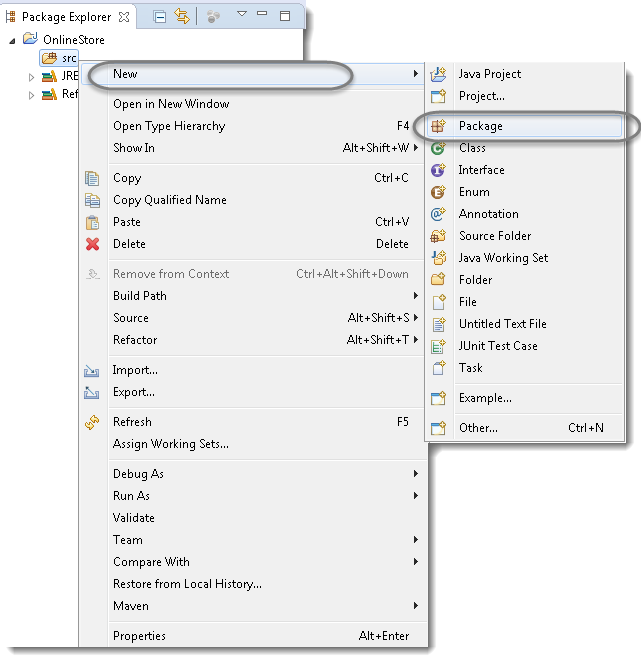
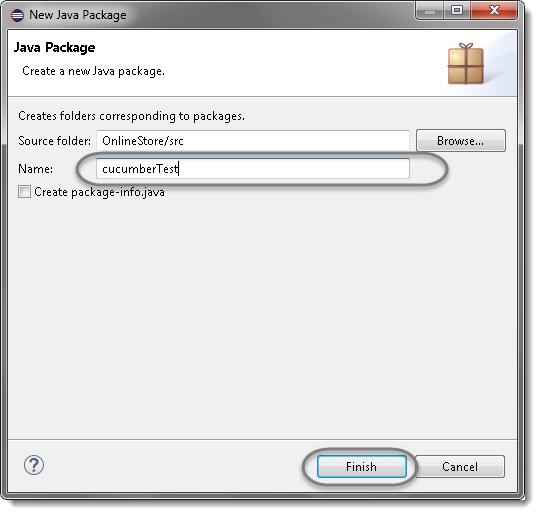
*Create Folder Structure*

Before moving head for writing the first script, let’s create a nice folder structure of the project.

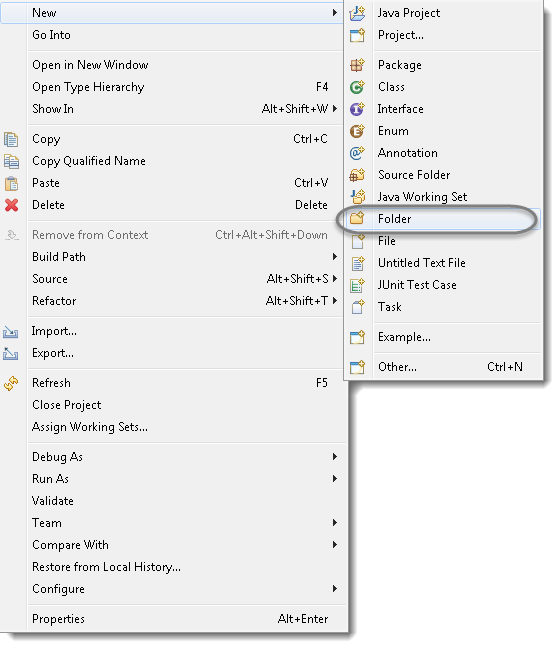
1) Create a new ***Package*** by *right click* on the ‘***src***‘ folder and select *New > Package*.



