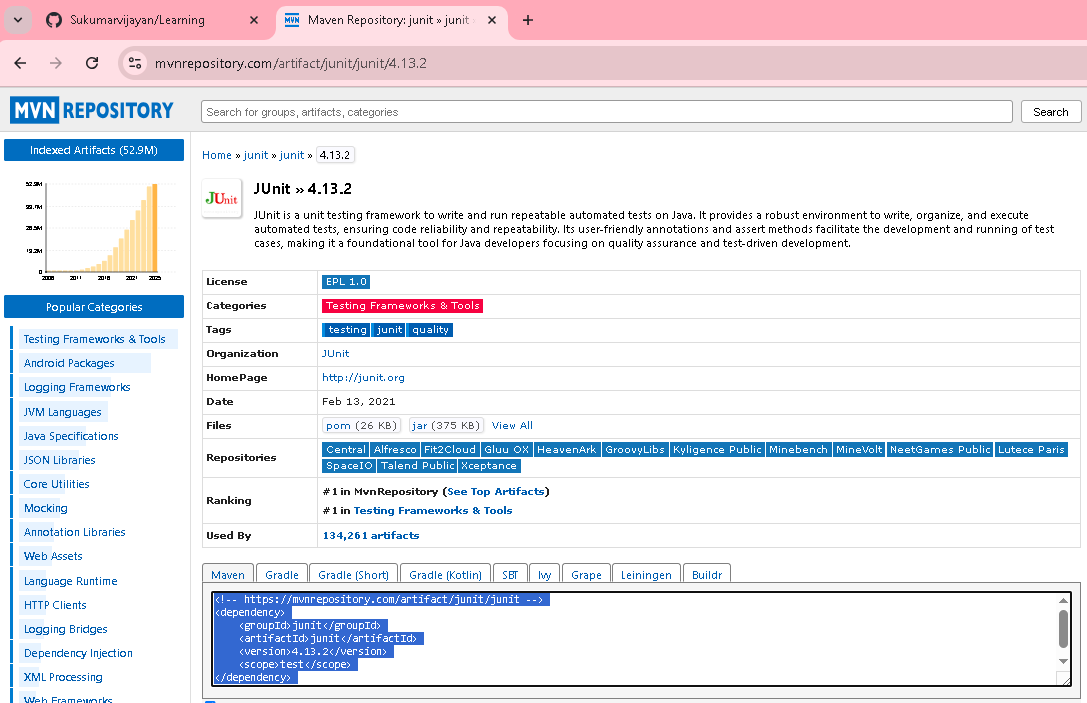
**Cucumber Java**

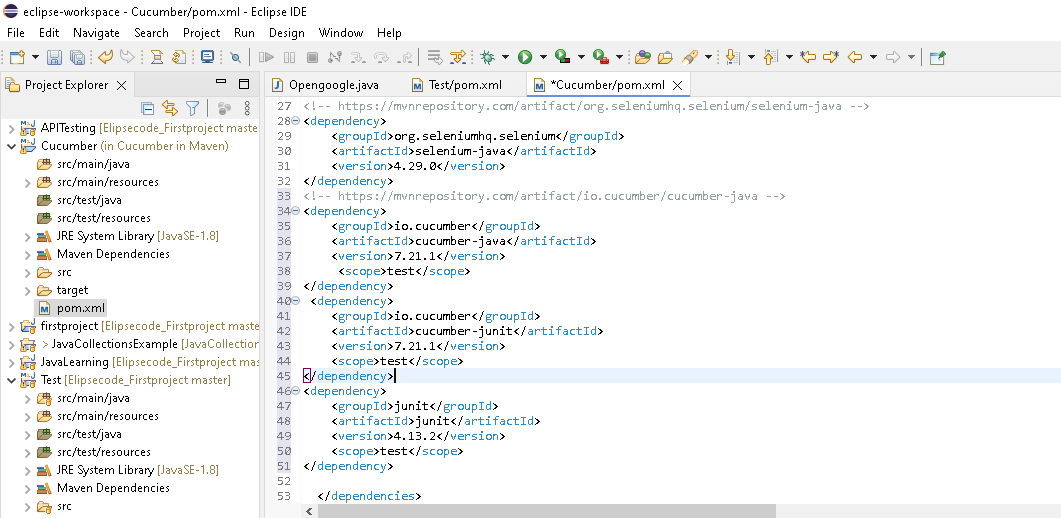
****

**Cucumber Junit**

****

**Junit**

****

****

**After updating dependency in porn.xml..save & refresh**

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.lao</groupId>

<artifactId>Cucumber</artifactId>

<version>0.0.1-SNAPSHOT</version>

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.5</version>

<configuration>

<source>1.8</source>

<target>1.8</target>

</configuration>

</plugin>

</plugins>

</build>

<dependencies>

<!-- https://mvnrepository.com/artifact/com.aventstack/extentreports -->

<!-- https://mvnrepository.com/artifact/org.seleniumhq.selenium/selenium-java -->

<dependency>

<groupId>org.seleniumhq.selenium</groupId>

<artifactId>selenium-java</artifactId>

<version>4.29.0</version>

</dependency>

<!-- https://mvnrepository.com/artifact/io.cucumber/cucumber-java -->

<dependency>

<groupId>io.cucumber</groupId>

<artifactId>cucumber-java</artifactId>

<version>7.21.1</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>io.cucumber</groupId>

<artifactId>cucumber-junit</artifactId>

<version>7.21.1</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

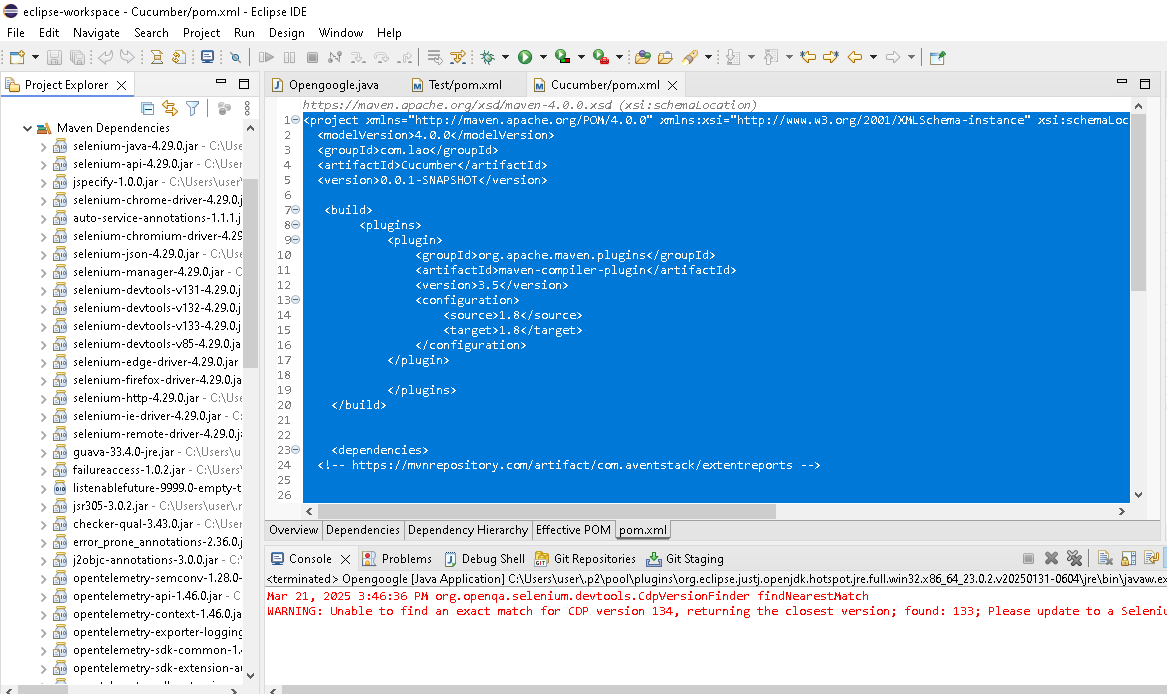
<version>4.13.2</version>

<scope>test</scope>

</dependency>

</dependencies>

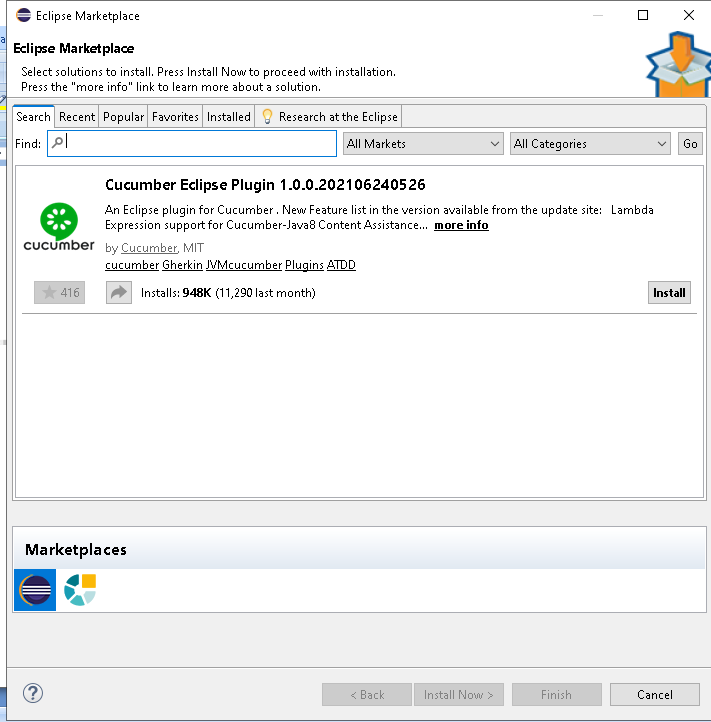
</project>

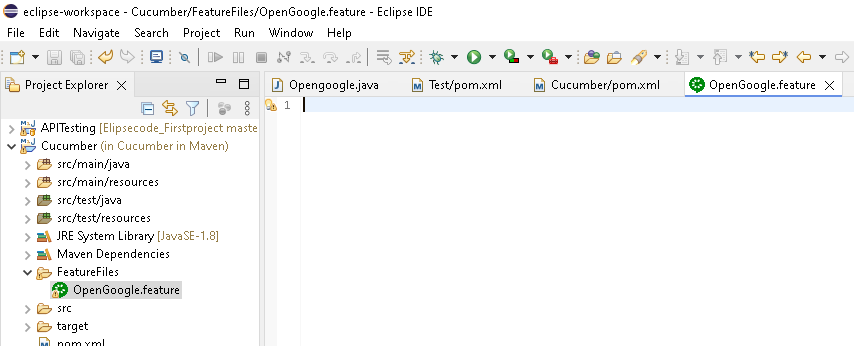


**Let us create our first feature file called “OpenGoogle.feature”**

**1.Create folder(feature) under cucumber project..in that create one feature file named OpenGoogle.feature**

Then install below extension in elipse marketplace for cucumber plugin(Feature, scenario, given, when, then all in colors)

****

****

Feature: This is to test google serach

Scenario: Google search scenario

Given user is enentering google.co.in

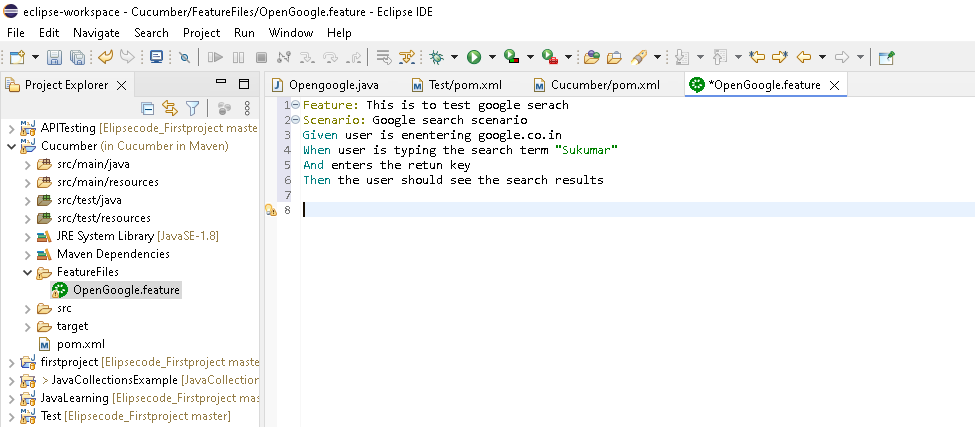
When user is typing the search term "Sukumar" [Here Sukumar given in “”, so its go as a parameter/test Date to that respective function]

And enters the retun key

Then the user should see the search results

Multiple scenario can be written for single feature file

Eg..for login feature file..we can write positive scenario & negative scenario



**2. Then Create Runner class..**Runner class will link the Feature file & step definition class

Under src/test/java ->create package & class (Runner class)

**package** runner;

**import** org.junit.runner.RunWith;

**import** io.cucumber.junit.Cucumber;

**import** io.cucumber.junit.CucumberOptions;

//it will run as cucumber

@RunWith(Cucumber.**class**)

//while running as cucumber, it will link feature file location & step definition [package]

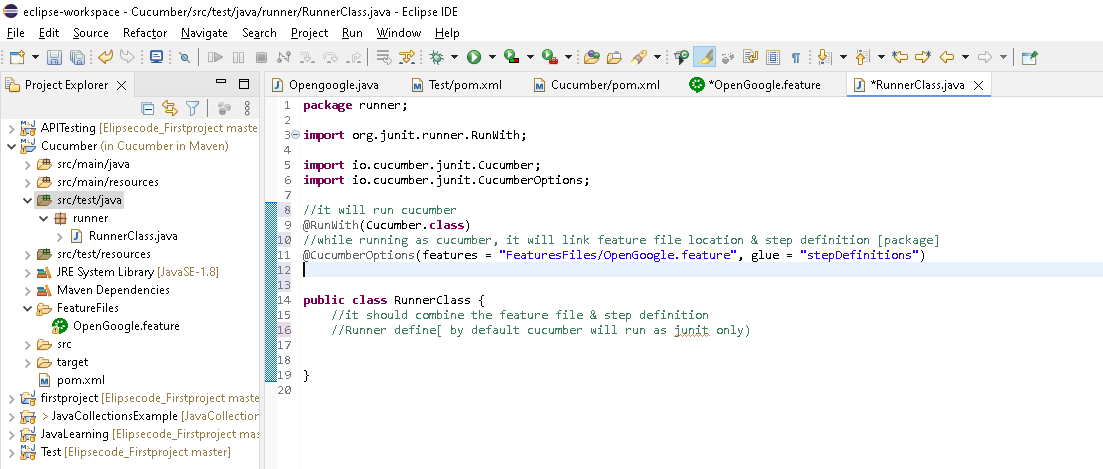
@CucumberOptions(features = "FeatureFiles/OpenGoogle.feature", glue = "stepDefinitions")

**public** **class** RunnerClass {

//it should combine the feature file & step definition

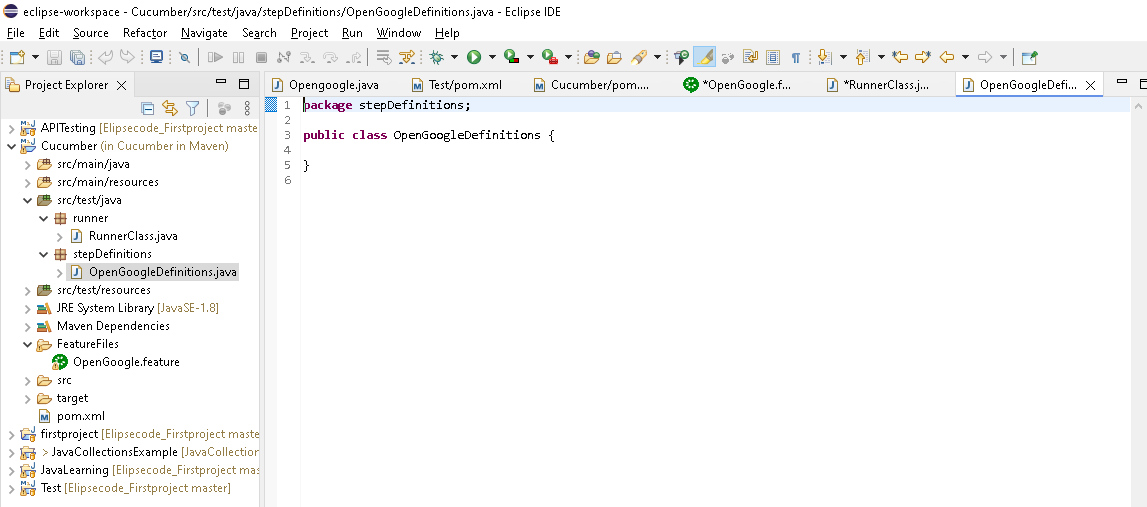
//Runner define[ by default cucumber will run as junit only)

}



3. **Then Create Step definition class..**

Under src/test/java ->create package & class (stepDefinations)



1. Next go to Runnerclass

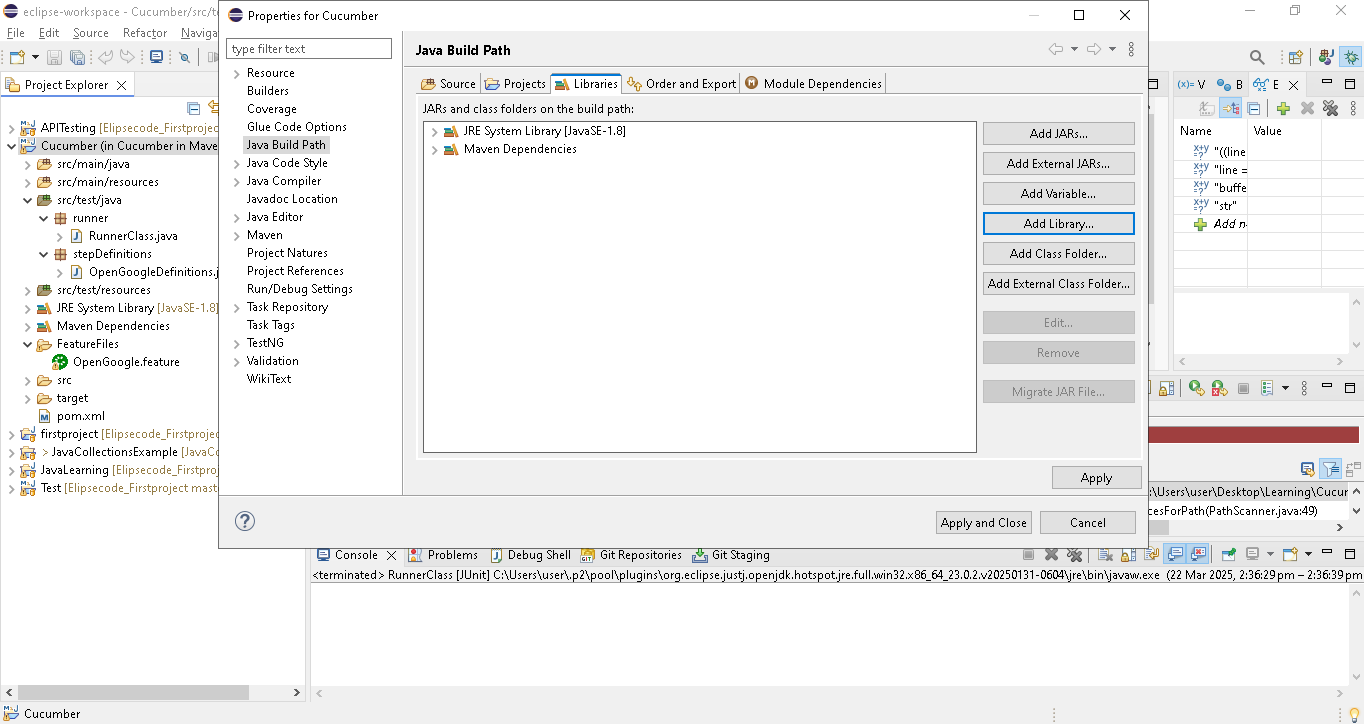
Right class run as ->Junit test...it will automatically create functions for each steps in feature file in console

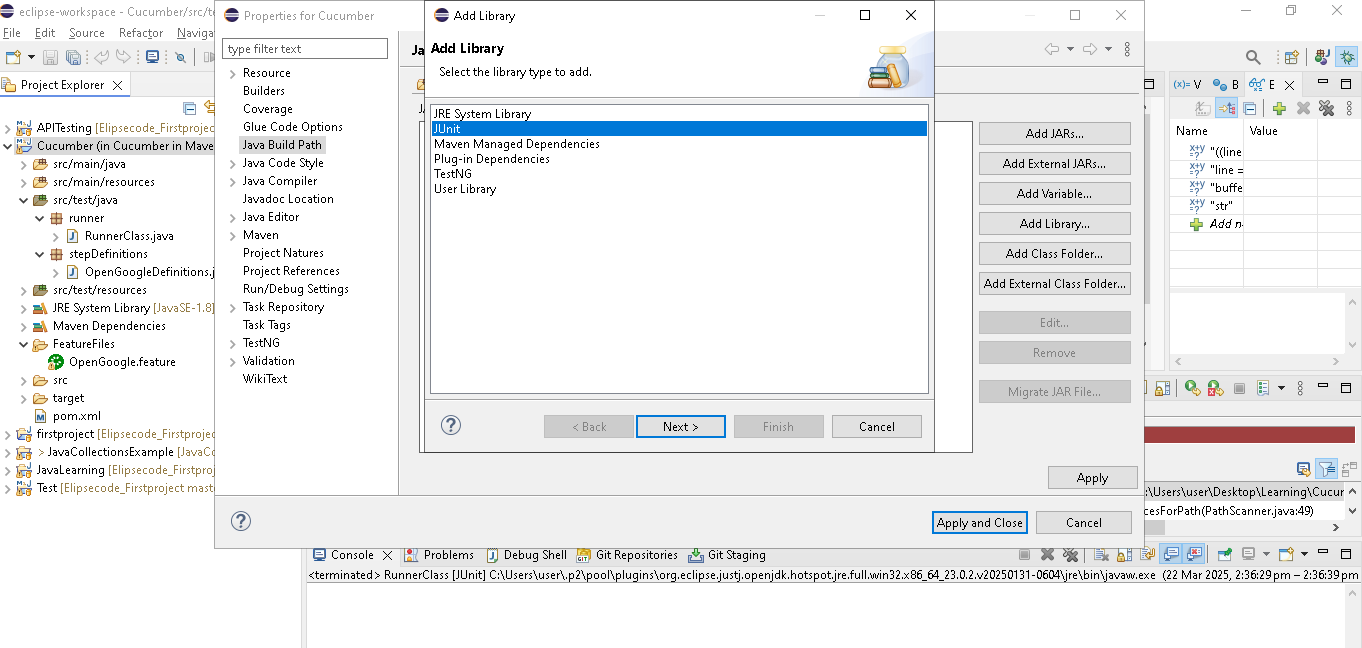
Copy the code & paste in stepdefinition class

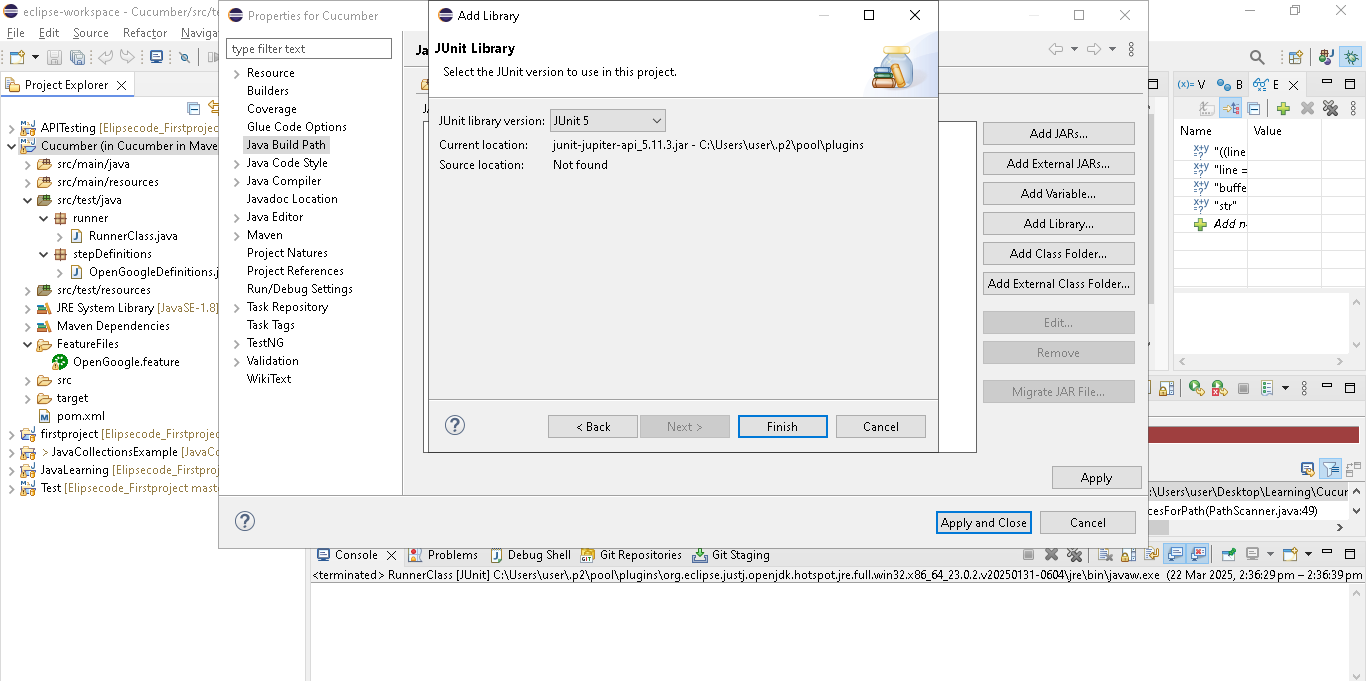
Write the code for each feature file steps

