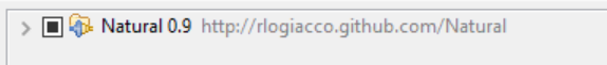
Cucumber 🡪 is an pre-defined framework and based on BDD 🡪 behavior driven development

1. A BDD will have three major things
2. **Feature file** 🡪 will have steps and definition in simple and plain english
3. **Step definition file** 🡪 “a” will be implemented here and the step definition will be implemented using Java and selenium
4. **Test Runner** where test cases will be executed using Junit class
5. Add Cucumber and natural in eclipse form market place or even using “install new software”. Refer “https://www.toolsqa.com/cucumber/install-cucumber-eclipse-plugin/#:~:text=Steps%20to%20follow%3A,Click%20OK.”



1. Then create maven project and add dependencies
2. **Junit**
3. **Cucumber-java**
4. **Cucumber-testng (for testng) or cucumber-junit(for junit)**

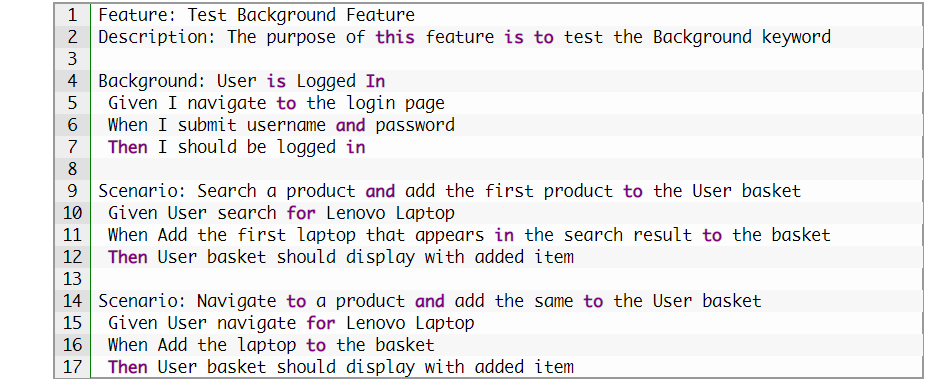
**// Note: junit can be converted to testng as well**

1. Cucumber-core (1.2.5)
2. Cucumber-java (1.2.5)
3. Cucumber-junit (1.2.5)
4. Cucumber-jvm-deps (1.0.5)
5. Cucumber-reporting (3.16.0)
6. Gherkin (2.12.2)
7. Mockito-all (1.10.19)
8. Cobertura(2.1.1)
9. Selenium-java (3.10.0)
10. Guava (25.0-jre)
11. Tools (1.6) (com.sum is the groupid). Note: After scope tag, please add

<systempath> location of tools.jar in lib folder of java</systempath> i.e.,

<systempath> C:\Program Files\Java\jdk1.8.0\_181\lib</systempath>

1. Create 3 packages for 3 components mentioned in point 1 say “Feature File”, “Step Definition” and “Test Runner”
2. Feature file can be created using New 🡪 Others / File 🡪 save it with “.feature” extension
3. Inside feature file mention the feature name as Feature:, Scenario as Scenario:, Given, When, Then and And gherkin keywords (the keywords will display in different color)
4. Also feature file will have Background keyword (to group steps that are common for all say login into a webpage) and scenario outline (used to run same scenario but with diff data)





1. Create a class in step definition and create one method for each keyword say you have a feature file as
2. Given Launch bank website
3. When User Enters valid credentials and on click continue
4. Then application should display home page
5. And with success message

I .the step definition should be mapped by annotation for each method

@Given(“^Launch bank website $”)

Public void launchurl(){

}

@When(“^User Enters valid credentials and on click continue$”)

Public void enterCredential(){

}

@Then(“^application should display home page$”)

Public void checkhomepage(){

}

@And(“^with success message$”)