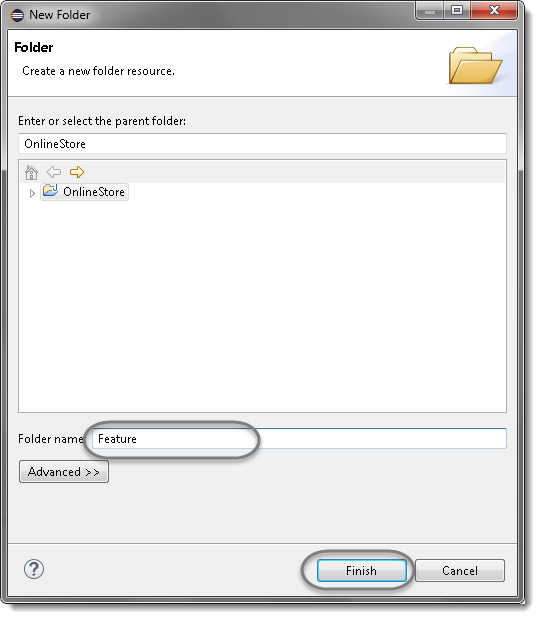
2) Name it as ‘***cucumberTest***’ and click on *Finish* button.  


3) Create another ***Package*** and name it as ‘***stepDefinition***’, by *right click* on the ‘***src***‘ folder and select *New > Package*.

4) Create a new ***Folder*** this time by *right click* on the *project* ‘***OnlineStore***‘ and *select New > Folder*.



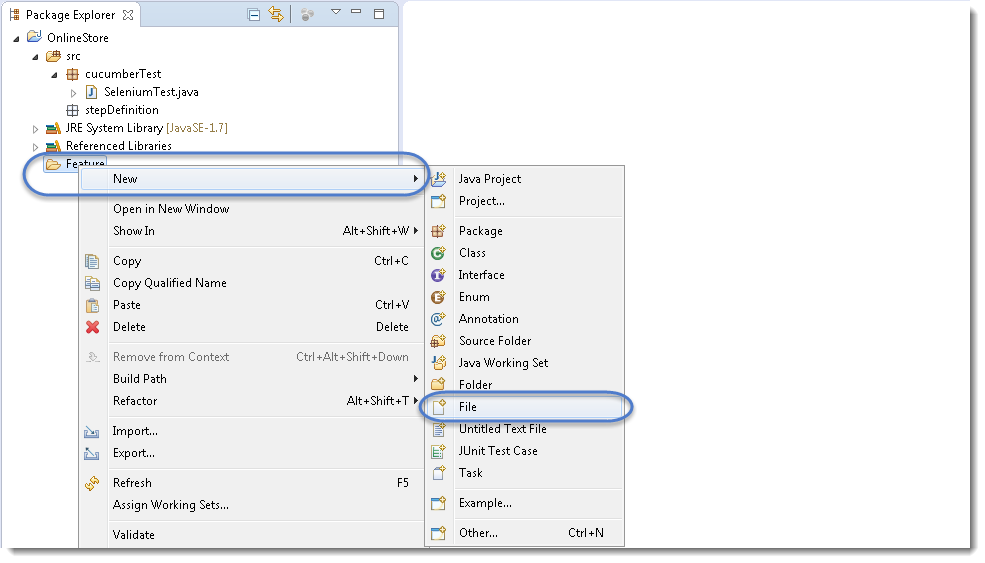
5) Name it as ‘***Feature***’ and click on *Finish* button.