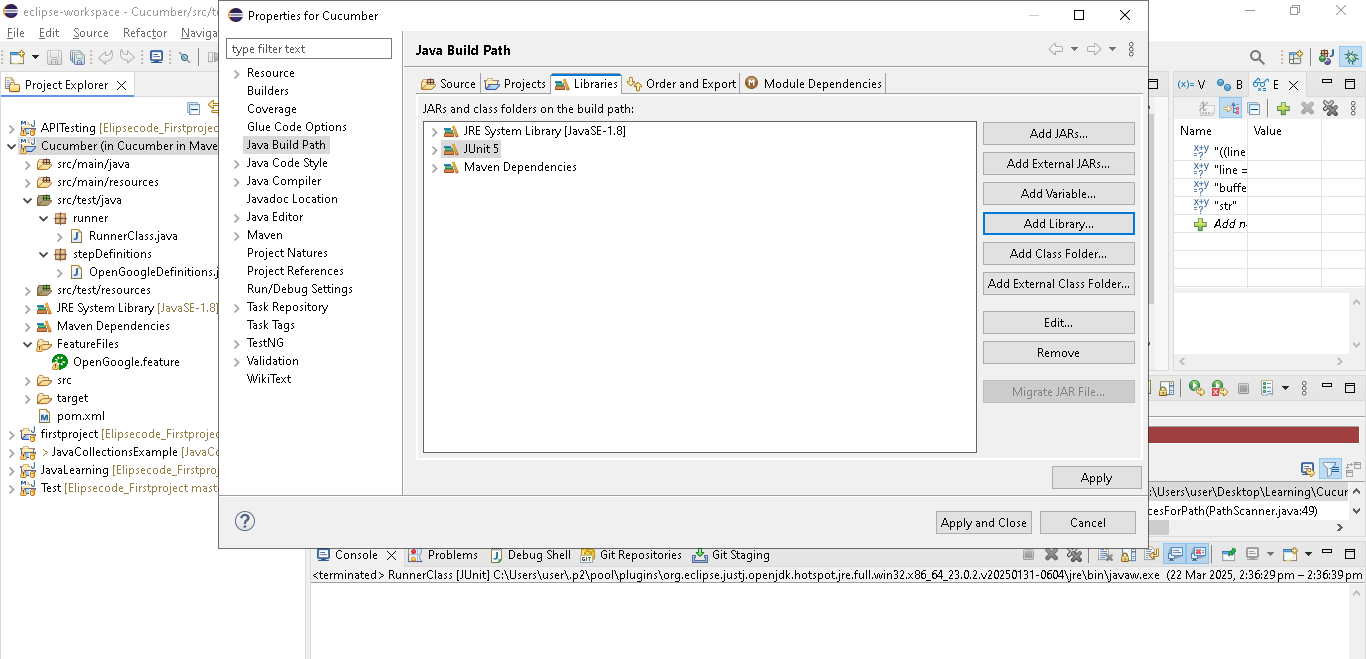
And then execute from runnerclass again

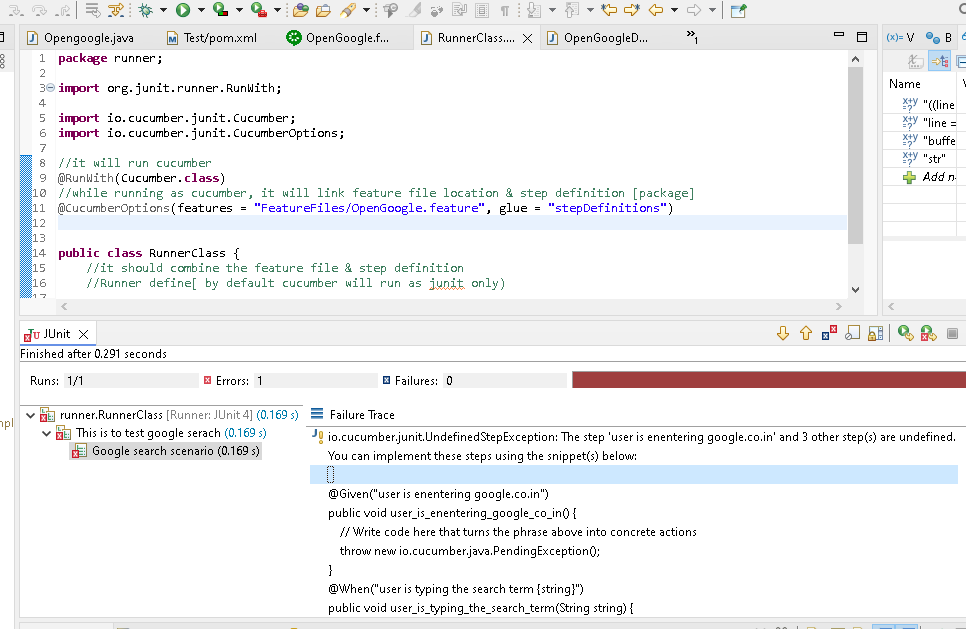
Need to add Junit lib in built path

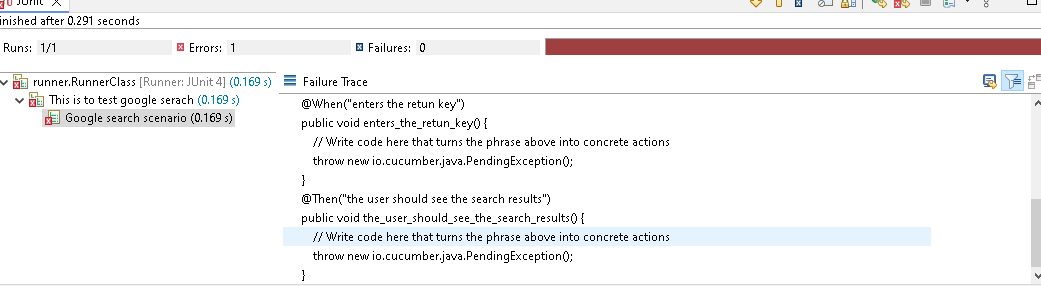




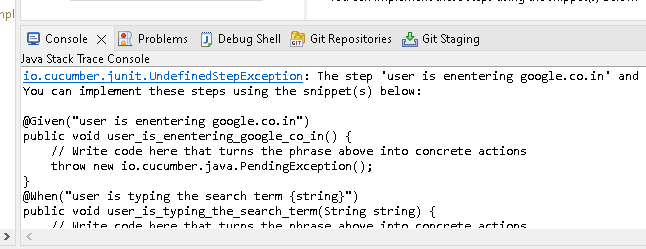


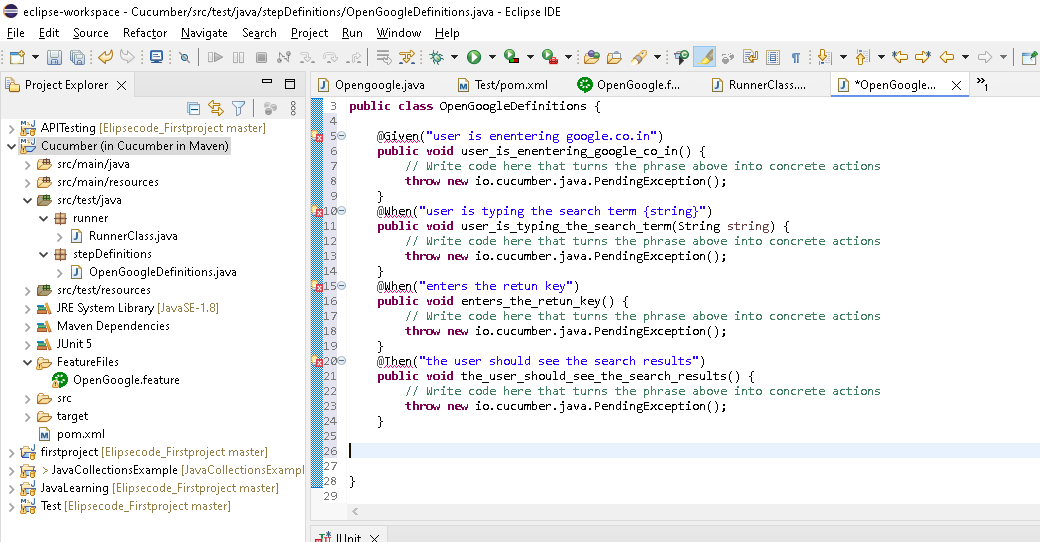


****

****

**Move the code under failure trace to console[icon present at right corner] , then copy & paste in stepdefinition class**

****

****

**Remove the throw lion & write code for google search**

**package** stepDefinitions;

**import** org.openqa.selenium.By;

**import** org.openqa.selenium.Keys;

**import** org.openqa.selenium.WebDriver;

**import** org.openqa.selenium.chrome.ChromeDriver;

**import** io.cucumber.java.en.Given;

**import** io.cucumber.java.en.Then;

**import** io.cucumber.java.en.When;

**public** **class** OpenGoogleDefinitions {

WebDriver driver;

@Given("user is enentering google.co.in")

**public** **void** user\_is\_enentering\_google\_co\_in() {

// Write code here that turns the phrase above into concrete actions

System.*setProperty*(

"webdriver.chrome.driver",

"C:\\Users\\user\\Downloads\\chromedriver.exe");

// Instantiate a ChromeDriver class.

driver = **new** ChromeDriver();

//C:\Users\\user\\Downloads\\chromedriver-win64\\chromedriver-win64

// Maximize the browser

driver.manage().window().maximize();

// Launch Website

driver.get("https://www.google.com/");

}

@When("user is typing the search term {string}")

**public** **void** user\_is\_typing\_the\_search\_term(String string) **throws** InterruptedException {

// Write code here that turns the phrase above into concrete actions

driver.findElement(By.*name*("q")).sendKeys(string);

//wait(50000);

}

@When("enters the retun key")

**public** **void** enters\_the\_retun\_key() {

// Write code here that turns the phrase above into concrete actions

driver.findElement(By.*name*("q")).sendKeys(Keys.***RETURN***);

}

@Then("the user should see the search results")

**public** **void** the\_user\_should\_see\_the\_search\_results() {

// Write code here that turns the phrase above into concrete actions

System.***out***.println("Flow Success");

/\*

\* boolean str1 = driver.findElement(By.partialLinkText("Java")).isDisplayed();

\* if (str1) { System.out.println("Pass"); } else { System.out.println("Fail");

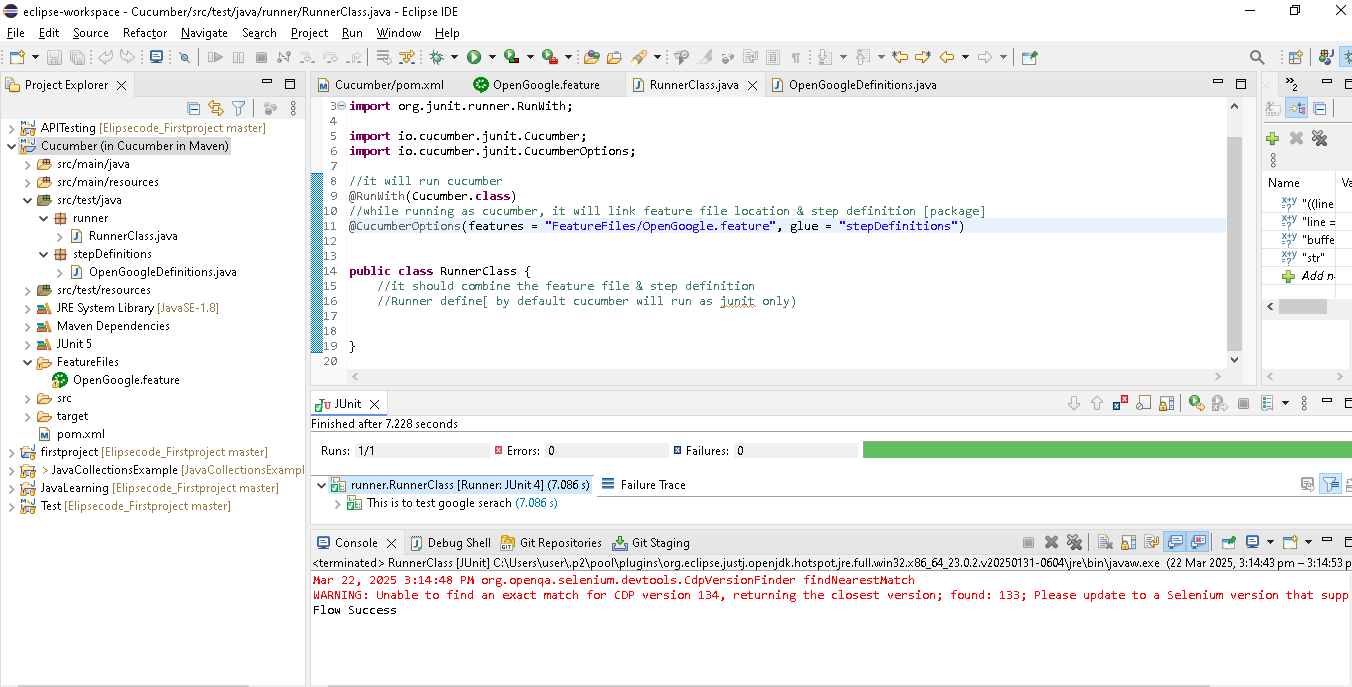
\* }

\*/

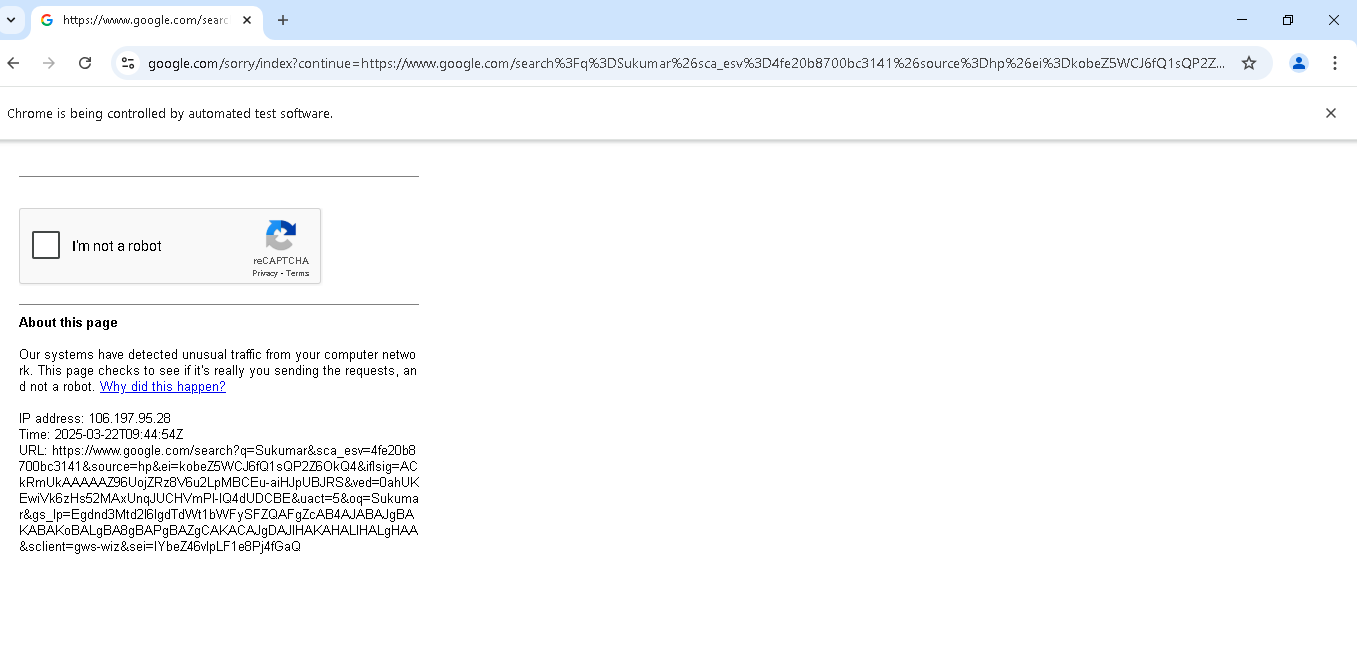
}

}

**Go to runner class & run as junit classs**

****

**Browser also launched & search with sukumar name..After that robot captcha screen coming**

****

**Create a test case for login screen**

**Step1: Create a login feature file & write the plain lang**

Feature: This is the login feature for orange HRM

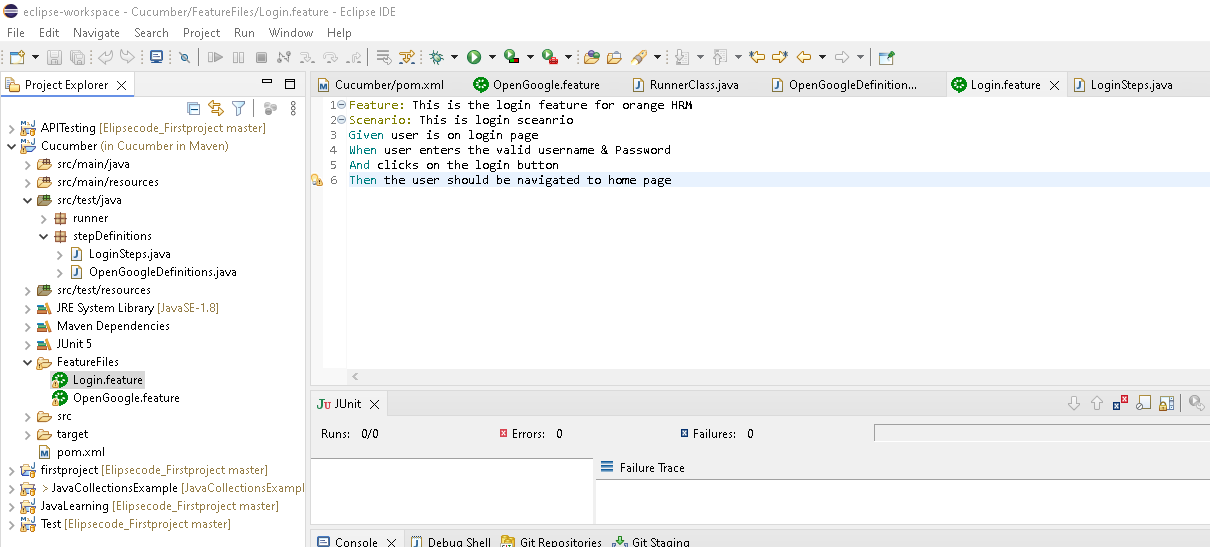
Scenario: This is login sceanrio

Given user is on login page

When user enters the valid username & Password

And clicks on the login button

Then the user should be navigated to home page

****

**Step2: Create a class (LoginStep) under step Definitions package**

**Step3: go to runner class,write below code for mapping feature file[Login.features] & step Definitions[Loginsteps class] & run as junit.test ->it will create a methods..copy the methods**

**[Instead of generating automatically, we can write the own methods but description(**user is on login page **)present in the feature file same should present here** @Given("user is on login page"), method can be different or same with “\_” included]

Given user is on login page

@Given("user is on login page")

**public** **void** user\_is\_on\_login\_page() {

// Write code here that turns the phrase above into concrete actions

}

@CucumberOptions(features = "FeatureFiles/Login.feature", glue = "stepDefinitions")

Note:

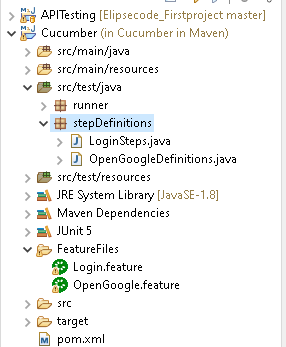
@CucumberOptions(features = "FeatureFiles/", glue = "stepDefinitions")

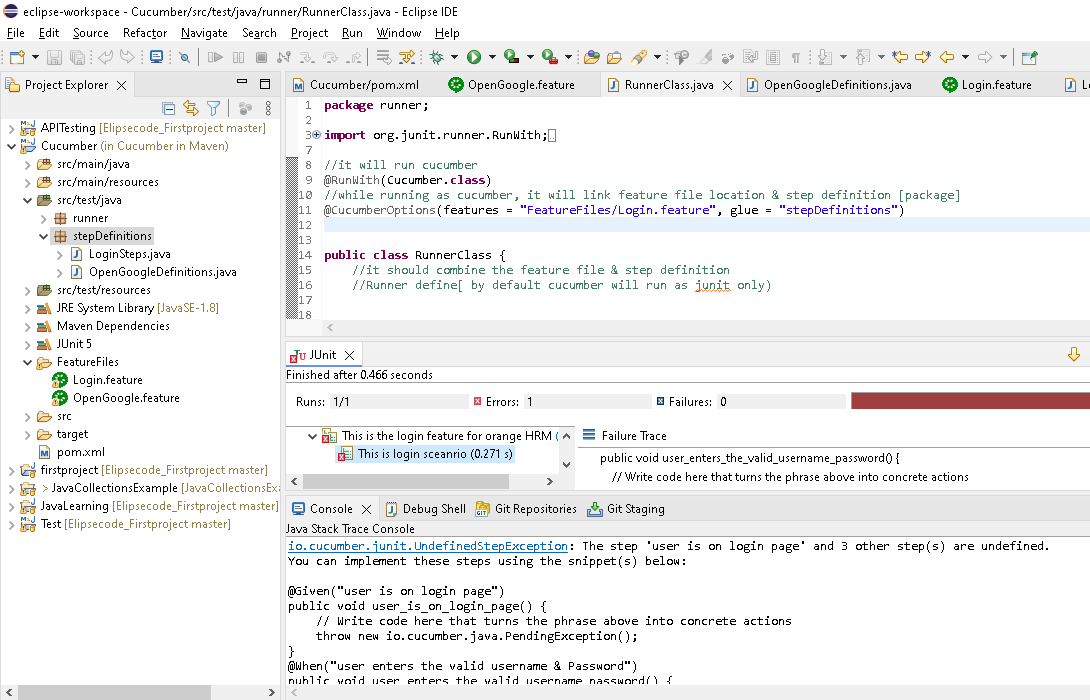
**If we have not provided any names for FeatureFiles as above..then it will run all the features present in FeatureFiles folder ..eg below**

****

**Here Login & OpenGoogle both will run**

glue = "stepDefinitions" -> stepDefinitions is a package name..