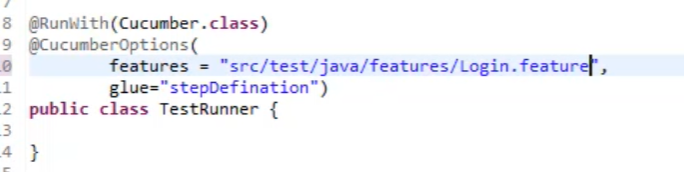
Public void successmsg(){

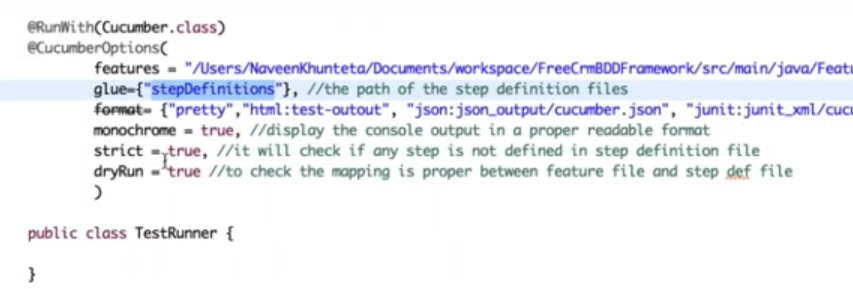
}

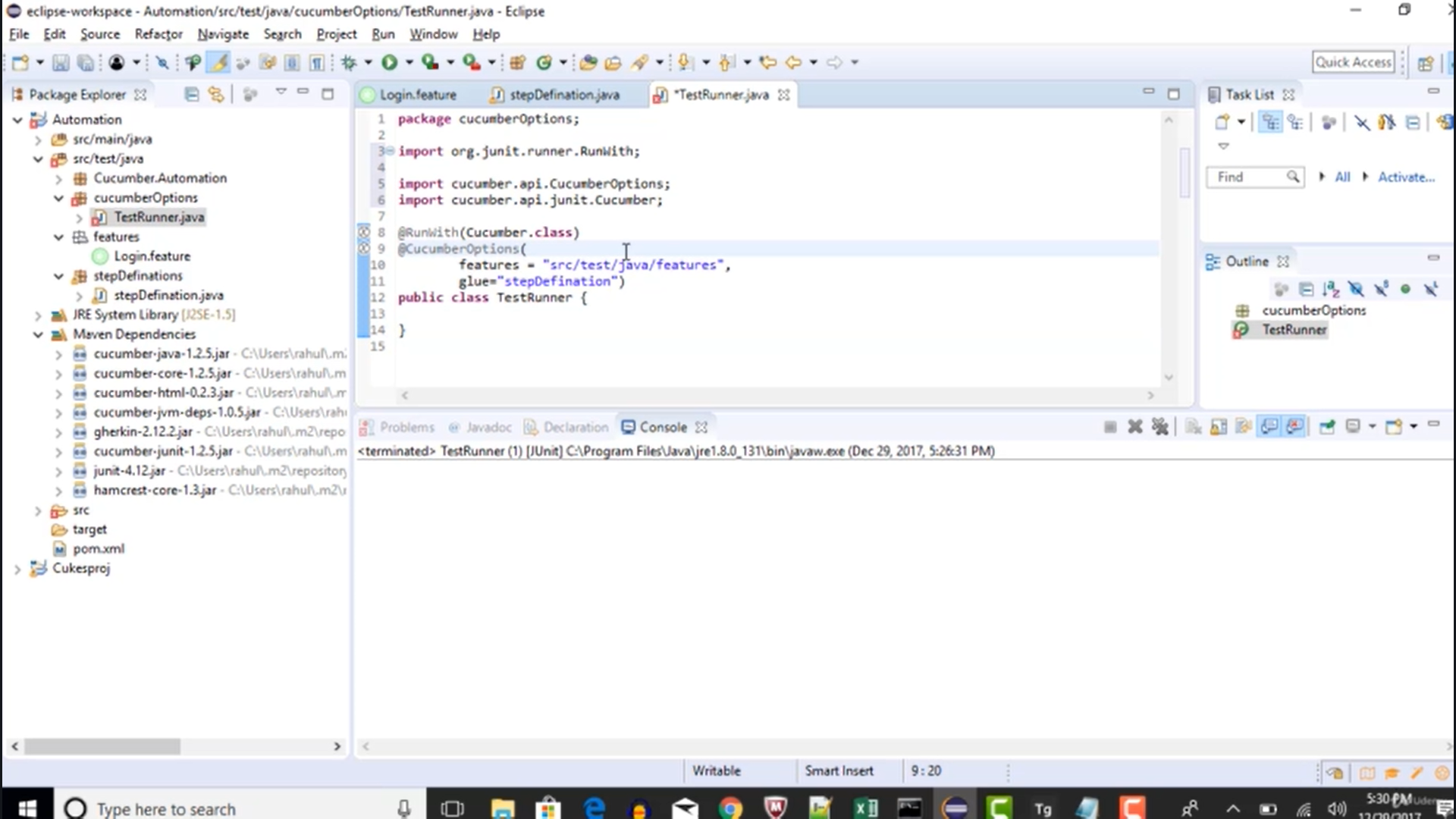
1. So this is how the mapping is done between feature file and the step definition
2. Also, you can use “Tidy Gherkin” chrome app extn, where in if you paste the feature file content, it will get us the “Step definition” content as mentioned above.

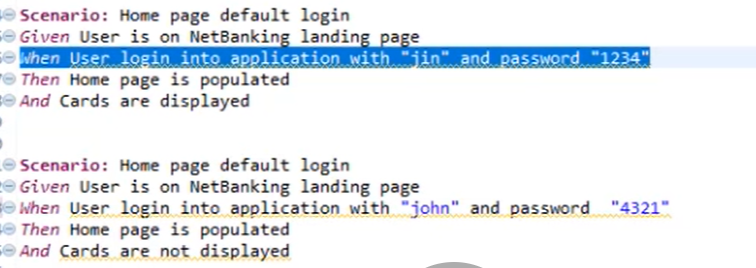
**Note: When tidy gherkin is not available, we can simply run the test runner file without doing the mapping between step definition and feature file, the console window will give us the list of methods as above**