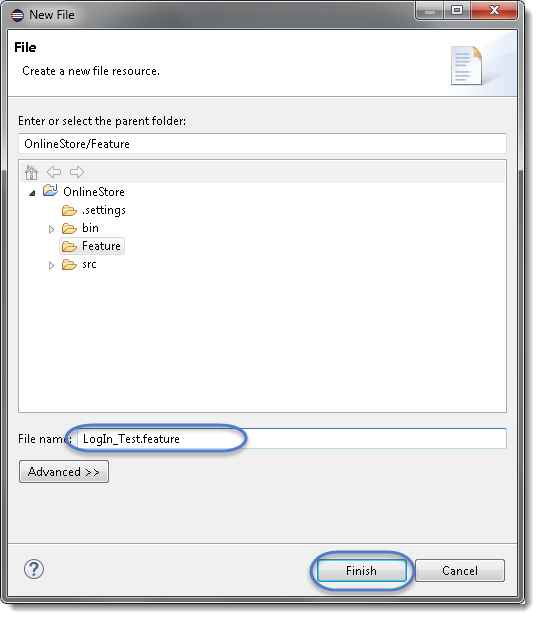
Next Step

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

1) On the ***Feature*** folder *Right click* and select ***New > File***



2) In order for Cucumber to automatically detect the stories (or ***features***, as they’re known in *Cucumber*), you need to make sure that they carry the ‘***.feature***‘ file extension. For example, in this case, I’ve named my user story ‘***LogIn\_Test.feature***‘. Every ‘*.feature*‘ file conventionally consists of a single feature.



***Note****: In case you get a pop up from Eclipse which suggest you to install the better Editor for BDD files, please go ahead and install that. At the botttom of the chapter, steps to install the better editor is given.*

3) Write the first cucumber script. In BDD terms the scenario would look like the following.

***Feature: LogIn Action Test***

***Description: This feature will test a LogIn and LogOut functionality***

***Scenario: Successful Login with Valid Credentials***

***Given User is on Home Page***

***When User Navigate to LogIn Page***

***And User enters UserName and Password***

***Then Message displayed Login Successfully***

***Note:****This is a simple test in Cucumber. Don’t worry about the syntax if you don’t understand it. Ideally you should be able to understand the intent of the test just by reading a test in feature file. We will discuss this in more details in next chapter.*

***Keywords***

Now moving forward we have just defined a test. You will notice colored part of the tests (***Feature, Scenario, Given, When, And and Then***). These are keywords defined by ***Gherkin***. *Gherkin* has more keywords and we will discuss those in following tutorials. But to start off we can quickly explain some of the keywords in one line. Note this is not complete listing of Keywords:

***Feature: Defines what feature you will be testing in the tests below***

***Given: Tells the pre-condition of the test***

***And: Defines additional conditions of the test***

***Then: States the post condition. You can say that it is expected result of the test****.*

***Gherkin***

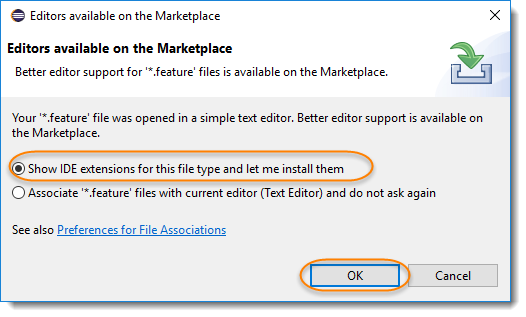
A language above is called ***Gherkin*** and it implements the principles of ***Business readable domain specific language(BRDSL)***. Domain specific language gives you the ability to describe your application behavior without getting into details of implementation. What does that mean? If we go back to our tutorial in [***TDD***](http://toolsqa.wpengine.com/cucumber/test-driven-development-tdd/) we saw that we wrote test code before writing any application code. In a way we described what is the expected behavior of our application in terms of tests. On *TDD* those tests were pure Java tests, in your case those might be a C++ or C# tests. But the basic idea is that those are core technical tests.

If we now come back to [***BDD/BRDSL***](http://toolsqa.wpengine.com/cucumber/behavior-driven-development/) we will see that we are able to describe tests in a more readable format. In the above test it’s quite clear and evident, just by reading, what test would do. At the same time of being a test it also documents the behavior of application. This is the true power of *BDD/BRDSL* and it will become the power of cucumber eventually because cucumber works on the same principles.