****

****

**Step4: go to LoginStep class & paste the methods and write the code for login screen**

**Feature File:**

Given user is on login page

When user enters the valid username & Password

And clicks on the login button

Then the user should be navigated to home page

**Class present in stepDefinitions**

**Note: Based on the Plain language(**Given user is on login page**) we write in feature file, same name generated/present in the method name with “\_” present(public** **void** user\_is\_on\_login\_page()**) as below in** stepDefinitions package

@Given("user is on login page")

**public** **void** user\_is\_on\_login\_page() {

// Write code here that turns the phrase above into concrete actions

}

@When("user enters the valid username & Password")

**public** **void** user\_enters\_the\_valid\_username\_password() {

// Write code here that turns the phrase above into concrete actions

}

@When("clicks on the login button")

**public** **void** clicks\_on\_the\_login\_button() {

// Write code here that turns the phrase above into concrete actions

}

@Then("the user should be navigated to home page")

**public** **void** the\_user\_should\_be\_navigated\_to\_home\_page() {

// Write code here that turns the phrase above into concrete actions

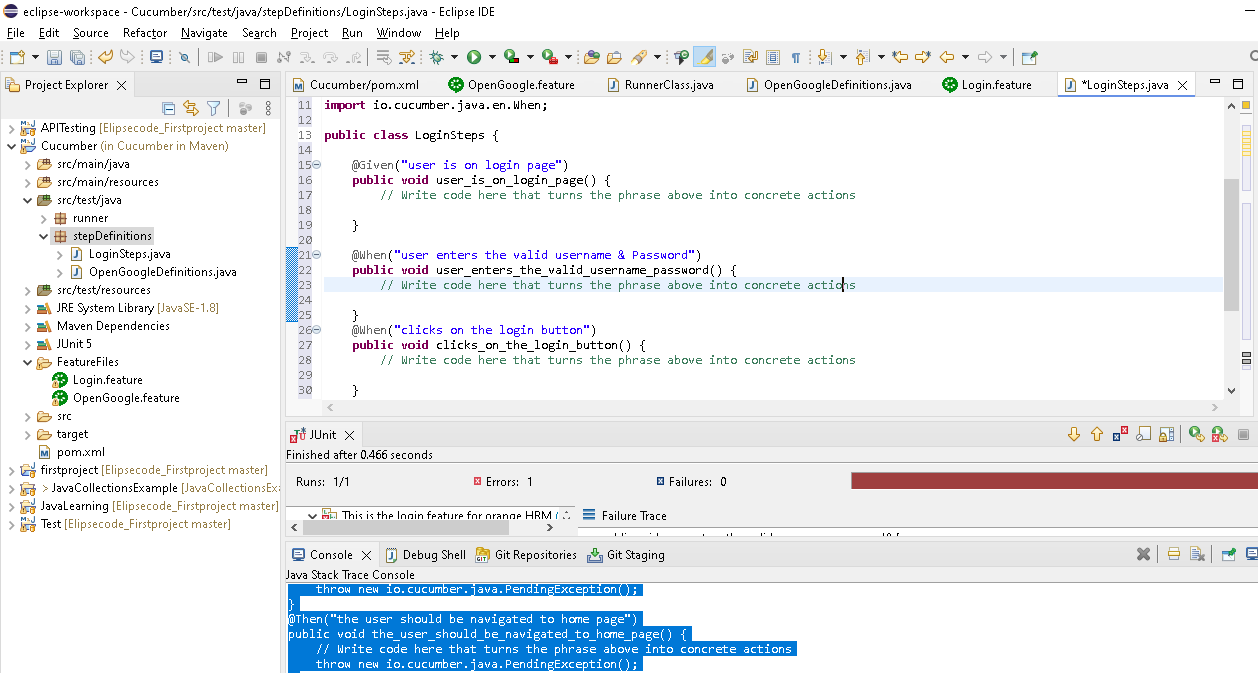
}

**Runner class: linked the feature file & step definition**

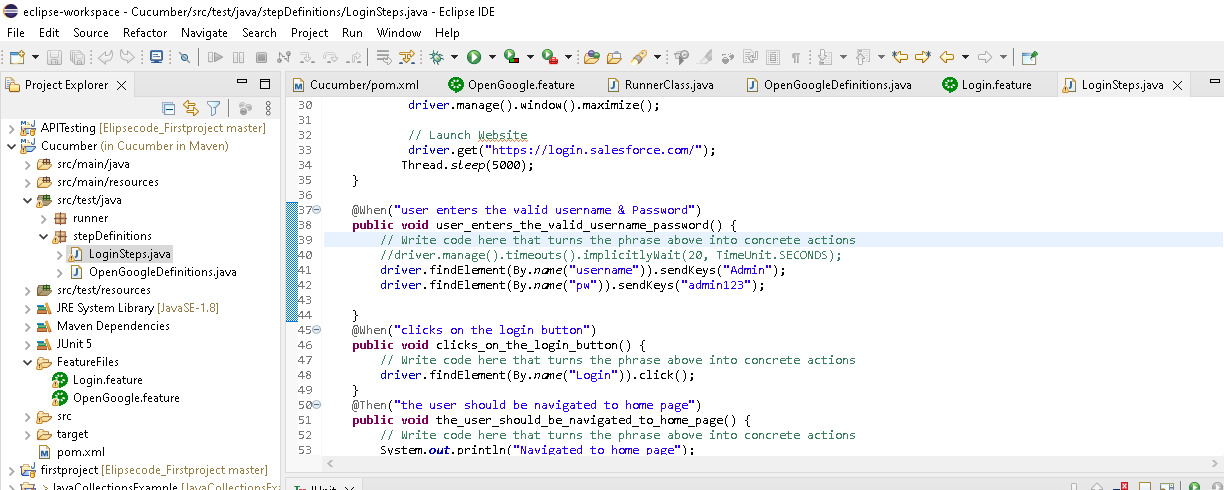
Under stepDefinitions package, in LoginSteps class, respexctive method public void user\_is\_on\_login\_page() will execute when Given user is on login page

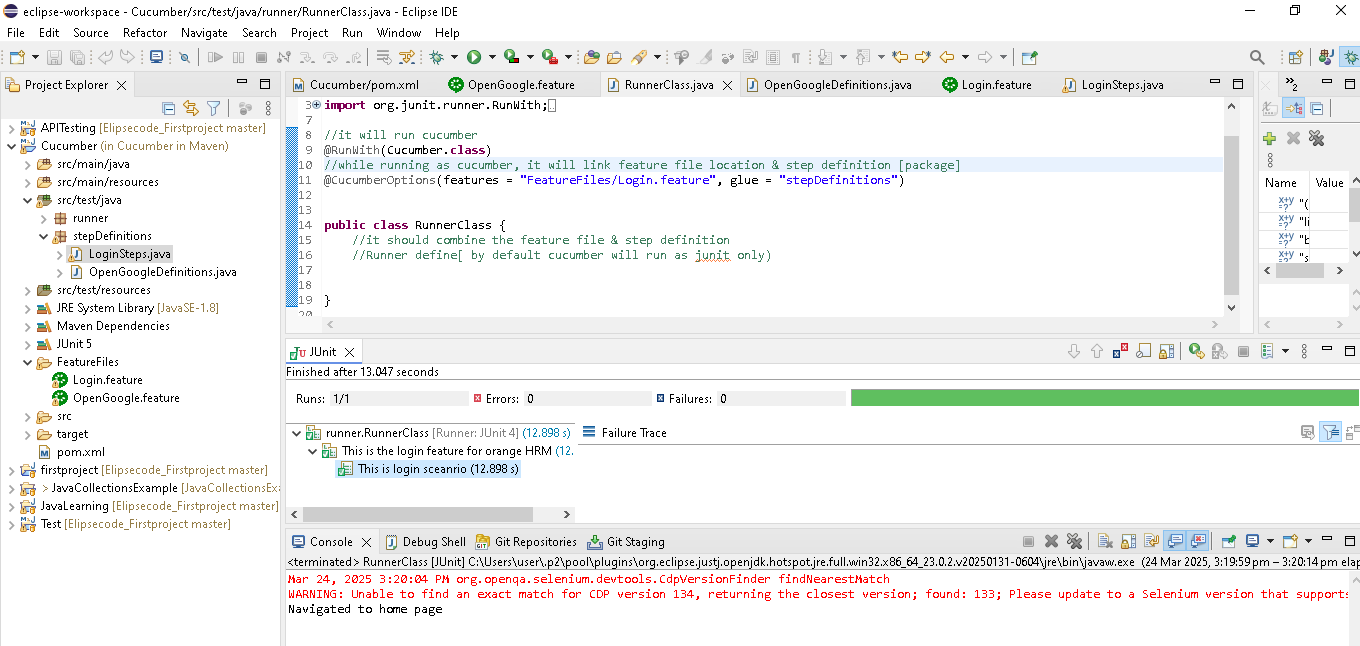
encounter

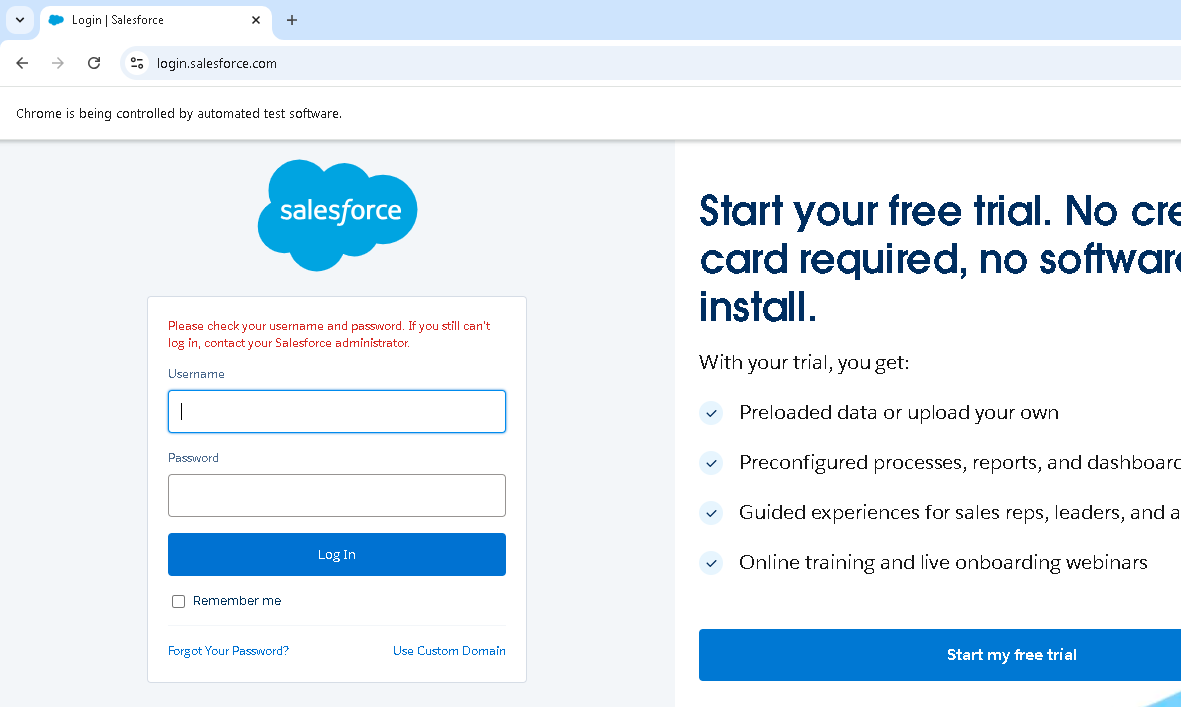
@CucumberOptions(features = "FeatureFiles/Login.feature", glue = "stepDefinitions")

****

**Code writeen**

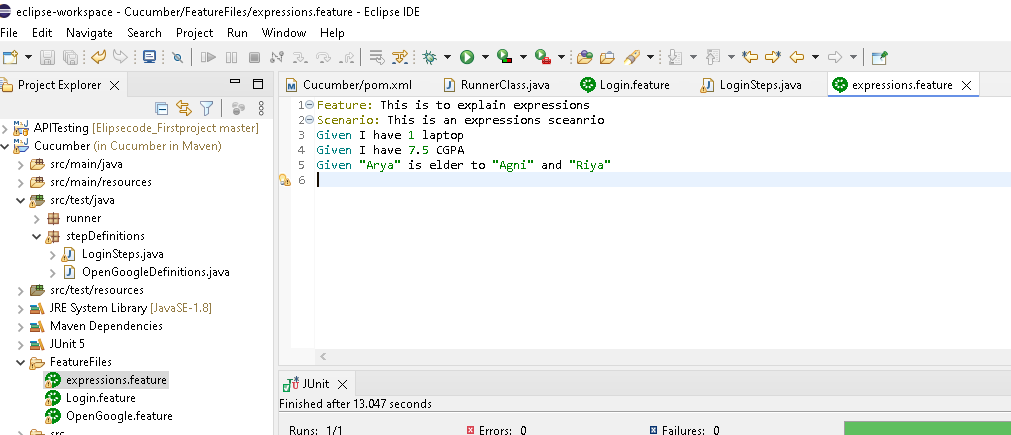
****

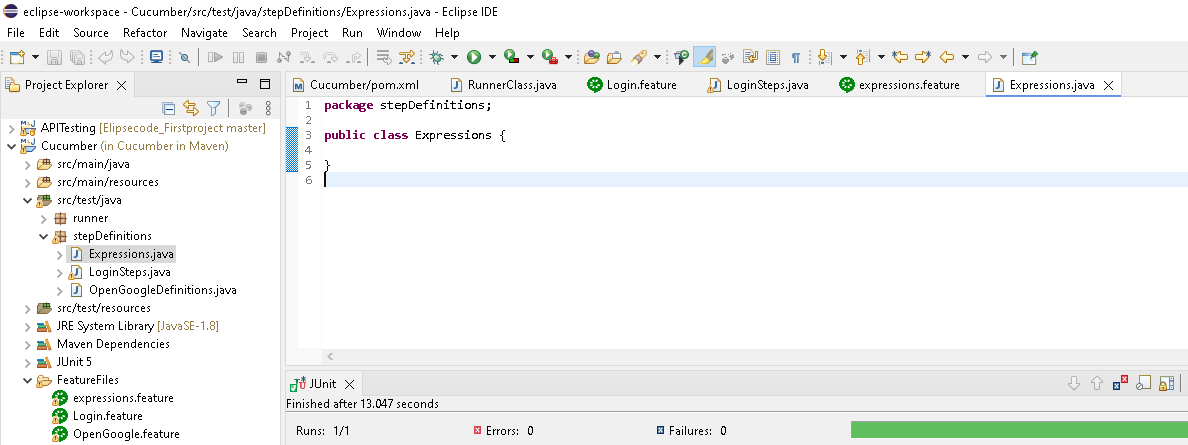
****

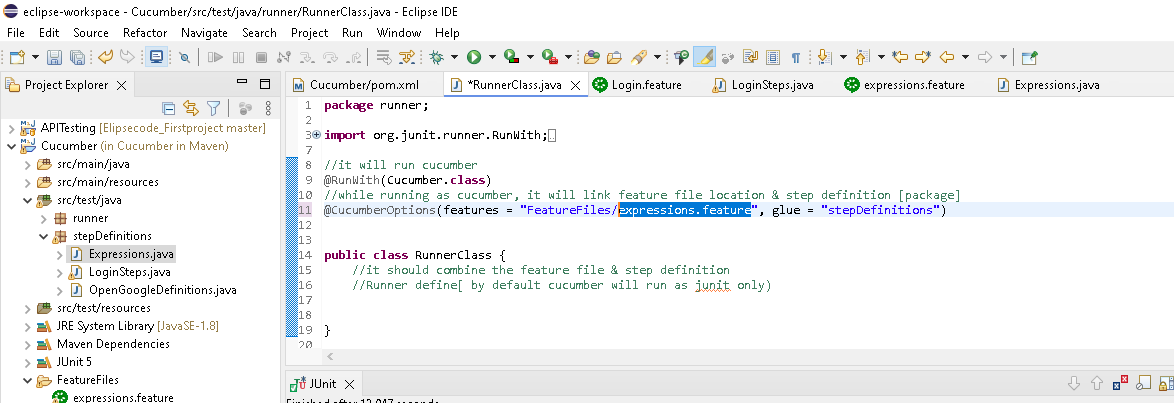
****

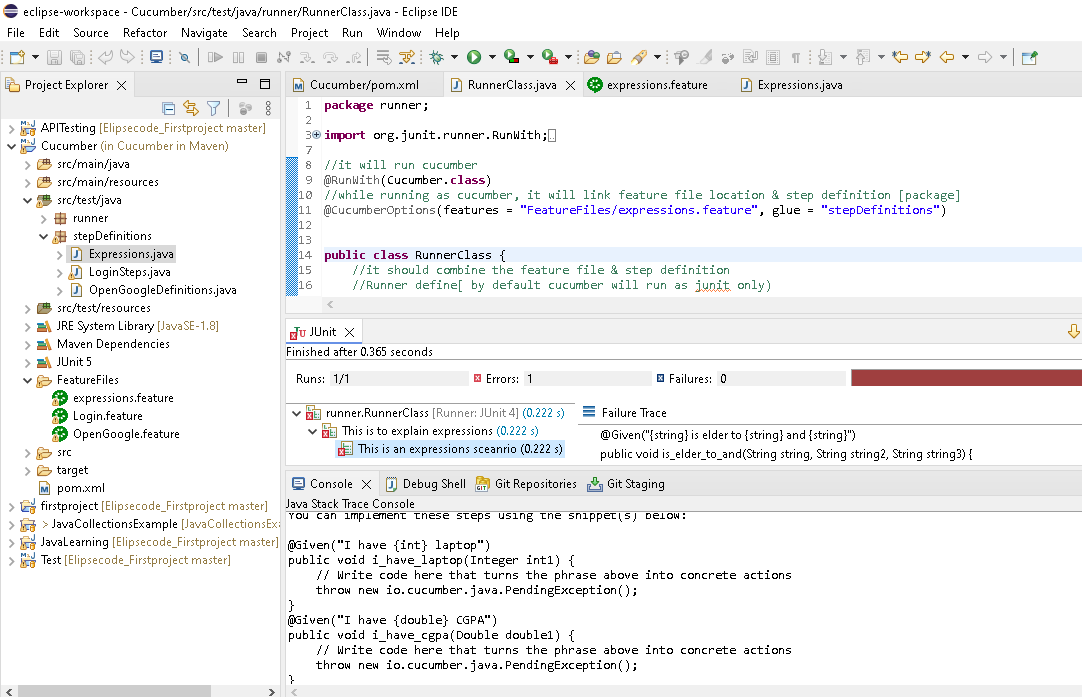
**Parameterization in Cucumber**

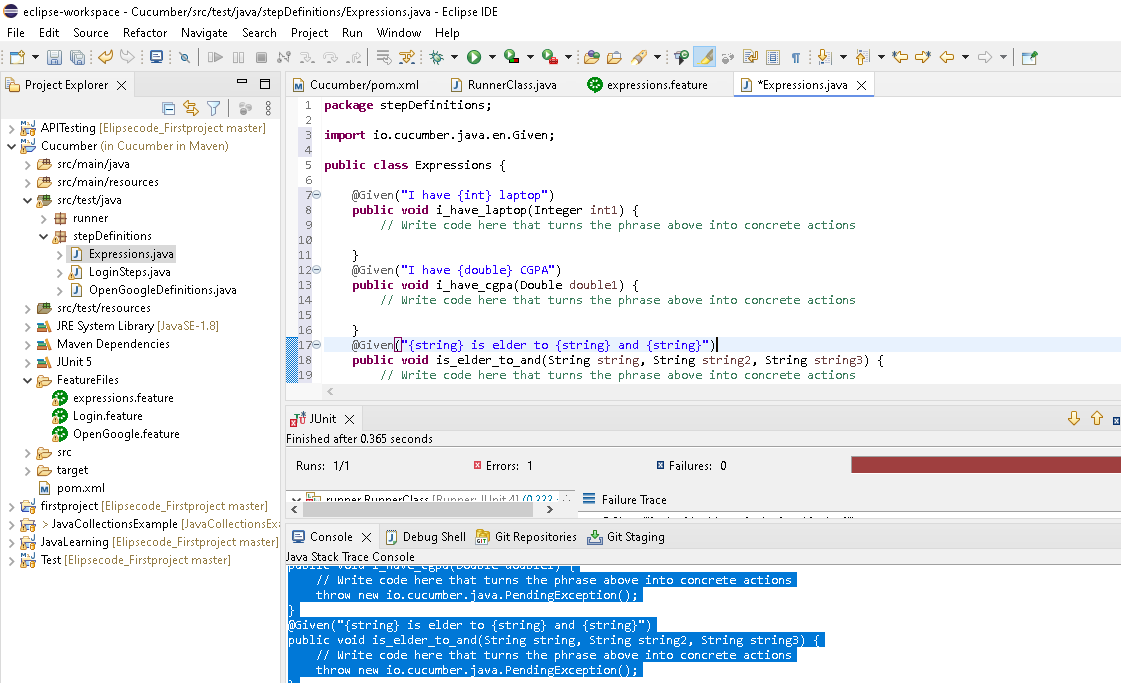
**Like sending the parameter(Number, String) from feature file to step definition**

****

****

****

****

****

**Kindly compare the feature file & stepdefinctions to understand how number, string moved as integer, double,string in their respective methods as input parameter**

**Feature file:**

Feature: This is to explain expressions

Scenario: This is an expressions sceanrio

Given I have 1 laptop

Given I have 7.5 CGPA

Given "Arya" is elder to "Agni" and "Riya"

**Stepdefinations:**

@Given("I have {int} laptop")

**public** **void** i\_have\_laptop(Integer int1) {

// Write code here that turns the phrase above into concrete actions

}

@Given("I have {double} CGPA")

**public** **void** i\_have\_cgpa(Double double1) {

// Write code here that turns the phrase above into concrete actions

}

@Given("{string} is elder to {string} and {string}")

**public** **void** is\_elder\_to\_and(String string, String string2, String string3) {

// Write code here that turns the phrase above into concrete actions

}

**package** stepDefinitions;

**import** io.cucumber.java.en.Given;

**public** **class** Expressions {

@Given("I have {int} laptop")

**public** **void** i\_have\_laptop(Integer int1) {

// Write code here that turns the phrase above into concrete actions

System.***out***.println(int1);

}

@Given("I have {double} CGPA")

**public** **void** i\_have\_cgpa(Double double1) {

// Write code here that turns the phrase above into concrete actions

System.***out***.println(double1);

}

@Given("{string} is elder to {string} and {string}")

**public** **void** is\_elder\_to\_and(String string, String string2, String string3) {

// Write code here that turns the phrase above into concrete actions

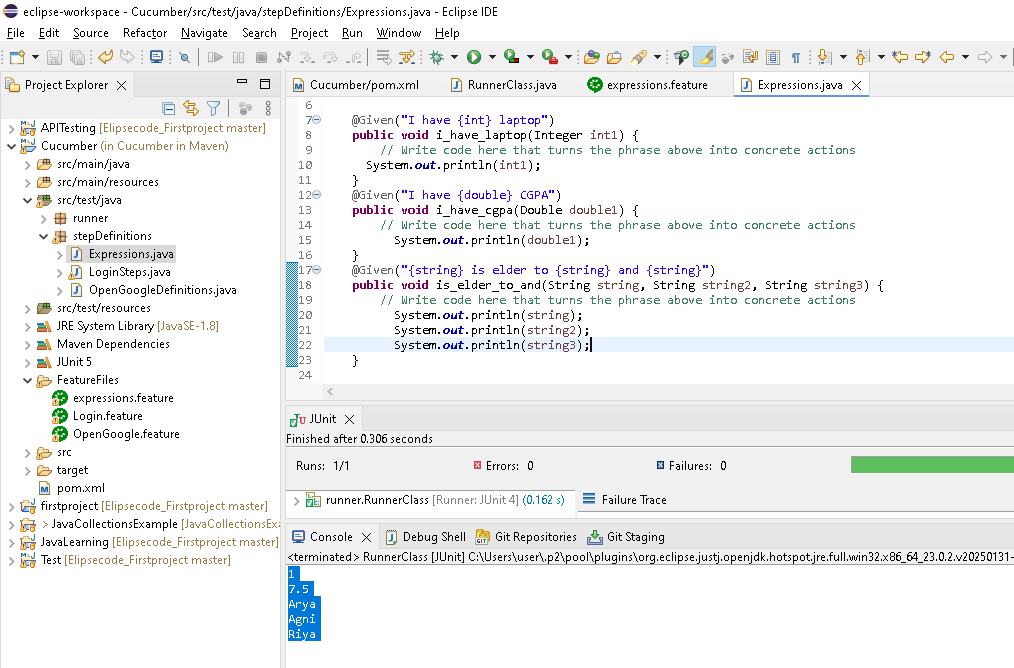
System.***out***.println(string);

System.***out***.println(string2);

System.***out***.println(string3);