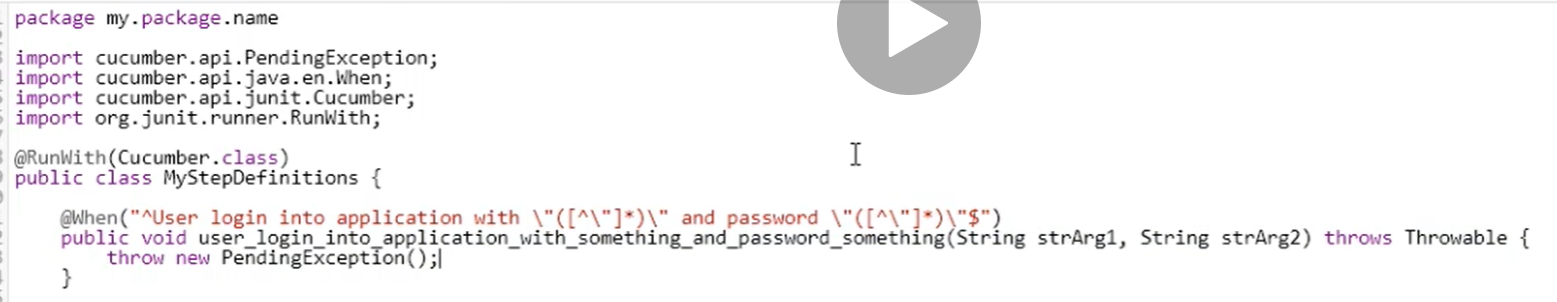
1. Now, in Test runner, create a class to execute the test case with two annotation to inform the test runner that where it need to get the feature file and respective step definition
2. To run a single feature file





1. To run all feature file, then use
2. When two gherkin line has same sentences with different parameters say two scenarios, in one login with “Ragav” and other Login with “Sharanya” then use regular expression as below….. Note: feature file with double quotes and step definition with regex



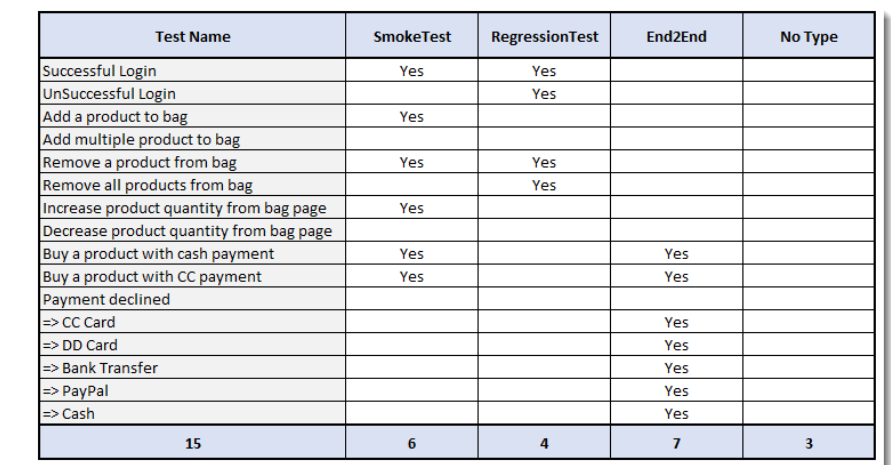


1. If tidy gherkin is not available, then we can simply run the test case without implementing step definition and eclipse console will give the exact things as same as step definition and copy paste and use the same
2. To handle the step definition when we deal with same function but with different variables use the parameters i.e., same method will be used by cucumber without user intervention and passes different parameters for every test step
3. By default, cucumber is based on Junit test case and if your project is created based out of TestNG, then just adding the class name (test runner class) in testng.xml will not work. So to make it work, we need to do
4. Add cucumber-testng jars in pom.xml
5. Convert the test runner class from junit to testng by
6. Comment out the @RunWith(Cucumber.class) annotation
7. Make the test runner class to extend **AbstractTestNGCucumberTests**

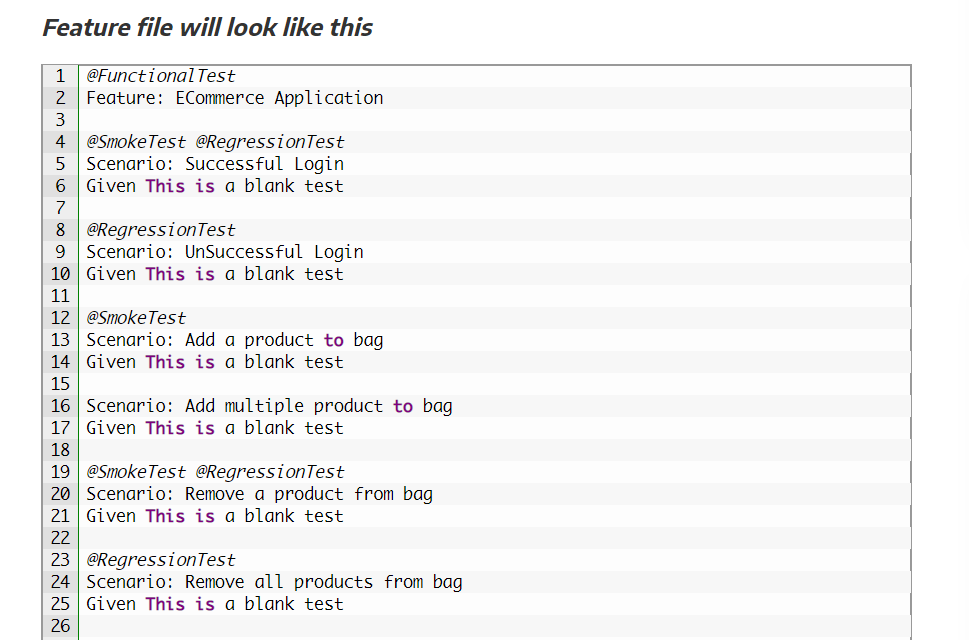
***Introduction of tags in cucumber***

There are scenarios where you need to run a particular test cases for multiple time says once for smoke, then for regression and then for E2E. To handle such scenarios, we can use the @tag as below and the same should be mentioned in test runner.