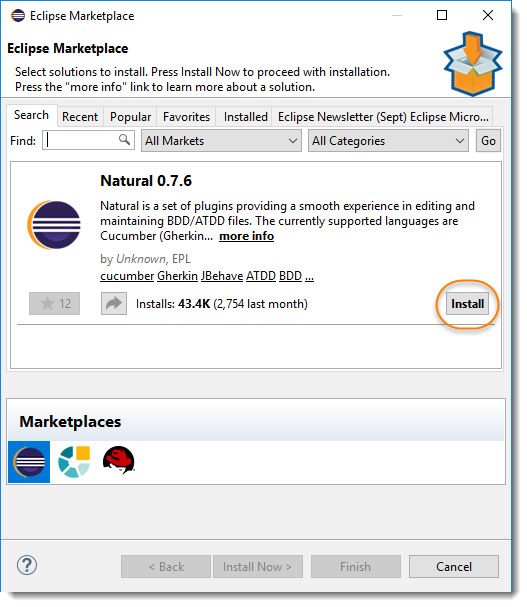
***Steps to install the Natural Eclipse Editor for Gherkin***

You get this option automatically when try to create a new file with .feature ext. But if you do not get that one, you can anytime go to Eclipse Marketplace and look for the same to install it.

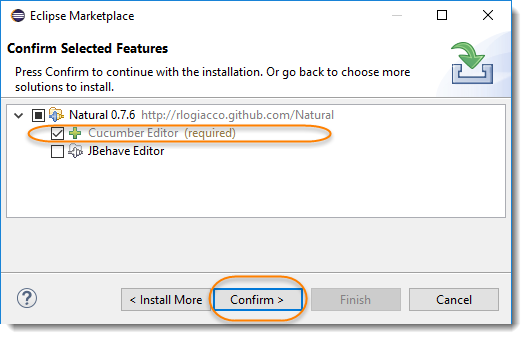
1)Just select the first option of ***Show IDE extensions*** if it is not pre-selected and click OK.



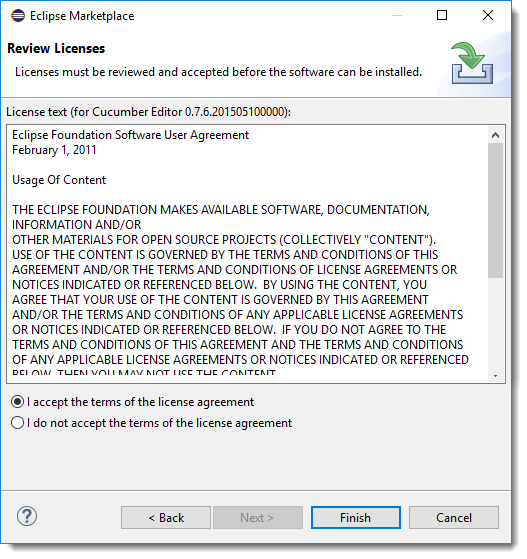
2) Natural is the name of the plugin, so this can also be found [***Eclipse Marketplace***](https://marketplace.eclipse.org/content/natural#group-details). Just click Install.



3) This will give you an option to select, whether you like to use it for Cucumber or JBehave(Another BDD Framework). Go for Cucumber.



4) Last step is to accept the Terms and Conditions.



Step

## **JUnit Test Runner Class**

Create a new ***Class*** file in the ‘***cucumberTest***‘ package and name it as ‘***TestRunner***‘, by right click on the Package and select ***New > Class.***This class just need annotations to understand that cucumber features would be run through it and you can specify feature files to be picked up plus the steps package location. There are bunch of other parameters that it can take, to be discussed later in [***Cucumber Options***](http://toolsqa.wpengine.com/cucumber/options/).

**package** cucumberTest;

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

**import** cucumber.api.junit.Cucumber;

**import** cucumber.api.CucumberOptions;

@RunWith(Cucumber.**class**)

@CucumberOptions(

features = "Feature"

,glue={"stepDefinition"}

)

**public** **class** TestRunner {

}

For the curios minds, I will explain this code. Note that it is covered in details in coming tutorials. Consider this as a limited description.

### Import Statements

First import statement ‘***org.junit.runner.RunWith***‘ imports @RunWith annotation from the Junit class. @RunWith annotation tells JUnit that tests should run using ***Cucumber class***present in ‘***Cucumber.api.junit***‘ package.