}

}



**Data Driven testing in cucumber (Sending data via feature File)**

1.Data tables without headers

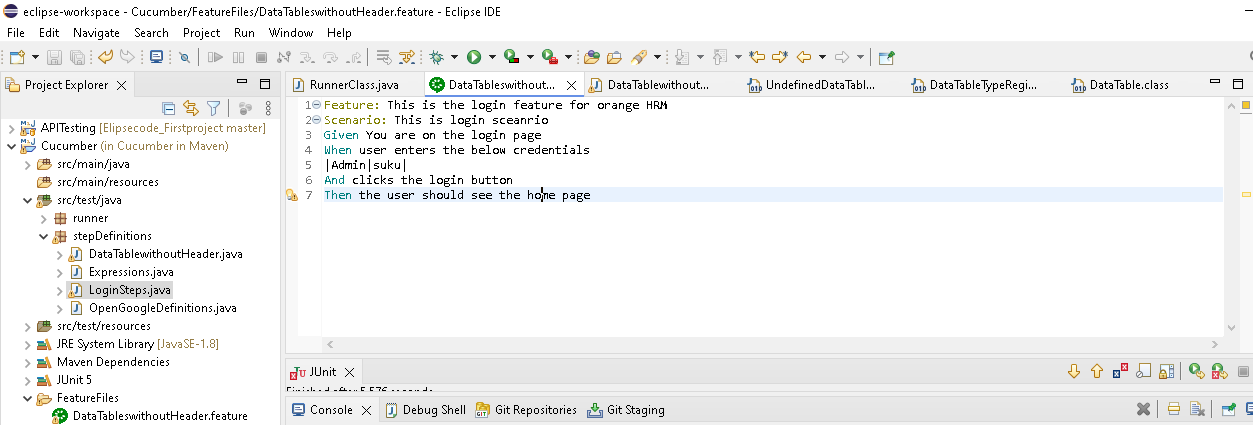
2.Data tables with headers

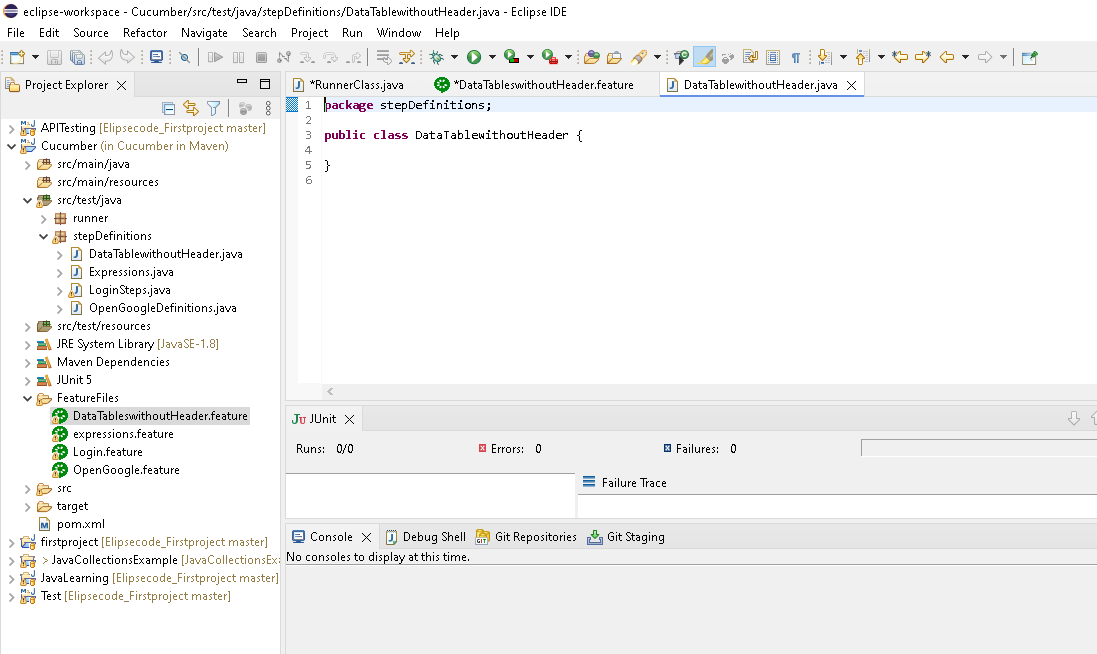
3.Using Data tables and examples

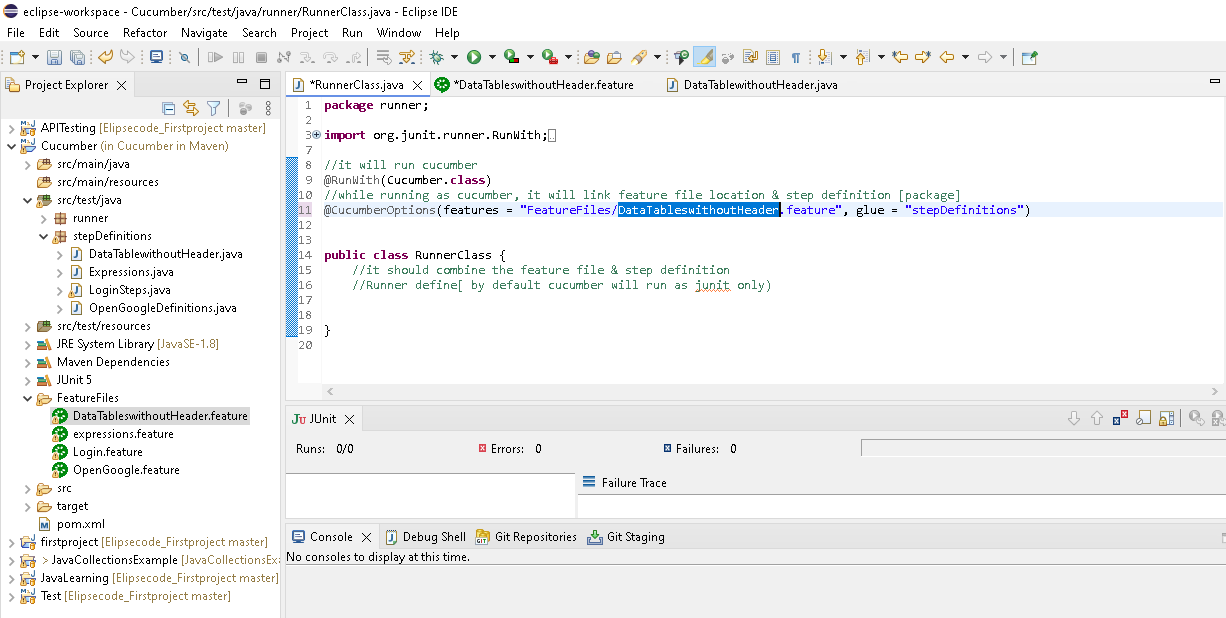
1.Data tables without headers

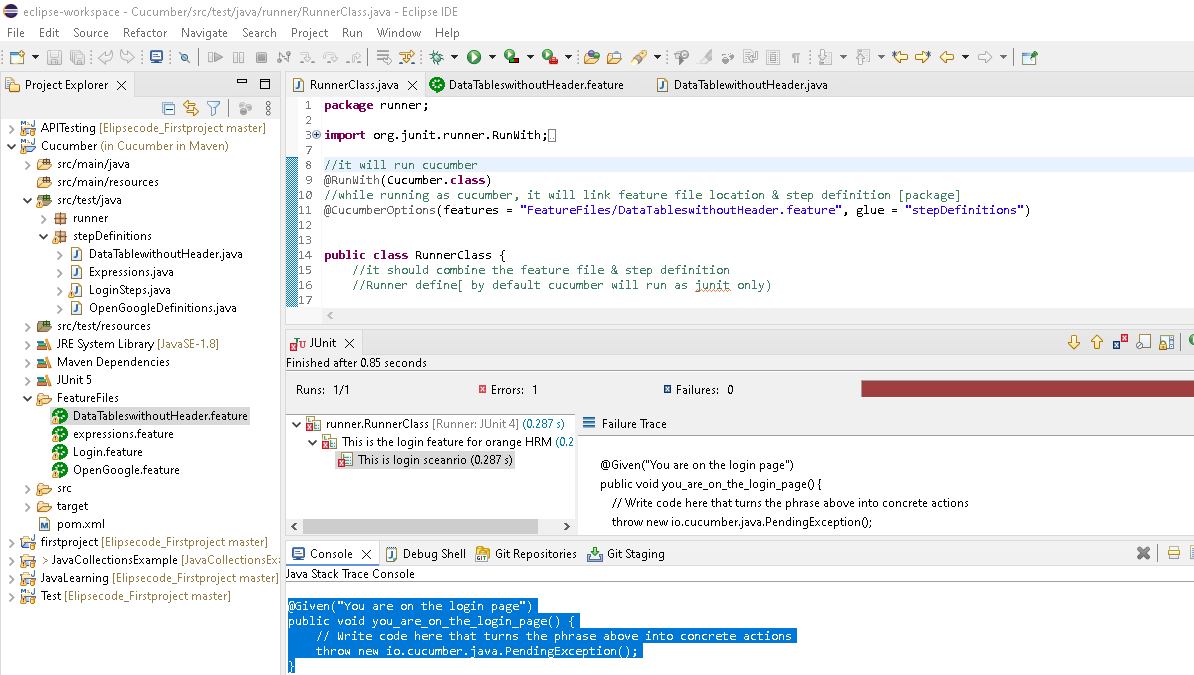
By using “|” symbol in feature file

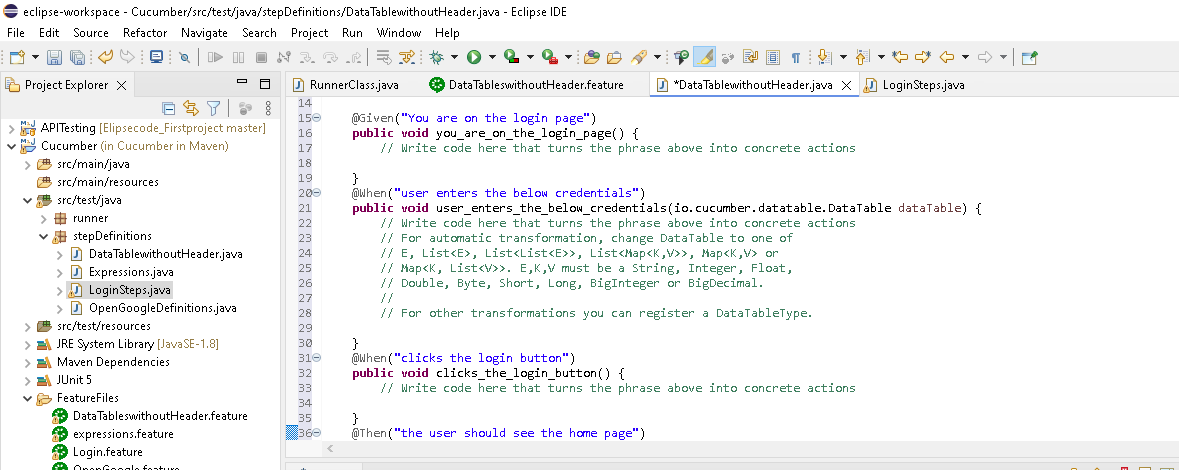
|Admin | Suku|











**package** stepDefinitions;

**import** java.util.List;

**import** org.openqa.selenium.By;

**import** org.openqa.selenium.Keys;

**import** org.openqa.selenium.WebDriver;

**import** org.openqa.selenium.chrome.ChromeDriver;

**import** io.cucumber.java.en.Given;

**import** io.cucumber.java.en.Then;

**import** io.cucumber.java.en.When;

**public** **class** DataTablewithoutHeader {

WebDriver driver;

@Given("You are on the login page")

**public** **void** you\_are\_on\_the\_login\_page() **throws** InterruptedException {

// Write code here that turns the phrase above into concrete actions

System.*setProperty*(

"webdriver.chrome.driver",

"C:\\Users\\user\\Downloads\\chromedriver.exe");

// Instantiate a ChromeDriver class.

driver = **new** ChromeDriver();

//C:\Users\\user\\Downloads\\chromedriver-win64\\chromedriver-win64

// Maximize the browser

driver.manage().window().maximize();

// Launch Website

driver.get("https://login.salesforce.com/");

// Thread.sleep(5000);

}

@When("user enters the below credentials")

**public** **void** user\_enters\_the\_below\_credentials(io.cucumber.datatable.DataTable dataTable) {

// Write code here that turns the phrase above into concrete actions

// For automatic transformation, change DataTable to one of

// E, List<E>, List<List<E>>, List<Map<K,V>>, Map<K,V> or

// Map<K, List<V>>. E,K,V must be a String, Integer, Float,

// Double, Byte, Short, Long, BigInteger or BigDecimal.

//

// For other transformations you can register a DataTableType.

//convert datatable to list by using "asList" & read the value

//|Admin | admin123 ->both string value only used (String.class)

/\*

\* List<String> str = dataTable.asList(String.class);

\*

\* String username = str.get(0); String PW = str.get(1);

\* driver.findElement(By.name("username")).sendKeys(username);

\* driver.findElement(By.name("pw")).sendKeys(PW);

\*/

List<List<String>> data = dataTable.asLists(String.**class**);

String username = data.get(0).get(0);

String PW = data.get(0).get(1);

driver.findElement(By.*name*("username")).sendKeys(username);

driver.findElement(By.*name*("pw")).sendKeys(PW);

}

@When("clicks the login button")

**public** **void** clicks\_the\_login\_button() {

// Write code here that turns the phrase above into concrete actions

driver.findElement(By.*name*("Login")).click();

}

@Then("the user should see the home page")

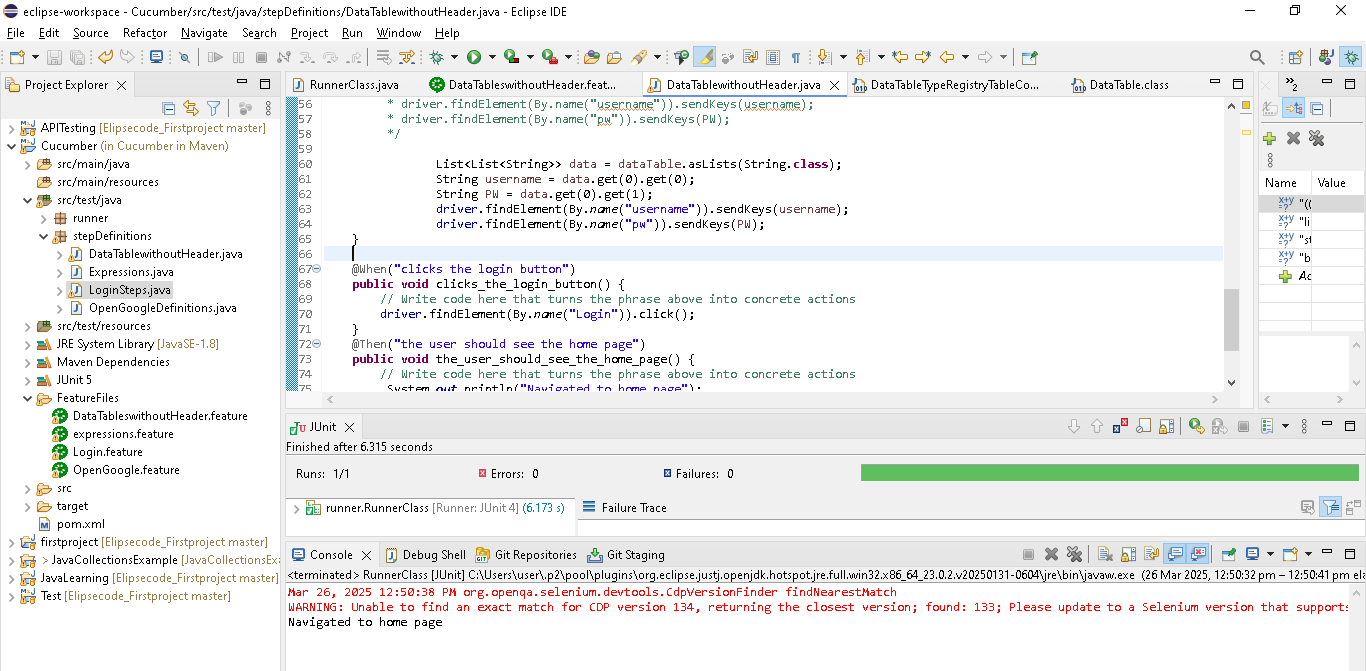
**public** **void** the\_user\_should\_see\_the\_home\_page() {

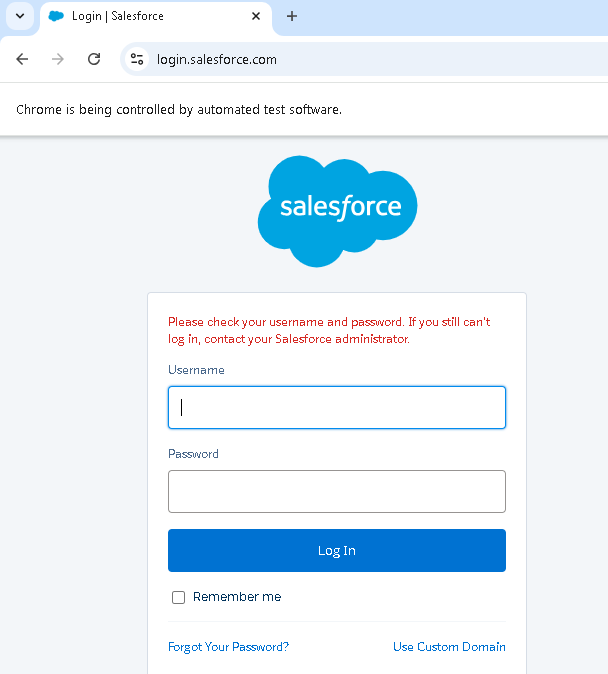
// Write code here that turns the phrase above into concrete actions

System.***out***.println("Navigated to home page");

}

}

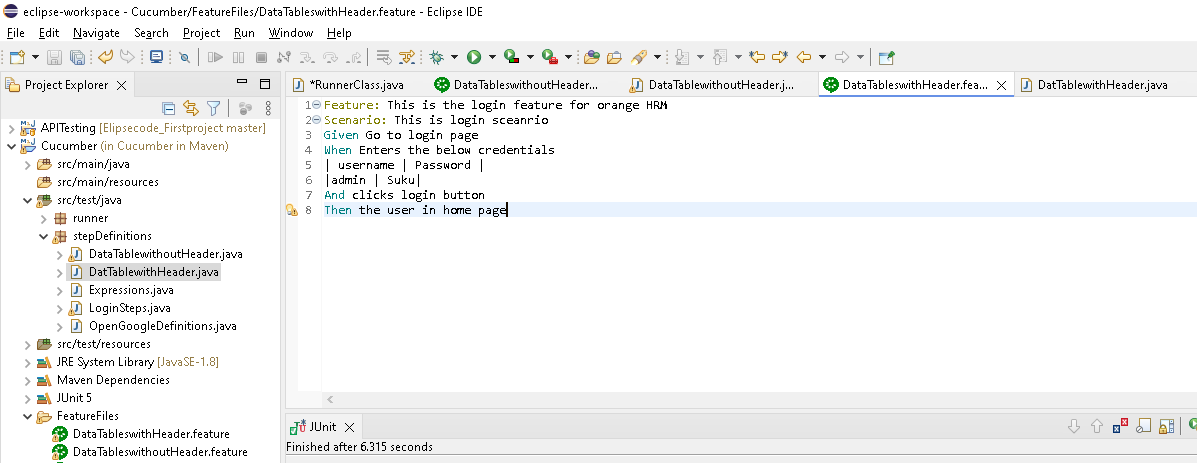


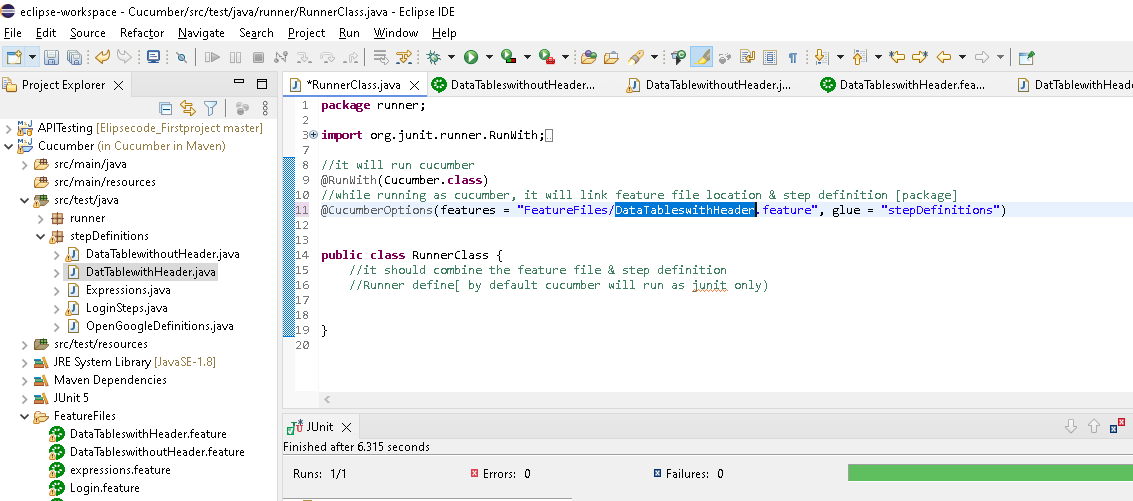


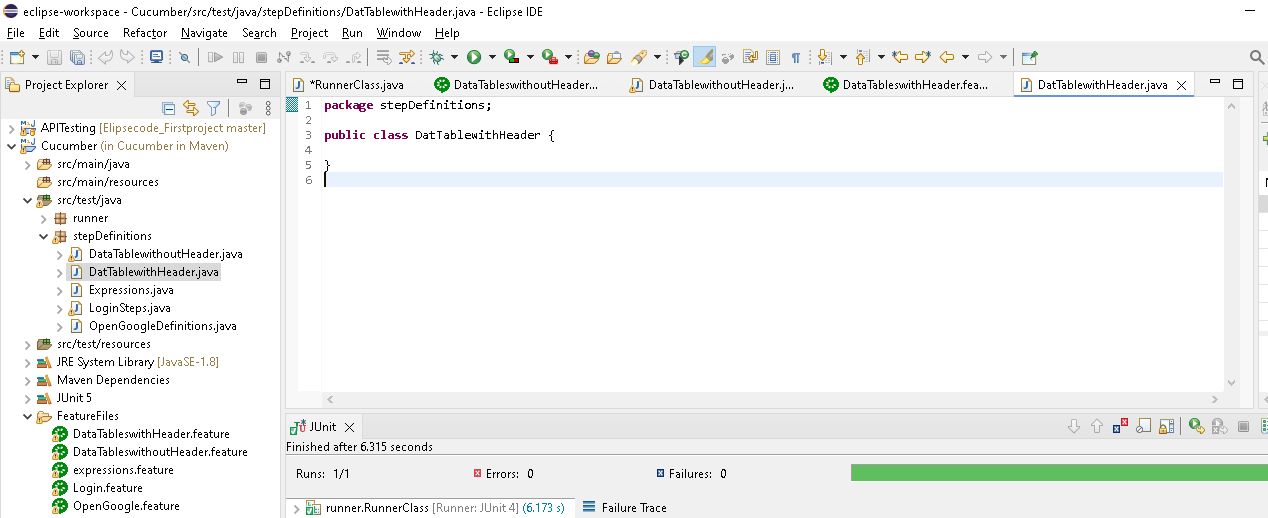
2.Data tables with headers

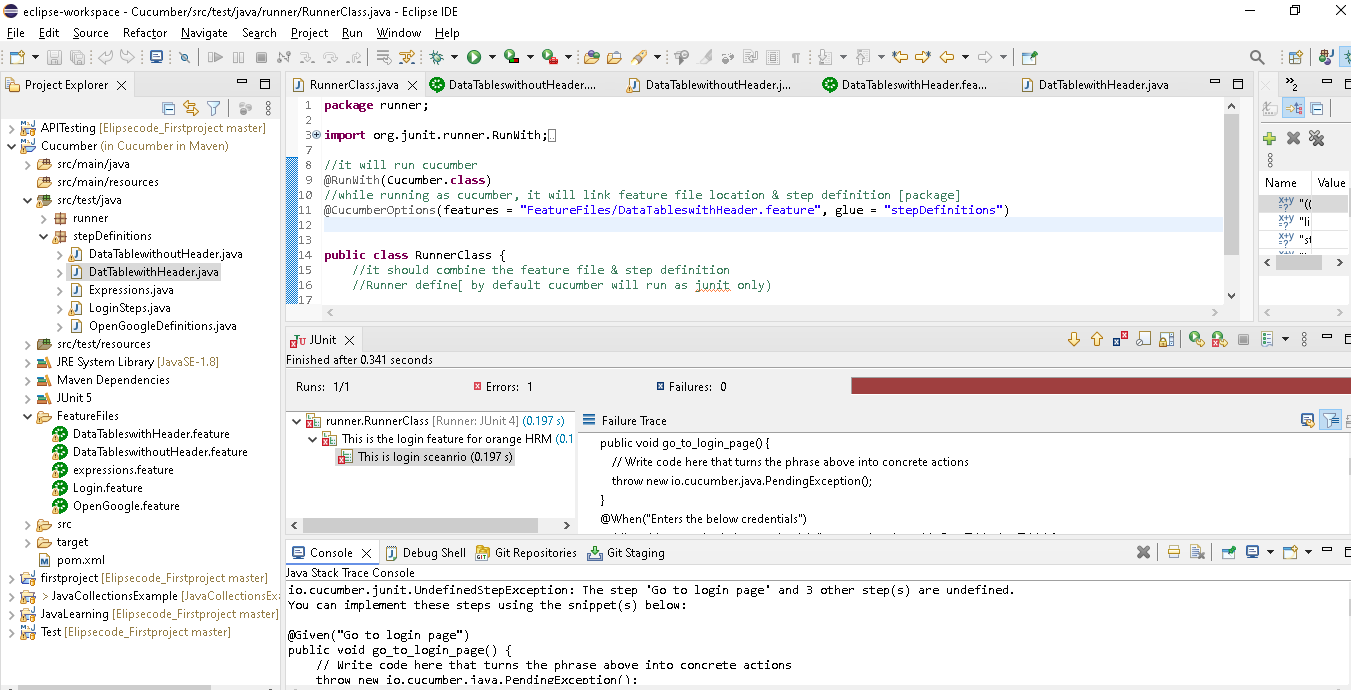
| username | Password |

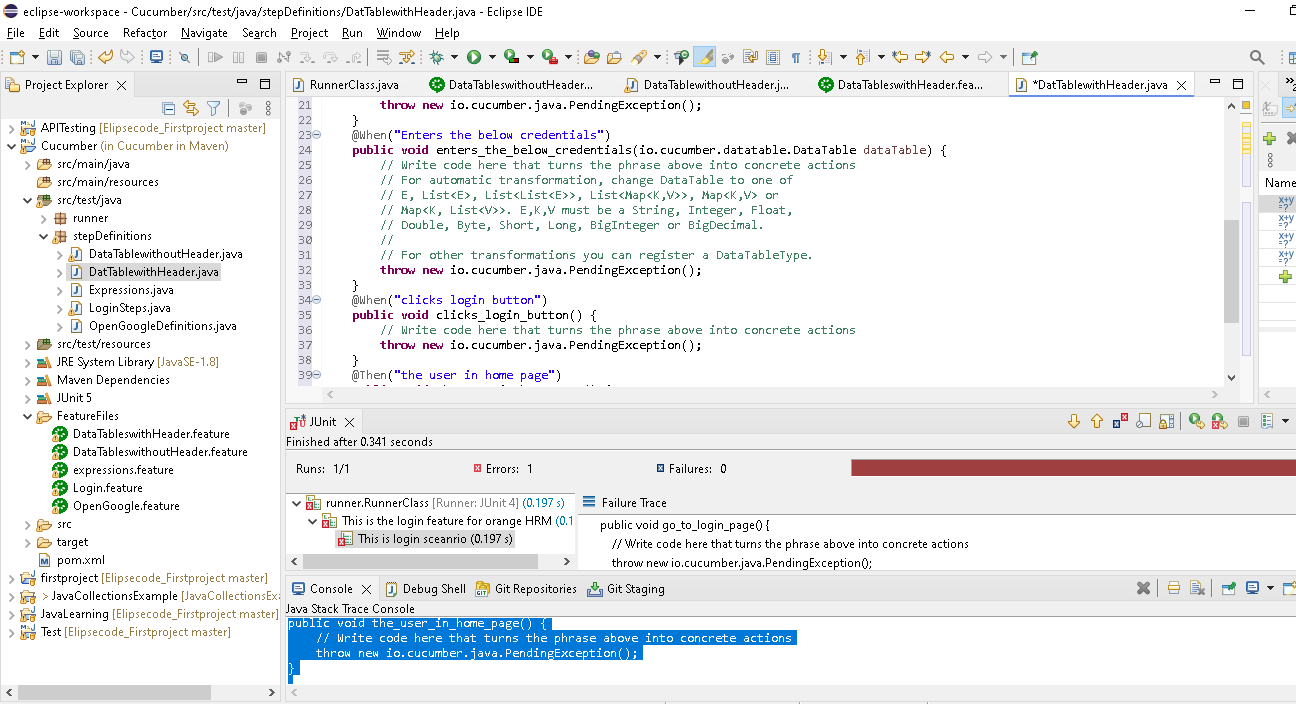
|admin | Suku|











**package** stepDefinitions;

**import** java.util.List;

**import** java.util.Map;

**import** org.openqa.selenium.By;

**import** org.openqa.selenium.Keys;

**import** org.openqa.selenium.WebDriver;

**import** org.openqa.selenium.chrome.ChromeDriver;

**import** io.cucumber.java.en.Given;

**import** io.cucumber.java.en.Then;

**import** io.cucumber.java.en.When;

**public** **class** DatTablewithHeader {

WebDriver driver;

@Given("Go to login page")

**public** **void** go\_to\_login\_page() {

System.*setProperty*(

"webdriver.chrome.driver",

"C:\\Users\\user\\Downloads\\chromedriver.exe");

// Instantiate a ChromeDriver class.

driver = **new** ChromeDriver();

//C:\Users\\user\\Downloads\\chromedriver-win64\\chromedriver-win64

// Maximize the browser

driver.manage().window().maximize();

// Launch Website

driver.get("https://login.salesforce.com/");

// Thread.sleep(5000);

}

@When("Enters the below credentials")

**public** **void** enters\_the\_below\_credentials(io.cucumber.datatable.DataTable dataTable) {

// Write code here that turns the phrase above into concrete actions

// For automatic transformation, change DataTable to one of

// E, List<E>, List<List<E>>, List<Map<K,V>>, Map<K,V> or

// Map<K, List<V>>. E,K,V must be a String, Integer, Float,

// Double, Byte, Short, Long, BigInteger or BigDecimal.