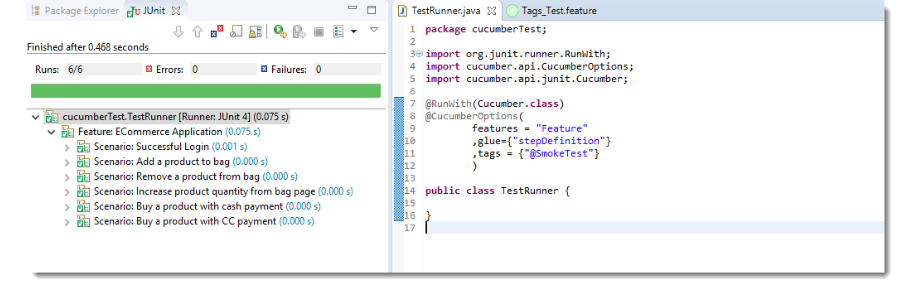
**Common scenarios e.g.**

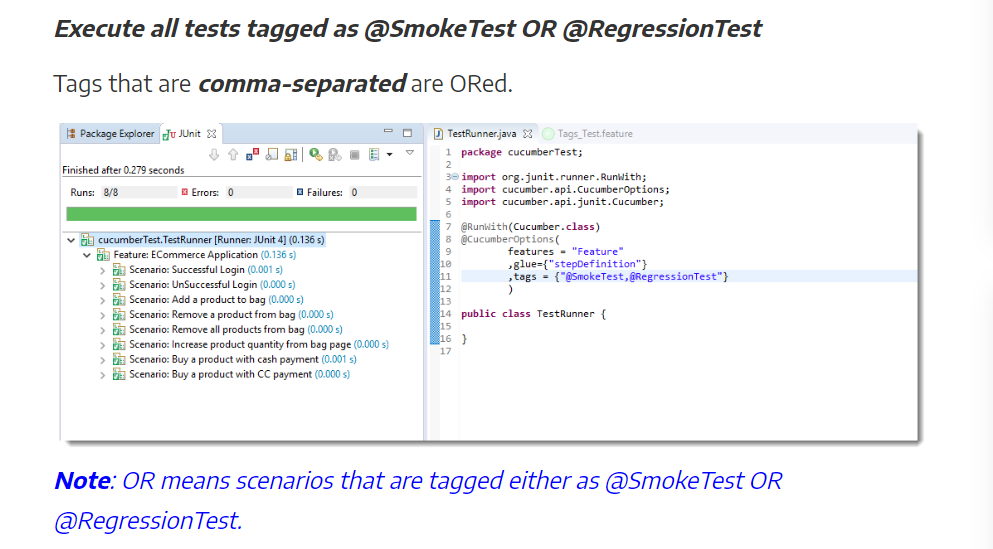


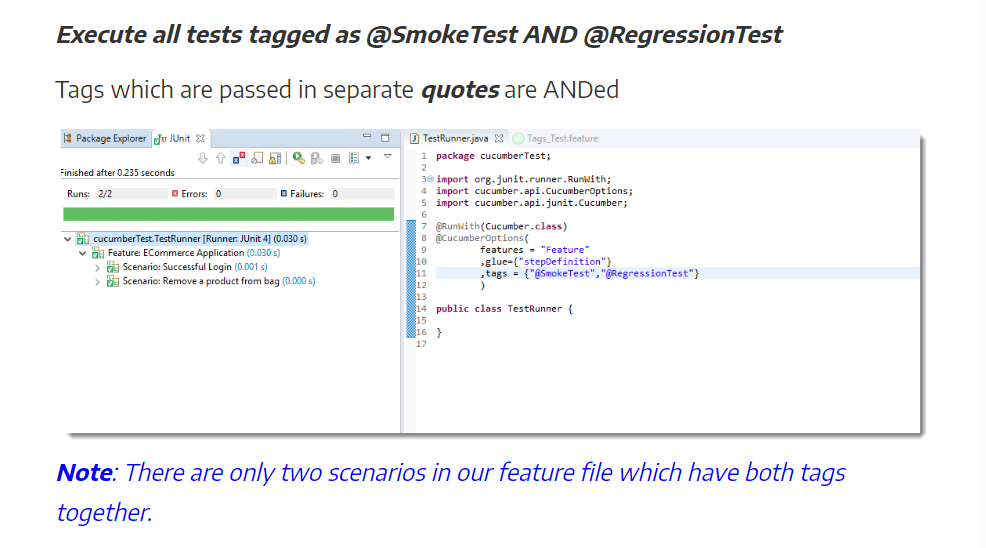
To tackle using @ tags



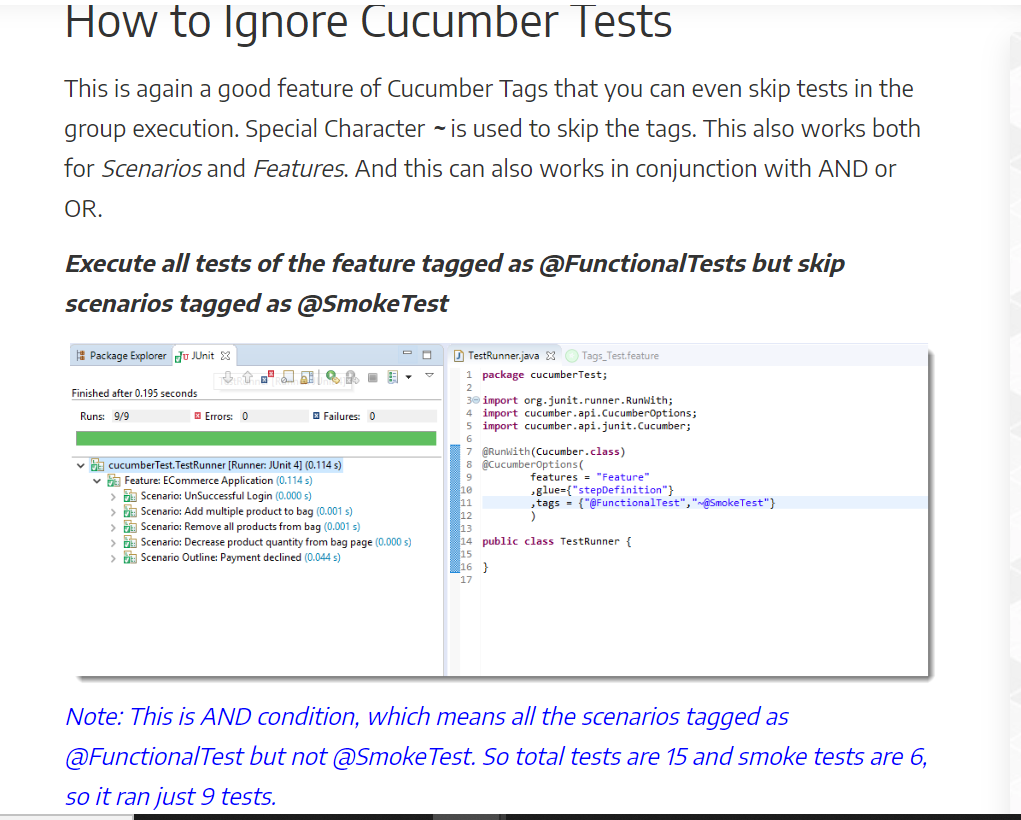
While running in test runner mention the tags as

Sometimes we need to run different combinations like smoke or regression so use



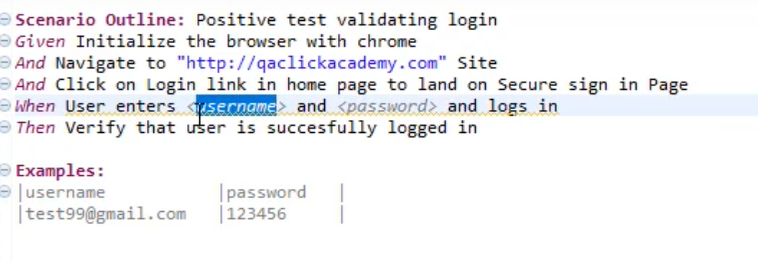