Second import statement ‘***cucumber.api.CucumberOptions***‘ imports the ***@CucumberOptions*** annotation. This annotation tells Cucumber a lot of things like where to look for feature files, what reporting system to use and some other things also. But as of now in the above test we have just told it for the Feature file folder.

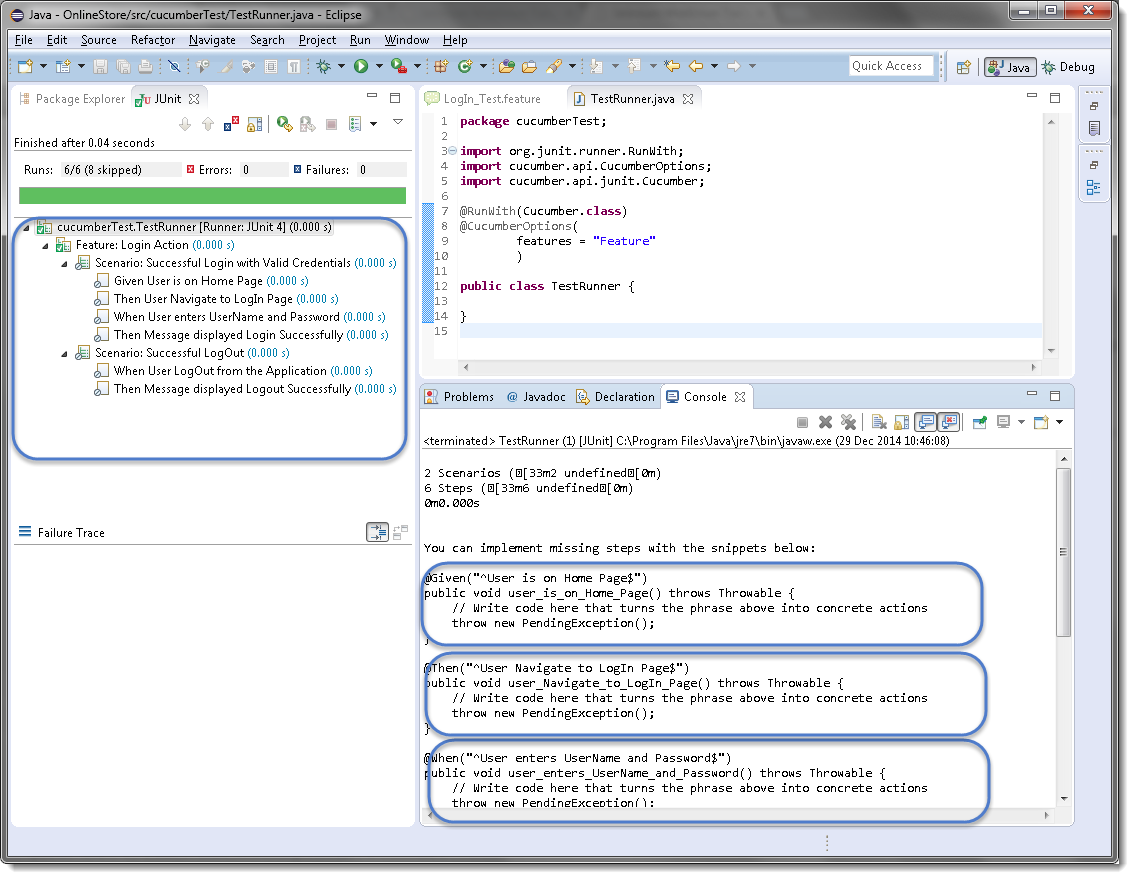
### Run the Cucumber Test

Now we are all set to run the first Cucumber test. There are multiple ways and runners to use when it comes to cucumber feature files. We would try to understand how to run it from the IDE first and then from a command line at a later point.