//

// For other transformations you can register a DataTableType.

//here we are using Map<K,V> (asmaps)....as input behave as key & value

//convert datatable to map by using "asMpas" & read the value

//|Admin | Suku ->both string value only used (String.class)

List<Map<String,String>> data = dataTable.asMaps(String.**class**, String.**class**);

String username = data.get(0).get("username");

String PW = data.get(0).get("Password");

driver.findElement(By.*name*("username")).sendKeys(username);

driver.findElement(By.*name*("pw")).sendKeys(PW);

}

@When("clicks login button")

**public** **void** clicks\_login\_button() {

// Write code here that turns the phrase above into concrete actions

driver.findElement(By.*name*("Login")).click();

}

@Then("the user in home page")

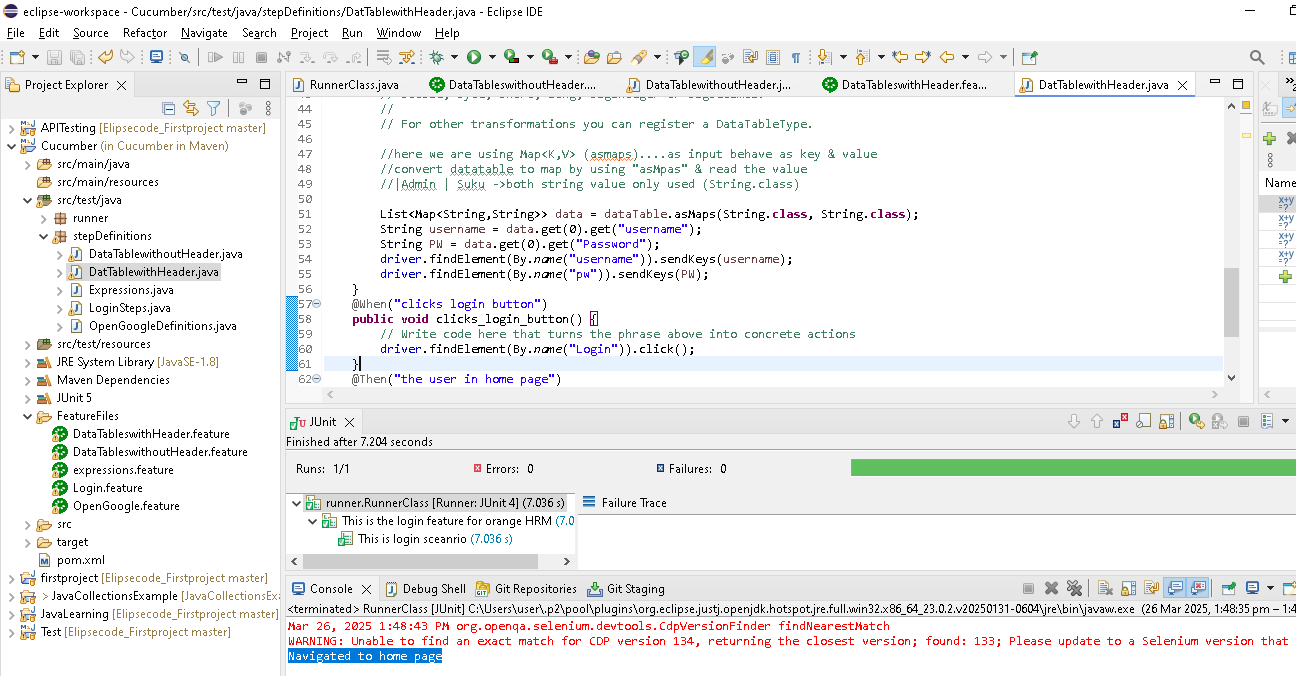
**public** **void** the\_user\_in\_home\_page() {

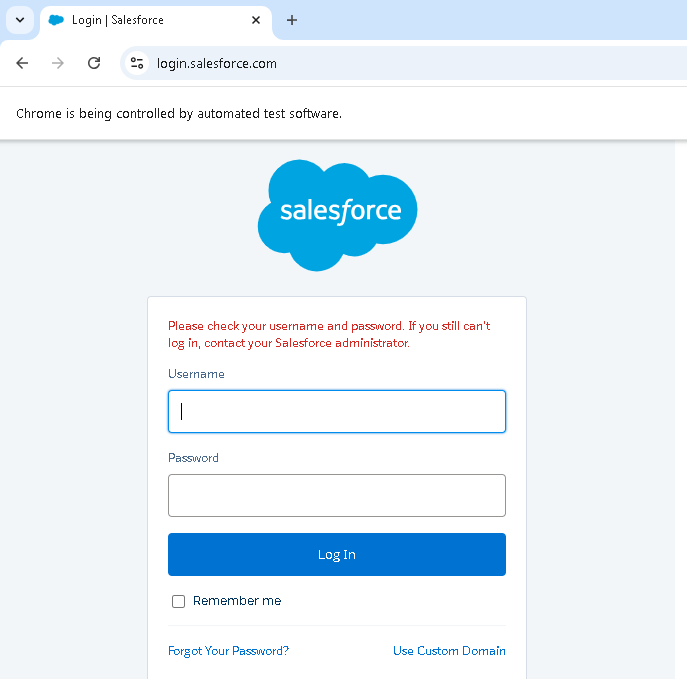
// Write code here that turns the phrase above into concrete actions

System.***out***.println("Navigated to home page");

}

}





3.Using Data tables and examples [ same as above only Examples: is present]

Examples:

| username | Password |

|admin | Suku|

**Whenuser enters "<username>" and password "<Password>" ->here we are give the value in “<>”(xml tag) tht means it will get the value from data table only**

**Note: Whenuser enters "username" and password "Password" -> “<>” not given,so its consider as normal string value as input to step definition**

Feature: This is the login feature for orange HRM

Scenario Outline: This is login sceanrio

Given Proceed to login page

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

And clicks login buttonss

Then User in home page

Examples:

|username | Password |

|admin | Suku|

**If we are using Examples:, then “Scenario Outline” should use in feature file instead of “Scenario”..Because if we have more no of data as below..then test case will execute twice as we have 2 test data in examples..**

**Scenario Outline having the capacity to execute the test cases multiple times based on no of test data**

Examples:

|username | Password |

|admin | Suku|

|admin12 | Suku12|

---------------------------------------------

Feature: This is the login feature for orange HRM

Scenario Outline: This is login sceanrio

Given Proceed to login page

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

And clicks login buttonss

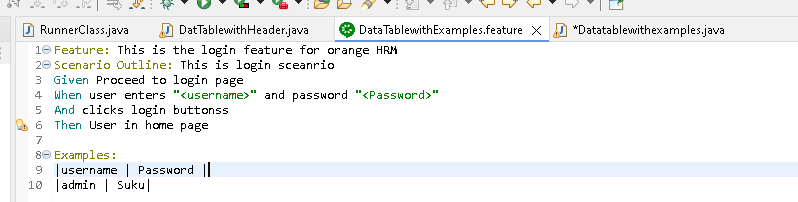
Then User in home page

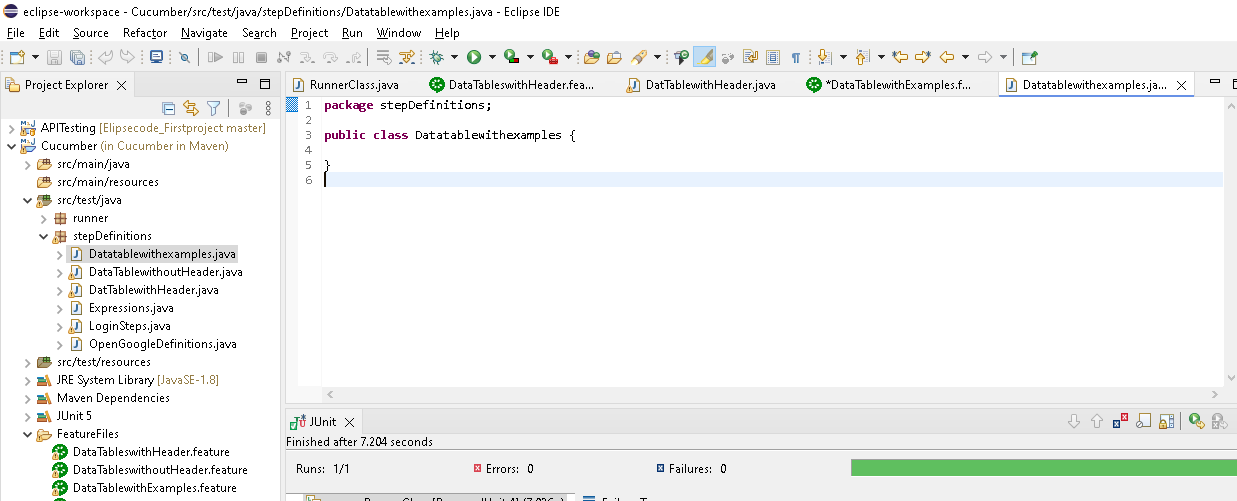
Examples:

|username | Password |

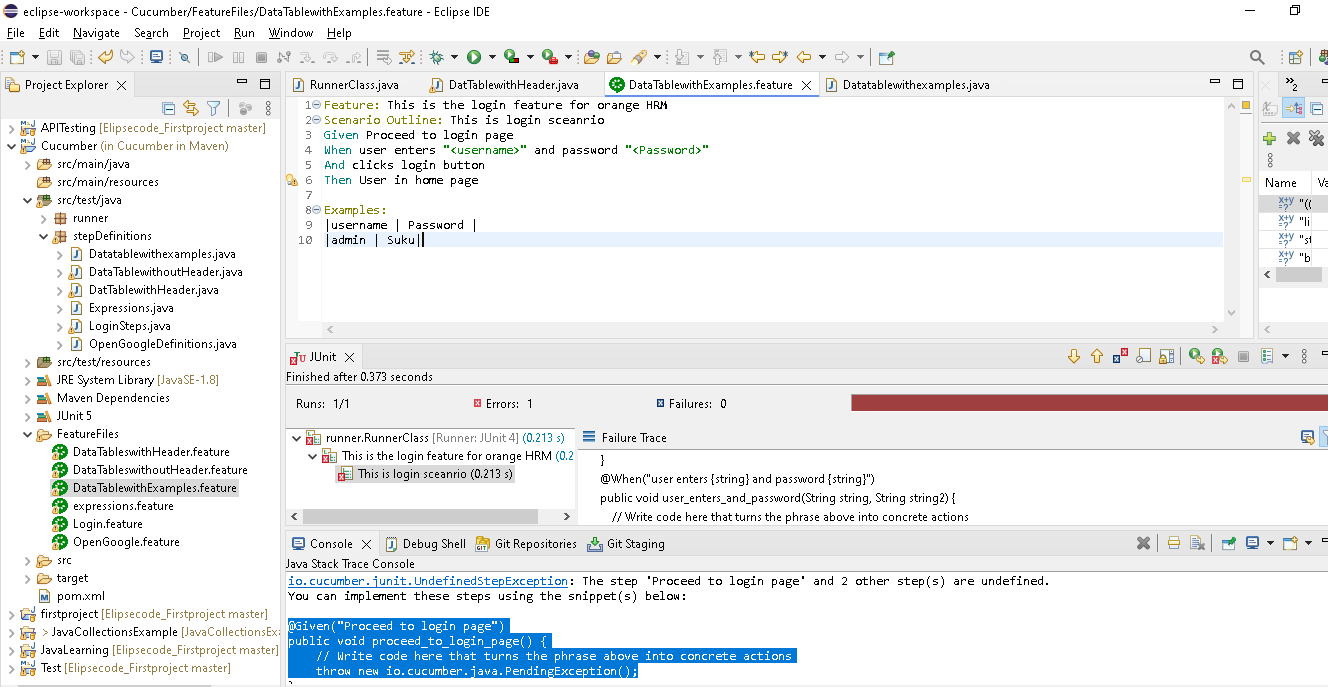
|admin | Suku|

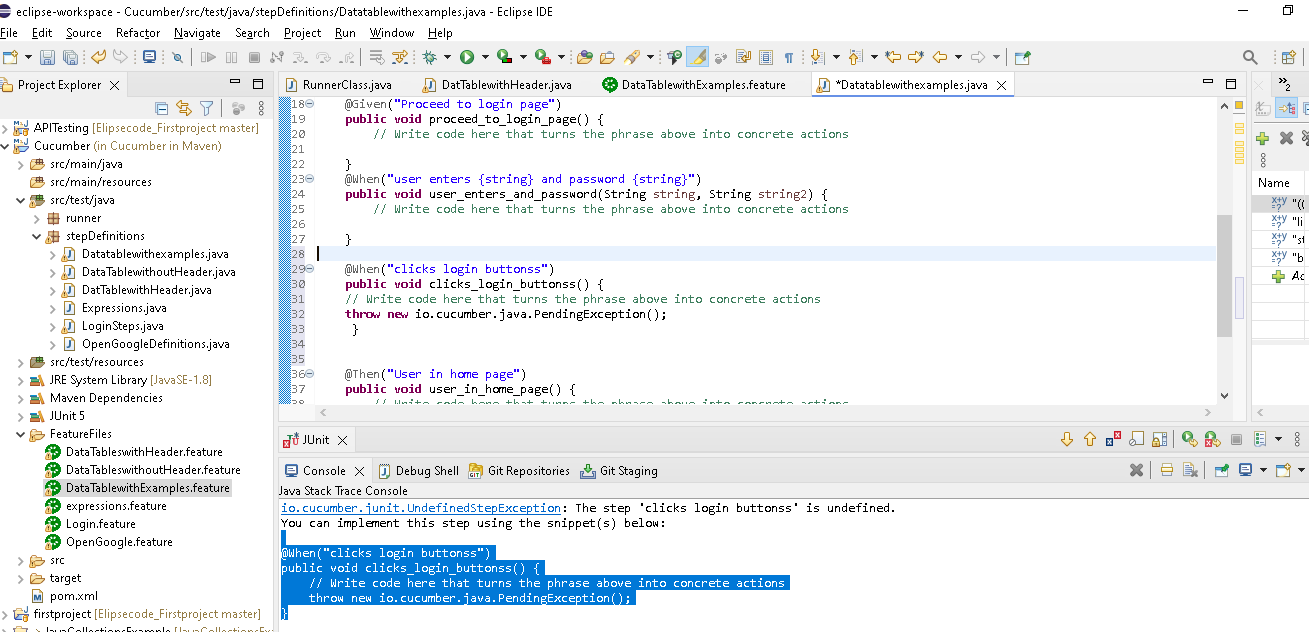
|admin12 | Suku12|











**package** stepDefinitions;

**import** java.util.List;

**import** java.util.Map;

**import** org.openqa.selenium.By;

**import** org.openqa.selenium.Keys;

**import** org.openqa.selenium.WebDriver;

**import** org.openqa.selenium.chrome.ChromeDriver;

**import** io.cucumber.java.en.Given;

**import** io.cucumber.java.en.Then;

**import** io.cucumber.java.en.When;

**public** **class** Datatablewithexamples {

WebDriver driver;

@Given("Proceed to login page")

**public** **void** proceed\_to\_login\_page() {

// Write code here that turns the phrase above into concrete actions

System.*setProperty*(

"webdriver.chrome.driver",

"C:\\Users\\user\\Downloads\\chromedriver.exe");

// Instantiate a ChromeDriver class.

driver = **new** ChromeDriver();

//C:\Users\\user\\Downloads\\chromedriver-win64\\chromedriver-win64

// Maximize the browser

driver.manage().window().maximize();

// Launch Website

driver.get("https://login.salesforce.com/");

// Thread.sleep(5000);

}

@When("user enters {string} and password {string}")

**public** **void** user\_enters\_and\_password(String string, String string2) {

// Write code here that turns the phrase above into concrete actions

driver.findElement(By.*name*("username")).sendKeys(string);

driver.findElement(By.*name*("pw")).sendKeys(string2);

}

@When("clicks login buttonss")

**public** **void** clicks\_login\_buttonss() {

// Write code here that turns the phrase above into concrete actions

driver.findElement(By.*name*("Login")).click();

}

@Then("User in home page")

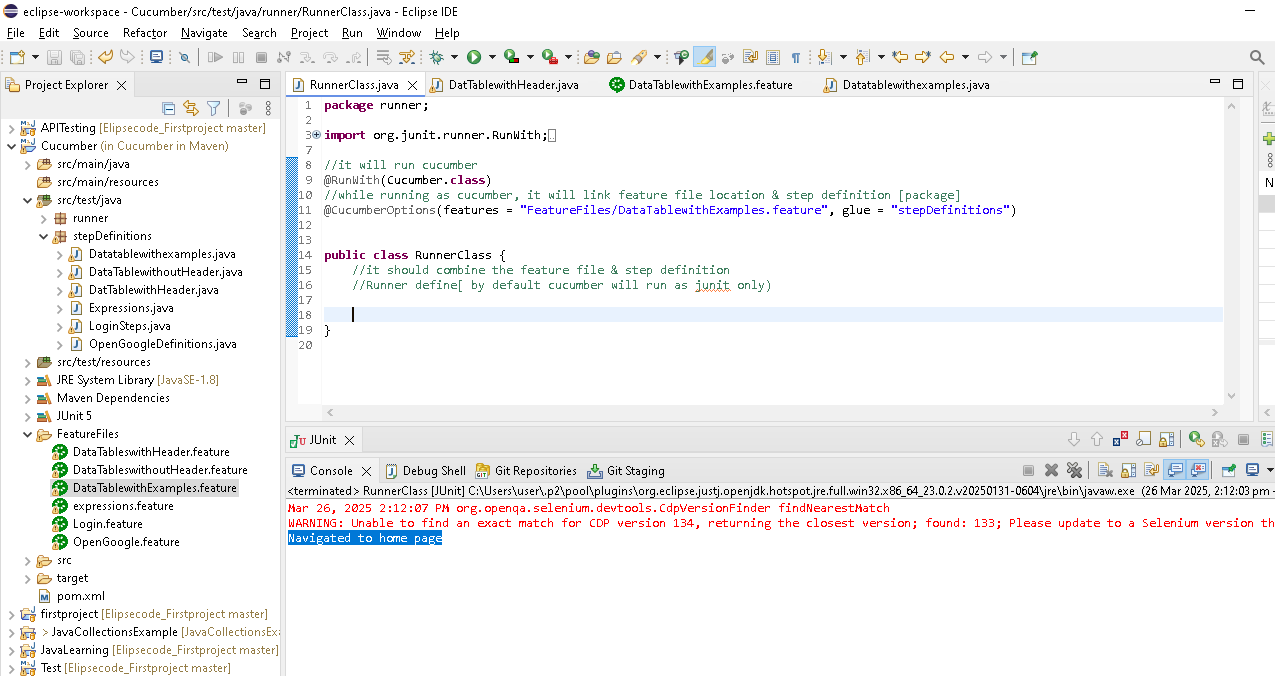
**public** **void** user\_in\_home\_page() {

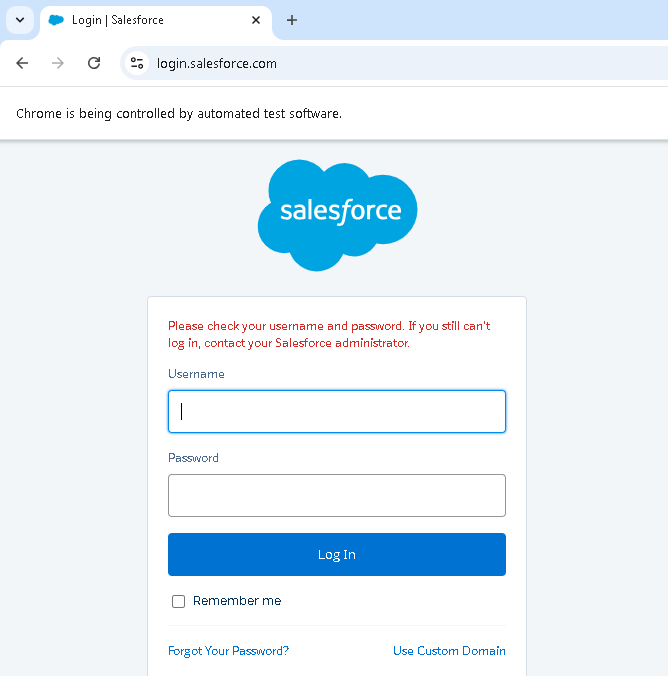
// Write code here that turns the phrase above into concrete actions

System.***out***.println("Navigated to home page");

}

}





**If we are using Examples:, then “Scenario Outline” should use in feature file instead of “Scenario”..Because if we have more no of data as below..then test case(give,when,and) will execute twice as we have 2 test data in examples..**

**Scenario Outline having the capacity to execute the test cases(give,when,and) multiple times based on no of test data**

Examples:

|username | Password |

|admin | Suku|

|admin12 | Suku12|

---------------------------------------------

Feature: This is the login feature for orange HRM

Scenario Outline: This is login sceanrio

Given Proceed to login page

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

And clicks login buttonss

Then User in home page

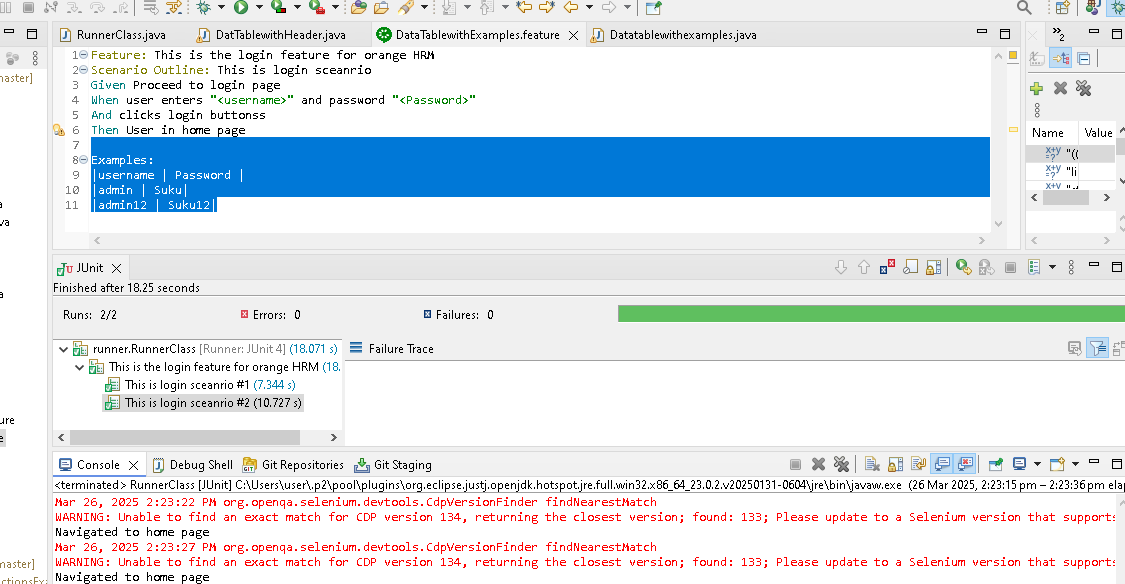
Examples:

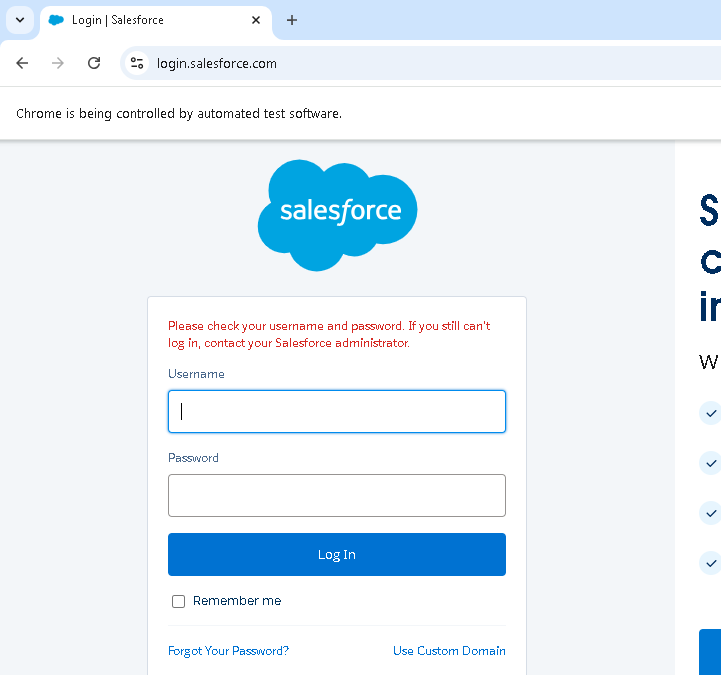
|username | Password |

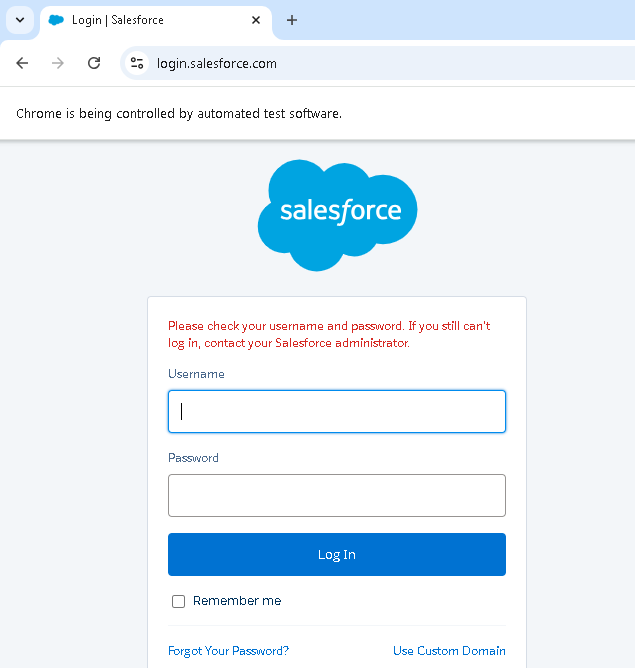
|admin | Suku|

|admin12 | Suku12|

Executed twice refer below







-----------------------------------------------------------------

**Cucumber Options**

1.**Features ->** contains feature file location

2.**Glue ->** have stepDefinitions (packages name)

@CucumberOptions(features = "FeatureFiles/DataTablewithExamples.feature", glue = "stepDefinitions")

3.**dry run ->**

if dry run is true check for mapping without execution

if dry run is false, just execute & not check the mapping

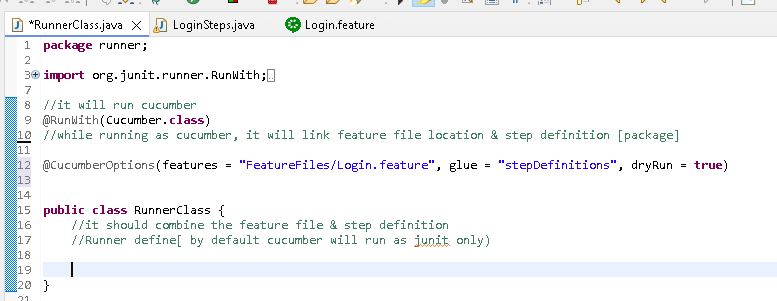
like self evolution of all feature file step corresponding/mapping method present in step definition packages & have Boolean value

If we set dry run= true , then it ill compare the feature file & step definition for all feature file step corresponding method present in step definition packages..it will not execute the application flow(like launching application etc)

Eg in real project we have multiple no of steps in feature file...in that **dry run= true** used to check all feature file step corresponding method present in step definition packages..once all mapping correct, then we can execute the code by dryRun = **false**

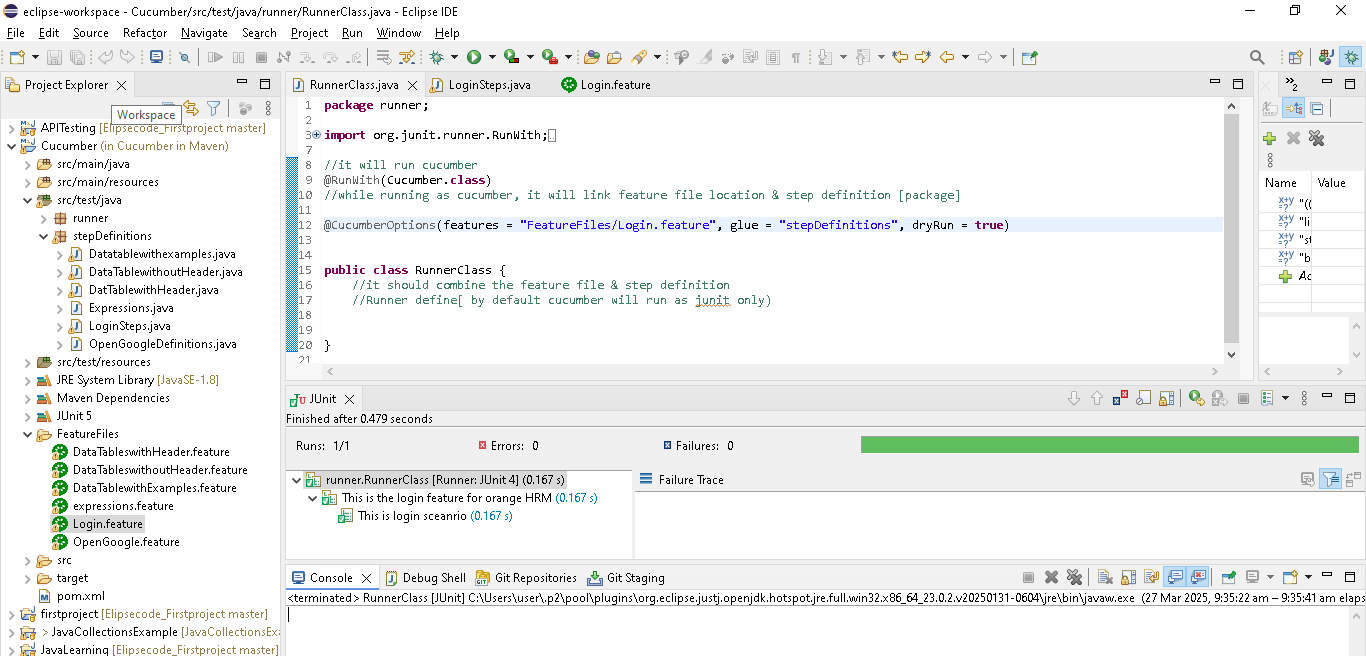
@CucumberOptions(features = "FeatureFiles/Login.feature", glue = "stepDefinitions", dryRun = **true**)

If dry run = false, then it will execute the methods..it will not compare the files & step defination

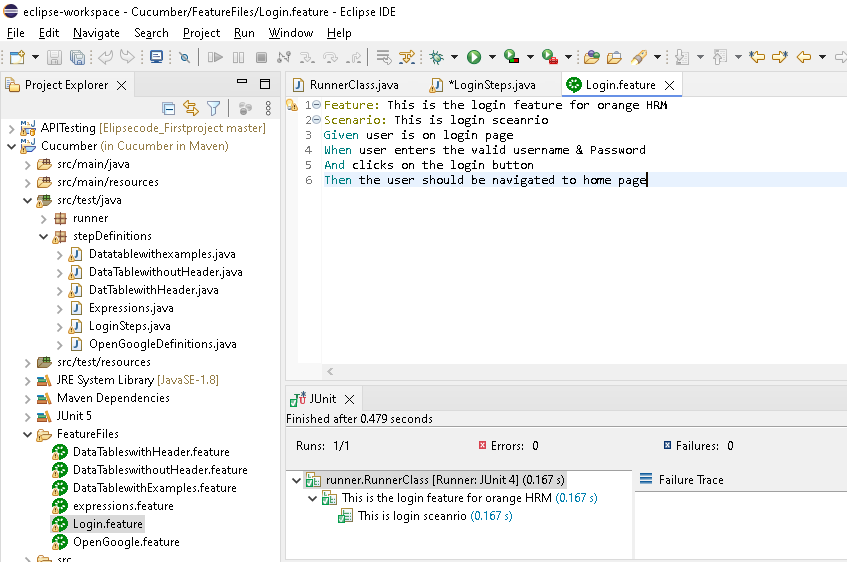


Only comparing done... it will not execute the application flow(like launching application etc)

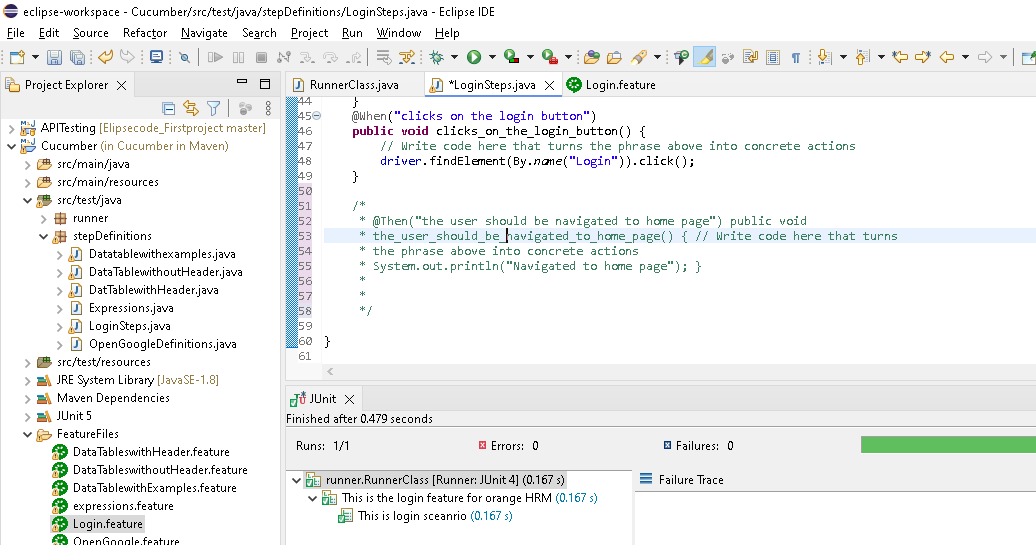
Under console..nothing present

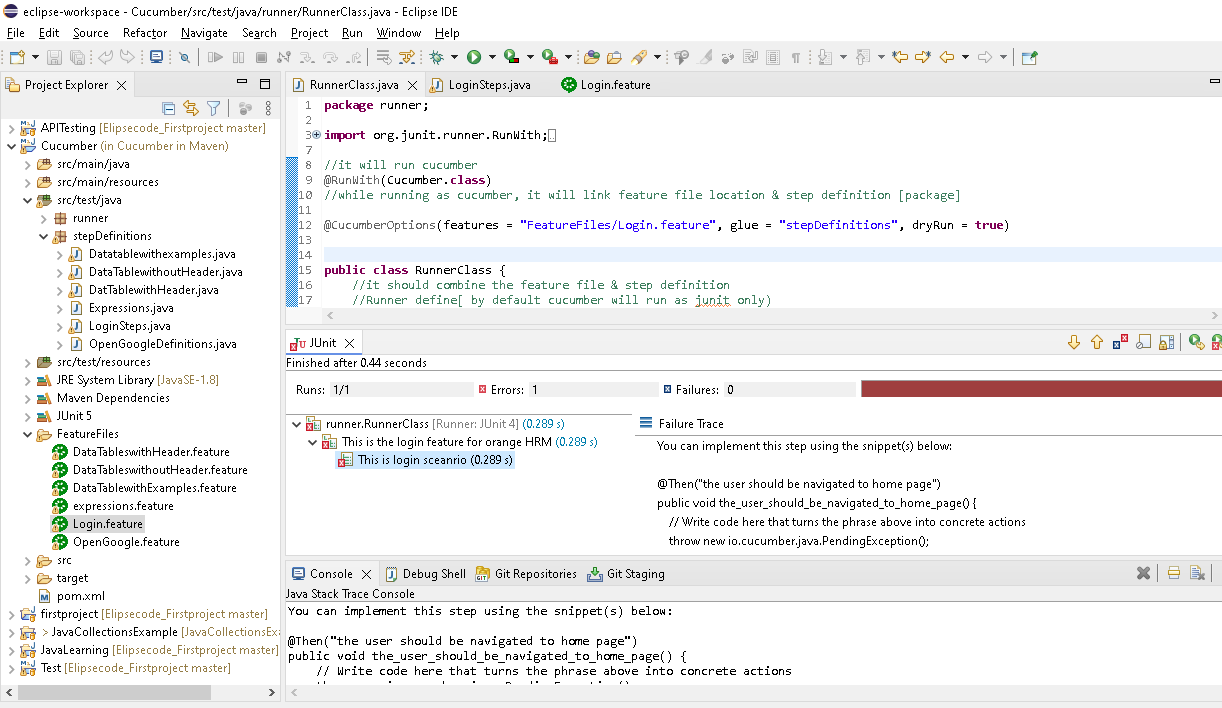


Remove one step definition & execute show the message



Then the user should be navigated to home page...comment in step definition



****

**Its showing error for missing step definitions**

**4.strict->Having Boolean value & it will execute the methods & parallel show the missing step after the execution**

For above example, here after the executions of first 3 steps , it will showing error for missing step definitions..

Note: currently strict option not available in cucumber options..Seems decommissioned

if strict is true, check for mapping with execution & throw the unimplemet step definition for missing steps & fail

if strict is false, check for mapping with execution & throw the unimplemet step definition for missing steps, but with pass..

**5.tags**

In some case, we need to run only specific test(.feature file ) instead of all

In some case, we need to run only specific scenario instead of all

In some case, we need to run only specific feature instead of all

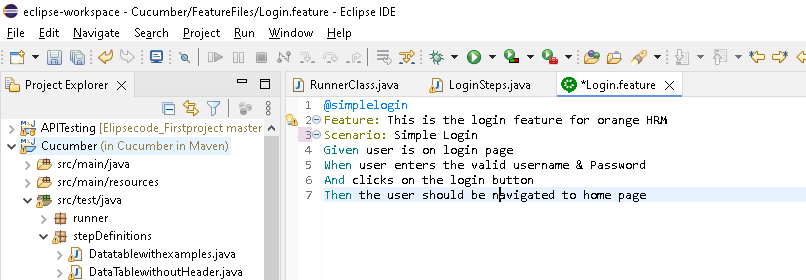
Tags start with “@”

In runner class, Under cucumber options tags = {“@tagname}

With cucumber we can use tags:

1. **Tags at feature file level**
2. **Tags at scenario/scenario outline level**
3. **Ignore the tests using tags**
4. **Club different tags and run**

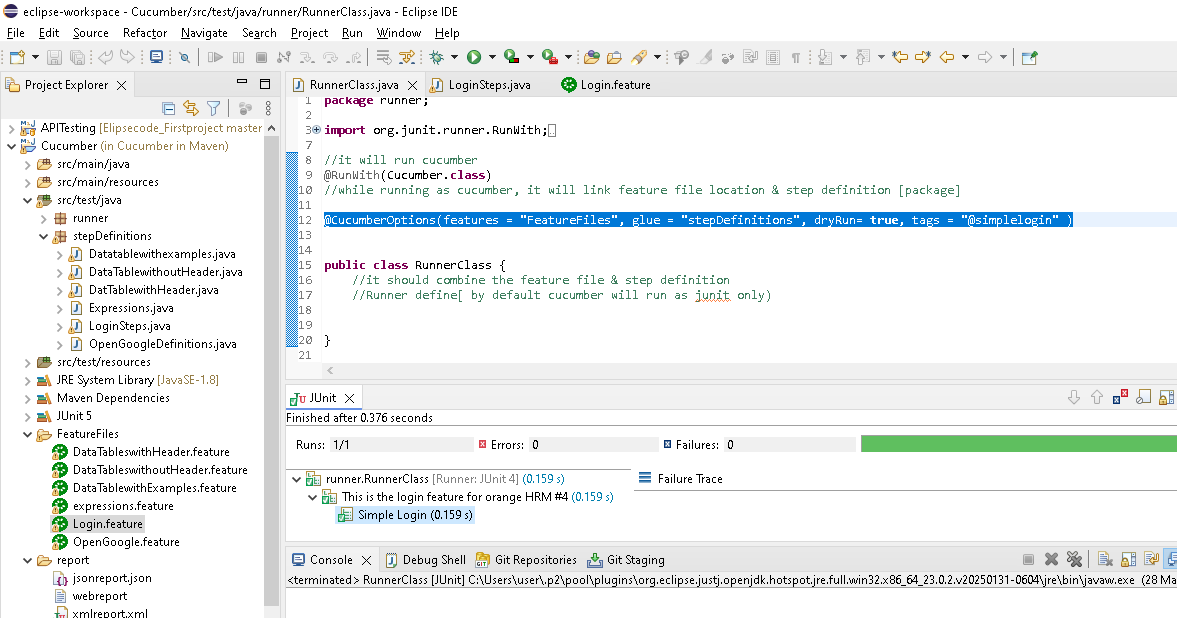
**1.** **Tags at feature file level...need to give @ with any tag name**

****

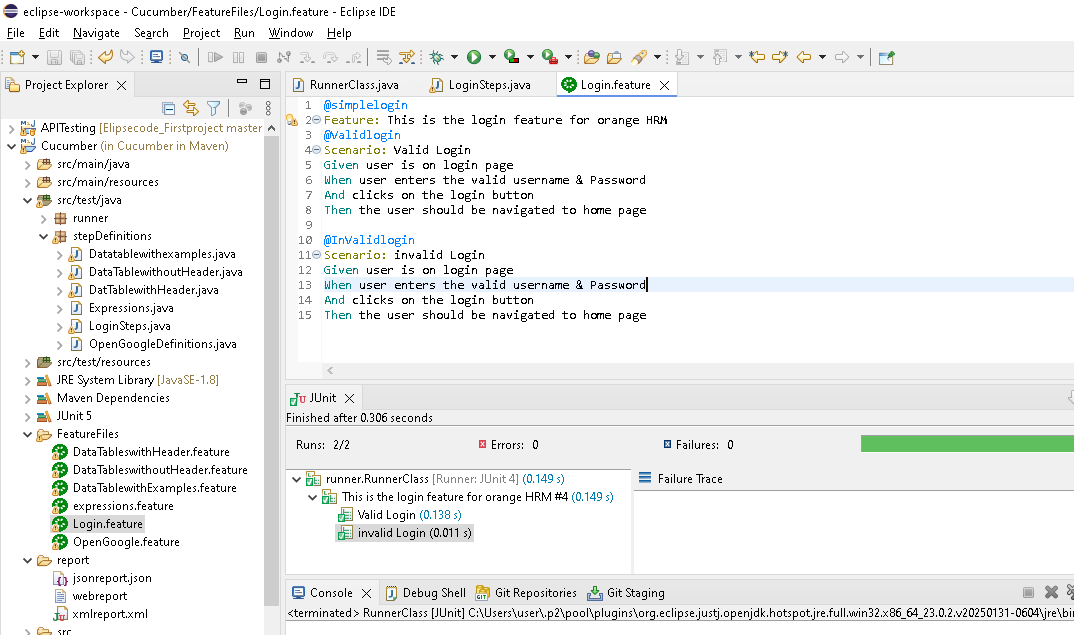
@CucumberOptions(features = "FeatureFiles", glue = "stepDefinitions", dryRun= **true**, tags = "@simplelogin" )

Here features = "FeatureFiles" ..under featurefile folder..which is having @simplelogin tag only will execute ere l .features files, it execute only @simplelogin scenarioad of all

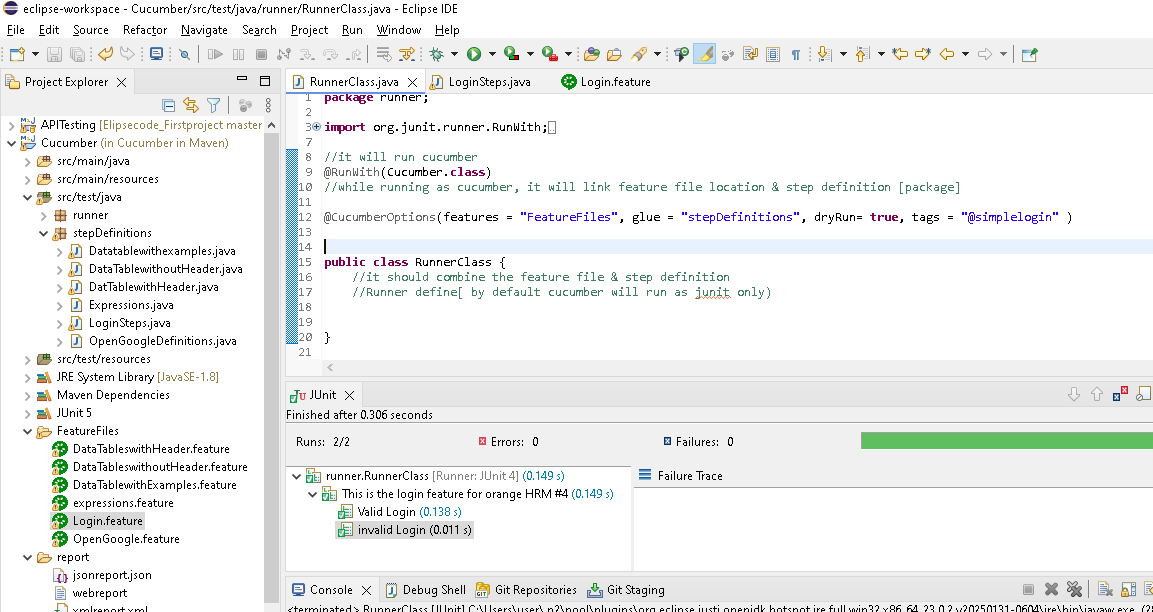
**Out of all .features files, it execute only @simplelogin scenario**

****

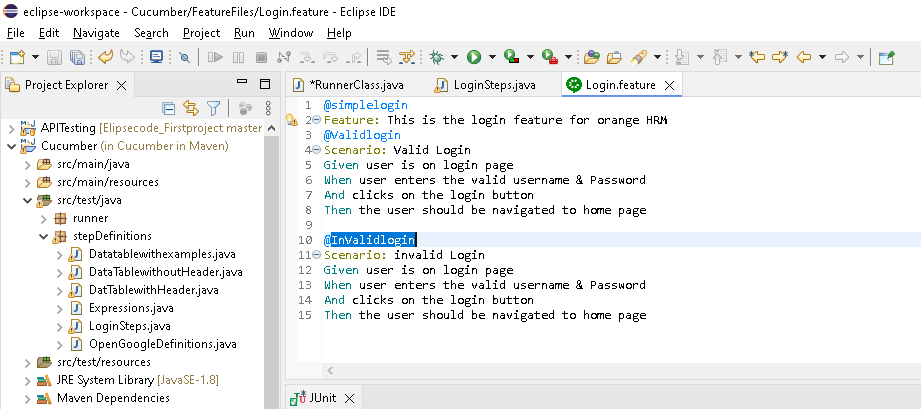
**With 2 scenario**

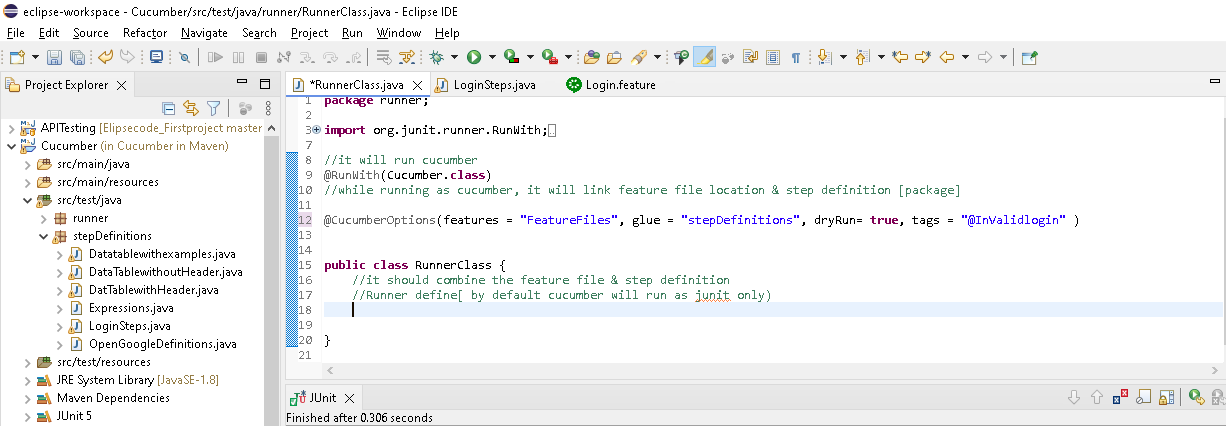
****

**As at feature level tag, it execute both valid login & invalid login sceanrio**

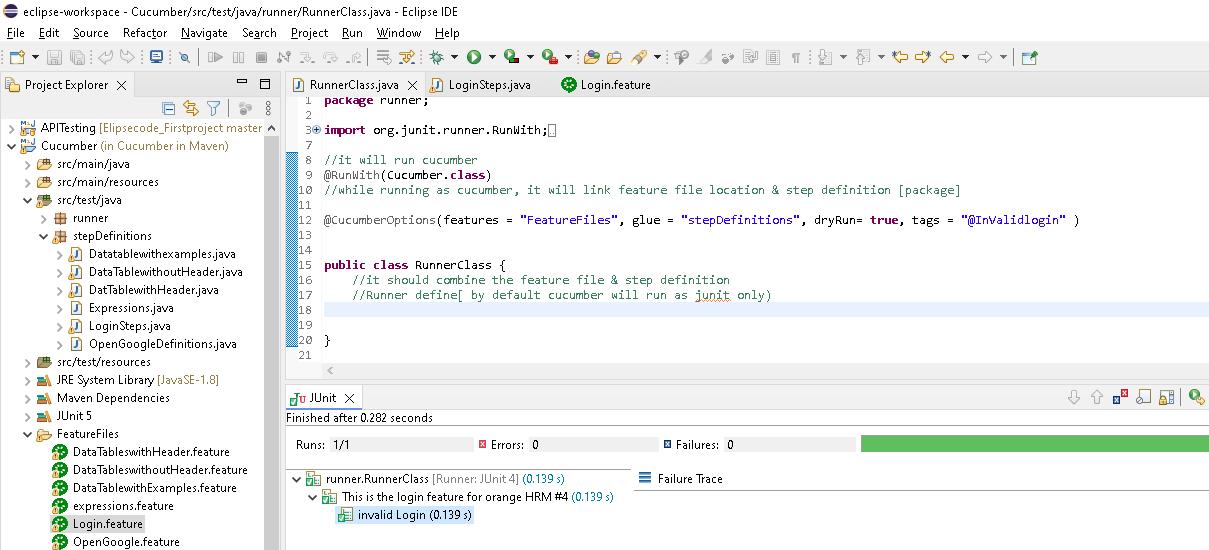
****

**2.Tags at scenario level**

****

****

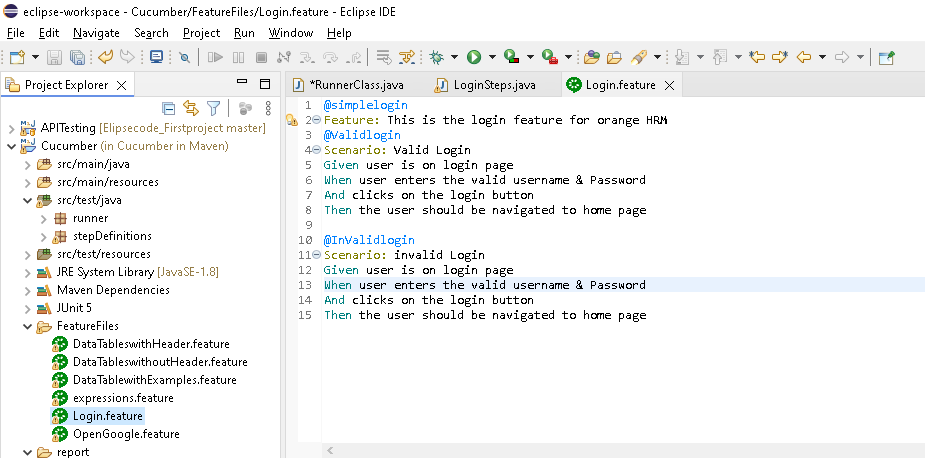
**Execute expected scenario only as mention in the tags**