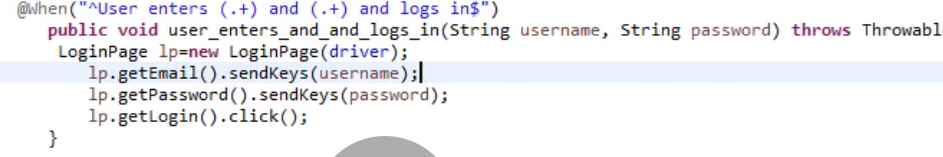


To ignore a test case use ~ symbol in tags as



**Scenario outline eg**

**===================**

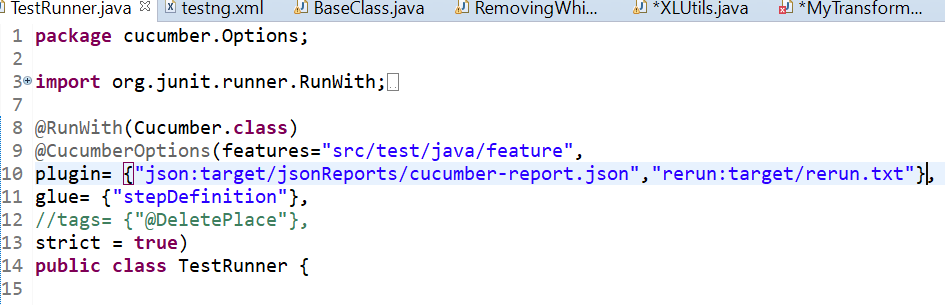
**How to rerun the failed test cases using cucumber:**

**===============================================**

In order to rerun the failed test cases in cucumber we need to make use of the “plugin” keyword in TestRunner class.