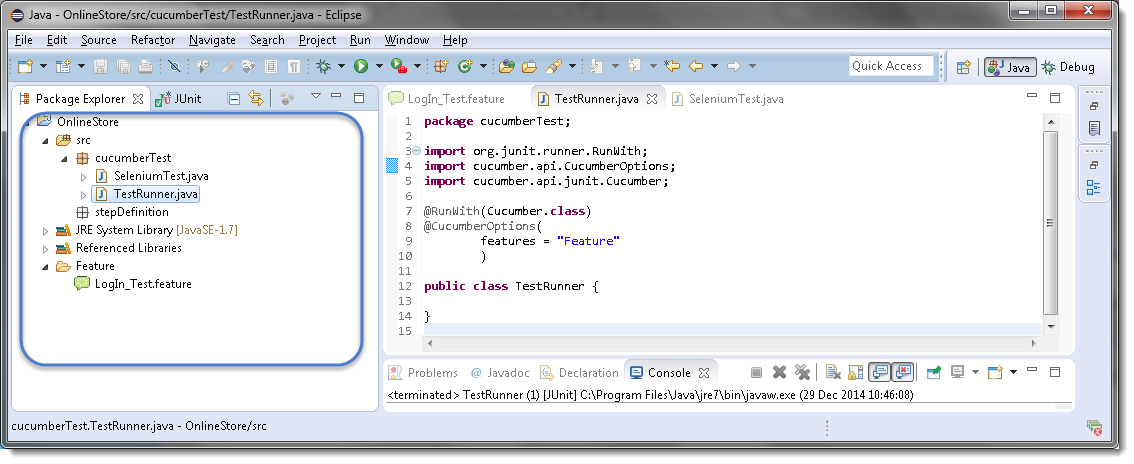
Even from the IDE, there are a couple of ways to run these feature files.

* Click on the ***Run*** button on eclipse and you have your test run
* Right Click on ***TestRunner*** class and Click ***Run As***  > ***JUnit Test Application***

You will think where is the java code that will execute for these tests? Well don’t worry about that at this moment. Let’s just see what we have on the console window. Here is the text that I got on my console. Look how Cucumber has suggested that you should implement these methods so that the Steps mentioned in the Feature file can be traced to Java methods, which can be executed while executing the feature file.



Now your project should look like this in Eclipse IDE:



### Errors on running Cucumber Feature

Exception in thread “main” cucumber.runtime.CucumberException: No backends were found. Please make sure you have a backend module on your CLASSPATH.

***Solution***

Most probably this means that your ***cucumber-java*** version and ***java*** version on your machine is not compatible with each other.  First check Java Version on your machine by going through this article [***How to check Java/JDK Version Installed on your Machine.***](http://toolsqa.com/java/check-java-version-installed-windows-machine/)

On my machine I have Java 1.8.0 with ***cucumber-Java8-1.2.5*** and it did not work. When I degraded my cucumber java version to ***cucumber-Java-1.2.5,***it worked fine for me. Just make sure that first you remove the cucumber-java which did not work for you from ***Project build path >> Libraries*** and than add new. Keeping both may create further issues for you.

Exception in thread “main” java.lang.NoClassDefFoundError: gherkin/formatter/Formatter

***Solution***

This means that Gherkin version you are using is not compatible with other Cucumber libraries. I tried using the latest ***gherkin3-3.0.0*** but it did not work for me, so I degraded it to ***gherkin-2.12.2***

***I got below versions on Oct’17 for Cucumber***

* cobertura-2.1.1
* cucumber-core-1.2.5
* cucumber-java-1.2.5
* cucumber-junit-1.2.5
* cucumber-jvm-deps-1.0.5
* cucumber-reporting-3.10.0
* gherkin-2.12.2
* junit-4.12
* mockito-all-2.0.2-beta

Step

***Feature****: LogIn Action Test*  
*Description: This feature will test a LogIn and LogOut functionality*

***Scenario****: Successful Login with Valid Credentials*  
***Given****User is on Home Page*  
***When****User Navigate to LogIn Page*  
***And****User enters UserName and Password*  
***Then****Message displayed Login Successfully*