****

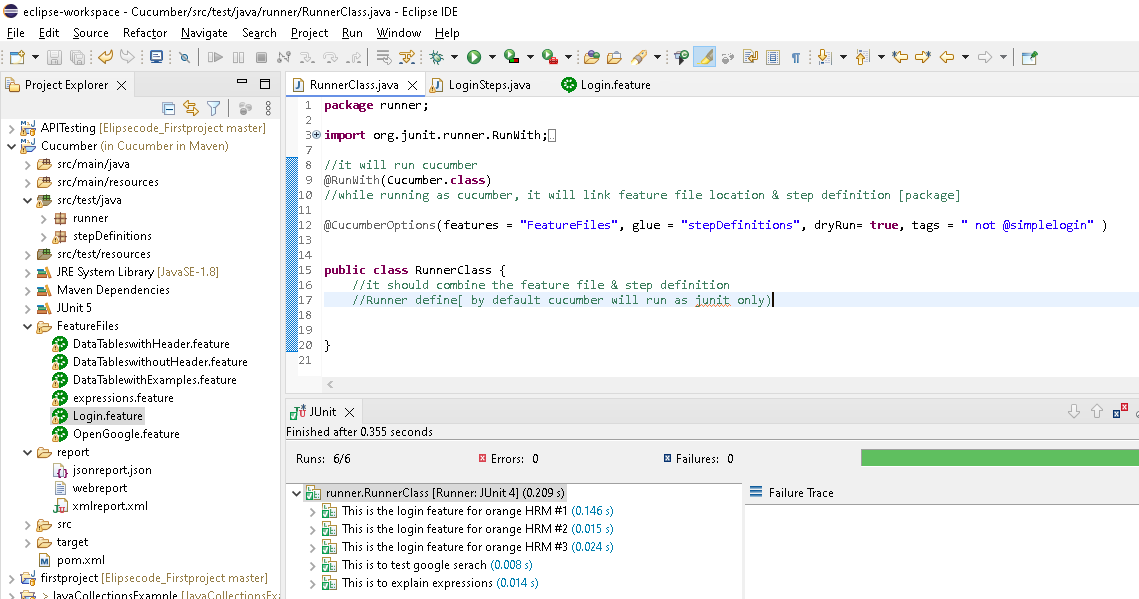
**3.Ignore the tests using tags ( by using “not”)**

@CucumberOptions(features = "FeatureFiles", glue = "stepDefinitions", dryRun= **true**, tags = " not @simplelogin" )

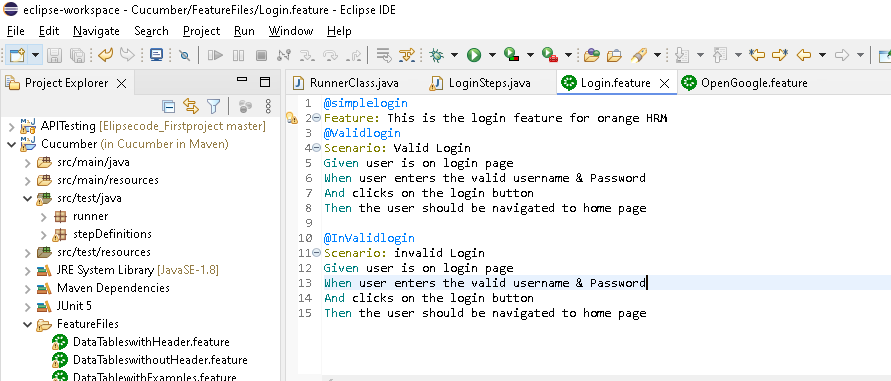
Here it wil execute all test except the @simplelogin test..only login feature file have this tag at feature level..that means except login.feature test..all remaining test will execute

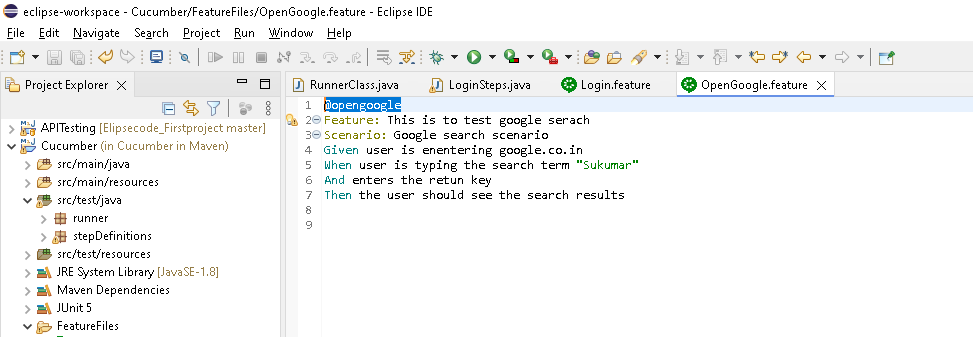


Out of 6 feature file, only 5 files executed..



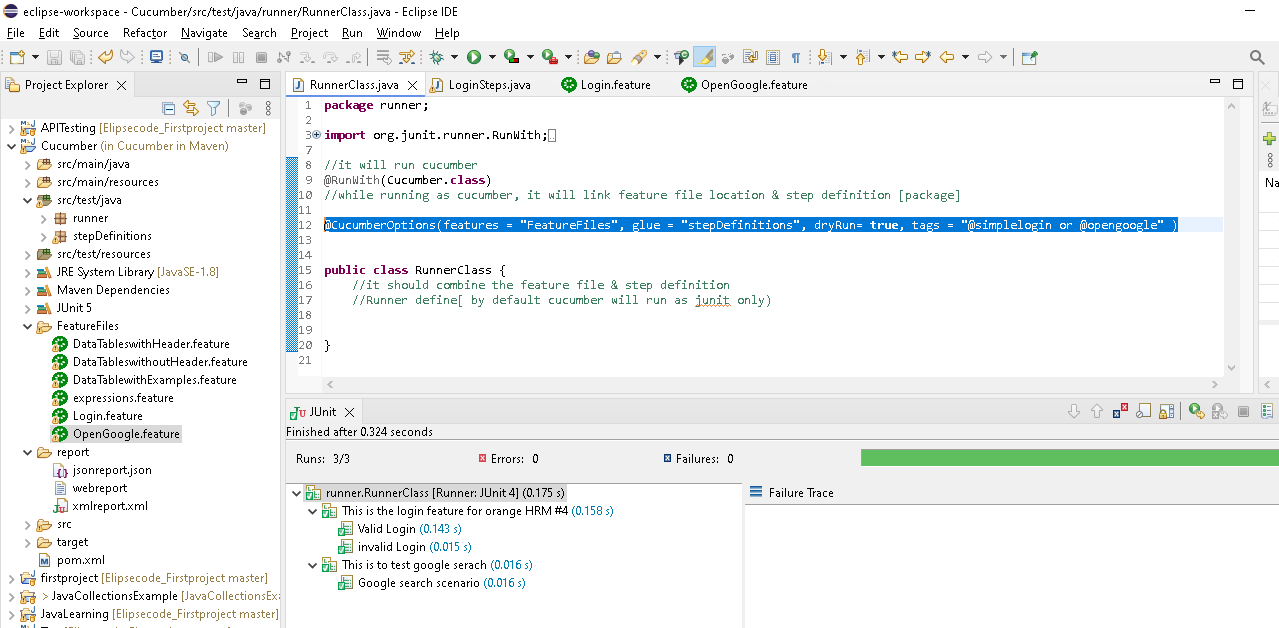
**4. Club different tags and run [ by using or , and]**

****

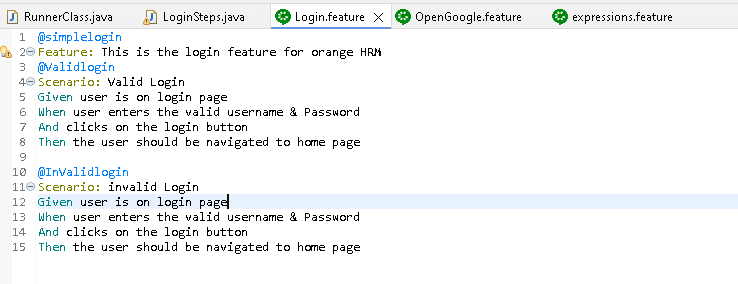
****

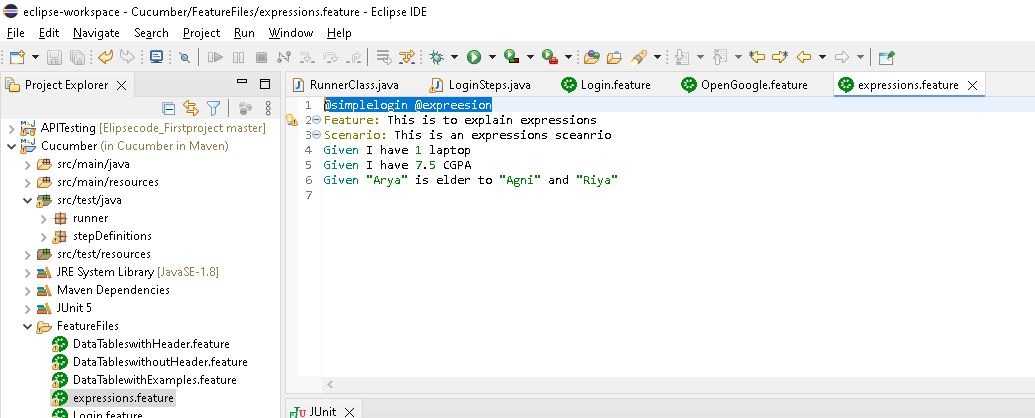
@CucumberOptions(features = "FeatureFiles", glue = "stepDefinitions", dryRun= **true**, tags = "@simplelogin or @opengoogle" )

Execute both simplelogin , opengoogle test

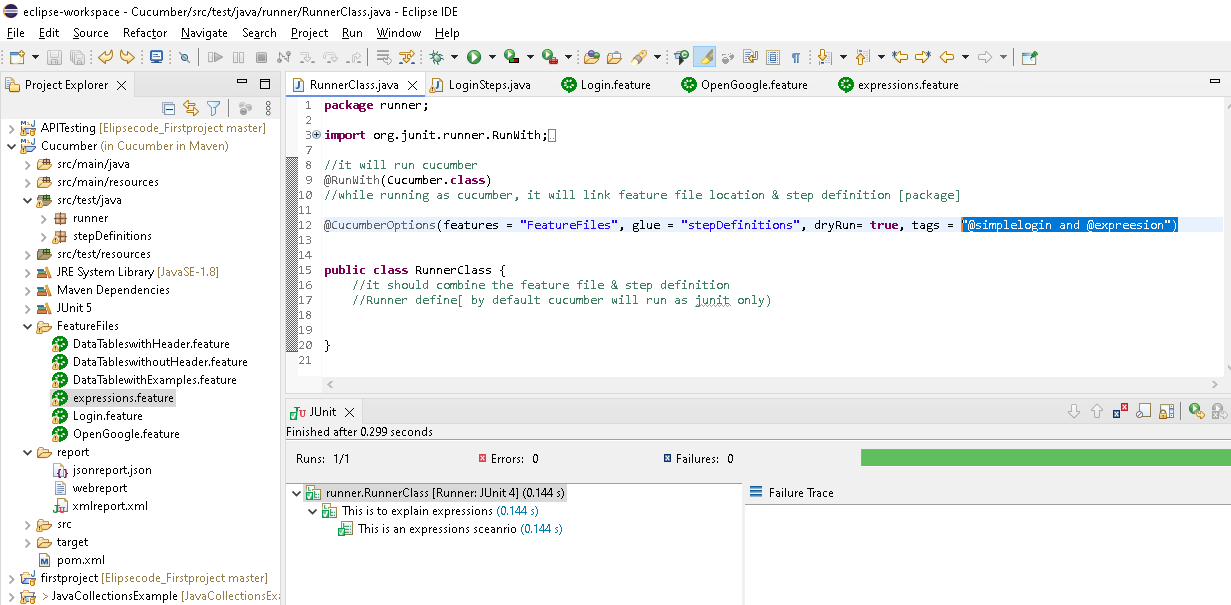
****

**Using and**

****

****

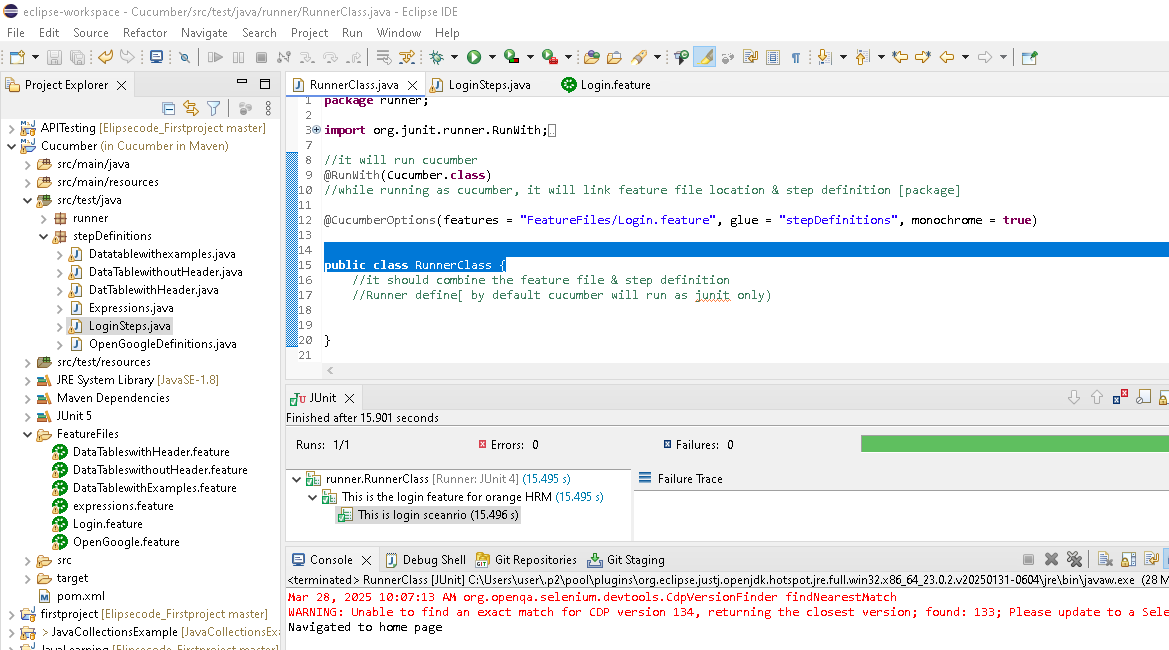
**Executed only one test**

****

**6.monochrome ->it have Boolean value**

It will convert/use the console message to the human readable format

In latest version , we are not seeing the message in console(scenario executed, time taken all)

****

**7.Format(plugin) -> used for reporting & we can generate report in format like“ html,jscon,xml “**

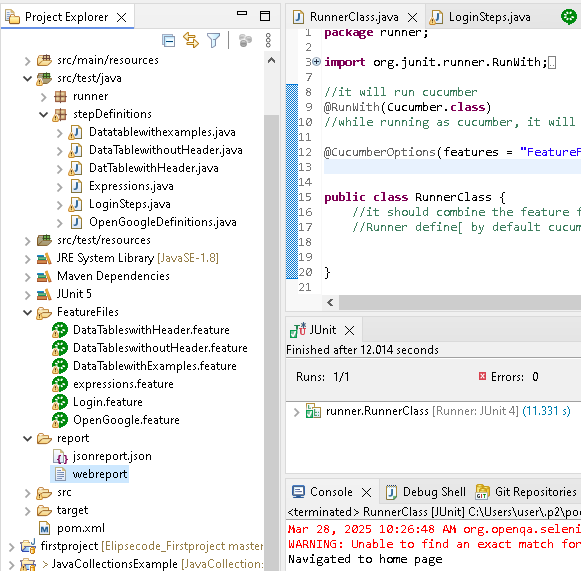
Format option decommissioned & now new version came is plugin..

@CucumberOptions(features = "FeatureFiles/Login.feature", glue = "stepDefinitions", plugin = {"html:report/webreport", "json:report/jsonreport.json" } )

It will create a folder “report” & subfolder “webreport”, in that both html & json format report present.

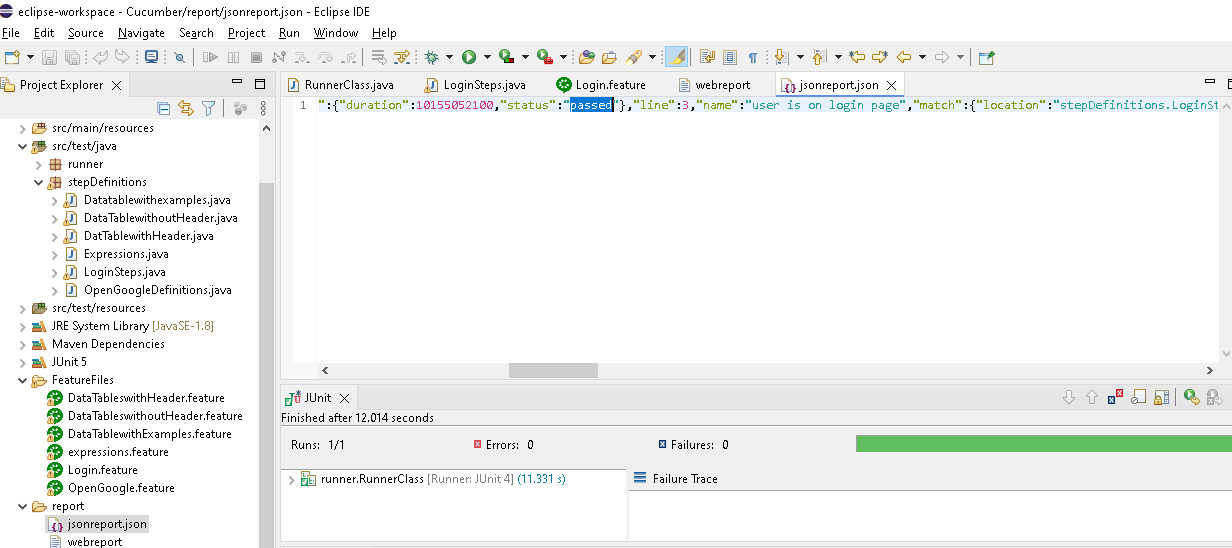
**note** for jscon only we need give .json(jsonreport.json) & html not needed as by default it will assign

after the execution, refresh the cucumber project & see the file under respective folder



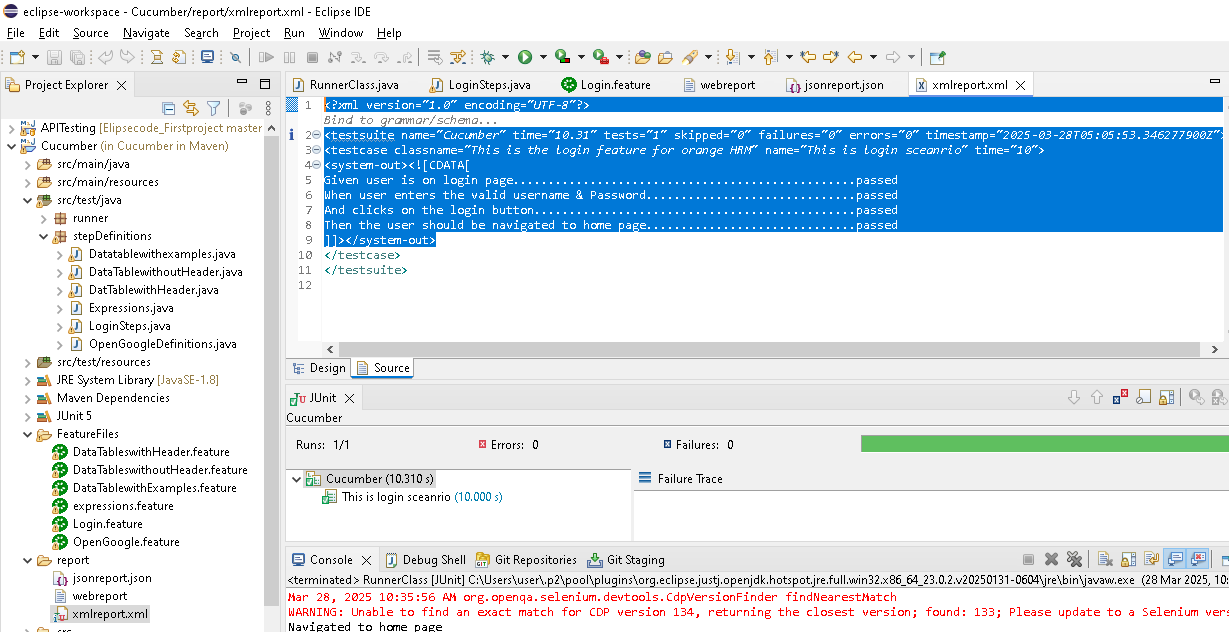
Double click & open

It will show status as pass for each step & time taken



Xml

@CucumberOptions(features = "FeatureFiles/Login.feature", glue = "stepDefinitions", plugin = {"html:report/webreport", "json:report/jsonreport.json", "junit:report/xmlreport.xml" } )



**Cucumber Hooks**

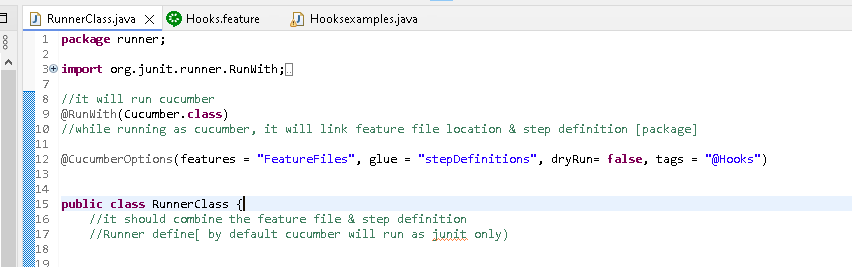
Used to set things up before/after doing something’s. [eg first application login[before], do some operations & then application logout[After]