1. We need to have 2 TestRunner class named “TestRunner” and “TestRunnerRerun”.
2. In “TestRunner” class add the plugin value along with other parameter separated by comma like “json:target/jsonReports/cucumber-report.json, rerun:target/rerun.txt”. So that all failed test scenarios will get stored in target folder 🡪rerun.txt file with info like scenario number as below
3. In TestRunnerRerun modify the features value to (“@target/rerun.txt”) with plugin value to create report for failed scenarios
4. We can run the project using maven in command prompt like “mvn clean install” or through jenkins, so that it will run both the first iteration as well as the failed test cases without user intervention

Add plugin value rerun:target/rerun.txt as below in TestRunner class for S.No1



Update the TestRunnerRerun as below and you can change your plugin value as per your standards if in case you are using diff format for report generation.

A screenshot of a computer program

Description automatically generated

Note: plugin attribute is used in cucumber option to generate a html report with the help of “verify” command from maven.

Command to execute to get the report – mvn test verify

To get reports – copy paste the code from <https://github.com/damianszczepanik/maven-cucumber-reporting> and update the version as given in the web page.

Attribute to add in @CucumberOptions

**Plugin={“json:target/jsonReports/cucumber-report.json”**

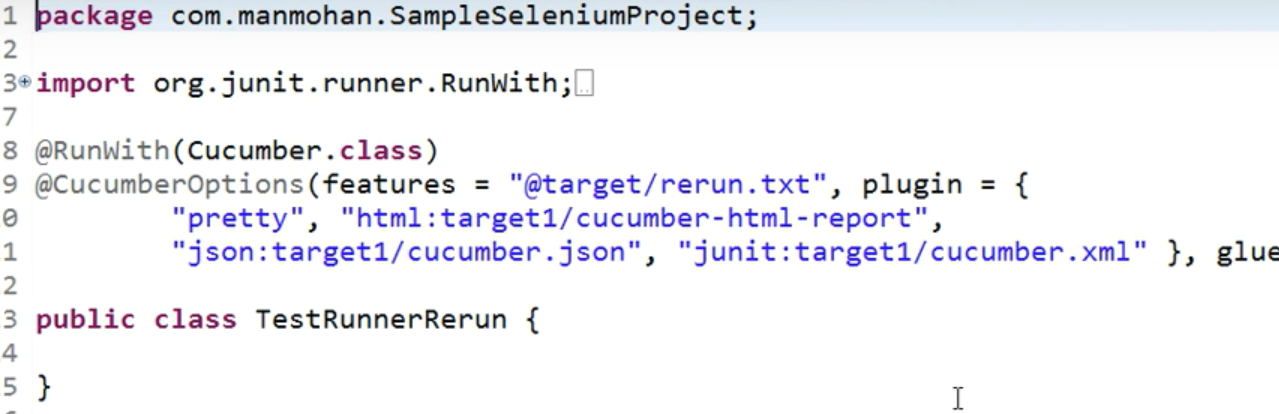
**Where json 🡪 format of the file**

**Target / jsonReport 🡪 path from source directory**

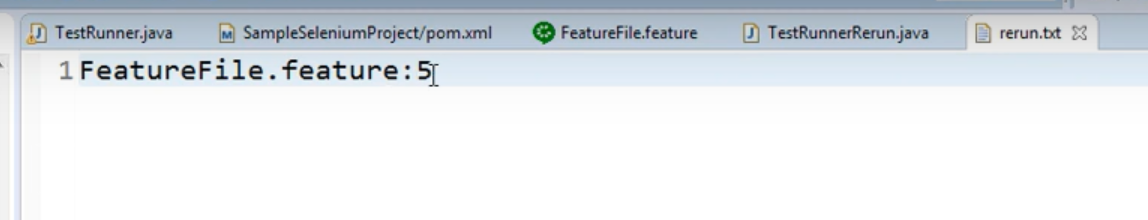
**Cucumber-report.json**

The result will be available at target/cucumber-html-reports/js/overviewfeatures.html. Copy the path of the file and paste in the browser to see the report.

Note: the folder name is target1 so that we can get 2 distinct folders with diff set of reports just to differentiate/ not to override the same report again. We can club the report later



Rerun.txt file will info in the below format

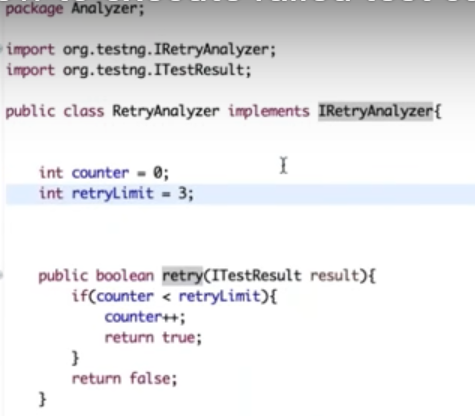


**Using TestNG:**

It can be achieved either at TestLevel (not effective) or at RunTime

1. **Test Level**

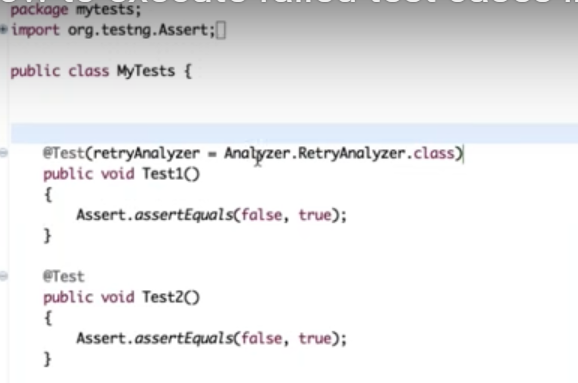
First create a package named “analyser” and create a class which implements “IRetryAnalyser” which takes args as “ITestResult result” as below then override the retry method