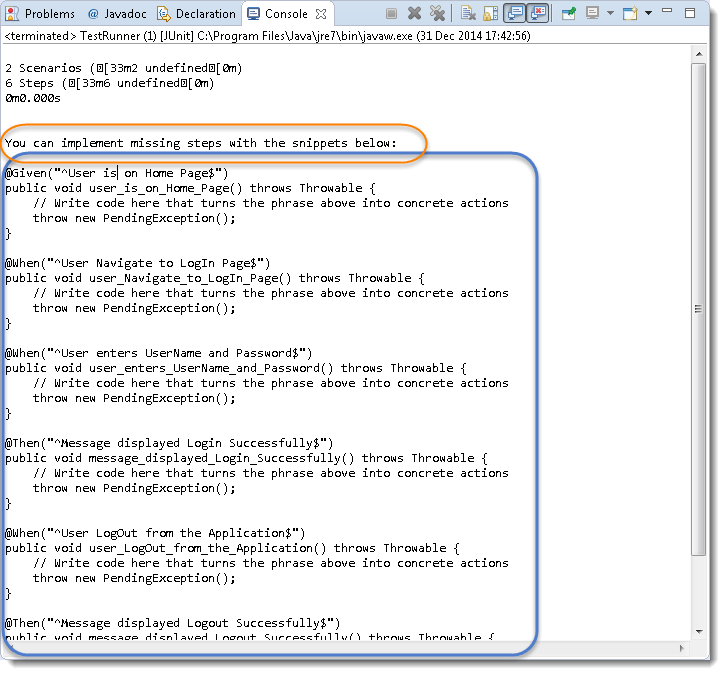
Step

A Step Definition is a small piece of code with a pattern attached to it or in other words a Step Definition is a java method in a class with an annotation above it. An annotation followed by the pattern is used to link the Step Definition to all the matching Steps, and the code is what Cucumber will execute when it sees a Gherkin Step. Cucumber finds the Step Definition file with the help of Glue code in ***Cucumber Options***. We will cover different Cucumber Optionsin next chapter.

## **Add a Step Definition file**

1) Create a new ***Class*** file in the ‘***stepDefinition***‘ package and name it as ‘***Test\_Steps***‘, by right click on the Package and select New > Class. Do not check the option for ‘***public static void main***‘ and click on ***Finish*** button.

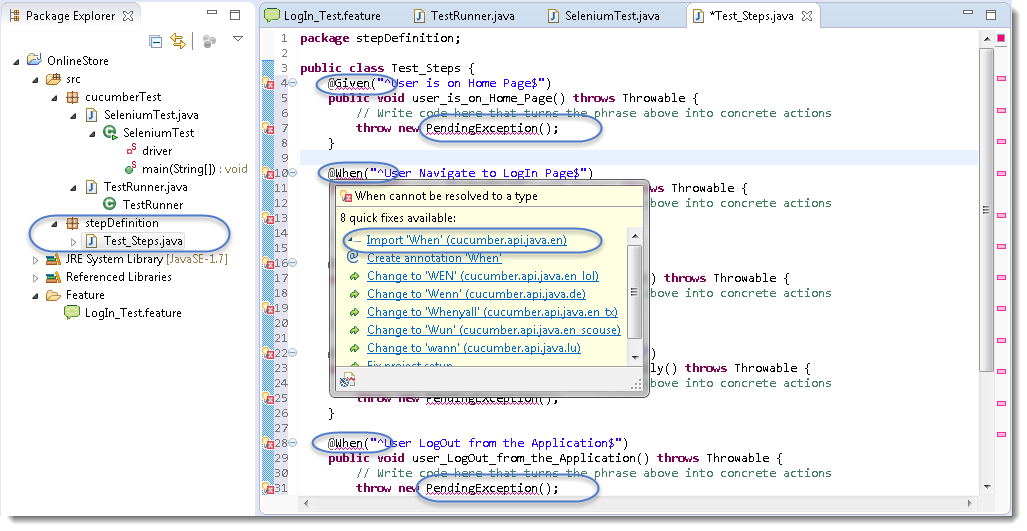
2) Take a look at the message in the console window. This message was displayed, when we ran the ***Test\_Runner*** class.



***Note:***Please go through the chapter of [***First Cucumber Selenium Test***](http://toolsqa.wpengine.com/cucumber/first-cucumber-selenium-java-test/)to understand the above message.