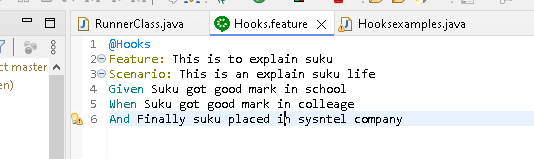
We have only 2 annotations

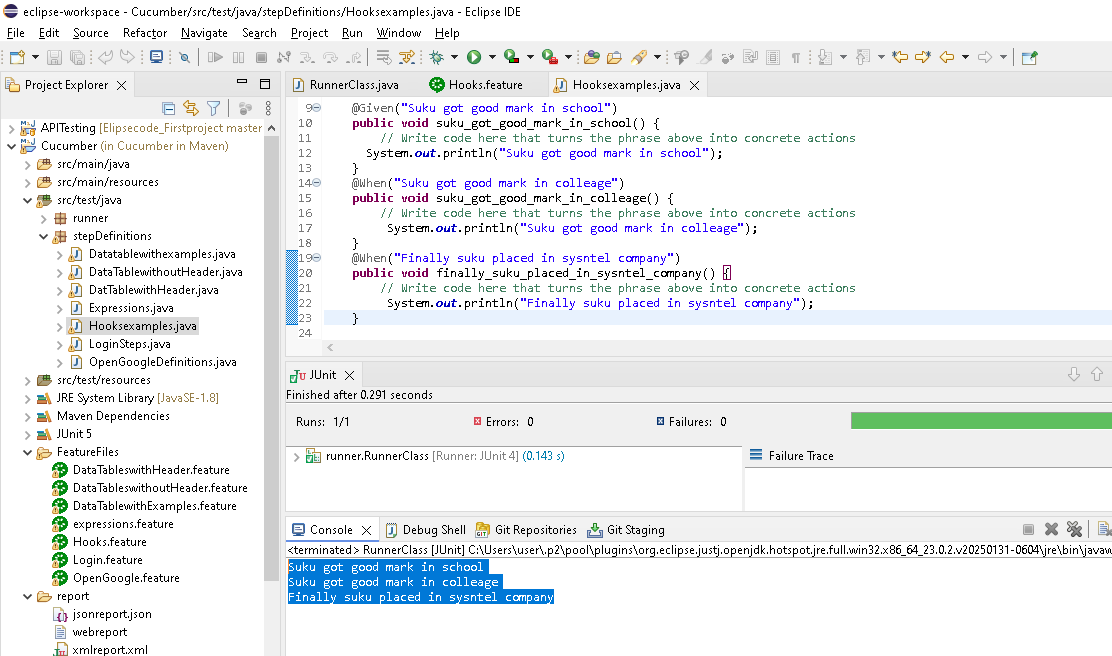
@Before, @After ->Cucumber annotations not junit..junit also have before & after..so we need import cucumber files

**import** io.cucumber.java.After;

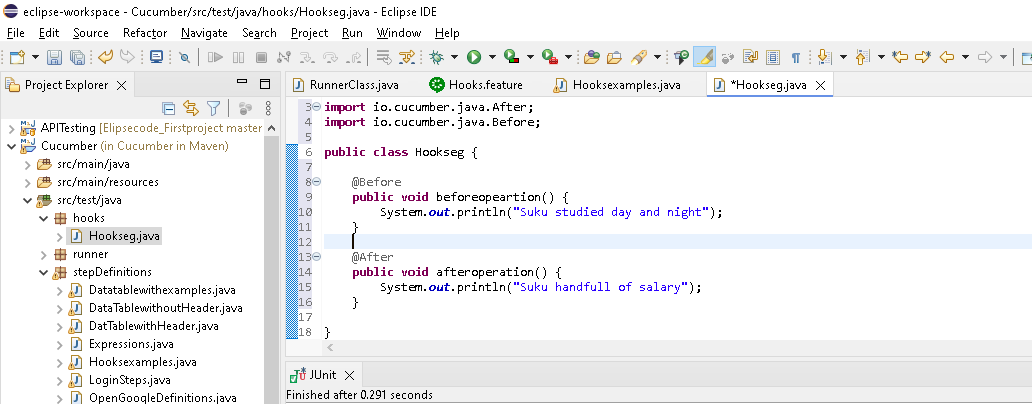
**import** io.cucumber.java.Before;







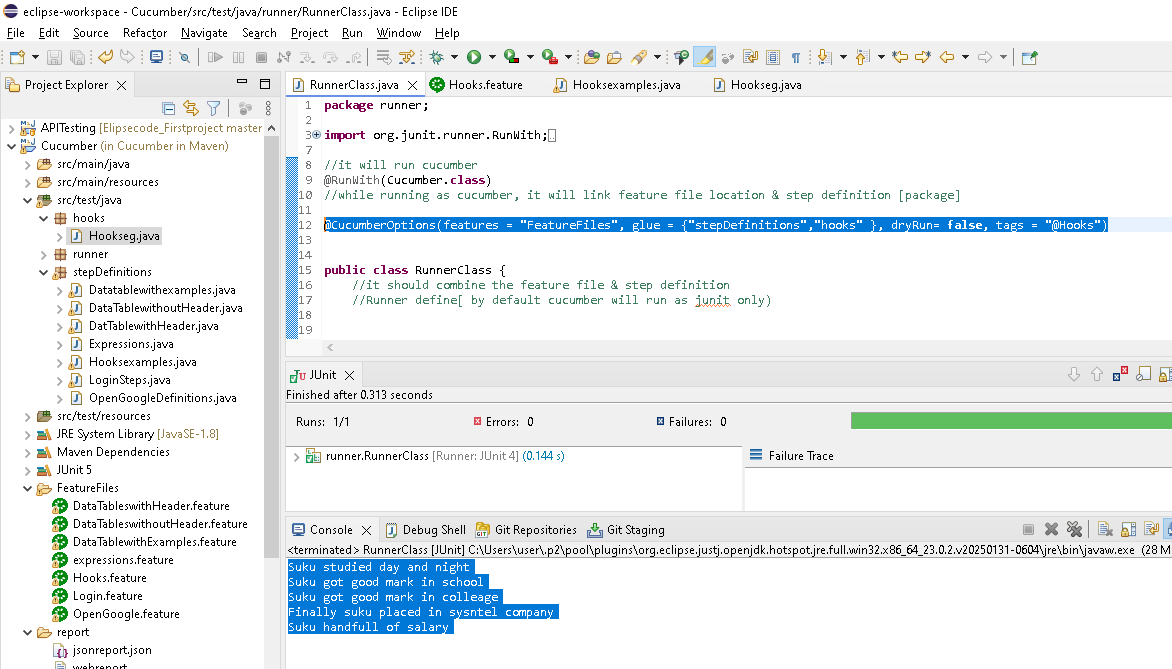
Create new package & create method with after and before annotations



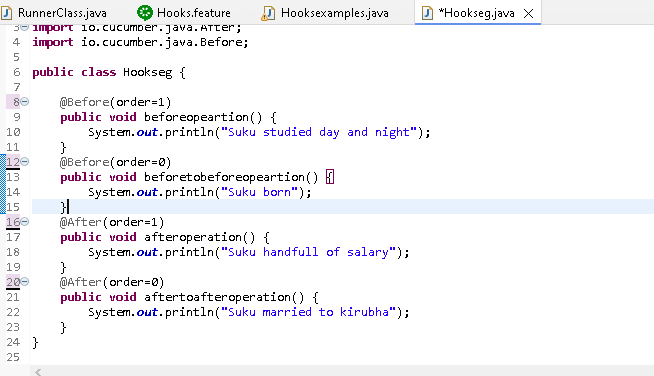
Go to runner class & execute as junit test

Note: in glue need to include hooks package also..before & after method present in that package only

@CucumberOptions(features = "FeatureFiles", glue = {"stepDefinitions","hooks" }, dryRun= **false**, tags = "@Hooks")



Now create 2 method with same annorations..in that need to mention the order which one need to execute first



Note: order be 0,1,2 etc

For before we need to maintain increment order (0,1,2..)

For after we need to maintain decrement order(2,1,0)



**Tagged Hooks**

Above we have read tags & hooks concept separately..here combine both

Tag + hooks ->Tagged hooks

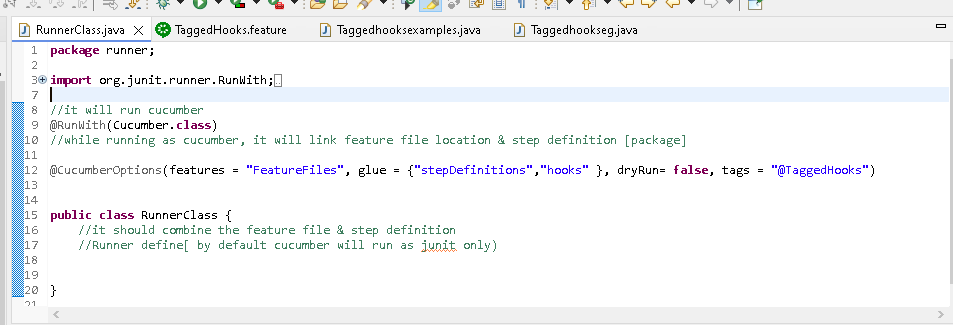
**Problem**: there will be different prerequisites before for different scenarios..what to do?

**Solutions**: customised tagged hooks (Tag + hooks) for scenarios

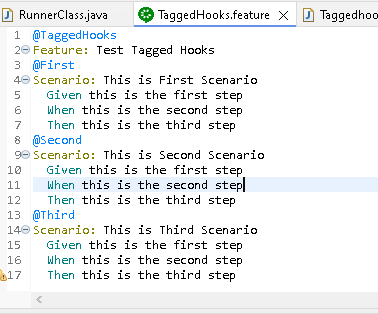
<https://toolsqa.com/cucumber/tagged-hooks-in-cucumber/>

code copied from above path

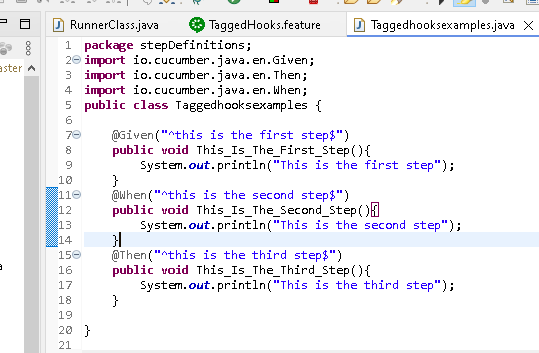
**runner class**



**Feature file**

****

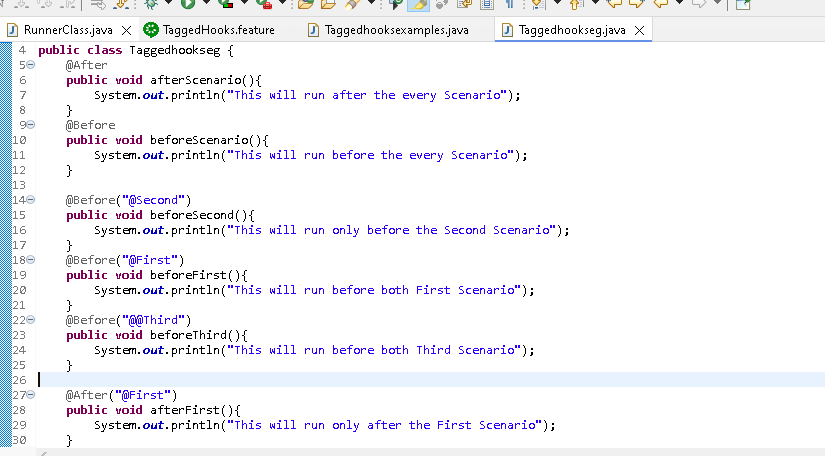
**Stepdefinatin:**

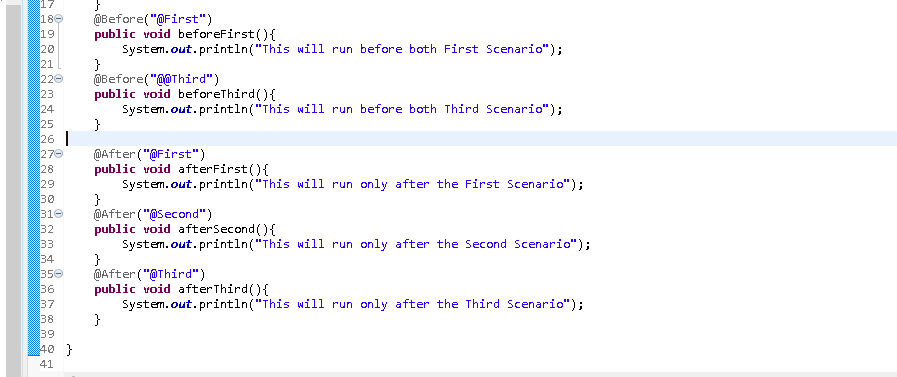
****

**Before & after conditions**

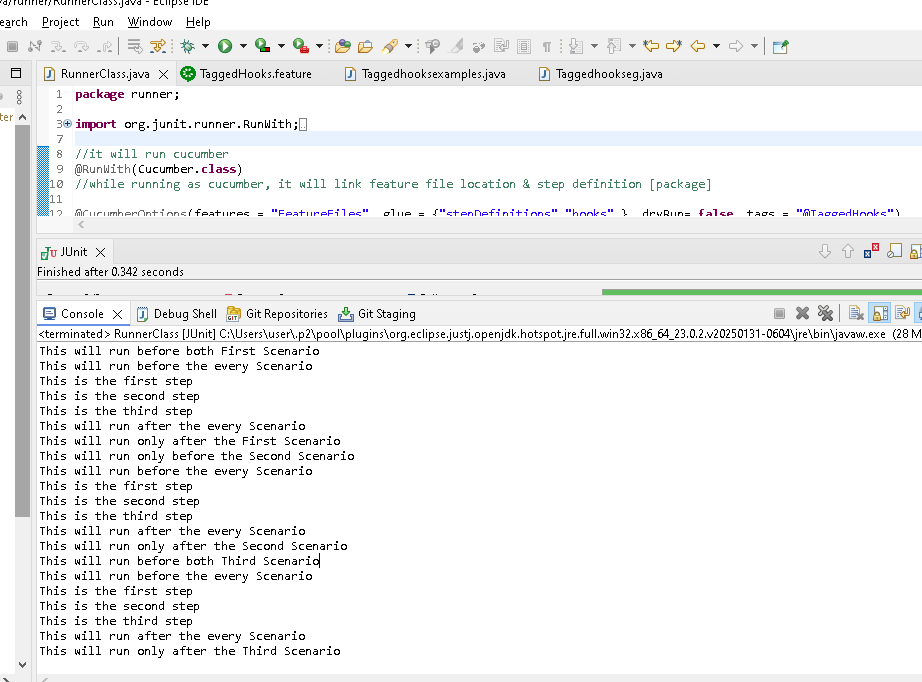
**Below is a tagged hooks[before,after ->Hooks & @secons->tags]**

@Before("@Second")

****

****

**Result:**

****

**Background in cucumber**

Its is to represent a collective list of pre conditions for all the scenarios

Its allow you to add some context(info) to the scenarios in the feature file [should have only one background in a feature file for all scenario...(like before & only have before..after not there)

It can contain one or more given steps

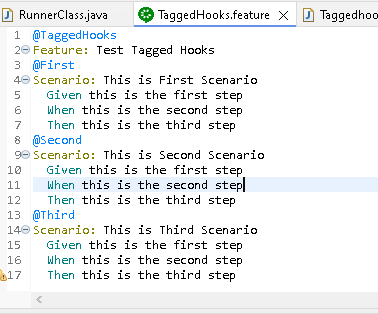
It performs much like hooks(@Before) that is runs before each scenario..Whereas hooks have both before & after

We can have multiple hooks (before,after) for a scenario but in case of background,

We can only have one set of background steps per feature file.In case of multiple background needed, we have write separet featue file

**Non technical person by seeing the fearture file , cant understand the what exeute before & after each scenarios..we overcome this business requirement background feature came**

**Eg**

****

**With background – user can understand what executed before each scenarios with given steps as below\**

**Note: before each scenario, background given will be execute first & then scenario steps**

@background

Feature: To explain the background

Background: Student complete school education

Given Student finish high school

Given Student finish higher school

Scenario: To study medicine

Given Student applied for medical

When Student clear the exam

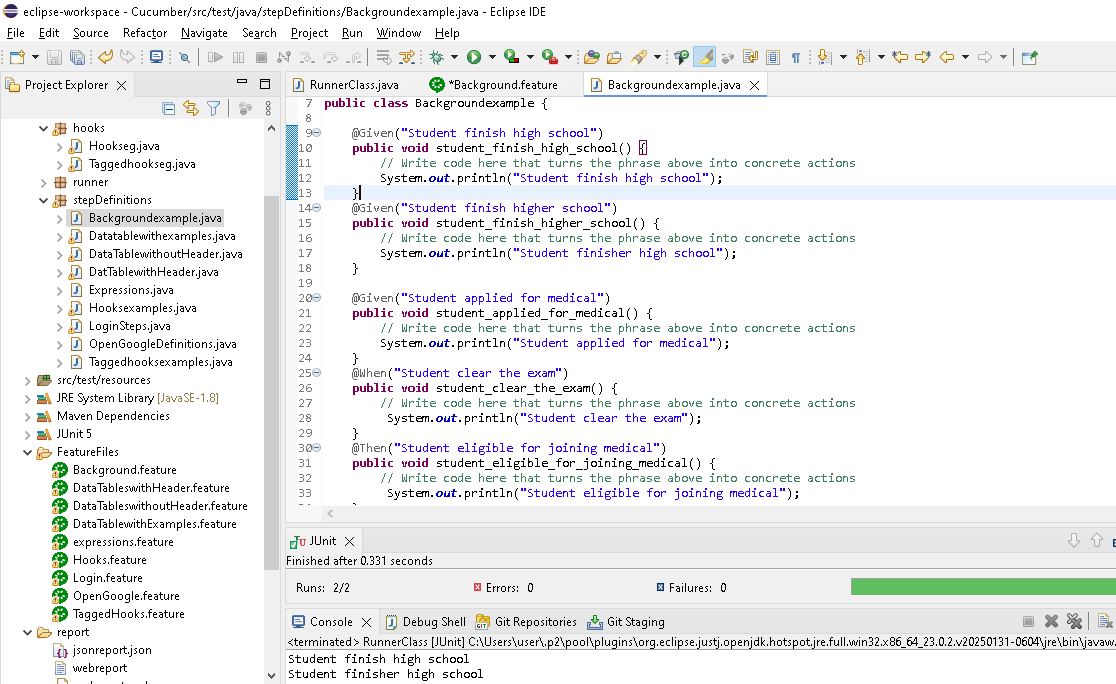
Then Student eligible for joining medical

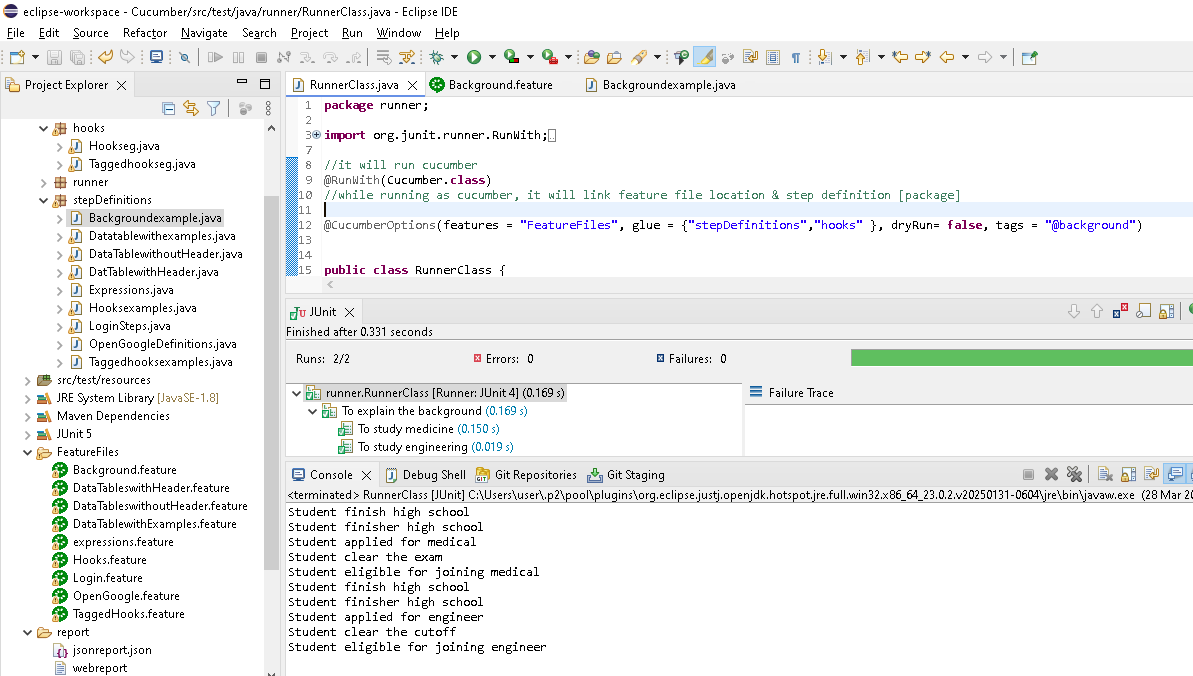
Scenario: To study engineering

Given Student applied for engineer

When Student clear the cutoff

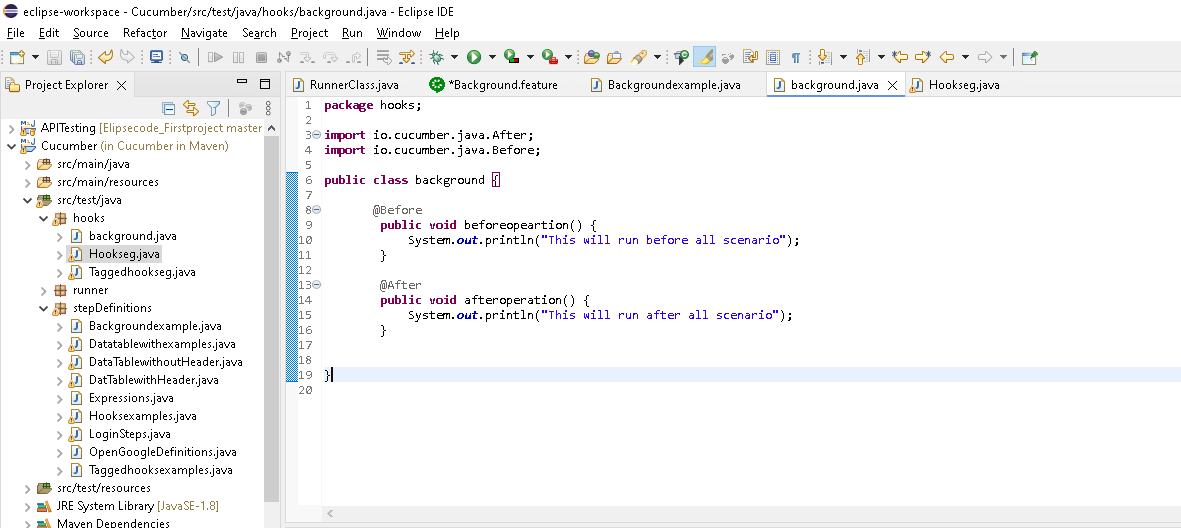
Then Student eligible for joining engineer

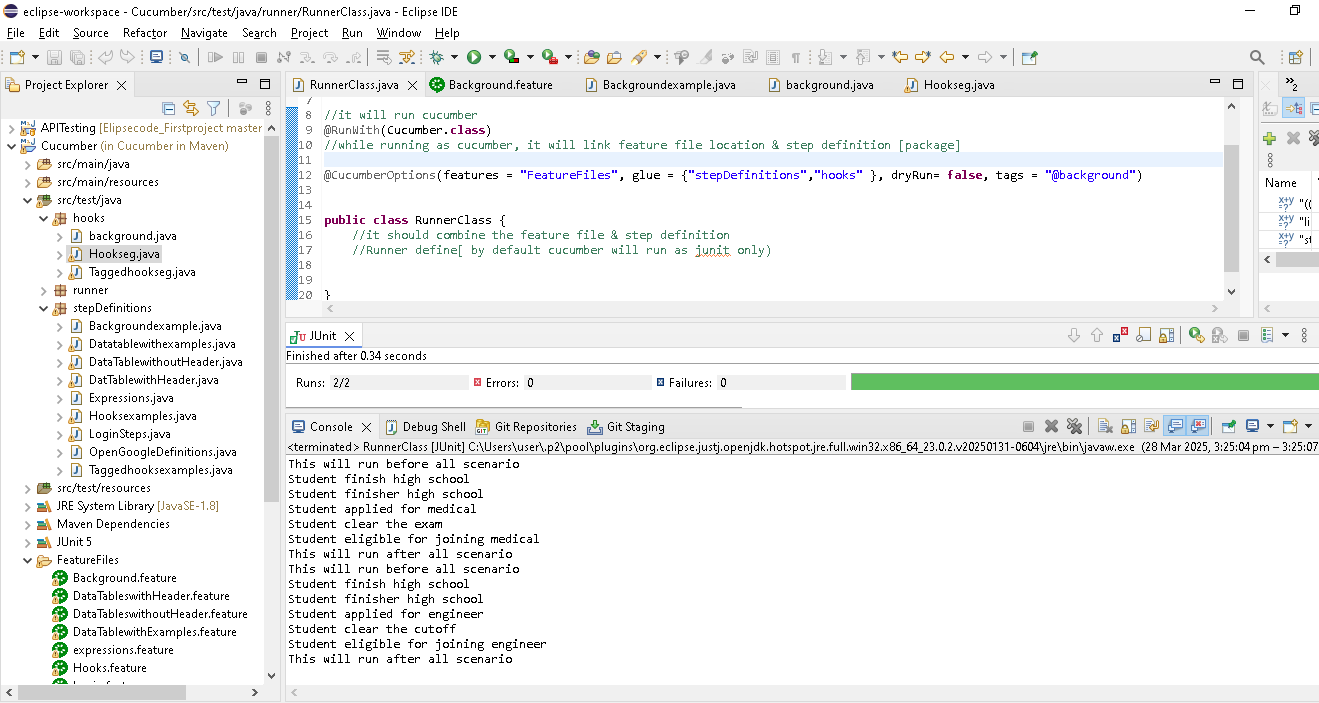


****

**What if i have both hooks and background?**

**Hooks executed first & then background**

****

****