After that go to the testNG class and add the following in brackets after the @Test annotation as in

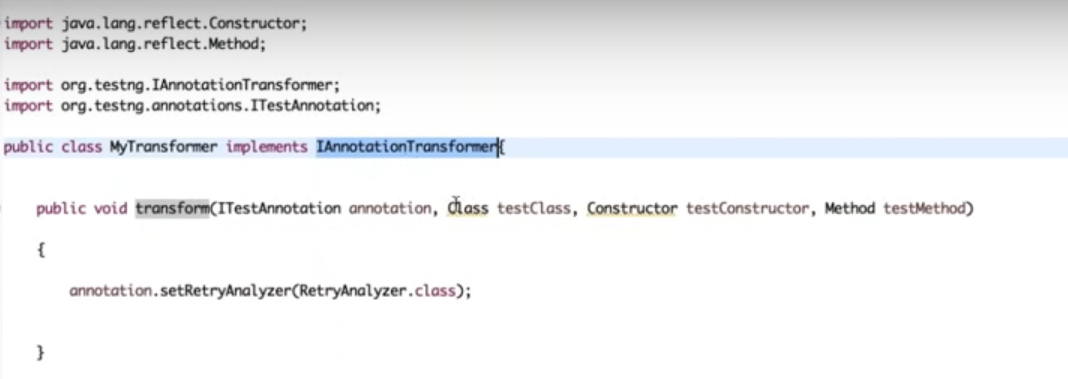
@Test(retryAnalyser=Analyser.RetryAnalyser.class)

Where analyser is package name, RetryAnalyser is class name

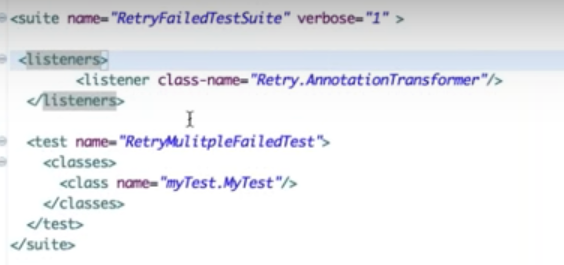


1. **Run Time**

Create a new class which implements IAnnotationTransformer in the same package as RetryAnalyser class. When we implements this way, we do not need to explicitly add the retryanalyzer at the test level

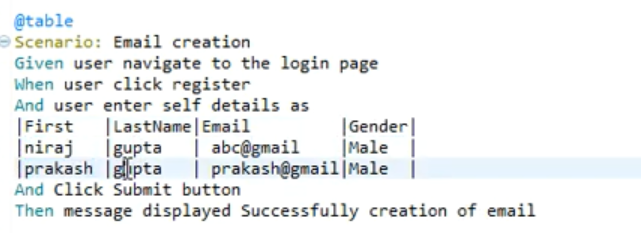


Where annotation.setRetryAnalyzer takes the argument of our RetryAnalyzer class. To give retry failed test case knowledge to the framework, we need to add a listener tag in testNG.xml file as below



**Data Tables in Cucumber:**

This is similar to Scenario outline with Example and the difference is Data table is applicable only for that respective steps where the data table is mentioned and won’t have example keyword



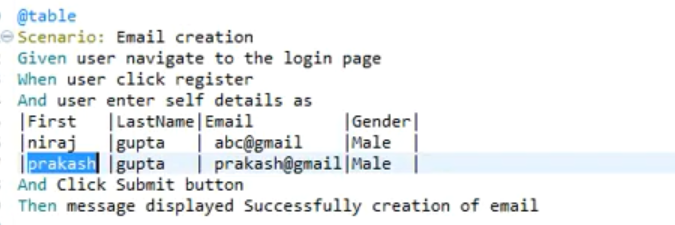
To access the data mentioned in the feature file, we need to write couple of code in step definition file using List or map as below

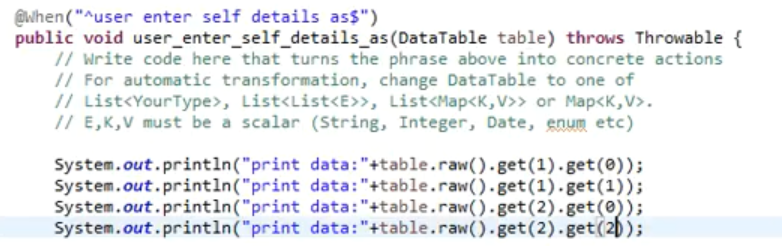
**Using List:**

We need to make the respective step method in step definition should take the parameter (DataTable table) and using the reference variable make use of raw method

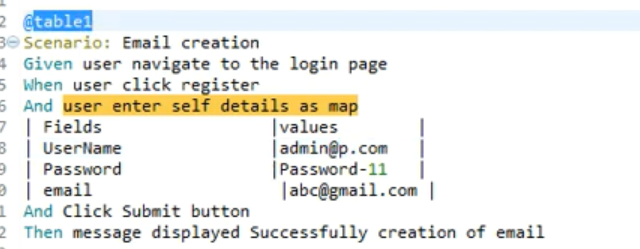
List<List<String>> table1 = table.raw();