2) Notice, the eclipse console window says ‘***You can implement missing steps with the snippets below:***‘. It is very easy to implement all the steps, all you need to do is to copy the complete text marked in a blue box and paste it in to the above created ***Test\_Steps*** class.

3) As of now the test will show many errors on ‘***@***‘ **annotations**. Mouse hover at the annotations and import the ‘***cucumber.api.java.en***‘ for all the annotations.



## **Add Selenium Java code in the Step Definition methods**

1) Now take out the Selenium Java code of the following steps from the ‘***SeleniumTest***‘ and paste it in to the first method ‘**@Given(“^User is on Home Page$”)**‘.

* Launch the Browser
* Navigate to Home Page

Method will look like this now:

@Given("^User is on Home Page$")

public void user\_is\_on\_Home\_Page() throws Throwable {

driver = new FirefoxDriver();

         driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);

       driver.get("http://www.store.demoqa.com");

}

2) Take out the code take out the Selenium Java code of the following steps from the ‘***SeleniumTest***‘ and paste it in to the second method ‘***@When(“^User Navigate to LogIn Page$”)***‘.

* Click on the LogIn link

Method will look like this now:

@When("^User Navigate to LogIn Page$")

public void user\_Navigate\_to\_LogIn\_Page() throws Throwable {

driver.findElement(By.xpath(".//\*[@id='account']/a")).click();

}

3) Take out the code take out the Selenium Java code of the following steps from the ‘***SeleniumTest***‘ and paste it in to the second method ‘***@When(“^User enters UserName and Password$”)***‘.

* Enter UserName and Password
* Click on Submit button

Method will look like this now:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | @When("^User enters UserName and Password$")  public void user\_enters\_UserName\_and\_Password() throws Throwable {  driver.findElement(By.id("log")).sendKeys("testuser\_1");      driver.findElement(By.id("pwd")).sendKeys("Test@123");      driver.findElement(By.id("login")).click();  } |

4) Do the same steps for rest of the methods as well and complete Test\_Steps class will look like this:

***Step Definition: Test\_Steps Class***

package stepDefinition;

import java.util.concurrent.TimeUnit;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.firefox.FirefoxDriver;

import cucumber.api.java.en.Given;

import cucumber.api.java.en.Then;

import cucumber.api.java.en.When;

public class Test\_Steps {

public static WebDriver driver;

@Given("^User is on Home Page$")

public void user\_is\_on\_Home\_Page() throws Throwable {

        driver = new FirefoxDriver();

        driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);

        driver.get("http://www.store.demoqa.com");

}

@When("^User Navigate to LogIn Page$")

public void user\_Navigate\_to\_LogIn\_Page() throws Throwable {

driver.findElement(By.xpath(".//\*[@id='account']/a")).click();

}

@When("^User enters UserName and Password$")

public void user\_enters\_UserName\_and\_Password() throws Throwable {

driver.findElement(By.id("log")).sendKeys("testuser\_1");

    driver.findElement(By.id("pwd")).sendKeys("Test@123");

    driver.findElement(By.id("login")).click();

}

@Then("^Message displayed Login Successfully$")

public void message\_displayed\_Login\_Successfully() throws Throwable {

System.out.println("Login Successfully");

}

@When("^User LogOut from the Application$")

public void user\_LogOut\_from\_the\_Application() throws Throwable {

driver.findElement (By.xpath(".//\*[@id='account\_logout']/a")).click();

}

@Then("^Message displayed Logout Successfully$")

public void message\_displayed\_Logout\_Successfully() throws Throwable {

        System.out.println("LogOut Successfully");

}

}

***Note***: Make sure to create your own Username and Password for the test and do not attempt to login with wrong credentials, as you will be blocked for few hours then on demo website.

## **Run the Cucumber Test**

Now we are all set to run the first Cucumber test. Right Click on ***TestRunner*** class and Click ***Run As***  > ***JUnit Test.***Cucumber will run the script the same way it runs in Selenium WebDriver and the result will be shown in the left hand side project explorer window in JUnittab.