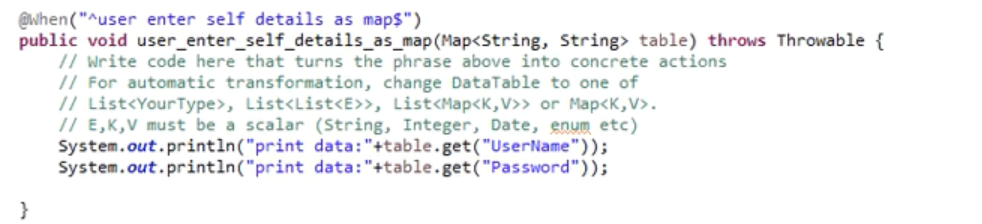
sendKeys(table1.get(1).get(0)) // where get(1) indicates the row # and get(0) indicated the column#



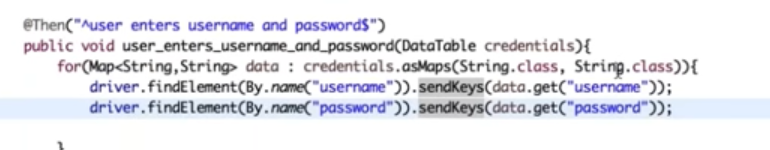


**Using Map:**





Map can be used in either ways as above or as below. Suggest going with below when we have multiple data to pass



**Hooks:**

Hooks are not part of the feature file. We either need to write in a separate class or along with step definition.

Types:

@Before – executed before each scenario

@After – executed after each scenario

@BeforeStep – before each step of the scenario

@AfterStep – after each step of the scenario

Note: When we have more than one hook lets say

@Before(order=1)

M1(){

}

@Before(order=2)

M2(){

}

Then add the order to it.

Annotate tags with hooks. Let us say I have segregated my scenarios into @smoke, @regression, @function etc.,

Now if I want the before hooks to run only for @smoke test then

@Before(“@smoke”)

Public void intanstiate(){

}

To give hook’s knowledge to cucumber BDD, add the hooks class path in testrunner file under glue section as mentioned below.

A screenshot of a computer program

Description automatically generated

To get a nice report in cucumber, add the following code from the below link in pom.xml

<https://github.com/damianszczepanik/maven-cucumber-reporting>

<build>

<plugins>

<plugin>

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

<artifactId>maven-surefire-plugin</artifactId>

<configuration>

<testFailureIgnore>true</testFailureIgnore>

</configuration>

</plugin>

<plugin>

<groupId>net.masterthought</groupId>

<artifactId>maven-cucumber-reporting</artifactId>

<version>(check version above)</version>

<executions>

<execution>

<id>execution</id>

<phase>verify</phase>

<goals>

<goal>generate</goal>

</goals>

<configuration>

<projectName>cucumber-jvm-example</projectName>

<!-- optional, per documentation set this to "true" to bypass generation of Cucumber Reports entirely, defaults to false if not specified -->

<skip>false</skip>

<!-- output directory for the generated report -->

<outputDirectory>${project.build.directory}</outputDirectory>

<!-- optional, defaults to outputDirectory if not specified -->

<inputDirectory>${project.build.directory}/jsonReports</inputDirectory>

<jsonFiles>

<!-- supports wildcard or name pattern -->

<param>\*\*/\*.json</param>

</jsonFiles>

<!-- optional, defaults to outputDirectory if not specified -->

<classificationDirectory>${project.build.directory}/classifications</classificationDirectory>

<classificationFiles>

<!-- supports wildcard or name pattern -->

<param>sample.properties</param>

<param>other.properties</param>

</classificationFiles>

<parallelTesting>false</parallelTesting>

<!-- optional, set true to group features by its Ids -->

<mergeFeaturesById>false</mergeFeaturesById>

<!-- optional, set true to get a final report with latest results of the same test from different test runs -->

<mergeFeaturesWithRetest>false</mergeFeaturesWithRetest>

<!-- optional, set true to fail build on test failures -->

<checkBuildResult>false</checkBuildResult>

</configuration>

</execution>

</executions>

</plugin>

</plugins>

</build>

After you add the above plugins in pom.xml, update your testrunner class to have an additional attribute called plugin as Plugin={“json:target/jsonReport/cucumber-report.json” }

A screenshot of a computer program

Description automatically generated

Now when we use verify in mvn command, it will read the json file mentioned above in target folder and create a nice report with the help of Damien plugin

**How to run test cases in parallel using TestNG:**

With Junit, we can run all the feature file in parallel but the scenario inside the feature file will in sequence whereas TestNG will help us to overcome this limitation.

Add the dependencies for cucumber testng in pom.xml file

Update the sure-fire plugin as below to set parallel = method and threadcount = unlimited

A computer screen shot of a program code

Description automatically generated

Important : We need to make changes with the package structure i.e., feature file from feature package moved to a folder named “parallel” under src/resource/java and both “step definition” and “test runner” will be in the same package src/test/java.

Testrunner class should extend abstracttestngcucumberclass and should implement the scenario method as below

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**Extent report in cucumber:**

Add the dependency ExtentReports cucumber 6 Adapter

A screenshot of a computer

Description automatically generated

We need to create a file named “extent.properties’ in the same place where your feature file package is located i.e., in src/test/java with content

A screenshot of a computer program

Description automatically generated